

WORDS



A Quarterly Bulletin for Technical Writers & Communicators

Volume 2 | Issue 3 | August 2010

The beginning

- In this issue we cover two tasks that take up most of a technical writer's time: writing procedures and wrangling with legacy documents. Procedure-writing is our bread-and-butter, and just as much thought is needed in structuring a procedure as in writing the procedure if our customary audience-centric ethic is to suffuse our work. And, as Darko Tausan points out in his article on translation, even more thought is needed if you are writing for localisation.
- Many thanks are owed to Bruce Ashley for opening up the toolkit he uses in updating legacy documents, no doubt one of the more tedious tasks of our profession. Bruce has even provided some sample files for you to experiment with.
- Bruce's tutorial files—along with the supporting file that accompanies Dave Reynolds's new article on customising Adobe FrameMaker—give an opportunity to demonstrate a useful, but rarely used, feature of Adobe Acrobat: the ability to add file attachments to PDFs. This issue explains how you do it and gives some potentially useful scenarios.
- If there is one constant throughout the long history of English, it has to be ... change. Natural, organic change. Practices come; practices go. Many language-lovers lament the death of some practices, especially if it deprives us of a distinction or a useful shorthand. But that's no reason to despair. English-speakers have been an inventive lot. Take, for example, punctuation. Every punctuation mark now in common use came into use at some time—not by divine intervention or decree, but by the gradual accumulation of a critical mass of use. As we lose punctuation marks (such as the en dash) or use them for so many purposes that ambiguity results (as with the solidus), what is to stop us inventing new marks that might halt any diminution of communicative power? To that end, we have a competition in this issue: to come up with new punctuation marks that eliminate ambiguity or return lost communicative power. See page 19 for details. The proposal judged best by the *Words* team wins the proposer a copy of Kate Burridge's new book (reviewed on page 17 of this issue). Happy inventing.

Geoffrey Marnell

Editor [geoffrey@abelard.com.au]

Contents

Writing for translation.....	1
Packaging files with Acrobat	3
Converting legacy MS Word documents.....	5
Custom toolbars in FrameMaker	8
Pitfalls in procedure writing.....	12
Journals.....	16
Book review.....	17
Tips and tricks.....	18
Miscellany (and Punctuation Contest)	19
Mindstretchers	20

Writing for translation

Darko Tausan

The style of technical writing suitable for translation is just one of the issues you need to consider in producing technical documentation for international markets. You should never assume that a good translation *on its own* will be enough, regardless of how well written the source document is.

Translation, adaptation or both?

Before one looks at the translation component for your documentation there are a number of tests and checks that need to be done. You need to check if there are any product-quality standards or legal requirements that need to be followed if your document is to be accepted in the target country. For example, the English version of your warranty in a simple translation may not be adequate. It may need to be reviewed and adapted by legal translators or lawyers in the target country. Different countries might have different service contracts and requirements for repairs.

Similarly, some countries insist that their particular system of weights and measures be used in any documentation imported into their country. What all this means is that the source documentation may well require *adaptation* as well as *translation* for a specific region.

Preparation

There are also production-specific considerations that require planning at the outset. Such planning reduces the chance of missed deadlines and cost blowouts. For example:

- There should be testing to ensure compatibility between the tools used by translators and the technical authoring tools used to generate the material to be translated.
- There should be agreement on the output method for layout, particularly if several languages are being produced.
- Allowances may need to be made for right to left script (Hebrew and Arabic, for example) and for the various idiosyncrasies in non-Roman alphabets.

Writing style

Writing effective documentation is critical to the penetration of brands and services in the global market. Creation of clear, concise English language content, without cultural and local jargon, creates a common international platform that will maximise the effectiveness of the translated material and reduce costs and production timeframes.

Translation mistakes are sometimes due to the source text being ambiguous. Translators use their common sense in trying to understand the source

text, but it is not uncommon for their inexpert knowledge of the product or subject to lead them to the wrong or unintended meaning.

Here are some tips for authors writing for a non-English audience, tips that will make it easier for translators to get across your intended meaning:

- Keep sentences short and order the parts of the sentence logically.
- Be clear when using *and* and *or*.
Does A + B or C mean A + (B or C) or (A + B) or C?
- Use articles for clarity.

In the absence of a clear context, articles help in understanding the construction of a sentence.

Instead of *empty recycle bin*, write *empty the recycle bin* or *the empty recycle bin* as appropriate.

- Use appropriate punctuation.

The omission of a comma can completely change the meaning of a sentence. For example, compare *The file data and executable are located in the Source folder* with *The file, data and executable are located in the Source folder*.

- Avoid line breaks or discretionary hyphens, as they might confuse the translator and break the translated text in inadequate places.
- Be careful when using abbreviations or acronyms. If they are not commonly used, make sure you give the full expression, usually between parentheses on the first occurrence. Remember that an acronym can stand for several expressions, depending on the context.
- Use puns and colloquial expressions with care.
There might be no equivalent in the target language. It is important to provide guidance if the translator needs to adapt the source text to find an alternative.
- Be aware of cultural differences in communication.

The tone and style used in commercial communication in English might be different in another language (Arabic or Japanese, for example). It is not the translator's responsibility to decide what will be appealing to the target audience. It is the writer's responsibility to take cultural difference into account.

Darko Tausan

Darko Tausan is the managing director of Commercial Translation Centre (CTC). CTC has been assisting Australian companies since 1987 in producing international documentation for the global delivery of branded services and products. CTC now has offices in Melbourne, Sydney, Perth, Brisbane, Canberra, Adelaide, Galway, Dublin, London and Auckland.



commercial
translation
centre

For Effective
Translation/Localisation
Of Technical Documentation
In 80 Languages

Australia Wide Service Since 1988
Call 1800 655 224 (Australia Toll Free)
Email: mail@ctc.com.au
Web: www.ctc4.com

commercial translation centre

Packaging files with Acrobat

Geoffrey Marnell

The Adobe Acrobat of yesteryear would to many contemporary technical writers be almost unrecognisable. It was a product that let you create a PDF and not do much with it. (Indeed, the fact that you couldn't do much with it was considered a virtue. You could send a file to someone, even an outside printer, and you knew that they couldn't change it. Nowadays, not only can you change the text in a PDF (with the **Text Touchup** tool), you can add text to a PDF (with the **Typewriter** tool) and add it anywhere. You can even block out text that you don't want readers to see (with the **Redaction** tool) and delete objects (with the **Object Touchup** tool).

Recent versions of Acrobat are packed to the rafters with features, but few users seem to exploit many of them. For instance, few use Acrobat Forms and yet it is a very simple way of conducting a survey or just eliciting data: from readers known or unknown. No need for an HTML editor, a web hosting company and server-side software (with its typically incomprehensible configuration instructions). Just design your form in Acrobat (or any application from which a PDF can be made), add some fields, add a click-here-to-send button, specify your email address and distribute the form (which is just a PDF document with fields on it, one readable even by the free Adobe Reader). There's even a back-end database, created on your computer, where responses can be automatically stored. A wonderful feature; sadly under-used.

Another very useful but rarely used feature is the Packager (now called *Portfolio* in version 9). The Packager lets you attach files to a PDF. The files don't even have to be PDF files. And there lies its special usefulness.

Suppose your company is about to release a minor update to a software product. The common scenario is that the new version is delivered in a zip file (or the like) and inside that zip file is a `readme.txt` file, an installer and possibly other files. Traditionally, the `readme.txt` file explains how to install the update and lists the main bugs that have been fixed since the last version. It might also describe some new features, if there are any.

But the `readme.txt` file is one of the least read documents that accompany software products. Most of us immediately launch the `setup.exe` and get on with it. And sometimes we come to regret our haste, especially if the `readme.txt` file contains important preparatory instructions (such as to uninstall the previous version before installing the new version). If only such important information could be put right under the nose of the updater.

Well that's exactly what the Acrobat Packager (aka Portfolio) lets you do. Updaters receive or download what appears to be just one PDF file, but there are files attached to it. They must open the PDF file before they can access the attached files (the new installer, new .dll files and so on). And right there, at the top of the first page of the PDF, is the ideal spot to place important messages: **WARNING: You must uninstall all previous versions of Newfangle before installing this version.** Or whatever.

You could also provide installation instructions, a bugs-fixed list and a new-features list, thus doing away with the need for a `readme.txt` document. (These days, the free Adobe Reader is as ubiquitous as Notepad.)

In Adobe Acrobat 9 Pro Extended, the Packager has been replaced, or so it might seem, by the Portfolio. Portfolio sits on the menus in the same place as Packager did in earlier versions: select **File > Combine > Assemble PDF Portfolio**. (You can also click **Create** or **Combine** on the **Task** toolbar and then select **Assemble PDF Portfolio**.) And, like the Packager, Portfolio enables you to combine numerous disparate files.

But there the similarities end. Portfolio, with its templates and skins, is more about presenting, in an

Effective Onscreen Editing: new tools for an old profession

Editors are increasingly being asked to edit on the screen using a word processor, but most are finding it challenging to transfer their skills to editing with a word processor. *Effective Onscreen Editing* teaches the basics you need to learn to make the transition, plus proven tips and tricks to maximise your productivity and effectiveness. The book describes general principles valid for any software, then illustrates the principles using Microsoft Word to make them more concrete.

