RAL-92-071

Science and Engineering Research Council
Rutherford Appleton Laboratory
Chilton DIDCOT Oxon OX11 0QX

RAL-92-071

SGML in Practice
The PHIGS Slide Set

R E Thomas

November 1992

SGML in Practice
The PHIGS Slide Set

R E Thomas

25 September 1992

SGML in Practice - The PHIGS Slide Set

1. INTRODUCTION

This paper describes the work undertaken to convert the text of a large set of slides (originally prepared using the Documenter's Workbench[1] tools - troff and preprocessors - under the Unix[2] Operating System) to make use of the Standard Generalized Markup Language (SGML). These slides are available to the Higher Educational Institutions (HEIs) either in hardcopy form as foils or as machine-readable text for use in teaching the basics of PHIGS (a three-dimensional graphics standard). The latter form is preferred, since most sites wish to be selective in the use of foils, incorporating them into existing lecture series. Since there are many types of formatter used in HEIs, it is not sufficient merely to supply the troff form. In particular, it is difficult to separate the semantics of a slide from its layout, which can lead to inconsistencies when edited or translated.

SGML is a structured markup language suitable for identifying the semantics of a document. It was therefore selected as a suitable means of making the meaning clear, and simplifying translation to a particular formatter. In addition, the use of a structured markup allowed slide consistency to be verified. The problems encountered, the results of the work and items for further investigation are described below.

2. BACKGROUND

PHIGS is an international standard for three-dimensional graphics (ISO 9592). Professors Hopgood and Duce have been very active in developing the standard (working on the ISO committees) and in promoting its use. They are the authors of "A Primer for PHIGS", aimed at Fortran Programmers, and the more recent C Programmers Edition. As part of their promotional activities, they have jointly developed a comprehensive set of lectures with accompanying slides, which have been used both in the UK and abroad. The Advisory Group On Computer Graphics (AGOCG, a joint HEI - Research Council Committee) has asked that these slides be made available to the HEIs for use either as complete lectures or as additions to existing lecture series.

Since the lectures describe a graphics package, much of the material on the slides is generated by PHIGS programs. The original slides were prepared using the troff text formatter running on a SUN Workstation. This package had been chosen because the authors already had experience using troff to publish other books. Its advantages were its availability, its preprocessors (especially tbl, eqn and pic for tables, equations and simple line diagrams respectively) and the ability to incorporate PHIGS graphics (as PostScript, using the psfig preprocessor).

The slides had been created over a period of time, having grown out of a variety of separate exercises. Consequently, there had been no overall planning of the set at the start. This led to a variety of layout styles. Some effort was made to give a measure of uniformity by the provision of a suitable macro set (a modified form of the ms macros) which gave a standard border

1. Documenter's Workbench is a registered trademark of AT&T in the USA and other countries
2. Unix is a registered trademark of AT&T in the USA and other countries

and slide naming, but did little to correct the differences in the body of the slide. The generation of new slides has still produced considerable variations, since the authors are not able to work together for long periods to iron out the differences, and there is little scope for control within troff itself. The net result is less pleasing than the authors would have liked. A previous attempt to produce a coherent set (for a different set of lectures) by employing professionals was not successful, since those with the necessary skills to produce good layout did not have any knowledge of the subject matter of the slides, and unwittingly distorted the meaning in their attempt to make the slides look better.

It is of course possible to distribute the slides in hardcopy form, as foils. The disadvantage of this is that most HEIs have their own house style for slide production, and these slides would have stood out as foreign additions. This is particularly true when incorporated into other lectures, where slides from other sources are also used. In addition, the use of slides in this way often requires some slight changes to the wording in order to merge them seemlessly with the rest. It is therefore much preferable to distribute the sets in machine readable form so that local editing is possible. However, the use of troff also poses a problem. While troff is widely available, it is by no means the first choice package at all HEIs. In particular, the modified ms macro package would have to be distributed with the slides in order for the recipient to interpret the set correctly (in fairness to troff, this would be true whatever package was used unless the authors restricted themselves to standard macros and basic commands).

There was therefore a requirement to select a method of production which

a) made clear the semantics of each slide, thus ensuring that anyone wishing to edit the text into another document would be sure of the significance of that text,

b) provided a simple means of constraining the authors to follow a single style,

c) was easily transmittable to HEIs in machine-readable form,

d) allowed simple translations to be provided for conversion into a form which the local formatter could handle.

For these reasons, SGML was selected.

3. SGML

SGML is an ISO standard (ISO 8879) for indicating the internal structure of a document. It is concerned with things such as whether you can have paragraphs in a section (semantic) rather than whether these paragraphs are indented (layout). Documents are classified into different types, and, for each type, a Document Type Definition (DTD) defines the meaning of the markup which will be used in each instance of the type. This markup will appear as a series of tags interspersed with the text. Since a DTD defines a tag set, it is possible for documents of different types to have the same physical representation of a tag (eg "<section>") but for the meaning to be completely different in the two examples (eg major division containing chapters in one: minor division within a chapter in the other). Thus a document marked up in SGML is only meaningful if accompanied by the appropriate DTD (so that the tags can be interpreted).

SGML is therefore capable of capturing the semantic content of a document (for example, indicating that a piece of text is actually a title for a particular picture rather than straight narrative), being parsed (thus automatically checking the accuracy of the document) and providing a canonical form (all tags present and no abbreviations) which is readily translatable into formatter commands.

