
The HCSS DocBook Guide

Ivan Valtchanov

Davide Rizzo

1. Introduction

This document gathers useful information to write DocBook documents for the Herschel project. The DocBook format is based on XML (eXtended Markup Language). A DocBook document is a plain text file whose contents are enclosed in *tags* (just like HTML for example). The tags are defined in a DocBook *Document Type Definition* (DTD) maintained by the DocBook Technical Committee of the OASIS organisation (Organization for the Advancement of Structured Information Standards). The DTD defines the vocabulary of content elements that an author can use and how they relate to each other. For example, a book element can contain a `title` element, any number of `para` elements for paragraphs, and any number of `chapter` elements. Using the DTD and XML syntax, authors mark up their text content with tag names enclosed in angle brackets like `<chapter>`. As we said the markup is similar to HTML, but with more tags and tighter rules.

Important

DocBook was originally created by a consortium of software companies as a standard for computer documentation. This imposes some limitations when used for example for scientific articles with formulae.

The DocBook/XML (for short DocBook) *source* file is not very convenient to read (as is the case for LaTeX or HTML source files), so it is necessary to perform post-processing (or *publishing* in DocBook terminology) in order to produce human readable documents in HTML (e.g. web pages) or PDF formats.

The publishing of a DocBook/XML document is done with an XSLT processor. Depending on the style, the output can be an HTML file or an XSL-FO file (Formatting Object). The HTML is ready to be published in a web page while the FO file can be further processed with FOP (Formatting Object Processor) to produce a PDF/PS/SGML/RTF/JavaDoc/OpenOffice/Microsoft Office etc. output document.

The XSLT processor used in the HCSS Documentation Framework is called **Xalan**, is developed by the Apache consortium and comes in a package called **FOP** which contains also the processor to convert the DocBook to XSL-FO and then to PDF (or PostScript/RTF/SGML etc). **FOP** is written in Java so it is platform independent.

2. DocBook Fundamentals

2.1. Document declaration

Start your DocBook source file with the following document declaration:

```
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">
```

This is the latest stable version of DocBook DTD (v4.5). If you don't have access to Internet then use a local version of `docbookx.dtd`, like this:

```
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
```

```
"../../../../docbook/dtd/docbookx.dtd">
```

Caution

Make sure to put the correct relative path to your local copy of `docbookx.dtd`. Also, beware of "/" and "\" when you work under Windows and Unix.

2.2. Entity declarations

These are shortcuts for frequently used names, phrases or input files. The **ENTITY** declaration comes inside the `<!DOCTYPE` tag as shown in the following example:

```
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
"../../../../docbook/dtd/docbookx.dtd"[
<!ENTITY microns '<phrase role="symbol">&#956;</phrase>m'>
<!ENTITY lambda '<phrase role="symbol">&#955;</phrase>'>
<!ENTITY He3 '<superscript>3</superscript>He'>
<!ENTITY HSPOT 'HSpot'>
]>
```

For example, use `&He3`; where you want `"3He"` to appear.

2.3. Typographic conventions

When writing in DocBook you should only worry about the *structure* of our document, i.e. its subdivision in chapters, sections, paragraphs and so on. You should not worry about a certain word appearing in bold or italic typeface, or a certain title being centred or left justified: this is the job of the stylesheet associated to the document, something the author does not have to worry about. What the author must worry about is to use the right tag in the right situation. Here are a couple of examples:

- Concept or words to be emphasized should be inside the `<emphasis> ... </emphasis>` tag. If that is not enough for you, use `<emphasis role="strong"> ... </emphasis>`.
- Words or concepts appearing for the first time have their own tag, `<firstterm> ... </firstterm>`.

Note that, depending on the stylesheet being used, words between the `<emphasis> ... </emphasis>`, `<emphasis role="strong"> ... </emphasis>` and `<firstterm> ... </firstterm>` tags could appear exactly the same (e.g. all in italics). This should not be a reason to use unrelated tags and other dirty hacks to make the document "look good". Just stick to the right tag, and if you do not like the final result ask for the stylesheet to be modified.

2.4. Sections

You may want to use `<sect1>`, `<sect2>` etc. for sections instead of nested `<section>` tags. This makes easier to locate inside which chapter-section-subsection hierarchy you are when you edit the DocBook source file.

2.5. Titles

The chapters, sections, tables, figures, examples, procedures etc. expressed in DocBook can have `<title>` tag for the title. If the title is too long for the table of contents then you may use the `<titleabbrev>` tag to put a shorter name, which will appear in the table of contents. The `<titleabbrev>` tag must come after `<title>`.

An empty `<title></title>` will produce the corresponding component label and number: i.e.

for `<figure>` an empty `<title></title>` will produce e.g. **Figure 2.3** without any additional text.

