Converting WordStar to HTML4

Jay Gattuso
Jay.Gattuso@dia.govt.nz
National Library of New Zealand
Cnr Molesworth & Aitken St
Wellington
New Zealand

Peter McKinney
peter.mckinney@dia.govt.nz
National Library of New Zealand
Cnr Molesworth & Aitken St
Wellington
New Zealand

ABSTRACT
In this paper, we describe the processing and verification work undertaken to migrate WordStar for MSDOS to HTML4 formatted files.

General Terms
Preservation strategies and workflows, specialist content types, digital preservation marketplace, theory of digital preservation, case studies and best practice.

Keywords
Preservation planning, preservation action, WordStar, HTML, significant properties, acceptable change, converters.

1. INTRODUCTION
In 2012, the National Library of New Zealand's (NLNZ) digital preservation business unit made the decision to undertake its first "in-anger" preservation planning and action activities to mitigate the risk to content that forms part of the Library's collections.

The criteria for the set was defined as (1) the format type should display a significant risk to its future use (2) of a manageable size; (3) from one collection group; and, (4) of a simple construction, (i.e. no compression or complex wrappers/containers).

In assessing the 137 uniquely identifiable formats[1] that currently appear in the digital preservation system, the best fit format for this work was WordStar. NLNZ holds 37 files that have been identified as WordStar for MSDOS formatted files.

We don't know exactly what version of WordStar files these are, as there are no signature based format identifications available for these file types. To that end, searching the system for all files with a .ws file extension would have resulted in the same corpus being constructed.

WordStar for MSDOS would be described by NLNZ as "functionally obsolete", meaning that it is highly unlikely that a normal user would have the tools to open and accurately render the content in a way that was in keeping with its original layout and intent.[2]

They are also all part of the same collection and therefore meet all of the criteria.

We had a second order requirement, to explore what a migration process feels like to all involved parties (technical, curatorial and managerial). This would serve as the starting point for more related activities in the future, and as such we wanted to ensure that we at least understand the basic framework that would underpin future migration work.

2. INITIAL ANALYSIS OF THE CONTENT
It was relatively simple to search the preservation repository for all the WordStar files. Simply searching the ~10 million files in the preservation repository for any of the PRONOM PUIDs that are registered against WordStar formats resulted in 37 files being identified. [3]

These files were retrieved from the system, and inspected in detail to ascertain their composition. The inspection demonstrated the following:

- there is no complex formatting or layout (e.g. tables), just text;
- "normally" encoded UTF-8 text is visible amongst the format structure;
- all the content is of a similar form, and is relatively straightforward to process;
- all files open OK with a not-quite contemporary copy of WordStar for MSDOS (see following paragraph);
- all files are transcripts of audio interviews that belong to our Oral History Unit. These files are highly restricted and cannot be shared outside of the Library at this time[4].

We also managed to locate a computer of approximately the correct age for the corpus. This was part of a relatively unmanaged collection of ICT equipment that has been put aside for testing purposes by the library.

[1] Identified using the DROID file format tools over 5 years.

[2] For a discussion on the Library's view on this, see [4].

[3] PRONOM PUIDs x-fmt/370, x-fmt/260, x-fmt/205, x-fmt/236, x-fmt/237, x-fmt/261, x-fmt/206 and x-fmt/262. [3].

[4] This by itself causes us problems for verifying our work. We cannot share these files for peer review, and any effort to create new sharable files may not result in the same "version" of WordStar files being created. This also precludes the ability to use any online conversion services to test their capability.

The computer is Toshiba Satellite T2130CT (circa 1995, running a 486 intel chipset, and MS Windows v3.11 / MSDOS v6).

We also found a copy of WordStar for MSDOS v5 on ebay.com, which was installed on the machine for testing / reference however, we do not know if any of the WordStar in the corpus were created with this version of WordStar for MSDOS.

WordStar, not unusually for word processors of its era (circa 1986), used a method of displaying markup on screen as tags, not unlike un-rendered HTML, rather than affecting the formatting visually as is more common today (rendered HTML). This means if an author wanted to bold format the words "`I am bold`", it would appear on screen as "`^BI am bold^B`" rather than the formatting being applied directly to the text - "**`I am bold`**". Of course, when the page is printed, the markup tags are not printed, but the intent of the formatting tags are.

Text formatting is not the only marked up feature that can be found in the WordStar formatted content.

Some example files were created on the original hardware, printed, and imaged by our internal image services to allow us to demonstrate the visible difference between the screen view, and the printed page:-

