

The KDE Documentation Primer

The KDE Documentation Team

Carlos Woelz <carloswoelz@imap-mail.com>

Revision 0.02 (2005-05-21)

Copyright © 2004 The KDE Documentation Team

Copyright © 2004 Carlos Woelz

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in [the section entitled "GNU Free Documentation License"](#).

This document tells you everything (hopefully!) you need to know to start writing documentation for KDE. Please report any errors or omissions to <kde-doc-english@kde.org>.

Table of Contents

1. Introduction

2. Getting Started

Things You'll Need

English Knowledge

Deciding What to Write About

Access to a Recent Version

3. KDE Writing Recommendations and Guidelines

Writing for your Audience

English Usage Guidelines and Recommendations

What to Include

4. Writing Documentation: Procedures and Tools

Getting the Documentation Sources

Working with plain text sources

Retrieving the DocBook sources

Quanta

Kate

Emacs and Psgml

Checking and Viewing the Documents

Using **checkXML**

Using **meinproc**

5. DocBook Introduction

Overview

Content and Presentation

Structure

KDE Specialities

Entities

Necessary Sections

6. Sending the New Documents and Changes to KDE

Respecting the Release Schedule

Managing the Sources with Subversion

Working With Other Documenters and Developers

Updating Documentation

Licenses for KDE Documentation

Using bugs.kde.org

7. Leveraging your Newly Acquired Knowledge

8. Credits and License

A. KDE DocBook Reference

General KDE markup style guide

Purpose of this document

Other reference material

The Prologue

book and the bookinfo section

Chapters and Sections

The linking elements

Lists

Tables

The GUI elements, menus, toolbars and shortcuts.

Describing actions and commands

Questions and Answers

Images and Examples

General markup (not covered elsewhere)

Admonitions: Tips, hints, and Warnings.

The synopsis elements

Markup for programming

Making Callouts

References, indexes, and glossaries

Making a glossary

Making an Index

Other Reference Sections

Tags we do not use

Alphabetical List of all elements

Credits and License

B. Widget Names

List of Examples

A.1. Managing translatable entities

- A.2. [Setting up the global `?kappname?` entity](#)
- A.3. [A KDE User Manual Prolog](#)
- A.4. [The bookinfo section from the KDE template](#)
- A.5. [A Segmented List](#)
- A.6. [An `<informaltable>` template](#)
- A.7. [A `<table>` template](#)
- A.8. [An example from a menu reference entry](#)
- A.9. [`<qandaset>` Template](#)
- A.10. [A screenshot example](#)
- A.11. [How to markup a command synopsis](#)
- A.12. [How to markup a function synopsis](#)
- A.13. [Marking up callouts with `<screenco>`.](#)
- A.14. [Marking up callouts by embedding directly in text](#)
- A.15. [How to markup a glossary](#)
- A.16. [Index](#)

Chapter 1. Introduction

The objective of this guide is to present all information required to make the experience of writing KDE documentation as easy as possible. The next chapter gives some information about what skills you'll need for the task. It is important to note that this guide is a joint effort of the KDE English Documentation Team and the KDE Quality Team. You can ask for support from both teams at any time.

The *KDE English Documentation Team* exists to provide end-user documentation for the whole of KDE. It's a big task, but an important one. Although KDE aims to be easy to use, not everything is obvious without some help, and, in a project this big, even an experienced user can't know every corner of KDE.

The team is made up of people doing several different tasks:

- Writing documentation for individual applications
- Writing wider documentation for the whole of KDE (like the User Guide, or this document).
- Proofreading and/or updating documentation to ensure that it is correct and up-to-date.

Contributors to all of these areas are always welcome. You can choose the area you would like to contribute to, based on your skills and what you enjoy doing. If you need any help with documentation issues, do not hesitate to ask at the KDE Documentation mailing list, [<kde-doc-english@kde.org>](mailto:kde-doc-english@kde.org), or on IRC in the channel `#kde-docs` on `irc.freenode.net`.

The *KDE Quality Team* provides support for new contributors, and to coordinates the efforts of the volunteers. The [KDE Quality Team Website](#) provides guides to help you with some general development tasks, such as getting the sources, [Building KDE From Source Step By Step](#), and [Working with Subversion](#), etc.. If these guides are not sufficient, and you are having problems with KDE development, we provide support for new contributors at the KDE Quality mailing list, [<kde-quality@kde.org>](mailto:kde-quality@kde.org), or on IRC in the channel `#kde-quality` on `irc.freenode.net`.

Chapter 2. Getting Started

Table of Contents

Things You'll Need

English Knowledge

Deciding What to Write About

Access to a Recent Version

If you got this far reading this document, you're probably interested in helping with KDE documentation. If so, welcome aboard! We're always happy to have new contributors, and whatever your skills, you can help make KDE even better.

Things You'll Need

To write documentation, there are only three things that are absolutely necessary: some English knowledge, knowing what you want to document, and access to a relatively recent version of the application you want to document.

Note

Notice that the list of requirements does *not* contain a requirement that you learn DocBook, or any of the other tools we use. We're very happy to receive documentation written in plain text. We would much rather have the content and have to add formatting than have no content at all.

English Knowledge

All KDE documentation is originally written in English, so you have to be able to write English to a reasonable level. That said, you *don't* need to be a native speaker, and you don't need to write word-perfect English. There are native English-speaking proofreaders on the documentation team, and we would much rather have *some* documentation that needs a little tweaking, than no documentation at all. If you don't feel comfortable writing in English, you might like to contribute to one of the KDE translation teams. You can find more information about translation on <http://i18n.kde.org>.

If you're a fluent English speaker with an eye for detail, you might be interested in joining the proofreading team. Just drop an email to [<kde-doc-english@kde.org>](mailto:kde-doc-english@kde.org) if you'd like to help the proofreaders.

Deciding What to Write About

KDE is a very large project, with many different parts and programs. Because of this, it can be hard to know where to start if you want to contribute. There are a few rules of thumb that can help you decide what to write about:

- Find a topic that you'll enjoy writing about; It will increase your motivation and help you to produce better documentation.
- Write about an application you know well. You'll be able to spend more time on writing and less time trying to work out how the application works. On the other hand, documenting an application can be a good way to learn about how it works, especially if you like a challenge!

If you are looking for an application to document, or just checking the status of the application you want to work with, the [KDE Quality Team Wiki](#) contains lists of applications, organized by modules, and their general status, including documentation status, and who is working on it. Not all modules and applications are included or up to date, but it is certainly worth checking.

If you start documenting one of the listed applications, please add your name to the wiki pages as well. But If you just can't find an application to work with, write to [<kde-doc-english@kde.org>](mailto:kde-doc-english@kde.org) and ask for suggestions. There's always something available to do, but there's no obligation to work on a particular application. Also, contributing to a document doesn't force you into keeping that document up-to-date (although if you can do that, it's very welcome!).

Another place to check is the KDE bug list at <http://bugs.kde.org>. This is usually more detailed than the wiki, and provides a place to list specific small changes that are needed to documents. These are often nice small jobs to get you started contributing. A set of quick links to ready made queries are available from the Documentation Project's <http://i18n.kde.org/doc/current.php> page.

It is also helpful to the team to file more bugs like these above. You will need a bugzilla account, and a recent copy of KDE. Simply open an application, choose Help->*appname* Handbook. Then just read through the document, following along in the application. KDE applications are a moving target to document, and sometimes the documentation has not yet caught up with a change to the interface or behavior of an application. Feel free to file bugs for any of these issues you find, in order of urgency:

- Inaccurate information about how an application works

For instance, if you previously needed to save changes to a file before they take effect in the GUI, and this now happens automatically, text referring to manual saving should be removed, or it will confuse readers.

- GUI options or menu items (or sometimes, entire dialogs)

This often happens in configuration dialogs, when new items are added, a new grouping of existing options may be created.

- New Features that are available and are not yet documented.

Access to a Recent Version

To make sure that the documentation you write is up-to-date, you'll need to run a recent version of the application you are working with. This normally means a recent beta version, a version of your application compiled from sources or a version of KDE compiled from sources in the Subversion repository. If you think that compiling from sources is too burdensome, and you cannot get some recent beta packages, there

are still some interesting possibilities to work around this requirement:

- Write about a stable application: there are many apps with a stable interface which are still lacking good documentation. In this case, the last stable version provided by your distribution will be sufficient to write about it, no compiling required.
- Using a remote desktop connection to preview the development version is an ideal solution to this problem. The FreeNX terminal server technology enables decent desktop performance even with dial up Internet connections. We are planning to offer this service to KDE documenters, but the infrastructure is not yet in place (as of May 2005). You may ask the kde-quality@kde.org mailing list about it, if you think this is the way to go.

If you want to try out building KDE from sources, the KDE Quality website provides [a detailed, step by step building guide](#). You can find even more information at the [KDE Developers Website](#). If you face any problems in the compiling process you can't solve by reading the building guide, don't hesitate to ask for help on kde-quality@kde.org.

Chapter 3. KDE Writing Recommendations and Guidelines

Table of Contents

[Writing for your Audience](#)

[English Usage Guidelines and Recommendations](#)

[What to Include](#)

To maintain a uniform documentation set, there are some consistency rules to be followed, that you should know before starting. In this chapter you will find guidelines about targeting your audience, English usage, and what to cover when you are documenting an application.

We also offer some general writing tips to help you to get started, provided by experienced KDE documenters.

Writing for your Audience

Since KDE is used by people with a wide range of abilities, from completely new users to long-time gurus, the documentation should be appropriate to this audience. Therefore, in general, documentation shouldn't assume too much about the knowledge of the reader, without being patronizing. There are no hard-and-fast rules, but here are some tips that should help:

- Remember that the audience varies with the application: for example, a server control module has a very different user base than a user of a game, and the manuals should reflect this. Don't insult the administrator intelligence, and don't assume knowledge for the gamer that might not be there.
- Keep a logical progression of difficulty: Keep the first few pages of the document simple, and accessible to users who have never seen the application before. More technical information should appear towards the end of the document.

Remember also that different types of documentation have different purposes:

Application Handbooks

These may go into great depths on the configuration, behavior and sometimes the philosophy of an application. There is scope to cover corner cases of configuration, commonly asked questions, and advanced troubleshooting techniques.

They should also always contain a complete reference to all the available menu functions and configuration options for the application (but while these are required, they should be certainly be considered a minimum of information to provide.)

The Application Handbook should be answering the question: ?What are all the things I can do with this application??

User Guide

A much higher level overview of KDE and its applications. This aims to be the first stop for users to look for information, and should be task based.

When writing for the User Guide, you should assume a working default installation of KDE, and you do not need to cover all cases of unusual configurations, only the very common variations, nor should you cover in-depth troubleshooting. You might provide answers to some very common configuration errors (or not, as appropriate) and refer to the Application Manual, the Application's Website, mailing list, and any appropriate man pages for more detailed information.

Most people reading this guide do not have an actual problem, they simply want to achieve a goal, and don't yet know how, or where to find that information.

The User Guide should be answering the question: ?How do I do *this common task*, e.g. *send an email, play a movie*??.

What's This Help

A very focused and specific type of assistance, about a single configuration or interface item. Again you should not really attempt to cover all cases here, only common ones, and explain what the option does, not why it is there. Refer users to the Application Handbook if appropriate, for more information.

Provide an example of the expected input, if that is not clear from the context.

The What's This Help is most often answering the question: ?Do I need to fill in this box? If so, what do I put in it??

English Usage Guidelines and Recommendations

KDE documentation is written in standard US English (rather than any other regional variety of English). We have a set of standard forms of certain words (such as ?email? instead of ?e-mail?) to improve consistency across all documentation. Work is underway to expand and formalize this list, but for the moment, it is located at <http://www.kde.me.uk/index.php?page=Consistency+rules>. There are also standard names for KDE widgets, which are listed in [Appendix B, *Widget Names*](#)

A good way to catch simple errors is to read the text out loud, or have someone else read it to you. Passages that don't flow easily or have obviously awkward construction of the type you may miss on the screen, will usually become blindingly obvious when you hear them. This is especially the case with detecting really long sentences, as you will run out of breath and turn blue.

Some tips about writing readable sentences:

- Use complete sentences. Not fragments. Like these ones.
- Avoid run-on sentences, sentences that cover several different subjects, or sentences that could be broken up into several sentences; avoid sentences that can fill a whole paragraph all by themselves and that are really long, like this one, which is all of the above.
- Use a comma before ?and? in compound sentences, e.g. ?Use the left mouse button to select and copy text, and the middle mouse button to paste it.?
- Keep to logical sentence order.