Available as a printed book from Lulu.com (15% discount until August 15: use the coupon code BEACHREAD305 when ordering). Also available as an eBook optimised for onscreen reading.

Learn more at the book's Web page:
<http://www.geoff-hart.com/books/oe/onscreen-book.htm>

 Diskeuasis
Publishing

attractive light, a body of work (such as a portfolio of your artwork or photographs). You can add text and images to a welcome page and a header page, but you are somewhat limited to how much, and what, you can put on these pages.

Still, the Packager functionality, if not the feature itself, is still available in Acrobat 9. You simply package files by adding them as *attachments*, functionality that was also available in Acrobat 8. (Perhaps it was this duplication that prompted Adobe to drop the Packager from Acrobat 9.)

This functionality can be used not just to place important set-up and configuration information in front of users: new versions of software, or a suite of patches that need to be installed in a particular order. A company could use this technique to advertise training courses they offer in the software attached to the PDF. Or they might send out (or make available from their website) marketing collateral as attachments to a marketing-oriented PDF. Indeed, the technique is useful wherever there is value in getting words in front of people before they start tinkering with software.

We have put this feature to use in this issue of *Words*. The articles by Bruce Ashley (see page 5) and Dave Reynolds (see page 8) are accompanied by attachments that help you understand and step through the various processes the authors are explaining. So here is yet another use for PDF attachments: as a way of providing supplementary learning aids to online instructional material.

So how does it work?

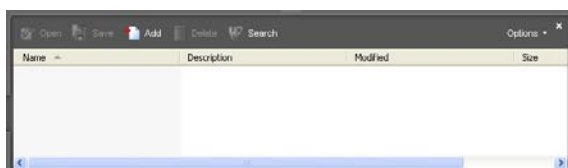
Adding a PDF attachment

When you open a PDF—in Adobe Acrobat or Adobe Reader—there will, by default, be a set of navigation panel buttons down the left side of the window. (If not, there will be a grey bar at the left. Right-click on the grey bar and select **Show Navigation Panel Buttons** from the context menu.)

1. Near the bottom of the grey bar will be the icon of a paper clip (shown at the right). Click on the icon to open the attachments panel (shown below).



If the icon is not visible, select **View > Navigation Panels > Attachments**.



2. Click the **Add** button on the toolbar of the attachments panel. The **Add Attachment** window appears. (This window is much like the

one that appears when you attach a file to an email.)

3. Select the file you want to attach to the PDF. It can be a file of any type, not just a PDF file. When you select a file, it is listed on the attachments panel.

You can also **SHIFT-CLICK** and **CTRL-CLICK** to select more than one file at a time.

4. Repeat from step 2 for each file you want to attach to the PDF.
5. When you have finished, click the close button on the attachment panel.

You can see the files you have attached by clicking the paper clip icon again.

Opening and saving attachments

When you receive a PDF that has an attachment, there is, alas, no indication in the PDF itself that it has an attachment. You need to find this out for yourself.

1. Open the attachments panel, as explained at the start of the previous section.
2. To open a file, select it and then click the **Open** button on the toolbar of the attachments panel.
3. To save a file, select it (and any others you want to save to the same location), click the **Save** button on the toolbar of the attachments panel and then select or specify a location.

There are five files attached to the PDF you are currently reading:

Name	Description	Modified	Size
ConversionDoc.doc		25/06/2010 8:04:00 PM	89 KB
FileA.txt		16/06/2010 8:08:00 PM	2 KB
FileA.doc		16/06/2010 8:08:00 PM	37 KB
FileB.rtf		16/06/2010 8:08:00 PM	30 KB
Custom toolbars in FrameMaker.txt		25/06/2010 8:04:24 PM	10 KB

If you are going to follow Bruce Ashley's Microsoft Word tutorial (see page 5), you will need to save the first four files listed above to the one folder. If you are going to follow Dave Reynolds's Adobe FrameMaker tutorial, you might benefit from saving, or at least opening, the final item in the list above.

Geoffrey Marnell

Contributions welcome

Everyone is welcome to contribute to *Words*. You can send us a letter, review or article. All contributions relevant to technical communication, or language in general, will be considered (and copyright is retained by the author).

Send contributions to: words@abelard.com.au

I ain't Normal no more! Converting legacy MS Word documents

Bruce Ashley

Open Microsoft Word and go to any directory with .doc files in it. Open one at random and move your cursor through it one line at a time. As you do, look at the paragraph styles in use. (You can see these in the **Styles** drop-down list on the **Formatting** toolbar.) Odds are, you will see paragraph after paragraph of only one style. This will be **Normal** style, with an occasional override for headings and the like (such as **Normal + 16 pt, Bold, Left**).

Even character formatting falls into the same pattern. Run your cursor over an emphasised word and the style shown will, in all likelihood, be **Normal + Italic**.

Try a few more files. I'm sure you'll find it's much the same. The non-technical writers around you simply won't understand the tears this has brought to your eyes and will no doubt mock you. They just don't understand the curse of the ubiquitous **Normal** style—Microsoft Word's default style—with all the manual formatting that it flags.

Traditionally, this hasn't caused technical writers too many problems, as most Microsoft Word documents never come before them. Typically these are the product of the amateur or accidental writer—the marketing folk, the business analysts, the engineers—and are rarely destined for the eyes of the end-user: our traditional readership.

But from time to time technical writers do need to work on documents born without the midwifery of a template or style palette. This is typically the case when organisations need to standardise a suite of documents to meet some contractual obligation, or to comply with ITIL, ISO 9000:2000 or some other standard.

I can hear some of you shudder at the thought of going through dozens, hundreds or even thousands of documents in an attempt to bring them into accord with some standard or other. (My own record is just over 66 000 documents in a single document repository.)

But it's not really that difficult.

Typically it's only the .doc, .rtf and .txt files you need to worry about and your first step is to get the repository owners to cull as much obsolete or unwanted documentation as possible. (My 66 000 documents were culled to 6 500 documents just by doing this.)

You then need to create a template with a standard set of styles and apply those styles to the remaining documents. This is where many have problems and where most standardisation attempts

get abandoned. The reason? Simply creating and then adding a new template will not fix the problem. You can only attach the template to another document. This in itself won't change the document. It won't, for instance, change a **Heading 1** style styled one way to a new **Heading 1** style styled another way. And it won't change **Normal + 16pt, Bold** to a **Title** style. You have to do that yourself. Now although this is not a major problem with a small document, it is with a 400-page document or with a repository of hundreds or even thousands of documents.

So if doing it manually is unworkable, how can it be done automatically? This is what we will look at now.

Setting up the conversion

I will illustrate this with a basic template (called `ConversionDoc.doc`) and three small documents—a .doc file, a .rtf file and a .txt file—and I'll use some basic styles that I see every day.

There is currently¹ no tool, plug-in or application to do style conversions for you, so you need a product such as *Visual Basic for Applications* (VBA) which will allow you to create a solution using a set of instructions known as a *macro*. VBA is shipped with the full version of all Microsoft Office products and thus is a convenient tool. Some light versions might not have it. To check if you do, open Microsoft Word and press **ALT + F11**. If the **VBA Project Window** appears, you have VBA.

I've attached to this PDF the template file, `ConversionDoc.doc` (the .doc instead of

If you are unsure how to open PDF attachments, see "Packaging files with Acrobat" on page 3.

.dot is deliberate) and the three files that need to be converted: `FileA.doc`, `FileB.rtf` and `FileK.txt`. Save all four files to a new folder. All four files must be in the same folder, and the folder should have no other files in it.

Open the three files needing conversion and have a look. Note that I haven't included bullets, lists, tables, colours, indenting, or the hundred or so other possible style attributes. This would just complicate things for now.

In File A you have four styles in use: **Title** (16pt Arial + bold), **Heading1** (14pt Arial + bold), **Normal + 12pt Arial + Bold**, and **Normal** (11pt Times Roman).

1. There are applications that aid in the conversion from Word to PDF, Word to FrameMaker, WordPerfect to Word, Word to XML and so on, but nothing that I have come across helps in the conversion of **Normal [+ override]** to pre-defined styles in Word.

In File B there are no defined styles. The author just made the text bold and increased the font size manually when a heading was needed.

In File K, the author wrapped some capitalised text in asterisks (**) to indicate the title, placed a string of equals signs (===) on the line under text to indicate a Heading 1, and placed a string of hyphens (---) on the line under text to indicate a Heading 2.

Before we begin, I want to emphasise that there are dozens of solutions to this task. I am not going to try to teach you VBA so that you can tackle the task in other ways. I also won't be explaining how to use the VBA editor. I will look at just one basic type of code that will help do the conversion.

If you want to learn more about VBA, start by using the macro recorder, open up the editor and look at what the macro recorded. Then buy a good book on the subject and practise, practise, practise.

Having said all that, let's begin.

There are three phases to writing any code: planning, writing and testing. I will step through the planning tasks to explain the logic behind the VBA code I have written. Having written the code, you don't need to do that task. You can modify or add to the code, in which case it would be a good idea to test that your code does what you want it to do.