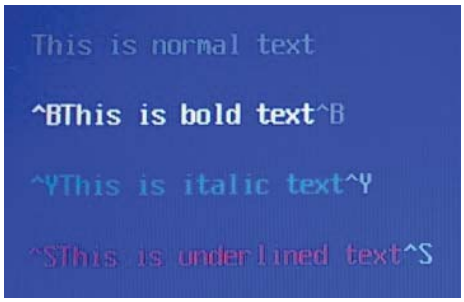


Figure 1 - Screen shot: WordStar on screen markup

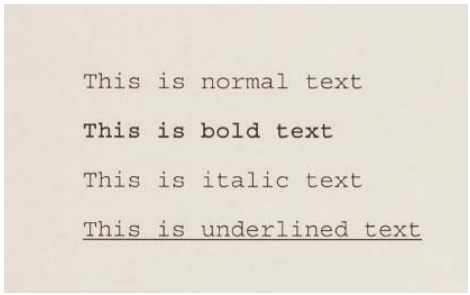


Figure 2 - WordStar printed output

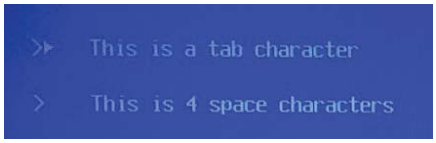


Figure 3 - Screen shot: WordStar on screen markup

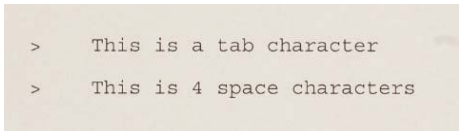


Figure 4 - WordStar printed output

Figure 5 - Screen shot: WordStar on screen markup

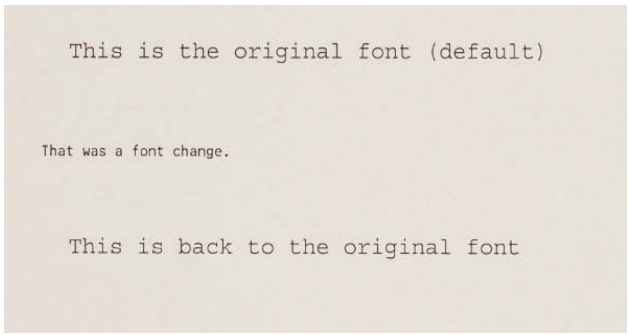


Figure 6 - WordStar printed output

The WordStar files in the corpus were inspected, and the following bytes were found, used in some way by WordStar to convey formatting or other information (see Table 1).

These all appear in the "control word" section of the UTF-8 text encoding standard.[6]

This was achieved by parsing the files in the corpus byte by byte, and returning any bytes that fall outside of the range of UTF-8 code points that have a normally associated printable glyph (`\x20` to `\x7e`)[5].

All of these code points needed to be addressed in some way to ensure that their meaning or purpose is properly conveyed by any converted files where applicable.

It was clear from the tested files opened with a "normal"/ modern text viewer, and the reference version of WordStar for MSDOS, that the control code-points in UTF-8 have an entirely different function in WordStar for MSDOS files.

Their individual functions in the WordStar files are compared to UTF-8 in the table below:-

[5] For the duration of this paper. any hexadecimal byte is represented by the hexadecimal value, preceded by "\x".

Table 1 - Control byte comparison of UTF-8 and WordStar

Bytes	UTF-8	WordStar[6]
\x02	Start of Text [STX]	Toggle Bold Print
\x05	Enquiry [ENQ]	User Print Command
\x0b	Vertical Tab [VT]	Odd/Even Page Offset
\x13	Device Control 3 [DC3]	Toggle Underline
\x14	NL Line Feed, New Line [LF]	Toggle Superscript
\x1a	Substitute [SUB]	End of File Marker
\x8d	Not Valid 8-bit Code [<control>]	Line Terminator on Word Wrap

3. INITIAL TOOL VALIDATION

After the initial content analysis we undertook an initial survey to identify potential tools for the conversion process. We found eight potential application/codecs; however, some simple trials very quickly indicated that none of the converters was able to accurately convert all the files in the set, to any format. The initial testing included two reference files being converted, and of the eight tools tested, only three managed to return even a valid file that could be measured against the original.

Given that we abandoned the testing of any commercially or otherwise available conversion product due to their inability to perform on our test files, the purpose of this paper is not to discuss the various functions and failings of each of the tools, suffice it to say, we could not find a single product that was able to offer us the ability to accurately convert our files.

Exploring these converters led to a complex part of the problem. How can we compare the accurate conversion of file A to a new format? What metrics can we use to convince ourselves of the efficacy of any conversion process?

At this early stage, it was enough to use simple word/character counts as the pass/fail metric. Each WordStar file was stripped of any non-printable UTF-8 characters, and each word/character was tallied. Once converted, and irrespective of the output format of the converter, the resulting text was also stripped of markup and each word/character tallied. These measures were collected and tables such as below were generated (Table 2 and Table 3).

Table 2 - File content metrics for file Ref1.WS

Ref_Name	Ref1.WS	App 1	Exact?
Pages	6	5	FALSE
Words	3639	3639	TRUE
Chars (no spaces)	15906	15906	TRUE
Chars(spaces)	19774	19515	FALSE
Paras	319	32	FALSE
Lines	328	222	FALSE

[6] This information was originally reverse engineered through inspection, and confirmed at a later date when the supporting WordStar (v3) manual [5] was discovered.

Table 3 - File content metrics for file Ref2.WS

Ref_Name	Ref2.WS	App 1	Exact?
Pages	34	32	FALSE
Words	24880	24880	TRUE
Chars (no spaces)	95005	95005	TRUE
Chars(spaces)	122530	119666	FALSE
Paras	1903	361	FALSE
Lines	1954	1434	FALSE

The second phase of testing was to try all the available WordStar files against the three known working tools to ensure that all of the WordStar files could be converted by the tools.

This phase highlighted the inconsistent and unsatisfactory performance of the tools, and a decision was made to write a new converter from scratch.

It is worth noting that the inconsistent performance encountered was largely found to be either a complete failure of the tool to return something usable as a converted file, or an inability to deal with all the variants of WordStar files found in the set.

4. CURATORIAL ENGAGEMENT

Having completed the initial trawl through the set and having a broad understanding of what was possible / viable, the next step was to ensure that curatorial concerns were fully understood.

An assessment template was created that formed the basis for a series of meetings with preservation and curatorial colleagues to ensure that the files under inspection were properly understood and thus properly migrated to a new format.

This assessment allowed the documents to be conceptually broken down into the various formatting and aesthetic features that comprise the original intellectual object and ensure that effort is expended in the right areas.

4.1 Working through the Original Content Review

Any preservation work undertaken by the NLNZ digital preservation team must be ratified by the "content owner". The content owner is the person within the Library who has overall responsibility for the collection items being preserved. The content in question are all transcripts from oral history recordings, so in this case, the owner is the Curator of Oral History and Sound. In addition, we enlisted the help of the digital archivist and assistant digital archivist to act as mediators between the Curator's content expertise and our digital expertise. They also gave valuable insights based on their own experiences.