2.6. Code and code examples

Code examples must appear within the `<code> ... </code>` tag. When reporting interactive sessions in JIDE, the `IA>>` prompt should be omitted, unless it is relevant to the discussion, in which case it should be enclosed between `<prompt> ... </prompt>` tags. The output given by HCSS should begin with a # (comment character). Actual comments to the code should begin with ##. Interactive sessions should be enclosed in `<screen> ... </screen>` tags.

When reporting complete examples or snippets of non-interactive code, comments should begin with a single # and be placed before the line(s) they refer to. Examples should be enclosed in `<example id="my-example-id"> ... </example>`. The `id` property will be used for references to the examples elsewhere in the text. You should not hardcode the example number, but write something like "as seen in `<xref linkend="my-example-id" />...`". In this way, even if you move the examples around, the numbers in the references will always be correct. This is also true for all the other elements that accept the `id` property (e.g. chapters, sections, figures...). It is good practice to add an `id` whenever you can, even if you do not plan to add references to that object elsewhere.

For entering key names like **Enter** or **Ctrl**, use the `<keycap> ... </keycap>` tag.

Another tricky business is how to treat items like commands, command arguments and user input. Some arguments are optionals, some are not. Sometimes users are meant to copy what is written in the documentation, sometimes they are meant to replace it with something else. There are several tags coming to the rescue, a few of which are explained below.

- When indicating command arguments, use the `<options> ... </options>` tag.
- For optional content, use the `<optional> ... </optional>` tag. Command arguments that are not mandatory should be enclosed in both this *and* the previous tags.
- For content that may or must be replaced by the user, use `<replaceable> ... </replaceable>`.
- For a command to start an executable file (like **firefox**) use the `<command> ... </command>` tag.
- Input to be inserted by the user has the `<userinput> ... </userinput>` tag.
- In addition to a tag for user's input, we have the `<computeroutput> ... </computeroutput>` tag for what comes back from the computer.
- Catch-all example: `ls [-1] directory_name`. And here how it was done:

```
<userinput> <command> ls </command> <optional> <option> -l </option>
</optional>      <replaceable>      directory_name      </replaceable>
</userinput>
```

2.7. Dealing with mathematical formulas

Support for writing math formulas is currently quite basic. For anything that is not covered by the following guidelines, the suggestion is to create the formula with an external tool (such as LaTeX or the math editor in your favourite word processor), save it as an image and then include it in your DocBook document.

Simple one-line formulas without exotic symbols can be rendered with the `<superscript> ... </superscript>` and `<subscript> ... </subscript>` tags. If you then need to include Greek characters, use the following construct:

```
<phrase role="symbol">&Sigma;</phrase>.
```

Capital initial means capital Greek letter: with "Σ" you will get Σ , while with "σ" you will get σ .

For example, look at the following formula, taken straight from the User's Manual:

$$f(x,y;p) = \sum_k p_k x^k y^{(d-k)}$$

The corresponding source code looks like this:

```
<emphasis> f(x,y;p) = </emphasis> <phrase role="symbol"> &Sigma;</phrase>
</phrase> <emphasis> p <subscript> k </subscript> x <superscript> k
</superscript> y <superscript> (d-k) </superscript> </emphasis>
```

Note that the Greek character is outside the `<emphasis> ... </emphasis>` tag. This is a precaution measure, as generally what is inside this tag is rendered as italic typeface and you may not have the right Symbol font installed (but feel free to experiment). Actually the use of the `<emphasis> ... </emphasis>` tag is a violation of the guidelines listed in this very document, since we are trying to influence how the formula will *look* on the page rather than just dealing with its *structure*. However, this is to be considered as a temporary workaround until better ways to include formulas are introduced.

Another way is to use numeric codes as shown in the following example. Note also how to modify the `<role>` tag to get sub- and superscripts.

```
<phrase role="symbol">&#955;</phrase> <!-- Lower case lambda -->
<phrase role="sup_symbol">&#955;</phrase> <!-- Lower case lambda, superscript-->
<phrase role="sub_symbol">&#955;</phrase> <!-- Lower case lambda, subscript-->
```

A reference for the symbols' number codes is given here:

<http://www.w3.org/TR/html401/sgml/entities.html#sym>.

2.8. Figures

Use the following template for figures.

```
<figure id="fig.one">
  <title>Long title</title>
  <titleabbrev>Short title</titleabbrev> <!-- this should come after <title>! -->
  <mediaobject>
    <imageobject>
      <imagedata format="PNG" align="center"
        fileref="myimage.png" scale="95" width="12cm"/>
    </imageobject>
    <textobject><phrase>My nice figure one.</phrase></textobject>
  </mediaobject>
</figure>
```

Note

In the current version of FOP (0.20.5) you have to use both `scale="50"` (for 50% of the actual figure size in the HTML output) and `width="10cm"` for the size of the figure in the PDF output. And ignore the warning messages from the processor.

