Computer Science & Software Engineering

http://www.csse.monash.edu.au/

CSE 1402: Technical Documentation for Software Engineers

Carlo Kopp, BE(Hons), MSc, PhD, PEng

Monash University, Clayton, Australia
email: carlo@csse.monash.edu.au

© 2004, Monash University, Australia
Revision Information

This document is currently at revision level:
Software Tools for Documentation
Text Based Documentation

- Text based tools for producing and displaying documentation are the simplest type of tool used for producing documentation on a computer.

- Text based documentation is the least demanding in terms of storage required, and in terms of computing performance to run.

- Any software tool which can manipulate text files can be used to produce text based documentation.

- Text based documentation can be viewed on simple displays such as “dumb terminals” or “consoles”, since no Graphical User Interface is required.

- Text based documentation can be printed on the most primitive printers, including dot matrix types with no graphics capabilities.
Text Based Documentation

- Unix systems, which are widely used in industry as commercial servers for databases, web servers, online storage, software development, all rely on text based documentation for the operating system.

- Because text based documentation needs very little computer performance to display, and does not require graphics, it can viewed very quickly.
The Limitations of Text Based Documentation

- Text based documentation cannot display graphics, other than primitive diagrams.
- Text based documentation can frequently be difficult to use, especially for novices.
- Proprietary operating systems may not support industry standard text documentation formats such as “Unix manual pages” or “GNU info pages”.
- Some dumb terminals may have difficulty handling control codes.
- Many users do not like text based documentation since it is not as pretty as graphically displayed documentation.
Representing Text

- Computers store information as binary numbers.
- The smallest addressable chunk of binary data on most computers is a byte, which contains 8 bits.
- To represent text we need to have binary numbers for each and every letter of the alphabet, numbers from 0 to 9, and various punctuation symbols.
- Two representation schemes are widely used, “ASCII” and “EBCDIC”,
- EBCDIC is an old IBM format, still found on mainframe systems.
- ASCII is the standard format used on most small systems, and the Internet.
- “Standard” ASCII maps 128 numbers, letters, symbols and terminal control codes.
<table>
<thead>
<tr>
<th>ASCII</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 nul</td>
<td>01 soh</td>
</tr>
<tr>
<td>08 bs</td>
<td>09 ht</td>
</tr>
<tr>
<td>10 dle</td>
<td>11 dc1</td>
</tr>
<tr>
<td>18 can</td>
<td>19 em</td>
</tr>
<tr>
<td>20 sp</td>
<td>21 !</td>
</tr>
<tr>
<td>28 (</td>
<td>29 )</td>
</tr>
<tr>
<td>30 0</td>
<td>31 1</td>
</tr>
<tr>
<td>38 8</td>
<td>39 9</td>
</tr>
<tr>
<td>40 @</td>
<td>41 A</td>
</tr>
<tr>
<td>48 H</td>
<td>49 I</td>
</tr>
<tr>
<td>50 P</td>
<td>51 Q</td>
</tr>
<tr>
<td>58 X</td>
<td>59 Y</td>
</tr>
<tr>
<td>60 ‘</td>
<td>61 a</td>
</tr>
<tr>
<td>68 h</td>
<td>69 i</td>
</tr>
<tr>
<td>70 p</td>
<td>71 q</td>
</tr>
<tr>
<td>78 x</td>
<td>79 y</td>
</tr>
</tbody>
</table>

Table 1: ASCII Representation in Hexadecimal.
Representing Text

- Most dumb terminals and character printers are hardwired to recognise ASCII representation and display or print the shape of the letter or number, respectively.

- Some software tools can insert additional control characters which may interfere with the operation of software tools on other operating systems, requiring conversion.

- The most common difference in text file formats is that between Unix systems and Microsoft systems.

- At the end of every line of text, a Microsoft system inserts a “carriage return” and a “newline”, whereas Unix inserts only a “newline”.

- A conversion filter tool such as ‘unix2dos’ or ‘dos2unix’ is required.
The Structure of Text Files

- A text file is typically divided into lines of text, each separated by a "newline" character.

- Tools for viewing, editing or printing text will assume a line length, usually of 80 or 132 characters.

- When a "newline" is encountered, the device pads out any remaining characters with spaces and starts a new line.

- Many tools which allow cutting and pasting on graphics screens may also pad out lines.

- Almost all computer source code is kept in text files, which are read and parsed by compilers.
Example:

The three lines of text viewed as:

Mary had a little lamb.
The lamb was eaten by a nasty wolf.
Mary mourned her lamb.

... are represented inside the file as:

Mary had a little lamb. ← The lamb was eaten by a nasty wolf. ← Mary mourned her lamb. ←

The ← symbol represents the “newline” character.
Manipulating Text Files

- There are a range of tools which can be used to manipulate text files.
- Screen editors like vi, emacs or Notepad can be used to edit a text file on a screen.
- Stream editing tools like sed and awk provide a scripting language to perform identical changes on large numbers of different files.
- Search tools like grep, fgrep allow you to find specific strings of characters in large numbers of different files.
- Sorting tools like sort allow you to sort files by alphabetic or numerical order.
- Unix systems have a standard toolset which includes these and other tools, which are used for manipulating text files.
Recognising a Text File

• On a Unix system the `file` command will identify the type of a file by looking inside it and interpreting the first few characters.

• Sometimes `file` may guess wrongly with a text file, and assume it is source code for a language like C or C++.

• Non text files usually incorporate a “magic number” in the first few bytes of a file to assist the `file` command on Unix systems.

• Unix does not care about the naming of a file, since it looks inside the file to determine its type.

• Microsoft Windows and NT depend on the naming of a file to determine its type: therefore on a Microsoft system you must obey the naming convention and use a `.txt` suffix, e.g. `myfile.txt`. 
Text Formatters

- Text formatters are tools which use control commands, embedded in a text file, to control the layout of the file on a screen or printer.

- The most widely used text formatting tool is \texttt{nroff} (Unix), which is closely related to the obsolete \texttt{troff} typesetting tool.

- \texttt{nroff} is the basis of the Unix online help facility, called \texttt{man}.

- Unix \texttt{man} online help files are usually stored as \texttt{nroff} files, basically text with added \texttt{nroff} commands and macros.

- Producing Unix online help therefore requires little more than taking an existing \texttt{man} page file, and using a text editor to put in whatever you require.
Spell Checkers

• *Spell checkers* are a special type of filter program that scan through a text file and compare every word in the file against a *dictionary* file.

• If a word does not match any word in the dictionary, the spell checker will either write it to a file, or prompt you to fix it on the screen.

• The most commonly used *spell checker* on Linux and other Unix operating systems is *ispell*.

• A good spell checker will “suggest” the proper spelling, or ask the user to type in his or her preferred spelling.

• Most *spell checkers* allow you to add words to the dictionary.
Demonstrations

- The vi text editor.
- The emacs text editor.
- The man online help tool.
- The info online help tool.
- The ispell spell checker.
Revision Control
Why Revision Management?

- Consider a situation where three programmers are each asked to contribute to a document describing the software they are developing.

- The project manager has started writing the document, and has produced the first chapter and the introductions to the chapters the programmers are to write (he is a very helpful manager).

- Programmer A takes a copy of the draft manual and starts to edit it. Programmer B takes a copy of the draft manual and starts to edit it. Programmer C takes a copy of the draft manual and starts to edit it ....

- What happens next?
An Ideal World ...

- All three programmers sit down together and agree upon which parts of the document will contain what, and who will write each part.
- All three programmers then compose the sections they are to write.
- All three programmers then take turns at incorporating the sections they have produced into the document.
- All three programmers then proofread the document to make sure it is correct.
- The document is finished on time and is free of errors.
Reality ...

- Programmer A is debugging a critical piece of code and can’t attend any meetings or work on the document for at least a week.
- Programmer B has time because the code module he has written is being beta tested, so he can start work on the document.
- The manager returns a week later and wants the format of the document changed. He has also changed his copy of the draft document and wants the changes incorporated. He puts his new copy of doco.txt in the work directory.
- A vital customer wants to read the first draft ASAP (As Soon As Possible).
- Programmer C starts editing the original draft document, between code compiles.
Reality ... Take II

- Programmer B finishes his edits and puts the edited file *doco.txt* into the common work directory, overwriting the original copy.

- Programmer C takes the copy of *doco.txt* produced by Programmer B and incorporates his changes, putting his copy into the common work directory.

- The manager returns a three days later and complains to Programmer A that he wants the writing finished.

- The vital customer gets a copy of the incomplete document.

- Programmer A asks Programmer B for his copy of the document and puts in his contribution.
Results?

- The manager’s second set of edits to the file got overwritten by Programmer B when he put his copy back into the work directory.
- When Programmer A is finished he puts his edited copy into the work directory and overwrites the changes made by Programmer C.
- The manager returns two days later and finds that his edits have vanished mysteriously. He then decides that Programmer C must be lazy since his contribution is missing and gives Programmer C a nasty lecture about his future job prospects.
- The vital customer concludes the organisation is incompetent and takes his business elsewhere.
- The manager and Programmers A, B and C all get retrenched when the company goes broke.
Observations

- If multiple workers are to edit a single file, or group of files, a mechanism must be used to prevent inadvertent overwriting of earlier work.

- This is a problem in every activity where multiple parties must edit shared documents, or source code. This is true of engineering drawings, documentation and software.

- The larger the project is, the more important it becomes to coordinate such work. Otherwise, the odds of serious problems developing become very high.

- The easier it is to alter a document, the greater the risk of problems.

- “Revision Control” is the means via which such coordination is produced.
Revision Control Technique

- The principles of revision control have been a part of established engineering disciplines since early in the last century.
- Revision control requires that an organisation keeps track of each and every change made to a document.
- Every document must have a “revision number”, “version number” or “revision level” - a label which is used to keep track of what changes have been made to it.
- The simplest method of document revision control is to maintain “master copies” in a segregated area called a “repository” or “master file”.
- **There can only ever be one master copy of a document at a given revision level. Any more defeats the purpose of revision control!**
Checking Documents In and Out

• Every time you need to make a change to the document, you duplicate the copy of the document with the highest revision number, modify that document as needed, then increment its revision number and put it back.

• The process of taking a copy of the master out of a repository is termed “checking out a document”. The process of putting a modified version back in the repository with a new revision number is termed “checking in a document”.

• To check a master document in, you need to log your name, the date, the revision level, and where possible a note describing the changes you made.
Coordinating Changes

• If somebody else wanted to modify the same document, they then knew that you had a copy which you were working on.

• When you completed your changes, before you could check in your changed document you would check to see if the other person had since checked in their copy.

• If yes, then you were obliged to incorporate his or her changes into your document before you could check it in, or use the revision he or she checked in to make your changes upon.

• In this fashion it was possible to keep track of many changes to a document, by many people, over the many years during which the document remained in use.
Revision Histories

• By comparing the consecutive revision entries made whenever a new revision level is checked in, you can observe the “revision history” or “modification history” of a document.

