

XLP Release 2

A.J.Hurst

Version 2.4.0

20080822:162322

Table of Contents

1	Introduction	3
1.1	Background	3
1.2	Synopsis	3
1.3	The source file structure (XLP files)	4
1.4	Some base definitions	4
2	User Manual	5
2.1	Overview of an XLP Document	5
2.2	Documentation in an XLP Document	5
2.3	Code in an XLP Document	7
2.4	Running XLP	8
3	The warp, or literate program phase	9
3.1	Templates defined in litprog.xsl	9
3.2	litprog.xsl: the warp translator	9
3.2.1	litprog: root template	10
3.2.2	litprog: define variables	10
3.2.3	litprog: define script parameters	11
3.2.4	litprog: define procedure templates	11
3.2.5	litprog: do-replace template	11
3.2.6	litprog: count-trailing-blanks template	12
3.2.7	litprog: lookupchunkno template	12
3.2.8	litprog: lookup template	14
3.2.9	litprog: substparms template	14
3.3	litprog: the tangle pass	17
3.3.1	litprog: tangle all litprog files	
3.3.2	litprog: tangle a single file definition chunk	
3.3.3	litprog: tangle a code definition	17
3.3.4	litprog: tangle variable declarations	19
3.3.5	litprog: discard documentation in tangle mode	19
3.3.6	litprog: tangle all chunks into a single use	20
3.3.7	litprog: default tangle template	24
3.4	litprog: the weave pass	24
3.4.1	litprog: weave a litprog	25
3.4.2	litprog: weave a code or file definition	25
3.4.3	litprog: weave a variable markup	27
3.4.4	litprog: u-weave template	28
3.4.5	litprog: p-weave template	29
3.4.6	litprog: section-weave template	29
3.4.7	litprog: listing templates	30
3.4.8	litprog: default weave template	31
4	The weft, or documentation phase	32
4.1	lit2html.xsl	32
4.1.1	lit2html: handle the root element	33
4.1.2	lit2html: handle the literate program translation	33
4.1.3	lit2html: handle documentation fragments	33
4.1.4	lit2html: handle code fragment references	34
4.1.5	lit2html: handle a code fragment use	35
4.1.6	lit2html: handle code chunk definitions	36
4.1.7	lit2html: handle notes and comments in code fragments	39

4.1.8	lit2html: handle variable definitions and uses	40
4.1.9	lit2html: handle maintaintable	41
4.1.10	lit2html: TableOfContents	41
4.1.11	lit2html: handle a code text element	43
4.1.12	lit2html: construct file definition list	43
4.1.13	lit2html: construct chunk definition list	44
4.1.14	lit2html: construct identifier definition list	45
4.1.15	lit2html: handle macro parameters	46
4.1.16	lit2html: handle document history	46
4.1.17	lit2html: special character templates	47
4.1.18	lit2html: default template	47
4.2	lit2tex.xsl	48
4.2.1	lit2tex: handle the root node	49
4.2.2	lit2tex: handle various documentation elements	49
4.2.3	lit2tex: handle a literate program translation	50
4.2.4	lit2tex: handle paragraphs	50
4.2.5	lit2tex: handle reference calls	50
4.2.6	lit2tex: handle code fragment definitions	51
4.2.7	lit2tex: handle code use calls	53
4.2.8	lit2tex: handle macro parameters	54
4.2.9	lit2tex: handle variable definitions and uses	54
4.2.10	lit2tex: handle in-line comments in code	55
4.2.11	lit2tex: handle maintenance table documentation	55
4.2.12	lit2tex: build table of contents	56
4.2.13	lit2tex: build file list	57
4.2.14	lit2tex: build identifier list	58
4.2.15	lit2tex: handle chunk definition list	59
4.2.16	lit2tex: handle document history	61
4.2.17	lit2tex: text node handling	63
4.2.18	lit2tex: special character templates	65
4.2.19	lit2tex: default template	65
5	weft: Common weft fragments	66
6	The litprog Document Type Definition	67
6.1	litprog.dtd: Literate Programming elements	68
6.2	litprog.dtd: Bibliographic elements	68
6.3	litprog.dtd: Heading and Sectioning elements	69
6.4	litprog.dtd: Miscellaneous elements	69
6.5	litprog.dtd: table elements	69
6.6	litprog.dtd: uri elements	69
6.7	litprog.dtd: DocumentHistory elements	70
6.8	litprog.dtd: Entities	70
7	The LitprogXML Document Type Definition	71
8	File Index	72
9	Macro Index	73
10	Identifier Index	76

1 Introduction

1.1 Background

Literate Programming is a technique developed by Donald Knuth, and described in his seminal paper in the *Computer Journal*, 1984. It is both a software process and a suite of tools for supporting that process. Programs written as literate programs go through two (conceptually) independent processes known as *tangling* and *weaving*. Tangling creates the actual source code for input to a compiler; weaving creates a documentation file suitable for processing by a document system such as \TeX .

More modern versions of literate program tools have attempted to decouple the $N \times M$ combinatorial explosion created by the choice of N programming languages and M documentation systems, by restricting the markup and presentational mechanisms. The tool currently in use in the work described here is such a tool: the markup of source documents is done in XML, and the documentation device is handled by XML translators. This reduces the problem from an $N \times M$ to an $N + M$ one, where $N = 1$.

XLP started life as a literate programming tool developed from nuweb, and modified to develop XML code rather than LaTeX or HTML. It was extensively rewritten during 2003 to use an XSLT script rather than a C program to do the tangle and weave processes, so the code is completely platform independent. To run the code, only an XSLT translator (any flavour) is needed. The price for this conversion (apart from time) has been the loss of a number of “syntactic sugar” features, but I don’t miss them much. They are gradually being reintroduced, generally in an improved form.

This new version is currently used to define and maintain a range of software that I use on a regular basis.

1.2 Synopsis

This program, or rather, suite of programs, performs literate program analysis and construction over a literate program file. Programs in any language can be constructed, as can documentation in any form. There are two phases to the use of the suite, each of which is handled by an XSLT file. The first of these phases uses a common XSLT file, while the second phase uses an XSLT file that is specific to the form of documentation being generated. For naming purposes, the first phase is known as the **warp phase**, while the second phase is called the **weft phase**.

The first phase (warp phase) comprises two passes, corresponding to the tangling and weaving operations of all literate program systems. It is important to make the distinction between *phase* and *pass* in this context. This phase is performed by the `litprog.xml` file. Each of the passes within this phase is handled by a *mode* (in the XSL technical sense). The first pass, or `mode=tangle`, generates the tangled program files, and the second pass `mode=weave` generates an intermediate file of the woven documentation file.

This documentation file is then processed in phase two of the suite (the weft phase) by a second XSLT file, which is documentation language specific. There is only one pass in the weft phase. Currently, there are two such XSLT files, `lit2html.xml` for HTML documentation, and `lit2tex.xml` for $\{\text{\TeX}\}$ documentation. The output of this phase is an HTML or \TeX document, suitable for processing by a browser (HTML) or by the \TeX system itself (\TeX). There are a standard set of macros (based upon plain \TeX) that are assumed by the \TeX translator, and which are available separately.

All three XSLT files (`litprog.xml` for the warp phase, and `lit2html.xml` and `lit2tex.xml` for the weft phase) are described in this document, which is itself a literate program document.

1.3 The source file structure (XLP files)

The warp (first) phase of XLP reads the input XLP file, a document type definition for which is given in section 5, The litprog Document Type Definition¹ (see also the file `<litprog.dtd 6.1>`). In BNF, the grammar of this file is as follows (terminals are represented in CAPITALS, although the corresponding element names are lower case, except for TableOfContents):

```
litxmlfile      = TITLE AUTHOR VERSION DATE
                  ( section | misc ) *
                  DocumentHistory?
.
section         = TITLE ( p | o | d | subsection | misc)* .
subsection      = TITLE ( p | o | d | subsubsection | misc)* .
subsubsection   = TITLE ( p | o | d | subsubsubsection | misc)* .
subsubsubsection = TITLE ( p | o | d)* .
misc           = TABLEOFCONTENTS | FILELIST | MACROLIST | IDENTIFIERLIST
.
o              = (CODE | u | v)* .
d              = (CODE | u | v)* .
u              = USE-REFERENCE .
v              = VARIABLE-REFERENCE .
p              = (TEXT | u | v)* .
DocumentHistory = Modified* .
Modified       = DATE AUTHOR VERSION COMMENT .
```

See the User Manual² (following) for explanations of these elements.

1.4 Some base definitions

Here we define some important system parameters.

The first one is the location for the library of XSL translations for various base templates. Update this to point at the directory used to store templates such as `basic2html.xsl`, `basic2tex.xsl`, and so on (see for example, `<lit2html.xsl 4.1>` and `<lit2tex.xsl 4.33>`).

```
<LIBXSL 1.1> =file:///home/ajh/lib/xsl ◊
```

Macro referenced in chunk 4.1,4.1,4.1,4.1,4.33,4.33,4.33,4.33

The following list defines all documentation template names. All the names of documentation templates that are passed straight through phase one (the *warp* phase) should be added to this list, in order to avoid any “No warp-tangle template” or “No warp-weave template” errors.

```
<litprog: define documentation template name list 1.2> =
  <xsl:variable name="docos">
    <xsl:text>|author|b|c|col|comment|date|description|dq|</xsl:text>
    <xsl:text>DocumentHistory|em|i|itemize|item|</xsl:text>
    <xsl:text>le|Modified|narrower|</xsl:text>
    <xsl:text>section|smiley|subsection|subsubsection|</xsl:text>
    <xsl:text>subsubsubsection|paragraph|parm|table|TableOfContents|TeX|</xsl:text>
    <xsl:text>term|td|th|title|tr|tt|uri|verb|verbatim|version|</xsl:text>
    <xsl:text></xsl:text>
  </xsl:variable>
```

◊

Macro referenced in chunk 3.3

¹ #TheLitprogDocumentTypeDefinition

² #UserManual

2 User Manual

2.1 Overview of an XLP Document

An XLP document is a well-formed XML document. For readers unfamiliar with XML, an XML document consists of a set of elements, each flagged with a starting and ending tag, and containing other elements, text, or both. Tags are defined by writing the element name within angle brackets `<>`, and the ending or closing tag has a leading `/` before the name. The term “well-formed XML document” means that all elements are strictly nested, and there is one outer level or root element.

`<p>An example of a paragraph element, with nested bold elements</p>`

The structure of the XLP document is formally defined elsewhere, but an informal description will suffice here. The basic outline of an XLP literate program looks like this:

```
<?xml version="1.0"?>
<!DOCTYPE litprog SYSTEM "/home/ajh/lib/dtd/litprog.dtd">
<litprog>
... literate program content goes in here ...
</litprog>
```

The first line of this program is a declaration that the document is an XML document, constructed according to the rules of XML version 1.0. This line is optional. The second line is mandatory, and specifies that the rules of this particular XML document, an XLP (or XML Literate Program) document, are defined in the separate file `/home/ajh/lib/dtd/litprog.dtd`.

The top-level or root element is a `litprog` element. This element contains all the literate programming content. Such content comes in two basic forms: documentation and code. Each of these two forms have their own set of elements. There is limited mixing of these elements, but generally an element is either pure code or pure documentation. These forms are described in the next section.

The basic model is that the document has the structure of a typical descriptive article: opening components are followed by substantive components, which are then followed by closing components. Opening material are things like title, author, date, which are required (if present) to be in a particular order. The substantive content is encapsulated within sections, each contain possible subsections, subsubsections, and even subsubsubsections. Closing material includes things like tables of content and indices.

Interspersed within the document are the code chunks, which are file and code definitions. These may generally appear at any point in the document, but it is their document order that defines how they are assembled into the overall code files. Readers familiar with other literate programming systems may view this arrangement as a conventional alternation of code and documentation fragments, with a superimposed document structure.

2.2 Documentation in an XLP Document

Documentation in an XLP document is built from the set of elements:

```
author b c col comment date description dq DocumentHistory em filelist i identlist
item le macrolist Modified p section subsection subsubsection subsubsubsection table
TableOfContents term TeX td th title tr uri verb verbatim version
```

Generally, these are constrained as to where they can appear, but some can be used in arbitrary order. Their usage is described in the following list, where the annotation “*” is used to mark those that are constrained.

- author*** The author of this literate program. Can appear in two places: at the top level, after the `title` element, or within a `Modified` element, after a `date` element.
- b** markup text in bold face
- c** markup text in code (typewriter) face
- col** The column specification element, used in tables. It is an empty element, with one attribute, `width`, defining the width of a column as a fraction (≤ 1.0) of the total page width.

comment*	The last element within a <code>Modified</code> element, describing the nature of this particular modification.
d	define a code chunk (see code)
date*	Can appear after <code>version</code> in the document header, or as the first element in a <code>Modified</code> element. It defines the date of the document, or modification, respectively.
DocumentHistory*	A sequence of <code>Modified</code> elements, each of which contains a <code>date</code> , <code>author</code> , <code>version</code> , <code>comment</code> sequence of nested elements, each of which documents the appropriate data about the various modifications to this literate program. If it is present, it must be the last element in a <code>litprog</code> element.
dq	place double quotes around text
filelist*	A place marker to indicate where the list of files generated should be inserted. It can appear at the top level, or within a (sub)section.
i	markup text in italic face
identifierlist*	A place marker to indicate where the table of identifier cross references should be inserted. It can appear at the top level, or within a (sub)section.
item*	Can only appear within a <code>description</code> , <code>enumerate</code> or <code>itemize</code> element, where it defines one of the list items.
le	The less than or equal to symbol, \leq .
macrolist*	A place marker to indicate where the list of macro names should be inserted. It can appear at the top level, or within a (sub)section.
Modified*	contains <code>date</code> , <code>author</code> , <code>version</code> , <code>comment</code> elements, defining attributes of each modification of the literate program.
o	define an output file (see code)
p	markup a paragraph
section*	Can only appear as a top-level element within the <code>litprog</code> element.
subsection*	Can only appear within a <code>section</code> element.
subsubsection*	Can only appear within a <code>subsection</code> element.
subsubsubsection*	Can only appear within a <code>subsubsection</code> element.
TableOfContents*	Defines the place in the literate program where the table of contents should be inserted. Can only appear at the top level of the document, or within a section or sub(sub...)section.
term*	Can only appear within a <code>description</code> element, where it defines the (documentation) term being defined.
TeX	The \TeX token.
td*	As for HTML, within a table row element (<code>tr</code>), defines a table data cell.
th*	As for HTML, within a table row element (<code>tr</code>), defines a table header cell.
title*	When this element appears immediately after the opening <code>litprog</code> tag, it is the title of this literate program. Within a section or sub..section, it is the title of the corresponding (sub..)section.
tr*	As for HTML, defines a table row.
u	A use reference to a code fragment. In documentation, during the weaving pass, the reference becomes a link to the code fragment, unless an optional attribute <code>expand</code> is present and non-empty, when the reference is expanded to its value. In code fragments, during the tangling pass, the reference is replaced by the expanded code fragment, as a (recursive) macro expansion.
uri	markup a Universal Resource Indicator
v	markup a variable (see code)
verb	markup verbatim text in-line
verbatim	markup verbatim text as a display (paragraph) item.

version* Can appear in either the literate program header or a **Modified** element. In both cases, it follows an **author** element.

2.3 Code in an XLP Document

Code elements in an XLP literate program take two forms: file definitions, and code chunks. Code chunks are fragments of code used to build other code chunks or file definitions. For this reason, the distinction between code chunks and file definitions is often blurred, and they are referred to collectively as code chunks. It should be obvious from the context when it is important for the distinction to be made.

The code fragments are assembled through a process of expansion (in technical terms, a “macro expansion”) that replaces references to other chunks by the code defined in that chunk.

Code within a chunk can be quite arbitrary: there is no interpretation or parsing of the code, **except** for XML elements. Remember that the document is one large XML file, and text within a code chunk is interpreted according to the rules of XML. In practice, this means that the characters “less than” (<) and “ampersand” (&) have special meanings, and must be escaped if they are used within code text. The escape sequences are `<`; and `&`; respectively.

Alternatively, the block quoting mechanism of XML can be used. This entails surrounding the entire text that needs to be quoted with the special sequences `<![CDATA[and]]>`. All text within these quote sequences will appear verbatim. It follows that if the sequence `]]>` is to appear within the quoted text, it too must be escaped, for example with `]]><![CDATA[]]><![CDATA[]]><![CDATA[]]><![CDATA[]]>` (the three separate characters, each surrounded by close then open escape sequences). Strictly speaking, breaking the sequence once would suffice, but this is a more sesquipedalian approach !

There are three code elements that may appear within documentation (and already mentioned above):

- o** An *output* element. This element defines a file built by the literate program. The name of the file is given by the value of a required attribute, the **file** attribute. Note that more than one file element may have the same name, when the file is considered to be the concatenation of all similarly named chunks, arranged document in order. All file definitions are collected into a list accessed by the element `filelist`.
- d** A code chunk *defining* element. The name of the code chunk (by which it may be referenced) is given by the value of a required attribute **name**. Note that more than one code chunk may have the same name, when the code chunk is considered to be the concatenation of all similarly named chunks, arranged in document order. It is an error to have names that are common to both code and file chunks. All code definitions and uses are collected into a list accessed by the element `macrolist`.
- v** A *variable* element. This may be either a defining occurrence, or a using occurrence, although within documentation it is unlikely to be the former. Within documentation, is usually used to highlight discussion about a particular variable within the code. See below for use within a code chunk. All variable definitions and uses are collected into a list accessed by the element `identifierlist`.

There are three elements that may appear within code chunks (i.e., within **o** or **d** elements). These are:

- u** A code chunk *using* element. This element is normally empty, and defines the point at which a code chunk is expanded into the current code chunk or output file. It has a mandatory attribute **name**, which cross-references the corresponding code chunk(s). It may also have the optional attribute **include**, which, when it has the value **no**, negates the inclusion of the reference code chunk. This may be used to “comment out” code, while still retaining the documentation surrounding the excluded code. The documentation is annotated accordingly.

A second optional attribute is `expand`, which normally has the default value `no`. When explicitly set to `yes` however, it forces the expansion of the chunk into the document at that point. When called within code chunks, this is the default behaviour, but when called within documentation chunks, this changes the representation of the macro from a link to the expanded text of the macro. This is useful for defining macros used within the documentation, rather than code generation (for example, the date at the head of this document was generated in this way). **Warning:** multiple chunks are not handled correctly, as of version 2.2.0.

v A *variable* element. See also `use` within documentation, above. This may be a defining occurrence, or a using occurrence, depending upon the optional attribute `var`. If `var` has the (default) value `use` (`var="use"`), then the occurrence is marked up as a use of the variable. If the attribute has the value `def` (`var="def"`), then the occurrence is marked up as a defining occurrence. Note that there is **no** analysis of the content of the variable element, which allows arbitrary lexical forms of variable names to be used. (But be wary of languages such as perl, or bash shell scripts, which use `$` and other signs to flag usage of the variable. These annotations should be excluded from the variable name.)

com A *comment* element. This element may be used to markup in-line comments within the code. How the comment is rendered depends upon the subsequent weft phase, but it will presumably capitalize upon the contextual position. (Aside: the tangling of the output file could conceivably include the comment as an in-line comment - the difficulty here is that the literate programming system has no knowledge of the syntactic form of such in-line comments.)

2.4 Running XLP

To use the XLP literate programming system, you need an XSLT translator. I use `xsltproc` (see LibXML³), but other XSLT translators will work just as well with the files described herein. In what follows, we assume the `xsltproc` translator.

There are two phases to running the system, called the warp and weft phases. The warp phase performs tangling and weaving, and generates an intermediate file for the weft phase. To run the warp phase, use the call:

```
xsltproc litprog.xsl yourXLPfile.xlp >yourXLPfile.xml
```

For the weft phase (document generation), use the call

```
xsltproc lit2html.xsl yourXLPfile.xml >yourXLPfile.html
```

This is for HTML generation. For \TeX documentation, replace the two occurrences of `html` in the above line with `tex`.