The format can also be "JPG" for JPEG figures. The `<textobject>` tag is useful when the figure is not displayed or one is viewing the page inside a text-only web browser (such as lynx).

2.8.1. How to put many figures on a grid

The short answer is: using `<informaltable>` to put them in a suitable table. Here is an example:

```

<figure id="myFig">
<title>The caption of my composite figure</title>
<titleabbrev>My short title (to go in the ToC)</titleabbrev>
<informaltable frame="none">
<tgroup cols="2" align="center">
<colspec colname="c1"/>
<colspec colname="c2"/>
<tbody>
<row>
<entry align="center">
<mediaobject>
<imageobject>
<imagedata format="PNG" align="center"
fileref="myimage1.png" scale="50" width="8cm"/>
</imageobject>
<textobject><phrase>My nice figure one.</phrase></textobject>
</mediaobject>
</entry>
<entry align="center">
<mediaobject>
<imageobject>
<imagedata format="PNG" align="center"
fileref="myimage2.png" scale="50" width="8cm"/>
</imageobject>
<textobject><phrase>My nice figure two.</phrase></textobject>
</mediaobject>
</entry>
</row>
</tbody>
</tgroup>
</informaltable>
</figure>

```

The outer `<figure>` tag is just to have the **Figure 2.22** bit and the caption (title), as well as the id for referencing the composite figure. You may need to play a bit with the scaling (scale and width) of the images to get the best results in both the HTML and the PDF outputs.

2.9. Tables

Very nice examples of tables in DocBook are given here. A useful general template is given below:

```

<table id="tab.one" frame="all">
<title>Long title.</title>
<titleabbrev>Short title.</titleabbrev>
<tgroup cols="6" align="left">
<colspec colname="c1" colwidth="1*"/>
<colspec colname="c2" colwidth="2*"/>
<colspec colname="c3" colwidth="1*"/>
<colspec colname="c4" colwidth="1*"/>
<colspec colname="c5" colwidth="2cm"/>
<colspec colname="c6" colwidth="2.5cm"/>
<thead>
<!-- the header of the table, in bold face,
column headings -->
<row>
<entry></entry>
<entry></entry>
<entry></entry>
<entry></entry>
<entry></entry>
<entry></entry>
</row>
</thead>
<tbody>
<!-- the table body starts here -->
<row>
<entry></entry>
<entry></entry>
<entry></entry>
<entry></entry>
<entry></entry>
<entry></entry>
</row>
</tbody>
</tgroup>
</table>

```

It would take too long to explain all the tags for tables, so have a look in the reference given above for advanced usage. One useful bit is how to control the output width of the table. Here is an example:

```
<table id="tab.one" frame="all">
  <title>Long title.</title>
  <titleabbrev>Short title.</titleabbrev>
  <?dbhtml table-width="75%" ?>
  <?dbfo table-width="12cm" ?>
  <tgroup cols="6" align="left">
```

Note the two `<?dbhtml ?>` and `<?dbfo ?>` processing tags: they tell the XSLT processor to make the table width 75% of the actual size (browser window size in HTML) or 12cm in FO->PDF.

Important

The two processing tags must be outside the `<tgroup>` tags and inside `<table>` tags.

2.9.1. How to centre a table

When your table size is smaller than the page size then it might be better to centre the table. This is tricky and there is no simple way, for instance the `align="centre"` attribute does not work (don't ask us why). So, here is a possible solution and the only suitable solution we have found. It uses an `<informaltable>` with one cell in which the original table is centered:

```
<informaltable frame="none">
  <?dbhtml table-width="100%" ?>
  <?dbfo table-width="15cm" ?>
  <tgroup cols="1" align="center">
    <colspec colname="c1"/>
    <tbody>
      <row><entry>
        <table id="myTab" frame="all">
          <!-- etc my original table pasted here -->
        </table>
      </entry></row>
    </tbody>
  </tgroup>
</informaltable>
```

2.10. Cross-referencing

To reference tables, figures, chapters, sections etc. from the document use the following template:

```
<xref linkend="fig.one"/>
<xref linkend="tab.one" xrefstyle="select: label"/>
<xref linkend="sec.one" xrefstyle="select: labelnumber"/>
```

Note

In the first example, the text will have the following style: *Figure 1, +Title*. In the second example it will be only *Table 1*, and in the last case only the number "2.1" (for example) will appear in the text.