Planning

In the approach I've taken, I want the code firstly to copy the contents of each file it encounters into the template. The conversion process will then take place on a copy of the file, leaving the original unchanged.

Next I want the code to identify the type of document it has encountered. This is because each type has its own formatting issues and I don't want the code to waste time looking for styles likely to be found only in other document types.



© Bruce Ashley 2010

So, how do I determine the document type? Simply by looking at the extension of the file. The code then jumps to a sub-procedure based on the extension found. The following shows the logic:

Copy the contents of the file to the template.

If the filename extension is .doc go to the doc sub-procedure.

If the filename extension is .rtf go to the rtf sub-procedure.

If the filename extension is .txt go to the txt sub-procedure.

Finally, create a document with the changes in it and save it as a Word document with a -1 prefix.

The sub-procedures work like this:

doc sub-procedure:

1. Go to the start of the document.
2. Identify the paragraph style. If you find a **Title** or **Heading 1** style, update it to your template's **Title** style or **Heading 1** styles. If you find a **Normal** style with, say, bold and 14pt, change it to **Heading 2** (or whatever). If you find **Normal**, no bold and 11pt, change the **Normal** style to your template's **Body** style.
3. Select the next paragraph and repeat step 2, continuing until the end of the document is reached.
4. Go back to the main procedure.

.rtf sub-procedure:

1. Go to the start of the document.
2. Identify the paragraph style, size and font style. If you find a **Normal** style that is greater than 14pt set as bold, this will be either a **Heading 1** or the **Title**. Make your call and tag it as the required heading. Do likewise with any other **Normal** + override styles.
3. Select the next paragraph and repeat step 2, continuing until the end of the document is reached.
4. Go back to the main procedure.

.txt sub-procedure:

1. Go to the start of the document.
2. Look for *****. If found, this is probably enclosing the title and so needs the **Title** style applied. Look at the following line to see if there are any === or --- as these will indicate the line above needs the **Heading 1** or **Heading 2** style applied.
3. Select the next paragraph and repeat step 2, continuing until the end of the document is reached.
4. Go back to the main procedure.

Looks simple, doesn't it.

The real challenge is in assessing the variations and the exceptions to each rule. The more complex a document the more code; the more exceptions the more code. The more user-friendly you want to make it, the more code.

Writing

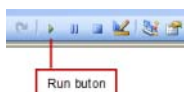
For this tutorial, there is nothing for you to write. The code has been written for you. You may want to change it or add to it, and I've added plenty of comments in the code to help you see what the various parts of it are doing.

Depending on your security setting, you might need to give permission for the VBA code to run when you open the template. Word uses the term *macro* for VBA code, so you need to allow macros to run. To do this, click on the **Enable Macros** button when asked.

Open the template file (`ConversionDoc.doc`) and press `ALT + F11`. This opens the **Microsoft Visual Basic** window where you can see the code.

Testing

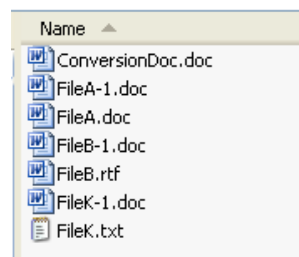
To run the macro, place your cursor anywhere within the code and press the play button on the toolbar. (The button's tooltip reads **Run Sub/User form**.)



Another way to run the conversion, and to see the files changing as they are being converted, is to:

1. Select **Tools > Macro > Macros**.
2. On the **Macros** window, double-click **mainMacro**.

The result of the conversion is a new set of case-study files, each with a -1 suffix: `FileA-1.doc`, `FileB-1.doc` and `FileK-1.doc`. The original files have been left unchanged so that you can compare the original file with the converted file. Note too that each converted file has been saved as a `.doc` file.



As with any code, you write it and then test it out. You need to ensure that it will run from start to finish and do what is intended. You need to cater for as many exceptions as you can think of. This is where the real test starts because every time you come across an exception, you will need to write more code to avoid it.

I recommend you add your own styles to the code and see the result. The more you play, the better you get.

So good luck with your conversion. I hope I have given you some food for thought.

Bruce Ashley

Bruce hails from Melbourne. He started life as a technical author while training finance staff in the late 1980s. After retraining in IT in the early 1990s, Bruce became an electronics software trainer. Since 1997 Bruce has been a dedicated IT technical writer, working as a consultant, contractor, senior technical writer and team lead.

Bruce has served two terms on the ASTC (Vic.) committee and served as the society's webmaster during that time.

You can contact Bruce at moo-man@optusnet.com.au.

FileK.txt	FileK-1.doc
***** FILEK *****	FILEK
THE FIRST HEADING 1	1 THE FIRST HEADING 1
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
THE FIRST HEADING 2	1.1 THE FIRST HEADING 2
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
The quick brown fox jumps over the lazy dog.	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.
	1.2 THE SECOND HEADING 2
	The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog. The quick brown fox jumps over the lazy dog.

Custom toolbars in Adobe FrameMaker

Dave Reynolds

Toolbars typically provide buttons that you can use to carry out operations without going through menus and submenus. The two main toolbars in Adobe FrameMaker—available on the **View** menu—are known as the **QuickAccess** bar and the **Formatting** bar. In addition to a set of persistent buttons (at the left of the bar), the **QuickAccess** bar has four variable sets of buttons at the right of the bar. You can step through each of the variable sets by pressing the next and previous arrows on the bar itself (see figure 1). These four sets of buttons provide commands for dealing with text, graphics (two sets) and tables.

I'm a great believer in using the mouse as little as possible when working at the computer. I do this for two main reasons: to reduce the chances of getting Repetitive Strain Injury (RSI) or Occupational Overuse Syndrome (OOS) through using the mouse, and because it speeds up operations that would otherwise require navigating through menus or even layers of menus. Having to frequently change the display of **QuickAccess** buttons while working on a document—from text to table to graphics and back to text again—can quickly become tedious, so having all the buttons you frequently use in one or two sets of buttons can reduce mouse clicks, thereby boosting productivity and minimising strain.

On many monitors and laptop screens, there can be a considerable amount of empty space to the right of the default set of buttons. You can use this space to display more buttons (and even repeat buttons that appear on other renditions of the **QuickAccess** bar). You do this by editing the file that controls the appearance of the **QuickAccess** and **Formatting** bars. This file is called `fmtoolbr.ini` and it contains simple code statements in plain text. Modifying this file will reconfigure the **QuickAccess** bar so that it displays, for example, the text and table buttons together in one set, or both sets of graphics buttons in another set. This is explained in “Method 1: Combining two sets of buttons” on page 9. A slightly different modification will allow all the buttons that you commonly use to be displayed on the screen simultaneously. This involves removing some

buttons, and is explained in “Method 2: Displaying all buttons simultaneously” on page 10.

This tutorial explains how to customise the **QuickAccess** and **Formatting** bars in FrameMaker by editing the `fmtoolbr.ini` file.

Constraints

I have used this method successfully in FrameMaker 4, 5, 6 and 8. I have not tested it in FrameMaker 7 or 9.

These instructions were written and tested on a monitor running at a resolution of 1280 × 1024. Monitors running at a lower resolution may not display all the buttons as described in these

instructions. On the other hand, monitors running at higher resolutions may be able to display more buttons across the screen.

Preparation

Locate `fmtoolbr.ini`

The `fmtoolbr.ini` file is in the `fminit` sub-folder. Navigate to your FrameMaker program folder and locate the `fminit` sub-folder. If you have a default installation of FrameMaker 8, the sub-folder is at:

```
C:\Program Files\
Adobe\FrameMaker8\fminit
```

Locate `fmtoolbr.ini` and make a copy of it. (Give the copy a meaningful name such as `fmtoolbr.ini.original`.) You may need to revert to the original file if your modifications don't work.

Note the structure of `fmtoolbr.ini`

Open `fmtoolbr.ini` in a text editor or other program that can save a file as plain text. (*WordPad* is suited to this task, more so than *Notepad*.) You will find some comments at the start of the file (preceded by semicolons) that explain the syntax for the statements in the file (see figure 2). You don't have to fully understand the syntax, as the modifications in this tutorial simply involve copying and pasting lines of code from one part of the file to another, or from an accompanying file.