Note that if documentation is not required, the weft phase may be omitted.

3 The warp, or literate program phase

This phase is responsible for reading and translating the source xlp file. There are two passes to this phase, known as the **tangle** and **weave** passes. Each pass performs a sweep over the input file, applying templates to the elements as they are seen. In the first pass, the XSLT mode of `mode=tangle` is used; in the second pass, `mode=weave` is used.

In the tangle pass, all documentation chunks are effectively ignored, while file chunks cause a new document to be written, containing the code within the file chunk, with all code chunk references expanded out through a process of macro expansion. Each file chunk creates a new text document named with the given file name, which can then be used for further processing by compilers, etc..

In the weave pass, a new translation of the source XLP document is made. This new document is given a `.xml` extension, and its contents consist of much the same contents as are in the original XLP file, but with some elements renamed, and additional housekeeping information issued to assist the final weft phase.

3.1 Templates defined in `litprog.xsl`

name	attributes	mode	purpose
/	(none)	(none)	The root node
litprog	(none)	tangle	tangle all output files
d	name	tangle	define a code chunk named "@name"
o	name	tangle	build a new output file, called "@name"
d/v — o/v	var	tangle	define (<code>var=def</code>) or reference (<code>var=use</code>) a variable (default is reference).
p		tangle	paragraph documentation, ignored
d/com — o/com		tangle	comment with a code/file chunk: ignored
u		tangle	chunk use, expand the referenced code chunk in-line
*		tangle	undefined element, issue a warning message
litprog		weave	Perform the weave pass
d — o		weave	document a code chunk (d) or output file (o)
d/v — o/v		weave	document a variable markup (either use or definition, defined by element attribute <code>var=use</code> (default) or <code>var=def</code>) in code (d/v) or file (o/v)
d/com — o/com		weave	document a comment markup in code (d/com) or file (o/com)
v		weave	document a value reference to a code or file chunk
u	expand	weave	document a use reference to a code or file chunk. If the attribute expand is present and non-empty, replace this element by the expanded code definition.
p		weave	document a paragraph of literate program text
section		weave	document a section of the literate program text
*		weave	undefined element, issue a warning message

3.2 `litprog.xsl`: the warp translator

```
"litprog.xsl" 3.1 =
  <?xml version="1.0"?>
  <!-- This is version 2 -->
  <!DOCTYPE xsl:stylesheet PUBLIC "-//MONASH-CSSE//DTD stylesheet 1.0/EN"
    "file:///home/ajh/lib/dtd/stylesheet.dtd">
  <xsl:transform
    version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```

xmlns:xinclude="http://www.w3.org/2001/XInclude"
xmlns:xi="http://www.w3.org/2001/XInclude"
>
<xsl:output method="xml" encoding="ASCII"/>
<litprog: define script parameters 3.4>
<litprog: define variables 3.3>
<litprog: define procedure templates 3.5>
<litprog: root template 3.2>
<litprog: the tangle pass 3.15>
<litprog: the weave pass 3.39>
</xsl:transform>

```

◇

This is the XSLT transformer script for phase one, the **warp phase** of the literate programming suite. It is responsible for *tangling* the output documents, and *weaving* the XML document for input to the weft phase.

3.2.1 litprog: root template

```

<litprog: root template 3.2> =
  <xsl:template match="/">
    <xsl:message><xsl:text>Tangling ...</xsl:text></xsl:message>
    <xsl:apply-templates mode="tangle"/>
    <xsl:message><xsl:text>Weaving ...</xsl:text></xsl:message>
    <xsl:element name="LitprogXML">
      <xsl:apply-templates mode="weave"/>
    </xsl:element>
  </xsl:template>

```

◇

Macro referenced in chunk 3.1

The root template is responsible for the two passes of tangling and weaving. Note that the tangle phase generates its own documents (in the XML technical sense), while all the documentation for the weft phase is generated as the “default” output of this phase by the tangle pass.

3.2.2 litprog: define variables

```

<litprog: define variables 3.3> =
  <xsl:variable name="crossrefs">
    <xsl:for-each select="//d//o">
      <xsl:value-of select="concat(@name,@file)"/>
      <xsl:text>@</xsl:text>
      <xsl:number format="1" level="multiple" count="section"/>
      <xsl:text>.</xsl:text>
      <xsl:number from="section" level="any" count="d|o"/>
      <xsl:text>&#xA;</xsl:text>
    </xsl:for-each>
  </xsl:variable>
  <xsl:variable name="premacro">
    <xsl:text>[[</xsl:text>
    <xsl:text>[[</xsl:text>
  </xsl:variable>
  <xsl:variable name="postmacro">
    <xsl:text>]]</xsl:text>
    <xsl:text>]]</xsl:text>
  </xsl:variable>
  <litprog: define documentation template name list 1.2>

```

◇

Macro referenced in chunk 3.1

The variable `crossrefs` is set to a value built from a newline separated list of code and file chunks, with each entry (or line) in the form `name@location`, where `name` is the name of the chunk or file, and `location` is the chunk number defining that file or chunk.

The value stored as the value of a chunk number is multi-valued, in the form `s.p`, where `s` is the section number and `p` is the part number within the section.

The two variables `premacro` and `postmacro` define the escape sequences used respectively to start and end a macro text.

3.2.3 litprog: define script parameters

The script parameters give some control over the behaviour of the literate programming system. Currently two are defined: `Verbose` and `machine`.

`Verbose` can be set to a non-negative integer. When 0 (the default), no tracing information is given. Progressively higher values give more information about the progress of tangling and weaving.

`machine` is used to define the target of compilation. Not currently interpreted.

```
<litprog: define script parameters 3.4> =  
  <xsl:param name="Verbose">1</xsl:param>  
  <xsl:param name="machine">ararat</xsl:param>
```

◇

Macro referenced in chunk 3.1

3.2.4 litprog: define procedure templates

Define all the callable templates for the warp phase.

```
<litprog: define procedure templates 3.5> =  
  <litprog: do-replace template 3.6>  
  <litprog: count-trailing-blanks template 3.7>  
  <litprog: lookupchunkno template 3.8,3.9>  
  <litprog: lookup template 3.11>  
  <litprog: substparms template 3.12>
```

◇

Macro referenced in chunk 3.1

3.2.5 litprog: do-replace template

```
<litprog: do-replace template 3.6> =  
  <xsl:template name="do-replace">  
    <xsl:param name="text"/>  
    <xsl:param name="replace"/>  
    <xsl:param name="by"/>  
    <xsl:choose>  
      <xsl:when test="contains($text,$replace)">  
        <xsl:value-of select="substring-before($text,$replace)"/>  
        <xsl:value-of select="$by"/>  
        <xsl:call-template name="do-replace">  
          <xsl:with-param name="text"  
            select="substring-after($text,$replace)"/>  
          <xsl:with-param name="replace"  
            select="$replace"/>  
          <xsl:with-param name="by"  
            select="$by"/>
```

```

        </xsl:call-template>
    </xsl:when>
    <xsl:otherwise>
        <xsl:value-of select="$text"/>
    </xsl:otherwise>
</xsl:choose>
</xsl:template>

```

◇

Macro referenced in chunk 3.5

do-replace is a procedure with three parameters, text, replace, and by. Multiple occurrences of replace in the string text are recursively replaced by the value of by, and this rebuilt value is returned as the value of the template call.

3.2.6 litprog: count-trailing-blanks template

```

<litprog: count-trailing-blanks template 3.7> =
    <xsl:template name="count-trailing-blanks">
        <xsl:param name="string"></xsl:param>
        <xsl:param name="count"></xsl:param>
        <xsl:choose>
            <xsl:when test="substring($string,string-length($string))=' '>
                <xsl:call-template name="count-trailing-blanks">
                    <xsl:with-param name="string"
                        select="substring($string,1,
                            string-length($string)-1)"/>
                <xsl:with-param name="count" select="$count+1"/>
            </xsl:call-template>
            </xsl:when>
            <xsl:otherwise>
                <xsl:value-of select="$count"/>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template> ◇

```

Macro referenced in chunk 3.5

count-trailing-blanks, as its name suggests, (recursively) counts the trailing blanks in the string parameter string and returns this value. The initial value of the parameter count should be set as 0. The actual value returned by the procedure is this initial value plus the number of trailing blanks, hence the need to initialize it correctly.

3.2.7 litprog: lookupchunkno template

```

<litprog: lookupchunkno template 3.8> =
    <xsl:template name="lookupchunkno">
        <xsl:param name="chunkname"></xsl:param>
        <xsl:call-template name="lookupchunknoR">
            <xsl:with-param name="chunkname" select="$chunkname"/>
            <xsl:with-param name="table" select="$crossrefs"/>
            <xsl:with-param name="sep"></xsl:with-param>
        </xsl:call-template>
    </xsl:template>

```

◇

Macro defined in chunk 3.8,3.9

Macro referenced in chunk 3.5

lookupchunkno returns a list of the chunk numbers of a named chunk. The name of the required chunk is passed as the parameter chunkname, and the procedure calls an auxiliary recursive template lookupchunknoR to find the actual chunk numbers.

```
<litprog: lookupchunkno template 3.9> =
  <xsl:template name="lookupchunknoR">
    <xsl:param name="chunkname"></xsl:param>
    <xsl:param name="table"></xsl:param>
    <xsl:param name="sep"></xsl:param>
    <xsl:variable name="find">
      <xsl:value-of select="substring-after($table,concat($chunkname,'@'))"/>
    </xsl:variable>1
    <litprog: lookupchunknoR debug message 1 3.10 Chunk omitted!>
    <xsl:if test="$find!=''">2
      <xsl:value-of select="$sep"/>
      <xsl:value-of select="substring-before($find,'&#xA;')"/>
      <xsl:call-template name="lookupchunknoR">
        <xsl:with-param name="chunkname" select="$chunkname"/>
        <xsl:with-param name="table" select="$find"/>
        <xsl:with-param name="sep"><xsl:text>,</xsl:text></xsl:with-param>
      </xsl:call-template>
    </xsl:if>
  </xsl:template>
```

◇

Macro defined in chunk 3.8,3.9

Macro referenced in chunk 3.5

Notes:

- 1) The first key part of the lookup algorithm. Jump straight to the match in the table of the required chunk name. If it is there, the find variable is set to the remainder of the table starting with the match. If it is not there, the find variable will be empty.
- 2) The second key part of the lookup algorithm. Return the first part of the find string (it is the desired chunk number, and use the (remainder of the) find string as the lookup table for the next recursion. The algorithm is order n, where n is the number of matches in the cross reference table for chunkname.

lookupchunknoR takes as parameters the required chunkname, a partially searched table of cross references, and the separator string used to separate matches for the chunk name. It returns a sep separated list of values that match the second half of all (chunkname, chunkno) pairs in the cross reference list.

The sep separator is passed as a parameter, since on the first call, it must be empty. Otherwise we would return lists of the form ,4,10,15 rather than 4,10,15.

```
<litprog: lookupchunknoR debug message 1 3.10> =
  <xsl:message>
    <xsl:text>lookupchunknoR(</xsl:text>
    <xsl:value-of select="$chunkname"/>
    <xsl:text>,</xsl:text>
    <xsl:value-of select="$table"/>
    <xsl:text>,</xsl:text>
    <xsl:value-of select="$sep"/>
    <xsl:text>')</xsl:text>
  </xsl:message>
```

◇

Macro referenced in chunk 3.9

This chunk is conditionally included to provide debugging information about the progress of the chunk number lookup procedure.

3.2.8 litprog: lookup template

```
<litprog: lookup template 3.11> =
  <xsl:template name="lookup">
    <xsl:param name="key"></xsl:param>
    <xsl:param name="table"></xsl:param>
    <xsl:variable name="pkey">
      <xsl:value-of select="$premacro"/>
      <xsl:value-of select="$key"/>
      <xsl:value-of select="$postmacro"/>
      <xsl:value-of select="$premacro"/>
    </xsl:variable>
    <xsl:variable name="tail" select="substring-after($table,$pkey)"/>
    <xsl:variable name="retval">
      <xsl:value-of select="substring-before($tail,$postmacro)"/>
    </xsl:variable>
    <xsl:if test="$Verbose>=2">
      <xsl:message>
        <xsl:text>lookup:</xsl:text>
        <xsl:text>&#xa; key=</xsl:text>
        <xsl:value-of select="$key"/>
        <xsl:text>&#xa; table=</xsl:text>
        <xsl:value-of select="$table"/>
        <xsl:text>&#xa; pkey=</xsl:text>
        <xsl:value-of select="$pkey"/>
        <xsl:text>&#xa; tail=</xsl:text>
        <xsl:value-of select="$tail"/>
        <xsl:text>&#xa; retval=</xsl:text>
        <xsl:value-of select="$retval"/>
        <xsl:text>&#xa;</xsl:text>
      </xsl:message>
    </xsl:if>
    <xsl:if test="$tail!=''">
      <xsl:value-of select="$retval"/>
    </xsl:if>
  </xsl:template>
  ◊
```

Macro referenced in chunk 3.5

3.2.9 litprog: substparms template

This callable template takes a text string representing a chunk definition, and substitutes for any formal parameters used within the chunk. Formal parameters are found by virtue of them being enclosed within 4 sets of square brackets, as defined by the global variables `premacro` and `postmacro`.

Note that there is a further complication in that formal parameters nested inside actual parameters have been escaped, by using 4 sets of curly brackets. These escapings must be restored before parameter expansion can begin. This is done by substituting square brackets for the curly ones.

```
<litprog: substparms template 3.12> =
  <xsl:template name="substparms">
    <xsl:param name="text"></xsl:param>
    <xsl:param name="parmtable"></xsl:param>
    <substparms: unescape the escaped formal parameters 3.13>
    <xsl:variable name="head">
      <xsl:value-of select="substring-before($etext,$premacro)"
        disable-output-escaping="no"/>
```

```

</xsl:variable>
<xsl:variable name="tail">
  <xsl:value-of select="substring-after($etext,$postmacro)"
    disable-output-escaping="no"/>
</xsl:variable>
<xsl:variable name="parmname">
  <xsl:value-of select="substring-before(
    substring-after($etext,$premacro),
    $postmacro)"
    disable-output-escaping="no"/>
</xsl:variable>
<xsl:if test="$Verbose>=2">
  <xsl:message>
    <xsl:text>substparms: text=</xsl:text>
    <xsl:value-of select="$etext"/>
    <xsl:text>&#xa;substparms: parhtable=</xsl:text>
    <xsl:value-of select="$parhtable"/>
    <xsl:text>&#xa;substparms: parmname=</xsl:text>
    <xsl:value-of select="$parmname"/>
  </xsl:message>
</xsl:if>
<xsl:variable name="value">
  <xsl:call-template name="lookup">
    <xsl:with-param name="key" select="$parmname"/>
    <xsl:with-param name="table" select="$parhtable"/>
  </xsl:call-template>
</xsl:variable>
<xsl:if test="$Verbose>=2">
  <xsl:message>
    <xsl:text>substparms: </xsl:text>
    <xsl:value-of select="$parmname"/>
    <xsl:text>=</xsl:text>
    <xsl:value-of select="$value"/>
  </xsl:message>
</xsl:if>
<substparms: build result string recursively 3.14>
<xsl:if test="$Verbose>=2">
  <xsl:message>
    <xsl:text>substparms: result=</xsl:text>
    <xsl:value-of select="$result"/>
  </xsl:message>
</xsl:if>
  <xsl:value-of select="$result"/>
</xsl:template>

```

◇

Macro referenced in chunk 3.5

This is a challenging routine, as evidenced by the number of debug components built in! Basically, we split the incoming text string into a pre-formal parameter part (the **head**), and a post-formal parameter part (the **tail**), as indicated by the premacro and postmacro strings of square brackets. The formal parameter itself is looked up in the parameter table, and a result string is built by concatenating the pre-string, the parameter value, and a recursively expanded post-string.

```

<substparms: unescape the escaped formal parameters 3.13> =
  <xsl:variable name="etext">
    <xsl:call-template name="do-replace">

```

```

<xsl:with-param name="text">
  <xsl:call-template name="do-replace">
    <xsl:with-param name="text">
      <xsl:value-of select="$text"/>
    </xsl:with-param>
    <xsl:with-param name="replace">
      <xsl:text>{</xsl:text><xsl:text>{</xsl:text>
    </xsl:with-param>
    <xsl:with-param name="by">
      <xsl:value-of select="$premacro"/>
    </xsl:with-param>
  </xsl:call-template>
</xsl:with-param>
<xsl:with-param name="replace">
  <xsl:text>}}</xsl:text><xsl:text>}}</xsl:text>
</xsl:with-param>
<xsl:with-param name="by">
  <xsl:value-of select="$postmacro"/>
</xsl:with-param>
</xsl:call-template>
</xsl:variable> ◊

```

Macro referenced in chunk 3.12

We build a new version of the text string parameter in the variable `etext`. Two nested calls on the `do-replace` template are made, one to replace the premacro escape, and one to replace the postmacro escape.

<substparms: build result string recursively 3.14> =

```

<xsl:variable name="result">
  <xsl:choose>
    <xsl:when test="$head=''">
      <xsl:value-of select="$etext"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$head"/>
      <xsl:value-of select="$value"/>
      <xsl:if test="$tail!=''">
        <xsl:call-template name="substparms">
          <xsl:with-param name="text">
            <xsl:value-of select="$tail"/>
          </xsl:with-param>
          <xsl:with-param name="parmtable" select="$parmtable"/>
        </xsl:call-template>
      </xsl:if>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable> ◊

```

Macro referenced in chunk 3.12

Build the result of macro expansion recursively. The (first) macro parameter is chopped out, leaving a head and tail string. The parameter is looked up to find its replacement `value`, and the result returned is the `head` string, the replacement text `value`, and the recursive expansion of the `tail` string.

3.3 litprog: the tangle pass

```
<litprog: the tangle pass 3.15> =  
  <litprog: tangle all litprog files 3.16>  
  <litprog: tangle a code definition 3.18>  
  <litprog: tangle a single file definition chunk 3.17>  
  <litprog: tangle variable declarations 3.23>  
  <litprog: discard documentation in tangle mode 3.24>  
  <litprog: tangle all chunks into a single use 3.25>  
  <litprog: default tangle template 3.38>
```

◇

Macro referenced in chunk 3.1

3.3.1 litprog: tangle all litprog files

```
<litprog: tangle all litprog files 3.16> =  
  <xsl:template match="litprog" mode="tangle">  
    <xsl:for-each select="/litprog//o">1  
      <xsl:variable name="thisfile" select="@file"/>  
      <xsl:if test="not(preceding::o[@file=$thisfile])">2  
        <xsl:message>3  
          <xsl:text>Writing file "</xsl:text>  
          <xsl:value-of select="@file"/>  
          <xsl:text>"</xsl:text>  
        </xsl:message>  
        <xsl:document href="{thisfile}" omit-xml-declaration="yes"  
          indent="yes" method="text">4  
          <xsl:apply-templates select="." mode="tangle"/>5  
          <xsl:for-each  
            select="following::o[@file=$thisfile]">6  
            <xsl:apply-templates select="." mode="tangle"/>  
          </xsl:for-each>  
        </xsl:document>  
      </xsl:if>  
    </xsl:for-each>  
  </xsl:template>
```

◇

Macro referenced in chunk 3.15

Notes:

- 1) see point 1
- 2) see point 2
- 3) see point 3
- 4) see point 4
- 5) see point 5
- 6) see point 6

To tangle a complete litprog, we scan through all file chunk definitions, expanding them into an appropriately named file. Issue a message as each one is processed.

(version 2.4.0) There was a bug here: multiple chunks of “o” elements did not get appended, since each one started a new document, thus overwriting the previous chunk. Here the new logic is this:

- 1) Do all tangling by searching for “o” elements.
- 2) For each one found, check if it is the first one with this particular file name. If it isn't, ignore it.
- 3) If it is, issue a message that we are tangling the file, then
- 4) ... start a new document with this file name ...

