The eCromedos User Manual (v1.0.1)

Tobias Koch

April 24, 2006

`http://www.ecromedos.net`

Contents

Contents

1 Introduction

1.1 What is eCromedos?

eCromedos is an integrated solution for XML-based publishing in print and web. It is specifically targeted at producing technical documentation in the field of computer science.

Documents are written in an XML-based markup language and translated to different formats with XSL transformations. At this time, eCromedos supports the target formats XHTML and LaTeX. Where LaTeX output can be further processed into high-quality printable formats by use of the TeX typesetting system (`http://www.ctan.org`).

The eCromedos Markup Language (ECML) is modelled closely after HTML. If you are already familiar with HTML, you will find adapting to ECML particularly easy.

eCromedos comes with a modified XSL processor that is specifically tailored to processing ECML and provides additional functionality that cannot be implemented efficiently with pure XSL transformations. It's most salient feature is the capability to apply syntax-highlighting to program listings.

1.2 License

eCromedos is free (as in speech) software, released under the GNU General Public License, **version 2** ([1]).

1.3 About this Document

This document describes the installation and usage of eCromedos and in particular the scope of the eCromedos Markup Language.

After studying this manual, you will be able to write feature-rich documents and transform them to HTML or LaTeX with the eCromedos Document Processor.

1.4 Conventions

The following conventions are used in this document:

- The names of ECML language elements will be set in typewriter letters.

- Path names and internet addresses will be set in typewriter letters.

- Program listings will be set in typewriter letters on a gray background.

- The dollar sign represents a command line prompt in listings.

1.5 Getting Involved

If you are dissatisfied with any particular part of the eCromedos system or feel, that something needs to be improved, you are welcome to propose changes and additions.

If you are a programmer and want to submit a patch or bugfix, please send it to the mailing list `ecromedos-development@lists.ecromedos.net` for review.

2 Installation

2.1 Obtaining the Software

The latest version of eCromedos can be obtained from the project's website at `http://www
.ecromedos.net`.

2.2 Installing Required External Software

In order for eCromedos to function properly, you first have to make sure that the following
packages are installed on your system:

Package	Description
python	The python interpreter
libxml	XML library of the GNOME project
libxml-python	Python bindings for libxml
libxslt	XSL library based on libxml
libxslt-python	Python bindings for libxslt
imagemagick	Library and tools for image manipulation
tetex	Common TeX distribution for Linux

These packages are included in all modern Linux distributions and chances are, they are al-
ready installed on your system.

2.2.1 On an OpenSuSE Installation

On SuSE the names of the required packages correspond exactly to the ones given in the table
above. In order to install them, start YaST and select the software management task.

2.2.2 On a Debian/GNU System

In order to install the neccessary packages on Debian, execute the following command as root:

```
$ apt-get install latex-ucs latex-ucs-contrib latex-ucs-dev \
    preview-latex-style python libxml2 python-libxml2 libxslt1.1 \
    python-libxslt1 imagemagick tetex-base tetex-bin tetex-extra
```

2.3 Installing eCromedos on Linux

eCromedos can be installed anywhere in the file system. It is recommended, however, that you place it in /opt. In order to do so, drop to a command line shell and become superuser *root* by typing

```
$ su -
```

Then change the working directory to /opt

```
$ cd /opt
```

and unpack the eCromedos tarball. The following example assumes that the package resides in the home directory of system user *tobias*:

```
$ tar -xvzf /home/tobias/ecromeds-x.y.z.tar.gz
```

Now change to /usr/local/bin and create a symbolic link to the main executable:

```
$ cd /usr/local/bin
$ ln -s /opt/ecromedos-x.y.z/bin/ecromedos
```

By placing this link in the system path, you will now be able to call the executable ecromedos directly without having to enter the canonical path.

2.4 Configuration

Having completed the installation procedure described in the previous section, you have to edit the configuration file ecmds.conf, which is located in the subdirectory etc below the installation folder.

If you installed eCromedos in /opt, you probably will not need to change anything. If you chose to install somewhere else, make sure the parameter base_dir contains the full path to the installation folder. In addition, check that each parameter ending in _bin contains the correct path to the corresponding program file.