Once there was agreement on the collection to be used, the next step was to undertake a review of the original content that could be used to measure the success (or not) of any proposed transformations.

Preservation planning must demonstrate that all aspects of the content has been considered and report on those aspects across the transformation. The discussions therefore focused not just on what must remain the same (significant properties), but just as

importantly, what could change: the measurement of a successful transformation must equally show what has and has not changed.[7]

The form we used looked at the mark-up and the formatting of the WordStar content. We could find no previous work on WordStar conversions that would aid this work, but did use various sources such as proof-reading notes for the types of mark-up and text-based features that we should look out for. The process of going through this form was deliberately elongated. This is the first "in-anger" preservation action that the Library has undertaken and we wanted to ensure that we were covering every eventuality and more importantly that the curator was entirely comfortable with what was being done to content. This would give us all confidence that the new representations of the content could stand as a true and accurate replacement for the original.

Our technique was to go through every possible aspect of the WordStar files and discuss their importance. The mutability of each aspect was the key currency for this conversation, for example; could the font-type change? What about margins in the documents? As we went through these twenty six identified aspects two primary issues arose:

1. What *facts* do we know about the original content?
2. Are we replicating the content seen on screen or as it was printed?

The first question speaks to the notion that we were not running equipment completely contemporaneous with the content. The version of WordStar used as our reference (WordStar for MSDOS v5) post-dates when we believe the transcripts were written. All decisions therefore are founded on the fact that we are not viewing the material in a completely original environment.[8]

The second question was only one that the curator could answer. What exactly were we trying to preserve? Is it the look and feel on the screen, including its markup? Or were we in fact preserving the output of the word processed document? We know that these files were created in order to be printed and given to those listening to the original recordings. It was decided that we were preserving how the content would have looked when it was printed.

The medium of presentation was not of any real concern. The Library made the decision a long-time ago that we are not a computer museum. We do not preserve the original hardware to present content nor do we wish to preserve content so that interaction with it is exactly as it was two decades ago. We always plan to represent the content with best efforts to retain original features, but always with an eye to allowing new and future use of it. That is to say, in this case, we were unconcerned with preserving the blue screen with the markup presented to the writer. We were more concerned with presenting the text in a fashion that the creator would recognise as their work. The content is king, not the medium.

4.2 What is a viable metric?

With any automated process, the question of measuring quality / effectiveness of any migration action arose as a concern. The WordStar documents were no different in this regard. Essentially, there is a need to find a "middle ground metric" that would allow the original file to be compared with the new files, and some automated decision making / logging to ensure that migration actions are accurate.

To simplify this process, the WordStar content was conceptually sliced into two concerns; aesthetic construction and intellectual content.

Each concern is considered in isolation of the other, and each has its own pass/fail measures that once satisfied will result in the final outcome fulfilling all concerns.

4.3 Concern 1: Aesthetic construction

This measure is essentially the visual appearance of the intellectual object. It cares not what the content "says"; it cares about capturing the "look and feel" of the original item.

Ostensibly this appears to concern itself with font, and text size. However, there is a deeper layer of considerations that address page layout and any stylistic application used in the document to convey intellectual concepts. In this case, the new paragraph indentation is regarded as the aesthetic evidence of the intellectual concept of "a paragraph" and the underline font used to denote speaker and time from the spoken words.

The discussion can be summarised as follows:

- of the files in the set, none have an explicit font type specified in the file object;
- of the files in the set, none have an explicit text / font size specified in the file object;
- the reference version of WordStar has a default font applied to any new document;
- this is assumed[9] to be a common feature of any version of WordStar;
- the reference version of WordStar has a default text / font size applied to any new document;
- this is assumed to be a common feature of any version of WordStar;
- unless explicitly stated, the font type, and text size used is assumed to be the default set by WordStar;
- unless explicitly stated / advised, the default font is taken to be "Courier";
- unless explicitly stated / advised, the default text size is taken to be 10 points.

[7] We have previously noted that we do not believe the current definition of significant properties is sufficient. The definition states that they are properties "determined to be important to maintain through preservation actions" [1]. Our opinion is that "all technical properties … [are] important, irrespective of whether or not they should remain across an action. Some properties we may actually want to deliberately take action on to remove from the file. These properties are significant and must be tracked across actions" [2].

[8] We work from the generalised position that you cannot view content in an environment that exactly matches the original environment (other than perhaps the original itself). There are too many unknown and known variables that can never be replicated to support the perfect rebuild of a system in its original environment.

[9] The word "assume" will trigger alarm bells for preservation specialists. We **must** make assumptions when we have exhausted other possibilities or else we would not be able to complete this work. We are comfortable with making assumptions as long as they are noted, consistent and based on a degree of contextual knowledge that must be used in the absence of any evidence to the contrary.

The full discussion is conveyed in the original content review outputs.

4.4 Concern 2: Intellectual content

In this measure, we are interested to ensure that all intellectual content accurately travels from file A to the migrated file B. There should be no translation of information, concepts or semantic-laden parts of the original file – only direct, absolute migration of content.

In principle, this seems a simple premise. However, there was need for significant discussion to ensure that we collectively understood the significance of various parts of the document.

One of the most interesting and far reaching discussions was on the purpose of counting "lines".