• This is very important since you can track changes through the “life cycle” of a product, which may be several years. If somebody makes a mistake and changes the document wrongly, you know when this happened, who did it and why it was done.

• This is especially important when attempting to isolate bugs in programs or documents, since bugs may slip through testing and may not be discovered until much later, when the product is in use.

• Bugs frequently result from modifications made by new staff who do not understand the product well.
Major and Minor Revisions

• A convention which is commonly used in industry is the use of “major” and “minor” revision numbers.

• A trivial scheme for revision numbering would start at ‘1’, and every time a change is made, the number would be incremented, e.g. ‘2’, ‘3’, ‘4’ ... ‘N’.

• This tells us nothing about the scope or scale of the change made.

• A better arrangement is to use a revision numbering scheme where two different labels are used, one for major changes and one for minor changes.

• An example is N.M, where ‘N’ is the major revision, and ‘M’ the minor revision. E.g. 6.3 is the third minor change to the sixth major change in the document.
Variants and Revision Branches

- A situation which frequently arises is where a special modification is needed, for instance a customer wants a special feature added and pays for it.
- Therefore you are producing two different ‘variants’ of the same product.
- This results in fork in the revision history, also known as a “revision branch”.
- If the version before the variant was produced has the revision level 2.4, then subsequent versions for that customer are 2.4.1, 2.4.2 and so on. The revisions to the basic product then become 2.5, 2.6 etc.
Consistency

- If a product is made up of many components, such as a software tool made up of several run time files and several manuals, confusion may arise if the revision numbering is not consistent.

- Consider a situation where three files and three manuals each have different revision numbers, but need to be used together. Every time you wanted to use them, you would have to check to see that you have the proper mix of components.

- Since this is very cumbersome, a common industry practice is to increment the revision levels of every component in that set, whenever any part is changed.

- Therefore, a user simply needs to make sure that all components have the same revision number to use them together safely.
Software Tools

• Revision management is considered to be a ‘do or die’ issue in software engineering. As a result, a wide range of software tools have been developed to facilitate the task.

• Many of these tools can be usefully employed for the revision management of documentation, especially text based documentation.

• The most widely used tool was the AT&T Source Code Control System (SCCS).

• The preferred tool today is Walter Tichy’s Revision Control System (RCS), available as free software. RCS is standard now on many Unix types, including Linux.

• RCS allows a user to embed revision control information into a document or source file, in “human readable form”.
RCS Commands

- `ci` is the command used to check in a revision.
- `ci -l` is the command used to check in a revision, check it out and lock it so another user cannot modify it.
- `co, co -l` are the commands used to check out a revision, optionally locking it.
- `rcsdiff` is the command used to produce a list of differences between two different revisions.
- `rlog` is the command used to produce a list of log messages put in during previous `ci` commands.
Using RCS

- **$Id$:** $ is the identification string you can put into a file, RCS will automatically update it whenever you use the `ci` command.

- **Example:** $Id: CSE-1402-T.tex,v 1.3 2000/08/05 16:26:08 carlo Exp carlo $, containing filename, revision-number, date, time, author, state, locker.

- **$Log$:** $ is the log message string you can put into a file, RCS will automatically update it whenever you use the `ci` command.

- It is customary to produce a subdirectory called `RCS` in your working directory. When RCS is run it uses this directory to store the master copies and revision management information.

- Walter Tichy’s paper about RCS is available in the Prac #3 directory.
Typesetters and Desktop Publishing
Tools for Electronic Publishing

- Text based documentation tools such as nroff and GNU info are useful for online documentation of software, but are not well suited to the production of hard copy materials, such as manuals and books.

- Hard copy materials are best presented with high quality typesetting, using proper fonts and graphics where needed.

- This allows the documentation writer to control the format of the document very precisely, using bold fonts, italic fonts, font size changes, paragraph formatting and layout to suitably convey a message to the reader.
Tools for Electronic Publishing ...

• Most “end users” (readers) have the expectation that hardcopy documentation should look good and be very easy to read and navigate. If they are paying thousands or hundreds of thousands of dollars for a software product, this is a reasonable expectation.

• Increasingly, “end users” also have the expectation that online documentation or manuals will be of similar quality in presentation to printed hard copy documents.

• Therefore, if we intend to produce professional quality documentation, we must use tools capable of producing a similar quality output to that traditionally produced by the typesetting industry.

• What capabilities should such a tool have?
Publishing Tools - Capabilities

We can list several capabilities which are considered essential, or very important:

1. The capability to produce output file formats which can be used by laser printers, photo-typesetting machines, or are compatible with software tools which drive laser printers or photo-typesetting machines. Much of the printing trade uses Adobe tools and thus the ability to produce high quality *PostScript* or *PDF* is essential.

2. The capability to use a very wide range of font types and font sizes, ideally arbitrary font types and font sizes.

3. The capability to format the document in the desired manner, providing intended paragraph layouts, page layouts, section layouts, chapter layouts, and other features such as tables of contents, indices, headers, footers, footnotes, tables, and glossaries.
Publishing Tools - Capabilities ...

4. The capability to easily cross reference or reference a document.

5. The capability to incorporate graphics in widely used or ideally arbitrary vector and bitmap graphics formats, without loss of image quality or colour distortion.

6. The capability to run on a wide range of operating systems and hardware platforms, or to produce file formats which can be used by other vendors’ publishing tools.

7. The capability to be useful with a modest amount of personnel training effort in the use of the tool. Do not confuse tool-specific skills with a general understanding of publishing tools and standards. You may be very adept at using tool xyz, yet still not understand the fundamental ideas in hardcopy publishing.
Publishing Tools - Types

There are three general categories or types of tool available today for the production of printed hardcopy documentation:

**Word Processors:** A Word Processor is intended for use by office staff, who need to produce letters, memoranda, notes, forms, and in some instances, simple manuals or reports. Word Processors are usually intended to produce output formats compatible with laser or inkjet printers.

**Typesetters and Desktop Publishing Systems:** Typesetters are intended to produce production quality output files suitable for printing by a press, including if needed colour separations for printing with registration marks. They usually provide large font libraries and powerful filters for the production of quality graphics. A DTP system is usually a typesetter with a GUI.
Word Processors

- Word Processors are usually intended to be easy to use by beginners, and frequently provide extensive template libraries and automated tools.

- Word Processors frequently incorporate many DTP features to make them more attractive in the marketplace, even if these features do not work well.

- Word Processors are usually very cheap to buy and install.

- Word Processors tend to have weak facilities for scientific, mathematical or engineering typesetting, weak facilities for handling books or large reports and usually have very basic graphics filters.

- Word Processors should only be used for small documents where production quality is not an issue.
Typesetters and DTP Systems

- TS/DTP systems are usually difficult for beginners, since production quality is the main design objective in the tool. Such tools assume the user does know what they are doing.

- TS/DTP systems can often be extremely expensive to buy and may be demanding in machine performance (speed/memory) if they are to run fast.

- TS/DTP tend to have very powerful facilities for scientific, mathematical or engineering typesetting, very powerful facilities for handling books or large reports and usually have very capable graphics filters.

- TS/DTP should be used for large or complex documents, or documents where graphics quality is important.
Examples

- \LaTeX{} and \TeX{} are typesetters, but with the LyX GUI become DTP tools.

- Adobe (Frame) \textit{Framemaker} and \textit{PageMaker} are typical modern DTP systems, used widely for publishing books and magazines.

- \textit{Microsoft Word} and \textit{StarOffice StarWriter} are typical word processors for general office use and lightweight publishing.
WWW Publishing

- The most commonly used file formats on the World Wide Web are HTML and PDF, the latter for documents intended to be printed.

- Most modern Word Processors and TS/DTP tools now incorporate HTML filters to produce web pages directly, and many also incorporate PDF filters.

- Word Processors and TS/DTP tools should not be confused with dedicated WWW publishing tools. The latter cannot produce the type of output required for production printing, although they may provide for printing drafts.

- Care should be taken especially with the use of Word Processors for WWW documents, since they frequently damage the graphics during the conversion.
WYSIWYG vs Markup Language

- Most conventional typesetting tools use a *markup language* through which the user tells the system how to lay out the document. The markup language has a specific syntax, and usually facilities for using powerful macro commands or libraries.

- The user will employ a text editor to add *markup language* commands to the text contained in the document. The typesetting tool will then interpret or compile these to produce the intended format. The output is then checked with a *viewer* tool or a laser printer run.

- WYSIWYG (What You See Is What You Get) is a term used to describe modern GUI based tools for DTP and Word Processing. The basic idea is that the user sees on his or her screen an exact rendering of what the final document looks like.
WYSIWYG vs Markup Language ...

• WYSIWYG tools may frequently not render the document exactly, and therefore will require that the user perform multiple laser printer runs to verify that the output is as intended.

• WYSIWYG tools are intended to hide the complexity of formatting the document from the user. Providing that the tool works as advertised, this is a nice feature to have because it allows beginners and less experienced users to produce work of acceptable quality.

• If a WYSIWYG tool misbehaves, the problem can be extremely difficult to debug since the formatting information is buried inside a proprietary binary file format.

• Users of WYSIWYG and markup based tools frequently argue over which is “better”. The reality is that both have their uses.
PostScript and PDF
References:

What is PostScript?

- PostScript is today without doubt the most widely used standard for rendering text and images with high quality, and is preferred by most print houses.

- PostScript is used mostly for laser printers and high quality photoplotters, but is also used for some screen displays.

- PostScript is an interpretive programming language which has been designed to describe text, graphical shapes and bitmap images.

- The GSView, Ghostview and GV PostScript viewers are display tools for use with the GhostScript interpreter, and are designed for viewing PostScript documents.

- Most word processors, DTP tools and typesetters can generate PostScript format output for quality printing.
PostScript Capabilities

- It can describe arbitrary shapes made of straight lines, arcs, rectangles and cubic curves.
- It provides painting operators which can outline or fill shapes with any colour. Colours may be described in RGB, CMYK and CIE schemes.
- It provides full integration of text with graphics. Text characters and fonts may be manipulated like any other graphics primitives.
- It can handle bitmap images such as photographs with arbitrary resolutions, using a range of colour models.
- It provides a coordinate model allowing linear transformations, including translation, scaling, rotation, reflection and skewing.
How Does PostScript Work?

• The PostScript language is text based. It is an interpreted, stack oriented language, similar to FORTH.

• A PostScript device, such as a printer or viewing tool, will execute a PostScript interpreter program. It interprets PostScript language commands and then executes them, rendering the image described on the device.

• Rendering is performed by “scan conversion”. The page or screen in divided into picture elements (pixels), the PostScript interpreter will render each pixel according to the PostScript language description it is given.

• Scan conversion builds up a page by rendering pixels, line by line.
How Does PostScript Work ... 