For example, ?Konqueror is a web browser with the ability to browse file systems and it includes a javascript interpreter.? (Do you see why this is awkward?)

- Try not to use the same word several times in the same sentence. An exception to this, is an application command or technical word, where this repetition is necessary, and improves clarity.
- Do not start sentences with any of ?and,? ?so,? ?but,? ?because,? or ?however.?
- Try to avoid contractions, rather spell out both words; e.g., ?it is? rather than ?it's?; ?can not? rather than ?can't?
- There is no need to worry about simple text formatting such as leaving two spaces after punctuation or indenting paragraphs. This is all handled by DocBook XML and the XSLT stylesheets in use.

Remember, we have also an active proofreading team, and there is always someone to help you with grammar, so just write and have fun!

What to Include

For most applications, a structure something like this would be appropriate:

1. Introduction: A basic description of what the application does and any noteworthy features, etc..
2. Using *KApp*: Task-based description of the most common uses of the application.
3. Program reference: Description of all of the features of the application. This would usually include a menu reference, but might also include command line options, syntax description, etc., if they are appropriate to the application.

This is required for all KDE applications that you at a minimum cover any application specific menu entries, and strongly recommended that you cover all the standard ones too (in case users are reading the manual outside of KDE, or yours happens to be the first one they read, and it provides consistency. Cut and paste is your friend here.)

Note that although this is a required section, and for some applications it is the only section, it should be considered a minimum.

4. Frequently Asked Questions: List the most common questions and problems that users have with the application, and their solutions. ?How do I ...??-type questions are especially appropriate.
5. Credits and License: A list of those who contributed to the documentation, and a link to the GNU Free Documentation License, under which all KDE documentation is licensed.

This chapter is required for all KDE documents, and must have *at least* the two license links (one for the document, and one for the application)

6. Installation: This chapter can be automatically generated, provided that the application follows the usual KDE compilation procedure (i.e. **./configure, make && make install**). If you need to add extra information about compiling or installing the application, it can go here.

You will find a template document with these sections in

trunk/KDE/kdelibs/kdoctools/template.docbook file in KDE Subversion repository.

Chapter 4. Writing Documentation: Procedures and Tools

Table of Contents

[Getting the Documentation Sources](#)

[Working with plain text sources](#)

[Retrieving the DocBook sources](#)

[Quanta](#)

[Kate](#)

[Emacs and Psgml](#)

Checking and Viewing the Documents

Using **checkXML**

Using **meinproc**

If you're worried about having to learn a lot of new tools and procedures in order to write documentation, you don't need to, because the information we've covered so far is everything you need to know to be able to contribute. Although we *do* have some tools we use and procedures we follow, it's not vital that everyone knows them in detail, especially when starting out.

For example, all KDE documentation is written in DocBook XML, but we're very happy to receive documentation written in plain text. There are people on the documentation team who are very familiar with DocBook, and can easily add the markup if the content is there.

Another example: if you are starting to document an application from scratch, you don't need to get the sources of the current documentation. But if you are starting from existing documentation, you don't need to know about how to get the sources, there are other means to do that.

Of course, if you want to learn about DocBook, you can. After a little practice, you will probably find that it's not as hard as it looks. And if you learn about dealing with a Subversion repository, you will be able to integrate yourself to the regular KDE development process (upload your changes, work together with other developers, etc.)

Getting the Documentation Sources

Note

If you are starting your document from scratch, you probably do not need to read this section, and may start working right now.

You are welcome to use plain text to contribute to KDE documentation. It is a great way to start, and we strongly encourage it. If you will miss the power of the DocBook format as you improve your documentation skills, then you can learn it. In the mean time, someone will manually edit the plain text to add the DocBook markup and commit it to KDE Subversion repository, removing the burden of doing most of the more complex stuff covered in this very guide. We'll take a look at writing in DocBook and using Subversion later in this document, so if you're interested, read on, but if you want to use plain text, you can go directly to [the section called ?Working with plain text sources?](#).

Documentation for KDE, like the rest of the source code, is kept in a Subversion repository. Subversion provides a way for many developers to work on the same source code (or in our case, the same documentation), and has many useful features to help with this. For example, previous versions of every file are saved so that any mistakes can be quickly backed out, if they can't be easily corrected.

The basic principle behind Subversion is simple: one server stores a definitive copy of the files making up a project. This is known as the *repository*. Each developer can download the files to make their own private copy, named *working copy* or *sandbox*. Using Subversion, the developers can upload their modifications to the main repository (a process called "committing") or update their own copy to reflect recent changes made by others.

There are two main ways edit the contents of a KDE document you want to improve: using plain text or DocBook.

Working with plain text sources

The docs.kde.org website displays most of the KDE documentation in HTML format, updated daily from the Subversion repository. There are two versions available in the website: the [stable version](#) and the [KDE from Subversion version](#). You will always use the latest version of the documentation, i.e. the [KDE from Subversion version](#).

The docs.kde.org website presents a quick and easy method of retrieving the latest version of the KDE documentation. Clicking the name of the application you want to document in the list will open the documentation in your web browser. Simply copy the text from the website to your favorite text editor, edit it, and submit the results in plain text to the KDE Documentation mailing list, [<kde-doc-english@kde.org>](mailto:kde-doc-english@kde.org). Please note that not all KDE applications are listed there. If you cannot find the documentation of the application you want to work with, then you can request it by sending a message to the KDE Documentation mailing list.

Now you know everything you need to start working. When you are finished writing, you may want to read [Chapter 7, Leveraging your Newly Acquired Knowledge](#). Have fun!

Retrieving the DocBook sources

The latest DocBook sources are located inside the KDE repository. Now you need to find and retrieve them.

The software inside the KDE repository is divided into *modules*, which are used to organize the different software projects inside the repository. Modules are the top-level folders in the Subversion repository folder tree, and each one contains a group of related applications. These modules are sometimes released in binary form as *packages*. If you know the name of the package your application belongs to, you probably know the module name as well, as they are frequently the same. You need to know in which module your application is, to retrieve its DocBook sources. For instance, KMail is in the *kdepim* module, Quanta in the *kdewebdev* module, Cervisia in the *kdesdk* module and so on. If you need any help in this process, don't hesitate to ask. Each module contains a folder named "doc", and inside it, you can find the DocBook sources.

To access the repository, you can use the **svn** command line application or browse the [KDE WebSVN website \(websvn.kde.org\)](http://kde.websvn.org).

The websvn.kde.org is a web based representation of the contents of the Subversion repository. It is easy to download files using websvn.kde.org, the operating system or desktop you use does not matter.

Retrieving your own working copy of the repository has many advantages. You will be able to use your working copy to create files containing the changes you made, to update your copy with changes made by other documenters, and if you get a KDE Subversion account, to upload your changes directly to the repository. But this is out of the scope of this section. Here we will show you simply how to retrieve the sources using Subversion the easiest way we can. You can get more information about these tools (they

are really useful) by reading the [the section called ?Managing the Sources with Subversion?](#).

Procedure 4.1. Retrieving documentation sources using WebSVN

1. Go to <http://websvn.kde.org> using your favorite web browser. Let's suppose you are looking for Cervisia's documentation sources.
2. The KDE repository is divided into ?trunk? (also known as HEAD, where development is going on, ?branches?, where both stable and working branches live, and ?tags?, where you can retrieve snapshots of sources at a release. Most work for documentation goes on in ?trunk?, so click there.
3. The main KDE modules are in the ?KDE? folder, so click on that.
4. Click the "trunk" link to get the main branch listing. Click on "KDE" to get the list of modules from a KDE release.
5. Cervisia is part of the kdesdk module (KDE software development kit module). Therefore, click the ?kdesdk? item on the list. The contents of the kdesdk module will be displayed.
6. Click the ?doc? item on the list, to see the contents of the documentation folder of the module. The contents of the doc (documentation) folder will be displayed.
7. Select the application you want to work with from the list (in our case, ?cervisia?). All Cervisia's documentation source files will be displayed, being images or DocBook files.
8. Now you reached the list of files that are part of Cervisia's documentation, including images and DocBook sources. The DocBook sources are files in the format * .docbook. In this case, there is only one file in this format: `index.docbook`. Click this file on the list. A list of *revisions* (versions) of that file will be displayed.
9. Click the ?download? link from the revision on the top of the list. It is the most recent one. Save the file. Repeat this process with all the files you want to download.

We use KMail's documentation sources as example in the following procedures.

Procedure 4.2. Retrieving documentation sources using Subversion

1. Check if you have the Subversion client installed (hint: enter `svn` in the terminal screen). If not, install the Subversion package using the tools provided by your distribution.
2. Now it is time to download, or *checkout* the sources. Using Subversion, type in the terminal:

```
mkdir path/to/working/folder
cd path/to/working/folder
svn checkout svn://anonsvn.kde.org/home/kde/trunk/KDE/module/doc/application
```

where *path/to/working/folder* is the folder you want to install the sources in your system, *trunk/KDE/module* is the application's module location in the repository and *application* is the application name. Remember to use small caps to type the application and module names. In our example, KMail is in the kdepim module, so you would enter:

```
svn checkout svn://anonsvn.kde.org/home/kde/trunk/KDE/kdepim/doc/kmail
```

Note that only applications which are part of a regular KDE release are under *trunk/KDE/*. Amarok docs, for instance, is in the multimedia module of extragear. Extragear contains mature applications which are not part of a KDE release. To get Amarok docs, type in the terminal:

```
svn checkout svn://anonsvn.kde.org/home/kde/trunk/extragear/multimedia/doc/amarok
```

Quanta

Quanta is a friendly editor for SGML and XML documents. Quanta features syntax highlighting, autocompletion, autoclosing and code folding for DocBook tags, easy access for the KDE documentation tools, **meinproc** and **checkXML**.

A screenshot of Quanta's main window

A screenshot of Quanta's main window

Some of the tools available for DocBook editing are the document structure sidebar, tag editor sidebar and, starting with Quanta 3.4 (which is part of KDE 3.4), Quanta offers a DocBook toolbar, complete with table and list wizards, ui elements, admonitions, KDE tools and other standard tags. While Quanta offers a visual page editor for html and xhtml pages, there is no support yet for DocBook visual editing. We highlight here some of these features.

DocBook Toolbars

The DocBook toolbars offer easy access to the most common DocBook tags, plus the list, table and image wizards. You can check your DocBook document using the [IMAGE] checkXML button from the Tools toolbar: the output of the script will be displayed in the Messages sidebar, in the bottom of Quanta's main window. If there is no output, that usually means no errors. To process the DocBook into html files, use the [IMAGE] meinproc button on the same toolbar.

Note

Depending on the version of some XML utilities used by Quanta, the [IMAGE] checkXML and [IMAGE] meinproc scripts can present bugs. Starting from the upcoming KDE 3.4.2 release, these bugs will not exist anymore. But until there, if you experience these bugs, (in special if Konqueror is not starting up when using the meinproc script or there is no output when using the checkXML script, you can get and install the [updated docbook scripts from kde-files.org](http://kde-files.org) to solve these issues.

A screenshot of Quanta's DocBook toolbar

A screenshot of Quanta's DocBook toolbar

Tag Editor

The tag or attributes editor is located on the right sidebar, and it shows the available attributes for the tag which is currently being edited. The tag editor helps you to edit the attributes for the current tag: just click on the Value column of any attribute to edit it.

A screenshot of Quanta's attribute editor sidebar

A screenshot of Quanta's attribute editor sidebar

Documentation Sidebar

Another useful feature is the documentation sidebar, which allows you to download and use documentation packages as offline reference. This guide is also available offline, using Quanta's documentation sidebar. Just grab and install the [KDE Doc Primer documentation package](#). The documentation sidebar is on the right side of the main window.

A screenshot of Quanta's documentation sidebar

A screenshot of Quanta's documentation sidebar, showing the KDE Doc Primer

Entities Autocompletion

Quanta offers autocompletion for entities. However, this feature is hardly useful without the KDE entities definitions. To generate the entities list for the KDE, follow the procedure below:

Note

The autocompletion feature still has some bugs in the 3.4.1 release. These bugs are fixed, and will be available starting from the 3.4.2 release.

