Majix Enhancement

Spenser H. Kao[1]

TECHNICAL REPORT NO: 2004 / XXX

March 2004

School of Computer Science and Software Engineering
Monash University
Clayton, Vic 3800
Australia

[1] **SpenserKao@optusnet.com.au**

Majix Enhancement

Spenser H. Kao

School of Computer Science and Software Engineering
Monash University, Clayton, Vic 3800, Australia
Spenser.Kao@infotech.monash.edu.au

Abstract

`Majix` is a software tool licensed under Mozilla Public Licence 1.1 and designed to convert RTF input into XML format. It converts RTF styles and some document characteristics into a XML file conforming to a DTD-based default template. Out of the default template, the names of the XML tags may be changed by the user. But an outstanding issue is its functionality of fully converting *bullets and numberings* items into expected XML elements: the latest `Majix` (version 1.2.2, released November 15, 2000) cannot recognise *bullets and numberings* items that are created with compliance of RTF specifications beyond version 1.3.

This document proposes a solution that `Majix` can be enhanced to support *bullets and numberings* items that are created compliant with post-version 1.3 RTF specifications.

The core of enhancement is the revision of two of `Majix's` java classes with two modified methods and 12 new ones. The test result suggests that the enhancement meets the expectation of fully converting *bullets and numberings* items into expected XML elements.

Keywords:
Majix, Bullets and Numbering, RTF.

1. Introduction

1.1. Definition

Two technical terms of RTF specifications [1, pp. 4] need clear emphasis here before being referenced across the report:

- **Control Word** A *control symbol* consists of a backslash followed by a single, nonalphabetic character. For example, \~ represents a nonbreaking space. Control symbols take no delimiters.
- **Destination** Certain control words, referred to as *destinations*, mark the beginning of a collection of related text that could appear at another position, or destination, within the document. Destinations may also be text that is used but should not appear within the document at all. An example of a destination is the \footnote group, where the footnote text follows the control word. Page breaks cannot occur in destination text.

1.2. Majix

Licensed under Mozilla Public Licence 1.1, `Majix` [2] was originated from `tetrasix.com`, but then shifted to SourceForge.net on 24 Feb. 2004. It enables the transformation of RTF [1] document input into XML format by converting RTF styles and some documents characteristics, such as document title, author and operator, into a XML file that conforms to a default `mydoc.dtd`-based template. The `mydoc.dtd` is a DTD (Document Type Definition) [3].

Out of the default template, `Majix` allows the renaming of the tags in the generated template XML through an accompanied user interface. For instance, a default tag `<p>`, which is used to mark up a parsed paragraph, can be renamed to a user-preferred tag, e.g., `<para>`.

`Majix`'s functionalities, including RTF styles parsing, template conversions, tools, XML templates handling and utilities, are implemented by java classes distributed through 140 source files. Two of the source files, `RtfAnalyzer.java` and `RtfReader.java`, are involved in the processing of control words associated with the *bullets and numbering* items.

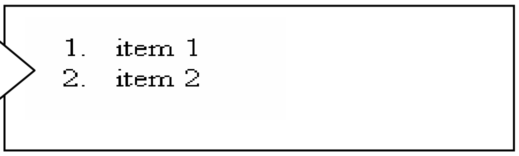
`Majix`'s java source files are part of the zip archive that can be downloaded from its web site [2].

1.3. Related Examples in HTML

HTML [4] markup tags `<ol>` and `<li>` can form a *numbering list* that resembles the *ordered list* of RTF format. Listing 1-1 shows such a HTML code example while Listing 1-2 is for the rendered result.

```
<ol>
<li>item 1</li>
<li>item 2</li>
</ol>
```

Listing 1-1: An HTML<ol> and <li> example.



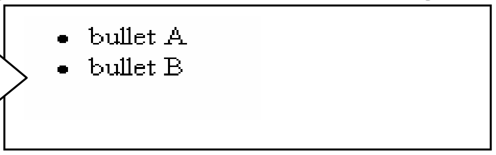
1. item 1
2. item 2

Listing 1-2: Rendered result of Listing 1-1

On the other hand, `<ul>` and `<li>` constitute a *bullets list*, which is similar to an *unordered list* of RTF specification, as in Listing 1-3, and is rendered as in Listing 1-4.

```
<ul>
<li>bullet A</li>
<li>bullet B</li>
</ul>
```

Listing 1-3: An HTML<ul> and <li> example



- bullet A
- bullet B

Listing 1-4: Rendered result of Listing 1-3

1.4. Unrecognisable list table and list override table

Through some simple tests, it has been pinpointed that the latest version 1.2.2 of `Majix` we use fails to support *bullets and numberings* items that are created compliant with the latest RTF specifications 1.7 [1] and 1.5 [5]. A sample of a block of *bullets and numbering* items are created by an authoring tool with RTF specification 1.7 [1] compliance, as rendered in Figure 1-1. Internally, the paragraph parts of the sample are saved in RTF format as in Listing 1-3. If the sample block is to be transformed by `Majix 1.2.2` to its proprietary XML (eXtensible Markup Language) [6] type `mydoc` template, ideally the output has to be as that in Listing 1-4. But in reality the output is as that in Listing 1-5.

```
Numbering items
  1. item 1
  2. item 2

Bullets items
  •  bullet A
  •  bullet B
```

Figure 1-1: Paragraph text of numbering and bullets items in RTF format.

```
\pard\plain \ql
\li0\ri0\nowidctlpar\faauto\rin0\lin0\itap0\pararsid15991558
\fs24\lang1033\langfe1033\cgrid\langnp1033\langfenp1033
{\insrsid15991558 List numbering items … }

\pard \ql \fi-429\li720\ri0\nowidctlpar{\*\pn
\pnlvlblt\ilvl0\ls5 … {
\insrsid15991558 item 1} … }

\pard \ql \fi-429\li720\ri0\nowidctlpar{\*\pn
\pnlvlblt\ilvl0\ls5  … {
\insrsid15991558 item 2} … }

\pard \ql
\li0\ri0\nowidctlpar\faauto\rin0\lin0\itap0\pararsid15991558
{\insrsid15991558 \par List bullet items … }

\pard \ql \fi-429\li720\ri0\nowidctlpar{\*\pn
\pnlvlblt\ilvl0\ls6 … {
\insrsid15991558 bullet A} … }

\pard \ql \fi-429\li720\ri0\nowidctlpar{\*\pn
\pnlvlblt\ilvl0\ls6  … {
\insrsid15991558 bullet B … }
```

Listing 1-3: Contents of an individual paragraph saved in RTF.