In order to make use of SGML, it is necessary to construct a suitable DTD. This involves analysing the example documents to extract the common structural forms. Once this has been achieved, the DTD can be constructed. The DTD used for the PHIGS slide set is listed in Appendix 1. It is not intended to give a full definition of SGML here, but the following explanation, together with the details in Appendix 2, should allow the reader to follow the example. The main constructions used in the DTD are:

a) ELEMENT. This defines the tag which will appear in the actual document to indicate a particular section. Angle brackets ("<",">") are normally used to identify tags. A section is ended by the repetition of the tag with a "/" inserted after the "<". The main part of the ELEMENT description is the content. This lists the tags which can appear within a section marked by this tag. Tag structure, such as tag sequences, alternatives, multiple occurrences, etc are also indicated. The term "#PCDATA" refers to straight text (which may contain tags) and CDATA indicates text where all tags other than end tags are ignored.

b) ATTLIST. This lists the arguments to be attached to the specified tag, together with their types and default values (if any).

c) ENTITY. This defines the name and a substitution string which will replace that name wherever it occurs.

d) SHORTREF, USEMAP. These constructs allow aliases to be used for tags within particular sections. It is possible to use the same alias to mean different things in different sections. Its main use is to reduce the number of keystrokes if tagging is not done automatically.

Design of a DTD, in particular the substructure of the tags, is similar to the formal specification of a language.

While it is possible to create a SGML document using any editor, there are various tools available to assist in the preparation, verification and translation. Within Informatics, there is a structured editor (Author/Editor on a Macintosh from SoftQuad) which assists with data entry (preventing illegal constructs) and provides menu lists for the tags (only presenting those which are legal at the time). There is also a parser (MarkIt on a SUN workstation from Sema) capable of verifying the correctness of a document against a given DTD. This product provides a means of translating the tags into other text sequences (by extending the definition of the LINK construct within SGML itself), and this has been used to format the SGML slides (an example is provided in Appendix 4).

4. DOCUMENT ANALYSIS

As stated in the previous section, the first step is to analyse the existing documents in order

to determine the required structure. This is not a simple task, since there is no single "correct" method of doing it. In addition, the variable nature of the slides meant that there was unlikely to be a useful structure which would encompass all the examples. The overall problem was simplified by:

a) looking for a structure which the majority satisfied. The authors agreed to redesign those which did not fit the chosen model.

b) attempting where possible to minimise the number of tags. Hence structures which, in other document types, might have been identified as separate features were treated as examples of a single feature. Thus simple lists and ordered lists were treated as one and two-column tables respectively.

c) avoiding well-known difficult areas such as table and equation tagging by treating each as a single identifiable tagged section. This simplification poses problems for those wishing to map the SGML document into a formatter other than troff. The solution to the problem has been left for the next version (when the SGML community view on the right way of handling such things may be clearer).

d) treating all pictures as single tagged sections. There is little alternative to this approach.

The chosen tag structure is given in Appendix 2, and this led to the construction of the DTD. Each slide was considered to have three parts (although some of the parts might be omitted from individual slides): a functional description of the slide, a block containing pictures or examples, and some descriptive text relating to the block. Two types of bulleted list and one enumerated list were identified in the description section. Two types of highlighting could also be used within the third section. Each slide set had a initial Title slide, followed by the members of the set. Each slide had a border, a title, a number and a copyright notice. The variable data for the initial slide was contained in the attributes of the "pslides" tag; no separate structured information was required. Data within a block could be an equation, a table, a PostScript picture, a diagram in pic format or an example program (eg Fortran).

The main part of the DTD is a realisation of the desired structure. However, there have been two extensions. The first concerns the use of short references. Since the first tests involved preparing slides by hand, it was decided to include aliases for some of the tags. Square brackets, curly brackets and hash were used in various places as a shorthand for tags. Thus at the start, curly brackets indicated a functional description while square brackets indicated the start of a block. Within the last section, however, square brackets indicated the start of a bulleted list. In addition, a separator tag had been defined within this section, which corresponded to a blank line in the original slide. It was therefore natural to select a blank line as a shorthand for this tag.

The second problem concerns the use of a blank line, as described above. Standard SGML identifies a blank line by detecting a Record Start (RS) character followed immediately by a Record End (RE). In a Unix system, however, there is usually no equivalent of the Record Start (since only a single character appears between adjacent lines of text). The problem can be overcome by changing the definitions in the SGML Declaration (part of the header before the

DTD, which is often inserted by default by the SGML tools), but many systems have difficulty coping with non-standard declarations. Consequently, a different method of detection was employed. Since the problem relates only to Unix systems, and since the slides are designed for general use, it is necessary to provide both mechanisms and switch between them. This is accomplished by using the "Marked Section" construct, which is driven by setting entities within the DTD.

5. SGML PRODUCTION

The first slides were produced by hand using an ordinary text editor. These helped to validate the DTD. Once the DTD had been agreed, new slides could be produced using the structured editor (Author/Editor). However, this left the problem of conversion of the original set (several hundred slides).

The use of a macro set within troff and the choice of DTD to match the majority of the slides meant that there was some hope that an automatic translation method could be found to achieve the conversion. Eventually, a script was prepared using the awk utility within Unix (see Appendix 3. A knowledge of awk is required to understand this). Allowance was made where possible for known variations, and those slides which could not be translated sensibly were returned to the authors for redesign. Several problems came to light during this activity, which resulted in a complete set of slides in SGML. These solutions are outlined below.