Procedure 4.3. Generating and installing the `entities.tag` file

1. Open Quanta. Choose the DTD->Load & Convert DTD menu item.
2. Now, we have to select the right dtd file to convert. On the dialog, select the KDE installation folder (usually `/usr` or `/opt/kde3`. If you cannot find it, type

```
$kde-config --prefix
```

on a terminal application. The dtd file we want is named `kdex.dtd` under `share/apps/ksgmltools2/customization/dtd/`. Select it and press OK. A new Document Type Editing Package (DTEP) for `kdex` will be created.

3. Now that you have converted the dtd, you can either use it directly, by choosing the DTD->Change the DTD... and selecting the kdex dtd. But the best solution is to install the `entities.tag` file for automatic use with the KDE docbook dtds.

Now, let's copy the file from the kdex dtep to the kde-docbook dtep. You can use a console application or a file manager to perform this action. These locations are under the `KDEHOME` folder, the folder that contains your KDE settings and application data, usually, `~/.kde`. If you cannot find it, type

```
$kde-config --localprefix
```

on a terminal application. The dtep folder is under `KDEHOME/share/apps/quanta/dtep`. The simplest way to do copy it is using a terminal application (e.g. Konsole).

Start a console application and enter the command:

```
$cp `kde-config --localprefix`/share/apps/quanta/dtep/kdex/entities.tag `kde-config \
--localprefix`/share/apps/quanta/dtep/kde-docbook-4.1.2/entities.tag
```

4. Restart Quanta.

A screenshot of Quanta's entities auto-completion feature

A screenshot of Quanta's entities auto-completion feature

Document Structure

finally, the document structure displays the logical representation of your document. By left mouse button clicking on an element, your cursor will taken to the element's position in the document. By right mouse button clicking on an element, you are presented with a number of actions that deal with navigating and updating the tree.

A screenshot of Quanta's document structure sidebar

A screenshot of Quanta's document structure sidebar

Quanta is part of the `kdewebdev` module, which is released as part of KDE. Binary packages are available for the majority of the distributions. Quanta can be easily extended to support custom scripts, toolbars and documentation sidebars. For more information, check the application handbook.

Kate

Kate is an extensible and powerful text editor which is part of the kdebase module. Kate can syntax highlight DocBook documents out of the box, and is generally a very powerful editor, but you can get even more XML specific functionality installing the XML plugin for Kate.

Procedure 4.4. Installing the XML plugin for Kate

1. The XML plugin for Kate is available as part of the kdeaddons module, which is released as part of KDE. Binary packages are available for the majority of the distributions. Install the binary package using your distribution tools or compile kdeaddons to install the plugin.
2. Open the Configure Kate dialog by choosing the Settings->Configure Kate... menu item.
3. Select the Plugins item from the Application tree. Check the Kate XML Completion and the Kate XML Validation boxes.

A screenshot of Kate's Plugin Manager Configure Dialog

A screenshot of Kate's Plugin Manager Configure Dialog

4. Press OK.

With the XML plugin for Kate installed, you will have autocompletion, autoclosing for DocBook tags and entities. Since KDE documentation uses entities widely, this is a very welcome feature. Additional XML tools will be available through the XML menu (in special, through the Validate XML menu item, which will allow you to check your DocBook documents). The output of this action will appear in the XML Checker Output button in the side bar located in the lower part of Kate's main window.

A screenshot of Kate's Main Window showing the XML Checker Output

A screenshot of Kate's Main Window showing the XML Checker Output

Emacs and Psgml

The venerable Emacs editor has a powerful SGML and XML editing mode called psgml. The price of this power is a steeper learning curve than the other editors, so if you haven't used Emacs before, you will probably want to try the other editors first. If, on the other hand, you're already familiar with Emacs, then psgml is your best choice.

Installation of psgml is beyond the scope of this document, but it should simply be a case of installing appropriate packages for your distribution. The relevant configuration for KDE-related documentation is simple. Just tell psgml where the KDE catalog files are located with the following line in your `.emacs` file:

```
(setq sgml-catalog-files
  (list "CATALOG" "KDEDIR/share/apps/ksgmltools2/customization/catalog"))
```

where you should replace *KDEDIR* with the path to your KDE installation. You might also want to use the following line to instruct Emacs to use psgml to open all `.docbook` files:

```
(setq auto-mode-alist
  (cons '("\\.docbook$" . sgml-mode) auto-mode-alist))
```

There are of course plenty of other settings in psgml mode which you can change to your taste: see the psgml **info** documentation for more details. A sample `.emacs` file, with some customizations useful for writing KDE documentation, can be found at <http://people.fruitsalad.org/phil/kde/dot-emacs-psgml>.

Some basic keystrokes in psgml are:

Ctrl+C /

End current element. This inserts an end tag for the currently open element.

Meta+Tab

Completes the current tag or entity, context-sensitively. This will only complete on tags that are allowed at the current point in the document. Note that, because indentation is rarely used in KDE documentation, it is generally safe to remap this function to just the Tab key.

Ctrl+C Ctrl+F Ctrl+E

Fold current element. This compresses the current element so that only the starting tag appears. One use of this is to fold all the `chapter` elements in a document, to get an overview of the document on one screen, and make navigation around a long document easier. You can unfold elements with the shortcut **Ctrl+C Ctrl+U Ctrl+E**.

One particularly useful psgml feature that isn't well documented is the `sgml-parent-document` variable. Setting this variable appropriately tells psgml that this file is part of a larger document. This enables the full range of psgml features for this file, such as context-sensitive element completion. To use this feature, place the following in a comment at the end of the child file (with the arguments adjusted appropriately):

```
Local Variables:
sgml-parent-document: ("index.docbook" "book" "chapter")
End:
```

The first argument is the name of the parent file (which will almost always be `index.docbook` in KDE documentation). The second argument is the top-level (or `?root?`) element of the whole document (i.e., in the parent file). The third argument is the top-level element in this file.

Checking and Viewing the Documents

There are a couple of KDE-specific tools for manipulating DocBook files, namely **meinproc** and **checkXML**. **checkXML** (as the name suggests) is used to check that documents are valid, well-formed XML, and **meinproc** converts DocBook files to HTML. Here's some hints on using each of them:

Using checkXML

checkXML is a simple command with only one argument: the file to check. However, the output can be a bit daunting, since one small mistake can cause a cascade of errors. The trick is to look at the first error, fix that error, save the file, and run **checkXML** again. Often, fixing that one error will get rid of all the other error messages. When running **meinproc**, the same procedure applies.

Most errors in DocBook sources fall into one of a few categories. Here are descriptions of some of the most common errors and their solutions:

Opening and ending tag mismatch

```
index.docbook:880: parser error : Opening and ending tag mismatch: para
line 879 and sect2
</sect2>
      ^
```

This is possibly the most common type of error. It's caused either by an element that hasn't been closed, or by tags that overlap. The error above was generated by the following markup:

```
<sect2>
...
878: running &meinproc;, the same procedure applies.</para>
879: <para>&checkxml; is a simple command with
880: </sect2>
...

```

The `<para>` tag on line 879 has not been closed before the `</sect2>` on line 880, causing the error. The simple fix in this case is to add a `</para>` before the closing `</sect2>`.

Element does not follow the DTD

```
index.docbook:932: element qandaentry: validity error : Element qandaentry content
does not follow the DTD, expecting (blockinfo? , revhistory? , question , answer*), got (answer)
</para></answer></qandaentry>
      ^
```

This error is caused by an element in the document not matching the requirements of the DocBook DTD (Document Type Definition). The DTD specifies what each element must contain. This list is shown after expecting in the error message. This so-called "content model" is quite difficult to understand at first: refer to the Duck Book and the section "Understanding Content Models" for full information.

The text after `got` shows the content actually found in the document.

In the example above, we have a `qandaentry` which is missing the required `question` element. This was generated by the following input:

```
<qandaset>
<qandaentry><answer><para>An answer
</para></answer></qandaentry>
</qandaset>
```

Adding a `question` element before the answer fixes the problem.

An easy mistake to make is to forget to put a `para` element around text in, for example, a `listitem` or a `sectn`. This will be shown as `CDATA` in the `got` section of the error.

Using `meinproc`

The most common way to run `meinproc` is simply as

```
meinproc docbook-file
```

where *docbook-file* is usually `index.docbook`. This command creates HTML pages from the DocBook file. Note that these pages are only viewable in KDE-based browsers (like Konqueror). If you need to view the HTML output in another browser (for example, if you're placing it on line), use

```
meinproc --stylesheet stylesheet-name docbook-file
```

where *stylesheet-name* is the full path to one of the XSL stylesheets in `$KDEDIR/share/apps/ksgmltools/customization`. To produce output suitable for the web, you can use `kde-web.xsl` or `kde-chunk-online.xsl`. See the `README` file in that directory for more details.

Chapter 5. DocBook Introduction

Table of Contents

[Overview](#)

[Content and Presentation](#)

[Structure](#)

[KDE Specialities](#)

[Entities](#)

[Necessary Sections](#)

All KDE documentation is produced in DocBook XML format, and writers are encouraged to learn it (although it's by no means necessary, and we're very happy to receive documentation written in plain text). Although DocBook can look somewhat intimidating to beginners, the markup is extremely self-descriptive, and many people find it easier than HTML to learn.

In this chapter, we'll just take a basic overview of the ideas behind DocBook. For detailed information about individual tags and so on, please see [Appendix A, KDE DocBook Reference](#).

Overview

DocBook is just an application of XML, so if you're familiar with XML, then you'll feel right at home. If not, don't worry, as most of the gory details aren't required knowledge for simply writing and updating documentation. A DocBook file (and, indeed, any XML file) consists of plain text, with tags surrounding some text to tell you (or a computer) what that text represents. So, a snippet from a DocBook file might look like:

```
<para>To display the clipboard history, click on the &klipper; icon
  in the &kde; panel, or press <keycombo
action="simul">&Ctrl;&Alt;<keycap>V</keycap></keycombo>. Previous
clipboard entries are shown
  at the top of the pop-up menu which
appears.</para>
```

The `<para>` and `</para>` show the start and end, respectively, of a paragraph. These delimiting marks are called *tags*, and the content they contain (along with the tags) is called an *element*. The `<keycombo>` tag has an extra piece of information specified: `action="simul"`. This is called an *attribute*, and makes the tag more specific. The words surrounded by `&` and `;` are *entities*. They're simply variables that expand to some other text, and are widely used in KDE documentation. See [the section called 'KDE Specialities'](#) for more information about entities. Tags, entities, comments and other parts of XML that aren't simple text are referred to as *markup*.

Content and Presentation

One of the basic principles behind the use of DocBook in KDE is that content and presentation are strictly separated. DocBook files contain the content, and XSL files contain the information about presentation. This has a number of advantages, some of which are:

- When writing, you don't have to worry about whether the information is well presented, just that the information you're writing is correct and readable.
- All KDE documentation has a similar look, so once readers are familiar with conventions in one document, they're familiar with all documents.
- Documentation is future-proofed, since by providing as much information about content as possible, future formats, search engines, etc. are likely to be catered for easily.

In practice, this means that you should add markup that describes what things *are* and not how they should appear. So, in the example above, the `<keycombo>` (a keyboard shortcut) tells the reader (or computer) that the keys **Ctrl**, **Alt** and **V** should be pressed simultaneously, but doesn't say anything about how that should be displayed in the final output. (In fact, it appears as *Ctrl+Alt+V*, but it could equally be converted to *C-M-V* à la Emacs or even some other way of showing keyboard shortcuts. What's important is that the DocBook source has the *information* necessary to work out what is being referred to.)

Structure

(<book> <chapter> <sectn> <para>)

KDE Specialities

KDE-isms: entities, necessary bits (credits, translation stuff)

Entities

Entities (which are simply variables which expand to some other text) are an important part of DocBook markup, and are used particularly widely in KDE documentation. For example, there are entities defined for almost all KDE applications. Therefore, when referring to, for example, Konqueror in documentation, you should use:

```
&konqueror; is, among other things, a  
web browser.
```

This has several advantages. Firstly, it ensures that applications are capitalized and marked-up consistently across all KDE documentation. This means that you don't have to remember whether the help center program is KHelpCenter, KHelpcenter or Khelpcenter: the entity (which is always entirely lowercase) automatically expands to the correct one.

There are entities defined for several classes of names:

All KDE applications

As mentioned before, all KDE applications have an entity. The entity name is in entirely lowercase, and expands to the correctly capitalized version of the application name. There is also an entity for KDE itself: `&kde;`.