• Programs such as word processors, DTP and typesetting packages will convert their internal representation of what is depicted on a page into PostScript language.

• The quality of this conversion can vary widely between products, from poor to excellent. It pays to compare the quality of PostScript generator tools.

• We can judge the quality of conversion by looking at a PostScript file which has been created, or testing it on a printer or viewer.

• The quality of a PostScript device (e.g. printer) is determined by the implementation of the language interpreter, and “resolution” in dots per inch (DPI).
Example (Ghostview)

Type the command `gs -sDEVICE=x11`. Then enter:

```
/FSD {findfont exch scalefont def} bind def
/SMS {setfont moveto show} bind def
/F1 10 /Helvetica FSD
(Hello World) 20 20 F1 SMS
/Helvetica-Bold findfont 48 scalefont setfont
20 40 moveto
.5 setgray
(Hello World ) show
/Helvetica-Bold findfont 72 scalefont setfont
20 100 moveto
.1 setgray
(Hello World Big) show
```
Recognising PostScript Files

• The established convention is to use the .ps suffix to identify a PostScript file. Unix tools usually obey this convention when they create a PostScript file.

• Some proprietary tools ignore this convention and use their own suffix type. Therefore you may get PostScript files which are not named as such.

• Every PostScript file can be identified by its header, which contains details describing the file.

• The format is: %!PS–Adobe–2.0, where the number –2.0 is the PostScript version number 1.0, 2.0 or 3.0.

• The Unix file command parses this header.
Example PostScript File Header

%!PS-Adobe-2.0
%%Title: bloggs.ps
%%Creator: fig2dev Version 3.2.3 Patchlevel
%%CreationDate: Sun Aug 6 21:33:53 2000
%%For: carlo@raptor.cs.monash.edu.au (Carlo Kopp)
%%Orientation: Portrait
%%BoundingBox: 0 0 105 148
%%Magnification: 1.0000
%%EndComments
Encapsulated PostScript

- Encapsulated PostScript (EPS) is a variant of PostScript which is used for embedding pictures in documents.

- Standard PostScript files are usually sized to standard formats such as A5, A4, US Letter. EPS files have arbitrary sizes.

- The size of the EPS image is determined by the “BoundingBox” parameter.

- The format of the EPS identification string is: `%!PS-Adobe-2.0 EPSF-2.0`, where the number `-2.0` is the PostScript version number 1.0, 2.0 or 3.0, and EPSF-2.0 identifies it as EPSF.

- Most drawing and image manipulation tools can produce EPS format output files.
What is Portable Document Format?

- Portable Document Format (PDF) is a closely related standard to PostScript, also devised by Adobe.
- While PostScript was devised for printing, and is verbose and slow, PDF was designed for viewing on a wide range of platforms, and is very compact.
- PDF uses a similar graphics representation model to PostScript, and can be easily translated into PostScript for printing. Many PostScript viewing tools can also display PDF, e.g. Ghostview, GSView and GV. Most PDF viewers, such as Acroread and xpdf, cannot interpret PostScript files.
- PDF incorporates hypertext features, annotation features and digital signature facilities. It also includes page transition facilities.
How is PDF Created?

- A conversion tool must either interpret PostScript and translate it into PDF, or translate its internal representation into PDF.

- For instance PDF can be produced by an application such as a DTP package. An example is the PDF\LaTeX system, which translates \LaTeX macro source directly into PDF.

- An alternative is to use the Adobe *Distiller* or public domain *ps2pdf* (Ghostview) tools to convert a PostScript file to PDF. The Adobe *PDFWriter* tool allows word processors such as *MS Word* to generate PDF output.

- The intent of the designers of PDF was to produce a standard which is portable across a wide range of operating systems, and viewable with a wide range of tools.
PDF Features

• While PDF uses the same graphics representation model as PostScript, it does not support PostScript program constructs. Therefore a PDF file is much simpler.

• Unlike PostScript, PDF incorporates compression techniques. Colour and grayscale pictures are compressed using the JPEG technique. Monochrome (pure black and white) images are compressed using CCITT Group 3 and 4 (fax), LZW and RLE techniques. Text and graphics are compressed using LZW and Flate techniques.

• You can easily recognise the difference in quality between PDF conversion tools by looking at the size of the PDF file produced, the smaller, the better.

• PDF files can include encryption, to restrict access.
PDF Features ...

- PDF includes a *font substitution* mechanism. If the font used in a document is not installed on the computer running the viewing tool, a generic font is substituted.

- PDF files include an internal cross reference table, allowing a page to be quickly found. The ability to randomly access parts of a PDF file is a requirement of the PDF specification.

- PDF files are binary and cannot be treated as ASCII text.

- The first line in a PDF file contains an ASCII text identification header in the format: `%PDF-1.3`. The number refers to the version of PDF used. Some versions of the Unix `file` command may be confused by this.
PDF File Structure

- A PDF file can be typically divided into four portions.
- The *header* `%PDF-1.3` identifies the file type.
- The *body* contains objects, such as text, images and fonts.
- The *cross reference table xref* contains information which allows random lookup of objects in the file.
- The *trailer* contains information needed to find the xref table, and also encryption information (MD5, RC4).
Markup Languages:

GML, SGML, HTML, XHTML

\LaTeX & \text{TEx}
References

- *W3C Technical Reports and Publications*: http://www.w3.org/TR/
- *Charles F. Goldfarb’s SGML SOURCE HOME PAGE*: http://www.sgmlsource.com/
What is a Markup Language?

- Historically, the term ‘markup’ referred to comments by copy editors on a manuscript, intended to be read by people performing typesetting.

- Markup of the form ‘Make this heading in 14 point bold typeface, set the body to 12 point typeface, use a Helvetica font’ were used by typesetting personnel to select the intended typeface, size and layout of a manuscript.

- With the advent of computers, publishers adopted formal ‘markup languages’ to describe such layout and formatting information in a manner which could be understood by typesetting software tools.
What is a Markup Language?

• A *markup language is thus a combination of keywords and grammar which can be interpreted by a typesetting or rendering software tool.* The particular commands will determine the layout and presentation of the document.

• A markup language is typically specific to a tool or family of tools written around the language.

• A well structured markup language can be translated into a different markup language.

• *Markup* is distinct from *content*. Two different documents can be marked up using the same markup language, both will have a similar layout, but they are still unique documents.
Content vs Form

- The *content* of a document is the text which is unique to that document. For instance, a copy of Dante’s *Purgatory* will have the same content regardless of the manner and format in which it is published.

- The *form* of a document is the manner in which it is presented, including the choices made in typeface, layout, and general appearance (i.e. markup). If we typeset all three volumes of Dante’s *Divine Comedy* using the same layout form we have three different titles each with unique content, yet all have the same appearance.

- Most publishers, computer manufacturers and software houses use their own, highly specific styles, designed to identify their products. An IBM manual will look similar to another IBM manual, but different from an SGI manual.
Generic vs Layout Markup

- A markup language which describes in detail the specifics of the typeface and layout on a page is called a ‘specific or layout markup language’. Such a markup language requires that the user understand the language syntax in detail, as well as understanding type-setting practices.

- The disadvantage of layout markup is that it is difficult to use, and it is often difficult to translate by machine.

- The alternative strategy is the use of ‘generic markup’, which allows the user to specify high level formatting commands, leaving the details of the particular document layout to the software. Such high level commands might be the markup of a chapter, section or bullet list.
Generic vs Specific Markup...

- Generic markup is easy to use, and relies on ‘prepackaged’ layout rules for a particular class or style of document.

- A well designed generic markup language will separate the information which controls the layout style of the document from the document itself. This ‘style’ information is usually held in a separate file termed a ‘style file’ or ‘style sheet’.

- If the user has access to a good collection of ‘style files’ he or she can easily produce a large number of documents which share a common layout style.

- A well designed generic markup language will allow the user to change the appearance of the whole document, simply by changing the ‘style file’ being used.
Markup Language Families

- At this time there are two major families of markup languages in wide use, across industry and academia.
- The SGML family of markup languages, including HTML, XHTML and now XML.
- The \TeX family of markup languages, including many variants of \LaTeX.
- Conversion tools exist to translate \LaTeX documents into other markup languages, such as HTML and XML. Therefore \LaTeX is a very flexible markup language, since it can be used to generate PostScript for printing, and PDF, HTML and XML for web publishing.
- At this time, \LaTeX is the best choice available for document layout, if we intend to present the document in different ways.
SGML Derivatives

- During the 1960s the Graphic Communications Association developed a markup language called GenCode, while IBM developed a proprietary markup language called Generalised Markup Language (GML).

- Aspects of both languages were combined to produce the Standard Generalised Markup Language (SGML), which was standardised in 1986 by the ISO as ISO:8879, 1986.

- SGML is a very powerful but also complex markup language which is not widely used in its basic form.

- It has the ability to specify rules via which other markup languages can be defined. It is frequently described as a ‘meta-language’.
SGML Derivatives ...

- HyperText Markup Language (HTML) is a subset of SGML designed for use on the web. The current version is HTML 4.01.

- eXtensible Markup Language (XML) is another subset of SGML, intended to replace HTML on the web in coming years.

- eXtensible HyperText Markup Language (XHTML) is a definition of HTML in XML, intended for use during the transition between HTML and XML.

- HTML 4.0, XML and XHTML all support the use of ‘style sheets’ to separate the form from the content of the document, whereas early HTML required layout markup.

- Style sheets are described in the Cascading Style Sheets (CSS) specification.
SGML Derivatives ...

- Early HTML was thus a layout markup language, whereas late HTML, XHTML and XML can be used as generic markup languages by exploiting CSS.

- Mathematical Markup Language (MathML) is based on XML and is intended for typesetting mathematical notation.

- Scalable Vector Graphics (SVG) is also based on XML, it is intended to display vector graphics in a manner not unlike PostScript does.

- Future web browsers will render XML, MathML and SVG, using CSS to precisely control the layout and presentation of documents.

- Until XML, MathML and SVG become widely used, PDF will remain the best format for distributing documents intended for viewing and printing.
TEX

- **TEX** was devised during the 1970s by Prof Donald Knuth (pronounced Ka-nooth) as a language for typesetting books containing mathematics and scientific notation, both considered difficult to typeset by conventional means.

- **TEX** is a layout markup language, widely considered difficult to use, complex, but also very powerful.

- The complexity of **TEX** means that very few people are competent in the language.

- **TEX** has been ported to most operating systems in use today.

- The preferred way in which **TEX** is used today is as the ‘back end’ to the **LATEX** generic markup language.

- This subject will be mostly concerned with the use of **LATEX**.
\textbf{\LaTeX}

- \LaTeX was devised during the 1980s by Leslie Lamport as a ‘front end’ language intended to simplify the use of \TeX.

- \LaTeX is a generic markup language, which is designed to provide a simple ‘user-friendly’ markup language for document layout.

