A Comparative Study on Performance of XML parser APIs (DOM and SAX) in Parsing Efficiency

DaYong Wu
Department of Electrical and Computing Engineering
Xiamen University Malaysia, Jalan Sunsuria Bandar Sunsuria
43900 Sepang, Selangor, Malaysia
dmt1609060@xmu.edu.my

Kien Tsong Chau
Centre for Instructional Technology and Multimedia
Universiti Sains Malaysia
11800 USM, Penang, Malaysia
chaukientsong@usm.my

JingYi Wang, ChuTing Pan
Department of Electrical and Computing Engineering
Xiamen University Malaysia, Jalan Sunsuria Bandar Sunsuria
43900 Sepang, Selangor, Malaysia
doriswangjingyi@yahoo.com; dmt1609021@xmu.edu.my

ABSTRACT

As a semi-structure language, XML is widely used in converting unstructured data to structured data due to its simplicity, extendibility and interoperability. There are numerous XML parser APIs that perform the same function of parsing XML document. This paper compares the performance between two famous XML parser APIs, DOM and SAX, in terms of speed, memory consumption and modifiability in parsing process. This experiment concluded that DOM API takes more time, more memory with higher level of modifiability while SAX API takes less time, less memory with lower level of modifiability.

CCS Concepts

• Information systems → Open source software • Applied computing → Document metadata

Keywords

Unstructured data; Extensible Markup Language; XML parsing technique.

1. INTRODUCTION

Extensible Markup Language (XML) has become emerging data representation and data exchange across the Internet in recent years. Numerous XML parser Application Program Interfaces (APIs) are available for choices in extracting data and creating XML documents. However, there are a variety of specifications and standards of XML parser APIs, and research on how to select a XML parser API best suited for certain XML processing is scarce. Therefore, researchers hereby conduct a comparative study on the performance of two popular XML parser APIs, namely Document Object Model (DOM) and Sample API for XML (SAX).

2. RESEARCH OBJECTIVES

The research seeks to compare the performance between two prominent XML parser APIs, namely DOM and SAX in terms of speed, memory consumption and modifiability in parsing process, and subsequently enables users to determine which is most appropriate to be selected in certain situation. Improper choice of parser will ruin the performance and subsequently leading to degradation in productivity. Apart from that, such comparative study is beneficial to web developer community in general because the strength and weaknesses of different types of XML parsers can be determined.

3. RESEARCH QUESTION

The research aims to answer the following question:

Does DOM perform better than SAX on parsing speed, memory consumption and modifiability?

4. LITERATURE REVIEW

API stands for application program interface, which is a set of routines, protocols, and tools for building software applications. Basically, an API specifies how software components should interact [1]. The parsing of XML documents is the process of transforming an unstructured sequence of characters representing an XML document into a structured component that conforms to XML specifications. DOM is an API used for parsing XML documents as well as accessing and manipulating documents (in particular, HTML and XML Documents)" [2]. SAX, on the other hand, is an event-based API for XML parsing. Through SAX, the XML document is parsed sequentially from the beginning until the end [3].

4.1 Past Research Works

Various research works had been performed which compare on conformance to standards, speed, flexibility, and memory usage. Mohseni's [4] research works on XML parsers revealed that Microsoft parser had the shortest loading time when it went through a 92KB XML file compared to Oracle parser, Sun and Xerces parser. Karre and Elbaum [5] indicated that Apache, IBM and Xerces performed similarly in terms of accuracy. Deshmukh, Bamnote, and Kale [6] compared XML parser APIs with respect to time and memory usage for a set of XML documents ranging from small-scale (below 1KB) to large-scale (above 6KB) file sizes. It was found that DOM took less time than SAX for small files, but has higher requirement for memory. The time taken by SAX was 23-fold longer than DOM when processing large-scale files. Despite consuming more memory when the document was very large, DOM was a better choice for database application than SAX. Lam, Ding and Liu [7] explored the performance features of four parsing models, SAX, DOM, StAX and VTD. The result showed that DOM tended to be memory intensive. In contrast, lower memory space was required by SAX. Holm & Gustavsson

ICCSP 2019, January 19–21, 2019, Kuala Lumpur, Malaysia
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6618-2/19/01…$15.00
https://doi.org/10.1145/3309074.3309124