Common English and technology abbreviations

For example, *?i.e.?* is written as `&ie;` and *?e.g.?* as `⪚`. This ensures that the same markup and capitalization are used for these abbreviations throughout KDE documentation. Technological abbreviations such as HTTP and XML also have entities, which are capitalized as usual (i.e., `&HTTP;` and `&XML;` for the previous examples).

Trademarks

Names of companies and their products are often trademarked. For this reason, it is important to mark them up with the `trademark` tag, using the `class="registered"` attribute if necessary. To reduce effort, and ensure that trademarks are given proper acknowledgment, many common technology-related trademarks have been given entities. For example, the entity `&X-Window;` expands to X Window System®.

Contributor Names

Names of contributors to KDE documentation have entities of the form `&Firstname.Lastname;` (or `&Firstname.Initial.Lastname;`). Email addresses of contributors have entities of the form `&Firstname.Lastname.mail;`.

Names of special keys

Names of keys on the keyboard are always marked up with either `keycap` or `keysym`. Since it can be difficult to distinguish between these two tags, entities have been created for common keys, e.g., `&Ctrl;` and `&Alt;`.

The definitions of these entities can be found in the following locations in KDE 3:

Items not requiring translation (KDE application names, technology abbreviations, trademarks)

```
kdelibs/kdoctools/customization/entities/general.entities
```

Contributor names and email addresses

```
kdelibs/kdoctools/customization/entities/contributor.entities
```

Language-specific terms and key names

```
kdelibs/kdoctools/customization/en/user.entities
```

Necessary Sections

There are several sections that appear in all KDE DocBook files, even though they are not required by DocBook itself:

-

```
<!ENTITY package "kde-module">
  <!ENTITY % addindex "IGNORE">
  <!ENTITY % English "INCLUDE">
```

This appears in the prologue immediately after the FPI. See [the section called 'The Prologue'](#) for more details about this section.

-

```
<!-- TRANS:ROLES_OF_TRANSLATORS -->
```

This appears after the `<authorgroup>` element, and is a required placeholder for use in translation (also known as `?i18n?` from the number of letters between the first and the last of the word `?internationalization?`).

Chapter 6. Sending the New Documents and Changes to KDE

Table of Contents

[Respecting the Release Schedule](#)

[Managing the Sources with Subversion](#)

[Working With Other Documenters and Developers](#)

[Updating Documentation](#)

[Licenses for KDE Documentation](#)

[Using bugs.kde.org](#)

As part of the wider KDE project, there are some things that documentation writers need to be aware of. There are a large number of other developers working on KDE, and working together with all of them is an important part of what we do.

Respecting the Release Schedule

String freezes, when we write, etc

Warning

This needs reviewing by someone who pays more attention to releases than I do.

The KDE release process, in which we go from the fast-moving and sometimes unstable world of the KDE Subversion repository, to a stable, polished product, is never exactly the same twice, but there are some common features:

A schedule for the next release of KDE is published at developer.kde.org, with the definitive guide to what will be happening and when. There will be two or more ?freezes?, when changes of a certain type are not allowed in the KDE Subversion repository:

Feature Freeze

When feature freeze is active, developers are not allowed to commit new features to the repository. This is a good time to start writing, since the features available in the application during this period are the same as the ones which will be available in the released version.

String Freeze

Text strings appearing in the KDE user interface and in the documentation are not allowed to be changed. This is to allow translators to provide thorough translations which will match the release. We are still considering how to work during this period of freeze. One method which we have tried is to continue writing, but hold back all changes to be committed in one go, immediately before the release.

Managing the Sources with Subversion

You can find detailed information about how to use Subversion in conjunction with KDE in the [Managing KDE Sources with Subversion guide](#)

Working With Other Documenters and Developers

One important and fun part of working on KDE is the community of other developers who you work with. The people you'll work with most often as a documentation writer are the documentation team, the quality team (if you're a new contributor) and the maintainer of the application that you're working on.

The documentation team is your main resource for help with doc writing and a central point of contact to ensure that everyone's work is co-ordinated. The main ways to contact the documentation team are via the [<kde-doc-english@kde.org>](mailto:kde-doc-english@kde.org) mailing list and on IRC in the `fkde-docs` channel on the server `irc.freenode.net`. If you plan to work on a particular application, please tell us, so that we can ensure that no-one else is working on it simultaneously, so that effort would be duplicated. Also, feel free to contact us with any problems or questions you might have about writing documentation. You don't need to feel like you're working entirely on your own ? there are plenty of people who are able to help.

The KDE Quality Team provides more broad support. If you have any general questions about KDE development, or how documentation fits into the wider KDE environment, the Quality Team mailing list is a good place to ask: [<kde-quality@kde.org>](mailto:kde-quality@kde.org). If you're not sure whether to ask a question on the `kde-quality` or `kde-doc-english` list, just pick one and ask. Many people who read one list read the other, and you'll be pointed to the appropriate list if necessary.

Working with programmers is a little less formal. The usual reason to contact a programmer is to ask about a feature or behavior of an application that you're documenting. To find the appropriate person to contact for a particular application, look in the Help->About *KAPP* menu item for the maintainer. If you can't find a maintainer, ask on [<kde-doc-english@kde.org>](mailto:kde-doc-english@kde.org) or [<kde-devel@kde.org>](mailto:kde-devel@kde.org). If asking on the `kde-devel` list, mention that you're writing the documentation for that application ? it helps to identify you to those reading a busy list. In general, programmers and other developers are happy to help, and willing to work with you, so don't feel afraid of asking them for information, and building up a working relationship.

Updating Documentation

With the pace of change of KDE applications, documentation can rapidly become out-of-sync with the application it is describing. To keep its value, documentation needs to be updated. Often this is simply a case of reading the existing documentation, and checking each description of an item against the latest version of the application. For example, are there new items in the menus that are not described in the documentation?

Sometimes, more extensive updates are needed. If new features of the application significantly change the way it works, then new sections of the documentation may be needed, or reorganization of the existing content might be necessary. In particularly severe cases, an entire rewrite might be necessary.

Licenses for KDE Documentation

KDE uses the FDL (Free Documentation License) for all documentation. This license has several variants, some of which place restrictions on how content is used in other contexts.

The specific terms we use are:

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts.

This is the only version of the license that is safe to use for documentation that is to be distributed with KDE.

The items that may differ in other uses of the FDL are as follows:

with no Invariant Sections

Invariant sections are, as you might expect, sections that *must not* be altered in any reproduction of the content. The reasoning behind this, is so nobody can make a subjective claim, and attribute it to you, by altering your words.

For instance, if you say 'Foo is a terrible piece of software' and the section is marked invariant, the developers of 'Foo' can not take your writing, change it to say 'Foo is a great piece of software' and still attribute that opinion to you.

For many people, this restriction is incompatible with the GPL and therefore some distributions choose not to include any user manuals that contain 'Invariant Sections'. Since they must be reproduced verbatim, this also means we are not able to reuse such content in our own manuals, without including this statement.

For this reason, 'Invariant Sections' are not permitted in documentation that is to be distributed with KDE, nor can we safely reuse content from other sources, if they include Invariant Sections.

It is not normally appropriate to write opinion pieces in KDE documentation. Such content should be restricted to your own website, or documentation that is not distributed with KDE, so the fact we outlaw 'Invariant Sections' in KDE documentation is not normally a problem. If you think you have a special case, please raise it with the documentation team, and understand that including such sections may prevent some distributions adding your manual (or the software itself) to their distribution.

with Front-Cover Texts *names of sections*, with Back-Cover Texts *names of sections*,

As with 'Invariant Sections', these are texts that may not be altered, and must be included in any reuse of any of the content. It also means we would have to alter our license to match that of the content we have reused. This leads to similar problems as that of the 'Invariant Sections'.

This one mainly comes up if we want to use FDL content found from other sources (for instance, books or websites.) In these cases, the best approach is to ask the authors to permit relicensing, and offer to include their front/back cover texts anyway, but without having to change our license terms.

Warning

The terms of the FDL as used by KDE documentation, are entirely GPL compatible, and do not restrict the reuse of the content. Any deviation from these terms, or any change in license could restrict distribution of your software or documentation, and should only be undertaken with full knowledge of the consequences, and with written permission of all copyright holders.

Using bugs.kde.org

Note how we use b.k.o (general to-do items). Also point to Carlos' guide on quality.k.o

The KDE bug tracking system, located at <http://bugs.kde.org>, is now part of the documentation team's toolkit. Issues with the KDE documentation can be filed in the ?docs? product of the bug tracker. Incorrect or outdated content, missing content, outdated screenshots and typos are all appropriate reasons to file bugs.

When filing bugs, especially for incorrect or outdated content, be specific about what's wrong. For example, if a certain page of a configuration dialog is incorrectly documented, say which page it is in the bug report. That way, someone fixing the bug can quickly find the appropriate part of the application and the documentation, and make the necessary changes with a minimum of effort.

For more information on using the KDE bug tracking system, see <http://quality.kde.org/develop/howto/howtobugs.php>.

Chapter 7. Leveraging your Newly Acquired Knowledge

After finishing documenting an application, you can leverage the knowledge you gained in the process and improve the application's level of quality in other areas. The Quality Team provide guides on how to perform many of these tasks.

- **Writing context help and configuration descriptions:** the handbook is not the only source of help available for KDE applications. Context help, or *whatsthis* provides invaluable support for users, and you will find it easy to write, especially after writing the handbook. Documenting configuration options available through the KConfig framework may require additional research, but configuration descriptions are often the only documentation available for configuration options.
- **Performing usability analysis and tests:** to document your applications, you probably tested most of the application functionality in a systematic way. Please take the time read the guide and report the usability issues and suggestions that appeared in the process.

- Writing guides and articles about the application: promotion is the key for a successful open source project, as widespread use means usually more probability of attracting prospect contributors, developers, documenters, translators, etc..

Chapter 8. Credits and License

This documentation is licensed under the terms of the [GNU Free Documentation License](#).

Appendix A. KDE DocBook Reference

Lauri Watts

Revision 1.00.00 (2004-08-22)

Copyright © 2000, 2001, 2002, 2003 Lauri Watts

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.1 or any later version published by the Free Software Foundation; with no Invariant Sections, with no Front-Cover Texts, and with no Back-Cover Texts. A copy of the license is included in [the section entitled "GNU Free Documentation License"](#).

A quick reference guide to the current KDE DocBook markup standards

Table of Contents

[General KDE markup style guide](#)

[Purpose of this document](#)

[Other reference material](#)

[The Prologue](#)

[book and the bookinfo section](#)

[Chapters and Sections](#)

[The linking elements](#)

[Lists](#)

[Tables](#)

[The GUI elements, menus, toolbars and shortcuts.](#)

[Describing actions and commands](#)

[Questions and Answers](#)

[Images and Examples](#)

[General markup \(not covered elsewhere\)](#)

[Admonitions: Tips, hints, and Warnings.](#)

[The synopsis elements](#)

[Markup for programming](#)

[Making Callouts](#)

[References, indexes, and glossaries](#)

[Making a glossary](#)

[Making an Index](#)

[Other Reference Sections](#)

[Tags we do not use](#)

[Alphabetical List of all elements](#)

[Credits and License](#)

General KDE markup style guide

- Format for readability, and content, not for a formatted document.

It is not your job or responsibility to make sure the final documentation looks good. If you use appropriate markup tags for the content of your documentation, the processing tools will ensure your document looks good. Do not substitute an inappropriate DocBook XML tag because you do not like the ?look? of the correct tag.

You should use white space to make the DocBook source more readable to the writer. Please do not indent unless it is absolutely necessary.

- Do what you can to ensure you turn in a *valid* DocBook file. The reviewers will correct any DocBook errors you create, but please try to reduce errors by checking your work before it is turned in. If you have the KDE tools installed, you can use the command **checkXML index.docbook** to check for syntax errors. No result from **checkXML** is a good result - it means there are no problems.
- Non-English words should be tagged with `<foreignphrase lang="de">Wort</foreignphrase>`.
- Underlining and CAPITALIZING entire words are leftovers from the days of typewriters. They are no longer appropriate for today's documents.
- Do not use quotation marks in your documentation. If you want a word to appear within quotation marks, simply enclose it between `quote` tags.

This software is provided `<quote>as is</quote>`.