The phrases in violet “List numbering items”, “item 1”, “item 2”, “List bullet items”, “bullet A” and “bullet B” are not part of control information and are treated as document text. As mentioned earlier, the rendered output is shown in Figure 1-1.

```
<mydoc>
<p>List numbering items</p>
<list style='numeric'>
<item><p>item 1</p></item>
<item><p>item 2</p></item>
</list>
<p>List bullet items</p>
<list style='bullet'>
<item><p>bullet A</p></item>
<item><p>bullet B</p></item>
</list>
</mydoc>
```

Listing 1-4: Expected mydoc.dtd-compliant XML output from Figure 1-3

```
<mydoc>
<p>List numbering items</p>
<p>.item 1</p
<p>.item 2</p>
<p>List bullet items</p>
<p>&#xb7;bullet A</p>
<p>&#xb7;bullet B</p>
</mydoc>
```

Listing 1-5: Actual XML output by `Majix 1.2.2` from Figure 1-3

According to the result in Listing 1-5, the formatting information (as the example in Listing 1-3) of *bullets and numberings* that is created compliant with RTF specification 1.7 is not recognised by `Majix,` nor reflected onto XML output as expected.

Another possible approach to probe the failed recognition is to take the perspective of the versions of authoring tool. We may, if possible, try to test all (historic and current) versions of the authoring tool to find out since which version the creations of *bullets and numberings* can no longer expect full support from `Majix.`

But other than strenuously acquiring all versions of authoring tools to do thorough tests, there must be a more generic way of verifying the failed recognition. One approach is to take a view from the version of RTF specification.

According to latest RTF specification 1.7 [1, pp.21], Word 97, Word 2000, and Word 2002 store *bullets and numberings* information very differently from earlier versions of Word. In Word 6.0, for example, number formatting data is stored individually with each paragraph. In Word 97 and later versions, however, all of the formatting information is stored in a pair of document-wide *List tables* that act as a style sheet, and each individual paragraph stores only an index to one of the tables, like a style index. The same specification also emphasises that there are two list tables in Word: the *List table* (destination **\listtable)**, and the *List Override table* (destination **\listoverridetable).**

The RTF specification 1.5 [5, pp.15] gives a nearly identical description, except it implies the addition of the use of *List table* since Word 97.

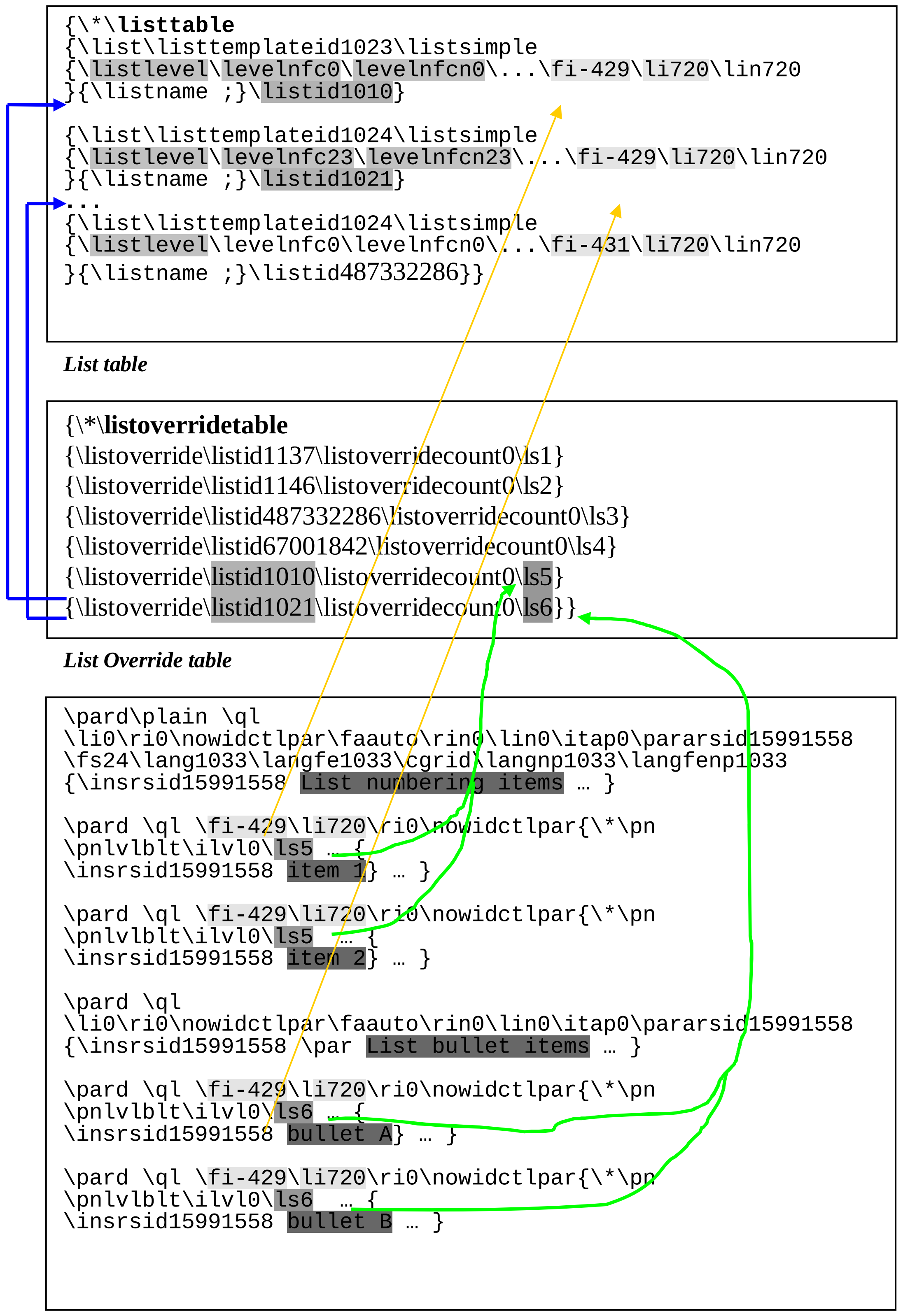
The earlier RTF specification 1.3 [7, pp.18 and 23-25], does not mention anything about *List table*; rather it uses control words **\pntext** and the associates to precede all numbered/bulleted paragraphs.

Semantically searching 140 java source files of `Majix` has revealed that `RtfAnalyzer.java`, which is the class file responsible parsing control words as for preparation for later XML transformation, implements no handling of *List table* (control word **\listtable**), nor of *List Override table* (**\listoverridetable**).