In WordStar, in the text editor, line endings and carriage returns are automatically inserted where required by the software. Lines appear on screen to be essentially fluid (a change at the top of a paragraph propagates line changes where needed along every line in the paragraph, as fitting within page margins). However, inspection of the file shows that these line endings are hard written into each line (using the hexadecimal marker \x8d\x0a\xa0\x0a):

```
6f 76 65 72  20 44 65 63   20 2d 20 4a  61 6e 2e 20   over Dec - Jan.
77 65 20 68  61 76 65 20   74 68 61 74  20 73 6f 72   we have that sor
74 20 6f 66  20 8d 0a a0   a0 73 79 73  74 65 6d 20   t of ‖.  system
61 73 20 77  65 6c 6c 20   2d 20 73 65  65 20 68 6f   as well - see ho
77 20 69 74  20 67 6f 65   73 2e 20 42  75 74 20 49   w it goes. But I
20 2e 20 2e  20 2e 20 2e   49 20 73 75  70 70 6f 73    . . . .I suppos
65 20 8d 0a  a0 a0 74 68   61 74 20 74  68 65 20 77   e ‖.  that the w
```

Figure 8 - Example word wrapped line ending

Hard typed carriage returns / line feeds (i.e. application of the "enter" key inside a text document) are indicated differently in the file stream to these "soft" line returns, (and are as expected in standard text documents \x0d\x0a):

```
73 3f  7b 20   2e 20  2e 20   2e 20  2e 20  8d 0a  a0 a0   s?{ . . . . ‖.
2e 2e  7d 0d   0a a0  a0 a0   a0 a0  a0 a0  13 30  31 32   ..}..       .012
20 2d  2d 13   20 57  65 6c   6c 2c  20 59  65 61  68 2e    --. Well, Yeah.
```

Figure 9 - Example explicit line ending[10]

From analysis of the corpus, it is apparent that some files have very differing page margins. This can be seen by making a histogram of the line lengths found in the files as a set (Figure 9).

The double peak is particularly interesting. If all documents had the same line margin, this would be observed as a single peak, as is found in most of the individual file analysis (see Figure 10).

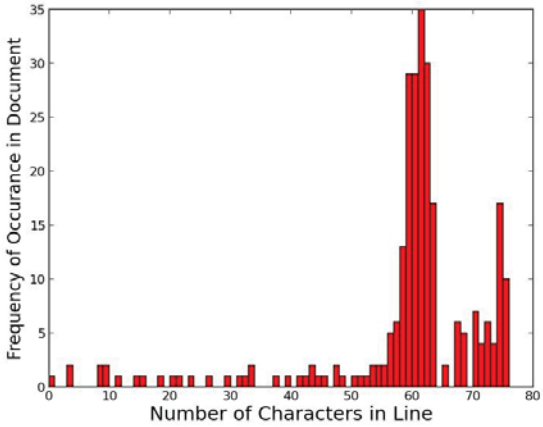


Figure 7 - Frequency of line length: All files in Corpus

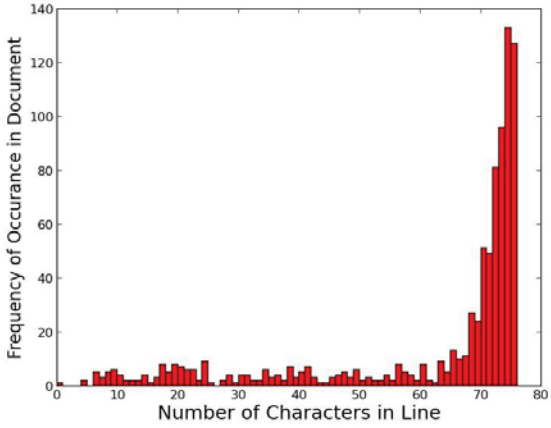


Figure 10 - Frequency of line length - File reference: #3

However, some files clearly show this double peak (Figure 11)

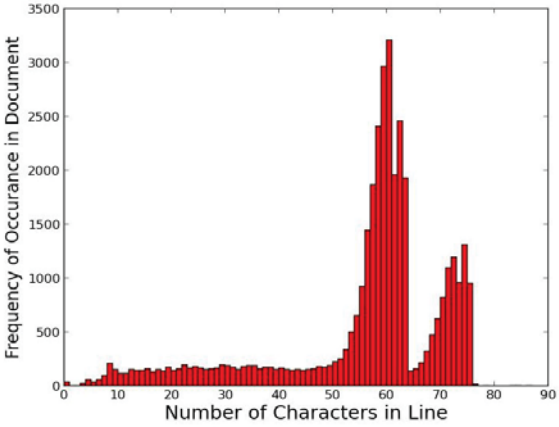


Figure 11 - Frequency of line length - File reference: #22

[10] Red text is a manual redaction of identifying names, places or initials that are found in the original document. This redaction method will be used through this paper.

In these files, it is notable that the line length varies in discrete "chunks" in the document, with no apparent explanation for the variation.

The ensuing conversation resolved that the concept of paragraph was primary, and there was no need to attempt to preserve the original line size, as this would likely impact negatively on the modern consumption of the intellectual object.

By deciding on this matter, the working group had essentially agreed that when the document was created, the original author had no explicit desire to reflect any meaning from the line length used. The length of a line was simply a by-product of the paragraph structure. In other cases, with other collections, this perhaps would not be a safe assumption and serves to reflect the importance of having curatorial involvement in the process.

This decision on the line-endings had an impact: the number of pages changes. We had to consider therefore if people had referenced these documents and how they did so. Did they (or would they in future) reference by page number? The decision was made that in this case, the movement of text across pages was allowable as accurate reference would be made through time-points noted in the text rather than page numbers. However, it was an impact that required some considerable attention.

Some other decisions made can be summarised. The use of underlined and bold fonts in the document was seen as of intellectual import and as such should be perfectly replicated in the migrated final. All standard text characters should be migrated with no change. Paragraph structure is essential to replicate accurately, original line length is not. The paragraph object and the "word" (an ordered group of printable characters) are considered the primary intellectual concerns to migrate and measure.