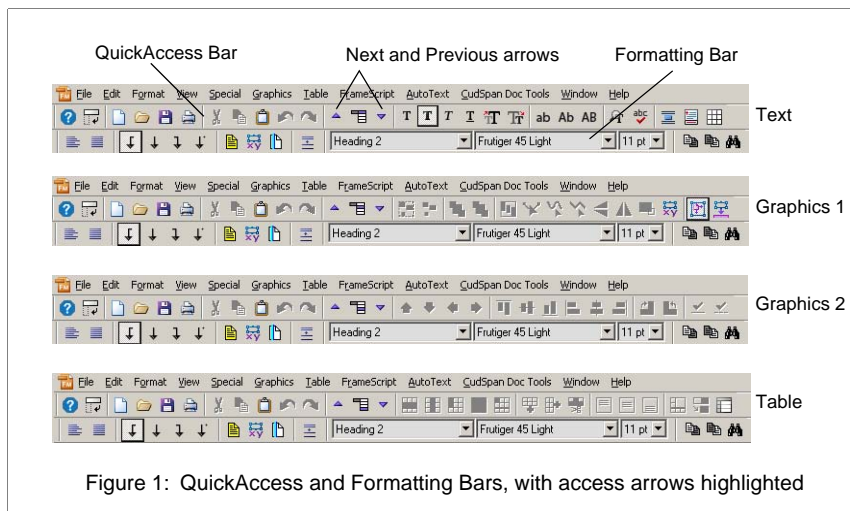


Figure 1: QuickAccess and Formatting Bars, with access arrows highlighted


```

; fmtreeolbr.ini
;
; Used for configuring QuickAccess Bar and Formatting Bar in Main Window
;
; Syntax for Statements:
; ICON: x=ICON , Icon Resource ID, Command Name [, Help String]
; COMBOBOX: x=COMBOBOX , Menu Name , Width [, Help String]
; POPUP: x=POPUP , Icon Resource ID, Menu Name [, Help String]
; SEPARATOR: x=SEPARATOR, Width
; LABEL: x=Label , String
;

[ToolBar]
HEIGHT=26
Y.OFFSET=1
TbMainPalette=On
TbFirstPage=On
TbSecondPage=Off
TbThirdPage=Off
TbFourthPage=Off

[TbMainPalette]
1=SEPARATOR , 5
2=ICON , ToolBarHelp , HelpQAB
3=ICON , ToolBarVertical , PaletteToggleHToV
4=SEPARATOR , 5
5=ICON , ToolBarNew , NewDocument
6=ICON , ToolBarOpen , Open
7=ICON , ToolBarSave , Save
8=ICON , ToolBarPrint , Print
9=SEPARATOR , 5
10=ICON , ToolBarCut , Cut
11=ICON , ToolBarCopy , Copy
12=ICON , ToolBarPaste , Paste
13=ICON , ToolBarUndo , Undo
14=ICON , ToolBarRedo , Redo

```

Figure 2: Part of the fmtreeolbr.ini file

If you page through the file you will see that the commands are grouped under headings. The code that controls the **QuickAccess** bar is located under the [ToolBar] heading, and the code for the **Formatting** bar is under the [RibbonBar] heading. Under each of these headings you will find the subheadings and applicable code for the various parts of each bar. These headings and subheadings are explained in table 1.

Most changes will be made in the [ToolBar] section of the file. However, Method 2 involves some changes to the [RibbonBar] section.

Syntax

Each line of code in fmtreeolbr.ini refers to one item on a bar, including the *separator*: the vertical bar between buttons.



Note that the word *Separator* is spelt incorrectly in this file, but it must be left that way.

Each line of code under a heading is numbered consecutively. If you change the code under a heading—by adding, deleting or moving lines—you must renumber the lines so that the numbering remains consecutive. If you don't, the toolbar may not appear at all and you will have to restart FrameMaker to get it back.

Any text following a semicolon is treated as a comment. This means you can add comments throughout the file to explain what you've done.

Table 1: Overview of the code in fmtreeolbr.ini

Heading	Function
[ToolBar]	Main heading of the QuickAccess bar
[TbMainPalette]	Controls the buttons to the left of the up and down arrows (that is, the persistent buttons)
[TbFirstPage]	Controls the up and down arrows and the text editing buttons to the right
[TbSecondPage]	Controls the up and down arrows and the first set of graphics editing buttons to the right
[TbThirdPage]	Controls the up and down arrows and the second set of graphics editing buttons to the right
[TbFourthPage]	Controls the up and down arrows and the table editing buttons to the right
[RibbonBar]	Main heading of the Formatting bar
[AlignmentPopup]	Controls the text alignment popup button
[SpacingPopup]	Code for the text alignment popup button
[TabWell]	Code is for the tab, import file, object properties, and view-only buttons; also for the symbols popup button
[ParaFormatPalette]	Controls the combo box for displaying and selecting paragraph formats, fonts and point sizes; also for the go-to-first page, go-to-last page and find-next buttons
[AlignWell]	Contains the commands for the popup that is produced by pressing the text alignment popup button
[SpacingWell]	Contains the commands for the popup that is produced by pressing the text spacing popup button
[TipText]	Controls the tip text that appears when the cursor is hovered over a button

Method 1: Combining two sets of buttons

This is a simple copy-and-paste operation that will configure the **QuickAccess** bar to display the text and table buttons combined in one set, and both sets of graphics buttons combined in another set.

1. Ensure that you have made a back-up copy of fmtreeolbr.ini and that FrameMaker is *not* running.

- If you have not already done so, open `fmttoolbr.ini` in a suitable text editor. (*WordPad* is suited to this task, more so than *Notepad*.)
- The code for the text buttons is in [TbFirstPage] and can be left alone. Scroll down and copy lines 5 to 22 of the [TbFourthPage] section.
- Paste the lines from [TbFourthPage] after the last line (line 22) in [TbFirstPage].
- Renumber each line of the pasted code so that it is consecutive with the original code: line 5 of the pasted material becomes line 23, and so on).
- The code for the first set of graphics buttons is in [TbSecondPage] and can be left alone. Scroll down and copy lines 5 to 22 of the [TbThirdPage] section.
- Paste the lines from [TbThirdPage] after the last line (line 22) in [TbSecondPage].
- Renumber each line of the pasted code so that it is consecutive with the original code.
- Save `fmttoolbr.ini`.
- Run FrameMaker and use the up or down arrows to display the modified toolbars. (On start-up FrameMaker displays the toolbar in the rendition that was showing when you last quit the program.)
 - Check that the modified text toolbar now has the table buttons at the right (as in figure 3).

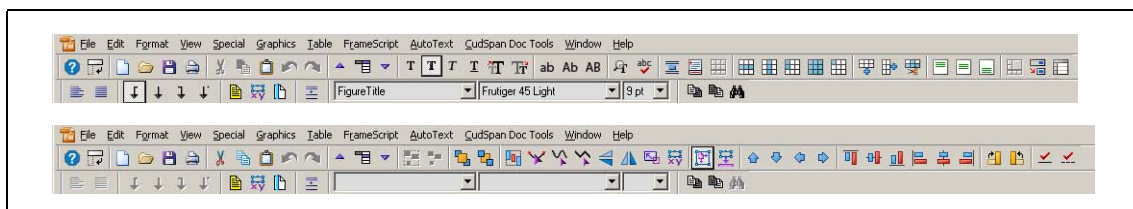


Figure 3: Modified Quick Access bars

- Check that the modified graphics toolbar now has the second set of buttons added to the right (see figure 3).

You now have all the toolbar buttons available in two sets instead of four: one has all the buttons mostly used when editing text, and the other has all the buttons mostly used when editing graphics.

Method 2: Displaying all buttons simultaneously

This procedure is a bit more advanced and will allow all the buttons to be displayed at once. The first stage is to configure the **QuickAccess** bar to display the text, table and first set of graphics buttons combined in one set (as [TbFirstPage]). We will also remove some buttons that can be considered redundant, as

they duplicate standard Windows keyboard shortcuts, such as `CTRL + O` and `CTRL + S`.

The second stage is to copy the second set of graphics buttons so that they appear at the right-hand end of the **Formatting** bar.

Edit the QuickAccess bar

- Ensure that you have made a back-up copy of `fmttoolbr.ini` and that FrameMaker is *not* running.
- If you have not already done so, open `fmttoolbr.ini` in a suitable text editor.
- Delete all 14 lines of code from [TbMainPalette], leaving just the heading in place. This removes the buttons for the following commands: help, new document, open, save, print, cut, copy, paste, undo, redo, and toggle the **QuickAccess** and **Formatting** bars from horizontal to vertical.

If you want to use the toggle button, leave lines 1 and 3 in place and renumber line 3 as line 2.

- Delete line 12 from [TbFirstPage]. This is just a separator bar and is not needed.
- Cut lines 5 to 22 from [TbFourthPage].
- Paste the lines from [TbFourthPage] after line 15 in [TbFirstPage].
- In [TbFirstPage] cut the last line:

```
22=ICON      , ToolBar1InsertTable , TableInsert
```

and paste it above the following line:

```
12=ICON      , ToolBar4AddRows      , AddRowsBelow
```

This has now grouped all the table-related buttons together.

- Cut lines 5 to 22 from [TbSecondPage].
- Paste the lines from [TbSecondPage] after the last line in [TbFirstPage].
- Renumber all the lines of code in [TbFirstPage] so that they are consecutive.

The code in [TbFirstPage] should now look like section E in the attached file (titled `Custom toolbars in FrameMaker.txt`).

The snippets of code that are referred to in this article are also provided in an attached file (titled `Custom toolbars in FrameMaker.txt`). You can copy and paste from this file if you prefer. If you are unsure how to find and open a PDF attachment, see "Packaging files with Acrobat" on page 3.

Edit the Formatting bar

1. Cut lines 5 to 22 from [TbThirdPage].
2. Paste the lines from [TbThirdPage] after the last line in [ParaFormatPalette].
3. Renumber all the lines of code in [ParaFormatPalette] so they are consecutive.
The code in [TbFirstPage] should now look like section G in the accompanying file.
4. Save `fmttoolbr.ini`.
5. Run FrameMaker and use the up or down arrows to display the modified toolbars.