- 5) ... tangle this chunk and ...
- 6) ... tangle all following “o” chunks with the same filename.

3.3.2 litprog: tangle a single file definition chunk

```
<litprog: tangle a single file definition chunk 3.17> =
  <xsl:template match="o" mode="tangle">
    <xsl:variable name="chunk">
      <xsl:apply-templates mode="tangle"/>
    </xsl:variable>
    <xsl:variable name="chunks">
      <xsl:choose>
        <xsl:when test='starts-with($chunk,"&#xA;")'>
          <xsl:value-of select="substring($chunk,2)"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="$chunk"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:value-of select="$chunks"/>
  </xsl:template>
  ◊
```

Macro referenced in chunk 3.15

This is done in a manner similar to code chunk tangling. Here, only the leading new line character is removed. Why? Good question. When I have an answer, I’ll add it to this documentation.

3.3.3 litprog: tangle a code definition

```
<litprog: tangle a code definition 3.18> =
  <xsl:template match="d" mode="tangle">
    <tangle code, debugging 3.22 Chunk omitted!>
    <tangle code, initial chunk 3.19>
    <tangle code, end trimmed chunk 3.20>
    <tangle code, start and end trimmed chunk 3.21>
    <xsl:if test="$Verbose&gt;1">
      <xsl:message>d/tangle returns <xsl:value-of select="$chunkes"/></xsl:message>
    </xsl:if>
    <xsl:value-of select="$chunkes"/>
  </xsl:template>
  ◊
```

Macro referenced in chunk 3.15

A code chunk is tangled in three stages.

Firstly, the chunk has any nested elements resolved and expanded, as appropriate. This expansion is saved in the local variable chunki (initial chunk).

```
<tangle code, initial chunk 3.19> =
  <xsl:variable name="chunki">
    <xsl:apply-templates mode="tangle"/>
  </xsl:variable> ◊
```

Macro referenced in chunk 3.18

Secondly, if the chunk ends in a newline character and the trim attribute is set to yes, the trailing newline is removed. The result of this operation is saved in the variable chunke (end trimmed chunk).

```
<tangle code, end trimmed chunk 3.20> =
```

```

<xsl:variable name="chunke">
  <xsl:choose>
    <xsl:when test='substring($chunki,string-length($chunki)) = "&#xA;"
                  and @trim="yes"'>
      <xsl:value-of select="substring($chunki,1,string-length($chunki)-1)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="$chunki"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable> ◊

```

Macro referenced in chunk 3.18

Thirdly, a leading newline character is removed, if present. The result of this operation is saved in the variable `chunkes` (start and end trimmed chunk).

```

<tangle code, start and end trimmed chunk 3.21> =
  <xsl:variable name="chunkes">
    <xsl:choose>
      <xsl:when test='starts-with($chunke,"&#xA;")'>
        <xsl:value-of select="substring($chunke,2)"/>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$chunke"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:variable> ◊

```

Macro referenced in chunk 3.18

The resultant value is returned as the result of tangling this code fragment or chunk.

```

<tangle code, debugging 3.22> =
  <xsl:message>
    <xsl:text>Depth of 'd' node = </xsl:text>
    <xsl:value-of select="count(ancestor::*)"/>
    <xsl:text></xsl:text>
  </xsl:message>
  <xsl:variable name="chunk">
    <xsl:for-each select="*/text()">
      <xsl:message><xsl:value-of select="name()"/></xsl:message>
      <xsl:apply-templates select="." mode="tangle"/>
    </xsl:for-each>
  </xsl:variable>
  <xsl:message><xsl:value-of select="$chunk"/></xsl:message>
  <xsl:variable name="chunki">
    <xsl:value-of select="$chunk"/>
    <xsl:call-template name="do-replace">
      <xsl:with-param name="text" select="$chunk"/>
      <xsl:with-param name="replace">=</xsl:with-param>
      <xsl:with-param name="by">:=</xsl:with-param>
    </xsl:call-template>
  </xsl:variable>
  ◊

```

Macro referenced in chunk 3.18

3.3.4 litprog: tangle variable declarations

```
<litprog: tangle variable declarations 3.23> =  
  <xsl:template match="d/v | o/v" mode="tangle">  
    <xsl:apply-templates mode="tangle"/>  
  </xsl:template>
```

◇

Macro referenced in chunk 3.15

In tangle mode, variable definitions and uses are basically passed straight through to the weft phase.

3.3.5 litprog: discard documentation in tangle mode

```
<litprog: discard documentation in tangle mode 3.24> =  
  <xsl:template match="p" mode="tangle">  
  </xsl:template>  
  <xsl:template match="d/com|o/com" mode="tangle">  
  </xsl:template>
```

◇

Macro referenced in chunk 3.15

All documentation fragments are ignored in tangle mode.

3.3.6 litprog: tangle all chunks into a single use

```
<litprog: tangle all chunks into a single use 3.25> =  
  <!-- collect all chunks with this name in tangle mode -->  
  <xsl:template match="u" mode="tangle">  
    <litprog: replace the machine parameter in chunk name 3.26>  
    <litprog: check whether to include a use chunk 3.27>  
    <litprog: save use parameters 3.28>  
    <litprog: compute indentations for use expansion 3.29>  
    <litprog: check definitions for non-use 3.30>  
    <litprog: include raw text and expand indentation 3.31>  
  </xsl:template>  
  <xsl:template match="formal" mode="tangle">  
    <xsl:value-of select="$premacro"/>  
    <xsl:value-of select="@name"/>  
    <xsl:value-of select="$postmacro"/>  
  </xsl:template>
```

◇

Macro referenced in chunk 3.15

A use call on a chunk name has been made, which must be expanded into the document-order concatenation of all code chunk definitions with the same name.

3.3.6.1 litprog: replace the machine parameter in chunk name

```
<litprog: replace the machine parameter in chunk name 3.26> =  
  <xsl:variable name="chunkname">  
    <xsl:call-template name="do-replace">  
      <xsl:with-param name="text" select="@name"/>  
      <xsl:with-param name="replace">{$machine}</xsl:with-param>  
      <xsl:with-param name="by" select="$machine"/>  
    </xsl:call-template>  
  </xsl:variable>
```

◇

Macro referenced in chunk 3.25

This bit is a hack. I really want a method to replace key strings with parameters, but there isn't anything I can think of other than using this "do-replace" with a hard-coded substitution string. Incidentally, do-replace is copied from Michael Kay's excellent book "XSLT Programmer's Reference".

3.3.6.2 litprog: check whether to include a use chunk

```
<litprog: check whether to include a use chunk 3.27> =
  <xsl:variable name="include">
    <xsl:choose>
      <xsl:when test="@include">
        <xsl:value-of select="@include"/>
      </xsl:when>
      <xsl:otherwise>yes</xsl:otherwise>
    </xsl:choose>
  </xsl:variable>
```

◇

Macro referenced in chunk 3.25

"include" is an attribute that allows the user to comment out chunks of code, simply by putting an 'include="no"' attribute into the chunk call. It defaults to "yes", so set up that default.

3.3.6.3 litprog: save use parameters

```
<litprog: save use parameters 3.28> =
  <xsl:variable name="actuals" select="*/>
```

◇

Macro referenced in chunk 3.25

3.3.6.4 litprog: compute indentations for use expansion

```
<litprog: compute indentations for use expansion 3.29> =
  <xsl:variable name="text-string">
    <xsl:value-of select="string(preceding-sibling::text()[1])"/>
  </xsl:variable>
  <xsl:variable name="count-blanks">
    <xsl:call-template name="count-trailing-blanks">
      <xsl:with-param name="string" select="$text-string"/>
      <xsl:with-param name="count">0</xsl:with-param>
    </xsl:call-template>
  </xsl:variable>
  <xsl:variable name="blanks">
    <xsl:value-of select="substring($text-string,
      string-length($text-string)-$count-blanks+1,
      $count-blanks)"/>
  </xsl:variable>
  <xsl:variable name="indent" select="$blanks"/>
```

◇

Macro referenced in chunk 3.25

Now some stuff to determine indentation. This is heavy XML. We first pull in the most recent text node in the tree. This will have the indentation blanks before this call on 'u', and so we count and extract those blanks.

Following that processing, the value of variable `\$blanks` will be inserted after all new lines in the text generated by this call on 'u'. That, together with the indentation that appeared before the call itself, will mean that all lines of the replacement text will have the same indentation, assuming that they are themselves aligned - any additional indentation is preserved.

3.3.6.5 litprog: check definitions for non-use

```
<litprog: check definitions for non-use 3.30> =  
  <xsl:if test="count(/litprog//d[@name=$chunkname])=0">  
    <xsl:message>  
      <xsl:text>No definition of &lt;</xsl:text>  
      <xsl:value-of select="$chunkname" disable-output-escaping="no"/>  
      <xsl:text>&gt;</xsl:text>  
    </xsl:message>  
  </xsl:if>
```

◇

Macro referenced in chunk 3.25

Now check that we have some definitions of this chunk. Count them and issue a warning if the count is zero.

3.3.6.6 litprog: include raw text and expand indentation

```
<litprog: include raw text and expand indentation 3.31> =  
  <litprog: include text and expand: start message 3.32>  
  <xsl:for-each select="/litprog//d[@name=$chunkname]">  
    <litprog: include text and expand: loop message 3.33>  
    <xsl:if test=" $include='yes'">  
      <litprog: include text and expand: define parmtree 3.34>  
      <litprog: include text and expand: define rawchunk 3.35>  
      <litprog: include text and expand: define raw2 3.36>  
      <litprog: include text and expand: replace text 3.37>  
    </xsl:if>  
  </xsl:for-each>
```

◇

Macro referenced in chunk 3.25

Firstly, cross reference this call on the chunk.

Then the real code expansion takes place. We collect up all matching chunk definitions, and add them in document order.

If the chunk is to be included, build the raw text of it, and then expand all newlines with indentation as computed above.

```
<litprog: include text and expand: start message 3.32> =  
  <xsl:if test="$Verbose&gt;1">  
    <xsl:message terminate="no">  
      <xsl:text>Using chunk named "</xsl:text>  
      <xsl:value-of select="$chunkname"/>  
      <xsl:text>" </xsl:text>  
      <xsl:for-each select="$actuals">  
        <xsl:value-of select="string(.)"/>  
      </xsl:for-each>  
    </xsl:message>  
  </xsl:if>
```

◇

Macro referenced in chunk 3.31

debug this call on expanding a chunk

```
<litprog: include text and expand: loop message 3.33> =  
  <xsl:if test="$Verbose&gt;1">  
    <xsl:message terminate="no">
```

```

    <xsl:text>Matches are "</xsl:text>
    <xsl:value-of select="@name" disable-output-escaping="no"/>
    <xsl:text>"</xsl:text>
  </xsl:message>
</xsl:if>

```

◇

Macro referenced in chunk 3.31

Show all the chunks that match this use instance.

<litprog: include text and expand: define parmtable 3.34 > =

```

<xsl:variable name="parmtable">
  <xsl:if test="$Verbose>=2">
    <xsl:message>
      <xsl:text>building parmtable,parms=</xsl:text>
      <xsl:value-of select="$actuals"/>
    </xsl:message>
  </xsl:if>
  <xsl:for-each select="$actuals">
    <xsl:value-of select="$premacro"/>
    <xsl:value-of select="./@name"/>
    <xsl:value-of select="$postmacro"/>
    <xsl:value-of select="$premacro"/>
    <xsl:call-template name="do-replace">
      <xsl:with-param name="text">
        <xsl:call-template name="do-replace">
          <xsl:with-param name="text">
            <xsl:apply-templates mode="tangle"/>
          </xsl:with-param>
          <xsl:with-param name="replace">
            <xsl:value-of select="$premacro"/>
          </xsl:with-param>
          <xsl:with-param name="by">
            <xsl:text>{</xsl:text><xsl:text>{</xsl:text>
          </xsl:with-param>
        </xsl:call-template>
      </xsl:with-param>
    <xsl:with-param name="replace">
      <xsl:value-of select="$postmacro"/>
    </xsl:with-param>
    <xsl:with-param name="by">
      <xsl:text>}</xsl:text><xsl:text>}</xsl:text>
    </xsl:with-param>
  </xsl:call-template>
  <xsl:value-of select="$postmacro"/>
</xsl:for-each>
</xsl:variable>
<xsl:if test="$Verbose>=2">
  <xsl:message>parmtable=<xsl:value-of select="$parmtable"/></xsl:message>
</xsl:if>

```

◇

Macro referenced in chunk 3.31

Build the parameter table. This is a string where each actual parameter is surrounded by the pre- and post- macro strings (as defined by the variables premacro and postmacro respectively), then concatenated together in parameter document order.

```
<litprog: include text and expand: define rawchunk 3.35> =
  <xsl:variable name="rawchunk">
    <xsl:apply-templates select="." mode="tangle"/>
  </xsl:variable>
```

◇

Macro referenced in chunk 3.31

Build the raw chunk corresponding to this use instance match.

```
<litprog: include text and expand: define raw2 3.36> =
  <xsl:variable name="raw2">
    <xsl:call-template name="substparms">
      <xsl:with-param name="text">
        <xsl:value-of select="$rawchunk"/>
      </xsl:with-param>
      <xsl:with-param name="parmtable" select="$parmtable"/>
    </xsl:call-template>
  </xsl:variable>
```

◇

Macro referenced in chunk 3.31

Now expand the actual parameters into the raw chunk, replacing all formal parameters.

```
<litprog: include text and expand: replace text 3.37> =
  <xsl:call-template name="do-replace">
    <xsl:with-param name="text">
      <xsl:value-of select="$raw2"/>
    </xsl:with-param>
    <xsl:with-param name="replace">
      <xsl:text>&#x0A;</xsl:text>
    </xsl:with-param>
    <xsl:with-param name="by">
      <xsl:text>&#x0A;</xsl:text>
      <xsl:value-of select="$indent"/>
    </xsl:with-param>
  </xsl:call-template>
```

◇

Macro referenced in chunk 3.31

Now rebuild the replacement text by expanding all newlines with their appropriate indentation. This is to preserve the indentation established in the actual call on this chunk.

3.3.7 litprog: default tangle template

```
<litprog: default tangle template 3.38> =
  <xsl:template match="*" mode="tangle">
    <xsl:variable name="check">
      <xsl:text>|</xsl:text>
      <xsl:value-of select="name()"/>
      <xsl:text>|</xsl:text>
    </xsl:variable>
    <xsl:if test="$Verbose and not(contains($docos,$check))">
      <xsl:message terminate="no">
        <xsl:text>No warp-tangle template for </xsl:text>
        <xsl:value-of select="name()"/>
      </xsl:message>
    </xsl:if>
    <xsl:element name="{name()}">
```

```

    <xsl:apply-templates mode="tangle"/>
  </xsl:element>
</xsl:template>

```

◇

Macro referenced in chunk 3.15

This is the default tangle template, invoked if there is no other template defined to handle this element. Provide an error message in the event of no tangle template.

3.4 litprog: the weave pass

```

<litprog: the weave pass 3.39> =
  <litprog: weave a litprog 3.40>
  <litprog: weave a code or file definition 3.41>
  <litprog: d/com—o/com-weave template 3.44>
  <litprog: weave a variable markup 3.45,3.46>
  <litprog: u-weave template 3.47,3.48>
  <litprog: p-weave template 3.49>
  <litprog: section-weave template 3.50>
  <litprog: listing templates 3.51>
  <litprog: default weave template 3.52>

```

◇

Macro referenced in chunk 3.1

3.4.1 litprog: weave a litprog

```

<litprog: weave a litprog 3.40> =
  <xsl:template match="litprog" mode="weave">
    <xsl:apply-templates mode="weave"/>
  </xsl:template>

```

◇

Macro referenced in chunk 3.39

Not really much to this, since the whole litprog file is scanned in document order, generating the intermediate documentation file as we go.

3.4.2 litprog: weave a code or file definition

```

<litprog: weave a code or file definition 3.41> =
  <xsl:template match="d | o" mode="weave">
    <xsl:variable name="chunkname">
      <xsl:choose>
        <xsl:when test="name()='d'">
          <xsl:value-of select="@name"/>
        </xsl:when>
        <xsl:when test="name()='o'">
          <xsl:value-of select="@file"/>
        </xsl:when>
      </xsl:choose>
    </xsl:variable>
    <xsl:element name="CODE">
      <xsl:attribute name="type">
        <xsl:choose>
          <xsl:when test="name()='d'">define</xsl:when>
          <xsl:when test="name()='o'">file</xsl:when>
        </xsl:choose>
      </xsl:attribute>
    </xsl:element>
  </xsl:template>

```

```

</xsl:attribute>
<xsl:attribute name="name">
  <xsl:value-of select="$chunkname"/>
</xsl:attribute>
<xsl:attribute name="number">
  <xsl:number format="1" level="multiple" count="section|DocumentHistory"/>
  <xsl:text>.</xsl:text>
  <xsl:number from="section" level="any" count="d|o"/>
</xsl:attribute>
<xsl:attribute name="defines">
  <xsl:call-template name="lookupchunkno">
    <xsl:with-param name="chunkname">
      <xsl:value-of select="$chunkname"/>
    </xsl:with-param>
  </xsl:call-template>
</xsl:attribute>
<xsl:attribute name="uses">1
  <xsl:variable name="defnode" select="."/>
  <xsl:for-each select="child::u">
    <xsl:if test="position()>1">
      <xsl:text>,</xsl:text>
    </xsl:if>
    <litprog: d-weave debug message-1 3.42 Chunk omitted!>
    <xsl:call-template name="lookupchunkno">
      <xsl:with-param name="chunkname" select="@name"/>
    </xsl:call-template>
  </xsl:for-each>
</xsl:attribute>
<xsl:apply-templates mode="weave"/>
</xsl:element>
<litprog: weave a code comment 3.43>
</xsl:template>

```

◇

Macro referenced in chunk 3.39

Notes:

- 1) Build the uses attribute list, by scanning all child u elements, and looking up their chunk numbers.

The d — o template, in the weave mode, builds a CODE element in the woven .xml file, CODE elements define code fragments. Such elements can have the following attributes:

type for a d fragment, the type is `define`, indicating a chunk definition. For an o fragment, the type is `file`.

name The name of the code chunk or fragment. For example, the name of the following code chunk is `litprog: d | o-weave template`. For a file fragment, the name attribute gives the name of the file.

number code chunks are numbered in sequence from the beginning, for cross referencing purposes.

```

<litprog: d-weave debug message-1 3.42> =
  <xsl:message>
    <xsl:text>Macro "</xsl:text>
    <xsl:value-of select="$defnode/@name"/>
    <xsl:text>" references "</xsl:text>
    <xsl:value-of select="./@name"/>
    <xsl:text>",</xsl:text>
    <xsl:call-template name="lookupchunkno">
      <xsl:with-param name="chunkname" select="@name"></xsl:with-param>

```

```

    </xsl:call-template>
</xsl:message>

```

◇

Macro referenced in chunk 3.41

Generate a helpful debug message to show what other fragments this code chunk uses.

```

<litprog: weave a code comment 3.43> =
  <xsl:if test='./com'>
    <xsl:element name='NOTES'>
      <xsl:for-each select="./com">1
        <xsl:element name="NOTE">
          <xsl:attribute name="HREF">
            <xsl:value-of select="position()"/>
          </xsl:attribute>
          <xsl:apply-templates mode="weave"/>
        </xsl:element>
      </xsl:for-each>
    </xsl:element>
  </xsl:if>

```

◇

Macro referenced in chunk 3.41

Notes:

- 1) Scan through all in-line comments (such as this one), and for each one, generate a NOTE element, cross referenced by the ordinal number of this comment within the chunk.

```

<litprog: d/com—o/com-weave template 3.44> =
  <xsl:template match="d/com | o/com" mode="weave">
    <xsl:variable name="mypos">
      <xsl:value-of select="count(preceding-sibling::com)+1"/>
    </xsl:variable>
    <xsl:element name="COMMENT">
      <xsl:attribute name="HREF">
        <xsl:value-of select="$mypos"/>
      </xsl:attribute>
    </xsl:element>
  </xsl:template>

```

◇

Macro referenced in chunk 3.39

This template, `d/com|o/com`, handles comments with code fragments in the weave pass. The comment itself is handled later, but for the moment, we just insert a forward reference to it as a link in the current code fragment. The reference is identified by the ordinal position of the comment within this code fragment.