Therefore it can be assumed that `Majix` recognises *bullets and numberings* only up to that specified by RTF specification 1.3. In other word, `Majix` needs to recognise the *List table* and the *List Override table* if it wants to support *bullets and numberings* up to the latest RTF specifications.

1.5. Bullets- and Numbering-Related Information

It is not the objective of this document to elaborate the whole theory of RTF, rather the focus is on the *bullets and numberings* information. While trying to decipher the information related to *bullets and numberings*, it would be less recondite if collated with real data. As an internal view to the sample block at Figure 1-1, Figure 1-2 shows three blocks of bullets- and numbering-related information: the *List table*, the *List Override table* and individual paragraph.



```
{\*\listtable
{\list\listtemplateid1023\listsimple
{\listlevel\levelnfc0\levelnfcn0\...\fi-429\li720\lin720
}{\listname ;}\listid1010}

{\list\listtemplateid1024\listsimple
{\listlevel\levelnfc23\levelnfcn23\...\fi-429\li720\lin720
}{\listname ;}\listid1021}
...
{\list\listtemplateid1024\listsimple
{\listlevel\levelnfc0\levelnfcn0\...\fi-431\li720\lin720
}{\listname ;}\listid487332286}}
```

List table

```
{\*\listoverridetable
{\listoverride\listid1137\listoverridecount0\ls1}
{\listoverride\listid1146\listoverridecount0\ls2}
{\listoverride\listid487332286\listoverridecount0\ls3}
{\listoverride\listid67001842\listoverridecount0\ls4}
{\listoverride\listid1010\listoverridecount0\ls5}
{\listoverride\listid1021\listoverridecount0\ls6}}
```

List Override table

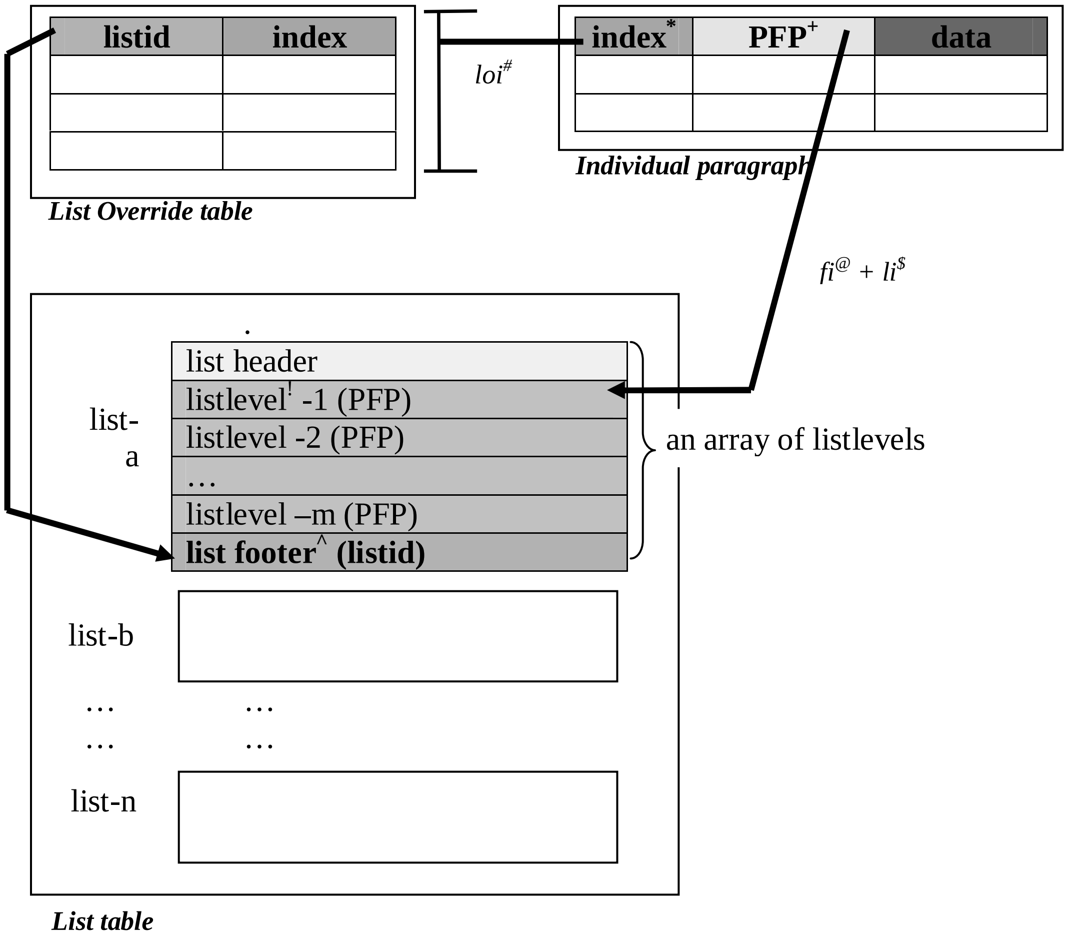
```
\pard\plain \ql
\li0\ri0\nowidctlpar\faauto\rin0\lin0\itap0\pararsid15991558
\fs24\lang1033\langfe1033\cgrid\langnp1033\langfenp1033
{\insrsid15991558 List numbering items … }

\pard \ql \fi-429\li720\ri0\nowidctlpar{\*\pn
\pnlvlblt\ilvl0\ls5  … {
\insrsid15991558 item 1} … }

\pard \ql \fi-429\li720\ri0\nowidctlpar{\*\pn
\pnlvlblt\ilvl0\ls5  … {
\insrsid15991558 item 2} … }

\pard \ql
\li0\ri0\nowidctlpar\faauto\rin0\lin0\itap0\pararsid15991558
{\insrsid15991558 \par List bullet items … }

\pard \ql \fi-429\li720\ri0\nowidctlpar{\*\pn
\pnlvlblt\ilvl0\ls6  … {
\insrsid15991558 bullet A} … }

\pard \ql \fi-429\li720\ri0\nowidctlpar{\*\pn
\pnlvlblt\ilvl0\ls6  … {
\insrsid15991558 bullet B … }
```

User paragraphs

Figure 1-2: An anatomy of Bullets and Numbering Information of sample list input.

1.6. Working Theory of *List Tables*

Using the syntax of the *List table* and the *List Override table* [1, pp.21-25], the information blocks in Figure 1-2 simplified with the data structure view below have been to illustrate how *List table, List Override table* and individual paragraph link to each other.