- Check that the modified **QuickAccess** bar now displays the text, table and first set of graphics buttons.
- Check that the modified **Formatting** bar now displays the second set of graphics buttons at the right-hand end.

You now have all the toolbar buttons available on-screen at the same time, as shown in figure 4.

Conclusion

Once you've successfully configured and used the new toolbars, you may find that you want to carry out further modifications.

The modifications described here are only examples of what can be done. Experimenting with different modifications will establish what works for you. As long as you keep a copy of the original `fmttoolbr.ini` file, you can always revert to the original file and start again if things go pear-shaped.

When you are satisfied with what you've done, I recommend you print out your `fmttoolbr.ini` file, date it, and file it as a reference of what you've done.

Advanced editing

Once you are familiar with editing `fmttoolbr.ini`, you may wish to further personalise the toolbars. Here are some suggestions.

Separators

You could tidy up the appearance of the toolbars by removing or moving the separator lines so that the buttons are grouped in a way that is logical to you.

Moving or deleting buttons

You could rearrange the buttons into a layout that you find more convenient. These changes are simply a matter of deleting or moving lines of code and renumbering the lines so that they are consecutive.

Modifying the Rotate Graphics buttons

The standard Rotate Clockwise and Rotate Counterclockwise buttons rotate the selected object in 15-degree steps. I prefer these buttons to rotate the selected object in 90-degree steps. Here is how to do this.

1. Locate the following two lines of code in `fmttoolbr.ini`:

```
18=ICON      , ToolBar3RotateClockwise,
RotateGfxClockwise
19=ICON      , ToolBar3RotateCounter ,
RotateGfxCounterClock
```
2. Change the command names to read exactly as follows (and note that they are case sensitive):

```
18=ICON      , ToolBar3RotateClockwise,
RotateClockwise90
19=ICON      , ToolBar3RotateCounter ,
RotateCounterclock90
```

The line numbers shown here are examples only and may be different from those in your file. As always, keep the line numbering consecutive.

3. You can also modify the tool tips to match the new commands. Locate the following lines of code under the [TipText] heading at the end of the file:

```
RotateGfxClockwise=Rotate Clockwise
RotateGfxCounterClock=Rotate Counterclockwise
RotateClockwise90=Rotate Clockwise 90
RotateCounterclock90=Rotate Counterclockwise 90
```

and change them to read as follows:

The tool tips that appear when you hover the cursor over the buttons will now correctly describe the commands.

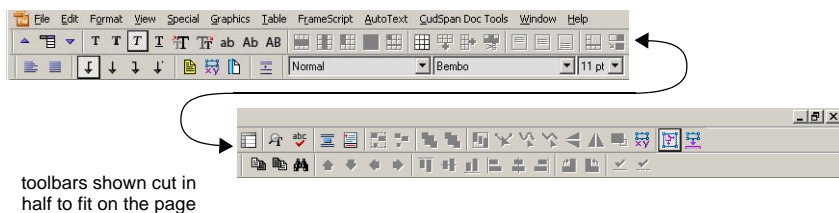


Figure 4: Modified Quick Access and Formatting bars

Dave Reynolds

Dave Reynolds is a senior technical author at Tait Electronics in Christchurch, New Zealand. He has been with the company since 1985, during which time he has worked on numerous documents relating to the company's two-way radio equipment.

Subscribe to Words

Words is a free, quarterly bulletin for technical writers and communicators—indeed, for anyone with a passion for words and for what you can do with them. To subscribe, send an email to:

words@abelard.com.au

Pitfalls in procedure writing

Geoffrey Marnell

Procedure writing is the bread-and-butter of technical writing. Most technical writers spend most of their working time writing procedures: instructions to explain how to use or service a product. Writing procedures is not rocket science. In large measure, it is just the application of primitive logic: what steps need someone take to achieve such-and-such a goal?

But if we want to write procedures from our customary *audience-centric* perspective—the perspective that in large part defines our profession—then we need to consider more than just *any* way of achieving some goal and letting that pass as a worthy procedure. We need to think about, and respect, our readers' safety and their time. We need to look out for our readers, warning them of risks and preventing them from rushing into unwanted circumstances. And we need to minimise the effort—physical and mental—our readers need to achieve their practical goals. Might it be easier, or quicker overall, to add in another step that, although logically unnecessary, is practically beneficial (such as a step to remove some unrelated component to make it easier for the user to get at the component they need to get at)? Or might it be easier or quicker for the majority of my likely readers to put one particular step before another?

These are issues that go beyond mere language. But they are issues that, when considered and applied, can turn an everyday workable procedure into a gem of simplicity and efficiency.

Our readers will rarely notice the extra effort that has gone into a well-considered procedure, and thus our extra efforts will be unappreciated. But that is also the case with writing in general. Clear, transparent writing mostly goes unnoticed. (When they read a well-written sentence in a novel or a user guide, readers rarely appreciate the behind-the-scene labours of drafting and redrafting, watered by the sweat of writer's block and editor surveillance). A procedure is much like a sentence in this regard: its structure is only noticed if there is something odd, unexpected or careless about it.

In what follows are some common problems in procedure writing, problems that get in the way of our over-arching goal: to have our readers complete their tasks with the least effort and least risk. The list of problems is in no way complete, and no list might ever be complete. But the point of this article is not to provide comprehensive advice on procedure writing. Rather it is to underscore the sometimes overlooked fact that logic as much as language is needed in writing audience-centric procedures.

Common problems

Placing the condition after a conditional action

A conditional action is one that needs to be done only if a particular condition is the case.

6. If you want to delete all the records, press **OK**.

6. Press **OK** if you want to delete all the records.

In this example—one direction expressed in two ways—the action of pressing **OK** is *conditional upon* the user wanting to delete all the records. But note how very different—practically as well as linguistically—the two versions are.

In the first version, the conditional clause (the *if* clause) precedes the imperative (or action statement); in the second it follows the imperative. That is the linguistic difference. But consider the *practical* difference, namely, the difference in the probabilities of readers doing something they didn't intend to do. The hurried user—as most of us are these days—is more likely to unintentionally do the action if the step is expressed in the second way—with the imperative before the condition—than in the first. They see the command and, swept along by the maddening rush of today's zeitgeist, obey it in the split-second between reading it and reading the condition that follows, that is, before realising that they actually had a choice, and that their choice might well have been *not* to do the action. (Oh bugger! I didn't really want to delete all the records.)

We should never assume that all our readers will have the time for slow and careful reading, free of breath on the neck. Thus in procedure writing, a conditional clause should always be placed *before* the imperative: “if *x* is the case, do *y*” and never “Do *y* if *x* is the case”.

More than the minimum number of steps

Economy is a worthy goal and not just in our choice of words: it is also a worthy goal in designing procedures. The least number of steps a user has to read in order to complete the procedure the better, *ceteris paribus*.

Example 1

Can you see the problem with this snippet?

5. If you have Windows 98 installed, do ...

6. If you have any other version of Windows installed, do ...

The problem is this: almost no one these days has Windows 98 installed, so almost every reader will have to read a step (or part of a step) that is of no relevance to them. And that is wasting their time. It is far better to reverse the order of these two steps. Now the majority of readers will know what to do after reading just one step.

Example 2

What is the problem with this similarly flawed snippet, aimed at a general readership?

1. If you are male and under 35, do ... and go to step 4.
2. If you are male and 35 or over, do ... and go to step 5.
3. If you are female, do ... and go to step 6.

One way to approach audience-segmented procedures like this is to employ what might be called the *effort index*. The effort index is the percentage of my overall audience multiplied by the number of steps that that segment has to read before they know what to do. So, as currently structured, 50% of my audience (the females) have to read three steps before they know what to do. So their effort index is 150. Youngish males have to read one step and they constitute about 25% of my audience (assuming a three-score-and-ten lifespan). So their effort index is 25. The rest of the males have to read two steps, so their effort index is 50. So the female effort index is 150 and the male effort index is 75, half as much. That is an unfair burden on the female readership. From an effort perspective, it is better to have written:

1. If you are female, do ... and go to step 6.
2. If you are male and under 35, do ... and go to step 4.
3. If you are male and 35 or over, do ... and go to step 5.

Now the effort index for the females is 50 and the effort index for any segment of males is no more than 50. To put it another way, in the rewritten version, the total effort index—for females and males—is 125, almost half of what it is in the first version (225). In other words, more people get to know what to do more quickly with version two (females first) than with version one (females last). Of course, steps 2 and 3 could be reversed without affecting the result (but only on the assumption that males under 35 are just as numerous as males 35 and over).

Example 3

Imagine two commercial greenhouses (A and B) in each of which grow various vegetables. Twenty degrees Celsius is the optimal temperature for growth in both A and B. In fact, the temperature should never be allowed to fall below 20, although sometimes it does. And sometimes it rises above 20 (which, although not optimal, is better than the temperature falling below 20). And often the temperature in A is different to the temperature in B. Moreover, temperatures are just as likely to be lower than optimal as they are to be higher than optimal.