Values of configuration parameters can be reused in arguments to other parameters by prepending a dollar sign to the name of the parameter to be referenced. For instance, if `base_dir` is set to `/opt/ecromedos-x.y.z/`, then you can set `lib_dir` to `/opt/ecromedos-x.y.z/lib` by simply writing `lib_dir = $base_dir/lib`.

3 Compiling Documents

eCromedos documents are written in eCromedos Markup Language (ECML), which is XML-based markup very similar to HTML. To translate an ECML document to one of the supported output formats, you call the eCromedos Document Processor from the command line.

3.1 Generating HTML

The default output format is HTML. Let us assume, you have written a document in ECML and saved it to the file `test.xml`. To transform this document to HTML, you simply type

```
$ ecromedos test.xml
```

You should preferably call this from a separate directory, to avoid having the output files in the same directory as the source files. Let us say, the source document is located in your home directory in the subfolder `documents`. Simply create another subfolder at the same level, change into it and call `ecromedos` from there. For example:

```
$ cd ..
$ mkdir output
$ cd output
$ ecromedos ../documents/test.xml
```

3.2 Generating LaTeX

To generate LaTeX output from an ECML source file, you have to call `ecromedos` with the `-f` command line switch:

```
$ ecromedos -f latex ../documents/test.xml
```

Depending on your setting of `secsplitdepth` (see below), you will obtain one or more output files with the extension "tex", where the main file will carry the name of the document

class you used. I.e. if the root node of your document is book, the main LaTeX file will be called book.tex.

To compile this file, you have to invoke the LaTeX parser:

```
$ latex book.tex
```

Note *LaTeX usually needs two or three passes to build a complete document. On the first run, LaTeX saves intermediate results in auxiliary files which are reused in consecutive runs to construct, among other things, the table of contents.*

The result will be a file named book.dvi. DVI is a device-independent page description language. You can use the following commands to convert the DVI file to PostScript and PDF:

```
$ dvips -Ppdf book.dvi
$ ps2pdf book.ps
```

This will generate the files book.ps and book.pdf, respectively.

3.3 Output Options

There are various attributes that you may set on the root element of a document, to control the document processor's behavior or influence the layout mechanisms of the formatting engine, i.e. your web browser or the LaTeX parser.

3.3.1 Chunking into Multiple Files

Use the secsplitdepth attribute in order to control, to which depth of sectioning the document will be split into individual files. For example, if you are writing a book and set secsplitdepth equal to one, every chapter will be written to a separate file.

When generating HTML output, it is always a good idea to chunk large documents into smaller parts. This will provide a better user experience, since single parts will load more quickly, especially over low-bandwidth connections, and the user's web browser does not have to keep the entire document in memory, at once.

3.3.2 Specifying the Document Language

The lang attribute is used to select the language for automatic titles, i.e. titles that are generated by the document processor and not by the user, such as "table of contents" or "bibliography". When generating LaTeX output, this also selects the hyphenation patterns for the specified language.

In version 1.0, eCromedos supports three languages, namely `english`, `german` (default) and `spanish`. Language support will be extended in future releases.

3.3.3 Chapter and Section Numbering

The `secnumdepth` attribute controls, to which depth in the section hierarchy sections will be numbered. Setting this to zero will result in the document having no section numbers, at all.

3.3.4 Generating the Table of Contents

During transformation, the document processor will automatically generate a table of contents (TOC) for your document. You do not need to, and in fact you cannot, create the TOC manually.

You can specify, to which depth sections should receive an entry in the TOC, by setting the `tocdepth` attribute appropriately. If you set this to zero, no TOC will be created, *unless* you are generating HTML and are chunking the output into multiple files.

3.3.5 Options for Printed Output

Use the `papersize` attribute to specify your printer's paper format. The default is `a4paper` which is the standard office paper size in Germany. Possible values are all ISO standard paper formats and in general all formats that the LaTeX package *KOMA-Script* ([2]) can deal with.

The `bcor` attribute lets you specify a bind correction. That is the amount in centimeters (cm) or points (pt) by which the text body should be indented to make up for margin space lost when binding the document.

The `div` attribute indirectly controls the dimensions of the text body. Its argument is passed through to the LaTeX package KOMA-Script which controls the layout of printed documents. KOMA-Script tries to automatically determine the optimal dimensions for the text body according to a set of typographic rules. To this end, it divides the page into *div x div* rectangles of equal size, which serve as the basic units for splitting the page into margins and text body. The greater you choose `div`, the bigger the text body will be. Try values between eight and 16.