5.1 Redundant Markup

The normal way to design slides using a markup system is to make an initial design, see the result and then reposition items until the slide looks right. Thus while the original design might keep to some agreed rules, the subsequent format changes tend to be done using whatever tags produced the desired visual result. In some cases, the evolution of the slide had mean that text had been removed while the underlying formatting instructions had been retained. This made automatic translation difficult. In general, the most likely addition was the insertion of extra white space (blank lines) and these could be detected. There were several cases where the remains of old lists still existed within the troff document; a potential trap for the unwary next time the slide is edited.

5.2 Non-standard Slides

Several slides could not be translated into legal SGML. In some cases, this led to slide redesign. In others, it highlighted the existence of further structure items which needed to be included in the DTD (but were sufficiently rare not to have been picked up in the original analysis). One example of this was the need to allow tables in second-level bulleted lists (originally, they were only legal at the top level).

Sometimes (though fortunately infrequently) the only solution was to perform the conversion by hand. For example, this could occur when one of the sections was too large to fit onto a single slide. On occasion, a picture within the middle block section would require a full A4 page. This would be achieved by having the description follow the functional definition on one slide and the picture on the next. The effort required to detect this automatically was considered too great, and its infrequence led to the decision not to change

the overall DTD to cope with this reordering of the major sections (which is format rather than semantic related).

5.3 SGML and Newlines

One of the features of SGML which gives rise to difficulties is the treatment of end-of-line codes. A newline between two tags for example is not considered significant, and a parser will treat this just as if the newline was not there. While the rules are clearly set out in the standard, it is not always intuitively obvious whether a particular newline is significant or not. In most cases, this does not matter. However, the troff commands for indicating bold and italic text lead to translation problems.

There are several ways of indicating highlights in troff. One method is to use the in-line macros ("\f2"). Another is to use the standard macros (such as ".I"). These can be followed either by the text to be highlighted or by a newline, which indicates that all the following text is to be highlighted until another command is used. This newline does not cause a break in the formatting however.

The method chosen for the original troff slides led to a need for another pass through awk, in order to detect whether highlighting occurred at the start of a tagged section or in the middle of one (the newlines being treated differently).

6. REVERSE TRANSLATION

Having produced the slides in SGML form, it is necessary to consider their translation to a formatter. It was decided to translate them back into troff. This not only verifies the whole concept (since the original troff can be compared with the generated markup) but is also an essential requirement for the production of foils from new slides. Since MarkIt was available, it was decided to make use of the extended LINK facilities to write a translator. The result is in Appendix 4. Sufficient detail is given below to enable the reader to understand the example.

The translator detects both start and end tags, and enables additional text to replace these tags (indicated by the keywords START and END). In some cases, the tag does not result in the generation of any new text (eg for block or body). Text to be inserted is enclosed in quotation marks (either single or double). The keyword #OUTPARSE indicates the output stream of the translator. Attribute values can be accessed on a start tag by referencing the attribute name preceded by "%#" and followed by ";". Global variables can be set. Conditional actions are possible (#ENTCOND and #ATTCOND) and the contents of external files inserted (SYSTEM ENTITY).

The aim of the translator is to produce logically correct slides that are close to being aesthetically acceptable, rather than duplicating the original troff. The same modified macro set has been used. The main differences are:

a) No attempt is made to provide automatic adjustment of the spacing to improve the layout. This has to be done by hand if it is considered necessary.

b) Similarly, no attempt has been made to deal with the large picture problem (see previous section).

c) Data for those tags which can refer to their content either in-line or by external file name is always inserted in-line.

d) The in-line highlight macros ("\f2","\f3") are used.

e) The initial slide for the set is generated from the information in the pslides tag. The translation command in troff is used to force the title to upper case for this slide (it appears in mixed case on the others).

f) There are several places where the appropriate troff is not generated as the result of the detection of the 'obvious' tag. For example, the troff start-slide macro call is generated from the title tag rather than the slide tag (since the equivalent troff macro requires the title as an argument).

In practice, this mapping has proved sufficient. There are only a handful of cases where manual intervention is needed in the final product.

7. FUTURE SLIDE PRODUCTION

Now that there is a SGML definition of the slides, it is possible to generate correct slides using the tools available. The current steps in production are:

a) Using a lap-top Macintosh (a PowerBook) with Author/Editor, develop the text of the new slides.

b) Connect the PowerBook to the Department network and transfer the SGML text to the SUN filestore (this action can be done very simply by allowing the PowerBook to make direct use of the SUN NFS file system).

c) Run the slides through MarkIt to generate troff.

d) Use a PostScript previewer to check on the layout and edit the troff to insert necessary corrections.

Files for insertion (eg PostScript pictures) can be generated independently. It is necessary to ensure that the SGML code has reference to the correct filename format for the SUN, since that is where the translation takes place.

8. COMMENTS

The slide authors have already noticed a considerable improvement in the quality of the slides produced. It is recognised however that the problems of handling tables and equations will have to be tackled to make the result less dependent on a knowledge of troff.

The problem of producing visually acceptable slides in all cases by automatic means would appear at best to involve considerable programming ingenuity and at worst be impossible. Thus it is likely that some manual intervention will be required at some stage, with a need to edit the troff directly. This could lead to the troff and SGML forms getting out of step, especially if semantic changes are made to the former. Discipline in production is therefore

still required.