Twice a day gardeners observe the temperature in A and B and adjust it if necessary. But they have just a single thermostat at their disposal, and whatever change of temperature it brings about in A, the same change occurs in B.

Your task is to write a procedure to instruct the gardeners what to do each time they check the temperature in the greenhouses. How do you proceed?

One obvious way is to list all the possibilities and write a procedural step for each one. In our example, admittedly contrived but still instructive, there are nine possibilities:

- $A > 20$ and $B > 20$, = 20 or < 20
- $A = 20$ and $B > 20$, = 20 or < 20
- $A < 20$ and $B > 20$, = 20 or < 20

Hence our procedure could take this exhaustive¹ form:

1. If A and B are both greater than 20, reduce the temperature until the temperature in the cooler greenhouse (or both) is 20 degrees.
2. If A is greater than 20 and B = 20, leave the settings as they are.
- ...
9. If A and B are both less than 20, increase the temperature until the temperature in the cooler greenhouse (or both) is 20 degrees.

But this is tedious: more so for the reader than the writer. For most conditions, four or more steps have to be read before the gardeners know what to do. But there is a better way.



**MACQUARIE
DICTIONARY
ONLINE**

www.macquariedictionary.com.au

Subscribe to the complete Macquarie Dictionary online, updated annually with new words and definitions.

Also available online is the full Macquarie Thesaurus - that perfect word is just a click away.

Try it out now for FREE!

Macquarie Online is offering free extended trial access. Simply contact Macquarie Online to set up your 3 months free access. Quote code: 3mfTrialAC

Macquarie Online Support
phone: 1800 645 349
email: support@macquarieonline.com.au



Australia's national dictionary

1. Pun intended.

If you consider which of the nine conditions *requires the same action*, it is possible to reduce this procedure to just three steps:

1. If A and B are both greater than 20, reduce the temperature until the temperature in the coolest greenhouse (or both) is 20 degrees.
2. If either A or B is less than 20, increase the temperature until the temperature in the coolest greenhouse (or both) is 20 degrees.
3. In all other cases, leave the settings as they are.

All conditions are covered: step 1 covers one condition; step 2 covers five and step 3 covers three. But now the most number of steps anyone needs to read in order to discover what to do is three: a vast improvement.

Being overly economical

But economical procedure writing—understood as minimising the number of steps needed for the majority of the anticipated audience to discover what to do—is not always what would please the majority of that audience.

Example 1

If economy of steps was our overriding goal—rather than minimising our readers' effort—we might be tempted, in very complex scenarios, to concatenate many conditions into the conditional clause introducing a step:

3. If A is true and either B or C is true or B is false but D is true, do ... and go to step 10.

Just as a sentence starts to become indigestible after the third clause (and sometimes before), a multi-condition conditional clause in a procedural step (as above) might well over-tax the cognitive abilities of many of your readers. Keeping a sentence to no more than two clauses is sound advice (overlooked by most academics). The parallel advice—keep your conditional clauses to no more than two conditions is equally sound advice. There is no point trumpeting the economy of your procedure if most of your readers cannot fathom its conditional complexities.

Example 2

Suppose, for instance, that you are writing a manual that explains how to service or replace certain parts in a piece of equipment. An economical procedure might state:

5. Remove the trim covering the power supply recess.
6. Remove the screws fastening the cable to the rear of the power supply.

But if only those with small hands and fingers are likely to reach with ease the screws at the rear of the power supply, then your economy of steps might have been purchased at the cost of time: the time it takes for the average service engineer to get those screws out. It may well have been easier for the majority of readers if you had added another step:

5. Remove the trim covering the power supply recess.
6. Remove the bracket securing the power supply to the main casing.
7. Slide the power supply forward a little and remove the screws fastening the cable to the rear of the power supply.

There is an extra step here, but it might save the majority of readers time, not to mention the pain of excoriated knuckles as they attempt what turns out for them to be next to impossible.

So economy is not just about minimising the number of steps in a procedure. That goal, laudable in the abstract, is merely a qualifiable component of the greater goal of minimising our readers' time and effort.

Undifferentiated literals

A literal is a word or string of words that the reader will see on the screen or on the product being documented. A field name is a literal, as is an error or confirmation message. If a step in your procedure must refer to a literal (as will often be the case), help the reader locate the literal on the screen or product by applying some typographical cueing. For example, you might adopt the convention—quiet common in contemporary technical writing—of setting field names in bold text.

Without some form of typographical cueing, the literal will appear undifferentiated from the surrounding text, and this could waste readers' time as they determine what exactly the instruction is.

4. Select GST and FBT, deselect invoices only and press Enter.
4. Select **GST and FBT**, deselect **invoices only** and press ENTER.

The second example makes it immediately clear that there are two options to select, not three (as the first example might suggest).

Typographical cueing is also commonly applied to the names of keys (as in the example immediately above) and to text that the user needs to enter.

7. Before the pressure reaches 100, type parameter 1887 and press **Enter**.
7. Before the pressure reaches 100, type parameter 1887 and press ENTER.

The Words team

- Artwork: Christine Weaver
- Copy-editor: Marcia Bascombe
- Cartoons: Bruce Ashley, Dave Carpenter
- Editor: Geoffrey Marnell
- Contact: words@abelard.com.au

© Individual contributors or Abelard Consulting Pty Ltd, 2010
[unless otherwise noted]

Without typographical cueing, what is the reader meant to type in: 1887 or parameter 1887?

Sub-branches not explicitly terminated before an unnecessary step

A procedure may have branches. At step 5, say, the user might be able to choose one of two paths. Steps 6, 7 and 8 take the user down one path; steps 9, 10, 11 and 12 take the user down the other path.

5. If you are running version 10 or earlier of Newfangle, do steps 6–8; otherwise do steps 9–12.

You need to make it crystal clear after step 8 that the reader who is using an early version of Newfangle has completed the task and can ignore the rest of the procedure. (In other words, there must be no chance that such a reader will think that step 9 is necessary once they have finished step 8.)

Here is one of a number of techniques.

8. Click OK.

You have now finished the pay-run. *Ignore the rest of this procedure.*

9. Select employees who have resigned in the last month.

If you think this is over-egging the soufflé, imagine yourself in a busy office. You have just completed step 5 and now know that you only have to do steps 6, 7 and 8. You are about to commence step 6 when the telephone rings. Ten minutes later you return to the procedure, finish step 6, start step 7, and your manager wanders across for a report on your project's progress. Twenty minutes later you return to the procedure, finish step 7, take a

desperately needed powder-room break and return to your desk and tackle step 8. You finish it and start step 9 and wonder why things are not working. Understandably, you didn't remember that step 8 is as far as you needed to go? Some would, but many wouldn't. And that's why the 15-seconds-to-write *buffer message* is essential in an embedded branch of a procedure.

Including steps that do not belong in the current procedure

Consider the following step:

4. Select **GST Liability**. You must subsequently create a GST Paid asset account. See "To set up GST asset accounts" on page 89.

This is a two-in-one step, and the second step is to be done independently of the current procedure. Presumably the writer of this procedure wasn't telling the reader to stop the current procedure now and go off and follow the cross-referenced procedure—something that was actually happening in this real-life example. Does the user then come back and start the current procedure all over again? That's unlikely, as it involves circularity: the user would never get past step 13. Or does the user resume the current procedure from step 14? If that's the intention, it's better to place the steps in the referenced procedure inside the current procedure. Branching to an external procedure is fraught with danger. You have to remember to add conditional steps to the external procedure to ensure that the user is appropriately redirected to the original procedure, which can become a nightmare if the procedure is referenced from a number of other procedures.

But if, as is more likely the case, the writer meant that the user should do the cross-referenced procedure after the current procedure, then the best place for the cross-reference is in the **Related Procedures** section of the current procedure. (It is too much to expect that the user, once they finish the current procedure, will go back and re-read the procedure looking for cross-references to other procedures that now need to be done.)

Late prerequisites

Do not delay specifying a prerequisite until the step that requires the prerequisite. All prerequisites should be specified in the preamble to the procedure so that the reader can attend to the prerequisites before commencing the procedure.

13. Select **GST liability**. This is possible only if you have clicked the **Business** option in the **Preferences** window—see 'Setting tax options' on page 23.

With a step like this, some users will have to back out of the procedure and start all over again once the prerequisite has been attended to, that is, once they have changed the specified preference.

HyperTrain dot Com

HyperTrain dot Com specialises in:

- Online Help
- HTML
- JavaScript
- CSS
- XML
- XSLT and
- DITA.

We also offer a full range of tech pubs services including group or individual training, project jumpstart, technical writing, and end-to-end project development.

Toll free (US): 888-722-0700
International: +1-760-214-0698
Web: www.hypertrain.com
Email: dgash@hypertrain.com

As a general rule, you need to tell the reader *in the preamble to a procedure* what tools, spare parts and information they need, and what previous actions they should have done. No engineer will be impressed with your bogey-removal procedure if you tell them, at step 20, to use a rattle gun to loosen the nuts on the bogey if it means that they need extricate themselves from the maintenance well, clean up and traipse back to the tools storeroom to find a rattle gun. You are just wasting their time.