3.4.3 litprog: weave a variable markup

The elements 'u' (use) and 'v' (value) both refer to 'd' (define) elements elsewhere in the document. The difference is that in weave mode 'u' inserts a hot link to the actual definition, whereas 'v' inserts the actual value defined in the 'd' element. This is equivalent to macro expansion, and is what happens to 'u' elements in tangle mode. In tangle mode, 'v' elements are ignored. In other words, 'v' elements are only relevant to document fragments.

```
<litprog: weave a variable markup 3.45> =
  <xsl:template match="v" mode="weave">
    <xsl:choose>
      <xsl:when test="@var='use' or not(@var)">
        <xsl:element name="VAR">
          <xsl:attribute name="VAR">use</xsl:attribute>
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:when>
      <xsl:otherwise>
        <xsl:element name="NAME">
          <xsl:value-of select="@name"/>
        </xsl:element>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

◇

Macro defined in chunk 3.45,3.46

Macro referenced in chunk 3.39

```
<litprog: weave a variable markup 3.46> =
  <!-- handle variable markup -->
  <xsl:template match="d/v | o/v" mode="weave">
    <xsl:element name="VAR">
      <xsl:attribute name="VAR">
        <xsl:choose>
          <xsl:when test="@var">
            <xsl:value-of select="@var"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:text>use</xsl:text>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:attribute>
      <xsl:if test="@name">
        <xsl:attribute name="NAME">
          <xsl:value-of select="@name"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>
```

◇

Macro defined in chunk 3.45,3.46

Macro referenced in chunk 3.39

3.4.4 litprog: u-weave template

The `u-weave` template is responsible for inserting a link to the definition of the referenced code fragment, in-line in the documentation. Note that at this stage, the `include` parameter is simply passed on: it is not interpreted as to whether the code fragment is actually included until the documentation construction phase.

```
<litprog: u-weave template 3.47> =
  <xsl:template match="u" mode="weave">
    <xsl:element name="USE">
      <xsl:attribute name="NAME">
        <xsl:value-of select="@name"/>
      </xsl:attribute>
      <xsl:if test="@expand">
        <xsl:attribute name="EXPAND">
          <xsl:value-of select="@expand"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:if test="@include">
        <xsl:attribute name="INCLUDE">
          <xsl:value-of select="@include"/>
        </xsl:attribute>
      </xsl:if>
      <xsl:for-each select="*">
        <xsl:choose>
          <xsl:when test="name()='name' or name()='expand'">
            </xsl:when>
          <xsl:when test="name()='formal'">
            <xsl:element name="formal">
              <xsl:attribute name="name">
                <xsl:value-of select="@name"/>
              </xsl:attribute>
              <xsl:value-of select="."/>
            </xsl:element>
          </xsl:when>
          <xsl:when test="name()='actual'">
            <xsl:element name="actual">
              <xsl:attribute name="name">
                <xsl:value-of select="@name"/>
              </xsl:attribute>
              <xsl:apply-templates mode="weave"/>
            </xsl:element>
          </xsl:when>
          <xsl:otherwise>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:for-each>
      </xsl:element>
      <xsl:if test="$Verbose>1">
        <xsl:message>
          <xsl:text>Macro "</xsl:text>
          <xsl:value-of select="normalize-space(@name)"/>
          <xsl:text>" referenced in weave mode in "</xsl:text>
          <xsl:value-of select="ancestor::d/@name"/>
          <xsl:text>"</xsl:text>
        </xsl:message>
      </xsl:if>
    </xsl:template>
```

◇

Macro defined in chunk 3.47,3.48

Macro referenced in chunk 3.39

```
<litprog: u-weave template 3.48> =
<xsl:template match="formal" mode="weave">
  <xsl:element name="formal">
    <xsl:attribute name="name">
      <xsl:value-of select="@name"/>
    </xsl:attribute>
    <xsl:apply-templates mode="weave"/>
  </xsl:element>
</xsl:template>
```

◇

Macro defined in chunk 3.47,3.48

Macro referenced in chunk 3.39

3.4.5 litprog: p-weave template

```
<litprog: p-weave template 3.49> =
<xsl:template match="p" mode="weave">
  <P><xsl:apply-templates mode="weave"/></P>
</xsl:template>
```

◇

Macro referenced in chunk 3.39

3.4.6 litprog: section-weave template

```
<litprog: section-weave template 3.50> =
<xsl:template match="section" mode="weave">
  <xsl:element name="section">
    <xsl:choose>
      <xsl:when test="@numbers">
        <xsl:attribute name="numbers">
          <xsl:value-of select="@numbers"/>
        </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="numbers">yes</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:choose>
      <xsl:when test="@newpage">
        <xsl:attribute name="newpage">
          <xsl:value-of select="@newpage"/>
        </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="newpage">yes</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:attribute name="xref">
      <xsl:value-of select="translate(./title,' ?():.,&#x2d;','')'"
        disable-output-escaping="no"/>
    </xsl:attribute>
    <xsl:apply-templates mode="weave"/>
  </xsl:element>
```

```

</xsl:template>
<xsl:template match="subsection" mode="weave">
  <xsl:element name="subsection">
    <xsl:choose>
      <xsl:when test="@numbers">
        <xsl:attribute name="numbers">
          <xsl:value-of select="@numbers"/>
        </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="numbers">no</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:choose>
      <xsl:when test="@newpage">
        <xsl:attribute name="newpage">
          <xsl:value-of select="@newpage"/>
        </xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="newpage">no</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:attribute name="xref">
      <xsl:value-of select="translate(./title,' ?():.,&#x2d;',',')"
        disable-output-escaping="no"/>
    </xsl:attribute>
    <xsl:apply-templates mode="weave"/>
  </xsl:element>
</xsl:template>
<xsl:template match="subsubsection" mode="weave">
  <xsl:element name="subsubsection">
    <xsl:attribute name="xref">
      <xsl:value-of select="translate(./title,' ?():.,&#x2d;',',')"
        disable-output-escaping="no"/>
    </xsl:attribute>
    <xsl:apply-templates mode="weave"/>
  </xsl:element>
</xsl:template>

```

◇

Macro referenced in chunk 3.39

3.4.7 litprog: listing templates

```

<litprog: listing templates 3.51> =
<xsl:template match="filelist" mode="weave">
  <xsl:text>&#xA;</xsl:text>
  <xsl:element name="filelist">
    </xsl:element>
</xsl:template>
<xsl:template match="macrolist" mode="weave">
  <xsl:text>&#xA;</xsl:text>
  <xsl:element name="macrolist">
    <xsl:if test="@namewidth">
      <xsl:attribute name="namewidth">
        <xsl:value-of select="@namewidth"/>
      </xsl:attribute>
    </xsl:if>
  </xsl:element>

```

```

</xsl:if>
<xsl:if test="@definewidth">
  <xsl:attribute name="definewidth">
    <xsl:value-of select="@definewidth"/>
  </xsl:attribute>
</xsl:if>
<xsl:if test="@usewidth">
  <xsl:attribute name="usewidth">
    <xsl:value-of select="@usewidth"/>
  </xsl:attribute>
</xsl:if>
</xsl:element>
</xsl:template>
<xsl:template match="identifierlist" mode="weave">
  <xsl:text>&#xA;</xsl:text>
  <xsl:element name="IdentifierDefinitions">
    <xsl:if test="@namewidth">
      <xsl:attribute name="namewidth">
        <xsl:value-of select="@namewidth"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="@definewidth">
      <xsl:attribute name="definewidth">
        <xsl:value-of select="@definewidth"/>
      </xsl:attribute>
    </xsl:if>
    <xsl:if test="@usewidth">
      <xsl:attribute name="usewidth">
        <xsl:value-of select="@usewidth"/>
      </xsl:attribute>
    </xsl:if>
  </xsl:element>
</xsl:template>

```

◇

Macro referenced in chunk 3.39

3.4.8 litprog: default weave template

```

<litprog: default weave template 3.52> =
<xsl:template match="*" mode="weave">
  <xsl:variable name="check">
    <xsl:text>|</xsl:text>
    <xsl:value-of select="name()"/>
    <xsl:text>|</xsl:text>
  </xsl:variable>
  <xsl:if test="$Verbose>0 and not(contains($docos,$check))">
    <xsl:message terminate="no">
      <xsl:text>No warp-weave template for </xsl:text>
      <xsl:value-of select="name()"/>
    </xsl:message>
  </xsl:if>
  <xsl:element name="{name()}">
    <xsl:for-each select="@*">
      <xsl:attribute name="{name()}">
        <xsl:value-of select="."/>
      </xsl:attribute>
    </xsl:for-each>

```

```
<xsl:apply-templates mode="weave"/>
</xsl:element>
</xsl:template>
```

◇

Macro referenced in chunk 3.39

This template handles all elements not elsewhere handled in the weave pass. Note that a table of documentation related elements is kept, and if an element name is matched in this table, no warning message is issued. All elements, whether flagged or not, are passed straight through. Note that all the attributes must be copied as well.

In performing the table lookup, note that the separator must be present at both ends. This is to ensure that names that are substrings are not matched implicitly.

4 The weft, or documentation phase

This phase retranslates the output file from phase 1, the XML file. There is a translation script for each class of documentation processors. Two are defined here, `lit2html.xsl`, for translation to HTML, and `lit2tex.xsl`, for translation to \TeX .

4.1 `lit2html.xsl`

```
"lit2html.xsl" 4.1 =
<?xml version="1.0"?>
<!DOCTYPE xsl:transform PUBLIC "-//MONASH-CSSE//DTD stylesheet 1.0//EN"
    "file:///home/ajh/lib/dtd/stylesheet.dtd">
<!-- This is for version 2 -->
<xsl:transform
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xinclude="http://www.w3.org/2001/XInclude"
  >
  <xsl:output method="html" encoding="ASCII"/>
  <xsl:include href="LIBXSL 1.1"/numbered2html.xsl"/>
  <xsl:include href="LIBXSL 1.1"/basic2html.xsl"/>
  <xsl:include href="LIBXSL 1.1"/biblio2html.xsl"/>
  <xsl:include href="LIBXSL 1.1"/tables2html.xsl"/>
  <xsl:param name="Verbose">1</xsl:param>
  <lit2html: handle the root element 4.2>
  <lit2html: handle the literate program translation 4.3>
  <lit2html: handle documentation fragments 4.4,4.5,4.6,4.7,4.8,4.9,4.10>
  <lit2html: handle code fragment references 4.11,4.12>
  <lit2html: handle a code fragment use 4.13>
  <lit2html: handle code chunk definitions 4.14>
  <lit2html: handle notes and comments in code fragments 4.21>
  <lit2html: handle variable definitions and uses 4.22>
  <lit2html: handle maintaintable 4.23>
  <lit2html: TableOfContents 4.24>
  <xsl:template match="br"><BR/></xsl:template>
  <lit2html: handle a code text element 4.25>
  <lit2html: construct file definition list 4.26>
  <lit2html: construct chunk definition list 4.27>
  <lit2html: construct identifier definition list 4.28>
  <lit2html: handle macro parameters 4.29>
  <lit2html: handle document history 4.30>
  <lit2html: special character templates 4.31>
  <lit2html: default template 4.32>
</xsl:transform>
◇
```

4.1.1 lit2html: handle the root element

```
<lit2html: handle the root element 4.2> =  
  <weft: handle the root element 5.1> weft root pre text=  
  <HTML><BODY>  
  ', weft root post text=  
  </BODY></HTML>  
  ,  
  ◊  
  Macro referenced in chunk 4.1
```

4.1.2 lit2html: handle the literate program translation

```
<lit2html: handle the literate program translation 4.3> =  
  <xsl:template match="LitprogXML">  
    <xsl:apply-templates/>  
  </xsl:template>  
  ◊  
  Macro referenced in chunk 4.1
```

4.1.3 lit2html: handle documentation fragments

```
<lit2html: handle documentation fragments 4.4> =  
  <xsl:template match="abstract">  
    <H2 ALIGN="center">Abstract</H2>  
    <DIV STYLE="margin-left:20pt">  
      <xsl:apply-templates/>  
    </DIV>  
  </xsl:template>  
  ◊  
  Macro defined in chunk 4.4,4.5,4.6,4.7,4.8,4.9,4.10  
  Macro referenced in chunk 4.1
```

The abstract gets a title, and indentation

```
<lit2html: handle documentation fragments 4.5> =  
  <xsl:template match="author">  
    <H2 ALIGN="center"><xsl:apply-templates/></H2>  
  </xsl:template>  
  ◊  
  Macro defined in chunk 4.4,4.5,4.6,4.7,4.8,4.9,4.10  
  Macro referenced in chunk 4.1
```

Centre the author in heading font

```
<lit2html: handle documentation fragments 4.6> =  
  <xsl:template match="date">  
    <H2 ALIGN="center"><xsl:apply-templates/></H2>  
  </xsl:template>  
  ◊  
  Macro defined in chunk 4.4,4.5,4.6,4.7,4.8,4.9,4.10  
  Macro referenced in chunk 4.1
```

Ditto the date

```
<lit2html: handle documentation fragments 4.7> =  
  <xsl:template match='title'>
```

```
<H1 ALIGN="center"><xsl:apply-templates/></H1>
</xsl:template>
```

◇

Macro defined in chunk 4.4,4.5,4.6,4.7,4.8,4.9,4.10
Macro referenced in chunk 4.1

The title is a bit more important

```
<lit2html: handle documentation fragments 4.8> =
<xsl:template match='subtitle'>
  <H2 ALIGN="center"><xsl:apply-templates/></H2>
</xsl:template>
```

◇

Macro defined in chunk 4.4,4.5,4.6,4.7,4.8,4.9,4.10
Macro referenced in chunk 4.1

The subtitle is less important

```
<lit2html: handle documentation fragments 4.9> =
<xsl:template match="version">
  <H3 ALIGN="center">
    <xsl:text>Version </xsl:text>
    <xsl:apply-templates/>
  </H3>
</xsl:template>
```

◇

Macro defined in chunk 4.4,4.5,4.6,4.7,4.8,4.9,4.10
Macro referenced in chunk 4.1

Label the version

```
<lit2html: handle documentation fragments 4.10> =
<xsl:template match="P">
  <P><xsl:apply-templates/></P>
</xsl:template>
```

◇

Macro defined in chunk 4.4,4.5,4.6,4.7,4.8,4.9,4.10
Macro referenced in chunk 4.1

Build documentation paragraphs

4.1.4 lit2html: handle code fragment references

```
<lit2html: handle code fragment references 4.11> =
<xsl:template match="VALUE">
  <xsl:variable name="myname" select="."/>
  <xsl:value-of select="//CODE[@name=$myname]"/>
</xsl:template>
```

◇

Macro defined in chunk 4.11,4.12
Macro referenced in chunk 4.1

I think this one is out-of-date. Check if VALUE elements are ever created by <>

```
<lit2html: handle code fragment references 4.12> =
<xsl:template match="NAME">
  <xsl:variable name="myname" select="."/>
  <xsl:element name="A">
    <xsl:attribute name="HREF">
```

```

    <xsl:text>#</xsl:text>
    <xsl:value-of select="$myname"/>
  </xsl:attribute>
  <xsl:value-of select="$myname"/>
</xsl:element>
</xsl:template>

```

◇

Macro defined in chunk 4.11,4.12

Macro referenced in chunk 4.1

Not sure about this one either.

4.1.5 lit2html: handle a code fragment use

<lit2html: handle a code fragment use 4.13> =

```

<xsl:template match="USE">
  <!--<xsl:text disable-output-escaping="yes">&lt;I&gt;</xsl:text-->1
  <xsl:variable name="myname" select="@NAME"/>
  <xsl:choose>
    <xsl:when test="@EXPAND">
      <xsl:value-of select="//CODE[@name=$myname]"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:element name="I">
        <xsl:element name="A">
          <xsl:attribute name="HREF">#<xsl:value-of select="@NAME"/>
        </xsl:attribute>
        <xsl:text disable-outputsescaping="yes">&lt;</xsl:text>
        <xsl:value-of select="@NAME"/>
        <xsl:text disable-outputsescaping="yes"> </xsl:text>
        <xsl:for-each select="//CODE[@name=$myname]">
          <xsl:if test="position()>1"><xsl:text>,</xsl:text></xsl:if>
          <xsl:value-of select="@number"/>
        </xsl:for-each>
        <xsl:text disable-outputsescaping="yes">&gt;</xsl:text>
      </xsl:element>
    </xsl:element>
    <xsl:if test="actual">
      <xsl:element name="span">
        <xsl:attribute name="style">color:brown</xsl:attribute>
        <xsl:text></xsl:text>
        <xsl:for-each select="actual">
          <xsl:if test="position()>1">
            <xsl:text>,</xsl:text>
          </xsl:if>
          <xsl:value-of select="@name"/>
          <xsl:text>='</xsl:text>
          <xsl:element name="span">
            <xsl:attribute name="style">color:green</xsl:attribute>
            <xsl:apply-templates/>
          </xsl:element>
          <xsl:text>'</xsl:text>
        </xsl:for-each>
        <xsl:text></xsl:text>
      </xsl:element>
    </xsl:if>
  <xsl:if test="formal">

```

```

        <xsl:text>formal, name=</xsl:text>
        <xsl:value-of select="@name"/>
    </xsl:if>
    <xsl:if test="@INCLUDE='no'">
        <xsl:element name="SPAN">
            <xsl:attribute name="STYLE">color:red</xsl:attribute>
            <xsl:text> **** Chunk omitted!</xsl:text>
        </xsl:element>
    </xsl:if>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

```

◇

Macro referenced in chunk 4.1

Notes:

- 1) I generate the I tag literally here because the use of xsl:element generates an I element with a leading newline - NOT what is wanted here!

4.1.6 lit2html: handle code chunk definitions

```

<lit2html: handle code chunk definitions 4.14> =
<xsl:template match="CODE">
  <!-- Issue anchor to bind by name of chunk -->
  <xsl:element name="A">
    <xsl:attribute name="NAME">
      <xsl:value-of select="@name"/>
    </xsl:attribute>
  </xsl:element>
  <!-- Issue anchor to bind by number of chunk -->
  <xsl:element name="A">
    <xsl:attribute name="NAME">
      <xsl:text>xlp:</xsl:text>
      <xsl:value-of select="@number"/>
    </xsl:attribute>
  </xsl:element>
  <!-- Handle file chunk -->
  <xsl:if test="@type='file'">
    <xsl:text disable-output-escaping="yes">"</xsl:text>
    <xsl:element name="a">
      <xsl:attribute name="href">
        <xsl:value-of select="@name"/>
      </xsl:attribute>
      <xsl:value-of select="@name"/>
    </xsl:element>
    <xsl:text disable-output-escaping="yes">"</xsl:text>
    <xsl:text disable-output-escaping="yes"> </xsl:text>
    <xsl:value-of select="@number"/>
    <xsl:text disable-output-escaping="yes"> =</xsl:text>
  </xsl:if>
  <!-- Handle code chunk -->
  <xsl:if test="@type='define'">
    <xsl:text disable-output-escaping="no">&lt;</xsl:text>
    <xsl:element name="SPAN">
      <xsl:attribute name="STYLE">
        <xsl:text>color:purple</xsl:text>
      </xsl:attribute>

```

```

    <xsl:value-of select="@name"/>
    <xsl:text disable-output-escaping="no"> </xsl:text>
    <xsl:value-of select="@number"/>
  </xsl:element>
  <xsl:text disable-output-escaping="no">&gt; =</xsl:text>
</xsl:if>
<!-- Format the code body, including cross references -->
<lit2html: format the code body 4.15>
<lit2html: build code chunk reference list 4.18>
<DIV STYLE="margin-left:30pt">
  <lit2html: display code chunk reference list 4.19>
  <lit2html: build code chunk definition list 4.20>
</DIV>
</xsl:template>

```

◇

Macro referenced in chunk 4.1

```

<lit2html: format the code body 4.15> =
<xsl:choose>
  <xsl:when test="<lit2html: body contains new line characters 4.17">">
    <xsl:element name="DIV">
      <xsl:attribute name="STYLE" >
        margin-left:30pt;white-space:pre;font-family:monospace;
        border:thin solid;background:pink
      </xsl:attribute>
      <xsl:apply-templates mode="stripleadingnl"/>
    </xsl:element>
  </xsl:when>
  <xsl:otherwise>
    <xsl:text> </xsl:text>
    <xsl:element name="SPAN">
      <xsl:attribute name="STYLE" >
        white-space:pre;font-family:monospace;
        border:thin solid;background:pink
      </xsl:attribute>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:otherwise>
</xsl:choose>

```

◇

Macro referenced in chunk 4.14

There are two basic choices for the code body. Short text only fragments containing no new lines are formatted directly onto the definition line, to emphasize the fact that no new lines are part of the code fragment. Code bodies containing more than one element *<lit2html: body contains several children >* (as when a macro call is expanded within the body), or containing at least one new line *<lit2html: body contains new line characters >*, are formatted into a separate paragraph.