2.11. Links to URL

Use the following to have links to web pages somewhere on the web:

```
<ulink url="http://herschel.esac.esa.int">underlined text</ulink>
```

2.12. Footnotes

Footnotes can be used in the following way:

```
<footnote id="foot1"><para>Some footnote text</para></footnote>
```

If you need to refer to an already existing footnote (i.e. use the same superscript character which points to the footnote) then you can use `<footnoteref>` in the following way:

```
<footnoteref linkend="foot1"/>
```

2.13. New lines and page breaks

To insert an empty line just use an empty paragraph:

```
<para></para>
```

Page breaks only work for PDF output:

```
<?hard-pagebreak?>
```

Important

Make sure that the above declaration is outside any `<para></para>` block. Note also that the Documentation Framework currently ignores hard page breaks.

2.14. Bibliography

If you want to have the bibliography in more or less astronomical notation then use as example the following template:

```
<bibliography id="sec.biblio">
  <title>References</title>
  <biblioentry id="bib.1">
    <abbrev id="bib.1.abbrev">Astronomer et al. (2007)</abbrev>
    <author><surname>Astronomer, A.N., et al. </surname></author>
    <title>2007, DocBook in the sky, ApJ 202, 34.</title>
  </biblioentry>
  ...
</bibliography>
```

You may place the `<bibliography>` section anywhere and you can have more than one (for example at the end of each chapter or section). In the text you have to refer to the *biblioentry* like is shown in this example:

```
<para>See the examples in <xref linkend="bib.1" endterm="bib.1.abbrev"/>.</para>
```

and the result will look like this in the text:

```
See the examples given in Astronomer et al. (2007).
```

2.15. Admonitions

Use the following templates:

```
<note><para>...</para></note>
<warning><para>...</para></warning>
```

```
<caution><para>...</para></caution>
<tip><para>...</para></tip>
<info><para>...</para></info>
<important><para>...</para></important>
```

2.16. <book> template

Below we give a DocBook book template that you can copy and paste and use as a starting point for a DocBook document.

```
<?xml version="1.0"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
"../../../../docbook/dtd/docbookx.dtd"[
  <!ENTITY microns '<phrase role="symbol">&#956;</phrase>m'>
  <!ENTITY lambda '<phrase role="symbol">&#955;</phrase>'>
  <!ENTITY He3 '<superscript>3</superscript>He'>
  <!ENTITY HSPOT 'HSpot'>
]>

<book lang="en-GB">

  <bookinfo>
    <mediaobject>
      <imageobject>
        <imagedata fileref="../images/herschel-doc-logo.png" format="PNG" />
      </imageobject>
    </mediaobject>
    <title>SPIRE Observers' Manual</title>
    <subtitle>Announcement of Opportunity for Key Programmes</subtitle>
    <authorgroup>
      <author> <firstname>Ivan</firstname><surname>Valtchanov</surname></author>
    </authorgroup>
    <edition>version 1.0.0</edition>
    <pubdate> version 1.0.0, 19-June-2007 </pubdate>
    <issuenum>HERSCHEL-HSC-DOC-xxx, version 1.0.0</issuenum>
    <date>15-June-2007</date>
  </bookinfo>

  <chapter id="ch-intro">
    <title>Introduction</title>
    <para>Some text here.</para>
    <sect1 id="sec-one">
      <title>Section One</title>
      <para>Some text here too.</para>
      <sect2 id="sub-one1">
        <title>SubSection One</title>
        <para>Some text.</para>
      <sect3>
        <title>SubSubSection One</title>
        <para>Some text here too.</para>
      </sect3>
    </sect2>
  </sect1>
</chapter>
...
</book>
```

2.17. DocBook Validity Issues

These are our recommendation for creating valid DocBook documents. In the current version of DocBook DTD (4.5) and XSLT-FO processor you may have DocBook document which is not valid (from XML point of view) but still produce acceptable output. This however should be avoided as any transition to the newer DocBook version 5 will be very painful and will require a lot of manual changes.

- Do not use strange characters in the `id=""` attribute. For example, don't use ":", but ".", "-", "_" are all right.
- All admonition text should be inside `<para></para>` as shown in Section 2.15.
- Figures, tables, equations, examples, procedures etc. should also be inside `<para></para>`. In fact, any text elements must be in paragraph mode (i.e. inside `<para></para>` tags).

2.18. Including external documents

This section shows how to apply the *Xinclude* technique to embed DocBook documents into other DocBook documents. This includes, for example, inserting instrument-specific documentation into DP help documents like the User's Manual.

2.18.1. The main document

Let us start from the document that will contain the hooks to the external content. There isn't really much to say about it: basically we are talking of a normal DocBook document (article, book, whatever you need) with *hooks* here and there pointing to other documents, like this:

```
<xi:include xmlns:xi="http://www.w3.org/2001/XInclude" href="otherdoc.xml">
  <xi:fallback>
  </xi:fallback>
</xi:include>
```

The first line looks for the `otherdoc.xml` file to be included in the main document. Here the secondary file is in the same directory as the main one; a relative path can be added if this is not the case.