- There are three different ?dashes? that are commonly found in documentation.
 - The hyphen combines two or more words into one. For example, ?mother-in-law?. The hyphen can be entered directly from the keyboard.
 - The en-dash is used to separate numbers/dates/etc.. For example, ?Sections 1?3 review basic concepts?. The en-dash can be encoded using `&ndash ;`.
 - The em-dash is used to separate sentences, or to show that something is missing. This is rarely used in technical documentation, but it can be used to show that one sentence is interrupted by another. The em-dash can be encoded using `&mdash ;`.
- When trying to decide between an ordered and unordered list, simply ask yourself the following question: ?Does the order of the listed items matter?? or ?If I change the order of the listed items, does that change the meaning of the list??. If you answer ?No? to either question, then an unordered

list is likely the logical choice.

- All `<chapter>` and `<sectN>` tags must have an `id`. The `id` must be in all lower case, and with dashes separating words. For example, `<sect1 id="how-to-obtain-kapp">`.
- All elements must have a full closing tag unless they are empty elements. Empty elements must still be closed with a `/`.

Incorrect: `<para>Blah blah/` or `<para>Blah blah</>`

Correct: `<para>Blah blah</para>`

- No attribute minimization.

Incorrect: `attribute=value` or `attribute='value'`

Correct: `attribute="value"`

- All entities must end with a semi-colon:

Incorrect: `%parameterentity` or `&generalentity`

Correct: `%parameterentity;` or `&generalentity;`

- Element GIs (the first word in a tag) must be written in lower case only.

Incorrect: `<MediaObject>` or `<MEDIAOBJECT>`

Correct: `<mediaobject>`

Entities are also case sensitive, and will result in validation errors if the case is wrong.

- Specify date and application's version in the format:

```
<date>2000-12-31</date>
<releaseinfo>1.02.03</releaseinfo>
```

The `<date>` is the date of the last update. The `<releaseinfo>` always matches the version number of the application that is described in the documentation (if any). A translated version of a documentation always has the same `<date>` and `<releaseinfo>` as the English original. Please respect this, it is the only way to manage efficiently both the writing and the translation processes.

- The list of entities for applications is maintained centrally. Entity names are the application name completely in lower case. In case the name you need does not exist yet, send a mail to `<kde-docbook@kde.org>` to have it added. You may add it in the prologue for validation purposes (in case it's new), but don't forget to remove it when you submit the document, because there should not be any 'extra' entities defined in the document prologue.
- For language-independent entities, use `kdelibs/kdoctools/customization/entities/general.entities` and for language-specific entities, use `kdelibs/kdoctools/customization/lang/user.entities`. Try to avoid clashes with existing KDE entities.

- The `en/user.entities` file should be updated keeping in mind that translation must be possible. Here is an example of how this translation can be managed:

Example A.1. Managing translatable entities

`&LMB;` is an entity which stands for "Left Mouse Button"

When translating to French for example, do not translate only the entity contents, please also translate the entity name to `&BGS;` (or `&bgs;`), to reflect the change in the initials:

```
<!ENTITY "LMB" "left mouse button"> becomes
<!ENTITY "BGS" "bouton gauche de la souris">
```

Languages that decline nouns like German and Russian can use something like the following:

```
<!ENTITY "LMT-n" "linke Maustaste">
<!ENTITY "LMT-d" "linken Maustaste">
```

- If you feel that some elements don't make fine enough a distinction, feel free to use the attribute `role` (but please tell the DocBook team, as otherwise you may find your document to be suddenly invalid).
- Use `<qandaset>` for FAQs, not an `<itemizedlist>`. Please split up a FAQ into several chapters or sections if it gets big. The HTML files get too big otherwise, which the users may not like.
- Abbreviations and acronyms should be marked up as well.

Use the DocBook tags `<abbrev>` and `<acronym>` respectively.

Please keep them apart: acronyms are things like GUI, KDE, GPL, while abbreviations are things like etc., i.e., e.g..

There are entities for the most common ones.

- Use `<glossterm>` or `<firstterm>` each time you introduce a technically significant new word.
- Keep in mind that the `??` sign is introduced by the shell, and is not part of an environment variable's name:

`ls -l $KDEDIR` is marked up as

```
<userinput><command>ls</command>
<option>-l</option>
<parameter>${envar}KDEDIR<envar></parameter>
</userinput>
```

`export $KDEDIR=/usr/local/kde` is marked up as:

```
<userinput>
<command>export</command>
<parameter>${<envar>KDEDIR</envar>=<filename>
/usr/local/kde</filename></parameter></userinput>
```

- Only use `<ulink>` for URL's and not for files, unlike `<A>` in HTML. Don't use it for email addresses either, they have their own element, `<email>`.
- The elements `<beginpage>` and `<bridgehead>` are disallowed and have been removed from the KDE customized DTD. (They are not meant for new technical documentation.) `<revisionhistory>` has been removed also: we are using CVS already.

Purpose of this document

The purpose of this document is to describe how markup has been standardized within KDE documentation only.

This document is *not* to be considered more authoritative than the DocBook documentation, including the O'Reilly Duck book. However, there are places where the KDE DTD is more restrictive than, or just differs from, the OASIS DTD, and these are noted in this document. In these cases, follow the instructions here.

Please read and make use of the other documentation available to you, which is much more comprehensive. This document is not intended to be more than a quick reference for KDE authors, to clarify how the DocBook XML elements are used within the KDE Documentation.

Other reference material

Please take a look at the following reference material, rather than relying on this document to answer all your questions.

The Duck book

The complete DocBook SGML (and now XML) reference. Available as a download in several formats, so you can keep a copy on your hard drive for reference. Also available for sale in hard copy - if you see yourself doing a lot of DocBook Authoring, you definitely ought to consider buying it.

The Crash Course to Docbook

A non-KDE specific crash course to marking up documentation. This is the starting point for all KDE documents, including the markup issues discussed here. Note that the current version is written for SGML, but the concepts are still correct for XML.

The KDE Documentation Template

Covers many things not mentioned here, including required and optional chapters, the preferred way to mark up the prologue and bookinfo sections, and how to deal with licensing and credits. It can be found in `kdelibs/kdoctools/template.docbook` in CVS.

DocBook-XML (in German)

A very nice book, in German only unfortunately, but comes highly recommended.

The Prologue

```
<?xml version="1.0" ?>
<!DOCTYPE book PUBLIC "-//KDE//DTD DocBook XML V4.2-Based Variant V1.1//EN" "dtd/kdex.dtd" [
  <!-- Define an entity for your application if it is not part of KDE
    CVS -->
  <!ENTITY kmyapplication "<application>KMyApp</application>">
  <!ENTITY kappname "&kmyapplication;"><!-- replace kmyapplication here
    do *not* replace kappname-->
  <!ENTITY package "kde-module"><!-- kdebase, kdedadmin, etc. Leave
    this unchanged if your
    application is not maintained in KDE CVS -->

  <!ENTITY % addindex "IGNORE">
  <!ENTITY % English "INCLUDE"> <!-- ONLY If you are writing non-English
    original documentation, change
    the language here -->
]>
```

In general, this needs minimal changing from the template. The items you *must* change are the entities ?kappname?, ?package?, and ?English?.

The entity ?kappname? looks like it's redundant (as the comment in the template notes), but it is important. This allows us to use one global text in all documents, and still refer to the specific application by its correct name. So it should be changed to refer to this new entity, but this time you should only change the part in quotes (?&kmyapplication;?) as follow:

Example A.2. Setting up the global ?kappname? entity

```
From:
<!ENTITY kappname "&kmyapplication;" -- this only *seems* redundant -->
To:
<!ENTITY kappname "&kate;" -- this only *seems* redundant -->
```

In short: change any occurrence of ?kmyapplication? to the real name of your application. Do *not* use ?kappname? or ?kapp? directly in a document yourself.

The entity ?package? is used similarly. It allows us to insert a single piece of ?boilerplate? text into every document, and have the correct package name inserted when the document is compiled. Use the cvs module name, in lower case, e.g. ?kdeedu? or ?kdebase?.

The entity %addindex; is a toggle. If set to ?INCLUDE? a document index will be automatically generated. It is normally set instead to ?IGNORE?, and should not be changed unless you really do want to generate an index. You can find out more about indexes in the section called ?References, indexes, and glossaries?.

Example A.3. A KDE User Manual Prolog

Here is an example of a completely set up prolog, as it normally looks. This is the prolog from the AMOR documentation

```
<?xml version="1.0" ?>
<!DOCTYPE book PUBLIC "-//KDE//DTD DocBook XML V4.2-Based Variant V1.1//EN" "dtd/kdex.dtd" [
  <!ENTITY kappname "&amor;">
  <!ENTITY package "kde toys">
  <!ENTITY % addindex "IGNORE">
  <!ENTITY % English "INCLUDE">
]>
```

Note

The entity ?English? should be changed to reflect your language, if you are either writing original documentation in another language, or you are translating a document. For KDE the original documentation should always be in English, so you should not need to change this when writing. For informational purposes, the currently supported languages are:

- Afrikaans
- British-English
- Bulgarian
- Catalan
- Czech
- Danish
- German
- Greek
- English
- Spanish
- Estonian
- Finnish
- Faroese
- French
- Hebrew
- Hungarian

- Indonesian
- Italian
- Japanese
- Dutch
- Norwegian (Note, this is only for compatibility, either Norwegian-Bokmal or Norwegian-Nynorsk should be used in preference.)
- Norwegian-Bokmal
- Norwegian-Nynorsk
- Polish
- Portuguese
- Brazilian-Portuguese
- Romanian
- Russian
- Slovak
- Slovenian
- Serbian
- Swedish
- Turkish
- Ukrainian
- Walloon
- Xhosa
- Continental-Chinese
- Traditional-Chinese

book and the bookinfo section

The `bookinfo` section is most easily prepared by copying the KDE template.

`<book lang="&language" >`

Contains the entire document. Most important thing to remember is the `lang` attribute, which must contain exactly `&language;`, and must not be changed. To set the language for the document, change the entity as described in the [prologue](#) section.

`<bookinfo>`

Wraps the `?meta?` information ? information about the document, not about the application it is documenting. Required in KDE documentation. No attributes.

`<authorgroup>`

Wraps the author information, and may also contain `<othercredit>` information. Required in KDE documentation. No attributes.

`<author>`

Required element in the header section of all KDE documentation. Use this element *only* for the author(s) of the document. Other contributors (developers, translators, and so on) should be credited in the `<othercredit>` section. No attributes.

`<personname>`

Used to wrap a person's name. You can use this directly in the text as well, but here it should be used to contain each author or contributor name.

`<firstname>`

The contributor's first name.

`<othername>`

If the author normally uses more than a first and surname, you can add further names here.

`<surname>`

The author's surname.

`<email>`

An email address for the maintainer of the document is required for KDE documentation. You do not have to use your primary private address, and you may be able to arrange for someone else (the developer perhaps) to receive the email regarding the document. In any case, there must be an address for users and translators to contact regarding errors and document bugs.

Note

In previous versions of DocBook, `<email>` could not be used directly inside `<author>`. Since DocBook XML V 4.2 (used by KDE for documents after KDE 3.1.x), this is possible, which simplifies this markup considerably.

In other contexts in the document, `<email>` is used to contain any email address, and is not used inside the `address` element.

```
<othercredit role="">
```

Similar to `author`, this is a wrapper around information describing other contributors to the document. Include here the contributor's name and email address as you do for the `author`. See the template for more details.

The `role` attribute is required, and can contain any one of the following:

- Translator
- Developer
- Reviewer
- Graphist
- Musician

The `othercredit` element also includes the `contrib` element.

```
<contrib>
```

The role this contributor played in the document or application preparation. This could contain something like:

- Developer
- *Deutsche Übersetzung*
- Reviewer
- *Traduction française*

```
<corpauthor>
```

This is used in very specific circumstances, where an organization (e.g. ?The KDE Team?) is being credited with authorship of a document. Authors writing about applications should not use this and should credit themselves. If you do find a need to use this, please be sure to include a maintainer's name and email address in the credits chapter of the document.

<copyright>

This is a wrapper for copyright information. `copyright` must contain these elements:

<year>

Add one `year` element for each year in which the document was changed or added to. Don't put more than one year in each tag, rather add more `year` elements, and use the 4 digit `?YYYY?` format.

<holder>

The usual full name of the copyright holder(s). If there is more than one copyright holder (the document was previously maintained by another person, or is written collaboratively), then add more `copyright` sections, rather than trying to fit multiple names in the one section.

Copyright is automatically held by the author of the document, but the `copyright` element is still required for all KDE documentation. None of the elements contained have any attributes.

Please do *not* add more names or years to existing `<holder>` or `<year>` elements. Add more, if they are required, or have multiple `copyright` sections.