Note:
loi= List Override Index
$ *li*= Left Indent
@ *fi*= Firstline Indent
* Index points to one of the list that contains a group of *listlevels*.
+ PFP stands for *Paragraph Formatting Property* (*PFP*), whose two major components *first-line indent* (*fi*) and *left indent* (*li*) are used to further refine the list style to *listlevel.*
! *Listlevel* contains *Paragraph Formatting Property*.
^ *List footer* contains *listid*.

The two arrows denote the pointings to the refined *listlevel* (via *PFP*) and *list footer* (via *listid*) respectively.

Figure 1-3: Links between List table, List Override table and individual paragraph.

A *List table* is a collection of *lists* (destination **\list**), such as `list-a` to `list-n` in Figure 1.3. Each *list* contains a number of *list properties* (the list header and footer) that pertain to the entire *list*, and a collection of *list levels* (destination **\listlevel**), such as `listlevel-1` to `listlevel-m` in Figure 1-3.

Each *list level* contains (a) *Paragraph Formatting Properties* (PFP) [1, pp.46] that pertains to that *level*, especially the differentiating *Firstline Indent* (keyword **\fi**) value and *Left Indent* (keyword **\li**) value; and (b) *Number Type Value* (keyword **\levelnfc** or **\levelnfcn**). According to [1 , pp.23], the pertaining items will be *bullet* type (no number at all) if the *Number Type Value* is 23; otherwise, *numbering* type.

Next to *List table* is the *List Override table* which is an array of *List overrides* (destination **\listoverride**). Each *List override* contains the **listid** of one (such as `list-a` in Figure 1-3) of the lists in the *List table*, and the **loi** (List Override Index) that pertains to that *list override*. The **loi** values in the list override table are ordered from low to high, as exampled in Figure 1-2.

As the information of the *List table* and the *List Override table* is applicable document-wide, the two main tables are physically stored in pairs as a style sheet ahead of individual paragraph, each of which stores only an index pointing to one of the entries of *List Override table*, like a style index.

Each paragraph has its own style (*bullet* or *numbering*) and is determined by a two-stage refinement process with the following pertaining information: (a) a *List override index* (keyword **\ls),** which is a 1-based index pointing to a *List override* entry, whose **listid** matches a list entry in the *List table*; and (b) a PFP, which comprises the values of **fi** and **li**, to further identify to a *listlevel* out of the matched list. Once target *listlevel* is found, expected number type value can be derived through keyword **`\levelnfc`** or **`\levelnfcn`**.

2. Design

At runtime, the formatting information of *List table* and *List Override table* needs to be available for lookup when processing succeeding individual paragraph. Also as mentioned above that physically the two main tables are stored in pair as a style sheet ahead of individual paragraph in a file that will be only parsed once top to down, they can not be available for lookup unless made memory based once parsed. To make the formatting information memory based, one optimal way is to store them in numerous `HashMaps` (if Java language is chosen) or the equivalents in other languages.

Each insertion to `HashMap` [8] takes an input pair of key and value, both of which are required to be object (String, Integer and etc) and can be null. Numerous `HashMap` instances shall be created for the following three purposes: the *List table*, the *List Override table* and the *Listlevel* groups. These `HashMap` instances form a three-tier data structure as illustrated in Figure 2-1.