From the experience with troff, the main problems in writing a translator for other formatters will be in coping with possible differences in treatment of newlines. Currently, MarkIt is unable to provide a method of indicating that the character detected prior to the current tag was a newline. It is possible that other translation systems may be able to offer more help (and, in its canonical form, SGML is easily handled by Unix tools such as yacc and lex, where this capability can be provided).

9. CONCLUSIONS

In summary, the use of SGML has improved the slide quality and provided a useful means of transmitting the semantic information in machine readable form to remote sites. Generation of the original text is much simpler. It is certainly much easier to consider translating from SGML than from troff (even if you have some prior knowledge).

The drawbacks concern the need to make occasional modifications in order to handle the visual layout, and the requirement for SGML forms for tables and equations. The former task would appear complex in the general case, but this would be true whatever system is used. There may be a case for adding attributes to assist in this task (eg to identify large picture files).

APPENDIX 1

Document Type Definition for Slide Set

```
<!DOCTYPE pslides [

<!-- slideset for the PHIGS slides. -->

<!ELEMENT pslides - O (slide+) >

 <!ATTLIST pslides
            title       CDATA       #REQUIRED
            number      NUMBER      1
            height      CDATA       "6.0i"
            bounds      NUMBERS     "0 0 544 544"

      -- attributes include the name of the set (for inclusion on each slide) and
      details of the default size of Postscript pictures.
      'number' is the number of the first slide. Others will be numbered
      sequentially. -->
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -- >

<!-- for use in marked section to overcome RS/RE problem -->

<!ENTITY % unix "IGNORE" >

<!ENTITY % other "INCLUDE" >

<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -- >

<!ELEMENT slide   - O ( title, body )

  --   Basic slide has to have a title.
            Slide number generated from initial figure.
            Everything else is optional. -- >

<!ATTLIST slide
            copyr    CDATA   "F.R.A. Hopgood and D.A. Duce, October 1991"
            --    copyright notice for the slide   -- >

<!ELEMENT title   O O (#PCDATA) >

<!ELEMENT body    O O (funcdef?, block?, desc?) >

<!-- body of slide can have a function definition (first if
      it appears, a body (with possibly several examples, pictures, etc)
      and possibly a description (which comes last)      -->

    <!ELEMENT funcdef - O (#PCDATA) >
```

```
<!ELEMENT block  O O  (program|picfig|psfig|table|eqn)* >

    <!ELEMENT program  - -  CDATA >
    <!ATTLIST program
      file        CDATA        ""
      -- for fortran or C examples, either in-line or in file -- >

    <!ELEMENT picfig  -  -  CDATA>
    <!ATTLIST picfig
       file       CDATA        ""
      -- for pic format pictures, either in-line or in file -- >

    <!ELEMENT psfig  - O  EMPTY >
    <!ATTLIST psfig
      file        CDATA        #REQUIRED
      -- postscript files, always included from file.
                  Size set at start for all figures  -- >

    <!ELEMENT table - - CDATA >
    <!ATTLIST table
      file        CDATA        ""
      -- data in tbl format, either in-line or in file -- >

    <!ELEMENT eqn - - CDATA >
    <!ATTLIST eqn
      file        CDATA        ""
      -- data in eqn format, either in-line or in file -- >

<!ELEMENT desc   - O  (text?, (list | elist)* ) +(hp1|hp2) >
<!ATTLIST desc
      posn        (left|centre)    centre
      -- section is either left justified or centred.
                  Highlights may occur anywhere -- >

    <!ELEMENT (hp1|hp2)  - - (#PCDATA | hp1 | hp2)  >

    <!ELEMENT text    O O (#PCDATA) >

    <!ELEMENT list    - O (it | sep | table | eqn)+
                  -- bulleted list -- >

        <!ELEMENT it  - O  (#PCDATA) -(list) +(lists)
                  -- lists of same type may not be nested -- >

        <!ELEMENT  lists  - O  (its | sep| table | eqn)+
                  -- second level bulleted list -- >
```

```
        <!ELEMENT  its  - O  (#PCDATA) -(lists)
                    -- lists of same type may not be nested -- >

        <!ELEMENT sep - O  EMPTY >

        <!ELEMENT elist   - O (et, lists)+
                    -- enumerated list -- >

        <!ELEMENT et   - O (#PCDATA)
                    -- element to be defined -- >
<!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -- >

<!-- short references for slide. -->

<!ENTITY startfc   STARTTAG     "funcdef" >

<!ENTITY endfc     ENDTAG       "funcdef" >

<!ENTITY startbl   STARTTAG     "block" >

<!ENTITY endbl     ENDTAG       "block" >

<!SHORTREF slidemap          "{"   startfc
                             "}"   endfc
                             "["   startbl
                             "]"   endbl >

<!USEMAP  slidemap       slide  >

<!-- short references for desc. -->

<!ENTITY starttx   STARTTAG     "text" >

<!ENTITY endtx     ENDTAG       "text" >

<!ENTITY startl    STARTTAG     "list" >

<!SHORTREF descmap           "{"         starttx
                             "}"         endtx
                             "["         startl >

<!USEMAP  descmap       desc  >

<!-- short references for list. -->

<!ENTITY startit   STARTTAG     "it" >

<!ENTITY startsp   STARTTAG     "sep" >
```

```
<!ENTITY startls   STARTTAG      "lists" >

<!ENTITY endl     ENDTAG        "list" >

<![ %unix; [<!SHORTREF listmap   "#"          startit
            "&#RE;&#RE;"         startsp  --  "&#RS;&#RE;" --
            "["                  startls
            "]"                  endl > ]]>

<![ %other; [<!SHORTREF listmap   "#"          startit
            "&#RS;&#RE;"         startsp
            "["                  startls
            "]"                  endl > ]]>

<!USEMAP  listmap              list  >

<!-- short references for lists. -->

<!ENTITY startnt   STARTTAG      "its" >

<!ENTITY endl1     ENDTAG        "lists" >

<![ %unix; [<!SHORTREF listsmap   "#"          startnt
            "&#RE;&#RE;"         startsp  --  "&#RS;&#RE;" --
            "]"                  endl1  > ]]>

<![ %other; [<!SHORTREF listsmap   "#"          startnt
            "&#RS;&#RE;"         startsp
            "]"                  endl1  > ]]>

<!USEMAP  listsmap             lists  >

<!-- short references for elist. -->

<!ENTITY startet   STARTTAG      "et" >

<!ENTITY startee   STARTTAG      "lists" >

<!ENTITY endel     ENDTAG        "elist" >

<!SHORTREF elistmap   "#"        startet
            "["                  startee
            "]"                  endel >

<!USEMAP  elistmap             elist  >

]>
```