4 Writing Documents

4.1 Getting Started with XML

The Extensible Markup Language is not a data description language in itself. Rather, it defines a syntax that lets you design your own customized markup languages for arbitrary data models. Take a look at the following example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cocktail alcoholic="yes">
    <name>Pina Colada</name>
    <ingredient>
        <name>rum</name>
        <amount unit="oz">3</amount>
    </ingredient>
    <ingredient>
        <name>coconut milk</name>
        <amount unit="tbsp">3</amount>
    </ingredient>
    <ingredient>
        <name>pineapple</name>
        <amount unit="tbsp">3</amount>
    </ingredient>
    <ingredient>
        <name>ice</name>
        <amount unit="cup">2</amount>
    </ingredient>
</cocktail>
```

What you see is a complete XML document describing the ingredients needed to make the popular Pina Colada cocktail. On the first line, you see the *XML declaration*. It specifies the XML version and the character encoding of the document. In general, you should use a unicode character encoding such as UTF-8 or UTF-16, since any standards conforming XML parser is required to be able to read these.

What follows is the root element cocktail that contains a name element, telling us the name of the cocktail, followed by the various ingredients that you need to make a Pina

Colada.

The textual elements delimiting the beginning and end of an XML element are called "tags". An opening tag has the form `<tag_name>` and the corresponding closing tag is written `</tag_name>`.

XML elements must always be properly nested. In addition, there must be only one root element. Thus, you can picture the logical structure of an XML document as a tree. A single, bare element of this tree is called a "node". Its direct descendants are called "children" and the node from which it originates the "parent".

Nodes may carry additional *attributes*. Our cocktail from above, for instance, has the attribute `alcoholic` which in this case is set appropriately to "yes".

The names of elements and attributes can be chosen arbitrarily to represent a given data model. In our example, we tried to model a beverage but you might just as well define a set of tags to describe the parts of an automotive vehicle.

When working with eCromedos, you will be using XML-based markup to describe the logical structure of standard text documents.

4.2 Available Document Classes

In version 1.0, eCromedos defines three document classes: `report`, `book` and `article`. The difference between these is mainly cosmetic and only visible in printed output. Their definition is formally layed down in a set of Document Type Definitions (DTD), which the document processor uses to verify the correct structure of documents before attempting to transform them.

Take a look at the following listing for an example of a simple *book* in eCromedos Markup-Language:

```
<book lang="english" secsplitdepth="1" secnumdepth="1" tocdepth="1">
    <head>
        <subject>Subject</subject>
        <title>Document Title</title>
        <author>Document Author</author>
        <date>Jan. 16, 1980</date>
        <publisher>Example Publisher</publisher>
    </head>
    <chapter>
        <title>My very First Document</title>
        <p>
            Hi Everybody!
        </p>
```

```
    </chapter>
</book>
```

Documents always have a `head`, regardless of the employed document class. In contrast to HTML, the order of the header elements is *not* arbitrary. The elements `title` and `author` are mandatory and you may specify multiple authors.

As you can see, our book has a chapter with a single paragraph of text. A *paragraph* is the simplest textual element that may occur inside a section.

A report is essentially the same as a book, except that books are layed out double-sided with uneven margins and reports are layed out one-sided with even margins.

Articles differ from books and reports in that the primary sectioning element is `section` instead of `chapter`. Furthermore, sections in an article are printed directly in sequence, whereas in books and reports a new chapter will always start a new page.

4.3 Structuring Documents

In general, you will be using the sectioning elements `chapter`, `section`, `subsection` and `subsubsection` to structure your documents.

Sectioning elements must be given a `title` and they must be nested hierarchically correct, i.e. you cannot have a chapter in a section and you cannot have a subsection in a chapter without first opening a section.

4.3.1 Minisections

Minisections are set with the `minisection` tag. They may appear anywhere in the section hierarchy below the primary sectioning element for the particular document class. The title of a minisection will not be numbered and will not receive an entry in the table of contents.

4.3.2 Prefaces

In books and reports you may use the `preface` element to set an abitrary number of prefaces right after the document head. The title of a preface will not be numbered and will not appear in the table of contents (TOC) when generating printed output. However, it will receive an entry in the TOC when generating HTML.

A preface may contain paragraphs of text, as well as block elements, such as figures and tables. It must not contain any deeper sections. If you feel, you need to section your preface, you should probably make it a chapter.

4.3.3 Appendices

An `appendix` is essentially the same as a chapter. Only the numbering will be different in that the first part of the section counter will be a latin letter instead of an arabic number. Appendices may occur only in document classes book and report. They are to be placed right after the last primary section of a document.

4.3.4 Glossaries

A `glossary` can be placed after the last regular section, which is, depending on the document class, either the last chapter, the last section or the last appendix. A glossary is basically an extra section that must contain nothing but a definition list (see section 5.1).

At this time, eCromedos does not provide functionality for creating and sorting glossaries automatically. This is due to the complexity of implementing this for arbitrary languages. An interface to *xindy, the flexible index generator* (see [3]) is planned for the future.

4.3.5 Bibliographies

Bibliographies are entered with the `biblio` tag and individual entries with `bibitem`. A bibliography may occur only after the glossary, if theres is one, or after the last section in the document, otherwise. Currently, eCromedos does not support bibliographies after individual sections. Here is an example:

```
<biblio number="yes">
    <bibitem label="KOCH06">
        Tobias Koch. eCromedos User Manual.
        <tt>http://www.ecromedos.org</tt>,
        2006.
    </bibitem>
    <bibitem label="WALSH03">
        Norman Walsh, Leonard Muellner.
        DocBook: The Definitive Guide.
        O'Reilly, 2003.
    </bibitem>
</biblio>
```

The `number` attribute is there to control, whether the individual items should be sequentially numbered or if the user-supplied labels should be used.

In the main part of your document, you can use the `cite` tag to cite an entry from the bibliography. For example, in allusion to the listing above, you could write `<cite label="KOCH06"/>`,

which the document processor would replace with "[1]" when numbering is turned on and "[KOCH06]" when numbering is off.

4.4 Formatting Text

From your word processor you may be used to being able to emphasize text by setting it in bold or italic letters or by underlining it. With eCromedos you can achieve this, by enclosing the span of text to be formatted inside the tags b for bold print, i for italic letters or u for underlining. You may also combine these arbitrarily.

Sometimes, you may want to set certain terms or expressions, such as internet addresses, in a font with fixed character width. To this end, there is the tt tag, which prints text in typewriter letters.

Examples of Formatting Text	
`<u>Underlined text</u>`	<u>Underlined text</u>
`<i>Italicized text</i>`	*Italicized text*
`<b>Bold letters</b>`	**Bold letters**
`<b><i>Bold face and italics</i></b>`	***Bold face and italics***
`<tt>Typewriter letters</tt>`	`Typewriter letters`

For the sake of completeness, there are also six elements for modifying the font size. In a serious document you should hardly have any reason to use these, though.

Examples of Modifying the Font Size	
`<xx-small>Text in XXS</xx-small>`	Text in XXS
`<x-small>Text in XS</x-small>`	Text in XS
`<small>Small letters</small>`	Small letters
`<medium>Regular size</medium>`	Regular size
`<large>Large letters</large>`	Large letters
`<x-large>Text in XL</x-large>`	Text in XL
`<xx-large>Text in XXL</xx-large>`	Text in XXL

4.5 Hyphenation

In printed output text is set justified over the entire width of the text body. In order to avoid large gaps between words on single lines, LaTeX applies language specifc patterns to automatically hyphenate and break words on the right margin border.

Unfortunately, LaTeX's hyphenation mechanism is not always able to split words correctly and in rare cases cannot hyphenate certain words, at all.

You can provide hints telling LaTeX in which places a given word may be split, by marking the corresponding spots with the `y` tag. For example, to tell LaTeX that it may hyphenate "bibliography" only in between "biblio" and "graphy" you would write `biblio<y/>graphy` in your markup.

4.6 Line and Page Breaks

In general, you should not worry about where a line breaks or where to start a new page, because it is the job of the formatting engine (i.e. LaTeX or your web browser) to take care of this.

In rare cases, however, you may have to intervene manually. You can use `<br/>` to break the current line and `<pagebreak/>` to start a new page. You should not use multiple `br`s or multiple `pagebreak`s in a row. Of course, a pagebreak is only visible in printed output.

When you need to *prevent* linebreaks in certain places, you can either use the non-breaking space (` `) or protect the specific strip of text with the `nobr` tag. For example, a title or degree should not be separated from the name that follows it. Consequently, you should write `Dr. Pepper` or `<nobr>Dr. Pepper</nobr>` to prevent the formatting engine from possibly breaking the line right before Pepper.

4.7 Cross-References

Sometimes you will want to refer to the contents of a different section in your manuscript, i.e. you may write something like *"[...] you will find out more about this on page XYZ"*. However, at the time of writing your markup, you cannot tell on which page the section you are referring to will actually be printed. The solution is to label the location you wish to reference and let eCromedos do the math.

To label a certain spot in your text, you use the `label` tag. This tag has a single, mandatory attribute, that is the `name` of the label. This must be a unique identifier among all labels in your document. Take a look at the following example:

```
<chapter>
    <title>The Show about Nothing</title>
    <p>
        Seinfeld<label name="seinfeld"> is the best
        sitcom of all times.
    </p>
</chapter>
```

You can now use the elements `ref` to obtain the section number and `pageref` to get the page number like this:

```
<chapter>
    <title>About Myself</title>
    <p>
        I really enjoy watching Seinfeld. You can read more
        about Seinfeld in section <ref name="seinfeld"/> on
        page <pageref name="seinfeld"/>.
    </p>
</chapter>
```

`ref` and `pageref` can also point to any other object with a label, such as a figure or a numbered equation. In that case `ref` will resolve to the corresponding object counter instead of the section counter.

4.8 Marginals and Footnotes

Marginal notes can be placed with the `marginal` tag. And yes, they also work in HTML output. Try this example:

```
<p>
    In this episode<marginal>The Summer of George</marginal>,
    George finally looses his job at the Yankee Stadium but
    gets an extra three months' pay-off.
</p>
```

LaTeX does not allow marginals in table cells. For HTML output this limitation does not exist.

Footnotes are placed in the same fashion by use of the `footnote` tag. They *do* work inside tables without restrictions.

4.9 Quoting

Unless you are setting your text in typewriter letters, you will not be able to enter the correct quotation marks for your language directly with your keyboard. You could use XML character entities to access the glyphs, but that is tedious. Instead you should use the `q` and `qq` tags for single and double quoting, respectively.

4.10 Predefined Entities

eCromedos predefines a number of entity names that may come in handy in certain situations. The following table lists all available names and how they are resolved:

Entity	Description
`&tex;`	Resolves to "TEX"
`&latex;`	Resolves to "LATEX"
` `	The non-breaking space
`&zwsp;`	The zero-width space
`&endash;`	The en-dash (–);
`&emdash;`	The em-dash (—);
`&dots;`	Resolves to "…"

Note *The zero-width space is particularly useful for making long path names or Internet addresses break across lines without introducing hyphens or spaces.*

In order to use these, you have to include a document type declaration at the top of your document, appropriate to the document class you are using. This might look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE report SYSTEM "http://www.ecromedos.net/dtd/1.0/report.dtd">
```

You can also insert these entities indirectly via the `entity` element, in which case you don't need the document type declaration.

5 Advanced Features

5.1 Lists

eCromedos knows three types of lists: ordered lists, unordered lists (a.k.a. bullet lists) and definition lists.

Unordered lists are set with the `ul` tag and ordered lists with `ol`. List items are enclosed by the `li` tag. In addition, these two types of lists may be nested arbitrarily, up to four levels deep. Take a look at the following example:

```
<ol>
    <li>First item</li>
    <li>Second item</li>
    <ul>
        <li>Subitem</li>
        <li>Subitem</li>
    </ul>
    <li>Third item</li>
    <ol>
        <li>Subitem A</li>
        <li>Subitem B</li>
        <li>Subitem C</li>
    </ol>
</ol>
```

Definition lists are set with the `dl` tag. Items in definition lists have two components: a term to be defined and its actual definition. Take a look at this example:

```
<dl>
    <dt>eCromedos</dt>
        <dd>
            A document publishing system that allows generating
            different target formats from one document source.
        </dd>
    <dt>ECML</dt>
        <dd>
```

```
            The eCromedos markup language is an XML based markup
            language for describing the logical structure of
            standard text documents, such as books.
        </dd>
</dl>
```

The most common use of definition lists is the creation of glossaries (see section 4.3.4).