[8] concluded that the performance of the parsers was varied with the changes of XML document structure. Overall, DOM performed the worst, and SAX had similar performance with VTD. Specifically, SAX had the best adaptability, closely followed by VTD. DOM had a relatively lower complexity on the code, but the longer parsing time compared to SAX. Other relevant studies included Shanmugasundaram et al. [9], Wan [10], Holm and Gustavsson's [11] comparative study on XML Parsers with respect to adaptability, Saxena & Kothari's [12] empirical analysis of XML parsing using various operating systems, Haw and Rao's [13] comparative study on Benchmarking XML Parsers. Oliveira, Santos and Belo [14], Zhao and Laxmi [15], Li [16] and Ruchita and Deshmukh [17]. Nevertheless, there lacks researches about displaying the performance of DOM and SAX progressively with changing sizes of XML files. A more thorough and dynamic experiment formulated for the purpose of testing the performance of DOM and SAX is illustrated in the following sections. The motivation for this comparison on parsing speed, memory consumption and modifiability is to create a guideline for choosing an XML parser that is most suitable in certain situation.

5. RESEARCH METHODOLOGY

The researchers consumed two hours per day for online search in the first two weeks of the overall three-week research. The researchers proceed to Google Scholar, Web of Science, and China Academic Journals database to study articles related to XML parsing. Meanwhile, the researchers employed three Search Engines, namely Google, Baidu, and Bing to identify updated news on XML parsing. The keywords adopted for searching were DOM parser, SAX parser, XML parsing. Incredible websites namely Wikipedia and blogs would not be searched.

5.1 Experiments

An experiment formulated for the purpose of testing the performance of DOM and SAX is illustrated in the following sections.

5.1.1 Process Description

The experiment was designed in two consecutive steps (Figure 1). First, data from the Internet was stored into Excel sheet tables in XLS foramt before they were converted into XML files. Second, XML files were then transferred to structured data. First conversion was accomplished by using Apache POI, the Java API for Microsoft Documents, to read and write Excel tables. Parsing was a process of sequentially scanning the documents. Defined functions would be called to process the beginning and end of documents and elements.

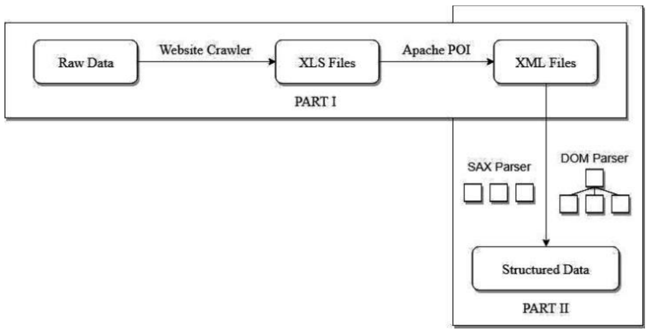


Figure 1. Experiment Process

5.1.2 Algorithm

While converting Excel tables to XML files, an algorithm was used to read Excel files through Java API. Traversing XLS files and obtaining cell location were involved to generate and export XML files. Another algorithm was deployed as well to parse XML files by utilizing SAX and importing the result to a database. This algorithm called XML Parser directly to parse the documents and sent each event to corresponding handlers. A class was needed to inherit the ContentHandler class which was provided by Android system. The algorithm included the beginning and end of reading the document, parsing an element, and processing character data. A sample of how algorithms worked was illustrated in Appendix.

5.1.3 Environment

The experiment was conducted on a laptop with an intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz and 8G memory while the software environment was under Microsoft windows10 operating system. The website crawler tool was Octoparse, which was free and powerful in extracting data from websites. Java was used as the developing language and Eclipse was used as the integrated development environment. The database server was SQL Server 2017. Researchers chose the software they were familiar with, which were also widely used in doing the corresponding tasks.

5.1.4 Data

All experiment data were taken from Amazon.com. Researchers randomly chose ten goods under Woman's Fashion Department and collected information of customers' comments on each product, which included Comment's Title, Stars, Author, Date, Content, and Purchase Model. Table 1 shows the number of comments and size of corresponding XML files converted from Excel files.

