A gallery of DTX files

Will Robertson

August 6, 2007

1 Introduction

This work is a collection of files that demonstrate simple things that are possible with the flexible and under-appreciated docstrip file format. It is intended as a companion to Scott Pakin's excellent and influential 'dtxtut' example of producing LaTeX packages in this way.

2 Files in the gallery

The DTX examples in this gallery are listed below in approximate order of both relevance and complexity.

single-source.dtx

This is an example of both the .sty and .ins files being extracted from the .dtx file. This allows maintenance of a single source file but still produces a 'standard-behaviour' DTX/INS pair for upload to CTAN.

conditional-code.dtx

This example shows conditional extraction of code into multiple files, allowing the easy generation of 'debug' versions of package, for example.

rearrange.dtx

This example demonstrates how source code may be written and documented in a different logical order than in which it is required to execute.

This can be useful, for example, when default values are provided at the last stage of the package loading, but you wish to describe them first in the main body of the (non-technical) documentation.

dtxgallery.dtx

This very document is an example of a DTX file that is itself the compilation of multiple, external, DTX files. It is useful for generating a single documentation file for a large software collection (e.g., the LATEX sources themselves).

3 *How to use this gallery*

It will be clearest to now stop reading this document and take a look at the individual DTX files (and their compiled PDF files) mentioned above. The rest of this file contains a compilation of the DTX files in the gallery, which is an example in itself. Read about it later on in File IV on page 6.

4 *Interlude: eliminating guards*

While I'm in a position to comment on some things, I may as well put in a few tips that I use when writing DTX files.

'Guards' are what delimit code sections in DTX files. Many packages will have no need for anything fancy in this regard, and it will be quite common to see things like:

```
% ...
%   \begin{macrocode}
%<*package>
⟨The entire, commented, package code⟩
%</package>
%   \end{macrocode}
% ...
```

with other guards used to section off things like the driver code in the beginning and possibly other bits and pieces. In such cases, it is not necessary to include the guards in the typeset documentation (if we're trying to make things as simple as possible for the reader), so I like to write instead:

```
% ...
% \iffalse
%<*package>
% \fi
%   \begin{macrocode}
⟨The entire, commented, package code⟩
%   \end{macrocode}
% \iffalse
%</package>
% \fi
% ...
```

This is also useful when logical part/section markup is used to separate the files in the DTX anyway.

5 The documented source code

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{single-source}[2005/07/15 v0.1 dtx: single-source]
\def\mymacro{hello :)}
⟨/package⟩
```

6 The contents of single-source.sty

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{single-source}[2005/07/15 v0.1 dtx: single-source]
\def\mymacro{hello :)}
```

7 The contents of single-source.ins

```
\input docstrip.tex
\keepsilent\askforoverwritefalse
\nopreamble\nopostamble
\generate{\file{\jobname.sty}{\from{\jobname.dtx}{package}
                              \from{\jobname.dtx}{defaults}}}
\endbatchfile
```

8 The contents of single-source-readme.txt

```
-----------------
single-source.dtx

This is an example of everything
being extracted from the .dtx file.
-----------------
```

FILE II — CONDITIONALLY EXTRACTING CODE

9 Conditional inclusion of code

From previous examples the `%<*guard>`...`%</guard>` syntax should be a little familiar. This example demonstrates conditional extraction of such elements in the source document into multiple files. This technique makes it easy, say, to maintain a 'debug' version of a package without polluting the public source with code for testing.

```
⟨A⟩  code in ‘A’
⟨B⟩  code in ‘B’
⟨!A⟩  code not in ‘A’
⟨!B⟩  code not in ‘B’
⟨A & B⟩  code in ‘A’ and ‘B’
⟨A | B⟩  code in ‘A’ or ‘B’
⟨(A | B)&!(A & B)⟩  code in ‘A’ xor ‘B’
```

Note the change in the typeset source when guards are nested:
(this is equivalent to `%<A&B>`)

```
⟨*A⟩
⟨B⟩  ‘B’ nested inside ‘A’
⟨/A⟩
```

10 Verbatim files that are produced

10.1 Generated from 'A'

```
code in ‘A’
code not in ‘B’
code in ‘A’ or ‘B’
code in ‘A’ xor ‘B’
```

10.2 Generated from 'B'

```
code in ‘B’
code not in ‘A’
code in ‘A’ or ‘B’
code in ‘A’ xor ‘B’
```

10.3 Generated from 'A' and 'B'

```
code in ‘A’
code in ‘B’
code in ‘A’ and ‘B’
code in ‘A’ or ‘B’
‘B’ nested inside ‘A’
```

FILE III — REARRANGING THE CODE FROM LOGICAL TO LINEAR

11 *Example of re-arranging docstrip source*

This is a test to show it's working: \test → “1”

12 *Documented code*

This is ‘defaults’ code that the *user* might want to see:

```
⟨*defaults⟩
\mytest{1}
⟨/defaults⟩
```

And this is the internal code that the user doesn't care so much about:

```
⟨*package⟩
\ProvidesPackage{rearrange}[2005/07/15 v0.1 docstrip: rearranging]
\def\mytest#1{\def\test{‘‘#1’’}}
⟨/package⟩
```

13 *Verbatim files that are produced*

13.1 *rearrange.sty*

```
\ProvidesPackage{rearrange}[2005/07/15 v0.1 docstrip: rearranging]
\def\mytest#1{\def\test{‘‘#1’’}}
\mytest{1}
```

13.2 *rearrange.ins*

```
\input docstrip.tex
\keepsilent\askforoverwritefalse
\nopreamble\nopostamble
\generate{\file{\jobname.sty}{\from{\jobname.dtx}{package}
                              \from{\jobname.dtx}{defaults}}}
\endbatchfile
```

This very file, `dtxgallery.dtx`, is an example of a DTX file that contains no code of its own but which loads *other* DTX files.

Note that code could be included in this file like any other DTX file and typeset along with the following examples, but I've chosen to strip this particular DTX file as far back as possible simply to demonstrate that I can. Therefore, it generates no files, and provides no code. It simply documents other DTX files.

Note that this file draws into relief the difference between the two locations that 'documentation sources' can be placed in a DTX file:

example.dtx

```
% \iffalse
% ...
%<*driver>
\documentclass{ltxdoc}
...
\begin{document}
  ⟨Documentation source — specific⟩
  \DocInput{example.dtx}
\end{document}
%</driver>
% \fi
% ⟨Documentation source — general⟩
% ...
```

The 'specific' documentation source is used when compiling only that DTX file, whereas *only* the 'general' documentation source appears when input via `\DocInclude`. In the case of a compilation of DTX files (which is what the document you're reading demonstrates), `\DocInclude` is referring to the hypothetical `example.dtx` in a completely different file, so the specific part of the code is invisible.

Therefore, it is a good idea to define formatting and document metadata (such as `\title` and `\maketitle`) in the 'specific' location, and include only the body of the documentation in the 'general' area.

This file is an example of how formatting code in the 'specific' area here changes the fonts and layout used only in the DTX gallery itself.