<legalnotice>

This contains, of course, a legal notice. This is absolutely required for any KDE document. In the context of this section, it should contain the `&FDLNotice;` entity, which inserts some information into the document about the document's license (and *not* the license of the application you are describing.)

<date>

The date is very important. It is used not only by scripts for automatic processing of documentation, but is also central to revision control and co-ordination of translations. You must change the date if you have changed the *original* document, and you must *not* change the date if you are a translator. The format of the date is very important. It *must* be in the ISO, with `?literal?` delimiters, in the form `?yyyy-mm-dd?`. Please be extremely careful about this, and triple check it before you send in the document.

<releaseinfo>

This should match exactly the version of the *application* you are documenting. It should normally conform to the format `X.x.xx` (where `X` is a major version number and `x` are minor version numbers, however, you no longer have to pad the content to this length. That is to say, if the application has released version `?1.4?`, you may write `<releaseinfo>1.4</releaseinfo>`, and you do not need to make it `<releaseinfo>1.04.00</releaseinfo>`

This is *not* the version of the document. There are no attributes, and this element is required in KDE documentation.

<abstract>

In KDE Documentation, the abstract is required. It should be a short one- or two-sentence summary of the document. The abstract is not the place to put version or contact information, but it should say something about the application and its purpose. For example `?KFoo is a small fast network enabled foo generator, suitable for both beginner and advanced foo users.?`

The abstract is your chance to sum up the application in a small paragraph `? in KHelpCenter it shows up on the first page as your document is selected, and the abstract frequently shows up in the summary of your document in web searches. A short overview of the application you are writing about is very valuable in this situation, ?This is the KFoo handbook and describes KFoo 1.2.? on its own, is not.`

<keywordset>

A wrapper for a set of keywords suitable for search engines. Required for KDE Documentation, and there are no attributes. The `keywordset` should contain several `<keyword>`s.

<keyword>

Add one `<keyword>` inside the `<keywordset>` for each search term. You must include at a minimum the terms `?KDE?`, the name of the application you are documenting, and the name of the package it is found in, for example `?kdegames?.` The keywords should be in order from most general first (that is, KDE) through less general, to the most specific. Add two or three more relevant words that people might search with, e.g., for the application KWrite you might add `?editor?` and `?text?.` This is required for KDE Documentation, and there are no attributes.

<!-- TRANS:ROLES_OF_TRANSLATORS -->

This line is specific to KDE documentation. Although it's a comment, it is *absolutely* required in documents. It is used by the translation system as a placeholder for the translation teams to add their own role info. Translators should add more `othercredit` sections here as appropriate.

Example A.4. The bookinfo section from the KDE template

```
<bookinfo>
<title>The &kmyapplication; Handbook</title>

<authorgroup>
<author>
<!-- This is just put in as an example. For real documentation, please
define a general entity in entities/contributor.entities, e.g.
<!ENTITY George.N.Ugnacious "<personname><firstname>George</firstname><othername>N.</othername><surname>Ugnacious</surname></personname>">
<!ENTITY George.N.Ugnacious.mail "<email>gnu@kde.org</email>">
and use '&George.N.Ugnacious; &George.N.Ugnacious.mail;' in the author element.
-->
<personname>
<firstname>George</firstname>
<othername>N.</othername>
<surname>Ugnacious</surname>
</personname>
<email>gnu@kde.org</email>
</author>
</authorgroup>

<!-- TRANS:ROLES_OF_TRANSLATORS -->

<copyright>
<year>2002</year>
<holder>George N. Ugnacious</holder>
</copyright>
```

```

<!-- Translators: put here the copyright notice of the translation -->
<!-- Put here the FDL notice. Read the explanation in fdl-notice.docbook
and in the FDL itself on how to use it. -->
<legalnotice>&FDLNotice;</legalnotice>

<!-- Date and version information of the documentation
Don't forget to include this last date and this last revision number, we
need them for translation coordination !
Please respect the format of the date (YYYY-MM-DD) and of the version
(V.MM.LL), it could be used by automation scripts.
Do NOT change these in the translation. -->

<date>2003-01-10</date>
<releaseinfo>1.1.</releaseinfo>

<!-- Abstract about this handbook -->

<abstract>
<para>
&kmyapplication; is an application specially designed to do nothing you would
ever want.
</para>
</abstract>

<!-- This is a set of Keywords for indexing by search engines.
Please at least include KDE, the KDE package it is in, the name
of your application, and a few relevant keywords. -->

<keywordset>
<keyword>KDE</keyword>
<keyword>kdeutils</keyword>
<keyword>Kapp</keyword>
<keyword>nothing</keyword>
<keyword>nothing else</keyword>
</keywordset>

</bookinfo>

```

Chapters and Sections

```
<chapter id=" " >
```

Use chapters to break up the document into smaller chunks. A chapter break should occur when a major subject change happens. Use sections within the chapter when the subject changes, but you are still discussing a particular aspect of a larger subject.

For example, going from discussing how to use the application, to how to configure the application would be worthy of a new chapter. Moving from discussing how to specifically configure the application on SuSE, to how to specifically configure the application on Red Hat®, would be a new section in a larger ?Configuration? chapter.

Chapters must have an id. This is the only attribute used in KDE documentation. For KDE Documents, this id must be in lower case, and with a hyphen (-) to separate words. Please don't use spaces, underscores, or run the words together. For HTML generation, the chapter id and most <sect1> id's are used to name the separate HTML pages, so take care to make them sensible and descriptive. For translators, these id's should be translated, but you will need to take care to also translate references to the id's in <link> and <xref> elements in other parts of the document.

```
<title>
```

Titles are used in many places, but the most common is the Chapter and Section headings. Make sure to use sensible titles, as these will also be that chapter's (or section's) entry in the table of contents, so people will rely on these to find the part of the document they are interested in.

`<sect1 id=" ">, <sect2>, <sect3>, <sect4>, <sect5>`

Use sections to break chapters up into smaller pieces. Use similar criteria on where to divide them as you would for chapters.

Sections require a `<title>`. Sections are nested according to the number - a `<sect2>` can contain any number of `<sect3>`, which can contain `<sect4>`, but a `<sect2>` can't directly contain a `<sect4>`.

`<sect1>` requires an `id` attribute, and you can use `id`'s on the other section tags if you want to later link directly to them from other parts of the document. `id` is the only attribute used in KDE Documentation.

`<sect1info>, <sect2info>, <sect3info>, <sect4info>, <sect5info>`

The section info elements are rarely used in KDE Documentation. They are appropriate for documents where some smaller sections are contributed by third parties, or where the document covers multiple applications. The contents are more or less the same as those of the `<bookinfo>` section, although they tend to be briefer.

Please ensure if you use these elements that you add the translation placeholder comments as you do in the prolog.

`<appendix>`

The standard installation instructions for all applications are contained in an `<appendix>`, and are normally required for KDE documents. Although the installation instructions as found in the template are reasonably complete, and need no customization for most applications, authors are very strongly encouraged to expand on them. For example, links to web pages, where to find libraries, plugins, screenshots of the application in a particular configuration, or any other information you can think of.

If the application is only distributed with KDE, there is little use in repeating the same installation instructions for every manual. You may leave it out entirely, unless you have further information to add.

For other purposes, appendices are used infrequently in KDE Documentation. An appendix can be found, for example, in the KPPP document, containing such things as Hayes Modem commands. Only use an appendix if you think it's very necessary. In most cases, the information it would contain would be better moved to the main document. In the example of KPPP, this information is vital to a few people, but extremely uninteresting to the majority, so it was placed in an appendix.

The linking elements

`<link linkend=" ">`

The most common link. Use this to turn a word or phrase into a link to another part of the document. `linkend` is the only attribute we use.

`<ulink url="">`

A link that refers to a document using it's URI. Use this for websites and ftp sites, but not for email addresses, which have their own specific tag. Please do *not* use this to link to other documents on the local system.

`<anchor id="" />`

Marks a place in the document, which you can use to link to. Note that the `id` attribute on any other element where it is valid, will automatically generate an HTML anchor in generated HTML, so you do not need to duplicate these. Use anchors only when you need to jump into the middle of a longer page, for example, to a particular menu item, or to a particular option in a preference dialog.

Note

`<anchor />` is an empty element, and must be closed with a `/`.

`<xref linkend="" />`

A cross reference to another part of the document. Use this when you want to refer to the section without the name. This is one of very few unclosed elements allowed. `linkend` is the only attribute we currently use.

Note

`<xref />` is an empty element, and must be closed with a `/`.

`<email>`

Use this to enclose an email address. Don't add `?mailto:?` to the email address, and don't use `<ulink url="">` for email addresses. No attributes required.

Lists

`<listitem>`

`<listitem>` is the main building block of almost all the lists. It should always contain some other markup, usually a `<para>`

`<orderedlist>`

Use this type of list when the order of the items matters, but they are not a set of steps that are carried out to achieve something. A good example is a list of things in order of importance.

`<itemizedlist>`

Use an itemized list when the order of the items is not important.

`<variablelist>`

A list that has two sections for each entry. Examples: A menu item, and what the menu item does, An action, and its result, or a term and its definition. This is a very common type of list. (Almost this entire document is composed of variable lists.)

`<variablelist>` contains the following elements:

`<varlistentry>`

A `<varlistentry>` is a wrapper around each pair in the variable list.

`<term>`

To reuse the above examples, the `<term>` for each pair would be the menu item you are describing, the action, or the term you are defining. You can use the `id` attribute for this element, which is quite convenient in long lists such as a menu reference, enabling you to link directly to a particular menu item from another part of the document.

`<listitem>`

As described above the `<listitem>` is used inside a `<varlistentry>` to hold the second part of the pair: The result of choosing that menu item, for example, the consequences of an action, or the definition of the term.

`<procedure>`

Use a procedure list when you are listing a sequence of steps which are performed in a particular order.

A procedure contains only one tag:

`<step>`

A step is one of the sequence of events that make up a procedure.

`<substeps>`

A step can contain substeps

`<simplelist>`

A simple list is just that - a simple list, with no formatting required. A simple list can contain only one type of element:

`<member>`

Members of a simple list.

```
<segmentedlist>
```

A Segmented list is a very particular type of list. Use sparingly, as it's very difficult to get these right, and most content appropriate for a segmented list could just as well fit the table model.

Example A.5. A Segmented List

```
<segmentedlist>
<segtitle>Name</segtitle>
<segtitle>Occupation</segtitle>
<segtitle>Favorite Food</segtitle>
<seglistitem>
<seg>Tux</seg>
<seg>Linux Mascot</seg>
<seg>Herring</seg>
</seglistitem>
<seglistitem>
<seg>Konqui</seg>
<seg>The KDE Dragon</seg>
<seg>Gnomes</seg>
</seglistitem>
</segmentedlist>
```

Name: Tux

Occupation: Linux Mascot

Favorite Food: Herring

Name: Konqui

Occupation: The KDE Dragon

Favorite Food: Gnomes

The segmented list contains the following elements:

```
<segtitle>
```

The title each segment will have

```
<seglistitem>
```

A set of entries in the list

```
<seg>
```

The contents of the entries in the list. In each `<seglistitem>` there is one `<seg>` for each `<segtitle>`.

Tables

`<informaltable>`

This is the table type used most in KDE Documentation. Please be very sure that what you are marking up as a table, is actually tabular data, as in many cases a `<variablelist>` is more appropriate. Please do not use any of the presentation attributes to make tables "look nice". The only attribute currently allowed in KDE Documents is `pgwide`.

An `<informaltable>` must contain a `<tgroup cols=" ">` entry. Informal tables have no specific title, if you wish the table to be titled and to have an entry in the table of contents, you should use `<table>`. Do not use any attributes other than `pgwide` on tables or informal tables for KDE documentation.

`<table>`

A formal table with a title. Tables will have their own separate entry in the table of contents. Other than the addition of a title, they are marked up the same as an `<informaltable>`.

`<tgroup cols=" ">`

A `<tgroup>` is a required element in a table. The `cols` attribute is required, and should be completed with the number of columns the table is to hold. No other attributes used in KDE Documentation.

A `tgroup` must contain a `tbody`

`<tbody>`

A `tbody` is a required element in a table. There are no attributes. The `tbody` contains rows.

`<row>`

A `row` corresponds directly with the rows of the table. Rows contain `<entry>` tags, one for each column in the table, as specified by the `cols` attribute on the `<tgroup>` tag.