Table 1. Number of Comments &Size of XML files

No.	1	2	3	4	5	6	7	8	9	10
Num of Comment	90	190	240	310	390	500	600	780	1330	1600
Size of XML file	61	128	159	207	260	332	402	523	887	1065

5.1.5 Experimental Methods

Researchers used a website crawler tool to extract information of online shopping comments on ten goods, storing results in Excel tables. Next, Excel files were converted into XML files in Eclipse by using Java language. Later, researchers parsed XML files by using DOM and SAX separately and imported the results into a database. In the database, the final result was well organized and clear, making it convenient for users to obtain information.

Principles of experimental designs were applied. Randomization on the choices of goods was to remove bias and other resources of extraneous variation. For each product, the treatment was repeated a number of times in order to increase the precision. During the experiment, operating time was recorded for successive analysis on parsing efficiency. By applying treatments uniformly and under standardized conditions, local control was exerted as well.

6. FINDINGS

It has been a trend that industries use XML documents to store large files nowadays. Whenever these files are used, data from these files need to be extracted frequently and the XML parser serves as the tool to read the document. The XML document tree is often divided by those XML parsers into different parts such as elements and roots. Then the extracted information will be passed to the applications that require it. In some scenarios, if the document is not formed legally, errors will be sensed by the parse and the parsing process would be pending.

If errors occur, parsers will only display the information regarding to the errors rather than the contents of the document. XML parsing tools can significantly affect the overall performance of a project. Hence it is of great importance to be familiar with the advantages and shortcomings of different types of XML parsers and choose wisely when it comes to parse XML documents in real projects.

6.1 Summary of Experiment Results

6.1.1 Parsing Speed

The contrast of the parsing time between DOM and SAX could be utilized to demonstrate the performance of the two parsers in terms of speed, memory consumption and modifiability. Experiment on speed of parsing time was based on 10 converted XML file in different sizes and finally generated the following outcomes.

Table 2. Parsing Time Comparison

No.	Size of XML file	DOM/ms	SAX/ms
1	61	98	79
2	128	106	82
3	159	113	90
4	207	144	91
5	260	130	105
6	332	131	102
7	402	135	100
8	523	137	104
9	887	148	107
10	1065	161	111

In table 2, files within 500 KB could be considered as relatively small, namely the file number from one to seven, and large was to define those outranged this size interval but were under 1 MB, which indicated the following two files. The one and only file, No10, which exceeded 1024 KB, was regarded as outrageous.

Table reveals that when the file size was small, namely between 61 KB and 207 KB, the parsing time of both DOM and SAX was kept incrementing and formed a positive correlation. However, within the whole file size interval, SAX performed much more efficient than DOM. The gap of parsing speed became exaggerated as the file size went up. Initially the gap was only 19 milliseconds, as the size reached 207 KB, the gap was enlarged to 54 milliseconds.

In the section of file size starting from 260 KB and ending at 523 KB, the time consumption of DOM demonstrated a surprisingly ebb compared with the former case. The biggest time requirement in this section was 137 milliseconds, far less than the 144 in the previous case. Nevertheless, the similarity between the two was that the positive correlation was reserved.

On the other hand, within the file size interval of 206 KB and 887 KB (from relatively small to large), SAX first performed a negative correlation concerning with the file size and the parsing speed, then it recovered the original relation. When the file size went beyond the boundary defined as outrageous, both DOM and SAX reached its peak respectively, namely 161 milliseconds and 111 milliseconds.

The experiment results clarify that SAX performed surprisingly stable and efficient among all cases in the process. Its parsing speed was faster than that of DOM in every file size interval.

Despite similarities, the two parsing APIs performed noticeably divergent. Data showed that SAX is significantly more time-saving in parsing large files than DOM. Instead of preloading, SAX is capable of scanning the document while parsing it. In this way huge amount of time is saved. Naturally, SAX usually requires less system memory allocation since it allows developers to decide what kinds of tags to use by themselves. This customized attribute is dramatically beneficial for developers when they only need to process part of the data, in this case the extensibility is well-reflected.

However, DOM’s merits are not negligible as well. DOM implements the Tree data structure to parse a file. Due to DOM is preloaded, the structure of the entire document is literally persistent in the system memory. Therefore, DOM is flexibly to be modified at any time with its life cycle so that applications are granted to make changes to data and structures. DOM also has the abilities to navigate up and down through the entire Tree structure and thus DOM parser is relatively friendly to use compared with SAX.