5.2 Figures

Figures are incorporated into a document via the `figure` element. You can give figures a `caption` and a `label`. Note that you may label a figure if and only if you also give it a caption. You may refer to a figure's label with the `ref` and `pageref` elements (see section 4.7). Here is an example:

```
<figure align="center">
    <caption>The Beach</caption>
    <label name="fig:thebeach"/>
    <img src="thebeach.jpg" print-width="100%" screen-width="400px"/>
</figure>
<p>
    Figure <ref name="fig:thebeach"/> shows a beautiful sunset at
    the Galveston beach.
</p>
```

With the `src` element you specify the location of the image on your harddisk. If the image's file format is not suitable for use with a particular output format, the document processor will automatically convert it. For instance, when generating LaTeX output, eCromedos automatically converts raster images to encapsulated postscript.

Note *You should supply images in a resolution high enough for proper representation in all target formats.*

The attributes `print-width` and `screen-width` determine the width of the image in printed output and in HTML output, respectively. For printed output this can be a value in points (pt) or centimeters (cm) or a relative width in percent of the text body's width. The width for HTML output is specified in pixels (px).

The figure's horizontal alignment can be controlled by setting the `align` attribute to `left`, `center` or `right`.

5.3 Tables

Generating good-looking tables with eCromedos is a little bit complicated. Therefore, we will start right out with an example:

```
<table print-width="100%" screen-width="600px" align="center">
    <caption>An Exemplary Table</caption>
    <label name="tab:example"/>
    <colgroup>
        <col width="40%"/>
        <col width="30%"/>
        <col width="30%"/>
    </colgroup>
    <tr>
        <td>1st row, 1st column</td>
        <td>1st row, 2nd column</td>
        <td>1st row, 3rd column</td>
    </tr>
    <tr>
        <td colspan="2">2nd row, cell over 2 columns</td>
        <td>2nd row, third column</td>
    </tr>
</table>
```

As you can see, tables may be given a caption and a label. A table's label can be referenced with the `ref` and `pageref` elements (s. section 4.7).

Before writing down the table's contents, you must first specify its column layout. This is done with the `colgroup` element. For each column in your table, you *must* specify its relative width with respect to the overall table width.

The overall width is specified via the table attributes `print-width` and `screen-width`. The table's width in printed documents can be a value in centimeters (cm) or points (pt) or a percentage of the overall width of the text body. The width in on-screen documents may be specified in pixels (px) or in percent of the window width.

The horizontal alignment can be controlled by setting the `align` attribute to `left`, `center` or `right`.

After the layout specification follows the actual content. In our example, you see two table rows, set with the `tr` tag. The first row contains three cells, set with the `td` tag, whereas the second row contains only two cells. This is because the first cell in the second row spans two columns, as specified by its `colspan` attribute.

5.3.1 Activating the Grid Rules

The table above does not have a grid. To activate the grid rules of a table, you have to use the `frame` attribute. Take a look at this example:

```
<table print-width="100%" screen-width="600px" align="center"
    rulecolor="#000000" print-rulewidth="2pt" screen-rulewidth="2px"
    frame="top,bottom,rowsep">
    <caption>An Exemplary Table</caption>
    <label name="tab:example"/>
    <colgroup>
        <col width="40%"/>
        <col width="30%"/>
        <col width="30%"/>
    </colgroup>
    <tr frame="colsep">
        <td>1st row, 1st column</td>
        <td>1st row, 2nd column</td>
        <td>1st row, 3rd column</td>
    </tr>
    <tr>
        <td colspan="2">2nd row, cell over 2 columns</td>
        <td>2nd row, third column</td>
    </tr>
</table>
```

The frame attribute contains a comma separated list of grid element names: `left`, `right`, `top` and `bottom` activate the outer table borders; `colsep` and `rowsep` activate column and row separators. Note that column and row separators can also be activated for individual rows or cells.

The thickness of the grid rules may be specified with the `print-rulewidth` and the `screen-rulewidth` attributes. The color of the rules can be controlled via the `rulecolor` attribute. The color value must be an HTML style RGB triplet in hexadecimal notation.