APPENDIX 2

SGML Tag Details

Tag Definitions

<block>	Main body of a slide
<body>	Body of slide
<desc>	Optional descriptive section of slide. Highlights may occur anywhere within this section. posn - placement of this section, either left or centre (the default).
<elist>	Enumerated list, containing enumerated type and sublist.
<eqn>	Figure in EQN format, either in-line of in a separate file (troff start and end tags assumed not to be specified). file - name of file containing EQN data. Assumed in-line if not specified.
<et>	Enumerated type (ie the thing being defined).
<funcdef>	Optional text at start of slide
<hp1>	Highlight (bold).
<hp2>	Higlight (italic).
<it>	Item in list.
<its>	Item in sublist.
<list>	List, possibly containing sublist, equation or table.
<lists>	Sublist, possibly containing equation or table.
<picfig>	Figure in PIC format, either in-line of in a separate file (troff start and end tags assumed not to be specified). file - name of file containing PIC data. Assumed in-line if not specified.

<program> Program example (C or Fortran), either in-line
of in a separate file.
file - name of file containing example.
Assumed in-line if not specified.

<psfig> Figure in Postscript format in separate file.
file - name of file containing postscript, must be specified.

<pslides> Name of document type.
title - name of slide set (must be present)
number - number of first slide of set, default 1
height - height of all postscript pictures, default 6.0i
bounds - bounds for all postscript pictures, default
0 0 544 544

<sep> Item separator in list or sublist.

<slide> Start of a slide.
copyr - "F.R.A. Hopgood and D.A. Duce, March 1990"

<table> Figure in TBL format, either in-line
of in a separate file (troff start and end tags
assumed not to be specified).
file - name of file containing TBL data.
Assumed in-line if not specified.

<text> Optional text at start of description.

<title> Title for slide, required.

Tag Structure

<block> picfig, program, psfig, table, eqn

<body> funcdef, block, desc

<desc> text, list, hp1, hp2

<elist> et, lists, hp1, hp2

<et> #PCDATA, hp1, hp2

<funcdef> #PCDATA

<hp1> #PCDATA, hp2

<hp2> #PCDATA, hp1

<it>	#PCDATA, lists, hp1, hp2
<its>	#PCDATA, hp1, hp2
<list>	it, sep, table, eqn, hp1, hp2
<lists>	its, sep, table, eqn, hp1, hp2
<picfig>	CDATA
<program>	CDATA
<psfig>	
<pslides>	slide
<sep>	
<slide>	title, body
<table>	CDATA
<text>	#PCDATA
<title>	#PCDATA