```
<lit2html: body contains several children 4.16> =count(child:*)>0 ◇
```

```
<lit2html: body contains new line characters 4.17> =contains(.,'&#xA;') ◇
```

Macro referenced in chunk 4.15

Two examples for the last paragraph! Note that the second half of the condition can be explained the same way, even though it contains a new line character, since the new line is escaped.

```

<lit2html: build code chunk reference list 4.18> =
  <xsl:variable name="thisname" select="@name"/>

```

```

<xsl:variable name="reflist">
  <xsl:for-each select="//CODE">
    <xsl:variable name="refnumber" select="@number"/>
    <xsl:for-each select="descendant::USE[@NAME=$thisname]">
      <xsl:text> </xsl:text>
      <xsl:element name="A">
        <xsl:attribute name="HREF">
          <xsl:text>#xlp:</xsl:text>
          <xsl:value-of select="$refnumber"/>
        </xsl:attribute>
        <xsl:value-of select="$refnumber"/>
      </xsl:element>
    </xsl:for-each>
  </xsl:for-each>
</xsl:variable>

```

◇

Macro referenced in chunk 4.14

Build a list of chunks in which this code chunk is referenced, and store it in the variable `reflist`.

<lit2html: display code chunk reference list 4.19> =

```

<xsl:if test="$reflist!=''">
  <xsl:element name="i">Chunk referenced in </xsl:element>
  <xsl:for-each select="//CODE">
    <xsl:variable name="refnumber" select="@number"/>
    <xsl:if test="descendant::USE[@NAME=$thisname]">
      <xsl:text> </xsl:text>
      <xsl:element name="A">
        <xsl:attribute name="HREF">
          <xsl:text>#xlp:</xsl:text>
          <xsl:value-of select="$refnumber"/>
        </xsl:attribute>
        <xsl:value-of select="$refnumber"/>
      </xsl:element>
    </xsl:if>
  </xsl:for-each>
  <xsl:element name="br"></xsl:element>
</xsl:if>

```

◇

Macro referenced in chunk 4.14

If the variable `reflist` is not empty, display the list of chunk numbers for each of the chunk names in the list. Only unique numbers need be displayed, hence the nested if statement within the `//CODE` for-each, rather than another for-each which would itemize repeated uses.

<lit2html: build code chunk definition list 4.20> =

```

<xsl:if test="count(//CODE[@name=$thisname])>1">
  <xsl:element name="i">Chunk defined in </xsl:element>
  <xsl:for-each select="//CODE[@name=$thisname]">
    <xsl:if test="position()>1"><xsl:text>,</xsl:text></xsl:if>
    <xsl:element name="A">
      <xsl:attribute name="HREF">
        <xsl:text>#xlp:</xsl:text>
        <xsl:value-of select="@number"/>
      </xsl:attribute>
      <xsl:value-of select="@number"/>
    </xsl:element>
  </xsl:for-each>

```

```

    <xsl:element name="br"></xsl:element>
</xsl:if>

```

◇

Macro referenced in chunk 4.14

Build a list of all the chunk numbers that also define this chunk (that is, they have the same name). All of these chunks are concatenated in document order to build the resultant code chunk.

4.1.7 lit2html: handle notes and comments in code fragments

<lit2html: handle notes and comments in code fragments 4.21> =

```

<xsl:template match="COMMENT">
  <xsl:variable name="notenum">
    <xsl:value-of select="ancestor::CODE/@number"/>
    <xsl:text>.</xsl:text>
    <xsl:value-of select="@HREF"/>
  </xsl:variable>
  <xsl:element name="A">
    <xsl:attribute name="HREF">
      <xsl:text>#xlp-note:</xsl:text>
      <xsl:value-of select="$notenum"/>
    </xsl:attribute>
    <xsl:element name="SPAN">
      <xsl:attribute name="STYLE">
        <xsl:text>color:green</xsl:text>
      </xsl:attribute>
      <xsl:text>{Note </xsl:text>
      <xsl:value-of select="$notenum"/>
      <xsl:text>}</xsl:text>
    </xsl:element>
  </xsl:element>
</xsl:template>
<xsl:template match="NOTES">
  <xsl:element name="DL">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
<xsl:template match="NOTE">
  <xsl:variable name="notenum">
    <xsl:value-of select="../preceding-sibling::CODE[1]/@number"/>
    <xsl:text>.</xsl:text>
    <xsl:value-of select="@HREF"/>
  </xsl:variable>
  <xsl:element name="DT">
    <xsl:element name="A">
      <xsl:attribute name="NAME">
        <xsl:text>xlp-note:</xsl:text>
        <xsl:value-of select="$notenum"/>
      </xsl:attribute>
      <xsl:element name="SPAN">
        <xsl:attribute name="STYLE">
          <xsl:text>color:green</xsl:text>
        </xsl:attribute>
        <xsl:text>{Note </xsl:text>
        <xsl:value-of select="$notenum"/>
        <xsl:text>}</xsl:text>
      </xsl:element>
    </xsl:element>
  </xsl:element>

```

```

    </xsl:element>
  </xsl:element>
  <xsl:element name="DD">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>

```

◇

Macro referenced in chunk 4.1

4.1.8 lit2html: handle variable definitions and uses

<lit2html: handle variable definitions and uses 4.22> =

```

<xsl:template match="VAR">
  <xsl:choose>
    <xsl:when test="@VAR='def'">
      <xsl:element name="A">
        <xsl:attribute name="NAME">
          <xsl:text>VAR-</xsl:text>
        <xsl:choose>
          <xsl:when test="@NAME">
            <xsl:value-of select="@NAME"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="."/>
          </xsl:otherwise>
        </xsl:choose>
        </xsl:attribute>
        <xsl:attribute name="STYLE">color:red</xsl:attribute>
        <xsl:apply-templates/>
      </xsl:element>
    </xsl:when>
    <xsl:when test="@VAR='use'">
      <xsl:element name="A">
        <xsl:attribute name="HREF">
          <xsl:text>#</xsl:text>
          <xsl:text>VAR-</xsl:text>
          <xsl:value-of select="."/>
        </xsl:attribute>
        <xsl:attribute name="STYLE">color:red</xsl:attribute>
        <xsl:apply-templates/>
      </xsl:element>
    </xsl:when>
  </xsl:choose>
</xsl:template>

```

◇

Macro referenced in chunk 4.1

Variable cross referencing generates links between the variable use and definition points. A definition generates an anchor of the form “VAR-*variable name*”, and a use generates a link to the anchor similarly formed. These links and anchors are highlighted in red.

4.1.9 lit2html: handle maintaintable

```
<lit2html: handle maintaintable 4.23> =
  <xsl:template match="maintaintable">
    <TABLE BORDER="1" ALIGN="center">
      <xsl:apply-templates select="./*/">
    </TABLE>
    <P>
      <xsl:text disable-output-escaping="yes">&nbsp;</xsl:text>
    </P>
  </xsl:template>
  <xsl:template match="maintaintable/date">
    <xsl:text disable-output-escaping="yes">&lt;TR&gt;</xsl:text>
    <TD>
      <xsl:apply-templates/>
    </TD>
  </xsl:template>
  <xsl:template match="maintaintable/author">
    <TD>
      <xsl:apply-templates/>
    </TD>
  </xsl:template>
  <xsl:template match="maintaintable/version">
    <TD>
      <xsl:apply-templates/>
    </TD>
  </xsl:template>
  <xsl:template match="maintaintable/comment">
    <TD>
      <xsl:apply-templates/>
    </TD>
    <xsl:text disable-output-escaping="yes">&lt;/TR&gt;</xsl:text>
  </xsl:template>
```

◇

Macro referenced in chunk 4.1

4.1.10 lit2html: TableOfContents

TableOfContents builds a table of contents by scanning all sections, subsections and subsubsections, using their ordinal positions to construct a stepped list of section numbers and titles

```
<lit2html: TableOfContents 4.24> =
  <xsl:template match="TableOfContents">
    <xsl:element name="H2">
      <xsl:attribute name="ALIGN">left</xsl:attribute>
      <xsl:text>Table of Contents</xsl:text>
    </xsl:element>
    <xsl:element name="TABLE">
      <xsl:attribute name="ALIGN">left</xsl:attribute>
      <xsl:for-each select="/LitprogXML/section">
        <xsl:variable name="titlestring">
          <xsl:value-of select="translate(./title,' ?','')"
            disable-output-escaping="no"/>
        </xsl:variable>
        <xsl:element name="TR">
          <xsl:element name="TD">
            <xsl:attribute name="ALIGN">right</xsl:attribute>
```

```

        <xsl:number select="position()"/>
    </xsl:element>
    <xsl:element name="TD">
        <xsl:attribute name="COLSPAN">3</xsl:attribute>
        <xsl:element name="A">
            <xsl:attribute name="HREF">
                <xsl:text>#</xsl:text>
                <xsl:value-of select="$titlestring"/>
            </xsl:attribute>
            <xsl:value-of select="title"/>
        </xsl:element>
    </xsl:element>
</xsl:element>
<xsl:for-each select="subsection">
    <xsl:element name="TR">
        <xsl:element name="TD"></xsl:element>
        <xsl:element name="TD">
            <xsl:value-of select="count(..preceding-sibling::section)+1"/>
            <xsl:text>.</xsl:text>
            <xsl:value-of select="position()"/>
        </xsl:element>
        <xsl:element name="TD">
            <xsl:attribute name="COLSPAN">2</xsl:attribute>
            <xsl:element name="A">
                <xsl:attribute name="HREF">
                    <xsl:text>#</xsl:text>
                    <xsl:value-of select="./title"/>
                </xsl:attribute>
                <xsl:value-of select="title"/>
            </xsl:element>
        </xsl:element>
    </xsl:element>
</xsl:for-each select="subsubsection">
    <xsl:element name="TR">
        <xsl:element name="TD"></xsl:element>
        <xsl:element name="TD"></xsl:element>
        <xsl:element name="TD">
            <xsl:value-of select="count(..../preceding-sibling::section)+1"/>
            <xsl:text>.</xsl:text>
            <xsl:value-of select="count(..preceding-sibling::subsection)+1"/>
            <xsl:text>.</xsl:text>
            <xsl:value-of select="position()"/>
        </xsl:element>
        <xsl:element name="TD">
            <xsl:element name="A">
                <xsl:attribute name="HREF">
                    <xsl:text>#</xsl:text>
                    <xsl:value-of select="./title"/>
                </xsl:attribute>
                <xsl:value-of select="title"/>
            </xsl:element>
        </xsl:element>
    </xsl:element>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:element>

```

```

    <xsl:element name="BR">
      <xsl:attribute name="CLEAR">all</xsl:attribute>
    </xsl:element>
    <xsl:element name="HR"></xsl:element>
  </xsl:template>

```

◇

Macro referenced in chunk 4.1

4.1.11 lit2html: handle a code text element

```

<lit2html: handle a code text element 4.25> =
  <xsl:template match="c">
    <xsl:element name="tt">
      <xsl:attribute name="style">color:red</xsl:attribute>
      <xsl:apply-templates/>
    </xsl:element>
  </xsl:template>

```

◇

Macro referenced in chunk 4.1

4.1.12 lit2html: construct file definition list

```

<lit2html: construct file definition list 4.26> =
  <xsl:template match="filelist">
    <xsl:element name="TABLE">
      <xsl:element name="TR">
        <xsl:element name="TH">File Name</xsl:element>
        <xsl:element name="TH">Defined in</xsl:element>
      </xsl:element>
      <xsl:for-each select="//CODE">
        <xsl:sort select="./@name"/>
        <xsl:variable name="thispos" select="position()"/>
        <xsl:variable name="thisname" select="./@name"/>
        <xsl:variable name="prevname" select="preceding::CODE[1]/@name"/>
        <xsl:if test="(not($prevname) or $thisname!=$prevname) and @type='file'">
          <xsl:element name="TR">
            <xsl:element name="TD">
              <xsl:value-of select="./@name"/>
            </xsl:element>
            <xsl:element name="TD">
              <xsl:attribute name="ALIGN">left</xsl:attribute>
              <xsl:for-each select="//CODE[@name=$thisname]">
                <xsl:if test="position()>1">
                  <xsl:text>, </xsl:text>
                </xsl:if>
                <xsl:element name="A">
                  <xsl:attribute name="HREF">
                    <xsl:text>#xlp:</xsl:text>
                    <xsl:value-of select="./@number"/>
                  </xsl:attribute>
                  <xsl:value-of select="./@number"/>
                </xsl:element>
              </xsl:for-each>
            </xsl:element>
          </xsl:if>
        </xsl:for-each>
      </xsl:element>
    </xsl:template>

```

```

    </xsl:element>
</xsl:template>

```

◇

Macro referenced in chunk 4.1

4.1.13 lit2html: construct chunk definition list

```

<lit2html: construct chunk definition list 4.27> =
<xsl:template match="macrolist">
  <xsl:element name="TABLE">
    <xsl:element name="TR">
      <xsl:element name="TH">Chunk Name</xsl:element>
      <xsl:element name="TH">Defined in</xsl:element>
      <xsl:element name="TH">Used in</xsl:element>
    </xsl:element>
    <xsl:for-each select="//CODE">
      <xsl:sort select="./@name"/>
      <xsl:variable name="thispos" select="position()"/>
      <xsl:variable name="thisname" select="./@name"/>
      <xsl:variable name="prevname" select="preceding::CODE[1]/@name"/>
      <xsl:if test="$thisname!=$prevname and @type='define'">
        <xsl:element name="TR">
          <xsl:element name="TD">
            <xsl:value-of select="./@name"/>
          </xsl:element>
          <xsl:element name="TD">
            <xsl:attribute name="ALIGN">left</xsl:attribute>
            <xsl:for-each select="//CODE[@name=$thisname]">
              <xsl:if test="position()>1">
                <xsl:text>, </xsl:text>
              </xsl:if>
              <xsl:element name="A">
                <xsl:attribute name="HREF">
                  <xsl:text>#xlp:</xsl:text>
                  <xsl:value-of select="./@number"/>
                </xsl:attribute>
                <xsl:value-of select="./@number"/>
              </xsl:element>
            </xsl:for-each>
          </xsl:element>
          <xsl:element name="TD">
            <xsl:attribute name="ALIGN">left</xsl:attribute>
            <xsl:for-each select="//CODE/USE[@NAME=$thisname]">
              <xsl:if test="position()>1">
                <xsl:text>, </xsl:text>
              </xsl:if>
              <xsl:element name="A">
                <xsl:attribute name="HREF">
                  <xsl:text>#xlp:</xsl:text>
                  <xsl:value-of select="ancestor::CODE/@number"/>
                </xsl:attribute>
                <xsl:value-of select="ancestor::CODE/@number"/>
              </xsl:element>
            </xsl:for-each>
          </xsl:element>
        </xsl:element>
      </xsl:if>

```

```

    </xsl:for-each>
  </xsl:element>
</xsl:template>

```

◇

Macro referenced in chunk 4.1

4.1.14 lit2html: construct identifier definition list

```

<lit2html: construct identifier definition list 4.28> =
<xsl:template match="IdentifierDefinitions">
  <xsl:element name="TABLE">
    <xsl:element name="TR">
      <xsl:element name="TH">Identifier</xsl:element>
      <xsl:element name="TH">Defined in</xsl:element>
      <xsl:element name="TH">Used in</xsl:element>
    </xsl:element>
    <xsl:for-each select="//VAR[@VAR='def']">
      <xsl:sort select="."/>
      <xsl:variable name="name" select="."/>
      <xsl:element name="TR">
        <xsl:element name="TD">
          <xsl:value-of select="."/>
        </xsl:element>
        <xsl:element name="TD">
          <xsl:attribute name="ALIGN">left</xsl:attribute>
          <xsl:element name="A">
            <xsl:attribute name="HREF">
              <xsl:text>#xlp:</xsl:text>
              <xsl:value-of select="ancestor::CODE/@number"/>
            </xsl:attribute>
            <xsl:value-of select="ancestor::CODE/@number"/>
          </xsl:element>
        </xsl:element>
        <xsl:element name="TD">
          <xsl:attribute name="ALIGN">left</xsl:attribute>
          <xsl:for-each select="//CODE/VAR[@VAR='use' and string()=$name]">
            <xsl:if test="position()>1">
              <xsl:text>, </xsl:text>
            </xsl:if>
            <xsl:element name="A">
              <xsl:attribute name="HREF">
                <xsl:text>#xlp:</xsl:text>
                <xsl:value-of select="ancestor::CODE/@number"/>
              </xsl:attribute>
              <xsl:value-of select="ancestor::CODE/@number"/>
            </xsl:element>
          </xsl:for-each>
        </xsl:element>
      </xsl:element>
    </xsl:for-each>
  </xsl:template>

```

◇

Macro referenced in chunk 4.1

4.1.15 lit2html: handle macro parameters

```
<lit2html: handle macro parameters 4.29> =  
  <xsl:template match="formal">  
    <xsl:element name="B">  
      <xsl:attribute name="style">color:brown</xsl:attribute>  
      <xsl:value-of select="@name"/>  
    </xsl:element>  
  </xsl:template>
```

◇

Macro referenced in chunk 4.1

4.1.16 lit2html: handle document history

```
<lit2html: handle document history 4.30> =  
  <xsl:template match="DocumentHistory">  
    <HR SIZE="4" NOSHADE="yes"/><TABLE BORDER="0">  
      <H1>Document History</H1>  
      <xsl:apply-templates select="Modified"/>  
    </TABLE>  
    <xsl:apply-templates select="*[name()!='Modified']"/>  
  </xsl:template>  
  <xsl:template match="DocumentHistory/Modified">  
    <TR>  
      <xsl:apply-templates/>  
    </TR>  
  </xsl:template>  
  <xsl:template match="Modified/date">  
    <TD VALIGN="top">  
      <xsl:apply-templates/>  
    </TD>  
  </xsl:template>  
  <xsl:template match="Modified/version">  
    <TD VALIGN="top">  
      <xsl:apply-templates/>  
    </TD>  
  </xsl:template>  
  <xsl:template match="Modified/author">  
    <TD VALIGN="top">  
      <xsl:apply-templates/>  
    </TD>  
  </xsl:template>  
  <xsl:template match="Modified/comment">  
    <TD VALIGN="top">  
      <xsl:apply-templates/>  
    </TD>  
  </xsl:template>
```

◇

Macro referenced in chunk 4.1

4.1.17 lit2html: special character templates

Define here all the special characters that cannot be represented exactly (either because there is no corresponding glyph, or because they require escaping in TeX)

<lit2html: special character templates 4.31> =

```
<xsl:template match="le">
  <xsl:text>&lt;=</xsl:text>
</xsl:template>
<xsl:template match="dollar">
  <xsl:text>$</xsl:text>
</xsl:template>
<xsl:template match="TeX">
  <xsl:text>TeX</xsl:text>
</xsl:template>
```

◇

Macro referenced in chunk 4.1

4.1.18 lit2html: default template

<lit2html: default template 4.32> =

```
<xsl:template match="text()" mode="strip-leadingnl">
  <xsl:choose>
    <xsl:when test="position()=1 and substring(.,1,1)='&#xA;'">
      <xsl:value-of select="substring(.,2)"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="."/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:template match="*" mode="strip-leadingnl">
  <xsl:apply-templates select="."/>
</xsl:template>
<xsl:template match="*">
  <xsl:if test="$Verbose">
    <xsl:message terminate="no">
      <xsl:text>No weft-html template for </xsl:text>
      <xsl:value-of select="name()"/>
    </xsl:message>
  </xsl:if>
  <xsl:element name="{name()}">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

◇

Macro referenced in chunk 4.1

Three templates, which a) strip leading newlines from text nodes, b) handle non-text nodes in stripping leading new-lines, and c) a catch-all default template for all other cases, that prints a warning message as appropriate. The element and its attributes are passed through unchanged.