The second intellectual concept to consider is the conventions used by the author of the document to convey informational components. In this set, we are fortunate that all the documents come from the same source, and as such share a common set of conventions. These were noted as:-

- A new paragraph is indented on the page
- A speaker is denoted by their initials and these are underlined. E.g. JG
    - These are occasionally bolded. E.g. **JG**
- Time elapsed in the interview is recorded as an underlined number. E.g. 005
    - These are occasionally bolded. E.g. **005**
- On occasions these features are combined with at least one white space char as a separator, E.g. JG 005
    - These are occasionally bolded. E.g. **JG 005**
    - Order is not controlled, both JG 005 and 005 JG are found in the texts

This means that we have up to three intellectual concepts that are clearly identifiable in each paragraph; a speaker, a chronological marker, and the spoken words. These features were to be retained.

These pieces of information (the aesthetic and intellectual pieces) formed the backbone of proofs that were presented to the content-owner in the preservation plan.

5. WRITING THE CONVERTR

Having spent time with the content files and the content curator ensuring that the WordStar files were well understood (technologically, intellectually and aesthetically) the next step was to choose a target format, and construct the converter.

5.1 Picking a target format

Given that these files are known to be relatively simple text only files, but do contain some basic formatting, we could rule out some formats and rule in some others.

The master list of options included any valid variant of PDF, MS DOC (OLE2 based), MS DOCX, RTF, ODF text variant or HTML (v4 or v5).

We already have content in all these formats, creating more of any of these would be viable, however we had to choose one, and that decision was made against the following points:

PDF – High ranked candidate. Can be problematic if not properly constructed.

MS DOC – Not ideal, proprietary standard. We would need a specific encoding library to create valid doc files.

MS DOCX – As above, but slightly more preferred due to the availability of its specification.

RTF – Not ideal. Known to be problematic at times if not implemented well.[11] Not well suited in our current delivery environment.[12]

ODF – Not ideal. Not widely used / supported / found in collections.

HTML – High ranked candidate (if all formatting requirements are supported). Easy to use, easy to create, open standards. Relatively transparent.

HTML v4 was picked as target new format. The main justifications were:

- very common open standard;
- very well supported standard;
- found in significant volume in the collections;
- supports the formatting requirements;
- easy to wrangle into preferred shape;
- results in low complexity files.[13]

5.2 Writing and testing the converter

The target language for the converter was Python. This was a natural choice as it has native support for text manipulation.

This was one of the first Python projects ever completed by the code developer – and as such it should be noted that the code used is not always the most efficient / simple / pythonic implementation.[14]

When planning the build, the process was broken into some core tasks.

[11] NLNZ has previously undertaken remediation work on RTF files prior to ingest

[12] Ease of access is one criteria in our preservation planning process.

[13] The full discussion of the merits of the formats is contained in the preservation plan.

[14] Examples of the code are given in Appendix 1.

1. Take WordStar file
2. Slice file into paragraphs
3. Per paragraph
    a. Strip formatting, make text only version for comparison
    b. Convert WordStar markup into HTML markup
    c. Recombine into a document
4. Apply HTML structure
5. Save new file
6. Open new file
    a. Strip formatting, make text only version for comparison
7. Compare reference versions with each other
8. Write log
9. End

During the conversion a number of challenges arose. One was based on the fact that WordStar markup tags do not contain "start" or "stop" information. They are a simple binary switch, or a "toggle". For example, bold is either turned on, or turned off.

Conversely, HTML tags do contain "start" or "stop" information. `<b>` de-marks the start of bold text and `</b>` indicate the end of bold text.

This poses two interesting challenges.

- What happens when an author fails to close the bold tag?
- As we have introduced the concept of a paragraph as a structural object, and must comply with HTML rules for nesting elements, what happens if a tag pair:

  `(<b>Some arbitrary text</b>)`

  overruns a paragraph boundary?

In the migration code above, the parser looks for the bold tag `\x13` and if detected, it flips the `bold_marker` flag. If it was `False`, it becomes `True`, (and visa versa), and inserts the corresponding tag into the text. At the end of the paragraph the decision was made to force any open tags to close. This prevents the formatting from "leaking" into the rest of the document when it's not closed properly due to an author omission.

A secondary issue emerged once all the formatting tags were implemented. HTML is very deliberate about tag order. Tags are expected to open and close in order.

For example,

```
<b><u>Some arbitrary text</b></u>
```

would not be valid HTML,

```
<b><u>Some arbitrary text</u></b>
```

would be. Note the positioning of the closing tags. This meant that tag nesting and detection of invalid tag sets was required to ensure that valid HTML was generated.

5.2.1 Conversion Principle Development

As the conversion code was tested and iterated, a conversion principle was refined. Namely that the conversion code should only emulate the performance and behavior of the original software, and not address any formatting errors viewed to be editorial. This became a useful principle to lean on as the conversion code became more complete.

This principle was tested at length when considering how to handle multiple spaces in a document. Analysis of the document corpus showed the number of times the author used white space in a way that would be suppressed by HTML unless mitigated:

- Double space between two printable chars: 372
- Triple space between two printable chars: 113
- Between 2 and 50 spaces between two printable chars: 870
- A single space between a printable char and a full-stop: 3,992

As HTML is not a whitespace preserving format, whitespace ranges of longer than one character would need to be processed in such a way that the browser was forced to render each character, and not to conflate spans of whitespace into a single character.

This was achieved by converting whitespace spans longer than one character to a mixture of breaking and non-breaking white space characters. The non-breaking white space character ("`\xc2\xa0`") is always rendered by the browser or HTML parser, and so was used to ensure that white space characters were reproduced exactly as per the original.

This was particularly important where whitespace was used between formatting tags.

For example, the WordStar section

```
 ^U This is some arbitrary text ^U
```

would natively convert to HTML as:-

```
<p> <u> This is some arbitrary text </u><p>
```

which in turn would render as:-

"This is some arbitrary text"

The actual expected WordStar formatted text should render as

" This is some arbitrary text "

Note the leading and trailing whitespace.

It was therefore important that this was handled correctly to ensure that the formatting as was found in the original files was accurately moved into the HTML files.

5.2.2 Addressing the aesthetic concerns

As previously discussed, there was much analysis of the aesthetic construction of the files. This concerned the font choice, font size, line width, and the various page margins.