APPENDIX 3

Conversion scripts for original Slide Set using the Unix tools awk and sed.

```
#
# script to convert the PHIGS slide sets from troff
#  to SGML. Handles most things, except the in-line codes
#\f1 \f2 \f3  for emphasis (handle with sed). Blank <text>
# sections picked up by second pass through awk (different file).
# <hp> tags handled by third pass through awk.
#
# uses flags to indicate hierarchy. Ignore everything
# other than pslides before the first proper slide
#
#
BEGIN  {      block = 0
              desc = 0
              list = 0
              lists = 0
              elist = 0
              figure = 0
              picfig = 0
              psfig = 0
              prog = 0
              slide = 0 }

#
# <!ELEMENT pslides - O (slide+) >
# Header info. Note that more than one word title possible
#
$1 == ".Sh"  { len = NF - 1
              for (i = 2; i <= len; i++) a[i] = $i
              str = a[2]
              if (len > 2)  for (i = 3; i <= len; i++) str = str " " a[i]
              else if (str !~ /\"/) str = "\"" str "\""
              printf ("<pslides title=%s number=%s >\n",str,$NF) }

#
# Slide end. Close previous block if necessary.
#  set flag to ignore all the header stuff.
#
$1 == ".Se"  { if (block == 1) print "</block>"
              block = 0
              slide = 1 }


#
```

```
#<!ELEMENT slide   - O ( title, body ) >
#<!ELEMENT body    O O (funcdef?, block?, desc?) >
#<!ELEMENT title   O O (#PCDATA) >
# slide title. Data comes between tags, not as attribute. Note
# that we need to remove the double quote marks.
# Also, may be a copyright notice, which is assumed to be quoted.
# Therefore possible forms are:
#     .Ss title
#     .Ss "title"
#     .Ss title "copyr"
#     .Ss "title" "copyr"
#
#
$1 == ".Ss" { if (slide == 1) {
                    p = 1
                    pp[1] = 0
                    for (i = 2; i <= NF; i++) {
                        a[i] = $i
                        if (a[i] ~ /^"/) pp[p++] = i
                        }
# NF = 2, p = 1: unquoted title, no copyr
# NF = 2, p = 2: quoted one-word title, no copyr
# NF > 2, p = 3, pp[1] = 2: quoted title, no copyr
# NF > 2, p = 3, pp[1] = 3: unquoted title, copyr
# NF > 2, p = 4: quoted one-word title, copyr
# NF > 2, p = 5: quoted title, copyr

                    if (NF == 2 || (pp[1] == 2 && p == 3)) {
# slide without copyright. Remove double quotes if present
                        if (p > 1) {
                            split (a[2], b, "\"")
                            a[2] = b[1]b[2]
                            split (a[NF], b, "\"")
                            a[NF] = b[1]b[2]
                        strt = a[2]
                        for (i = 3; i <= NF; i++) strt = strt " " a[i]
                            }
                            else strt = a[2]
# print title, no copyright.
                        printf ("<slide>\n<title>%s</title>\n<body>\n", strt) }
                    else {
# slide with copyright. Remove title double quotes if present
                        cp = pp[1]
                        if (p == 4) {
                            tp = pp[1]
                            cp = pp[2] }
                        if (p == 5) {
```

```
                    tp = pp[2]
                    cp = pp[3]}
                if (p > 3) {
                    split (a[2], b, "\"")
                    a[2] = b[1]b[2]
                    split (a[tp], b, "\"")
                    a[tp] = b[1]b[2]
                strt = a[2]
                for (i = 3; i <= tp; i++) strt = strt " " a[i]
                    }
                    else strt = a[2]
# copyright as well; do not split quotes
                    strc = a[cp]
                    for (i = cp + 1; i <= NF; i++) strc = strc " " a[i]
                printf ("<slide copyr=%s>\n<title>%s</title>\n<body>\n", strc, strt)
                }

}}
#
#<!ELEMENT funcdef - O (#PCDATA) >
#
$1 == ".Fs" { print "<funcdef>" }

$1 == ".Fe" { print "</funcdef>" }

#
#<!ELEMENT block   O O  (program|picfig|psfig|table|eqn)* >
# <!ELEMENT program  - -  CDATA >
# need to start block if not already started
#
$1 == ".Ps"  { if (block == 0) print "<block>"
               block = 1
               prog = 1
               print "<program>" }

$1 == ".Pe"  { prog = 0
               print "</program>" }

#
#<!ELEMENT picfig  -  -  CDATA>
#
$1 == ".PS"  { if (block == 0) print "<block>"
               block = 1
               picfig = 1
               print "<picfig>" }
```

```
$1 == ".PE"  { picfig = 0
               print "</picfig>" }

#
#
#<!ELEMENT psfig  - O  EMPTY >
#
$1 == ".F+"  { if (block == 0) print "<block>"
               block = 1
               psfig = 1 }

$1 == "figure" && psfig == 1 { printf ("<psfig file=\"%s\">\n",$2) }

#
# no end tag for this, since it is empty
#
$1 == ".F-"  { psfig = 0 }

#
#<!ELEMENT table - - CDATA >
# table can come in block or in a list or lists
#
$1 == ".TS"  && slide == 1 { if (desc == 0) {
                    if (block == 0) print "<block>"
                    block = 1 }
               else {
                    if (list == 0) print "<list>"
                    list = 1 }
               print "<table>" }

$1 == ".TE"  && slide == 1 { print "</table>" }

#
#<!ELEMENT eqn - - CDATA >
# eqn can come in block or in a list or lists # ignore before first slide
#
$1 == ".EQ" && slide == 1 { if (desc == 0) {
                    if (block == 0) print "<block>"
                    block = 1 }
               else {
                    if (list == 0) print "<list>"
                    list = 1 }
               print "<eqn>" }

$1 == ".EN"  && slide == 1 { print "</eqn>" }