`<entry>`

The entry is the basic building block of a table. Each entry corresponds to one "data cell" in the table. There must be as many `<entry>` tags in each row as the `cols` attribute on the `<tgroup>` tag. There are no attributes used in KDE Documentation.

`<thead>`

`<thead>` can be used to create a heading row for the table. It must appear before the `tbody` element, and should normally contain one `row` and as many `entry` elements as the rest of the table.

<tfoot>

<tfoot> is not currently used in KDE Documentation. If you want to use it, please see the Duck book for information.

Example A.6. An <informaltable> template

```
<informaltable>
<tgroup cols="2">
<tbody>
<row>
<entry></entry>
<entry></entry>
</row>
</tbody>
</tgroup>
</informaltable>
```

Example A.7. A <table> template

```
<table>
<title></title>
<tgroup cols="2">
<tbody>
<row>
<entry></entry>
<entry></entry>
</row>
</tbody>
</tgroup>
</table>
```

The GUI elements, menus, toolbars and shortcuts.

<action>

The result of a user action. This does not need to be a complete sentence, or even more than a single word. For example, ?This button <action>closes the dialog</action>?. The main place you will find this in KDE Documentation is in the Menu and Command reference chapters of the manuals.

<guibutton>

The text on a button that you click on. Icons, Radio buttons and check boxes are not considered buttons in this sense.

<guiicon>

The name or description of an icon.

<guilabel>

The text of anything that is labelled on screen, and isn't a button, icon, menu, or menu item. For example, the name of a dialog box, the name of a tab in that dialog box, and the name of a label by a checkbox.

Take care that the text exactly matches the label on screen. If it has a `:` on the dialog box, put the `:` into your documentation. Match the capitalization. There is a script in the `kde-i18n` module called `check-gui-texts` which you can use to help check that your text matches exactly what is in the application. During translation, the translators can use this script to generate translations from their translations of the GUI itself, but this will only work if the English text matches precisely.

`<guimenu>`

The top level name of a menu (that is, the name you can see on the menu bar when the menu isn't open).

`<guimenuitem>`

The final item you select on the menu, that actually performs an action.

`<guisubmenu>`

A submenu. That is, a menu which has items both above and below it in the hierarchy.

`<keycap>`

A keycap is a key as it is labelled on your keyboard. **Home** is a keycap on a standard English keyboard. **Alt Gr** is a standard key on many European keyboards.

`<keycode>`

The internal identifier for a key on the keyboard. Used very infrequently, but you may find need for it, for example when describing entries in rc files.

`<keysym>`

Right arrow is the `<keysym>` for the `<keycap>` that looks like `->`. Please note this is a KDE specific use of `<keysym>`, and does not precisely follow the examples in the Duck Book.

`<menuchoice>`

A menuchoice describes a menu entry. You should use `<menuchoice>` anywhere you are describing how to reach a menu item. In normal text, there are no particular requirements. In a menu reference, the `<menuchoice>` should also contain a `<shortcut>` element describing the keyboard shortcut, and the contents should also be marked up with `<accel>` as appropriate.

`<shortcut>`

A key combination that is a shortcut for a menu item. This is *only* used inside `<menuchoice>` and contains `<keycombo>` or `<keycap>` that is defined as the keyboard shortcut in the menu. In the markup, it appears before the actual menu entries inside the `<menuchoice>`. You do not need to describe the shortcut every time the menu item is mentioned in the text, although it may be appropriate to do so on some occasions.

`<mousebutton>`

The normal name of a mouse button. It will be normally be one of:

- `<mousebutton>left</mousebutton>` or the entity `&LMB;`
- `<mousebutton>middle</mousebutton>` or the entity `&MMB;`
- `<mousebutton>right</mousebutton>` or the entity `&RMB;`
- `<mousebutton>wheel</mousebutton>`

Wheel is used only in specific instructions for applications that support it, of course.

Use the entities where possible, they are a lot less typing and are simple to remember (which is why we have provided them.) If you are translating, check with your team leader, as the entities above are *not* translated, but you may have your own language specific ones to use in their place.

`<keycombo action="">`

A keycombo is a sequence or combination of keypresses that are performed together. A keycombo can contain `<keycap>`, `<keySYM>` or `<mousebutton>`, or any combination of these, in any order.

It is normal to have them in the order `modifier`, `Alpha-numeric`, `Mouse`. That is, **Ctrl-A**, not **A-Ctrl**, unless pressing **A** then **Ctrl** actually is the shortcut.

Keycombo requires an `action` attribute, describing exactly how the keys (or mouse buttons) are combined. The choices are:

- `Click`
- `Double-Click`
- `Other`
- `Press`
- `Seq`
- `Simul`

You will most likely need to use `Seq` (for a sequence of keys that are pressed one after the other), or `Simul` for a combination of keys that are pressed at the same time.

`<accel>`

The accelerator key that can be used to access a GUI menu without a mouse. This is indicated in the menu by an underlined letter. Although we previously used this in the menu references, we have since decided not to, the maintenance is too high, and it causes an enormous amount of work during translation.

Example A.8. An example from a menu reference entry

```
<varlistentry>
<term><menuchoice>
<shortcut>
<keycombo action="simul">&Ctrl;
<keycap>C</keycap></keycombo>
</shortcut>
<guimenu>Edit</guimenu>
<guimenuitem>Copy</guimenuitem>
</menuchoice></term>
<listitem><para><action>Copy the selected text</action> to the
clipboard</para></listitem>
</varlistentry>

<varlistentry>
<term><menuchoice>
<shortcut><keycombo action="simul">&Ctrl;
<keycap>V</keycap>
</keycombo></shortcut>
<guimenu>Edit</guimenu>
<guimenuitem>Paste</guimenuitem>
</menuchoice></term>
<listitem><para><action>Paste</action> the contents of
the clipboard at the cursor.</para></listitem>
</varlistentry>
```

Please note, this is very complicated markup, and until you have written a few it's very hard to follow, but it does get much easier with practise! Although indenting is discouraged in general, this is one place where you might want to use some indenting and white space to make it clearer while writing, at least when you are beginning. There are also no rules as to when you must start a new line for a new element, so format the markup to suit your own taste while you are writing, if that makes it easier for you to follow.

Describing actions and commands

`<replaceable>`

Use this for placeholder or sample text, that a user would not actually type, but would instead replace with the correct text for their environment. For example, Edit the file `<filename><replaceable>/usr/local/foo/bar</replaceable></filename>`, because it may already be established that `/usr/local` is only the default location of this file, and the user may have it installed to e.g. `/opt/` instead.

`<application>`

Use this to mark up the name of any software program mentioned in the text. Don't use this to mark up the actual command issued to execute the application. For example, `<application>Kate</application>` is the name of the editor, but `<command>kate</command>` is the name of the command that starts the Kate application.

Note

All KDE applications, and several non-KDE but very common applications, are provided as entities.

For the KDE applications, using the entities will save you much typing, and will ensure that applications are always referred to with their correct name across all documentation. The entity is always the application's executable name, in lower case, e.g. `&kcontrol;`, `&konqueror;` or `&kmail;`.

For non-KDE applications, one of the major reasons to use the entities is that there are legal implications, so far as we are required to acknowledge trademarks and copyrights held by others outside our organisation. You will find in (TODO: write this section!) a list containing a list of the more common non-KDE application entities.

`<interface>`

Catch all element for gui interface items that do not have a more specific tag. You can use this to markup things like the `?View pane?` in `KHelpCenter`, or the `?Board?` in `KJumpingCube`.

`<userinput>`

Any text that the user must type, including commands and data entry.

`<screen>`

Used to represent the computer screen (usually to represent a terminal or console.) Text contained in `<screen>` is considered to be literal text ? line breaks and white space are honored and it will be rendered with a mono-spaced font. Don't use screen when what you really want is an example, or an informal example.

`<command>`

Text the user enters to instruct the computer or an application to do something. `ls -al` is a command (it's also `userinput`, and has options.) / `join #kde` in an irc client is a command (and again, is `userinput`.)

Commands are not `userinput` when you are not expecting the user to actually type them, for example in the sentence `?The output from the ls command should show you...?`, the text `?ls?` is a command, but is not `userinput` in this context.

Applications not marked up with the `<application>` tag are also considered commands, for example, **gcc**, **automake** and **autoconf**.

`<prompt>`

The prompt at which a user types input. For most KDE Documentation, this has been standardised as `<prompt>&percent ; </prompt>` (which is the % character).

`<option>`

An optional parameter to a command. Since we write about UNIX® platforms, an option on the commandline is almost always indicated by a `?-?`, but there are exceptions (e.g., **tar `zxvf filename.tar.gz`** or **ps `ax`**, which are marked up as `<userinput><command>tar</command><option>zxvf</option><replaceable>filename.tar.gz</replaceable></userinput>` and `<userinput><command>ps</command><option>ax</option></userinput>` respectively.

`<envvar>`

An environment variable. Note that the variable indicator (usually \$ for UNIX®) is not part of the name of the environment variable, so it is correct to do this: `$<envvar>KDEDIR</envvar>`. There are no attributes in use in KDE Documentation.

`<errorcode>`

A (usually numeric, but not always) error code. SIGSEGV is an errorcode, as is 404 as you might receive when you are web browsing.

`<errorname>`

The actual text of an error message - to reuse the 404 example, the `<errorname>` might be Page not found.

`<errortype>`

The type of error, e.g. fatal or recoverable.

`<filename>`

Use `<filename>` for all occurrences of file names including:

- Directory names ? with the attribute `class="directory"`
- Paths
- File names
- File name placeholders (which should also be tagged with `<replaceable>`)

Do not use `<filename>` for file fragments or extensions (i.e. *.tgz which should instead be marked up as `<literal role="extension">`).

`<symbol>`

Symbols are things that are replaced by the computer when they are processed. It's difficult to say when things are a symbol and when they are not - if there is a more specific element to use (e.g. `<envar>` or `<constant>`) then you should use that instead.

Questions and Answers

`<qandaset>`

A set of questions and answers, suitable for a FAQ. `<qandaset>` must contain `<qandaentry>`.

`<qandaentry>`

Each question and answer pair is a `<qandaentry>`.

`<question>`

The question being asked. It must be inside a `<qandaentry>`, and it must have a matching answer.

`<answer>`

The answer to the matching question in the same `<qandaset>`.

Example A.9. `<qandaset>` Template

```
<qandaset>
<qandaentry>
<question>
</question>
<answer>
</answer>
</qandaentry>
</qandaset>
```

Images and Examples

`<screenshot>`

Wrapper around screenshots. Use this when you are including a screenshot in your document.

`<screeninfo>`

Screeninfo is a description of the screenshot. It's common (but not required) to reuse this text in the `textobject` element, as it saves translation time.

`<mediaobject>` and `<inlinemediaobject>`

Use `inlinemediaobject` to insert an inline image (that is, one that is inside a paragraph of text, or is the only item in a table entry). Use `mediaobject` for all other images. If the image is a screenshot, the `mediaobject` should be wrapped with a `screenshot` element. `mediaobjects` contain the following items:

`<imageobject>`

`Imageobject` contains information about one specific image. DocBook allows you to add more than one `imageobject`, in order to provide alternatives if the user is unable to see the preferred image. We don't currently use this functionality in KDE Documentation, but may do at some time in the future.

`<imagedata fileref=" " format=" " />`

This element holds the actual image reference. The `fileref` indicates the location of the image. You should always keep images in the same directory as the document itself, so you need only put the filename into the `fileref` attribute. The `format` indicates the type of image you are including. For KDE this should be `PNG`. Do *not* use `gif` format images for KDE documents.

This is one of few `?empty?` elements in use in KDE Documentation. This means there is no `</imagedata>`, but you should *always* close the element as shown above, with a final `?/?`.

Keep the images in the same directory as your `index.docbook`, don't create a separate directory to store them in.

`<textobject>`

Encloses the text part of a screenshot, which for KDE Documentation means it contains a `<phrase>` element.

`<phrase>`

A short descriptive phrase about the image contents, this element is contained in the `<textobject>` element.

`<caption>`

If you want the image to have a caption when displayed, you can add this. It's not required for KDE documents, but recommended, especially if there are several images near each other and there could be confusion as to which you are referring in the text.

`<informalexample>`

Use this element to enclose any informal examples you use in your document. There are no attributes. An informal example can contain almost any markup, so feel free to use them liberally. They should generally not be part of a paragraph.

<example>

An example is a more formal example, which has a title and an entry in the table of contents. Use sparingly, because having a hundred examples listed in the contents of a 5 page document lessens their usefulness. However, don't hesitate to use when you think it's necessary.