In the WordStar files, the font selection, and page margins are defined primarily in the default template. It is possible for an author to manually change these values. However, this would have left something of a footprint in the files and was not detected.

A decision was made to use an internal CSS declaration in the HTML documents to declare the font, font size and margins. The font was set to "courier" and the margins adjusted to ensure that the page layout follows the norms found in the original files.

This allowed the "speech" line starts to be indented as per the originals, and the wrapped lines to be pulled away from the edge of the HTML frame replicating the margin found in the original.

The CSS declaration used was:-

```
<STYLE TYPE="text/css">
```

```
        <!--
        BODY { margin: 1px 1px; text-indent:-2em;
font-family:courier; }
        p { text-indent:-2em; padding-left: 2em;
margin:4px 0px; }
        -->
</STYLE>
```

5.2.3 *Dealing with exceptions*

The main exception was a single line of formatting found in one document.

The line of interest was:

```
\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2
\xa0\xc2\xa0\xc2\xa0\x14\x05\x14But I mean,
when you say there are lots of
contradictions, there \xc2\x8d
```

Breaking this line down gives:

`\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2 \xa0\xc2\xa0\xc2\xa0` (The normal indentation)

`\x14\x05\x14` (The problem area)

`But I mean, when you say there are lots of contradictions, there` (The "text")

`\xc2\x8d` (The normal hard coded line wrap)

It is worth noting the context of the line. It comes from a longer piece of speech by one of the speakers and so is found "mid flow" and the problem bytes are unique to this line, in this file, in the corpus.

The byte `\x14` is used by WordStar to denote superscript text, (toggling on and off as per other formatting markers). The byte `\x05` is used by WordStar to support user generated codes to be sent to the printer directly.

This has some interesting connotations. Because the original install is not available for inspection and there are no supporting notes, it is impossible to know what code might have been used here. This would have been a printer specific instruction and set up in the installed instance of WordStar by the user. The version of WordStar these files were created on supports up to four of these user codes and so even if the printer and the possible codes that could have been used were known, the specific code bound to the byte is long forgotten.

The combination of bytes essentially says:

<toggle superscript on>

<send unknown user code to printer>

<toggle superscript off>

Given that any user code is unknown and its impact on the document is impossible to guess, it was decided to remove both these bytes ("toggle superscript", and "send user code to printer") from the document. The justification being that it was either an error by the author, or that its impact cannot be sensibly guessed. There is no known visible text inside the superscript tags and when printed on the reference build of WordStar, it had no affect on the printed text.

5.3 Building a text comparison tool

Having agreed on the aesthetic treatment of files, there was an outstanding question of how the intellectual accuracy could be demonstrated to the curator. The conversion code never actually "touches" normally encoded text characters; it simply moves them into the new HTML file. In the conversion process, a word by word check is made to ensure this is true and a log generated to record this fact.

It was undesirable to require the curator colleagues to read the conversion script and produced file to assure themselves that every word was there. A decision was made therefore to build a simple text comparison tool to allow the curator to inspect the new file, comparing it with the original.

The comparison tool was built in python and was designed to allow a reader to step through a file, paragraph by paragraph. The tool displays the original paragraph the new HTML paragraph, and a summary of any differences found in the use of alpha numeric characters, punctuation and whitespace.

The reader was able to toggle between a "cleaned" version of the text (with all formatting removed) and the native paragraph as found in each file. It displayed filenames, and paragraph numbers to allow any discrepancies to be recorded and later investigated.

Some basic navigation tools were included (such as "jump to paragraph number n") and key bindings to allow any file pair, and their associated text parts to be swiftly assessed.

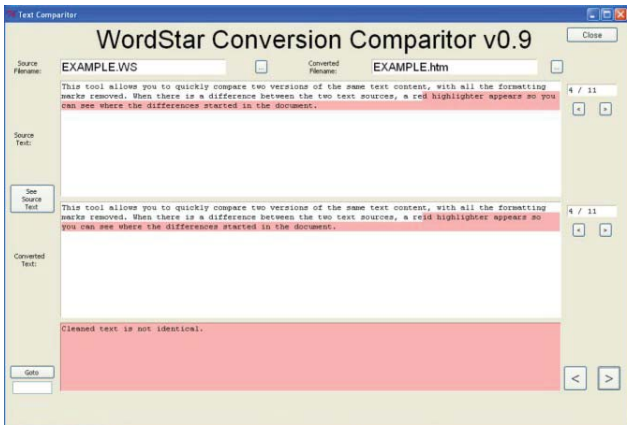


Figure 12- Screen shot of the comparison tool

This proved to be an invaluable tool in allowing the curator to demonstrate to their satisfaction that the files were accurately converted.

The tool allowed the curator to spend time with the content, at their own pace, assessing the original files in a meaningful way, and comparing the proposed conversions. They were able to see behind the relatively dense conversion code, and look at the raw information found in the source files being migrated.

6. LESSONS LEARNED

Because of the exploratory nature of the project, the end to end process took a long time to complete. Each step was very carefully considered by technical and curatorial staff alike, and it was deemed valuable to explore every question or concern in detail when it was encountered.

It would be unreasonable to attempt to calculate the amount of effort that went into completing this work, not least because one of the stated aims of the project was to give us the time and space

to explore the concept of migration, to develop key skills in this area, build tools, wrangle files and otherwise build a strong foundation to help support our broad program of work.

The first lesson was one of comfort. Whilst the number of files in the corpus being converted was low, the methodical and thorough nature of the assessments, as presented in the preservation plan, resulted in a strong comfort that the conversion was accurate, thus satisfying the curator. Further to this point, the way our preservation system is designed means that the original WordStar files are never actually replaced by the HTML files in the system. They are superseded in the versioning model used to describe the intellectual entity. This means that if better tools were developed for this migration, it would be a trivial exercise to return to the WordStar originals and make new converted versions directly from the original content.