- \LaTeX provides simple commands to structure chapters, sections, indices, bibliographies, cross references, tables of contents.

- \LaTeX provides means for including both bitmap and vector graphics.

- \LaTeX provides a very powerful mathematical typesetting language.

- \LaTeX employs a powerful ‘style file’ mechanism, using \texttt{.sty} suffix files containing layout information. Changing the \texttt{.sty} file allows the user to globally change the layout and presentation of a document.
**\LaTeX** Output

- \LaTeX{} can be directly compiled into *Device Independent* or .dvi format using the basic \texttt{latex} compiler.

- \LaTeX{} can be directly compiled into *Portable Document Format* or .pdf format using the \texttt{pdflatex} compiler.

- \LaTeX{} .dvi format can be translated into *PostScript* using the \texttt{dvips} tool.

- \LaTeX{} can be translated into *HTML* using the \TeX4ht tool, \texttt{tex4ht}, or the \LaTeX2HTML tool, \texttt{latex2html}.

- \LaTeX{} can be translated into *MathML/XML* using the \TeX4ht tool, \texttt{tex4ht}, or the \Omega tool, \texttt{omega}.
Introductory \LaTeX
Using \textsc{latex}

- \textsc{latex} is designed for the production of various basic document types, ranging from letters through to books and reports.

- The current version of \textsc{ latex} is \textsc{latex} 2e, most compilers will however safely handle older \textsc{latex} 2.09 format source files.

- \textsc{latex} systems comprise a \textsc{latex} compiler, such as latex, pdflatex or htlatex, and a large library of “packages”.

- The basic syntax of \textsc{latex} is very simple, however \textsc{latex} documents can become very complex if a user chooses to employ various additional package features.

- The aim of this subject will be to provide basic skills in \textsc{latex}. 
\textbf{\LaTeX\ Classes and Packages}

- The layout and basic structure of a \LaTeX\ document is determined by the document “class” which is used.

- The most commonly used classes for software documentation are \texttt{book} and \texttt{article}.

- Additional document layout features can be introduced by using one or more “packages”. Packages are used for instance to incorporate artwork.

- In addition, a user can further manipulate layout using commands to change font types and presentation.

- \LaTeX\ is also integrated with the \texttt{BibTeX} tool, which is a powerful facility for handling references and bibliographies.
Document Class Syntax

- A user will set the document class using the `{documentclass}` command at the beginning of a \LaTeX file.

- `{documentclass}` is always invoked with an argument, e.g. `{documentclass{book}}` or `{documentclass{article}}`.

- In addition, `{documentclass}` can be given additional arguments or qualifiers. Arguments or qualifiers are always placed in square brackets, i.e. `{documentclass[options]{class}}`.

- An important option for `{documentclass}` is `a4paper`, which forces the layout to an A4 paper size.

- Because `{documentclass}` is used by the compiler to set the basic layout of the document, it must always be placed at the beginning of the document.
Package Syntax

- Specific packages are called by including the `\usepackage{}` command at the beginning of a \LaTeX file.

- `\usepackage{}` is always invoked with an argument, e.g. `\usepackage{multicol}` or `\usepackage{epsfig}`.

- In addition, `\usepackage{}` can be given additional arguments or qualifiers, if the package supports these. Arguments or qualifiers are always placed in square brackets, i.e. `\usepackage[german]{babel}`.

- \LaTeX packages provide a wide range of features and facilities in addition to the basic \LaTeX macro library. They cause a `.sty` file to be loaded.
Document Preambles

• A document preamble is a group of \LaTeX\ commands which are entered at the beginning of a document.

• The purpose of the preamble is to “set up” the \LaTeX\ environment for the document.

• It is important to note that commands in the preamble set parameters for the whole layout of the document.
Document Preamble Example:

\documentclass[a4paper]{article}
\usepackage{epsfig}
\setlength{\hsize}{16cm}
\setlength{\textwidth}{\hsize}
\setlength{\vsize}{22cm}
\setlength{\textheight}{\vsize}
\setlength{\parskip}{5mm}
\setlength{\parindent}{0cm}
\setlength{\hoffset}{-2cm}
\setlength{\voffset}{-1cm}
Document Structure

- The basic structure of a \LaTeX\ document is a preamble, followed by the document proper, which is enclosed between a `{\begin{document}}` and a `{\end{document}}` command.

- The `{\begin{}}` and `{\end{}}` commands are in concept similar to `begin` and `end` constructs in \textit{Pascal}, or “curly braces” in \textit{C}.

- The \LaTeX\ compiler first loads the preamble commands, then processes the contents of the document between the `{\begin{}}` and `{\end{document}}` commands.
Document Structure Example:

\documentclass[a4paper]{article}
\usepackage{epsfig}
\begin{document}

This is a trivial example \LaTeX{} document.

It has only two lines and doesn’t use the included package.

\end{document}

This is a trivial example \LaTeX{} document.
It has only two lines and doesn’t use the included package.
The Article Class

- Small reports, papers, journal articles and similar documents are easily produced using the “article” document class.

- A document formatted using this class requires that the user master several basic \LaTeX commands.

- These are \texttt{title{}}, \texttt{author{}}, \texttt{abstract{}}, \texttt{maketitle} and \texttt{section{}}.

- A document may or may not include all of these commands.

- The basic structure of a preamble and document content is retained.
Example Trivial Article:
\documentclass[a4paper]{article} % Preamble
\begin{document} %Document begins
\title{My Title} %Title
\author{Jane Doe, etc} %Author details
\maketitle %Produce title block
\begin{abstract} %Start of abstract
This is a trivial abstract.
\end{abstract} %End of abstract
\section{Introduction} %Sections
\section{First Section}
\section{Second Section}
\section{Conclusions}
\end{document} %Document ends
Example Title Block:

\title{\Large\bf{My Document Title}}
\author{Jane Doe, BSc(Hons), MComp,\
Monash University, Clayton, Australia\
email: {\tt jane.doe@csse.monash.edu.au}\
{\bf Computer Science \& Software Engineering}\
{\bf http://www.csse.monash.edu.au/}\
{\bf\copyright ~2004, Monash University, Australia}\
\maketitle

NB the use of layout formatting commands \Large, \bf, \tt and the special character \copyright.
Sectioning Commands

- Sectioning commands are used to produce numbered headings for portions of a document.

- In an article class document, a simple hierarchy of \section commands are used.

- \LaTeX{} will automatically number these according to their position. The text inside the argument field (curly braces) becomes the text of the heading.

- Sectioning commands are one of the most powerful features of \LaTeX{}, yet are very simple to use.

- Judicious use of sectioning commands can much improve the clarity of a document.
Sectioning Example:

\section{A Section}
\subsection{A Subsection}
\subsubsection{A Subsubsection}
A Section

A Subsection

A Subsubsection
Other Sectioning Commands

- \LaTeX\ provides additional sectioning commands for the book and report document classes.
- The \texttt{\part} command is used to divide the document into parts. This command is also available for \texttt{article} class documents.
- The \texttt{\chapter} command is used to divide the document into chapters.
- The \texttt{\section}, \texttt{\subsection} and \texttt{\subsubsection} commands are the same as in the \texttt{article} class.
- The \texttt{\paragraph} command is used to divide the document into paragraphs.
- The \texttt{\subparagraph} command is used to divide the document into subparagraphs.
Another Sectioning Example:

\section{A Section}
  \subsection{A Subsection}
  \subsubsection{A Subsubsection}
  \paragraph{A Paragraph}
  \subparagraph{A Subparagraph}
A Section

A Subsection

A Subsubsection

A Paragraph

A Subparagraph
\LaTeX\ Layout Markup Commands
Layout Markup Commands

- Layout markup commands are used to manipulate details of a document layout, rather than changes to global layout.

- Layout markup commands should be used carefully, because they bypass the global setup of the document layout and style.

- When using layout markup commands, it is important to always be consistent in their use.

- An example of consistent use is where every caption to a picture uses the same font type, font weight, font size and is set to italic or bold.

- An example of inconsistent use is where captions to pictures vary widely in the font type, font weight and font size, with some in italic or bold.
Manipulating Font Size

- Sometimes we need to use a larger or smaller font to attract the attention of the reader.

- One technique is to change the size of the font. This is very commonly used for more complex header or title pages. Font sizes are measured in ‘points’ or ‘pt’, there are 72.27 pt/inch or 2.85 pt/mm.

- \TeX uses a very simple method - a ‘size change’ command, which is applied to the following text which is enclosed in curly braces.

- The basic syntax is \{\texttt{\string size\_change\_cmd text\_to\_be\_changed} \}.

- \TeX provides ten basic font size commands, which are scaled to match the standard font size in the document, e.g. 12 pt.
Font Size Commands

{\tiny Font Size}
{\scriptsize Font Size}
{\footnotesize Font Size}
{\small Font Size}
{\normalsize Font Size}
{\large Font Size}
{\Large Font Size}
{\LARGE Font Size}
{\huge Font Size}
{\Huge Font Size}
{\HUGE Font Size}
Font Size Commands ...

Font Size
Font Size
Font Size
Font Size
Font Size
Font Size
Font Size
Font Size
Font Size
Font Size
Manipulating Font Weight

• Most font families provide only one or two weights, typically ‘light’, ‘medium’ and ‘bold’.

• \LaTeX \ uses ‘Computer Modern Roman’ as its standard font, this is a ‘serifed’ font with only ‘medium’ and ‘bold’ weights. Some font families will provide ‘semi light’, ‘light’, ‘extra light’ weights.

• \LaTeX \ provides a simple shorthand to change the font to ‘bold’.

• The basic syntax is \{\texttt{\textbf{text_to_be_changed}} \).

Example:

This chunk of text is part in medium weight and \texttt{\textbf{part in bold}}.

This chunk of text is part in medium weight and \texttt{part in bold}.
Emphasis with Italics

- *Italics* are used to *emphasise* words or whole paragraphs in a document. This is used to focus the reader’s attention to the word or paragraph.

- LaTeX uses the `\emph{}` environment for emphasis.

- LaTeX provides a simple shorthand to set emphasis.

  - The basic syntax is `{\em text_to_be_changed }`.

Example:

This chunk of text is part in medium weight and `{\em part in italic.}`

This chunk of text is part in medium weight and *part in italic*. 
Changing Font Types

- Most documents will use a single font type throughout. Sometimes it is useful to change the font type to draw attention. An example is when we cite command lines or program source in document.

- \LaTeX{} provides three basic font types: ‘Roman’, ‘Sans Serif’ and ‘Typewriter’.

Example:

\begin{verbatim}
\texttt{This text is in Typewriter font.}
\end{verbatim}

This text is in Roman font. This text is in Sans Serif font. This text is in Typewriter font.
Verbatim and {\tt}

- \LaTeX{} provides two convenient environments for citing source code or commands.
- The ‘teletype’ or {\tt} environment is shorthand for the \ttfamily command.
- The \verb|verbatim| environment ignores all \LaTeX{} commands between \verb|\begin{verbatim}| and \verb|\end{verbatim}|.