4.2 lit2tex.xsl

"lit2tex.xsl" 4.33 =

```
<?xml version="1.0"?>
<!DOCTYPE xsl:transform SYSTEM "file:///home/ajh/lib/dtd/xslt.dtd">
<xsl:transform
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xinclude="http://www.w3.org/2001/XInclude"
  >
  <xsl:output method="text" encoding="ASCII"/>
  <!-- pull in some common totex translations -->
  <xsl:include href="LIBXSL 1.1"/numbered2tex.xsl"/>
  <xsl:include href="LIBXSL 1.1"/basic2tex.xsl"/>
  <xsl:include href="LIBXSL 1.1"/biblio2tex.xsl"/>
  <xsl:include href="LIBXSL 1.1"/tables2tex.xsl"/>
  <xsl:param name="Verbose">1</xsl:param>
  <lit2tex: handle the root node 4.34>
  <lit2tex: handle various documentation elements 4.35>
  <lit2tex: handle a literate program translation 4.36>
  <lit2tex: handle paragraphs 4.37>
  <lit2tex: handle reference calls 4.38>
  <lit2tex: handle code fragment definitions 4.39>
  <lit2tex: handle code use calls 4.42>
  <lit2tex: handle macro parameters 4.43>
  <lit2tex: handle variable definitions and uses 4.44>
  <lit2tex: handle in-line comments in code 4.45>
  <lit2tex: handle maintenance table documentation 4.46>
  <lit2tex: build table of contents 4.47>
  <lit2tex: build file list 4.51>
  <lit2tex: build identifier list 4.52>
  <lit2tex: handle chunk definition list 4.53>
  <lit2tex: handle document history 4.56>
  <lit2tex: text node handling 4.58>
  <lit2tex: special character templates 4.63>
  <lit2tex: default template 4.64>
</xsl:transform>
```

◇

This file was originally constructed on 20030529:215331

The purpose of this file is to convert the XML output of the literate programming XML tangle and weave translator *<litprog.xsl 3.1>* to a tex document, suitable for processing by plain T_EX.

It is assumed that the original litprog document doc.xlp has been processed by *<litprog.xsl 3.1>* to doc.xml. This script converts it to doc.tex

4.2.1 lit2tex: handle the root node

```
<lit2tex: handle the root node 4.34> =
  <!-- the root node: do all the TeX setup and sign off -->
  <weft: handle the root element 5.1> weft root pre text='
    <xsl:text disable-output-escaping="no">
      \input mytexMacros
      \input litprogverb
      \input crossref
      \def\nl{\hfil\break}
      \def\url{\tt}
      \parskip 4pt plus 10pt minus 2pt
      \def\parameters{\enumerate\itemskip=2pt
      \def\itemformat##1{\hss\$
      \enumerateflag{\enumeratelevel}{##1}\enspace}}%
      \let\endparameters=\endenumerate
      \advance\vsizer by 10pt
      \advance\hsizer by -20pt
      \tolerance=1600
    </xsl:text>
  ', weft root post text='
    <xsl:text disable-output-escaping="no">\bye</xsl:text>
  ',
  \diamond
```

Macro referenced in chunk 4.33

4.2.2 lit2tex: handle various documentation elements

```
<lit2tex: handle various documentation elements 4.35> =
  <!-- to be rewritten -->
  <xsl:template match="abstract">
    <H2 ALIGN="center">Abstract</H2>
    <DIV STYLE="margin-left:20pt">
      <xsl:apply-templates/>
    </DIV>
  </xsl:template>
  <xsl:template match="author">
    <xsl:text>\begin{centre}\large\bf </xsl:text>
    <xsl:apply-templates/>
    <xsl:text>\strut\end{centre}</xsl:text>
  </xsl:template>
  <xsl:template match="date">
    <xsl:text>\begin{centre}\large </xsl:text>
    <xsl:apply-templates/>
    <xsl:text>\strut\end{centre}</xsl:text>
  </xsl:template>
  <xsl:template match='title|subtitle'>
    <xsl:text>\begin{centre}\Large\bf </xsl:text>
    <xsl:apply-templates/>
    <xsl:text>\strut\end{centre}</xsl:text>
  </xsl:template>
  <xsl:template match="version">
    <xsl:text>\begin{centre}\large\rm Version </xsl:text>
    <xsl:apply-templates/>
    <xsl:text>\strut\end{centre}</xsl:text>
```

```

</xsl:template>
<xsl:template match="VALUE">
  <xsl:variable name="myname" select="."/>
  <xsl:value-of select="//CODE[@name=$myname]"/>
</xsl:template>
<!-- to be rewritten -->
<xsl:template match="br"><BR/></xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.3 lit2tex: handle a literate program translation

```

<lit2tex: handle a literate program translation 4.36> =
  <xsl:template match="LitprogXML">
    <xsl:apply-templates/>
  </xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.4 lit2tex: handle paragraphs

```

<lit2tex: handle paragraphs 4.37> =
  <xsl:template match="P">
    <xsl:apply-templates/>
    <xsl:text>\par </xsl:text>
  </xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.5 lit2tex: handle reference calls

```

<lit2tex: handle reference calls 4.38> =
  <xsl:template match="REF">
    <xsl:variable name="myname" select="."/>
    <xsl:text disable-outputescaping="yes">&lt;</xsl:text>
    <xsl:apply-templates/>
    <xsl:text disable-outputescaping="yes">=</xsl:text>
    <xsl:for-each select="//CODE[@name=$myname]">
      <xsl:if test="position()>1"><xsl:text>, </xsl:text></xsl:if>
      <xsl:value-of select="@number"/>
    </xsl:for-each>
    <xsl:text disable-outputescaping="yes">&gt;</xsl:text>
  </xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.6 lit2tex: handle code fragment definitions

This template formats a code fragment. There is special work to be done on the name, because of escaping T_EX characters. Underlines are replaced with their escaped versions and the result saved in the variable `escname`. Note that the distinction between the escaped string `\$escname` and the unescaped string `@name` must be preserved, depending upon whether we are using the string for T_EX purposes, or for XML purposes.

```
<lit2tex: handle code fragment definitions 4.39> =
  <xsl:template match="CODE">
    <xsl:variable name="escname">
      <xsl:call-template name="replacesubstring">
        <xsl:with-param name="string" select="@name"/>
        <xsl:with-param name="pattern">
          <xsl:value-of select="'_'"/>
        </xsl:with-param>
        <xsl:with-param name="replacement">
          <xsl:value-of select="'\_'" />
        </xsl:with-param>
      </xsl:call-template>
    </xsl:variable>
    <xsl:if test="@type='file'">
      <xsl:text>\vskip\parskip</xsl:text>
      <xsl:text>\begingroup\vskip\parskip</xsl:text>
      <xsl:text>\leavevmode\advance\leftskip by \codemargin</xsl:text>
      <xsl:text>\kern-\codemargin{\it' '</xsl:text>
      <xsl:value-of select="$escname"/>
      <xsl:text>"} </xsl:text>
      <xsl:value-of select="@number"/>
      <xsl:text> =</xsl:text>
    </xsl:if>
    <xsl:if test="@type='define'">
      <xsl:text>\leavevmode\begingroup\advance\leftskip by \codemargin</xsl:text>
      <xsl:text>\kern-\codemargin&&lt;${\it </xsl:text>
      <xsl:value-of select="$escname"/>
      <xsl:text> </xsl:text>
      <xsl:value-of select="@number"/>
      <xsl:text>\}/&&gt;${ =</xsl:text>
    </xsl:if>
    <xsl:text>\begin{codeverbatim}</xsl:text>
    <xsl:apply-templates/>
    <xsl:text>\end</xsl:text>
    <!-- vitally important that these two text fragments be separate! -->
    <xsl:text>{codeverbatim}\kern6pt$\diamond$\par&#xA;</xsl:text>
    <xsl:text>\par&#xA;</xsl:text>
    <xsl:variable name="thisname" select="$escname"/>
    <xsl:variable name="reflist">
      <xsl:for-each select="//CODE">
        <xsl:variable name="refnumber" select="@number"/>
        <xsl:if test="descendant::USE[@NAME=$thisname]">
          <xsl:text> </xsl:text>
          <xsl:value-of select="$refnumber"/>
        </xsl:if>
      </xsl:for-each>
    </xsl:variable>
    <lit2tex: display code chunk definition list 4.40>
    <lit2tex: display code chunk reference list 4.41>
```

```

    <xsl:text>\par\endgroup&#xA;</xsl:text>
</xsl:template>
<xsl:template match="actual">
  <xsl:text>\end</xsl:text>
  <!-- vitally important that these two text fragments be separate! -->
  <xsl:text>{codeverbatim}\kern4pt</xsl:text>
  <xsl:text>{\it </xsl:text>
  <xsl:value-of select="@name"/>
  <xsl:text>}</xsl:text>
  <xsl:text>\begin</xsl:text>
  <!-- vitally important that these two text fragments be separate! -->
  <xsl:text>{codeverbatim}</xsl:text>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

<lit2tex: display code chunk definition list 4.40> =

```

<xsl:if test="count(//CODE[@name=$thisname])>1">
  <xsl:text>{\it Macro defined in chunk </xsl:text>
  <xsl:for-each select="//CODE[@name=$thisname]">
    <xsl:variable name="refnumber" select="@number"/>
    <xsl:if test="position()>1">
      <xsl:text>\discretionary{,}{,}{,}\hskip0pt plus 0.2pt</xsl:text>
    </xsl:if>
    <xsl:value-of select="$refnumber"/>
  </xsl:for-each>
  <xsl:text>}&#xA;</xsl:text>
  <xsl:if test="$reflist!=''">
    <xsl:text>\hfil\break&#xA;</xsl:text>
  </xsl:if>
</xsl:if>

```

◇

Macro referenced in chunk 4.39

<lit2tex: display code chunk reference list 4.41> =

```

<xsl:if test="$reflist!=''">
  <xsl:text>{\it Macro referenced in chunk </xsl:text>
  <xsl:for-each select="//CODE/descendant::USE[@NAME=$thisname]">
    <xsl:variable name="refnumber" select="ancestor::CODE/@number"/>
    <xsl:if test="position()>1">
      <xsl:text>,\hskip 0pt plus 0.2pt </xsl:text>
    </xsl:if>
    <xsl:value-of select="$refnumber"/>
  </xsl:for-each>
  <xsl:text>}&#xA;</xsl:text>
</xsl:if>

```

◇

Macro referenced in chunk 4.39

If the variable `reflist` is not empty, display the list of chunk numbers for each of the chunk names in the list. Only unique numbers need be displayed, hence the nested if statement within the `//CODE` for-each, rather than another for-each which would itemize repeated uses.

4.2.7 lit2tex: handle code use calls

```
<lit2tex: handle code use calls 4.42> =
<xsl:template match="CODE/USE">
  <xsl:text>\end</xsl:text>
  <!-- vitally important that these two text fragments be separate! -->
  <xsl:text>{codeverbatim}$&lt;it </xsl:text>
  <xsl:variable name="myname" select="@NAME"/>
  <xsl:variable name="include" select="@INCLUDE"/>
  <xsl:value-of select="@NAME"/>
  <xsl:text> </xsl:text>
  <xsl:for-each select="//CODE[@name=$myname]">
    <xsl:if test="position()>1"><xsl:text>,</xsl:text></xsl:if>
    <xsl:value-of select="@number"/>
    <xsl:if test="$include='no'">
      <xsl:text> {\bf Chunk omitted!}</xsl:text>
    </xsl:if>
  </xsl:for-each>
  <xsl:text>}&gt;${</xsl:text>
  <xsl:if test="actual">
    <xsl:for-each select="actual">
      <xsl:if test="position()=1">
        <xsl:text> </xsl:text>
      </xsl:if>
      <xsl:value-of select="./@name"/>
      <xsl:text>=</xsl:text>
      <xsl:text>'\begin{codeverbatim}</xsl:text>
      <xsl:apply-templates/>
      <xsl:text>\end</xsl:text>
      <xsl:text>{codeverbatim}\kern4pt'</xsl:text>
      <xsl:if test="position()&lt;last()">
        <xsl:text>,</xsl:text>
      </xsl:if>
    </xsl:for-each>
  </xsl:if>
  <xsl:text>\begin{codeverbatim}</xsl:text>
</xsl:template>
<xsl:template match="USE">
  <xsl:variable name="myname" select="@NAME"/>
  <xsl:choose>
    <xsl:when test="@EXPAND">
      <xsl:value-of select="//CODE[@name=$myname]"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:element name="I">
        <xsl:element name="A">
          <xsl:attribute name="HREF">#<xsl:value-of select="@NAME"/>
          </xsl:attribute>
          <xsl:text disable-outputsescaping="yes">${&lt;it </xsl:text>
          <xsl:value-of select="@NAME"/>
          <xsl:text disable-outputsescaping="yes"> </xsl:text>
          <xsl:for-each select="//CODE[@name=$myname]">
            <xsl:if test="position()>1"><xsl:text>,</xsl:text></xsl:if>
            <xsl:value-of select="@number"/>
          </xsl:for-each>
          <xsl:text disable-outputsescaping="yes">\} &gt; ${</xsl:text>
        </xsl:element>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

```

</xsl:element>
<xsl:if test="@INCLUDE='no'">
  <xsl:element name="SPAN">
    <xsl:attribute name="STYLE">color:red</xsl:attribute>
    <xsl:text> **** Chunk omitted!</xsl:text>
  </xsl:element>
</xsl:if>
</xsl:otherwise>
</xsl:choose>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.8 lit2tex: handle macro parameters

```

<lit2tex: handle macro parameters 4.43> =
<xsl:template match="formal">
  <xsl:text> \end</xsl:text>
  <!-- vitally important that these two text fragments be separate! -->
  <xsl:text>{codeverbatim}{\bf </xsl:text>
  <xsl:value-of select="@name"/>
  <xsl:text>}</xsl:text>
  <xsl:text>\begin{codeverbatim}</xsl:text>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

The space before the end verbatim seems to be required, as the end verbatim seems to consume the blank immediately before it. Don't know why.

4.2.9 lit2tex: handle variable definitions and uses

```

<lit2tex: handle variable definitions and uses 4.44> =
<xsl:template match="VAR">
  <xsl:choose>
    <xsl:when test="@VAR='def'">
      <xsl:element name="A">
        <xsl:attribute name="NAME">
          <xsl:choose>
            <xsl:when test="@NAME">
              <xsl:value-of select="@NAME"/>
            </xsl:when>
            <xsl:otherwise>
              <xsl:value-of select="."/>
            </xsl:otherwise>
          </xsl:choose>
        </xsl:attribute>
        <xsl:attribute name="STYLE">color:red</xsl:attribute>
        <xsl:apply-templates/>
      </xsl:element>
    </xsl:when>
    <xsl:when test="@VAR='use'">
      <xsl:element name="A">
        <xsl:attribute name="HREF">
          <xsl:text>#</xsl:text>
          <xsl:value-of select="."/>
        </xsl:attribute>

```

```

        <xsl:apply-templates/>
      </xsl:element>
    </xsl:when>
  </xsl:choose>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.10 lit2tex: handle in-line comments in code

```

<lit2tex: handle in-line comments in code 4.45> =
  <xsl:template match="COMMENT">
    <xsl:text>\end</xsl:text>
    <xsl:text>{codeverbatim}\kern4pt$^{</xsl:text>
    <xsl:value-of select="@HREF"/>
    <xsl:text>}$</xsl:text>
    <xsl:text>\begin</xsl:text>
    <xsl:text>{codeverbatim}</xsl:text>
  </xsl:template>
  <xsl:template match="NOTES">
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>Notes:\par\begin</xsl:text>
    <xsl:text>{enumerate}&#xA;</xsl:text>
    <xsl:apply-templates>
  </xsl:apply-templates>
    <xsl:text>\end</xsl:text>
    <xsl:text>{enumerate}</xsl:text>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>
  <xsl:template match="NOTE">
    <xsl:text>\item </xsl:text>
    <xsl:apply-templates/>
    <xsl:text>&#xA;</xsl:text>
  </xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.11 lit2tex: handle maintenance table documentation

```

<lit2tex: handle maintenance table documentation 4.46> =
  <xsl:template match="maintaintable">
    <xsl:text>\halign{\tabskip=1em plus 1em minus 0.5em#</xsl:text>
    <xsl:text>\hfil&amp;#\hfil&amp;#\hfil&amp;#</xsl:text>
    <xsl:text>\vtop{\hsize=100mm#\strut\hfil}\hfil\cr </xsl:text>
    <xsl:apply-templates select=".*"/>
    <xsl:text>\crcr}</xsl:text>
  </xsl:template>
  <xsl:template match="maintaintable/date">
    <xsl:text disable-output-escaping="yes"></xsl:text>
    <xsl:apply-templates/>
    <xsl:text>&amp;</xsl:text>
  </xsl:template>
  <xsl:template match="maintaintable/author">
    <xsl:apply-templates/>
    <xsl:text>&amp;</xsl:text>

```

```

</xsl:template>
<xsl:template match="maintaintable/version">
  <xsl:apply-templates/>
  <xsl:text>&amp;</xsl:text>
</xsl:template>
<xsl:template match="maintaintable/comment">
  <xsl:apply-templates/>
  <xsl:text>\cr&#xA;</xsl:text>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.12 lit2tex: build table of contents

```

<lit2tex: build table of contents 4.47> =
<xsl:template match="TableOfContents">
  <xsl:text>\heading{Table of Contents}&#x0a;</xsl:text>
  <xsl:text>\halign to \hsize{\tabskip=1em#\hfil&amp;#\hfil&amp;</xsl:text>
  <xsl:text>#\hfil&amp;#\hfil\tabskip=1em plus 1fil&amp;\hfil#</xsl:text>
  <xsl:text>\tabskipOpt\cr&#x0a;</xsl:text>
  <xsl:for-each select="//section">
    <lit2tex: build toc: section 4.48>
    <xsl:for-each select="subsection">
      <lit2tex: build toc: subsection 4.49>
      <xsl:for-each select="subsubsection">
        <lit2tex: build toc: subsubsections 4.50>
        </xsl:for-each>
      </xsl:for-each>
    </xsl:for-each>
  </xsl:for-each>
  <xsl:text>}&#x0a;</xsl:text>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

The table of contents is constructed using an `halign` macro, set to the page width, and indenting each section heading according to the depth of the heading. The last template is right justified, and carries the page number for the start of the section (actually, the heading of the section). We define the table template, then enter loops to process headings at each level with a (nested) loop. Close the table of contents with a closing brace and a newline.

```

<lit2tex: build toc: section 4.48> =
  <xsl:number select="position(.)"/>
  <xsl:text>&amp;\multispan{3}</xsl:text>
  <xsl:value-of select="normalize-space(title)"/>
  <xsl:text>\hfil&amp;\page{</xsl:text>
  <xsl:value-of select="@xref"/>
  <xsl:text>}\cr&#x0a;</xsl:text>