There is an existing formula to calculate API’s parsing efficiency. The efficiency P is derived by letting the size of the file M divided by the parsing time T. In the following table, it is very prone to perceive that the parsing efficiency of SAX is higher than DOM in every file size interval. It was true that they had almost same parsing efficiency at the beginning when the file size was too small. As the size went higher, the gap between the two also enlarged and the rate of growth is increasing sharply.

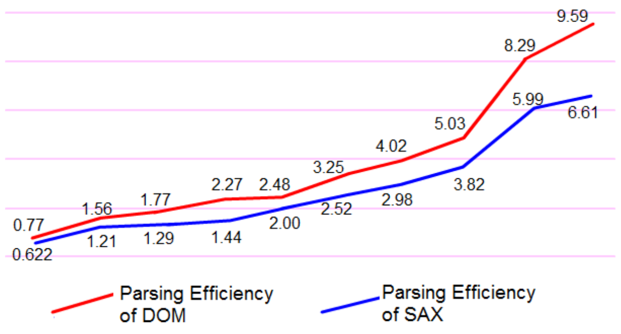


Figure 2. Parsing Efficiency of DOM and SAX

6.1.2 Memory Consumption

Besides parsing speed, DOM and SAX also differentiate in terms of consuming system resources. In this dissertation an XML file has been used to test the memory consumption of the two parsing methods.

A XML parser can be constructed by extracting the start and end tags from a document. With the assistance of data structure (mainly trees), we can parse it. Firstly, a XML document must be parsed at one time if DOM is implemented, namely the entire XML tree must be read into memory and parsed in sequence, as in Figure 3.

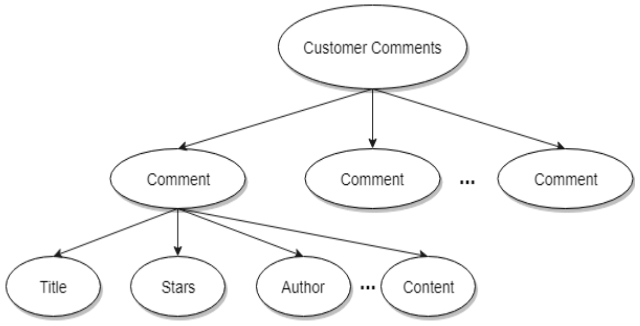
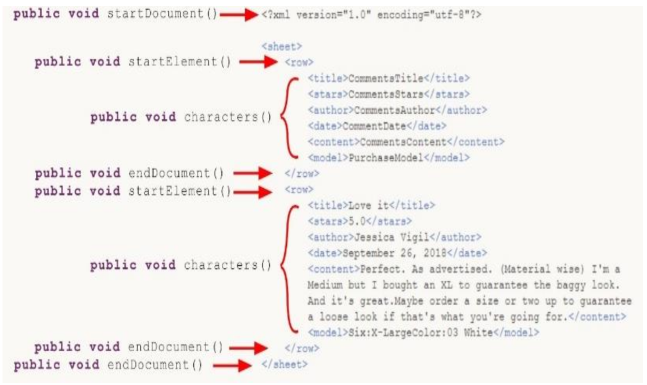


Figure 3. The structure of a XML tree

On the other hand, as for SAX, handlers are used successively to perform the parsing task. SAX is driven by events, which is a program operating method based on a callback mechanism. Its parsing process is carried out concurrently together with file loading. As a result, SAX serves as a light-weight solution compared with DOM when it comes to parse XML files.

For SAX, the size of the document usually is not that noteworthy since the parsing process is mainly done by extracting data in order. However, since the tree of the XML document is created in the memory, the bigger the XML file, the bigger the tree document is generated. Therefore it can become noticeably when the file size becomes extremely large and the size of the XML file serves as the crucial part in choosing any kind of parser methods in terms of memory consumption.



```
public void startDocument() → <?xml version="1.0" encoding="utf-8"?>

                              <sheet>
   public void startElement() → <row>
                                  <title>CommentsTitle</title>
                                  <stars>CommentsStars</stars>
          public void characters() { <author>CommentsAuthor</author>
                                  <date>CommentDate</date>
                                  <content>CommentsContent</content>
                                  <model>PurchaseModel</model>
   public void endDocument()  → </row>
   public void startElement() → <row>
                                  <title>Love it</title>
                                  <stars>5.0</stars>
                                  <author>Jessica Vigil</author>
                                  <date>September 26, 2018</date>
          public void characters() { <content>Perfect. As advertised. (Material wise) I'm a
                                  Medium but I bought an XL to guarantee the baggy look.
                                  And it's great.Maybe order a size or two up to guarantee
                                  a loose look if that's what you're going for.</content>
                                  <model>Six:X-LargeColor:03 White</model>
   public void endDocument()  → </row>
public void endDocument()  → </sheet>
```