```

```
#
#<!ELEMENT desc    - O  (text?, (list | elist)* ) +(hp1|hp2) >
#<!ELEMENT text    O O (#PCDATA) >
# end block if necessary. No newline after <text> so we can
# detect null section in second scan.
#
$1 == ".Cs" { if (block == 1) print "</block>"
                block = 0
                desc = 1
                printf ("<desc centre>\n<text>")  }

$1 == ".Ls" { if (block == 1) print "</block>"
                block = 0
                desc = 1
                printf ("<desc left>\n<text>")  }

#
# close whole slide at this point
#
$1 == ".Ce" || $1 == ".Le" { if (list == 1) print "</list>"
                list = 0
                desc = 0
                printf ("</desc>\n</body>\n</slide>\n") }

#
#<!ELEMENT list    - O (it | sep | table | eqn)+ >
#<!ELEMENT it  - O  (#PCDATA) -(list) +(lists) >
# set list start if this is the first bullet
#
$1 == ".Bu"  { if (list == 0) print "</text>\n<list>"
                list = 1
                print "<it>"  }

#
#<!ELEMENT  lists  - O  (its | sep | table | eqn)+ >
#
$1 == ".RS" { lists = 1
                print "<lists>" }

$1 == ".RE" { print "</lists>"
                if (elist == 1) print "</elist>"
                lists = 0
                elist = 0 }
```

```
#
#<!ELEMENT  its  - O  (#PCDATA) -(lists) >
#
$1 == ".Sq" { print "<its>" }

#
#<!ELEMENT elist   - O (et, lists)+ >
#<!ELEMENT et   - O (#PCDATA) >
# set up enumerated list if first time.
# note that we need to remove double quotes and cater for
# multiple words
#
$1 == ".Et" { if (elist == 0) print "</text>\n<elist>"
                    elist = 1
                    for (i = 2; i <= NF; i++) a[i] = $i
                    if ($NF ~ /\"/) len = NF; else len = NF - 1
                    split (a[2], b, "\"")
                    a[2] = b[1]b[2]
                    split (a[len], b, "\"")
                    a[len] = b[1]b[2]
                    str = a[2]
                    if (len > 2)  for (i = 3; i <= len; i++) str = str " " a[i]
                    printf ("<et>%s</et>\n",str) }

#
#<!ELEMENT (hp1|hp2)  - - (#PCDATA | hp1 | hp2)  >
# ignore emphasis codes before first slide
#
$1 == ".I"  && slide == 1  { print "<hp2>" }

$1 == ".B" && slide == 1  { print "<hp1>" }

$1 == ".R"  && slide == 1  { print "</>" }

#
#<!ELEMENT sep - O  EMPTY >
# treat blank lines as separators if within list
# and preserve if in a program or picfig(otherwise ignore)
#
$0 == ""  || $1 ~ /^\sp[a-z0-9.]*/ {
                    if (list == 1 || lists == 1) print "<sep>"
                    if (prog == 1 || picfig == 1) print }

#
# Save troff commands within a picfig (avoid duplicating .PS)
#
$1 ~ /^\[a-zA-Z0-9]+/ && picfig == 1 && $1 != ".PS" { print }
```

```
# print all lines which have not been detected specially, as long
# as one slide head has been read. Ignore data within the
# F+,F- brackets
#
$1 !~ /^\[a-zA-Z0-9]+/ && slide == 1 && $0 != "" && psfig == 0 { print }

END  { print "</pslides>" }

-----------------------------------------------------------------

#
# removes blank <text> sections.
#
$0 != "<text></text>" {print}

-----------------------------------------------------------------

#
#
#  scans file to
# correct the problem of the <hp> tags and significant
# newlines. Correct tagging requires a knowledge of
# the preceeding line state.
#
# Variables:    start      indicates whether a line has been
#                          read. It is known that first line of
#                          file will start with <pslides ...
#
#               endhl      set if </> read.
#
#               endmk      set if line ends with a tag.
#
BEGIN  { endhl = 0
               endmk = 0
               start = 0 }

#
# for highlight start, either delete previous newline character
# (if previous line ended in a tag) or replace by a space.
# Clear 'end highlight'.
#
$0 == "<hp2>" || $0 == "<hp1>"  {
               if (endmk == 1) printf("%s",$0)
                    else printf(" %s",$0)
               endhl = 0
               endmk = 1 }
```

```
#
# for 'end highlight' (used to end both highlights), flag it and
# suppress previous newline if last line ended in a tag.
#
$0 == "</>"  {  if (endmk == 1) printf("%s",$0)
                    else printf("\n%s",$0)
                endhl = 1
                endmk = 1 }

#
# for all other lines starting with a tag, suppress previous
# newline if either last line was an 'end highlight' or else this
# is the first line of the file.
# Set flag if line ends with a tag as well (could be same tag).
# Clear 'end highlight'.
#
$0 ~ /^</  && $0 != "</>" && $0 != "<hp2>"  && $0 != "<hp1>"  {
                if (endhl == 1) printf("%s",$0)
                    else if (start == 1) printf("\n%s",$0)
                        else  printf("%s",$0)
                start = 1
                if ($0 ~ />$/) endmk = 1
                    else endmk = 0
                endhl = 0 }

#
# For lines not starting with a tag, replace previous newline with
# space if last line was an 'end highlight', otherwise print.
# Set flag if line ends with a tag.
# Clear 'end highlight'.
#
$0 !~ /^</  {  if (endhl == 1) printf(" %s",$0)
                    else printf("\n%s",$0)
                if ($0 ~ />$/) endmk = 1
                    else  endmk = 0
                endhl = 0 }

#
# put in final newline at end of document
#
END  { printf("\n") }
```

```
# sed command which
# picks up the in-line codes for highlights.
# also removes copyright symbols.
#
sed             -e '/\f2/ s?\f2?<hp2>?g' \
                -e '/\f3/ s?\f3?<hp1>?g' \
                -e '/\(co/ s?\(co??g' \
                -e '/\f1/ s?\f1?</>?g'
```