```

◇

Macro referenced in chunk 4.47

Construct a (one-level) section number, no indent, format with the title text and a page reference using the sanitised xref title text

```

<lit2tex: build toc: subsection 4.49> =
  <xsl:text>&amp;</xsl:text>
  <xsl:number level="single" count="section"/>

```

```

<xsl:text>.</xsl:text>
<xsl:number from="section" level="single" count="subsection"/>
<xsl:text>&amp;\multispan{2}</xsl:text>
<xsl:value-of select="normalize-space(title)"/>
<xsl:text>\hfil&amp;\page{</xsl:text>
<xsl:value-of select="@xref"/>
<xsl:text>}\cr&#x0a;</xsl:text>

```

◇

Macro referenced in chunk 4.47

Construct a two-level section number, indent it once, format with the title text and a page reference using the sanitised xref title text

```

<lit2tex: build toc: subsubsections 4.50> =
  <xsl:text>&amp;&amp;</xsl:text>
  <xsl:number level="single" count="section"/>
  <xsl:text>.</xsl:text>
  <xsl:number from="section" level="single" count="subsection"/>
  <xsl:text>.</xsl:text>
  <xsl:number from="subsection" level="single" count="subsubsection"/>
  <xsl:text>&amp;</xsl:text>
  <xsl:value-of select="normalize-space(title)"/>
  <xsl:text>&amp;\page{</xsl:text>
  <xsl:value-of select="@xref"/>
  <xsl:text>}\cr&#x0a;</xsl:text>

```

◇

Macro referenced in chunk 4.47

Construct a three-level section number, indent it twice, format with the title text and a page reference using the sanitised xref title text

4.2.13 lit2tex: build file list

The file list macro builds a fairly simple T_EX table of file identifiers, together with a list of the chunk numbers in which they are defined. Once the table parameters are setup, scan through all CODE elements looking for file definitions with unique file names. Escape underline characters from these names. Then generate a list of all chunk numbers defining this file. Note the need to keep the unescaped file name `\$thisname` distinct from the escaped file name `\$escname`.

```

<lit2tex: build file list 4.51> =
  <xsl:template match="filelist">
    <xsl:text>\halign{\tabskip=1em#\hfil&amp;#\hfil\cr&#xA;</xsl:text>
    <xsl:text>\bf File Name</xsl:text>
    <xsl:text>&amp;</xsl:text>
    <xsl:text>\bf Defined in</xsl:text>
    <xsl:text>\cr </xsl:text>
    <xsl:for-each select="//CODE">
      <xsl:sort select="@name"/>
      <xsl:variable name="thisname" select="@name"/>
      <xsl:variable name="prevname" select="preceding::CODE[1]/@name"/>
      <xsl:if test="(not($prevname) or $thisname!=$prevname) and @type='file'">
        <xsl:variable name="escname">
          <xsl:call-template name="replacesubstring">
            <xsl:with-param name="string" select="$thisname"/>
            <xsl:with-param name="pattern">
              <xsl:value-of select="'_'"/>
            </xsl:with-param>

```

```

        <xsl:with-param name="replacement">
            <xsl:value-of select="'\_,'"/>
        </xsl:with-param>
    </xsl:call-template>
</xsl:variable>
<xsl:value-of select="$escname"/>
<xsl:text> & </xsl:text>
<xsl:for-each select="//CODE[@name=$thisname]">
    <xsl:if test="position()>1">
        <xsl:text>, </xsl:text>
    </xsl:if>
    <xsl:value-of select="@number"/>
</xsl:for-each>
<xsl:text>\cr&#xA;</xsl:text>
</xsl:if>
</xsl:for-each>
<xsl:text>}</xsl:text>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.14 lit2tex: build identifier list

```

<lit2tex: build identifier list 4.52> =
<xsl:template match="IdentifierDefinitions">
    <lit2tex: build macrolist width variables 4.54>
    <lit2tex: define macrolist table template 4.55>
    <xsl:for-each select="//VAR[@VAR='def']">
        <xsl:sort select="."/>
        <xsl:variable name="name" select="."/>
        <xsl:element name="TR">
            <xsl:element name="TD">
                <xsl:value-of select="."/>
            </xsl:element>
            <xsl:text> & </xsl:text>
            <xsl:element name="TD">
                <xsl:attribute name="ALIGN">center</xsl:attribute>
                <xsl:element name="A">
                    <xsl:attribute name="HREF">
                        <xsl:text>#xlp:</xsl:text>
                        <xsl:value-of select="ancestor::CODE/@number"/>
                    </xsl:attribute>
                    <xsl:value-of select="ancestor::CODE/@number"/>
                </xsl:element>
            </xsl:element>
            <xsl:text> & </xsl:text>
            <xsl:element name="TD">
                <xsl:attribute name="ALIGN">right</xsl:attribute>
                <xsl:for-each select="//CODE/VAR[@VAR='use' and string()=$name]">
                    <xsl:if test="position()>1">
                        <xsl:text>, </xsl:text>
                    </xsl:if>
                    <xsl:element name="A">
                        <xsl:attribute name="HREF">
                            <xsl:text>#xlp:</xsl:text>
                            <xsl:value-of select="ancestor::CODE/@number"/>
                        </xsl:attribute>
                    </xsl:element>
                </xsl:for-each>
            </xsl:element>
        </xsl:element>
    </xsl:for-each>

```

```

        </xsl:attribute>
        <xsl:value-of select="ancestor::CODE/@number"/>
    </xsl:element>
</xsl:for-each>
</xsl:element>
<xsl:text>\cr </xsl:text>
</xsl:element>
</xsl:for-each>
<xsl:text>}</xsl:text>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

4.2.15 lit2tex: handle chunk definition list

The challenge in tabulating a list of chunk definitions is in handling the variable widths of the fields, given that some of them may require multiple lines to fit within the page boundaries. This really requires a two pass algorithm, of greater complexity than we can afford here. Consequently, we settle for some manually determined widths, which can be overridden if necessary. The default widths are 50% for the chunk name, and 25% for each of the definition and use lists.

Scan through all CODE elements to find the definitions, and build these into a table of identifiers, showing both the point of definitions, and all uses of the identifier.

```

<lit2tex: handle chunk definition list 4.53> =
  <xsl:template match="macrolist">
    <lit2tex: build macrolist width variables 4.54>
    <lit2tex: define macrolist table template 4.55>
    <xsl:for-each select="//CODE">
      <xsl:sort select="./@name"/>
      <xsl:variable name="thispos" select="position()"/>
      <xsl:variable name="thisname" select="./@name"/>
      <xsl:variable name="prevname">
        <xsl:choose>
          <xsl:when test="preceding::CODE[1]/@name">
            <xsl:value-of select="preceding::CODE[1]/@name"/>
          </xsl:when>
          <xsl:otherwise> </xsl:otherwise>
        </xsl:choose>
      </xsl:variable>
      <xsl:if test="$thisname!=$prevname and @type='define'">
        <xsl:value-of select="./@name"/>
        <xsl:text> &#amp; </xsl:text>
        <xsl:for-each select="//CODE[@name=$thisname]">
          <xsl:if test="position()&gt;1">
            <xsl:text>, </xsl:text>
          </xsl:if>
          <xsl:value-of select="./@number"/>
        </xsl:for-each>
        <xsl:text> &#amp; </xsl:text>
        <xsl:for-each select="//CODE/USE[@NAME=$thisname]">
          <xsl:variable name="thisnum" select="ancestor::CODE/@number"/>
          <!-- calc of prevnum is wrong ! -->
          <xsl:variable name="prevnum" select="preceding::./@number"/>
          <xsl:if test="position()&gt;1">
            <xsl:text>, </xsl:text>
          </xsl:if>

```

```

        <xsl:value-of select="$thisnum"/>
      </xsl:for-each>
      <xsl:text>\cr&#xA;</xsl:text>
    </xsl:if>
  </xsl:for-each>
  <xsl:text>}</xsl:text>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

We define three variables to store the widths of the three chunk name list columns: chunk name, chunk definitions, and chunk uses. These values are either defined by the user with attributes to the `macrolist` element, or are set by default to the values 0.50, 0.25, 0.25 respectively.

<lit2tex: build macrolist width variables 4.54> =

```

<xsl:variable name="namewidth">
  <xsl:choose>
    <xsl:when test="@namewidth">
      <xsl:value-of select="@namewidth"/>
    </xsl:when>
    <xsl:otherwise><xsl:text>0.50</xsl:text></xsl:otherwise>
  </xsl:choose>
</xsl:variable>
<xsl:variable name="definewidth">
  <xsl:choose>
    <xsl:when test="@definewidth">
      <xsl:value-of select="@definewidth"/>
    </xsl:when>
    <xsl:otherwise><xsl:text>0.25</xsl:text></xsl:otherwise>
  </xsl:choose>
</xsl:variable>
<xsl:variable name="usewidth">
  <xsl:choose>
    <xsl:when test="@usewidth">
      <xsl:value-of select="@usewidth"/>
    </xsl:when>
    <xsl:otherwise><xsl:text>0.25</xsl:text></xsl:otherwise>
  </xsl:choose>
</xsl:variable>

```

◇

Macro referenced in chunk 4.52,4.53

This next bit is really messy, basically because of the horrendous \TeX code being produced. This is what gets produced by this bit of code for the `macrolist` reference in this program itself (see later).

```

\halign{\tabskip=0em\advance\hsize by -1em
\vtop{\hsize=0.60\hsize#\hfil}\hskip0.5em&
\vtop{\hsize=0.20\hsize#\hfil}\hskip0.5em&
\vtop{\hsize=0.20\hsize#\hfil}\cr
\bf Identifier&\bf Defined in&\bf Used in\cr

```

Each of the table columns gets a width that can be set in the original XML call on `macrolist` via the optional attributes `namewidth`, `definewidth` and `usewidth`. Each of these parameters should be a fraction of 1.0, defining the fractional width of the page taken up by that column, and their sum should be ≤ 1.0

<lit2tex: define macrolist table template 4.55> =

```

<xsl:text>\overfullrule=0pt&#xA;</xsl:text>
<xsl:text>\halign{\tabskip=0em\advance\hsize by -1em&#xA;</xsl:text>
<xsl:text>\vtop{\hsize=</xsl:text>

```

```

<xsl:value-of select="$namewidth"/>
<xsl:text>\hsize#\hfil}\hskip0.5em</xsl:text>
<xsl:text disable-output-escaping="yes">&#xA;</xsl:text>
<xsl:text>\vtop{\hsize=</xsl:text>
<xsl:value-of select="$definewidth"/>
<xsl:text>\hsize#\hfil}\hskip0.5em</xsl:text>
<xsl:text disable-output-escaping="yes">&#xA;</xsl:text>
<xsl:text>\vtop{\hsize=</xsl:text>
<xsl:value-of select="$usewidth"/>
<xsl:text>\hsize#\hfil}\cr&#xA;</xsl:text>
<xsl:text>\bf Identifier</xsl:text>
<xsl:text>&#xA;</xsl:text>
<xsl:text>\bf Defined in</xsl:text>
<xsl:text>&#xA;</xsl:text>
<xsl:text>\bf Used in</xsl:text>
<xsl:text>\cr&#xA;</xsl:text>

```

◇

Macro referenced in chunk 4.52,4.53

4.2.16 lit2tex: handle document history

Generate a table to capture the document history.

```

<lit2tex: handle document history 4.56> =
  <xsl:template match="DocumentHistory">
    <lit2tex: define Document History variables 4.57>
    <xsl:text>\heading{Document History}</xsl:text>
    <xsl:text>\halign{\tabskip=0pt\advance\hsize by -1.5em&#xA;</xsl:text>
    <xsl:text>\vtop{\hsize=</xsl:text>
    <xsl:value-of select="$datewidth"/>
    <xsl:text>\hsize#\hfil}\hskip0.5em</xsl:text>
    <xsl:text disable-output-escaping="yes">&#xA;</xsl:text>
    <xsl:text>\vtop{\hsize=</xsl:text>
    <xsl:value-of select="$authorwidth"/>
    <xsl:text>\hsize#\hfil}\hskip0.5em</xsl:text>
    <xsl:text disable-output-escaping="yes">&#xA;</xsl:text>
    <xsl:text>\vtop{\hsize=</xsl:text>
    <xsl:value-of select="$versionwidth"/>
    <xsl:text>\hsize#\hfil}\hskip0.5em</xsl:text>
    <xsl:text disable-output-escaping="yes">&#xA;</xsl:text>
    <xsl:text>\vtop{\hsize=</xsl:text>
    <xsl:value-of select="$commentwidth"/>
    <xsl:text>\hsize#\hfil}\cr&#xA;</xsl:text>
    <xsl:text>\bf Date</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>\bf Author</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>\bf Version</xsl:text>
    <xsl:text>&#xA;</xsl:text>
    <xsl:text>\bf Comment</xsl:text>
    <xsl:text>\cr&#xA;</xsl:text>
    <xsl:apply-templates select="Modified"/>
    <xsl:text></xsl:text>
    <xsl:apply-templates select="*[name()!='Modified']"/>
  </xsl:template>
  <xsl:template match="DocumentHistory/Modified">

```

```

    <xsl:apply-templates/>
    <xsl:text disable-output-escaping="no">\cr</xsl:text>
</xsl:template>
<xsl:template match="Modified/date">
  <xsl:apply-templates/>
  <xsl:text>&amp;</xsl:text>
</xsl:template>
<xsl:template match="Modified/version">
  <xsl:apply-templates/>
  <xsl:text>&amp;</xsl:text>
</xsl:template>
<xsl:template match="Modified/author">
  <xsl:apply-templates/>
  <xsl:text>&amp;</xsl:text>
</xsl:template>
<xsl:template match="Modified/comment">
  <xsl:apply-templates/>
</xsl:template>

```

◇

Macro referenced in chunk 4.33

Define the various column width parameters for the Document History table. Each of these is set to a default value, unless there is an explicit attribute setting, when they are set instead to the value passed in the attribute.

<lit2tex: define Document History variables 4.57> =

```

<xsl:variable name="datewidth">
  <xsl:choose>
    <xsl:when test="@datewidth">
      <xsl:value-of select="@datewidth"/>
    </xsl:when>
    <xsl:otherwise><xsl:text>0.17</xsl:text></xsl:otherwise>
  </xsl:choose>
</xsl:variable>
<xsl:variable name="authorwidth">
  <xsl:choose>
    <xsl:when test="@authorwidth">
      <xsl:value-of select="@authorwidth"/>
    </xsl:when>
    <xsl:otherwise><xsl:text>0.10</xsl:text></xsl:otherwise>
  </xsl:choose>
</xsl:variable>
<xsl:variable name="versionwidth">
  <xsl:choose>
    <xsl:when test="@versionwidth">
      <xsl:value-of select="@versionwidth"/>
    </xsl:when>
    <xsl:otherwise><xsl:text>0.09</xsl:text></xsl:otherwise>
  </xsl:choose>
</xsl:variable>
<xsl:variable name="commentwidth">
  <xsl:choose>
    <xsl:when test="@commentwidth">
      <xsl:value-of select="@commentwidth"/>
    </xsl:when>
    <xsl:otherwise><xsl:text>0.64</xsl:text></xsl:otherwise>
  </xsl:choose>

```

```
</xsl:variable>
```

◇

Macro referenced in chunk 4.56

4.2.17 lit2tex: text node handling

```
<lit2tex: text node handling 4.58> =
```

```
<lit2tex: callable template replacesubstring 4.59>
```

```
<lit2tex: replace dollar signs with escaped dollar signs 4.60>
```

```
<lit2tex: filter at signs in text nodes 4.61 Chunk omitted!>
```

```
<lit2tex: filter underlines in non-verbatim text nodes 4.62>
```

◇

Macro referenced in chunk 4.33

These templates handle text nodes in the \TeX weft phase. The callable template `replacesubstring` is used in a couple of the templates.

4.2.17.1 lit2tex: callable template `replacesubstring`

```
<lit2tex: callable template replacesubstring 4.59> =
```

```
<xsl:template name="replacesubstring">
```

```
<xsl:param name="string"></xsl:param>
```

```
<xsl:param name="pattern"></xsl:param>
```

```
<xsl:param name="replacement"></xsl:param>
```

```
<xsl:choose>
```

```
<xsl:when test="contains($string,$pattern)">
```

```
<xsl:value-of select="substring-before($string,$pattern)"/>
```

```
<xsl:value-of select="$replacement"/>
```

```
<xsl:call-template name="replacesubstring">
```

```
<xsl:with-param name="string">
```

```
<xsl:value-of select="substring-after($string,$pattern)"/>
```

```
</xsl:with-param>
```

```
<xsl:with-param name="pattern">
```

```
<xsl:value-of select="$pattern"/>
```

```
</xsl:with-param>
```

```
<xsl:with-param name="replacement">
```

```
<xsl:value-of select="$replacement"/>
```

```
</xsl:with-param>
```

```
</xsl:call-template>
```

```
</xsl:when>
```

```
<xsl:otherwise>
```

```
<xsl:value-of select="$string"/>
```

```
</xsl:otherwise>
```

```
</xsl:choose>
```

```
</xsl:template>
```

◇

Macro referenced in chunk 4.58

This callable template takes a string, and returns a new string with substrings replaced consistently.

4.2.17.2 lit2tex: replace dollar signs with escaped dollar signs

```
<lit2tex: replace dollar signs with escaped dollar signs 4.60> =
```

```
<xsl:template match="c//text()">
```

```
<xsl:call-template name="replacesubstring">
```

```
<xsl:with-param name="string" select="."/>
```

```
<xsl:with-param name="pattern">
```

```

        <xsl:value-of select="'&#x24;'" />
    </xsl:with-param>
    <xsl:with-param name="replacement">
        <xsl:value-of select="'\&#x24;'" />
    </xsl:with-param>
</xsl:call-template>
</xsl:template>

```

◇

Macro referenced in chunk 4.58

A straightforward replace of any dollar sign by their escaped versions in \TeX . These otherwise would start mathematics mode.

4.2.17.3 lit2tex: filter at signs in text nodes

This template sanitises code text to double up any at signs. This is a hangover from the C version of the litprog, and will be removed at a future date. The offending code is the \TeX routine `codeverbatim` in file “almostverb”.

```

<lit2tex: filter at signs in text nodes 4.61> =
  <xsl:template match="CODE//text()">
    <xsl:call-template name="replacesubstring">
      <xsl:with-param name="string" select="." />
      <xsl:with-param name="pattern">
        <xsl:value-of select="'@" />
      </xsl:with-param>
      <xsl:with-param name="replacement">
        <xsl:value-of select="'@@'" />
      </xsl:with-param>
    </xsl:call-template>
  </xsl:template>

```

◇

Macro referenced in chunk 4.58

4.2.17.4 lit2tex: filter underlines in non-verbatim text nodes

We need to make sure that text in text nodes gets appropriate markup from the \TeX point of view. Particular cases are underscores and dollar signs. Munge through all text nodes to replace these with their sanitised text equivalents.

The exceptions are where the text occurs in various verbatim modes. The XPath expression in the match filters out these exceptions.