Poorly placed risk messages

A topic akin to late prerequisites is poorly placed risk messages: but it is much more serious. Risk messages are commonly placed:

- in the preamble, if the risk is especially serious, and
- before or with the step at which they apply.

Placing a serious risk message in the preamble can help prepare the reader for the task ahead. Knowing that there are serious risks *before* they embark on a

procedure is likely to cause the reader to approach the task with the right frame of mind—with seriousness, concern and care—which may be unlikely at 4.30 on a Friday afternoon.

Repeating the risk message at the step where there is a risk reinforces the possibility of unwanted results. Moreover, such messages will be the only notification of possible risk for those hurried readers who skip the preamble.

Where the action itself is risky (as opposed to the state that the action brings about), it is imperative that the risk statement *precedes* the action. The following snippet—which breaks this rule—shows no concern for the welfare of the reader:

12. Turn the hydraulic key anticlockwise. The jack, and car, return to ground position.

WARNING: Make sure that you are not underneath the car when you perform this step.

Too late, she cried.

Geoffrey Marnell

Journals

Journal of Technical Writing and Communication


Contents of the current issue

Click the heading below to read the editorial, abstracts and reviews, or to purchase an article or subscribe.

Volume 40, Issue 3, 2010

- From the Editor's Desk, Charles H. Sides, Executive Editor
- The Children of Aramis, Michael J. Salvo, Ehren Helmut Pflugfelder, and Joshua Prenosil
- Communicating the Risk of Scientific Research, Timothy D. Giles
- Document-Based Decision Making and Design: An Analysis of Medicare Part D, Eva R. Brumberger
- The Banality of Rhetoric? (Part 2): Alternate Views of Technical Communication and the Holocaust, Mark Ward, Sr.
- An Application of Robert Gagné's Nine Events of Instruction to the Teaching of Website Localization, Pinfan Zhu and Kirk St. Amant
- The Coaching and Mentoring Process: The Obvious Knowledge and Skill Set for Organizational Communication Professors, Robert H. Stowers and Randolph T. Barker

award-winning, authoritative international voice, publishing the latest research by recognized scholars from around the globe




Every article ever published in the JTWC is now available in clear, concise, comprehensive PDF format.

New Online-Only Delivery Options

- Enhanced subscription (all articles from volume 1 through current volume)
- Current volume subscription
- Individual articles on pay-per-view basis

Click here for all the details and ordering information.

- Read all article abstracts free
- Download a complimentary sample issue

 **BAYWOOD PUBLISHING COMPANY, INC.**
<http://baywood.com>

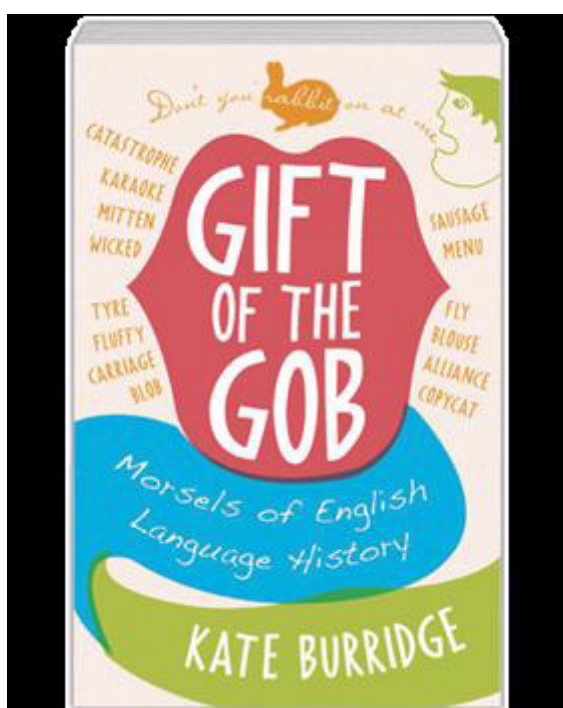
Book review

Viân Lawson

Gift of the Gob: Morsels of English Language History by Kate Burridge (ABC Books, 2010, 306 pages, \$28.00)

Communication professionals strive for simple, neutral and universally understood vocabulary in their daily meat-and-potatoes writing, but this book is the equivalent of the dessert course. Words like *grog-blossomed* and *firkytoodling* are delicious, whimsical, a little decadent and terribly moreish. Kate Burridge's *Gift of the Gob*, the third book based on her television and radio appearances, is a lively and affectionate exploration of modern English, lent subtlety, wit and scholarship by her careful observation of the past. The main argument of the book is in the title—spoken language is full of delights, to be savoured, rolled around on the tongue and appreciated for their sensuality as well as mere sense.

As she did in *Blooming English* and *Weeds in the Garden of Words*, Burridge bases her chapters in *Gift of the Gob* on the various segments she has presented for ABC Radio and the TV show *Can We Help*. This is no mean feat given that she bases her broadcasts on whatever has hit her mailbox and taken her fancy. She uses questions sent in by ABC listeners and viewers as a jumping-off point to share insights into the rise of words, their use and change over time, and their decline. She manages to weave in key linguistic phrases and concepts, but never at the expense of entertainment value. You may learn something in passing, but the journey, not the destination, is the joy here. Even with comprehensive endnotes, index and bibliography, *Gift of the Gob* is less a reference and more a pleasure-read.



Burridge, a Professor of Linguistics, is a good old-fashioned descriptivist. She does not believe that there is one standard or correct English. She does not see the rise of Leetspeak¹ or the other demotics of the age as a sign of the End Times. For her, English is not made up of the words and phrases people should use: it's made up of the words they *do* use. She is either positive or carefully neutral when discussing neologisms, which provide the most colourful and dynamic evidence that English is alive and flourishing. She's enthusiastic about portmanteau words like *affluenza* and *coca-colonization*, for the new ideas they encapsulate, and links *iGadgets* and *McMansions* to their linguistic predecessors, *Xerox* and *Kleenex*. She also takes pleasure in pointing out that many of the things which are tut-tutted by the purists have long, inglorious histories. For example, *irregardless* has been a word in use for centuries, and frowned upon for nearly as long. She occasionally admits that certain modern trends are not to her taste—leaving the second *l* out of 'groveling' at the behest of her editor, for example—but she is not grumpy about the fluidity or infinite variety of language.

Burridge reserves her rare negative comments for the prescriptivists—those arbiters of linguistic hygiene who get a lot of attention, and give the rest of us a bad name, by using a style guide like a snob uses an etiquette manual. Burridge sees them as archaic and, as often as not, wrong. There's a whiff of *schadenfreude* about her account of Lynne Truss's error in correcting the film title *Two Weeks Notice*, which morphs nicely into a discussion of the difficulty of determining possession or lack thereof in such phrases.

Burridge differentiates between the self-appointed Language Police and the descriptivist authorities, such as the compilers of dictionaries. The pleasant job of a linguist is to chronicle what is happening in the language, rather than pronouncing a word correct or incorrect. In the past, writers such as Samuel Johnson noted when a word was only used by the low or rough, but they did not counsel against using it. Indeed, the most cursory reading of Johnson's diaries shows that he was a man who was not afraid of low language when he thought it was needed!

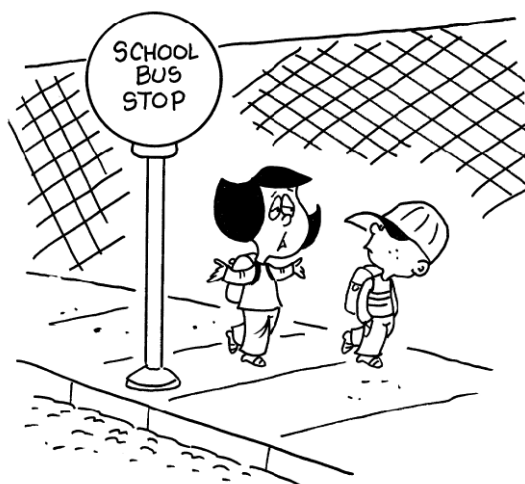
The constant adaptation and appropriation of language to express new ideas, Burridge and many others argue, gives English its flexibility and strength, and is the reason it has become the *lingua*

1. *Leetspeak* is the type of writing in which the writer replaces standard letters with other characters (mostly preserving the original pronunciation). For example, *xceler8* is leetspeak (aka *leet*) for *accelerate*.

franca of the electronic age. Burridge argues that the vitality and flexibility of English are driven from below. In the past, thieves' cant and the words of the urban poor were the wellspring of language. Now, interesting new trends spring from the lips of the young and from people who have little formal schooling in English. Dynamic turns of phrase also come from geeks, who were marginalised until they became technocrats in the mid-90s. Even conservative speakers *google* and *tweet* now, when they are not too busy *texting* or *blogging*— verbs for activities we didn't even have a decade or so ago.

Burridge does not fear that we are going to lose our Australian voice to an influx of web-driven Globish. Strine is not dead, but transformed. *Mate* and *strewth* may only be used ironically by today's urban sophisticates, but *yeah-no* and *versing* (*versus* as a verb) are on the rise.