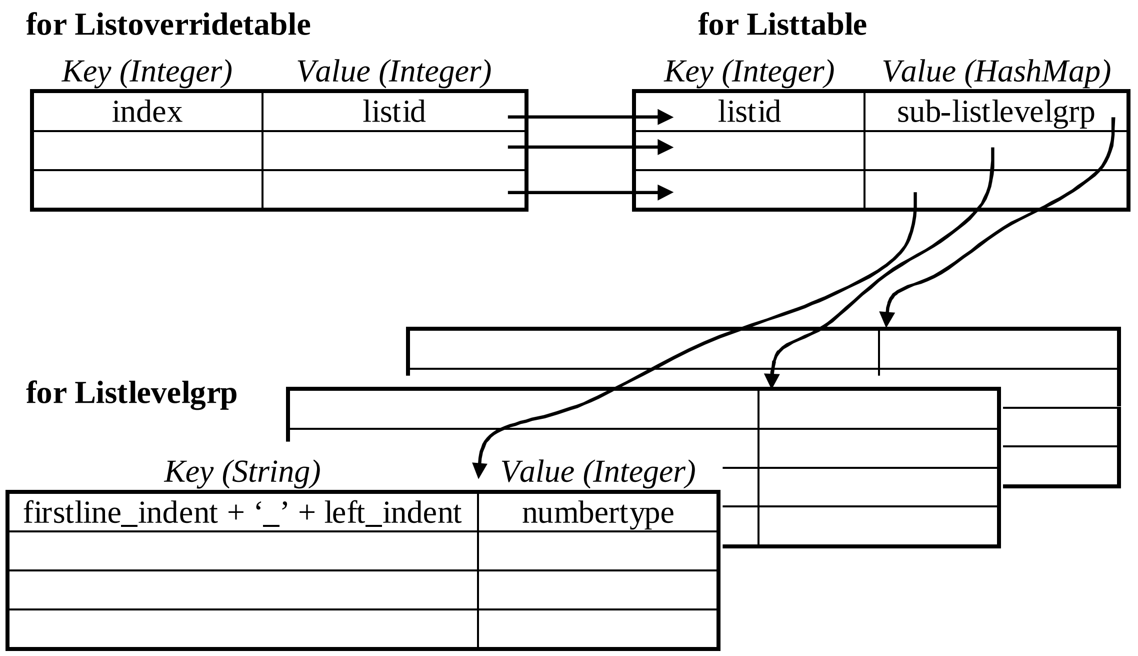


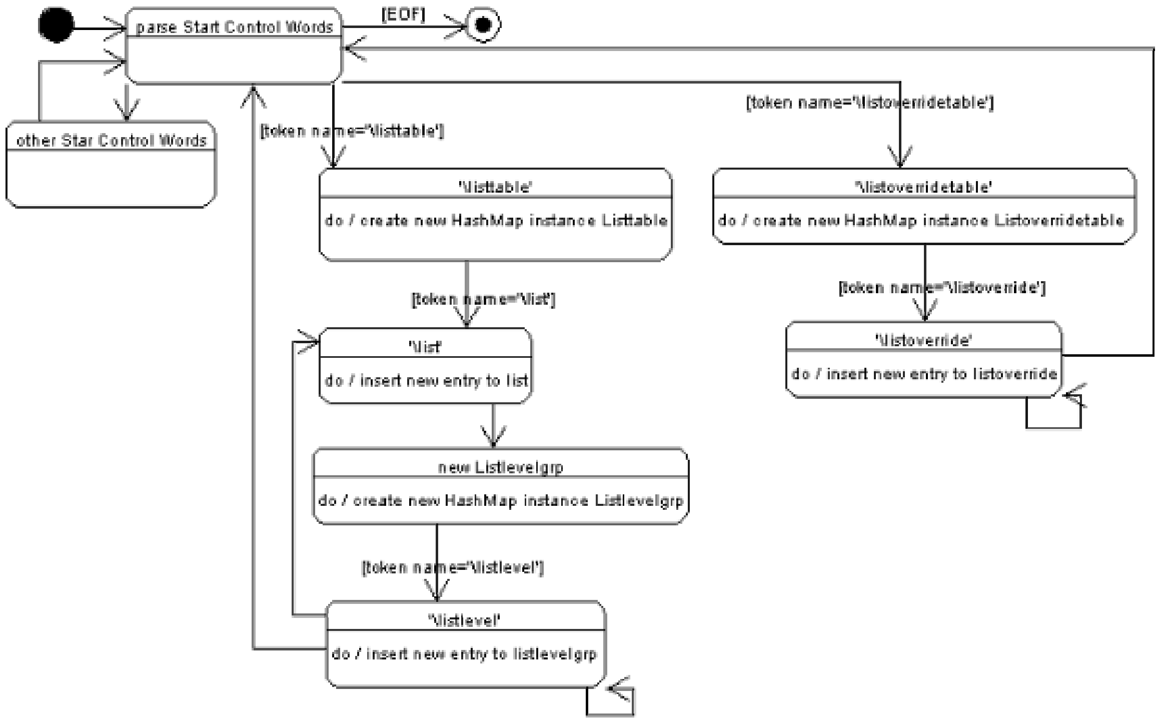
Figure 2-1: Data structure diagram – for showing relationship among HashMap instances.

The arrows in the diagram denote the reference pointers between the `HashMap` instances.

From the perspective of state transition, Figure 2-2's depicts that the `HashMap` instances are created in the order of their appearances in the file:

a) `Listtable` for the *List table*;
b) multiple `Listlevelgrps` for the `listlevels` that are extracted from the *List table*; and
c) `Listoverridetable` for the *List Override table*.

The State Diagram (STD) is created by the tool `Poseidon for UML Community Edition` [9].

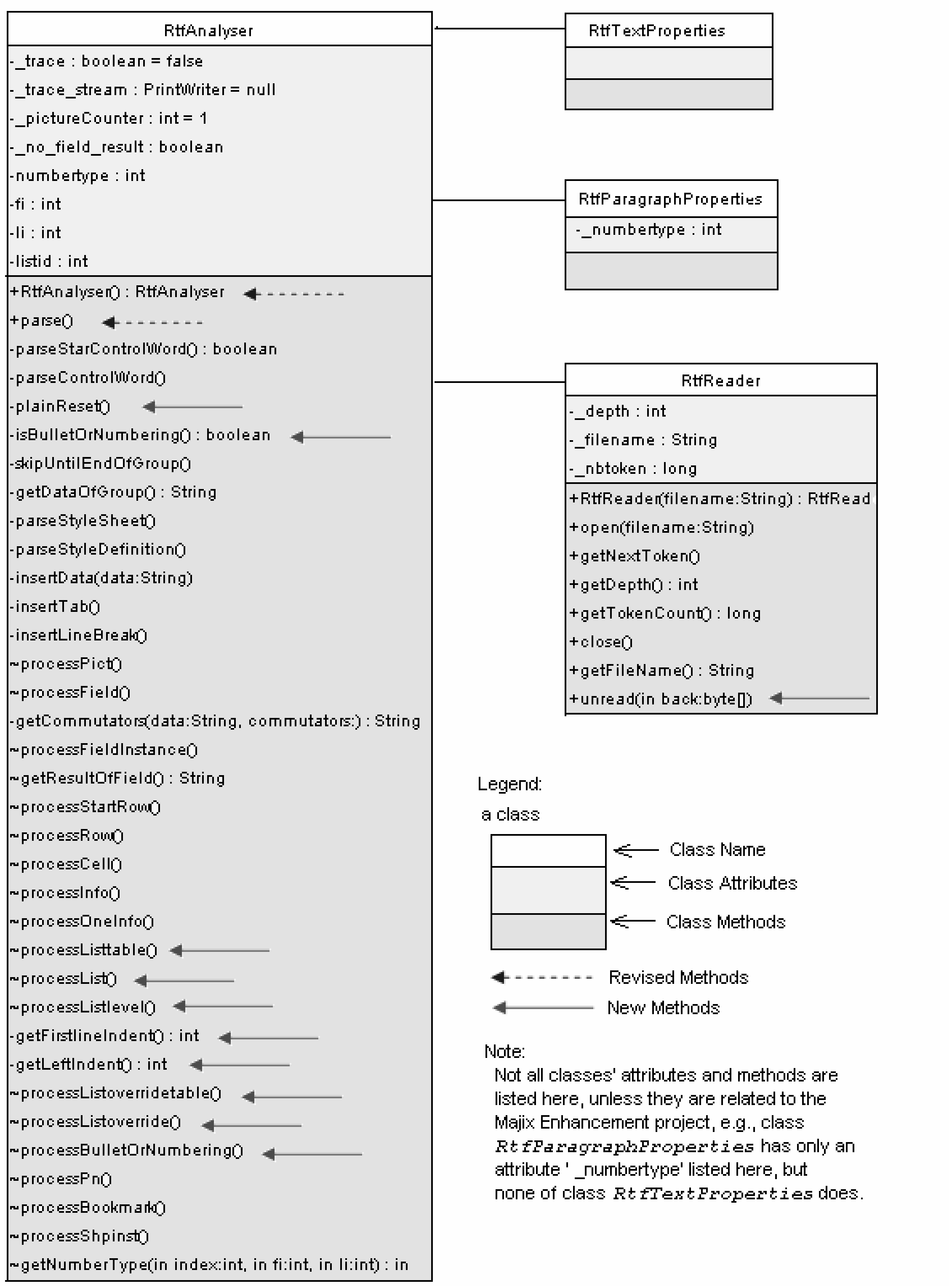


parse Start Control Words
[EOF]
other Star Control Words
[token name='\listtable']
[token name='\listoverridetable']
'\listtable'
do / create new HashMap instance Listtable
'\listoverridetable'
do / create new HashMap instance Listoverridetable
[token name='\list']
[token name='\listoverride']
'\list'
do / insert new entry to list
'\listoverride'
do / insert new entry to listoverride
new Listlevelgrp
do / create new HashMap instance Listlevelgrp
[token name='\listlevel']
'\listlevel'
do / insert new entry to listlevelgrp

Figure 2-2: State diagram – for the creation order of memory-based List tables.