Managing expectations was critical in creating the environment for all actors to be comfortable with the results. We are not trying to recreate absolutely the original, but rather create a version of the original content that can stand in the original's stead and allow use and reuse of that content.

The second lesson was that there should be no assumptions about the context and knowledge that colleagues bring to the process. During the work a surprising paradigm shift was made by project workers. In the early exploration of conversion tools, any suggestion that converted artifacts might be further processed beyond what the tool had already done to produce more accurate results was met with stern a stern "no". That "no" described an unfamiliarity with methods of conversion and the separation of content from medium. In the early stages of the project, before we created our own tools, a suggestion was mooted that a line could be added to the HMTL files (once the conversion utility had finished with them) that would lock the font as courier, rather than allowing the browser to choose the font. This was met with some resistance. The main argument used was that the HTML would have to be changed, resulting in the HTML created by the commercial conversion tool being somehow "disrupted".

The counter argument was that this should not be an issue. The commercial conversion tool was effectively a black box, and there was no deep knowledge of what was happening inside the tool to create the converted files. To that end, it should not be problematic to make some known changes to the files (the adding of the defined font) when the rest of the processing used by the tool was unknown. Of course, this argument became irrelevant once the decision was made to build a conversion tool from scratch, however at the time it was an interesting point to explore.

The third lesson was figuring out how to succinctly demonstrate technical processes to non technical colleagues. Some of the decision making was very technical but required the support and validation of non technical colleagues. We used the original content review as a method of framing these discussions and deliberately took the time to ensure that a full understanding was achieved. In the future, and with other curators, it may not be necessary to take them on the entire technical journey. In this case though, we felt the time and effort taken to build a good relationship with the curator was important to ensure acceptance of what we were proposing. It also had the added advantage of tightening our own understanding of the processes and attitudes towards them.

By working to the agreed principles and making simple tools that allowed technical processes to be easily demonstrated, it was possible to put the right level of detail in the hands of decision makers to enable them to understand what was happening at all times during the project. Ultimately, education on both sides of the technical divide took place across the entire process.

7. CONCLUSIONS AND NEXT STEPS

At the culmination of this project, we are satisfied that we achieved the two aims we set out with.

The first aim was to explore the migration process for the Library, and get a sense of the complexity we face when attempting to move content from one format to another. This process needed to be robust, thorough, and transparent.

We noted that it took far longer than we would normally expect, and we are happy that the time taken we needed to ensure that all involved parties had the time and space to understand what we needed to do, and could contribute to the process in a meaningful way. We have identified some areas that can be significantly expedited and remain confident that the next iteration of this process would take less than half the effort we expended on this project.

Secondly we are confident that we have successfully moved the WordStar content in to the HTML format in an accurate and transparent way.

7.1 Next Steps

As a direct outcome of this project we will start to process our WordStar2000 content in a similar way. This content is related to the corpus addressed in this project, but different in its technical composition.

The learnings and tools from this project will be leveraged to deliver this next migration.

Given the very bespoke nature of the resulting conversion code, we do not plan to release the code as a supported application, or as "abandonware". The risk that it is used on content without the appropriate amount of technical / curatorial assessment is a liability we do not wish to hold. However, the code can be requested from the Library / paper authors, who would be happy to oblige.

8. ACKNOWLEDGMENTS

Our thanks to Linda Evans (Curator Oral History and Sound) Leigh Rosin (Digital Archivist) and Jessica Moran (Assistant Digital Archivist) for working with us through the process of migrating these files.

9. References

[1] PREMIS Editorial Committee, PREMIS Data Dictionary for Preservation Metadata, version 2.1, (January 2011), pg. 39.

[2] Jailani, H., McKinney, P. 2012. 'Compliance Conundrums: Implementing PREMIS at two National Libraries', *Proceedings of IS&T Archiving 2012*, Copenhagen.

[3] The National Archives, *PRONOM Technical registry*, 31st March 2014. http://www.nationalarchives.gov.uk/PRONOM/.

[4] DeVorsey, K., McKinney P., 2010. 'Digital Preservation in Capable Hands: Taking Control of Risk Assessment at the National Library of New Zealand', *Information Standards Quarterly,* Spring 2010, 22:2, pp 41-44. http://www.niso.org/apps/group_public/download.php/4242/IP_DeVorsey_McKinney__Risk_Assessment_isqv22no2.pdf.

[5] MicroPro International Corporation, 1981. *WordStar Reference Manual.*
[6] The Internet Engineering Taskforce, 2003. *RFC 3629. UTF-8, a transformation format of ISO 10646,* http://www.ietf.org/rfc/rfc3629.txt.

10. Appendix 1: Examples of code

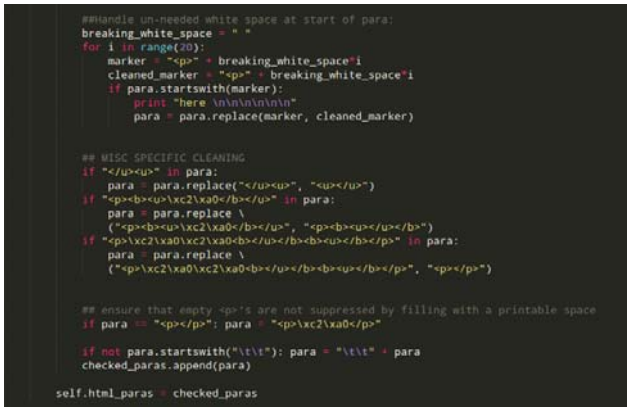
```
def make_paras(master):
    """converts the text to hex,
    looks for the forced line endings (not user inserted)
    and forms a list of paragraph blocks"""
    paras = []
    linePartial=''
    for line in master:
        line2 = binascii.hexlify(line)

        ### using the automatically inserted WordStar line ending
        ### (\x0d0a) for run on lines we can find the paragraph blocks
        if linePartial!='':
            line2 = linePartial+line2
        if '0d0a' in line2:
            paras.append(line2)
            linePartial=''
        else:
            linePartial = line2
    return paras
```