Figure 4. Parsing Example of SAX

6.2 Comparison with Previous Research

Deshmukh's [6] disclosed that DOM performed faster than SAX when file size was below 1 KB but memory consumption was bigger. In this research, DOM did require larger time blocks to finish parsing. However, the size interval of the file is not just narrowed under 1 KB. It has been proved through rigorous experiments that the finding is also true within the file size ranging from 1 KB to 1024 KB (1 MB). The range of limitation of the research result found by Deshmukh could be further expanded at present since new and robust evidence has been testified.

In general, SAX consumes less system memory than DOM in most scenarios. Therefore, in real project the choice of selecting XML parsers should be made wisely since resources and time are limited. There should be a trade-off between resources and time. How to achieve an equilibrium is also a challenge to engineers. In the case of parsing files within 1 MB, DOM may serve as a better choice since the file size is not outrageous while the parsing speed is still fast. DOM is particularly useful when it comes to parse relatively small files. Nevertheless, when the size of file goes far beyond than 1 MB, DOM stops serving as the best selection due to SAX is both efficient and memory-saving.

6.3 Comparison with Previous Research

Deshmukh's thesis in 2014 disclosed that DOM performed faster than SAX when file size was below 1 KB but memory consumption was bigger. In this research, DOM did require larger time block to finish parsing. However, the size interval of file is not just narrowed under 1 KB. It has been proved through rigorous experiment that the finding is also true within the file size range from 1 KB to 1024 KB (1 MB). The limitation of the research result found by Deshmukh could be expended at present since new and robust evidence has been testified.

In general, SAX consumed less system memory than DOM in most scenarios. Therefore, in real project, the choice of selecting XML parsers should be made wisely since resources and time are limited. There should be a trade-off between resources and time. How to achieve equilibrium is a challenge to engineers as we. In the case of parsing files within 1 MB, DOM may serve as a better choice since the file size is not outrageous and the parsing speed is time-saving. DOM is particularly useful when it comes to parse relatively small files. Nevertheless, when the size of file goes far beyond than 1 MB, DOM stops serving as the best selection since SAX is both efficient and memory-saving.

7. CONCLUSION

This experiment concluded that DOM API takes more time, more memory with higher level of modifiability while SAX API takes less time, less memory with lower level of modifiability. As a consequence, DOM API will be helpful in modifying files and is not suitable for operation on files of big size. SAX API is capable of operating on files of big size, especially in reading specific content, and it also allows users to create their own object models. Nevertheless, performance is not the only decisive factor; while choosing an XML parser, other criteria should be considered as well, such as user's demand, license fees, and technical competence. The only limitation of the research is the researchers solely utilised XML documents to evaluate DOM and SAX. Any future research is recommended to be expanded on alternative types of APIs and files with varying size.

8. REFERENCES

[1] Beal, V. 2017. *What is API - Application Program Interface?*. Retrieved from https://www.webopedia.com/TERM/A/API.html

[2] DOM. 2018. *Introduction to the DOM*. Retrieved from https://dom.spec.whatwg.org/#introduction-to-the-dom

[3] Brownell, D. 2002. *SAX2*. Sebastopol, CA, USA: O'Reilly & Associates, Inc.

[4] Mohseni, P. 2001. *Choose Your Java XML Parser*. Retrieved from http://www.devx.com/xml/Article/16921

[5] Karre, S. and Elbaum, S. 2002. *An Empirical Assessment of XML Parsers*, 6th Workshop on Web Engineering, 2002, pp. 39-46.

[6] Deshmukh, V. M., Bamnote, G.R., Kale, P. V. April 2014. *A Comparative Study of XML Parsers across Application.* International Journal of Computing and Technology, vol. 1, no. 3, pp.4-6.