If the file is not found, whatever is inside the `<xi:fallback>` tag is written. In this case we have nothing there, i.e. the missing document is ignored. Instead we could put any valid DocBook construct, e.g. to write *Document not found* or something to that effect.

2.18.2. The external documents

Now it is time to assemble the external documents that will be included into the main file.

Again, these are normal DocBook documents. For example, let's suppose we are writing an article and each section resides in an external file. Any of these files will look like this:

```
<!DOCTYPE sect1 PUBLIC "-//OASIS//DTD DocBook XML V4.5//EN"
"http://www.oasis-open.org/docbook/xml/4.5/docbookx.dtd">
<sect1 id="aSection"><title>A Section</title>
<para>Blah Blah Woof Woof.</para>
</sect1>
```

Of course a "hooked" document can include hooks to other documents, and so on and so forth.

Note that these external documents are *complete* DocBook documents, which can be checked and validated on their own. This was *not* true with the old system used, for instance, with the User's Manual. Back then each chapter file would be defined as an entity in the main file `um.xml`, like this:

```
<!ENTITY chap1 SYSTEM "chapter-01.xml">
```

The chapter file would later be included by calling the entity, like this:

```
&chap1;
```

Two disadvantages of the old approach: the chapter files weren't complete DocBook documents, and there was no graceful fallback in case a file was not found.

2.19. Other useful documents

Please refer to the *DocBook Authors Guide*, available here:

<ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/ia/editorial-board/dbkag.pdf>.

For a more comprehensive list, you may want to browse *DocBook: The Definitive Guide* here:

<http://www.docbook.org/tdg/en/html/docbook.html>.

The amount of available tags is overwhelming, and you might feel guilty for not enclosing in tags every other word you write. Just do not overdo it. You are not supposed to learn what every tag does. A much better way to get started is to have a look at the DocBook source of an existing document, in order to get used to the most common tags.

3. Generating PDF, HTML and JavaHelp

3.1. Using the Documentation Framework

The Documentation Framework is the infrastructure used to generate all the DocBook-based documentation included in each HCSS build. Below are the steps needed to get it up and running.

First of all, download and install the latest development build of HCSS you can put your hands on. Then download the current version of the HCSS Reference Platform package, available here: <ftp://ftp.rssd.esa.int/pub/HERSCHEL/csdt/refPlatformDownloads/>.

You need to make sure that your CLASSPATH includes a bunch of JAR files used by the Doc Framework. One of them is part of Java, the others are contained in the Reference Platform:

- `tools.jar`, in the `lib` directory of your Java Development Kit (JDK);
- `fop.jar`, in the `fop-0.20.5/build/` subdirectory of the Reference Platform;
- `xalan.jar`, in the `xalan-j_2_7_0/` subdirectory of the Reference Platform;
- `avalon-framework-cvs-20020806.jar` and `batik.jar`, in the `fop-0.20.5/lib/` subdirectory of the Reference Platform.

Next, make sure that your `user.props` file contains a line like this:

```
var.hcss.workdir = ${user.home}/tmp/hcss
```

Here `${user.home}/tmp/hcss` represents a temporary directory for which you have write permission. Put a suitable path for your system.

Now we are ready. Go to the directory where your XML files are and type the following:

```
make_static manual_id manual_xml_file [manual_description]
```

For example (omitting the optional description):

```
make_static myDocument myDocument.xml
```

When you launch the command a GUI will pop up, and hopefully after a while you will have PDF, HTML and JavaHelp versions of your document. If the original XML files reside in a `xml/` directory, then the finished documents will be in `../pdf`, `../html` and `../javahelp`, otherwise these three directories will be at the same level as the source files.

For more information please see Chapter 12 of the Developer's Manual, *Writing Documentation*.

3.2. Using the standalone Documentation Framework

This package contains the same software described in the previous section, packaged and preconfigured so that:

- all the dependencies are taken care of;
- there is no need of a working HCSS build.

You can download it following this link:

<http://soleil.i4ds.ch/~soldati/docfw/docfw-latest.zip>

The latest version at the time of writing is 0.3 and weighs about 18 MB.