Example:
\begin{verbatim}
This text is cited verbatim, and includes {\tt Typewriter Font}.
\end{verbatim}
This text is cited verbatim, and includes Typewriter Font.
Footnotes

- Footnotes are frequently used in documents to explain details which might clutter the text, clarify points, or list references.

- \LaTeX\ uses the \texttt{\footnote{}} environment for footnoting. Footnotes are numbered automatically, and should be put at the end of a sentence or paragraph.

- The basic syntax is \texttt{\footnote{footnote\_text}}.

Example:

This sentence has a footnote\footnote{Which contains this trivial comment.}.

This sentence has a footnote\footnote{ Which contains this trivial comment.}.

\footnote{Which contains this trivial comment.}
Quotes

- Quotes are widely used in printed text. In documentation they are most commonly used for titles of papers, texts and other documents.

- In printed documents and books, it is customary to always use paired quotes, whether single or double.

- \LaTeX{} uses a very simple arrangement for single and paired quotes.

Example:

This is a ‘single paired quote’.
This is a ‘double paired quote’.

This is a ‘single paired quote’.
This is a “double paired quote”.
\texttt{noindent} and \texttt{\newpage}

- Default \LaTeX{} styles typically indent the first line in a paragraph.
- Sometimes this is not convenient, especially if the paragraph comprises less than a line, and is followed by a picture or table.
- The \texttt{\noindent\{\}} environment stops indenting for the beginning of the first paragraph which is enclosed.
- Frequently we might wish to have a section start on the beginning of a new page.
- \LaTeX{} provides a command to force the following text to start at the beginning of a new page.
- This is done using the \texttt{\newpage} command.
Example: \noindent

\noindent{Example:}\
\noindent{This is a paragraph without indenting.}

This is a paragraph with indenting.}

Example:
This is a paragraph without indenting.
   This is a paragraph with indenting.
Lists

• \LaTeX provides some very powerful environments for producing lists.
• A ‘bullet’ list can be produced using the itemize environment.
• A ‘numbered’ list can be produced using the enumerate environment.
• An ‘description’ list can be produced using the description environment.
• Each entry in the list must be preceded by a \texttt{item} command.
• All items between the \texttt{begin} and \texttt{end} delimiters are formatted into the list.
Example Bullet List

\begin{itemize}
\item Bullet list item one.
\item Bullet list item two.
\item Bullet list item three.
\end{itemize}

- Bullet list item one.
- Bullet list item two.
- Bullet list item three.
Example Numbered List

\begin{enumerate}
\item Numbered list item one.
\item Numbered list item two.
\item Numbered list item three.
\end{enumerate}

1. Numbered list item one.

2. Numbered list item two.

3. Numbered list item three.
Example Description List #1

\begin{description}
\item[1st Entry:] Entry list item one.
\item[2nd Entry:] Entry list item two.
\item[3rd Entry:] Entry list item three.
\end{description}

1st Entry: Entry list item one.

2nd Entry: Entry list item two.

3rd Entry: Entry list item three.
Example Description List #2

\begin{description}
\item[Open:] This command is used to open a file. When it is used, the program will open a window, into which the user types the name of the file. The file then opened.
\item[Close:] This command is used to close a file. When it is used, the program will close the currently opened file.
\end{description}
Customising Lists

- Sometimes it is convenient, if not necessary, to customise a list environment.

- LaTeX provides very powerful facilities to do this. These are described in any good text.
Tables

• \LaTeX{} provides some very powerful environments for producing tables.
• Table entries can be aligned left, right or centred.
• Tables can be positioned at the top or bottom of the page.
• Horizontal separator lines can be incorporated easily using \texttt{\textbackslash hline}.
• The basic syntax is \texttt{\begin{tabular} [pos] {cols} \end{tabular}}, where \textit{pos} is the position specifier and \textit{cols} is the column format specifier, followed by \texttt{\end{tabular}}.
• The position specifiers are \texttt{t} for the top of the page and \texttt{b} for the bottom of the page.
• The column specifiers are \texttt{l}, \texttt{r}, \texttt{c} for left, right and centre, respectively, entries are separated by \&.
## Table Example #1

\begin{tabular}{|l|l|l|l|} 
\hline 
Name & Title & Signature & Date \ 
\hline 
Jane Doe & Programmer & & Today’s Date \ 
\hline 
My Tutor & Project Leader & & Today’s Date \ 
\hline 
Carlo Kopp & Project Director & & 28/08/2000 \ 
\hline 
\end{tabular}
# Table Example #2

\begin{tabular}{|l|c|l|c|} \hline
Name & Title & Signature & Date \\ \hline
Jane Doe & Programmer & & Today’s Date \\ \hline
My Tutor & Project Leader & & Some Date \\ \hline
Carlo Kopp & Project Director & & 28/08/2000 \\ \hline
\end{tabular}
### Table Example #3

\begin{tabular}{llll}
  \hline
  Name       & Title           & Signature & Date
  \hline
  Jane Doe   & Programmer      & -         & Today’s Date
  My Tutor   & Project Leader & -         & Some Date
  Carlo Kopp & Project Director & -       & 28/08/2000
  \hline
\end{tabular}
# Table Example #4

\begin{tabular}{|l|l|l|l|} 
\hline
Name & Title & Signature & Date \hline
Jane Doe & Programmer & - & Today’s Date \hline
My Tutor & Project Leader & - & Some Date \hline
Carlo Kopp & Project Director & - & 28/08/2000 \hline
\end{tabular}
Floating Tables

- \LaTeX provides facilities for floating tables, which include headers, footers, captions and labels.

- Floating tables can be positioned at the top or bottom of the page.

- The basic syntax is \begin{table} [pos] \{header table \footer/caption label\}, followed by \end{table}.

- The header appears at the top of the floating table.

- The footer appears at the bottom of the floating table.

- The caption appears at the specified position.
Floating Table Example #1

\begin{table}[h] \textbf{Signoff Table Header}\\
\begin{tabular}{|l|l|l|l|} \hline 
Name & Title & Signature & Date \hline
Jane Doe & Programmer & - & Today’s Date \hline
My Tutor & Project Leader & - & Some Date \hline
\end{tabular}\\ \textbf{My Project Footer}\\
\caption{Signoff Table Example} \label{MyTab}\\
\end{table}

\noindent{The example table is found at Table \ref{MyTab}.
# Floating Table Example #1 ...

## Sign-off Table Header

<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane Doe</td>
<td>Programmer</td>
<td>-</td>
<td>Today’s Date</td>
</tr>
<tr>
<td>My Tutor</td>
<td>Project Leader</td>
<td>-</td>
<td>Some Date</td>
</tr>
</tbody>
</table>

## My Project Footer

Table 2: Sign-off Table Example

The example table is found at Table 2.
Labels and References

- \LaTeX{} provides a powerful facility for automatic referencing inside documents.

- Automatic referencing is extremely useful since a user need not worry about the specific numbers attached to tables, pictures or sections. \LaTeX{} generates the numbers automatically.

- The label syntax is \verb|\label{mylabel}|.

- The reference syntax is \verb|\ref{mylabel}|.

- Typically the labels and references require two consecutive compiles before they are properly resolved.

- The preceding example of a floating table includes the use of a label and reference.
Incorporating Pictures

- \LaTeX\ provides powerful and flexible facilities for incorporating pictures into documents.

- If we are using \LaTeX\ directly, and converting DVI files into PostScript, then we can easily incorporate pictures in EPS format.

- If we are using PDF\LaTeX, then we can incorporate pictures in JPEG, PNG and PDF formats. N.B. PDF format pictures can be produced from EPS using the `epstopdf myfile.eps` command, which adjusts the bounding box parameters for PDF\LaTeX.

- Picture environments are usually defined as floating.

- Care should always be taken with pictures since problems with choice of picture format are very quickly noticed by readers.
Picture Example #1:

\usepackage[pdftex]{graphicx} % in the file header
\includegraphics[scale=0.5]{PDF/mecl-or.pdf}

Physical
(DIL, SOIC, SOJ, PLCC)

Logical
(OR/NOR Gate)

Emitter Coupled Logic (ECL) Gate
MECL III/10K/10KH/10E/100E, NS 100F, NEC 600 Series
Gate Delay ~ 0.3–1 ns, Gate Power ~ 25–60 mW

'0' = − 1.75 V
'B' = − 0.95 V
'1' = − 0.95 V

A picture rendered in PDF format.
Picture Example #2:

\usepackage[pdftex]{graphicx} % in the file header
\includegraphics[scale=0.5]{PDF/mecl-or.jpg}

A picture rendered in JPEG format.
Picture Example #3:

\usepackage[pdftex]{graphicx} % in the file header
\includegraphics[scale=0.5]{PDF/mecl-or.png}

A picture rendered in PNG format.
Picture Example #4:

\usepackage{epsfig} % in the file header
\epsfig{file=PDF/mecl-or.eps}

Emitter Coupled Logic (ECL) Gate
MECL III/10K/10KH/10E/100E, NS 100F, NEC 600 Series
Gate Delay ~ 0.3–1 ns, Gate Power ~ 25–60 mW

A picture rendered in EPS format (fudged with PDF).
BibTeX

- The BibTeX tool provides a very powerful and flexible facility for managing references in LaTeX documents.

- BibTeX searches through a myfile.tex LaTeX source file and locates citations identified by the `\cite{mykey}` tag. It then searches the bibliographic database file bib.bib and produces a bibliography file called myfile.bbl.

- When LaTeX is run it uses the bibliography file myfile.bbl to produce the bibliography list in the document.

- The user needs to edit the bib.bib database to add references, and insert appropriate `\cite{mykey}` tags in the document. LaTeX and BibTeX perform the hard work.
The \textit{BibTeX} Database

- The \textit{BibTeX} database contains entries which have specific formats for references such as books, journal papers, technical reports and other types of documents.

- The entry type field determines which other fields the entry should contain.

- \textit{BibTeX} entries in the default format will have their capitalisation changed to fit a default, which is frequently incorrect for many reference types.

- Therefore it is necessary to use a syntactic ploy, and encapsulate each field in the database entry with \{\{ double curly braces\}\} to force \textit{BibTeX} to preserve the original capitalisation.

- It is customary for the key field to be in the format surname:year.
The \textbf{BibTeX} Database Entry

@TechReport{ W3C:99–2,  
author = {{Dave Raggett, et al}},  
title = {{HTML 4.01 Specification}},  
year = {{W3C Recommendation, 24 December 1999}},  
institution = {{W3C HTML Working Group.}},  
The \textsc{BibTeX} Bibliography

\begin{thebibliography}{1}

\bibitem{W3C:99-2}
{Dave Raggett, et al}.
\newblock {HTML 4.01 Specification}.
\newblock {Specification, http://www.w3.org/TR/1999/REC-html401-19991224},
\newblock {W3C HTML Working Group.},
\newblock {W3C Recommendation, 24 December 1999}.