5.3.2 Coloring Table Cells

You may color individual rows or cells by setting a `color` attribute on the corresponding tag. For example, to give the first cell in the first row from the previous example a gray background, you could write:

```
<tr frame="colsep">
    <td color="#dddddd">1st row, 1st column</td>
```

```
    <td>1st row, 2nd column</td>
    <td>1st row, 3rd column</td>
</tr>
```

Note *Colored cells may overlap with dark grid rules when viewing PostScript or PDF documents on screen. Therefore, you should avoid using colored cells and grid rules together or instead use white rules when working with colored tables.*

5.3.3 Text Alignment in Table Cells

The vertical alignment of text in tables can be controlled only for entire rows, but not for individual cells. This is due to LaTeX's limited capabilities in this respect. To determine the vertical alignment of text in a table row, set the `valign` attribute on the corresponding row element to one of the specifiers `top`, `middle` or `bottom`.

Horizontal text alignment must be controlled for each cell individually, by setting the cell's `align` attribute to `left`, `center` or `right`. By default, text is set left-aligned.

5.4 Floating Objects

By default, figures and tables are placed exactly where specified in the source document. Imagine though, you are generating printed output and so far the page has been filled by two thirds with text. Technically, the next thing to be inserted would be a picture, but it occupies more space than is remaining and thus has to be moved to the next page, leaving the page before empty by one third. This is visually unpleasant and in addition bloats your document unnecessarily.

As a soluting, you can turn figures or tables into floating objects by setting the `float` attribute on the main element to "yes". Making an object *float* means basically that you give LaTeX permission to move it to a different location in the text in order to warrant optimal text flow across pages.

5.5 Program Listings

You can use the `verbatim` element when you need to represent scripts or shell code. Text inside the `verbatim` tag will be printed in typewriter letters and whitespace will be printed just as it appears in your editor.

For program code, you should use the `listing` element which, for now, has a single child, namely the `code` element. You can have your code syntactically highlighted by specifying the name of the programming language in the `syntax` attribute. Take this for an example:

```
<listing>
    <code syntax="c" colorscheme="ide-eclipse"
        strip="yes" startline="1" linestep="100">

#include &lt;stdio.h&gt;
#include &lt;stdlib.h&gt;

int main(void) {

    printf("Hello World!\n");
    return 0;
}

    </code>
</listing>
```

The document processor can highlight a wide range of programming languages. For a complete list, take a look into the folder `lib/highlight/langDefs` below the installation folder.

With the `colorscheme` attribute you can choose between available coloring schemes. For a complete list, take a look into the folder `lib/highlight/themes` below the installation folder.

If you specify a `startline`, the syntax highlighter will number each line in your code. The `linestep` attribute specifies the increment from one line to the next.

Setting the `strip` attribute to "yes" will result in whitespace being stripped from the beginning and end of your listing.

5.6 Mathematic Formulas

Mathematic formulas are entered in TEX notation. Explaining TEX is beyond the scope of this document. For more information, please refer to appropriate literature, such as [4].

5.6.1 Inline Math

In order to set mathematic expressions inline, i.e. in the running paragraph, you use the m tag. Take a look at this example:

```
<p>
    The main proposition of Einstein's
```

```
    theory of relativity is <m>e = mc^2</m>.
</p>
```

5.6.2 Block Formulas

Formulas can also be set as block elements. To do so, you enclose the math element inside an equation element. To have your equation automatically numbered, set the number attribute to "yes". The following listing shows, how to set the eqation from above as a block element:

```
<equation number="yes">
    <m>e = mc^2</m>
</equation>
```

Support for math is not yet very sophisticated and the mechanisms presented above will not allow for more than the occasional mathematical one-liner. Future versions of eCromedos will provide better control over alignment and grouping of formulas.

Bibliography

[1] The Free Software Foundation. The GNU General Public License, version 2. June 1991. Available from `http://www.fsf.org` .

[2] Frank Neukam, Markus Kohm, Axel Kielhorn. The KOMA-Script bundle. March 2005. Available from `ftp://ftp.ctan.org/tex-archive/macros/latex/contrib/koma-script/scrguien.pdf` .

[3] Roger Kehr. Xindy – A Flexible Indexing System. Cahiers GUTenberg Vol. 28-29, March 1998, p. 223-230. Available from `http://www.xindy.org` .

[4] Tobias Oetiker, Hubert Partl, Irene Hyna and Elisabeth Schlegl. The Not So Short Introduction to LaTeX 2_ε. Version 4.16, May 08, 2005. Available from `ftp://tug.ctan.org/pub/tex-archive/info/lshort/english/lshort.pdf`.