To install the package:

- Extract it to any location on your system (on Windows it should even work if your path contains spaces).
- Open a cmd console (e.g. on Windows: Start - Run ... - "cmd").
- Change dir (cd) to the root dir of the extracted ZIP file (e.g. cd foo/docfw-0.3)

To use the package follow these steps:

- Copy your manual to docfw-x.y/doc/<manual>/xml (choose any meaningful identifier for <manual>)
- As an example and for testing an old version of the Herschel FAQ is included in the downloaded ZIP.
- Execute the doc framework:

```
Syntax: make_static "manual_id" "manual_file"
"manual_id"      Id of the manual, used as the main name for output files.
"manual_file"    Absolute or relative path of the docbook xml file.

e.g  bin\make_static.bat faq doc\manual\xml\faq.xml (Windows)
     bin/make_static faq doc/manual/xml/faq.xml (Linux/Mac)
```

The directory structure of the standalone docframework looks like this:

```
docfw
|--bin
|   make_static.[bat|sh] # Shell scripts to create a manual.
|--doc      # Source and output directory.
|  |--hcss
|  |  |--manual # Root folder for manual to be generated
|  |  |--xml   # Docbook files of manual
|  |  |--html  # Generated html output (dir will be created by doc fw)
|  |  |--pdf   # Generated pdf output (dir will be created by doc fw)
|  |  |--javahelp # Generated javahelp output (dir will be created by doc fw)
|--lib
|   hcss.jar
|   xerces.jar
|   xalan.jar
|   ...
|--temp
docbook # Content of temp dir will be created by doc framework.
```

The current version has these known limitations:

- The standalone doc framework only works for static manuals, i.e. not the URM, DRM
- It is not possible to specify the output dir
- It is not possible to specify the output format
- Icons may not be displayed correctly.

Note

You can safely ignore a message like the following:

```
12-Jan-08 02:02:24.906 XsltUtil$2: SAX Parser warning:
Include operation failed, reverting to fallback.
Resource error reading file as text (href='.././././././
../BUILDNUM'). Reason: C:\data\herschel\src\standalone_
docfw\BUILDNUM (The system cannot find the file
specified) at 'SystemId unknown' [28:126]
```

The maintainer of the standalone Documentation Framework is Marco Soldati. Please contact him at marco.soldati at fhnw.ch if you think you have found a bug or if you want to request a new feature.

3.3. Using the dbpack Package

Warning

This package has now been deprecated. The recommended tool for generating HTML, PDF and JavaHelp documents from DocBook sources is the Documentation Framework, either in its standalone form (Section 3.2) or as part of an HCSS build (Section 3.1).

3.3.1. Package installation and contents

Note

FOP version 0.20.5 is used in the *dbpack* but the Java classes were taken from the updated version of FOP 0.94. The updated fop 0.94 processor is claimed by apache.org to be better but the output is not of good quality.

To install *dbpack* simply unzip *dbpack_version.zip* somewhere. It will create the following directory structure:

```
dbpack/
VERSION
bin/
docbook/
custom/           # tree for customisation
xsl/              # customised stylesheets for HTML and PDF.
admonitions/     # admonitions images
css/              # custom style sheet for HTML
dtd/              # DocBook tags definitions
xsl/              # the standard style sheet
common/
fo/               # style for PDF output
html/            # style for HTML output
lib/
lib/
fop-0.20.5/       # XMLTS-FO processor
fop-0.20.5/lib/   # XMLTS-FO processor jar files (from FOP 0.94)
projects/
admonitions/     # the admonition figures are kept here.
css/              # the custom stylesheet CSS for HTML is here.
example/
xml/              # DocBook XML files for your project called example
images/          # Images you use in the XML and you link in HTML/PDF.
```

```
html/      # HTML output (many files).
html_one/  # HTML output (one single file).
pdf/      # PDF output.
dbgguide/
xml/      # DocBook XML files for your project called dbgguide
images/    # Images you use in the XML and you link in HTML/PDF.
html/     # HTML output (many files).
html_one/  # HTML output (one single file).
pdf/     # PDF output.
```

The file `VERSION` in the root `dbpack` directory contains the version number of the package.

When you create or copy your project to `dbpack` you have to keep the directory structure under `project/yourProject/` as shown for *dbgguide* (i.e. `html`, `xml`, `pdf` directories are needed for the scripts, see below).

Important

The base XML file must have the same name as the project name. In the above example, for `myProject` I must have my XML DocBook file in `projects/myProject/xml/` and to be called **myProjects.xml**.

Note

The example of the XML and generated HTML and PDF for an early version of this guide is given under *project/dbgude*.

3.3.2. Revisions of the document

With `dbpack` you can conveniently mark document revisions using the *revisionflag* attribute. This attribute can have the following values:

```
<para revisionflag="added"></para>
<para revisionflag="changed"></para>
<para revisionflag="deleted"></para>
```

Some tags will produce warning messages if *revisionflag* is used when they do not accept this attribute (for example `<figure>`). The *revisionflag* tag has no effect on the output document if the standard generation tools are used, i.e. **gen_html**, **gen_html_one**, **gen_pdf**, except for *revisionflag="deleted"* (see the warning below). When **gen_html_changes** is used then those parts with *revisionflag* will be marked in different colour, depending on the value. For example, the "added" paragraphs/figures/tables will have green background, the "changed" will have yellow background, the "deleted" will have red background and the text will be cross-lined.