```

<lit2tex: filter underlines in non-verbatim text nodes 4.62> =
  <xsl:template match="//*[name()!='verb' and name()!='CODE'
                                and name()!='c']/text()">
    <xsl:call-template name="replacesubstring">
      <xsl:with-param name="string" select="." />
      <xsl:with-param name="pattern">
        <xsl:value-of select="'_'" />
      </xsl:with-param>
      <xsl:with-param name="replacement">
        <xsl:value-of select="'\_'" />
      </xsl:with-param>
    </xsl:call-template>
  </xsl:template>

```

◇

Macro referenced in chunk 4.58

The following template filters text nodes to replace underlines with their T_EX escaped equivalents. Note that the filtering is NOT applied when the containing node is a verb or CODE node, as these contexts have their own verbatim quoting mechanisms.

4.2.18 lit2tex: special character templates

Define here all the special characters that cannot be represented exactly (either because there is no corresponding glyph, or because they require escaping in T_EX

```
<lit2tex: special character templates 4.63> =
  <xsl:template match="le">
    <xsl:text>\ifmmode\le\else$\le$\fi</xsl:text>
  </xsl:template>
  <xsl:template match="dollar">
    <xsl:text>\$</xsl:text>
  </xsl:template>
  <xsl:template match="TeX">
    <xsl:text>\TeX</xsl:text>
  </xsl:template>
  <xsl:template match="htmlonly">
    <xsl:text>{\bf HTML only source omitted!}</xsl:text>
  </xsl:template>
```

◇

Macro referenced in chunk 4.33

4.2.19 lit2tex: default template

```
<lit2tex: default template 4.64> =
  <xsl:template match="*">
    <xsl:if test="$Verbose">
      <xsl:message terminate="no">
        <xsl:text>No weft-tex template for </xsl:text>
        <xsl:value-of select="name()"/>
      </xsl:message>
    </xsl:if>
    <xsl:element name="{name()}">
      <xsl:apply-templates mode="tangle"/>
    </xsl:element>
  </xsl:template>
```

◇

Macro referenced in chunk 4.33

A catch-all for any undefined templates. Issue a warning message and pass the element through unchanged.

5 weft: Common weft fragments

<weft: handle the root element 5.1> =

```
<xsl:template match="/">  
weft root pre text  
  <xsl:apply-templates/>  
weft root post text  
</xsl:template>
```

◇

Macro referenced in chunk 4.2,4.34

6 The litprog Document Type Definition

"litprog.dtd" 6.1 =

```
<!ENTITY % text "b|c|d|dq|em|i|o|p|u|v|verb|version|quote|uri|nl|
                case|description|itemize|enumerate|narrower|
                verbatim|bibliography|table|figure|footnote|
                formal|actual
                ">
1
<!ENTITY % misc "TableOfContents|filelist|macrolist|identifierlist|d">
2
<!ELEMENT litprog (title,author,version,date,
                  ( section | %misc; )*,
                  DocumentHistory?)>
<!ENTITY % stdenvironments PUBLIC "-//MONASH-CSSE//DTD stdenvironments 1.0//EN"
          "/home/ajh/lib/dtd/stdenvironments.dtd">
%stdenvironments;
<!ELEMENT narrower (#PCDATA|%text;)*>
<!ELEMENT figure (image,caption)>
<!ELEMENT image EMPTY>
<!ATTLIST image src CDATA #REQUIRED
               width CDATA #IMPLIED>
<!ELEMENT caption (#PCDATA|%text;)*>
<!ELEMENT footnote (#PCDATA|%text;)*>
<!ELEMENT descr (#PCDATA|version|it|em|anchor)*>
<!ELEMENT hrule EMPTY>
<!ELEMENT quote (#PCDATA|p)*>
<!ELEMENT verb (#PCDATA)>
<!ELEMENT verbatim (#PCDATA)>
<!ELEMENT case (choice)*>
<!ATTLIST case tag CDATA #IMPLIED>
<!ELEMENT choice (#PCDATA)>
<!ATTLIST choice tag CDATA #IMPLIED>
<!ELEMENT anchor (#PCDATA)>
<!ATTLIST anchor name CDATA #IMPLIED>
<litprog.dtd: Literate Programming elements 6.2>
<litprog.dtd: Heading and Sectioning elements 6.4>
<litprog.dtd: Miscellaneous elements 6.5>
<litprog.dtd: table elements 6.6>
<litprog.dtd: uri elements 6.7>
<litprog.dtd: Bibliographic elements 6.3>
<litprog.dtd: DocumentHistory elements 6.8>
<litprog.dtd: Entities 6.9>
◇
```

Notes:

- 1) The text parameter entity defines the standard element choices that go into the various documentation chunks. Add any extra document element choices in here, and they will be automatically distributed into all doco fragments.
- 2) The misc parameter entity defines the miscellaneous element choices that go into the various documentation chunks. These elements are usually only specified once per document, but there is no restriction on them being specified more often.

6.1 litprog.dtd: Literate Programming elements

<litprog.dtd: Literate Programming elements 6.2> =

```
<!ELEMENT o (#PCDATA|u|v|com)*>
<!ATTLIST o file CDATA #REQUIRED>
<!ELEMENT d (#PCDATA|u|v|com|formal)*>
<!ATTLIST d name CDATA #REQUIRED>
<!ELEMENT u (actual*)>
<!ATTLIST u name CDATA #REQUIRED>
<!ATTLIST u expand CDATA #IMPLIED>
<!ELEMENT v (#PCDATA)>
<!ATTLIST v var (def|use) "use">
<!ELEMENT formal (#PCDATA)>
<!ATTLIST formal name CDATA #REQUIRED>
<!ELEMENT actual (#PCDATA|%text;)*>
<!ATTLIST actual name CDATA #REQUIRED>
<!ELEMENT com (#PCDATA|%text;)*>
```

◇

Macro referenced in chunk 6.1

- o** o elements are file definitions. The **file** attribute defines the name of the file.
- d** d elements are code chunk definitions. The **name** attribute defines the name of the code chunk, by which it can be referenced in **u** elements
- u** u elements are uses of code chunks. They are normally empty elements, and may appear in either **o** or **d** elements, or in documentation. In tangling, they are replaced by the expanded form of the reference code chunk. In weaving, they are replaced by links to the referenced code definition. If a weave mode **u** element is accompanied by the attribute **expand**, which is non-empty, it is replaced by its expanded form.
- v** v elements are variable instances. The variable name forms the element content. If accompanied by the attribute **var** set to **def**, the instance is taken to be a defining instance, otherwise if the **var** attribute is set to **use**, or does not appear, the instance is regarded as a use of the variable.
- com** com elements are comments. They may be used within code or file chunks to annotate the code. Depending upon the context, these comments may be marked-up into the documentation.

6.2 litprog.dtd: Bibliographic elements

<litprog.dtd: Bibliographic elements 6.3> =

```
<!ELEMENT bibliography (bib)*>
<!ELEMENT bib (author,
               title,
               ((journal,year,volume?,number?,pages)|
                (publisher,year,ISBN?))|
               uri)
               )*>
<!ATTLIST bib idref ID #REQUIRED>
<!ELEMENT volume (#PCDATA|%text;)*>
<!ELEMENT number (#PCDATA|%text;)*>
<!ELEMENT pages (#PCDATA|%text;)*>
```

◇

Macro referenced in chunk 6.1

6.3 litprog.dtd: Heading and Sectioning elements

```
<litprog.dtd: Heading and Sectioning elements 6.4> =
  <!ELEMENT title (#PCDATA|nl)*>
  <!ELEMENT subtitle (#PCDATA|nl)*>
  <!ELEMENT author (#PCDATA|%text;)*>
  <!ELEMENT version (#PCDATA|%text;)*>
  <!ATTLIST version id NMTOKEN "1.0"
    delete (yes|no) "no"
    date CDATA #IMPLIED>
  <!ELEMENT date (#PCDATA|%text;)*>
  <!ELEMENT abstract (#PCDATA|%text;)*>
  <!ELEMENT section (title,(%text;|%misc;|subsection)*)>
  <!ATTLIST section numbers CDATA #IMPLIED>
  <!ELEMENT subsection (title,(%text;|%misc;|subsubsection)*)>
  <!ELEMENT subsubsection (title,(%text;|%misc;|subsubsubsection)*)>
  <!ELEMENT subsubsubsection (title,(%text;|%misc;|subsubsubsubsection)*)>
```

◇

Macro referenced in chunk 6.1

6.4 litprog.dtd: Miscellaneous elements

```
<litprog.dtd: Miscellaneous elements 6.5> =
  <!ELEMENT TableOfContents EMPTY>
  <!ELEMENT filelist EMPTY>
  <!ELEMENT macrolist EMPTY>
  <!ELEMENT identifierlist EMPTY>
```

◇

Macro referenced in chunk 6.1

6.5 litprog.dtd: table elements

```
<litprog.dtd: table elements 6.6> =
  <!ELEMENT table (tr|LabGroup)*>
  <!ATTLIST table border CDATA #IMPLIED
    align (left|center) #IMPLIED>
  <!ELEMENT tr (td|th)*>
  <!ELEMENT td (#PCDATA|%text;)*>
  <!ATTLIST td width CDATA #IMPLIED
    align CDATA #IMPLIED>
  <!ELEMENT th (#PCDATA|%text;)*>
```

◇

Macro referenced in chunk 6.1

6.6 litprog.dtd: uri elements

```
<litprog.dtd: uri elements 6.7> =
  <!ELEMENT uri (#PCDATA)>
  <!ATTLIST uri href CDATA #IMPLIED
    foot CDATA #IMPLIED
    alt CDATA #IMPLIED>
```

◇

Macro referenced in chunk 6.1

6.7 litprog.dtd: DocumentHistory elements

```
<litprog.dtd: DocumentHistory elements 6.8> =  
  <!ELEMENT DocumentHistory (Modified*)>  
  <!ELEMENT Modified (date,author,version,comment)>  
  <!ELEMENT comment (#PCDATA|%text;)*>
```

◇

Macro referenced in chunk 6.1

6.8 litprog.dtd: Entities

```
<litprog.dtd: Entities 6.9> =  
  <!ENTITY lt "&#x3c;";>  
  <!ENTITY ltt "$&#x3c;$" <!-- less than, TeX -->  
  <!ENTITY gt "&#x3e;";>  
  <!ENTITY gtt "&#x3e;" <!-- greater than, TeX -->  
  <!ENTITY at "&#x40;";>  
  <!ENTITY amp "&#x26;";>  
  <!ENTITY ampt "\&"; <!-- ampersand, TeX -->  
  <!ENTITY dollar "\$"; <!-- dollar, TeX -->  
  <!ENTITY ldq "&#x27;";>  
  <!ENTITY percent "\&#x25;"; <!-- percent, TeX -->  
  <!ENTITY euro "€";>  
  <!ENTITY verbar "&#x7c;"; <!-- vertical bar -->
```

◇

Macro referenced in chunk 6.1

7 The LitprogXML Document Type Definition

“LitprogXML.dtd” 7.1 =

```
<!ELEMENT LitprogXML (title,author,version,date,section*,DocumentHistory?)>
```

◇

8 File Index

File Name	Defined in
LitprogXML.dtd	7.1
lit2html.xsl	4.1
lit2tex.xsl	4.33
litprog.dtd	6.1
litprog.xsl	3.1

9 Macro Index

Identifier	Defined in	Used in
LIBXSL	1.1	4.1, 4.1, 4.1, 4.1, 4.33, 4.33, 4.33, 4.33
current date	11.2	
current version	11.1	
lit2html: TableOfContents	4.24	4.1
lit2html: body contains new line characters	4.17	4.15
lit2html: body contains several children	4.16	
lit2html: build code chunk definition list	4.20	4.14
lit2html: build code chunk reference list	4.18	4.14
lit2html: construct chunk definition list	4.27	4.1
lit2html: construct file definition list	4.26	4.1
lit2html: construct identifier definition list	4.28	4.1
lit2html: default template	4.32	4.1
lit2html: display code chunk reference list	4.19	4.14
lit2html: format the code body	4.15	4.14
lit2html: handle a code fragment use	4.13	4.1
lit2html: handle a code text element	4.25	4.1
lit2html: handle code chunk definitions	4.14	4.1
lit2html: handle code fragment references	4.11, 4.12	4.1
lit2html: handle document history	4.30	4.1
lit2html: handle documentation fragments	4.4, 4.5, 4.6, 4.7, 4.8, 4.1 4.9, 4.10	4.1
lit2html: handle macro parameters	4.29	4.1
lit2html: handle maintaintable	4.23	4.1
lit2html: handle notes and comments in code fragments	4.21	4.1
lit2html: handle the literate program translation	4.3	4.1
lit2html: handle the root element	4.2	4.1
lit2html: handle variable definitions and uses	4.22	4.1
lit2html: special character templates	4.31	4.1
lit2tex: build file list	4.51	4.33
lit2tex: build identifier list	4.52	4.33
lit2tex: build macrolist width variables	4.54	4.52, 4.53
lit2tex: build table of contents	4.47	4.33
lit2tex: build toc: section	4.48	4.47
lit2tex: build toc: subsection	4.49	4.47
lit2tex: build toc: subsubsections	4.50	4.47
lit2tex: callable template replacesubstring	4.59	4.58
lit2tex: default template	4.64	4.33
lit2tex: define Document History variables	4.57	4.56
lit2tex: define macrolist table template	4.55	4.52, 4.53
lit2tex: display code chunk definition list	4.40	4.39
lit2tex: display code chunk reference list	4.41	4.39
lit2tex: filter at signs in text nodes	4.61	4.58
lit2tex: filter underlines in non-verbatim text nodes	4.62	4.58
lit2tex: handle a literate program translation	4.36	4.33
lit2tex: handle chunk definition list	4.53	4.33
lit2tex: handle code fragment definitions	4.39	4.33
lit2tex: handle code use calls	4.42	4.33
lit2tex: handle document history	4.56	4.33
lit2tex: handle in-line comments in code	4.45	4.33
lit2tex: handle macro parameters	4.43	4.33
lit2tex: handle maintenance table documentation	4.46	4.33
lit2tex: handle paragraphs	4.37	4.33

lit2tex: handle reference calls	4.38	4.33
lit2tex: handle the root node	4.34	4.33
lit2tex: handle variable definitions and uses	4.44	4.33
lit2tex: handle various documentation elements	4.35	4.33
lit2tex: replace dollar signs with escaped dollar signs	4.60	4.58
lit2tex: special character templates	4.63	4.33
lit2tex: text node handling	4.58	4.33
litprog.dtd: Bibliographic elements	6.3	6.1
litprog.dtd: DocumentHistory elements	6.8	6.1
litprog.dtd: Entities	6.9	6.1
litprog.dtd: Heading and Sectioning elements	6.4	6.1
litprog.dtd: Literate Programming elements	6.2	6.1
litprog.dtd: Miscellaneous elements	6.5	6.1
litprog.dtd: table elements	6.6	6.1
litprog.dtd: uri elements	6.7	6.1
litprog: check definitions for non-use	3.30	3.25
litprog: check whether to include a use chunk	3.27	3.25
litprog: compute indentations for use expansion	3.29	3.25
litprog: count-trailing-blanks template	3.7	3.5
litprog: d-weave debug message-1	3.42	3.41
litprog: d/com—o/com-weave template	3.44	3.39
litprog: default tangle template	3.38	3.15
litprog: default weave template	3.52	3.39
litprog: define documentation template name list	1.2	3.3
litprog: define procedure templates	3.5	3.1
litprog: define script parameters	3.4	3.1
litprog: define variables	3.3	3.1
litprog: discard documentation in tangle mode	3.24	3.15
litprog: do-replace template	3.6	3.5
litprog: include raw text and expand indentation	3.31	3.25
litprog: include text and expand: define parmtable	3.34	3.31
litprog: include text and expand: define raw2	3.36	3.31
litprog: include text and expand: define rawchunk	3.35	3.31
litprog: include text and expand: loop message	3.33	3.31
litprog: include text and expand: replace text	3.37	3.31
litprog: include text and expand: start message	3.32	3.31
litprog: listing templates	3.51	3.39
litprog: lookup template	3.11	3.5
litprog: lookupchunkno template	3.8, 3.9	3.5
litprog: lookupchunknoR debug message 1	3.10	3.9
litprog: p-weave template	3.49	3.39
litprog: replace the machine parameter in chunk name	3.26	3.25
litprog: root template	3.2	3.1
litprog: save use parameters	3.28	3.25
litprog: section-weave template	3.50	3.39
litprog: substparms template	3.12	3.5
litprog: tangle a code definition	3.18	3.15
litprog: tangle a single file definition chunk	3.17	3.15
litprog: tangle all chunks into a single use	3.25	3.15
litprog: tangle all litprog files	3.16	3.15
litprog: tangle variable declarations	3.23	3.15
litprog: the tangle pass	3.15	3.1
litprog: the weave pass	3.39	3.1
litprog: u-weave template	3.47, 3.48	3.39
litprog: weave a code comment	3.43	3.41

litprog: weave a code or file definition	3.41	3.39
litprog: weave a litprog	3.40	3.39
litprog: weave a variable markup	3.45, 3.46	3.39
substparms: build result string recursively	3.14	3.12
substparms: unescape the escaped formal parameters	3.13	3.12
tangle code, debugging	3.22	3.18
tangle code, end trimmed chunk	3.20	3.18
tangle code, initial chunk	3.19	3.18
tangle code, start and end trimmed chunk	3.21	3.18
weft: handle the root element	5.1	4.2, 4.34

10 Identifier Index

Identifier	Defined in	Used in
Verbose	3.4	
by	3.6	3.6, 3.6, 3.6
count	3.7	
count-trailing-blanks	3.7	
crossrefs	3.3	3.8
do-replace	3.6	3.6
filelist	3.51	
identifierlist	3.51	
lookupchunkno	3.8	
machine	3.4	
macrolist	3.51	
postmacro	3.3	
premacro	3.3	
reflist	4.18	4.19, 4.40, 4.41
replace	3.6	3.6, 3.6, 3.6
replacesubstring	4.59	
string	3.7	
text	3.6	3.6, 3.6, 3.6

Document History

Date	Author	Version	Comment
20050115:172716	ajh	2.0	first literate version of release 2
20050121:143117	ajh	2.0.1	revised grammar of output .xml file to embrace NOTES with a NOTES element. This facilitates processing at phase 2
20050705:170139	ajh	2.0.2	refined <code>lit2html.xsl</code> to better format short code chunks. Changed <code>v</code> element within documentation to be <code>n</code> , to avoid clash with variable markup elements, also named <code>v</code>
20050708:155431	ajh	2.0.3	fixed <code>macrolist</code> and <code>identifierlist</code> templates in <code>litprog.xsl</code> . Significant formatting and re-chunking of file.
20050711:102903	ajh	2.0.4	working over <code>lit2tex.xsl</code> and <code>codeverbatim</code>
20050712:151429	ajh	2.1.0	further “literacising”, and removed the <code>n</code> element from XML intermediate file.
20050715:130910	ajh	2.1.1	Recovery from the schemozzle caused by trying to upgrade to Tiger on ballarat
20050722:121800	ajh	2.1.2	Further work on the user manual
20050725:164813	ajh	2.1.3	Refinement of the dtd
20050728:141654	ajh	2.1.4	Revised and extended User Manual, revised cross referencing linkages
20050913:143424	ajh	2.1.5	Updated T _E X weft: reformat code and file chunks to give more accurate presentation of starting and ending line breaks. Also added non-CODE USE handling.
20050915:165408	ajh	2.1.6	Various minor bug fixes; added width control to <code>macrolist</code> and Document History.
20051012:153812	ajh	2.2.0	Added macro parameters to code chunks. Currently only <code>litprog</code> and <code>lit2html</code> handle these.
20051014:105557	ajh	2.2.1	Added macro parameter handling to <code>lit2tex</code> .
20061014:133445	ajh	2.3.0	start on separate parts structure
20061027:083327	ajh	2.3.1	refined macro parameters
20070509:184652	ajh	2.3.2	finally got round to fixing macro parameters, by separating into formals and actuals. Extended this to the T _E X phase as well
20070524:145633	ajh	2.3.3	fix logic bug in recursive macro expansion

20080528:095515 ajh 2.3.4 moved list of templates that are passed through without comment to be common to both tangle and weave in litprog.xml. Added 'parm' to that list.

20080822:162322 ajh 2.4.0 fixed bug in multiple chunk output files

<current version 11.1> =2.4.0 ◊

<current date 11.2> =20080822:162322 ◊