\end{thebibliography}
Example \textbf{BIBTEX} Usage

... candidates are listed in the references under \cite{Pepper:2000}, \cite{Mozilla:2000}, \cite{AGOCG:2000}.

\begin{verbatim}
\bibliographystyle{plain}
\bibliography{bib}
\end{verbatim}

... candidates are listed in the references under [2], [3], [1].
References


Running \texttt{BibTeX}

- To use \texttt{BibTeX}, we must first run \texttt{latex} to produce the auxiliary file \texttt{myfile.aux} from \texttt{myfile.tex}.

- We then run \texttt{BibTeX} using \texttt{bibtex myfile} to produce the bibliography \texttt{myfile.bbl}.

- Finally we must run \texttt{latex} again to resolve the references properly.

```
raptor[carlo]1022% pdflatex CSE-1402-T-B.tex
raptor[carlo]1024% bibtex CSE-1402-T-B
This is BibTeX, Version 0.99c (Web2C 7.3.2x)
The top-level auxiliary file: CSE-1402-T-B.aux
The style file: plain.bst
Database file #1: bib.bib
raptor[carlo]1022% pdflatex CSE-1402-T-B.tex
```
Advanced \LaTeX\ Features
Useful Packages - `hyperref` and `pdfcrypt`

- PDF\LaTeX{} is used to compile \LaTeX{} directly to PDF output, using the \texttt{pdftex} package.
- The \texttt{hyperref} package enhances this function by including internal hyperlinks.
- This permits the direct compilation of PDF output with many advanced PDF functions included.
- Encryption using a 40-bit key can be activated if required, using the \texttt{pdfcrypt} package.
- \texttt{pdfcrypt} also permits control of other security features such as copying or printing.
Syntax Example - \texttt{hyperref}

\texttt{\usepackage[pdftex,}
\texttt{  pdfauthor={Jane Doe},}
\texttt{  pdftitle={My Document Title in Here},}
\texttt{  pdfsubject={My Document Subject in Here},}
\texttt{  pdfkeywords={hyperref, pdftex, pdflatex},}
\texttt{  pdfproducer={pdfTeX},}
\texttt{  pdfcreator={$Id: \$},}
\texttt{  pdfmenubar=true,}
\texttt{  pdfhighlight=/I,}
\texttt{  pdftoolbar=true,}
\texttt{]}\texttt{\{hyperref\}}
Syntax Example - pdfcrypt

\usepackage[owner=Carlo, user=userpassword, edit=false, copy=false, annotate=true, print=true]{pdfcrypt}
Useful Packages - \textit{fancyhdr}

- This package is used to set up attractive headers and footers in a document.

- It permits control of headers, footers and rulers, using user defined strings or internal \LaTeX{} variables such as page or section numbers.

- The hardcopy lecture notes are presented using \textit{fancyhdr}. 
Syntax Example - fancyhdr

\usepackage{fancyhdr}
\pagestyle{fancy}
\renewcommand{\sectionmark}{\markright{\thesection\ #1}}
fancyhf{} % delete current setting for header and footer
fancyhead[LE,RO]{\bfseries\thepage}
\fancyhead[LO]{\bfseries\rightmark}
\fancyhead[RE]{\bfseries\leftmark}
\fancyfoot[C]{\bfseries\copyright 2003, SCSSE}
\renewcommand{\headrulewidth}{0.5pt}
\renewcommand{\footrulewidth}{0pt}
\addtolength{\headheight}{0.5pt} % make space for the rule
\fancypagestyle{plain}{{%
\fancyhead{} % get rid of headers on plain pages
\}}
\renewcommand{\headrulewidth}{0pt} % and the line
Useful Packages - shadow

\usepackage{shadow}
\shabox{\parbox{\linewidth}{
‘Shadow Boxes’ can be used to emphasise a block of text to advantage. They are very useful in reports and manuals.
}}

‘Shadow Boxes’ can be used to emphasise a block of text to advantage. They are very useful in reports and manuals.
Useful Packages - endnotes

• \LaTeX uses the \texttt{\endnote{}} environment for endnoting. Endnotes are numbered automatically, and should be put at the end of a sentence or paragraph like footnotes.

• The basic syntax is \texttt{\endnote{ endnote_text }}.

\usepackage{endnotes}
This sentence has a endnote\endnote{ Which contains this trivial comment.}.
\renewcommand{\notesname}{Document Endnotes}
\begingroup
\setlength{\parindent}{0pt}\setlength{\parskip}{2ex}
\renewcommand{\enotesize}{\normalsize}
\theendnotes \bigskip
\endgroup
Useful Packages - endnotes

This sentence has two endnotes\(^1\) \(^2\).
Document Endnotes

1 Which contain this trivial comment.

2 And which also contain this trivial comment.
Graphics
References:

Graphics

- A wide range of file formats are available for including graphics in documents, be they hardcopy or on-line.

- The quality of artwork can have a strong impact on reader perceptions of the quality of a document, therefore it pays to choose file formats wisely.

- Graphics can be in *bitmap* or *vector* formats. Each have particular applications they are best suited to.

- A major issue in considering the flexibility and quality of DTP and typesetting tools is their support for particular graphics file formats. A limited ability to use vector format graphics is a good indicator of a limited tool or package.
Bitmap Formats

• Bitmap formats represent a picture by dividing it into a regular array of rectangles or squares, each of which is uniquely described by colour, saturation and brightness information. Viewed together, the elements in the array form a picture.

• Each rectangle or square is termed a ‘picture element’ or ‘pixel’.

• Colour, saturation and brightness are usually represented by three or four values, which additively produce the intended result.

• The two most common schemes used are Red-Green-Blue (RGB) or Cyan-Magenta-Yellow-Black (CMYK).

• RGB is used mostly for on-screen rendering, CMYK mostly for printing. PostScript and PDF files can support both schemes.
Bitmap Formats ...

N Columns of Pixels

Each pixel is described by a number, or several numbers, which define its colour hue, purity and brightness.

M Rows of Pixels

Colour
Purity
Brightness
... Pixels

Each pixel is described by a number, or several numbers, which define its colour hue, purity and brightness.
Bitmap Formats ...

N Columns of Pixels

Zooming in shows the individual pixels within the white square. At this resolution the illusion of a picture can no longer be maintained.
Bitmap Image Quality

- The conversion of a real image into a digital representation as an array of numbers describing the properties of pixels involves necessary compromises in quality.
- The image must be ‘quantised’ in the X and Y axes to break it up into pixels, and each pixel must be ‘quantised’ in colour, saturation and brightness.
- Quantisation amounts to an approximation, and determines the quality of the bitmap image.
- An image which is spatially quantised into 3000 x 2000 pixels has a higher quality than one quantised into 30 x 20 pixels.
- An image in which each pixel uses 24 bits to describe the pixel has a higher quality than one using 16, 8, 4 or 1 bit per pixel.
Vector Formats

- Vector formats represent a picture as a collection of ‘graphics primitives’ or basic shapes, each of which is described by a set of numbers.

- For instance, a line can be described by its two end points, \((X_1, Y_1)\) and \((X_2, Y_2)\), a circle by its centre \((X_C, Y_C)\) and a radius value.

- A pure vector format is limited in that only objects which can be properly described in such a form can be represented.

- Many vector formats include provisions for embedding bitmap images into the vector format image.

- Most vector formats include means of filling areas or shapes with specific colours, some even allow colour gradients.
Vector Formats ...
Vector Graphics Quality

- A vector representation of a picture is exact, therefore the quality of rendering depends completely on the quality of the rendering device, such as a display or a printer. Therefore, vector graphics always produce the highest possible quality when displayed or printed.

- Another advantage of vector representation is that it can be very compact, in comparison with a bitmap rendering of the same picture. This is because large areas can be completely described with very few numbers.

- Where the choice exists, vector graphics should always be used in preference to bitmap graphics.
Vector Graphics Quality ...

- Most drawing packages can produce both vector and bitmap output files, however, DTP and typesetting packages are frequently limited in what vector formats they can import.

- A nice attribute of well designed vector formats is that pictures in one format can be converted to another with no loss of information or quality.

- For instance, a picture in EPS format can be converted to PDF and vice versa, a picture in SVG into EPS and so on ...
Tagged Image File Format 6.0

- TIFF was devised by Aldus Corp in 1986 as a bitmap standard for supporting scanners and photo-retouching tools.
- The current version of TIFF is 6.0, the specification is maintained now by Adobe.
- TIFF is widely used in the graphics industry as a portable format for bitmap imagery.
- While early TIFF compatible tools lacked lossless compression facilities, current implementations usually support them.
- TIFF supports full 24-bit colour, Version 6.0 can represent colour in RGB, CMYK, YCbCr, and CIE formats.
- TIFF files with no compression are frequently very large and cumbersome to handle.
Tagged Image File Format 6.0 ...

- TIFF is usually supported by retouching and drawing tools, less frequently by DTP and TS tools, and browsers.

- Therefore it should be converted to another format for use with the latter, either JPEG or PNG.

- TIFF files usually have a .tif or .tiff suffix.
GIF87a, GIF89a

- The GIF format is now considered largely obsolete, but remains very widely used for web publishing.

- GIF is an ‘encumbered’ defacto-standard, since it incorporates a compression mechanism owned by Unisys. Therefore royalties are supposed to be paid on tools which can process GIF files.

- GIF has the nice attribute of lossless compression, as a result of which GIF files are always compact, and it represents line drawings without artifacts.

- GIF has an important limitation in poor colour handling. This is a consequence of its origins being in a period when most displays used only 8-bit colour representation.

- When producing new web documents, PNG should be used instead.
Portable Network Graphics

- PNG is an unencumbered bitmap graphics standard intended to replace GIF, and in many applications also TIFF.
- PNG employs lossless compression, transparency, gamma correction, cyclic redundancy check (CRC) error detection and progressive display.
- Colour handling in PNG is powerful, allowing 48-bit per pixel true colour, 16-bit per pixel grayscale and gamma correction to compensate for display non-linearity.
- PNG is the best current standard for web publishing, since it provides similar colour handling to TIFF, but employs lossless compression which allows it to render line drawings sharply and without artifacts.
Joint Photographic Experts Group

- JPEG is a bitmap representation standard which employs a lossy compression scheme. It is very widely used on the web.
- JPEG is commonly used by cheaper digital cameras for disk storage.
- The principal limitation of JPEG is its lossy compression mechanism, which can result in a highly variable quality of rendering and the appearance of decoding artifacts such as speckle.
- Lossy compression will ‘throw away’ some detail in the picture to achieve a higher compression ratio. Most tools allow some control of image quality by manipulating the compression parameters.
- JPEG can provide very good representation of imagery such as scanned pictures, but frequently performs poorly on line drawings.
Joint Photographic Experts Group ... 