[7] Lam, T. C., Ding, J. J., and Liu, J. C. September 2008. *XML document parsing: Operational and performance characteristics*. IEEE Computer, vol. 41, no. 9, pp. 30-37.

[8] J. Holm & M. Gustavsson. 2018. XML Parsers - A comparative study with respect to adaptability. Retrieved from http://www.diva-

[9] Shanmugasundaram, J., Shekita, E., Barr, R. et al. 2001. *Efficiently publishing relational data as XML documents.* Proceedings of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000.

[10] L. Wan. May, 2013. *Research and Implementation of the Transformation from Unstructured to Structured Data.* Retrieved from http://www.doc88.com/p-9435446778415.html

[11] Holm, J. and Gustavsson, M. (2018). XML Parsers: A comparative study with respect to adaptability. Bachelor Degree Project in Information Technology.

[12] Saxena, A. & Kothari, S. 2015. An Empirical Analysis of XML parsing using various operating systems. International Journal of Engineering and Applied Sciences (IJEAS). Volume-2, Issue-2, February 2015. pp 37-40. ISSN: 2394-3661.

[13] S. C. Haw and Rao, R. K. (2007). A Comparative Study and Benchmarking on XML Parsers. Advanced Communication Technology, The 9th International Conference (Volume:1, pp. 321-325).

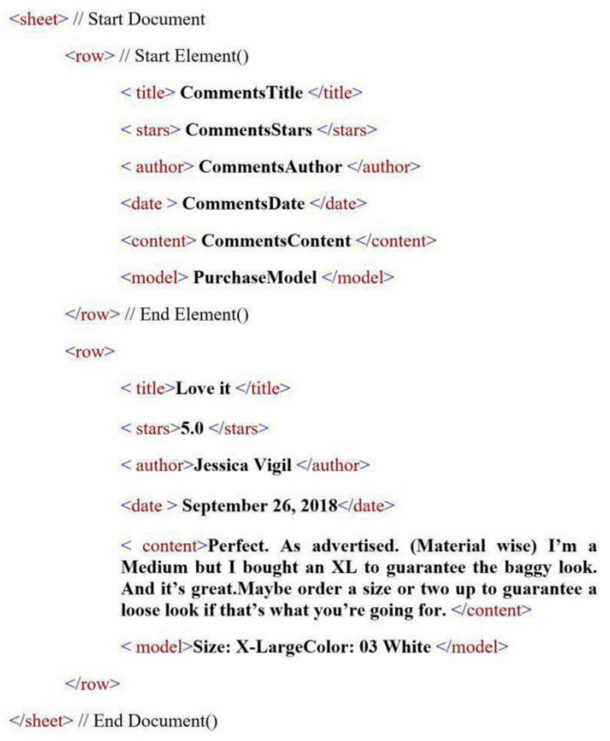
[14] Oliveira, B., Santos, V. and Belo, O. 2013. *Processing XML with Java – A Performance Benchmark.* International Journal of New Computer Architectures and their Applications. 3. 72-85.

[15] Zhao, L., Laxmi B. 2006. *Performance Evaluation and Acceleration for XML Data Parsing.*

[16] Chengkai Li. 2017. *XML Parsing, SAX/DOM.*

[17] Ruchita, A. K., V. M. Deshmukh. 2014. *Performance Evaluation of XML Parsing for Tree-Branch Symbiosis Algorithm.* International Journal of Advance Engineering and Research Development (IJAERD) (Volume 1, Issue 6).

Appendix



```
<sheet> // Start Document
        <row> // Start Element()
                < title> CommentsTitle </title>
                < stars> CommentsStars </stars>
                < author> CommentsAuthor </author>
                <date > CommentsDate </date>
                <content> CommentsContent </content>
                <model> PurchaseModel </model>
        </row> // End Element()
        <row>
                < title>Love it </title>
                < stars>5.0 </stars>
                < author>Jessica Vigil </author>
                <date > September 26, 2018</date>
                < content>Perfect. As advertised. (Material wise) I'm a
                Medium but I bought an XL to guarantee the baggy look.
                And it's great.Maybe order a size or two up to guarantee a
                loose look if that's what you're going for. </content>
                < model>Size: X-LargeColor: 03 White </model>
        </row>
</sheet> // End Document()
```