APPENDIX 4

Translation from SGML to troff using MarkIt LINK extension

```
<!LINKTYPE pslidesl PSLIDES #IMPLIED [
<!-- Translation to troff for the PHIGS slideset.
                Assumes the resulting program will use the
                modified ms troff macros -->

<!LINK #INITIAL

                -- nothing required for block, body --

                -- output left or centred list --
DESC
                --APP START
                ASSIGN "%#POSN;" TO posn
                #ENTCOND posn = "LEFT" ADD "
.Ls
" TO #OUTPARSE
                #ENTCOND posn = "CENTRE" ADD "
.Cs" TO #OUTPARSE

                END
                #ENTCOND posn = "LEFT" ADD "
.Le" TO #OUTPARSE
                #ENTCOND posn = "CENTRE" ADD "
.Ce" TO #OUTPARSE --

                -- equation, possibly in file or in line. --

EQN
                --APP START ADD "
.EQ" TO #OUTPARSE
                #ATTCOND FILE != "" ADD "
" TO #OUTPARSE
                #ATTCOND FILE != "" ADD "%#FILE;" TO fname3
#ATTCOND FILE != "" ADD temp3 SYSTEM ENTITY fname3 TO #OUTPARSE

                END ADD "
.EN" TO #OUTPARSE --

                -- enumerated type (the definition) --

ET
--APP START ADD '
.Et "' TO #OUTPARSE
```

```
                END ADD '"' TO #OUTPARSE--

                -- function definition of slide --

FUNCDEF
                --APP START ADD "
.Fs
" TO #OUTPARSE

                END ADD "
.Fe" TO #OUTPARSE --

                -- bulleted item, no end tag --

IT
                --APP START ADD "
.Bu
" TO #OUTPARSE --

                -- sub-bulleted item, no end tag --

ITS
                --APP START ADD "
.Sq
" TO #OUTPARSE  --

                -- highlight - bold --

HP1
                --APP START ADD "\f3" TO #OUTPARSE
                END ADD "\f1" TO #OUTPARSE --

                -- highlight - italic --

HP2
                --APP START ADD "\f2" TO #OUTPARSE
                END ADD "\f1" TO #OUTPARSE --

                -- nothing required for list or elist (covered in desc)  --

                -- sublist   --

LISTS
                --APP START ADD "
.RS" TO  #OUTPARSE
```

```
                END  ADD "
.RE" TO  #OUTPARSE --

                -- PIC, possibly in file or in line.  --

PICFIG
                --APP START ADD "
.PS" TO #OUTPARSE
                #ATTCOND FILE != "" ADD "
" TO #OUTPARSE
                #ATTCOND FILE != "" ADD "%#FILE;" TO fname
#ATTCOND FILE != "" ADD temp SYSTEM ENTITY fname TO #OUTPARSE

                END ADD "
.PE" TO #OUTPARSE --

                -- PROGRAM, possibly in file or in line.  --

PROGRAM
                --APP START ADD "
.Ps" TO #OUTPARSE
                #ATTCOND FILE != "" ADD "
" TO #OUTPARSE
                #ATTCOND FILE != "" ADD "%#FILE;" TO fname1
#ATTCOND FILE != "" ADD temp1 SYSTEM ENTITY fname1 TO #OUTPARSE

                END ADD "
.Pe" TO #OUTPARSE --

                -- PSFIG, in file. Parameters defined on Pslides. --

PSFIG
                --APP START ADD "
.F+
figure " TO #OUTPARSE
                ADD "%#FILE;" TO #OUTPARSE
                ADD "
height " TO #OUTPARSE
                ADD height TO #OUTPARSE
                ADD "
bounds " TO #OUTPARSE
                ADD bounds TO #OUTPARSE

                END ADD "
.F-
.sp -2.0i" TO #OUTPARSE --
```

```
                -- PSLIDES. No end action needed.
                    Generate initial Title slide.
                    Save height and bounds for psfig. --

PSLIDES
                --APP START ADD ".EQ
delim $$
.EN
" TO #OUTPARSE
                ADD '.Sh "' TO #OUTPARSE
                ADD "%#TITLE;" TO #OUTPARSE
                ADD '" ' TO #OUTPARSE
                ADD "%#NUMBER;" TO #OUTPARSE
                ADD '
.Ss ""
.sp 2.0i
.ps 22p
.nr PS 22
.vs 26
.nr VS 26
.tr aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ
.TS
center;
lB .
' TO #OUTPARSE
                ADD "%#TITLE;" TO #OUTPARSE
                ADD "
.TE
.tr aabbccddeeffgghhiijjkkllmmnnooppqqrrssttuuvvwwxxyyzz
.Se" TO #OUTPARSE
                SSIGN "%#HEIGHT;" TO height
                ASSIGN "%#BOUNDS;" TO bounds --

                -- separator is a blank line. No end. --

SEP
                --APP START ADD "
.sp 0.2i" TO #OUTPARSE  --

                -- slide start just saves copyright (done for title) --

SLIDE
                --APP  START ASSIGN "%#COPYR;" TO copyr
                END ADD "
.Se" TO #OUTPARSE --
```

```
                -- table,  possibly in file or in line.  --

TABLE
                --APP START ADD "
.TS" TO #OUTPARSE
                #ATTCOND FILE != "" ADD "
" TO #OUTPARSE
                #ATTCOND FILE != "" ADD "%#FILE;" TO fname2
#ATTCOND FILE != "" ADD temp2 SYSTEM ENTITY fname2 TO #OUTPARSE

                END ADD "
.TE" TO #OUTPARSE --

                -- Newline only for text (at start of desc) --

TEXT
                --APP START ADD '
' TO #OUTPARSE --

                -- Start of Slide (end done in Slide itself) --
                -- Copyright notice set up in slide --

TITLE
                --APP START ADD '
.Ss "' TO #OUTPARSE

                END ADD '" "\(co' TO #OUTPARSE
                ADD copyr TO #OUTPARSE
                ADD '"' TO #OUTPARSE --

>
]>
```