Warning

When you use *revisionflag="deleted"* this part is not deleted from the PDF output so once you have the final version of the DocBook document you need to manually delete or comment out those parts which have *revisionflag="deleted"* attribute.

Here is one example with a changed paragraph:

```
<para revisionflag="changed">This is my revised paragraph</para>.
```

And below are three paragraphs with the three options. There will be no change in the output when you generate PDF or HTML 1, but you may want to see the difference using **gen_html_changes**.

This is newly added paragraph.

This is one changed paragraph.

This is a deleted paragraph. You can see it in the `gen_html` and `gen_pdf` output! So be careful with this and comment it out for the final document release.

3.3.3. Publishing using *dbpack*

Note

The following examples and scripts are for Unix system (i.e. Linux, MacOS etc.). How to do this in Windows is not covered here.

There are a number of different scripts in `dbpack/bin/` directory. These are just convenience scripts which

1. Declare some necessary Java classes for the processor
2. Run the XSLT processor to produce HTML.

This processor uses the customised XSL `html.xsl` in `dbpack/docbook/custom/xsl/`.

3. Run the two-stage XSLT-FO processor to produce first the formatted objects file (.fo file) and then from the .fo file the PDF output.

This processor uses the customised XSL `fo.xsl` in `dbpack/docbook/custom/xsl/`.

In the following I assume you are in the directory where your base XML DocBook file is, for the example here this means you are in `dbpack/projects/dbguide/xml/`. This is not necessary but it illustrates the usage. in fact you probably can be anywhere in the `dbpack` tree.

- Publish HTML output, in HTML files per chapter and section:

if you are in `dbpack/projects/dbguide/xml/` then

`../../bin/gen_html.sh dbguide`

The base HTML file is `dbguide/html/dbguide.html` and there are many chunks like `ar01s04.html` (meaning "article, section 04). For `<book>` the chunks will have different names: `ch01s04.html` for example for chapter 01, section 4.

Important

The directory `projects/dbguide/html/` must exist.

In order to have better HTML style you need to have the style sheet file `common.css` in `projects/css/`.

- Publish HTML output, one single HTML file:

if you are in `dbpack/projects/dbguide/xml/` then

`../../bin/gen_html_one.sh dbguide`

The HTML file is `dbguide/html_one/dbguide.html` (a single file).

Important

The directory `projects/dbguide/html_one/` must exist.

In order to have better HTML style you need to have the style sheet file `common.css` in `projects/css/`.

- Publish HTML file with revisions marked in colour:

```
~/dbpack/bin/gen_html_changes.sh dbguide
```

This is really useful in conjunction with *revisionflag* attribute (see Section 3.3.2 for more details).

- Create the PDF output:

```
~/dbpack/bin/gen_pdf.sh dbguide
```

The PDF output file is *dbguide/pdf/dbguide.pdf*.

Important

The directory *projects/dbguide/pdf/* must exist.

To have the admonition figures displayed in the PDF file you must have the admonitions (i.e. warning, caution, note etc.) PNG versions (i.e. note.png, warning.png etc.) in *project/admonitions/*.

- What is needed to have HTML correctly displayed?
 1. You can directly point your browser to *projects/dbguide/html/dbguide.html* and as long as you have your images in *../images/*, common.css in *../css/* and admonitions in *../admonitions/* then everything will be all right.
 2. If you need to deliver the HTML file then you need to do the following:
 - a. `cd dbpack/projects/`
 - b. `zip -r dbguide_html.zip admonitions/* css/* dbguide/html/* dbguide/images/*`and that's it. Unzip the *dbguide_html.zip* anywhere and point the browser to *dbguide/html/dbguide.html*

4. Writing the DocBook sources

As DocBook files are plain text, any editor can be used for writing or modifying them. However you might want to use a tool with some more advanced DocBook-specific features. There are several such specialised editors, and listing them all is impossible. One possibility is to use Emacs (<http://www.gnu.org/software/emacs/emacs.html>) together with the nXML plugin at <http://hacks.oreilly.com/pub/h/2044>. Another possible choice is the XXE Editor, which is the subject of the following section.

4.1. Using the XXE Editor

This section is for those who want to use/try one nice what-you-see-is-what-you-get editor called XMLmind (for short **xxe**). You can download the editor from <http://www.xmlmind.com/xmlmind/>. Here are some pros and cons:

Pros:

- Nearly WYSIWYG. You can print the file and the printed file will look as you see it on the screen.
- Based on Java and consequently can be used on all platforms.
- The source DocBook/XML file is human readable and can be further easily edited by any other text editor (jEdit, nedit, emacs, notepad, etc. you name it).