3. Implementation

3.1. Class Diagram



Note:
Not all classes' attributes and methods are listed here, unless they are related to the Majix Enhancement project, e.g., class `RtfParagraphProperties` has only an attribute ' _numbertype' listed here, but none of class `RtfTextProperties` does.

Figure 3-1: Class diagram of `RtfAnalyser` *and* `RtfReader`

Two modified methods and 12 newly added are involved with the implementation of processing the *List tables* and *List overribe tables*. The 14 methods belong to two java classes `RtfAnalyser` and `RtfReader`, situating two separate source files among total 140 of `Majix`: `RtfReader.java` and `RtfAnalyser.java`.

Class `RtfAnalyser` accommodates methods performing most of the jobs of parsing the *List tables* and *List overribe tables.* Among them, two are modified:

`parseStarControlWord` and **`parseControlWord`**,

as marked by blue or dash arrows in Figure 3-1 class diagram. In the same class, 11 new methods, as marked by red or solid arrows, are added:

`plainReset, isBullerOrNumbering, processListtable, processList, processListlevel, getFirstlineIndent, getLeftIndent, processListoverridetable, processListoverride, processBulletOrNumbering` and **`getNumberType`**.

Class `RtfReader` is to be instantiated by the methods of class `RtfAnalyzer` for reading tokens out of control words pipe through its existing method `getNextToken`. Only one method is added to the class for pushing token back to the pipe when required:

`Unread`.

The class diagram is created by the tool `Poseidon for UML Community Edition` [9].

3.2. Data Flow Diagram

Figure 3-2 is a Data Flow Diagram (DFD) that is a concept adopted from classic Structural Analysis [10] to provide an in-depth view of the data flowing between processes inside the classes: `RtfAnalyser` and `RtfReader`. The Data Flow Diagram in Figure 3-2 is created by the tool `CASE Studio 2 demo version` [11].

Each bubble in the Data Flow Diagram stands for a process, while each pair of parallel bars represents a data following into or from neighbouring processes.

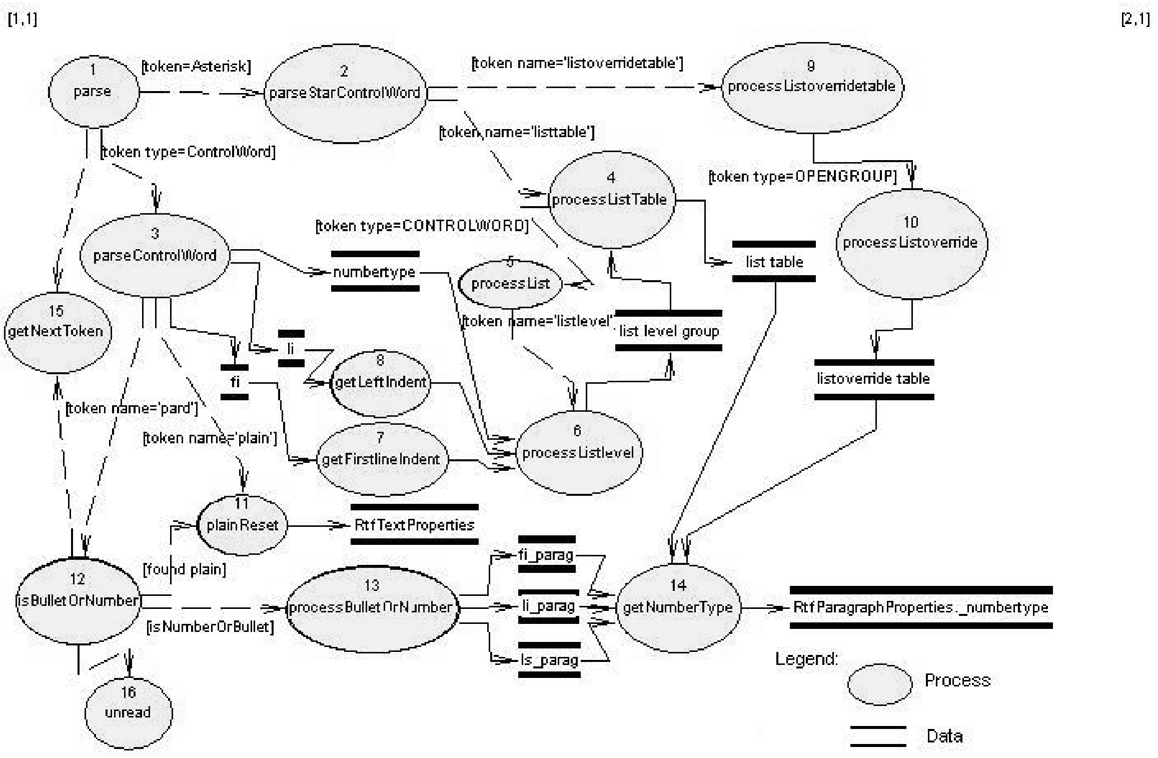


Figure 3-2: Data flow diagram – of class `RtfAnalyser` *and* `RtfReader`

For the purpose of processing *bullets and numbering* formatting information, the 16 methods act together to fulfill two major tasks: (a) proliferating the memory-based *List tables*; and (b) processing individual paragraph through looking up the memory-based *List tables*.

The proliferations of the memory-based *List tables* are achieved by following methods: `parse` (process#1), `parseStarControlWord`(2), `parseControlWord` (3), `processListTable`(4), `processList`(5), `processListlevel`(6), `getFirstlinelIndent`(7), `getLeftIndent`(8), `processListoverridetable`(9), `processListoverride`(10) and `getNextToken(15)`.

The processing of individual paragraph is in the corporation of the methods: `plainReset`(11), `isBulletOrNumber`(12), `processBulletOrNumber`(13), `getNumberType` (14), `getNextToken(15)` and `unread(16)`.

As these methods are all self explanatory, with the aid of the state diagram, class diagram and data flow diagram explained above, it should not be too difficult to picture how these methods are implemented. To obtain the source code of class *RtfAnalyzer*, please contact the author.

4. Building

The building of the `Majix Enhancement` is facilitated through `Ant` [12], with as the configuration file `build.xml` (Appendix A) to specify rules such as cleanup, dependence, compilation and archiving. In theory, `Ant` is similar to UNIX command `Make`, but more verbose and machine processable – as its configuration file is XML-based. `Ant` is a Java-based build tool and has the full portability of pure Java code.