An entire chapter reassures us that we still have the highest rates of casual swearing and abuse in the English-speaking world, although the nature of blue words has changed from religious to anatomical as social attitudes have altered. Most real swear words today, she argues—that is, words which retain a value to shock and offend—are racial and religious terms, like the *n*-word. Prime-time television has worn the barbs off most of the traditional cusswords. When Burridge speaks of the degradation of language, she speaks of good words going bad: the semantic drift downwards endured by words like *language* (as in *Caution: contains ...*) and *attitude* (as in *cop an*), just as *cunning* did centuries ago.



"BUT IF YOU DON'T LEARN TO READ AND WRITE, HOW ARE YOU EVER GOING TO TEXT?"

© Dave Carpenter 2010

But for most of the book, Burridge is content to pick out the choicest, most rollicking and evocative bits of language to illustrate how words come into existence, change, and eventually fall into desuetude (like *desuetude*). For her, spoken language, and particularly the informal language of any age, has sensory as well as semantic delights, and words are toys as well as tools. Every page has fascinating word origins and lexicographical gems to admire. She points out *Kleenices* as a gorgeous plural form, and encourages us to pillage the language of the past when next we want to insult someone, as it's been far too long since someone's been called a *shotten herring slubberdegullion*.

Most of us work with language because we love it, but this book is about the words we play with. It reassures us that playing with language is an important and universal instinct which both amuses us and drives the language forward. It's a fast and easy read, but no less fascinating for it, and for those of us who revel in odd and evocative words, it's a banquet.

Viân Lawson

Viân Lawson teaches instructional writing, corporate writing and writing for the web in the Department of Professional Writing and Editing at Holmesglen Institute. She also teaches technical writing for Abelard Consulting. Viân has worked for over a decade as a technical communicator, corporate communicator and educator in Australia and in the US.

Tips and Tricks

You can link to a PDF document from an HTML page quite easily. Just specify the name of the PDF file in the HTML anchor:

```
<a href="pdf_file_name.pdf">text</a>.
```

Click *text* and the PDF file opens ... at page 1.

What is less well-known is that you can also set the link so that a particular page in the PDF file opens when the link is clicked. The likely suspects—named anchors and PDF bookmarks—are not, surprisingly, the way to do it. Instead, you have to directly specify the page *by its number*, as in:

```
<a href="pdf_file_name.pdf#page=56">text</a>.
```

In this case, clicking *text* opens the PDF at the top of page 56.

Miscellany

Lose some; make some up: a competition

Thanks in large part to the designers of the computer keyboard, the limitations of on-screen keyboards and the all-too-human disposition to seek out the path of least effort, some once-useful punctuation marks are becoming moribund and soon, no doubt, will be dead. Think of the en dash (aka en rule). Think of the apostrophe of elision. Think of the double inverted comma.

But the fact that some punctuation marks are leaving common usage doesn't mean that all punctuation will eventually die out. New marks could well come into existence (and old ones could well be exhumed). All the marks we now have were born at some stage, and some surprisingly recently. For example, the singular possessive apostrophe came into widespread use only in the eighteenth century, and the plural possessive apostrophe in the century following.

Now someone must have started the movement to show possessiveness well before a critical mass of users made it common usage. Likewise, someone ninety or so years ago must have stopped putting a space between the last word in a sentence and the question mark or exclamation mark that followed, or a space before a colon and semicolon. Now, of course, it is common practice to leave out that space. The sticklers in our midst might even say that it's wrong or incorrect to put a space before a colon or semicolon, overlooking the unavoidable historical relativism of language-*qua*-convention.

So, on the one hand we are losing punctuation marks (and some quite useful ones, as the en dash is, or was) and on the other there is nothing stopping us inventing new ones (and perhaps becoming the originator of a twenty-second century common practice). So here is your chance to be such an originator, to be the person that language manuals of the future thank as the inventor of a useful punctuation practice. Send us your suggestions for new punctuation marks. The one judged the most useful—in contributing most to the effortless transfer of meaning should it become common usage and being relatively easy to insert into a document—wins a copy of Kate Burridge's new book (reviewed on page 17 of this issue).

Here's one suggestion to get you in the mood. Inverted commas have got out of control. They are used to enclose quotes, show irony, indicate an invented expression, to enclose technical words in non-technical documents, to indicate screen or product literals, and so on. With such an abundance of usages, readers often have to work out what particular usage has been adopted in a particular

case. And when readers have to work out what the writer intended, as opposed to getting the message immediately, the writing is less than optimal. Ambiguity is lurking. So what about using, say, angle quotation marks to indicate irony?

The government's <policy> on science education has been sent back to the finance committee.

The characters might not be on the keyboard, but they are not hard to insert: ALT + 0139 and ALT + 0155 respectively.

Or maybe the following:

The government's 'policy, on science education has been sent back to the finance committee.

In this case the characters are directly enterable from the keyboard. No need to remember any ASCII or unicode code-points. Or perhaps to avoid the potential misreading of the final mark as a common comma, we can swap the marks:

The government's ,policy' on science education has been sent back to the finance committee.

Odd and unconventional now, but with just one intended meaning—irony—these marks eliminate ambiguity and thus effort on the reader's part ... should the usage become common practice.

There are many other cases where everyday ambiguity could be overcome with the help of punctuation. So, over to you. Send your ideas to words@abelard.com.au before 30 September 2010. They will be published in the November issue of *Words*.

Fun with ambiguity

On the subject of ambiguity, the following newspaper headlines were no doubt intentionally ambiguous. But they're amusing nonetheless, if only in part and only to some minds.

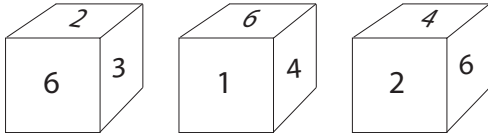
- Drunk gets nine months in violin case
- Enraged cow injures farmer with axe
- Teacher strikes idle kids
- Juvenile court to try shooting defendant
- Two sisters reunited after 18 years in check-out counter
- Iraqi head seeks arms
- Squad helps dog bite victim
- Miners refuse to work after death
- Stolen painting found by tree
- Two Soviet ships collide, one dies

Mindstretchers

Geoffrey Marnell

A Little Cubist Nonsense

What is the sum of the digits on the faces of the following cube, shown from three angles?



The solution will appear in the next issue of *Words*.

Solutions to the last puzzle

Puzzle

The puzzle in the last issue of *Words* asked:

In what climatic conditions are each of the following sentences, taken in turn, true: When it is raining? When it is fine? When it is either wet or dry? Or under no circumstances at all?

1. It is raining and this sentence is false.
2. Either it is raining or this sentence is false.
3. If it is raining, then this sentence is false.
4. It is raining and at least two of these four sentences are true.

Let's take each sentence in turn.

Sentence 1

This sentence could not be true under any circumstances. Consider the complete sentence as composed of two component sentences: "It is raining" and "This sentence is false". The complete sentence is true only if both these component sentences are true. But if this were the case, "This sentence is false" (which refers to the complete sentence) would be true, making the complete sentence false. Hence it is impossible for the complete sentence to be true.

Sentence 2

The sentence is true only if it is raining. The four possibilities can be represented like this:

	It is raining	This sentence is false
1	T	T
2	T	F
3	F	T
4	F	F

Now, a sentence composed of two disjuncts (i.e. two statements separated by *or*) is true only if at least one of the disjuncts is true. Possibility 1 requires that the complete sentence be true, but this would mean that the complete sentence is false (see the second disjunct). This contradiction renders the whole sentence logically inconsistent. Likewise with possibility 3.

Possibility 4 can be immediately ruled out, as it makes the complete sentence necessarily false. (Anyway, this possibility is inconsistent, for the complete sentence must be false and yet it contains a component—the second disjunct—which says, under these circumstances, that it is false that the whole sentence is false.)

Only possibility 2 is consistent. In it, at least one disjunct is true—making the whole sentence true—and this is consistent with it being false that the complete sentence is false (see the second disjunct). And the disjunct that is true, here, is "It is raining".

Sentence 3

This sentence is true only when the weather is fine. A conditional sentence (i.e. one in the form "If x , then y ") is always true except in the case where the antecedent (i.e. the component represented by x) is true and the consequent (the component represented by y) is false. Consider, first, the case where the consequent (i.e. the sentence "This sentence is false") is true. This sentence, if true, is saying that the complete sentence is false, and yet, by the abovementioned logic of conditional sentences, the complete sentence must in this case be true.

This leaves, as the one remaining possibility where the complete sentence would be true, the case where both antecedent and consequent are false. If the consequent is false, then it is false that the complete sentence is false, which is, obviously, consistent with the truth of the complete sentence. And the antecedent is false only when the weather is fine. In other words, the complete sentence can be true only if it is not raining.

Sentence 4

Given that sentence 1 could not possibly be true (see above) and that we are, here, testing the truth of sentence 4, the only combinations of at least two sentences we need to consider are (i) sentences 2, 3 and 4, (b) sentences 3 and 4 and (iii) sentences 2 and 4. We have already established (see above) that 2 is true only if it is raining, and that 3 is true only if it is not raining. Now, for sentence 4 to be true, it must be raining. Hence sentences 3 and 4 cannot both be true (for one is true if it is raining and the other is true if it is not). This tiles out combinations (i) and (ii), leaving just the third. And this latter combination is true only if it is raining.