- The XML file can be used **directly** with the Documentation Framework to produce HTML or PDF output.
- No licence fee for the personal version but restricted to non-commercial use.
- You cannot write non-conform DocBook documents: i.e. your XML file will always be valid.
- If you import an already written DocBook file it checks for validity and points out the errors and warning on wrong tag usage.
- A spell checker is included.

Cons:

- No access to conversion tools to produce HTML, PDF etc.
- Requires some practice to get used to use it.
- Cannot use **ENTITY** entries. This has some implications: cannot include files or make shortcuts/macros for frequently used entities.
- It is very easy to put special characters and they work directly for the HTML output, but you need to manually change them to the example reported in Section 2.7 in order to make them work for the PDF file.
- The cross-linking style (i.e. xrefstyle tags) has no effect in the WYSIWYG output. You only see the id="" bit which links to the linkend.
- Paste does not work in some dialogue boxes (Java thing?).
- There is no "Spell as you type" in the personal edition.

4.1.1. Some practical tips for working with xxe

You may want to follow the tutorial here. Below some tips from me:

1. Creating a new document:

This is rather simple: *File -> New...* from the menu and then choose the Document template from the pop-up window: *DocBook -> article* for example.

2. Importing already existing DocBook file:

This is more tricky, especially if you have ENTITY declarations in the document. These will be lost and any input documents will be ignored. This is for the personal edition.

Very likely when you try to open your document with **xxe** it will give that the document is not valid but still will load it (hopefully). You can see and locate the errors quite easily (there is a tab on the right) but sometimes it will be necessary or much easier to open the concatenated document in a text editor and correct the errors before trying again with **xxe**.

5. Instructions for UM Editors

This section was written for editors of the **HCSS User's Manual**. You can safely skip it if you are not a UM editor, although some of the tips may also be applied to other documents.

So, what are the mandatory steps UM editors have to follow (besides doing what the Editorial Board told them to, that is)? What follows is a basic checklist.

- Have a look at the *What's New* page. Go to this address

<http://www.rssd.esa.int/SD-general/Projects/Herschel/hscdt/docsDp.shtml>

and click on the *What's new?* link. Any changes not yet documented in the UM should be added.

- Now it is time to see if the examples still work. By *example* we do not mean every snippet of code, but just the longer, more comprehensive scripts enclosed in `<example> ... </example>` tags. Run them with the latest developer's build and see if they do what they are supposed to. If that is not the case, it could be due to an intentional change to the software, in which case the script will have to be modified accordingly. It could also be due to a bug, in which case an SPR should be raised (if that has not happened yet) and the script should be left unchanged. Since the boundary between bugs and features has always been somewhat fuzzy, do not be afraid to ask for help.

Checking *every* snippet of code can be very time consuming. A better way is to look at the *What's New* page cited before and see what has changed since the DP version on which the UM was last based. Then you will just have to check these relevant sections.

- Now that you have added and modified what you were supposed to, it is time to record your changes. Open `preface.xml` and go to the *Versioning* section. Change the User Release number this version of the UM will be associated with. Then move on to the following section, *What's New and Previous Versions of DP User's Manual*. Update the build number used to test the examples and write down the changes you have made. You can use what's already there as a template.
- The final step is to generate the PDF and HTML versions of the UM, following the instructions of the package you are using (see Section 3.3 for `dbpack` or Section 3.1 for the Documentation Framework). You may probably want to perform this step fairly often during your work, to catch errors early. Should you get an endless avalanche of errors, do not panic and scroll back up to the first one. Fix it and try again. Many errors are just a consequence of previous ones, so the problem may not be as big as it appears.
- That's it: once you've got your PDF file, send it to the Editorial Board, at `<dp-eboard@ster.kuleuven.be>` and wait to be congratulated on your great job. When no other modifications are requested, the XML source files should be committed to CVS before the documentation freeze date. The CVS directory is:

```
develop/main/herschel/ia/document/doc/ia/document/um/xml
```

And now a couple of recommendations for a spotless editing.

- Remember that spelling rules for British English should be used in the UM, which means that you will have to write *colour* instead of *color*, *catalogue* instead of *catalog* and so on. However that is only true for the text. When writing code, American English has to be used, to conform with the Java standard. This means for example that to change *colours* in your plots you have to use the *Color* class.
- You may have to produce or update images of plots. In this case it is important that the plots you produce are the same as those obtained from a fresh installation of the DP software. In other words, there should be no custom plot settings in your `PlotXY.props` file or in any other configuration file.