A DOS batch file (namely `runant.bat)` that is a wrapper around `Ant` under Windows environment can be found at Appendix B. An environment setting batch file `env.bat` that is called by `runant.bat` can be found at Appendix C.

4.1. Rules of Building

With the guidance of the `target` elements of the `build.xml`, the command syntax of rebuilding the `Majix Enhancement` is as follows:

```
     runant [clean] [compile] [jar]
```

The `runant(.bat)` is a DOS batch file wrapping around `Ant` under Windows environment and can be found at Appendix B for its content. An environment setting batch file `env.bat` that is called by `runant.bat` can be found at Appendix C.

The meanings of `runant`'s options are:

- `clean` – for cleaning up existing class and jar files
- `compile` – with the dependence on `clean` target, for compiling source codes into class files
- `jar` – with the dependence on `compile` target, for archiving the class files into one jar file

These options can be specified in combination, but the dependence rules will result in that the last specified option takes real effect. The options are the reflection of the three `target` elements, in the `build.xml,` that have the values of the attribute `name` correspondent to the three options as excerpted below:

```
<target name="clean" ...>
    <delete ...>
    ...
    </delete>
</target>
<target name="compile" ...>
    <javac ...>
    ...
    </javac>
</target>
<target name="jar" ...>
    <jar ...>
    ...
    </jar>
</target>
```

According to the `build.xml`, when there is no option specified, the `target` element with `name` attribute that has following value will take effect: `all`, as which is the *default* target and depends on the three targets `clean`, `compile` and `jar`, which will be executed in order. The default rule is excerpted below:

```
<project name="Ant" default="all" ...>
...
<target name="all"
      depends="clean, compile, jar"
      ... />
</target>
</project>
```

5. Test

A DOS batch file `fft2st2.bat`, which can be found at Appendix D, performs a two-stage transformation: (1) convert RTF input to intermediate XML output through `Majix.jar;` and (2) transform the intermediate XML into domain-specific XML target.

Just for the purpose of testing *bullets and numberings,* the error message coming through the stage-2 must be ignored as the parameters required by stage-2 transformation have been deliberately omitted.

Here is the command syntax for testing:

```
fft2st2 <src_rtf>
```

Once through stage 1, a file namely `interm.xml` is generated at the local directory and contains the expected intermediate XML output.

One test input been created as in Appendix E to contain four styles: three numbering styles and one bullet. The test output can be found at Appendix F.

6. Discussion

Based on its current specification, `Majix 1.2.2`, is not capable of collecting the number type and bullet symbol [1, pp.23] from the RTF input source. Although it is more of presentational issue, it is still the will of the document authors to decide in what format and style the items should be enumerated or bulleted. We may consider implementing the functionality of collecting number type and bullet symbol it in the next version of `Majix`'s enhancement.

The delimiters, such as the dot "." and code &#xb7 (a "middle dot") in Listing 1-5 and Appendix F, are part of the number type and bullet symbol, rather than being treated as items' content as they are now. Therefore, in the next version enhancement, the `list` element of the intermediate XML output shall contain two more attributes in addition to the existing `style`: `type` and `delimiter`, where attribute `type` is not required if the `list` element is for bullet items.

With the new functionality implemented, the examples of the `list` element will be as follow:

`<list style=”numeric” type=”1” delimiter=”.”>` for reflecting input such as

```
Numbering items
  1. item 1
  2. item 2
```

; or `<list style=”numeric” type=”A” delimiter=”,”>` for reflecting input such as

```
Please follow steps below:
   A, insert the floppy
   B, press the Go button
```

; or <list style=”numeric” delimiter=”&#xb7”> for reflecting input such as

```
Bullet items
   . Mongo
   . Apricot
```

According to [1, pp.7], the numeric parameter ***N*** in the **\rtfN** control word only identifies the major version of the compliant RTF specification. This means that whenever it is necessary to tell with which version of RTF specification the underlying source file is compliant with, there is no way of doing that, as the ***N*** will be always “1”, the major number, regardless of whether the real version is 1.7, or 1.5, or even earlier 1.3. That is just one of many generic issues associated with the RTF specifications.

7. Conclusion

Comparing the RTF input source at Appendix E with the test result at Appendix F, the numbering types are truly reflected in the output. This result supports the claim that the enhancement to `Majix` in the regard of *bullets and numbering* is working as expected.

The current `Majix` enhancement is implemented and functions well under the Windows 2000/XP environment. With minimum modifications of DOS batch files to shell scripts files, the porting of `Majix` enhancement to UNIX environment should not be difficult as the java codes are platform natural. The source code of this enhancement can be obtained by contacting the author through the email address indicated in the cover page.

8. References

[1] Microsoft. (2001). *Rich Text Format Specification 1.7.* URL: http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=E5B8EBC2-6AD6-49F0-8C90-E4F763E3F04F (Accessed: 5 Apr 2003).

[2] SourceForge.net. (2004). *Majix.* URL: URL: http://sourceforge.net/projects/majix/ (Accessed: 28 Mar 2004).

[3] Software AG. (2003). XML Glossary - *Document Type Definition.* URL: http://www.softwareag.com/xml/about/glossary.htm#D (Accessed: 28 Mar 2004).

[4] NCSA. (2003). *A Beginner's Guide to HTML.* URL: http://archive.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerAll.html#LI (Accessed: 16 Jan 2004).

[5] Microsoft. (1997). *Rich Text Format Specification 1.5.* URL: http://www.snake.net/software/RTF/RTF-Spec-1.5.rtf (Accessed: 5 Apr 2003).

[6] W3C. (2003). *XML – eXtensible Markup Language 1.0* (2nd Ed.). URL: http://www.w3.org/TR/REC-xml (Accessed: 21 Nov 2003).

[7] Microsoft. (1994). *Rich Text Format Specification 1.3.* URL: http://www.snake.net/software/RTF/RTF-Spec-1.3.rtf (Accessed: 5 Apr 2003).

[8] Java Technology. (2003). *Class HashMap*. URL: http://java.sun.com/j2se/1.4.2/docs/api/ (Accessed: 23 Dec 2003).

[9] Gentleware. (2003). *Poseidon for UML.* URL: http://www.gentleware.com (Accessed: 25 Dec 2003).

[10] T. DeMarco. *Structural Analysis and System Specification.* Yourdon Press, 1978.

[11] Charonware. (2003). *Case Studio.* URL: http://www.casestudio.com/enu/download.aspx (Accessed: 25 Dec 2003).

[12] Apache Ant Project. (2003). *Apache Ant.* URL: http://ant.apache.org/ (Accessed: 24 Dec 2003).