Figure 13 – Split text into paragraphs

```
def para_proc_clean(para):
    """takes the hex decoded file returns the cleaned para """
    para_bytes = re.findall('..', para)
    new_para_bytes = para_bytes
    for i, char in enumerate(para_bytes):
        if char == '0b': # handles the "physical" line end marker
            new_para_bytes[i] = '20'
        if char == '02': # handles the bold marker
            new_para_bytes[i] = ''
        if char == '13': # handles the underline markup
            new_para_bytes[i] = ''
        if char == '1a': # handles the eod of doc char
            new_para_bytes[i] = ''
        para = ''.join( new_para_bytes)
    para = para.replace("8d0a","")  # handles the line boundary
    para = dehexlify_text(para)
    return para
```

Figure 14 - Making plain text (no formatting)

```
def make_html_paras(self):
    temp_chars = "".join(self.source_paras)

    # deal with the structoral parts of the byte encodings
    # odd/even new page marker
    temp_chars = temp_chars.replace("\x0b", "\n")
    # makes valid / websafe non-breaking space
    temp_chars = temp_chars.replace("\xa0","\xc2\xa0")
     #removes the soft line-endings and new line gutter
    temp_chars = temp_chars.replace("\x8d\n\xc2\xa0\xc2\xa0","")
    # removes the not needed soft line ending
    temp_chars = temp_chars.replace("\x0a", "")
    # removes the unused user print code insert
    temp_chars = temp_chars.replace("\x05", "")
    # removes the unused superscript toggle
    temp_chars = temp_chars.replace("\x14", "")


    # tops and tails paragraph with html <p> </p> tags
    temp_chars = temp_chars.replace \
    ("\r\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0\xc2\xa0", "\n<p>")
    temp_chars = temp_chars.replace("\n", "</p>\n")
    # needed because not all lines have \r marker
    temp_chars = temp_chars.replace("\r", "\n").replace("\n\n", "\n")
    # deals with the native markup in a brute force way
    underline_flag = False
    bold_flag = False
    new_chars = []
    for char in temp_chars:
        if char == "\x13" and underline_flag:
            underline_flag = False
            char = "</u>"
        elif char == "\x13" and not underline_flag:
            underline_flag = True
            char = "<u>"
        if char == "\x02" and bold_flag:
            bold_flag = False
            char = "</b>"
        elif char == "\x02" and not bold_flag:
            bold_flag = True
            char = "<b>"
        if char == "\x8d":
            char = ""
        # finish working on document as a full block
        new_chars.append(char)

    new_chars = "".join(new_chars)


    # protect double spaces with markup
    new_chars = new_chars.replace(" </u> ", "\xc2\xa0</u>\xc2\xa0")

    # work on paras as a block
    checked_paras = []
    self.html_paras = new_chars.split("\n")

    for para in self.html_paras:

        if para == "":
            para = "<p>\xc2\xa0</p>"
        if not para.startswith("<p>") and para != "":
            para = "<p>" + para
        if not para.endswith("</p>"):
            para = para  + "</p>"
        if para == "<p><u></u></p>":
            para = para.replace("<p><u></u></p>", "<p>\xc2\xa0<u></u></p>")

        # handle unclosed tag marker
        marker_tokens = [["<u>", "</u>"], ["<b>", "</b>"], ["<i>", "</i>"]]

        for markers in marker_tokens:
            marker_open, marker_close = markers
            number_of_opens = para.count(marker_open)
            number_of_closes = para.count(marker_close)
            if number_of_opens > number_of_closes:
                para = para.replace("</p>", marker_close+"</p>")
            if number_of_opens < number_of_closes:
                para = para.replace("<p>", "<p>"+marker_open)


        ##Handle empty underline sections:
        non_breaking_white_space = "\xc2\xa0"
        for i in range(20):
            marker = "<u>"+ non_breaking_white_space*i + "</u>"
            cleaned_marker = non_breaking_white_space*i + "<u></u>"
            if marker in para:
                para = para.replace(marker, cleaned_marker)

        ##Handle un-needed white space at start of para:
        non_breaking_white_space = "\xc2\xa0"
        for i in range(20):
            marker = "<p>" + non_breaking_white_space*i
            cleaned_marker = "<p>" + non_breaking_white_space*i
            if para.startswith(marker):

                para = para.replace(marker, cleaned_marker)
```



```
        ##Handle un-needed white space at start of para:
        breaking_white_space = " "
        for i in range(20):
            marker = "<p>" + breaking_white_space*i
            cleaned_marker = "<p>" + breaking_white_space*i
            if para.startswith(marker):
                print "here \n\n\n\n\n\n"
                para = para.replace(marker, cleaned_marker)


        ## WISC SPECIFIC CLEANING
        if "</u><u>" in para:
            para = para.replace("</u><u>", "<u></u>")
        if "<p><b><u>\xc2\xa0</b></u>" in para:
            para = para.replace \
            ("<p><b><u>\xc2\xa0</b></u>", "<p><b><u></u></b>")
        if "<p>\xc2\xa0\xc2\xa0<b></u></b><b><u></b></p>" in para:
            para = para.replace \
            ("<p>\xc2\xa0\xc2\xa0<b></u></b><b><u></b></p>", "<p></p>")


        ## ensure that empty <p>'s are not suppressed by filling with a printable space
        if para == "<p></p>": para = "<p>\xc2\xa0</p>"

        if not para.startswith("\t\t"): para = "\t\t" + para
        checked_paras.append(para)

    self.html_paras = checked_paras
```

Figure 15 - Making HTML4 text