- JPEG files can usually be identified by a .jpg or .jpeg suffix.

- Since you cannot control the quality of the tool being used to view your JPEG image, you can never guarantee the quality of the rendering seen by a user. JPEG should not be used for line drawings.
Computer Graphics Metafile

- The ISO Computer Graphics Metafile standard (ISO/IEC 8632:1992) is a vector graphics representation scheme, which is widely used in engineering tools.

- CGM is important for two reasons. It is the basis of the new W3C WebCGM standard, and it is the only vector graphics scheme at this time which is compatible with Microsoft tools and widely used Unix tools.

- A CGM file can include embedded and compressed bitmaps using CCITT group 4, JPEG, and LZ77 derivative compression.

- WebCGM employs RGB colour representation and supports transparent backgrounds.
Computer Graphics Metafile Example

BEGMF 'xfig-fig001442';
mfversion 1;
mfdesc 'Converted from /tmp/xfig-fig001442 using fig2dev -Lcgm';
mfelemlist 'DRAWINGPLUS';
v dctype integer;
fontlist 'Hardware','
'Times New Roman';
BEGMFDEFAULTS;
vdcext (0,0) (1284,1284);
clip off;
colrmode indexed;
colrtable 1
  0 0 0
  255 214 0;
linewidthmode abs;
edgewidthmode abs;
backcolr 255 255 255;
transparency ON;
ENDMFDEFAULTS;
BEGPIC 'xfig-fig001442';
BEGPICBODY;
% Polygon %
intstyle SOLID;
fillcolr 14;
edgevis ON;
edgetype 1;
edgewidth 15;
edgecolr 1;
polygon (1272,687) (597,1272) (12,597) (687,12) (1272,687);
% End of Picture %
ENDPIC;
ENDMF;
Scalable Vector Graphics (SVG)

- SVG is a vector graphics representation scheme based upon the XML standard, and is intended for use in web publishing.
- SVG may become the most widely used of all vector graphics representation schemes, once it matures.
- SVG supports vector graphics images, embedded bitmap images and text, and also includes two mechanisms for animation.
- Since SVG is based upon XML, it incorporates XML features such as style sheets and the Document Object Model.
- Once SVG and WebCGM become widely adopted, browsers will become capable of rendering line drawings with similar quality to that achieved by PostScript viewers such as gv and PDF viewers such as Acroread.
SVG Example

W3C Example “lingrad01.svg”.
SVG Example ...

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000802//EN"
  svg-20000802.dtd">
<svg width="8cm" height="4cm">
  <desc>Example lingrad01 - linear gradient</desc>
  <g>
    <defs>
      <linearGradient id="MyGradient">
        <stop offset="5%" style="stop-color:#F60"/>
        <stop offset="95%" style="stop-color:#FF6"/>
      </linearGradient>
    </defs>
  </g>
</svg>
```
SVG Example ...

<!-- Outline the drawing area in blue -->
<rect style="fill:none; stroke:blue"
    x=".01cm" y=".01cm" width="7.98cm"
    height="3.98cm"/>

<!-- The rectangle is filled using a linear gradient paint server -->
<rect style="fill:url(#MyGradient); stroke:black"
    x="1cm" y="1cm" width="6cm" height="2cm"/>

</g>
</svg>
Hypertext
References:


*Charles F. Goldfarb’s SGML SOURCE HOME PAGE:*
http://www.sgmlsource.com/
What is Hypertext?

- Hypertext is a mechanism which enables a reader to access another portion of a local or remote document by invoking a link.

- One important aim of hypertext is to hide the details of accessing the document or portion of a document which a link points to.

- Hypertext is therefore in concept an automated implementation of the process of finding a reference in a footnote or bibliography, and then using that reference to find another document.

- A hypertext system can be designed to support only links within a document, or only links on a single host system, or links across a large number of systems. The W3 is an example of the latter.
Hypertext Implementations

- Two very common hypertext schemes are the HTML/XML model used on the W3, and the simple GNU info scheme for online documentation.

- The GNU info scheme uses source files which are written in TeX layout markup language, with additional commands. These source files can be compiled into DVI format for conversion to hardcopy, or converted into online text documentation. A good overview can be found by typing info Texinfo on a Unix host with GNU info installed.

- The most widely used hypertext scheme is the W3C HTML/XML model, which allows links inside documents, links to documents on the same system, and links to documents anywhere on the W3.
HTML Limitations

- Invalid HTML Implementations. HTML syntax is simple, and is thus frequently abused. Some vendors create proprietary HTML which misbehaves on other browsers. No mechanism exists for ‘validating’ the syntactic correctness of HTML generators.

- Handling of Broken Links. When a web page is moved to another web server, all links to it are broken. No mechanism exists for handling this automatically.

- HTML Syntax is Fixed. The syntax of HTML is described by an SGML Document Type Definition (DTD) and cannot be changed. Therefore support for new features requires a new version of HTML.
HTML Limitations ...

- **Poor Metadata Facilities.** Metadata tags are important for automated web searches, the HTML `<meta>` tag is very limited and frequently not used.

- **Layout Markup.** HTML is primarily used for layout markup and does not have the powerful generalised markup facilities of languages such as \LaTeX. This makes it difficult to structure documents, and requires very disciplined use of layout markup commands by HTML programmers.

- **Lack of Object Oriented Features.** The OO model which is very popular in programming (e.g. C++ and Java) is not supported in basic HTML.
HTML Limitations ...

- Poor Support for Multilingual Text. Languages which use character sets other than the Latin type, e.g. Cyrillic, Kanji/Chinese, Katakana, Bengali, Hindi, Malayalam, are difficult to represent.

Goosens & Rahtz, Section 3.4, contains an excellent overview of \LaTeX\ schemes for representing non-latin character sets.

*The practical consequence of the limitations which exist in the early HTML versions (1-3) is that every vendor will add their own proprietary features to add functionality. In turn this results in serious compatibility problems between authoring tools and browsers.*
Why Not SGML For Web Pages?

- SGML is a meta-language, which like BNF syntax allows the definition of other languages. Therefore it was considered around 1995 as a direct replacement for HTML.

- Replacing HTML with SGML would provide the full functionality of SGML markup in webpages.

- The penalty for using SGML is that browsers would become significantly more complex, since the more complicated syntax of SGML would result in more complicated parsers.

- Most large software vendors objected to the use of SGML and insisted upon an alternative.

- The W3C standards body therefore defined the eXtensible Markup Language (XML) as a subset of SGML.
XML Goals (W3C REC-xml)

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
XML Goals (W3C REC-xml)

10. Terseness in XML markup is of minimal importance.

The XML specification reflects these ‘ten commandments’ very closely. XML is thus designed mainly for machine generated markup, rather than markup written by humans. This fits very closely with the industry trend to generate web pages using publishing tools rather than text editors. XML output filters are now becoming available for a range of tools, e.g. FrameMaker. Note the following observations:

1. XML is genuine industry standard and is not proprietary.
2. XML is not an variant of HTML, it is a new language.
3. XML web pages will require separate style sheets to define appearance.
4. XML is not limited to uses in presenting or exchanging data.
XML Features

- XML is a heavily restricted subset of SGML, the XML standard is about 26 pages long, the SGML standard around 600 pages long.
- An XML file can be viewed by an SGML tool by adding an SGML declaration at the beginning of the file.
- An XML document contains a number of ‘entities’ (objects), each with some logical ‘elements’, each element can have a range of ‘attributes’.
- The preferred technique for defining attributes is through a ‘Document Type Definition’ or DTD (a DTD is analogous to a documentclass or package in \LaTeX).
- Unlike HTML which has a fixed set of tags, XML allows user defined tags.
XML Syntax

- XML tag and entity reference syntax is fixed. This is to simplify the design of parsers.

- Elements and attributes are declared between matched pairs of angle brackets, i.e. `< ... >`, with attribute values always placed between single or double quotes i.e. `<ename attr1="value1" attr2='value2' .... >`.

- A reference to an entity always starts with an ampersand and ends with a semicolon, i.e. `&eref;`.

- The names of entities, elements and attributes are case sensitive.

- Comments are delimited by `<!--` and `-->`, i.e. `<!-- This is an XML comment :-D -->`. 
XML Syntax ...

- Empty elements are denoted by a slash before the closing bracket, i.e. `<empty/>`.

- A trivial XML example is:

```xml
<!-- This is a comment in my XML example -->
<myxmlexample>This is my XML example</myxmlexample>
```

Note the use of named tags, and paired opening and closing tags. XML will not tolerate missing closing tags in a statement, this is an intended design feature.
XML External DTD Syntax

- An XML DTD can be placed in a file external to the document, or embedded within the document. If the DTD is embedded, the document is said to be a ‘standalone’ document.

```xml
<?xml version="1.0"?>
<!DOCTYPE docu SYSTEM "http://www.blah.com/~doe/my.dtd">
<!-- This is an external DTD -->
<docu>
... <element1> ... <empel/> ... </element1>
</docu>
```
XML Embedded DTD Syntax

- The syntax for an embedded DTD is also very simple. Square brackets are used to encapsulate the contents of the DTD.

```xml
<?xml version="1.0"?>
<!-- This is an embedded DTD -->
<!DOCTYPE myxml [
  <!ELEMENT myxml (#PCDATA)>]

<myxml> This is my XML example! </myxml>
```
XML DTD Syntax

- An *element declaration* starts with `<!ELEMENT` and contains the type of the element and the model for its content.

- An *attribute declaration* starts with `<!ATTLIST` and for the given element contains one or more attributes, with type and default values.

- An *entity declaration* starts with `<!ENTITY` and contains the name and definition of an entity.

- A *notation declaration* starts with `<!NOTATION` and contains the name and external or public identifier used with a notation.

- A *processing instruction* is delimited with `<? ...?>` and contains data which is passed through to an application.

- A *comment* is delimited with `<!-- ... -->`.
DTD Element Declarations

• Every element in an XML document must be declared.

• The declaration must always include the type and content model of the element.

• XML supports four different content models:
  1. An empty element, e.g. `<ELEMENT emptyelement EMPTY>`.
  2. ‘Any’ element compatible with XML, e.g. `<!ELEMENT blah ANY>`.
  3. A list of, or nested list of child elements.
  4. A mixed content element, which includes both character data and child elements.
DTD Attribute Declarations

- The attributes of every element in an XML document must be explicitly declared.

- XML supports three categories of attribute types:
  1. String types defined by CDATA, e.g. `<!ATTLIST bloggs name CDATA #REQUIRED>`.
  2. Tokenised types which may be:
     (a) ID uniquely identifying a name.
     (b) IDREF, IDREFS referring to elements with an ID.
     (c) ENTITY, ENTITIES referring to entity names elsewhere in the DTD.
     (d) NMTOKEN, NMTOKENS referring to name tokens.
3. **Enumerated types** take one of a list of values defined in the DTD.

   (a) **notation types** with the keyword `NOTATION` followed by names of notations.

   (b) **enumerations** which are lists of name tokens associated with an attribute value.
DTD Attribute Declarations Example

Consider this example attribute declaration for a picture:

```xml
<!ATTLIST pict name ID #REQUIRED
    size CDATA #IMPLIED
    title CDATA 'Default Title'
    bordercolour NMTOKEN #FIXED 'blue'>
```
Entity Declarations

- Foreign material such as pieces of text, special characters, images and external files can be included in an XML document by using entity references.

- XML supports two types of entity:

  1. General entities, with declarations such as 
     \( <!ENTITY \text{GenEntityName} \text{GEDef} > \). These are referred to with an \& ... ; pair, e.g. \&GEDef;.

  2. Parameter entities, occurring only in the DTD with declarations of the form \( <!ENTITY \text{ParEntityName} \text{PEName} > \). These are referred to by using a % ... ; pair, e.g. %PEName;.
External vs Internal Entities

- An **internal** entity has its value specified inside the DTD itself.
- An **external** entity is any entity not specified inside the DTD.
Style Sheets, CSS, XSL
References:


The Style Sheet in HTML4 and XML

• Both HTML version 4. and XML employ a style sheet mechanism, based upon the Cascading Style Sheets (CSS2) specification.

• The aim of this mechanism is to provide these markup languages with a genuine capability to function as generic markup languages, in which style information is separated from content information.

• The style sheet mechanism is currently implemented using the CSS2 standard, with ongoing work being done on the eXtensible Stylesheet Language (XSL) and the Document Style Semantics and Specification Language (DSSSL).
CSS2 versus XSL?

• While the CSS2 and XSL schemes both provide means of manipulating the style information in a document, independently of content, they are somewhat different in their exact aims.

• The CSS2 scheme is optimised for use on the W3, and is designed to accommodate both HTML and XML.

• The XSL scheme is optimised for more complex documents and a print publishing environment.

• If you intend to produce W3 documents using HTML4 and XML, familiarity with the CSS2 specification is very useful.
CSS2 Syntax

• The CSS2 scheme uses a very simple model, in which rules are defined to specify the properties used in rendering a document.

• Each rule comprises two parts:
  1. A ‘selector’ which identifies the feature of the document, e.g. in HTML a header such as `<H1>`.
  2. A ‘declaration’ which describes the properties specified by the rule, e.g. `color:blue`.

• The CSS2 specification has more than 100 properties which can be manipulated.
CSS2 HTML Example (W3C)

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
  <head>
    <title>Bach's home page</title>
  </head>
  <body>
    <h1>Bach's home page</h1>
    <p>Johann Sebastian Bach was a prolific composer.</p>
  </body>
</html>
**CSS2 HTML Example (W3C)**

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
  <head>
    <title>Bach's home page</title>
    <style type="text/css">
      body {
        font-family: "Gill Sans", sans-serif;
        font-size: 12pt;
        margin: 3em;
      }
    </style>
  </head>
  <body>
  </body>
</html>
```
<h1>Bach's home page</h1>

<p>Johann Sebastian Bach was a prolific composer.</p>
CSS2 XML Example (W3C)

<TITLE>
  <HEADLINE>Fredrick the Great meets Bach</HEADLINE>
  <AUTHOR>Johann Nikolaus Forkel</AUTHOR>
</TITLE>

<PARA>
  One evening, just as he was getting his
  <INSTRUMENT>flute</INSTRUMENT> ready and his
  musicians were assembled, an officer brought him
  a list of the strangers who had arrived.
</PARA>

</ARTICLE>
CSS2 XML Example “bach.css” (W3C)

INSTRUMENT { display: inline }
ARTICLE, HEADLINE, AUTHOR, PARA { display: block }
HEADLINE { font-size: 1.3em }
AUTHOR { font-style: italic }
ARTICLE, HEADLINE, AUTHOR, PARA { margin: 0.5em }
Literate Programming Tools
References:

Literate Programming Website,
http://www.literateprogramming.com
Literate Programming FAQ,
http://shelob.ce.ttu.edu/daves/lpfaq/faq.html
Norman Ramsey’s noweb page,
http://www.cs.virginia.edu/~nr/noweb
Briggs P., Nuweb Version 0.91, A Simple Literate Programming Tool,
LP Toolsets

- The first toolset for LP was developed by Donald Knuth. It was based upon the use of Pascal and \TeX, and called \textit{WEB}.
- Knuth later adapted \textit{WEB} to use the C language and called the new toolset \textit{CWEB}.
- Since then, a wide range of LP toolsets have been developed, and many researchers have continued working in this area.
LP Toolsets ...

- The best known LP toolsets are:
  1. Norman Ramsey’s *noweb*, which uses any programming language and generates \TeX, \LaTeX and HTML documentation.
  2. Preston Briggs’ *nuweb*, which uses any programming language and generates \LaTeX documentation.
  3. Ross Williams’ *FunnelWeb*, which uses any programming language and generates \TeX or HTML documentation.
  4. Werner Lemberg’s *c2cweb*, which uses C and C++, and generates \TeX.
  5. Markus Oellinger’s *mCWEB*, which uses C and C++, and generates \TeX.
Tangling and Weaving

- An LP toolset must perform two basic tasks.

- **Weaving** is the process of creating documentation from the LP source files.

- **Tangling** is the process of creating program source code for compilation from the LP source files.

- Tools for weaving mostly generate \( \text{T\kern-.1667em\lower.5ex\hbox{E}\kern-.125emX} \) or \( \text{L\kern-.125em\lower.5ex\hbox{A}\kern-.125em\lower.5ex\hbox{T\kern-.125em\lower.5ex\hbox{E}\kern-.125emX}} \) source files, which can then be compiled into formatted documents.

- Tools for tangling may be specialised, and work only with specific languages, or they may be more general and thus work with any language.
Nuweb

• The Nuweb toolset is one of the simpler LP packages, and has the advantage of being very fast to execute.

• Nuweb employs a single tool which performs both tangling and weaving, unlike most LP toolsets.

• Nuweb generates \LaTeX documentation output, and exploits \LaTeX features where possible to simplify its own syntax.

• Nuweb permits the use of multiple output files from a single Nuweb source file, to facilitate the use of multiple languages and work by multiple programmers.

• Nuweb does not support the ‘pretty printing’ function used in WEB / CWEB and also cannot generate an index of identifiers and variables.
Nuweb ...

- Most of a Nuweb source file (filename.w) will be written in \LaTeX\ syntax, and is directly copied through to the documentation file during weaving.

- Fragments of program source in a Nuweb file are termed *scraps*. A scrap is typeset in *verbatim* format in the documentation, or used to generate a program source file.

- Scraps are identified by a beginning and an end delimiter, in this manner: @\{myscrap@\}. Everything in between the delimiters is copied through without change.

- Scraps may also be used to encapsulate math mode expressions in \LaTeX\. The syntax is then @\{mathmodescrap@\).
Nuweb Syntax (Cited from Briggs)

Files and Macros:

@o file-name flags scrap Output a file. The file name is terminated by whitespace.

@d macro-name scrap Define a macro. The macro name is terminated by a return or the beginning of a scrap.
Nuweb Syntax (Cited from Briggs)

Scraps:

@\{ anything @\} where the scrap body includes every character in anything—all the blanks, all the tabs, all the carriage returns. This scrap will be typeset in verbatim mode.

@[ anything @ ] where the scrap body includes every character in anything—all the blanks, all the tabs, all the carriage returns. This scrap will be typeset in paragraph mode, allowing sections of \TeX documents to be scraps, but still be pretty printed in the document.

@\( anything @ \) where the scrap body includes every character in anything—all the blanks, all the tabs, all the carriage returns. This scrap will be typeset in math mode. This allows this scrap to have a formula which will be typeset nicely.
Nuweb Syntax (Cited from Briggs)

Macros inside scraps:

@<macro-name> Causes the macro *macro-name* to be expanded in-line as the code is written out to a file. It is an error to specify recursive macro invocations.

Note that macro names may be abbreviated, either during invocation or definition. For example, it would be very tedious to have to repeatedly type the macro name

@d Check for terminating at-sequence and return name.

Therefore, we provide a mechanism (stolen from Knuth) of indicating abbreviated names.

@d Check for terminating...
Nuweb Syntax (Cited from Briggs)

File output flags:

-\texttt{-d} Forces the creation of \texttt{#line} directives in the output file. These are useful with C (and sometimes C++ and Fortran) on many Unix systems since they cause the compiler’s error messages to refer to the web file rather than the output file. Similarly, they allow source debugging in terms of the web file.

-\texttt{-i} Suppresses the indentation of macros. That is, when a macro is expanded in a scrap, it will \textit{not} be indented to match the indentation of the macro invocation. This flag would seem most useful for Fortran programmers.

-\texttt{-t} Suppresses expansion of tabs in the output file. This feature seems important when generating make files.
Nuweb Syntax (Cited from Briggs)

Minor commands:

@@ Causes a single “at sign” to be copied into the output.

@_ Causes the text between it and the next @_ to be made bold (for keywords, etc.)

@i file-name Includes a file. Includes may be nested, though there is currently a limit of 10 levels. The file name should be complete (no extension will be appended) and should be terminated by a carriage return.
Nuweb Syntax (Cited from Briggs)

Indexing commands:

@f Create an index of file names.

@m Create an index of macro name.

@u Create an index of user-specified identifiers.
Nuweb Example (nuweb.w)

Global include file definition:

@o global.h
@{@<Include files@>
@<Type declarations@>
@<Global variable declarations@>
@<Function prototypes@>
@}
Nuweb Example (nuweb.w)

Include file macro definition:

@d Include files
@{#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
@
FILE stderr exit fprintf fputs fopen fclose
cgetc putc strlen toupper isupper islower isgraph
isspace tempnam remove malloc size_t @}
Nuweb Example (nuweb.w)

Preprocessor constant definition:

@d Type dec...
@{#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE 1
#endif
@| FALSE TRUE @
@| FALSE TRUE @}
Nuweb Example (nuweb.w)

Main file definitions:

@o main.c
@{#include "global.h"
@
}

The first pass over the source file is contained in \verb|pass1.c|. It handles collection of all the file names, macros names, and scraps (see Section\ref{pass-one}).
@o pass1.c
@{#include "global.h"
@
}
The \verb|.tex| file is created during a second pass over the source file. The file \verb|latex.c| contains the code controlling the construction of the \verb|.tex| file (see Section~\ref{latex-file}).

\begin{verbatim}
@o latex.c
@include "global.h"
\end{verbatim}

The file \verb|html.c| contains the code controlling the construction of the \verb|.tex| file appropriate for use with \LaTeX{2}HTML (see Section~\ref{html-file}).

\begin{verbatim}
@o html.c
@include "global.h"
\end{verbatim}
End CSE-1402 Tools Stream Lectures