Appendix A - `Build.xml`

```
<?xml version="1.0" ?>
<!-- =============================================================== -->
<!--  Ant own build file                                             -->
<!-- =============================================================== -->
<project name="Ant" default="all" basedir=".">
<property name="src.dir" value="source" />
<property name="lib.dir" value="lib" />
<property name="build.classes" value="classes" />
<property name="build.javadocs" value="javadoc" />
<property name="manifest" value="${src.dir}/manifest.txt" />
<!-- =============================================================== -->
<!--  Delete the binary output                                       -->
<!-- =============================================================== -->
<target name="clean" description="Removes all generated class files.">
<!--  IMPORTANT: ONLY delete files and subdirectories under          -->
    <delete includeEmptyDirs="true">
        <fileset dir="${build.classes}"
        includes="**/*" defaultexcludes="no" />
    </delete>
</target>
<!-- =============================================================== -->
<!--  Compile the source code                                        -->
<!-- =============================================================== -->
<target name="compile">
    <mkdir dir="${build.classes}" />
    <javac srcdir="${src.dir}" destdir="${build.classes}"
    classpath="${classpath}" debug="on" deprecation="on"
    optimize="off">
    <classpath>
    <pathelement location="${build.classes}" />
    <pathelement location="${lib.dir}/xp.jar" />
    <pathelement location="${lib.dir}/xt.jar" />
    <pathelement location="${lib.dir}/xerces.jar" />
    <pathelement location="${lib.dir}/xalan.jar" />
    <pathelement location="${lib.dir}/saxon71.jar" />
    <pathelement location="${lib.dir}/jacob.jar" />
    </classpath>
    <include name="**/*.java" />
    </javac>
</target>
<!-- =============================================================== -->
<!--  Creates the jar archive                                        -->
<!-- =============================================================== -->
<target name="jar" depends="compile">
    <mkdir dir="${lib.dir}" />
    <jar jarfile="${lib.dir}/majix.jar" basedir="${build.classes}"
    includes="com/**" manifest="${manifest}" />
</target>
<!-- =============================================================== -->
<!--  The default job                                                -->
<!-- =============================================================== -->
<target name="all"
      depends="clean, compile, jar"
      description="clean, compile and jar." />
</target>
</project>
```

Appendix B - `Ant` wrapper `runant.bat`

```
@echo off
setlocal

call env.bat

if exist %JAVA_HOME% goto check_rtf2fo_home
echo %JAVA_HOME% does not exists.
goto done

:check_rtf2fo_home
if exist %RTF2FO_HOME% goto check_ant_home
echo %RTF2FO_HOME% does not exists.
goto done

:check_ant_home
if exist %ANT_HOME% goto check_antlr_home
echo %ANT_HOME% does not exists.
goto done

:check_antlr_home
if exist %ANTLR_HOME% goto check_jdom_home
echo %ANTLR_HOME% does not exists.
goto done

:check_jdom_home
if exist %JDOM_HOME% goto ok
echo %JDOM_HOME% does not exists.
goto done

:ok

set CLASSPATH=%CLASSPATH%;%ANT_HOME%\ant.jar
set CLASSPATH=%CLASSPATH%;%ANTLR_HOME%\antlr.jar
set CLASSPATH=%CLASSPATH%;%JDOM_HOME%\jdom.jar
set CLASSPATH=%CLASSPATH%;%PARSER_HOME%\xerces.jar
set CLASSPATH=%CLASSPATH%;%RTF2FO_HOME%\jar\rtf2fo.jar
set PATH=%ANT_HOME%\bin;%PATH%

set CLASSPATH
set PATH

echo on
ant -buildfile build.xml %*
@echo off

:done

endlocal
```

Appendix C - Environment Setting `env.bat`

```
SET JAVA_HOME=D:\tools\j2sdk1.4.2_01
SET ANTLR_HOME=D:\tools\antlr
SET JDOM_HOME=D:\tools\jdom
SET PARSER_HOME=D:\tools\xmlparser
SET ANT_HOME=D:\tools\ant
SET FFT2ST_HOME=D:\Projects\FFT2ST\ANT_test
SET SAXON_HOME=D:\tools\saxon
SET FOP_HOME=D:\tools\fop
SET RTF2FO_HOME=D:\tools\rtf2fo
SET LOG4J_HOME=D:\tools\fop
SET JFO_HOME=D:\tools\jfo
SET XALAN_HOME=D:\tools\xalan
SET JXRTF_HOME=D:\tools\jxrtf2_0a2
```

Appendix D - `fft2st2.bat`

```
@echo off
REM 17-Dec-2003 v1.2
REM   replaced fo2xml with hope to execute the stage-2 transformation.
REM
IF "%*"=="" (
echo usage: fft2st2 {src_RTF} {style_XSL} {dict_OWL} {schema_XSD}
[{output_XML}]
GOTO end2
)

del interm.xml
echo
*******************************************************************
echo * Stage 1: RTF to mydoc-dtd compilant intermediate XML conversion
*
echo *      by: Majix
*
echo
*******************************************************************
java -cp D:\tools\majix\lib\majix.jar com.tetrasix.majix.MajixBatch -
styles D:\tools\majix\default.sty %1 interm.xml
IF not exist interm.xml GOTO done
echo
echo *****************************************************************
echo * Stage 2: Intermediate XML to Domain-Specific XML              *
echo *      by: SAXON wrapper 'hope'                                 *
echo *****************************************************************

IF "%5"=="" (
      hope interm.xml %2 dict="%3" schema="%4"
) ELSE (
      echo Outputing xml to %5 ...
      hope -o %5 interm.xml %2 dict="%3" schema="%4"
)

REM del interm.xml

:done
echo End of Conversions
:end2
```

Appendix E - Test Input

```
List numbering items
     1. item 1
     2. item 2

List numbering items
     I. item I
    II. item II

List numbering items
     a. item a
     b. item b

List bullet items
     ▪   Mongo
     ▪   Orange
```

Appendix F - Test Output

```
<mydoc>
     <p>List numbering items</p>
    <list style="numeric">
        <item><p>.item 1</p></item>
        <item><p>.item 2</p></item>
    </list>
    <p>List numbering items</p>
    <list style="numeric">
        <item><p>item I</p></item>
        <item><p>item II</p></item>
    </list>
    <p>List numbering items</p>
    <list style="numeric">
        <item><p>item a</p></item>
        <item><p>item b</p></item>
    </list>
    <p>List bullet items</p>
    <list style="bullet">
        <item><p>&#xb7;Mongo</p></item>
        <item><p>&#xb7;Orange</p></item>
    </list>
</mydoc>
```