I've used them in this document to make it easy to quickly go to the small `?template?` examples for complex markup, because you can find them directly from the table of contents. Less difficult examples in this document have `<informalexample>` instead. Use your best judgement. As with `<informalexample>`, they can contain almost any markup.

Example A.10. A screenshot example

```
<screenshot>
<screeninfo>An example image</screeninfo>
<mediaobject>
<imageobject>
<imagedata fileref="example.png" format="PNG" />
</imageobject>
<textobject>
<phrase>An example image</phrase>
</textobject>
</mediaobject>
</screenshot>
```

General markup (not covered elsewhere)

<abbrev>

Abbreviations are shortened forms of longer words.

Abbreviations are not normally pronounced in speech. Examples are e.g. and i.e.. This is a KDE specific distinction, please stick to it.

<acronym>

Acronyms are shortened forms of words or phrases, often made up of the initials of the words in a phrase. Acronyms are normally pronounced in speech as well as written. Examples are GUI and KDE. As with `<abbrev>`, this is a KDE specific distinction.

<attribution>

If you use `<quote>` or `<blockquote>`, the source of the quote (that is, *who* you are quoting) should be cited with this tag.

<blockquote>

Use this when you want to quote a passage of text that should be set off from the main text, for example, an entire paragraph from a book or other source. Use `<quote>` to quote a passage of text that is not to be set off, for example a short sentence or comment from another person. Use both of

them as little as you can, there are copyright issues to quoting from other works inside KDE Documentation.

`<emphasis>`

Use this to emphasise text. Don't use it to mark up file names, commands, or anything else. Use it where you might type in all caps in an email, for emphasis of one word or short phrase, and try not to use it too much. Emphasis loses its power when over used.

`<computeroutput>`

Text the user can see on the computer screen. For example, a listing of a directory as produced after the command `ls` would be `computeroutput`.

`<epigraph>`

A short quote or saying at, sometimes used at the beginning of a chapter as an introduction. Use sparingly, no attributes used by KDE.

`<equation>`

Equation is used if you need to mark up a mathematical equation. You are unlikely to need to use this in KDE Documents.

`<hardware>`

Used when referring to a piece of computer hardware, e.g. Floppy Drive or Monitor.

`<lineannotation>`

A comment, for example in a `<programlisting>`. This is *not* for comments contained in the text, it is for comments by the author (you) *about* the text.

`<literal>`

In KDE Documentation, this is markup of last resort (or ?the least of all evils?) Use it only for things that must be marked up, but have no appropriate tag, and preferably only for the following things (already decided on:)

- `<literal role="extension">*.tar.gz</literal>`

`<literallayout>`

Use very sparingly, when it is absolutely vital that some text is presented exactly as it appears, including white space and line breaks. There is almost always a better tag to use than this (screen and computeroutput together, or even a screenshot).

`<markup>`

Use to wrap markup examples, for text that should be represented literally. Examples are this document, and documents that have HTML markup included literally in them. Other than meta-documentation like this, you probably won't have much need for `markup`.

`<optional>`

Optional information, usually in user input. Not used to date in KDE Documentation, but it may be appropriate in some circumstances.

`<para>`

A paragraph. This is the most common tag. You do not need to enclose lists, tables, or other markup with `<para>`. Sometimes however, you might want to do so, especially with `<screen>` and some types of lists, when they actually are still part of the current paragraph.

`<quote>`

Use when you are quoting something or someone, inside a sentence. Also use if you want a word or phrase to be "enclosed in quotes" like this.

`<trademark class=" ">`

Used to denote that a word is a trademark. There is the optional attribute `class` which should contain one of the following, if appropriate:

- `copyright`
- `registered`
- `service`
- `trade`

If there is no `class=" "` attribute, `?trade?` is assumed.

We have provided entities, marked up appropriately, for very commonly met trademarks, including Qt? (`&Qt;`), UNIX® (`&UNIX;`), Linux® (`&Linux;`) and many more.

`<sgmltag>`

An SGML tag. This includes XML and XHTML tags. Use this for marking up individual components, but use `<markup>` when you need to display a block of markup.

`sgmltag` will generate the correct markup characters for you, based on the `class` attribute.

Attribute values available:

- `attvalue`, for the contents of an attribute.
- `attribute`, for attributes.
- `element`, for element names.
- `endtag`, for closing tags (e.g. `</para>`).
- `emptytag`, for tags which are 'empty?', such as `
` in XHTML.
- `genentity`, for markup up general entities. For example, ` ` in XHTML.
- `numcharref`, to mark up a numbered character reference. ` `, for example, could also be referred to as ` `.
- `paramentity`. You are unlikely to need this for any KDE documentation.
- `pi`. Note this is an SGML PI, not an XML one. You are very unlikely to need this for any KDE documentation.
- `xmlpi`. An XML processing instruction, such as
- `starttag`. An opening tag, such as `<para>`. Most of this document is marked up this way.
- `sgmlcomment`.

`<superscript>`

Superscript as in x^2 . Unlikely to be required in most KDE Documentation.

`<msgtext>`

The actual text of an informational message. Use `<errorname>` for error messages.

`<subscript>`

Used to create things like H_2O . Unlikely to be found in most KDE Documents.

`<foreignphrase lang=" ">`

Use this any time you need to use text in a language different than the main language of the document. This should be rare, but may occur especially in credits information. The `lang` attribute should contain the normal two letter designation of the language. Please be careful with these, the *Country* and *Language* codes are sometimes different, e.g. `?se?` is the country code for Sweden, but the language code is `?sv?`. Using `?uk?` for British English would give you possibly unexpected results, as this is actually the language code for Ukrainian.

Admonitions: Tips, hints, and Warnings.

Admonitions are set off from the main body of the text. Use these sparingly, as they disturb the flow of the writing, but don't be afraid to use them where necessary. Just make sure they *are* necessary when you do use them.

We have settled on a preliminary order of importance for these elements, which differs from that explained in the Duck Book. Note that this particular order is for KDE Documentation only, and use your own judgement which is the most appropriate element if your situation differs from those outlined.

`<warning>`

Use warning when data loss could occur if you follow the procedure being described.

`<caution>`

A note of caution. Use this for example when the reader may lose easily recovered or replaceable information (e.g. user settings), or when they could cause data loss if they don't correctly follow the procedure being outlined.

`<important>`

When there is no danger of data loss, but you wish to make clear to the reader a consequence that isn't immediately obvious (e.g. when changing the font for one instance of a program also changes the default setting, and this isn't clear from the GUI.)

`<note>`

Information the user should be aware of, but is peripheral to the actual task being described.

`<tip>`

When you're giving a hint to make things easier or more productive for the reader.

`<footnote id=" ">`

Use very sparingly for things that really are footnotes. An example might be to note that the situation being described will be changing at some currently unknown future time. Most footnotes would better be marked up as notes, or tips.

`<footnoteref linkend=" ">`

You can refer to a footnote more than once, by using this element to refer to it's unique id. The footnote does not need to be in the same chapter. Use this very sparingly.

The synopsis elements

<cmdsynopsis>

Example A.11. How to markup a command synopsis

```
<cmdsynopsis>
<command>more</command>
<group choice="opt"><option>-d</option>
<option>l</option><option>f</option>
<option>p</option><option>c</option>
<option>s</option><option>u</option>
</group>
<arg>-num</arg>
<arg>+ / pattern</arg>
<arg>+ linenum</arg>
<arg rep="repeat"><replaceable>file</replaceable></arg>
</cmdsynopsis>
```

This should generate:

more [-dlfp`csu`] [-num] [+ / pattern] [+ linenum] [*file*...]

There are several very nice examples in the Duck book at www.docbook.org

<funcsynopsis>

Example A.12. How to markup a function synopsis

```
<funcsynopsis>
<funcprototype>
<funcdef>void <function>setFile</function></funcdef>
<paramdef>QString <parameter>file</parameter></paramdef>
</funcprototype>
</funcsynopsis>

<funcsynopsis>
<funcprototype>
<funcdef>void <function>setAutoSize</function></funcdef>
<paramdef>bool <parameter><replaceable>val</replaceable></parameter></paramdef>
</funcprototype>
</funcsynopsis>

<funcsynopsis>
<funcprototype>
<funcdef>QString <function>getVideoCodec</function></funcdef><void/>
</funcprototype>
</funcsynopsis>
```

These would generate the following, respectively.

```
void setFile(file);
QString file;

void setAutoResize(val);
bool val;

QString getVideoCodec();
```

A function synopsis can contain the following:

<funcprototype>

Contains a prototype of the function. It can contain <void>, <varargs>, <paramdef> or most commonly, a <funcdef> which actually defines the function.

<funcdef>

A function and its return type.

<funcparams>

Contains the list of parameters for the function.

<paramdef>

Information about the parameters of a function.

<void>

An empty element in a function indicating there are no arguments.

<varargs>

An empty element in a function indicating there are multiple arguments, without specifically listing them. This is generally represented with an ellipsis (...). For example `int max(...);`

<functsynopsisinfo>

Not used in KDE documentation.

<arg>

Used inside <cmdsynopsis>. Since most KDE applications are GUI only, you won't see this very often. See the entry for <cmdsynopsis> for a full explanation and example.

<group>

Group

<sbr>

sbr

<synopfragment>

synopfragment

<modifier>

A modifier modifies a class, field, or method synopsis. Examples are the words ?public?, ?private? or ?virtual?

<fieldsynopsis>

A field synopsis.

Markup for programming

For formally marking up code examples or making a synopsis, you should study the Duck Book and the [Synopsis](#) chapter. The elements described below are mainly for marking up of pieces of source code that appear in the running text. Remember that KDE and KDE applications are written almost exclusively in C++, so our usage may differ in places from the examples in the Duck book, which may be describing other programming languages.

To Developers reading this, remember most of the authors who may be documenting your work are unfamiliar with source code, and many of them like it that way. Therefore, the explanations here are more concerned with how to tell things apart than what they are for, and may make you cringe.

To everyone reading this, this section is very much under construction so to speak. If you already need to use this markup, you can ask questions on the kde-docbook mailing list, which is the most likely place to get correct and up to date answers.

<classname>

Used to identify the name of a class in a programming language. In KDE Documentation, you won't see this much in the user documentation, except for those applications which contain an API reference chapter, and occasionally in others. You will find it used a lot in the KDevelop documentation.

For non-programmers, as we're almost exclusively discussing KDE applications written in C++ and using Qt?, classnames are fairly easy to distinguish: They start with a capital Q or K, and are usually one word only, in the form of KApplication or QListBox.

<function>, <methodname>

A function or ?subroutine?. In C++, a function generally looks something like this: `foo() ;`. The semi-colon may not always be present and there may or may not be content inside the braces.

If you see things that have the form `Kfoo::bar()` these are not just functions, but also methods, so you would use the `<methodname>` for these.

Constructors are methods where the parts before and after the `::` are the same, e.g.

`Kfoo::Kfoo()`. Destructors look like Constructors, but have a `~` after the `::` e.g.

`Kfoo::~~Kfoo()`. The same things apply as with functions and methods: there may or may not be a `;` at the end, and there may or may not be content inside the braces of a constructor (there is never content for a destructor).

These are normally marked up as `<methodname>`, but if you need to make a synopsis of a method, there are specific elements available: `<constructorsynopsis>` and `<destructorsynopsis>`

To recap:

Function

`foo()`

Methodname

`Kfoo::bar()`

Constructor

`Kfoo::Kfoo()` These are methods in ordinary text, but when making a synopsis, have a more specific tag to use.

Destructor

`Kfoo::~~Kfoo()` These are methods in ordinary text, but when making a synopsis, have a more specific tag to use.

Sometimes you really can't tell the difference, especially when they are being mentioned in passing in the text. Also, programmers tend to shorten and make shortcuts when referring to snippets of source. If it's very unclear what something is, mark it up with `<function>` and ask the developer.

Tip

Asking a developer "What is foo?" will likely result in a two page explanation of a finer point of C++ programming, which, if you could understand it, you wouldn't have needed to ask the question in the first place. It saves everyone a lot of time and frustration if you word the question "Out of function, method, constructor and destructor, which is the best fit for foo?".

<varname>

The name of a variable.

<returnvalue>

The value returned by a function.

<token>

A token is a placeholder, something that is replaced by an actual value during processing. (I need to come up with a useful example for a token)

<constant>

A constant. In the snippet:

```
enum MyType { Red = 0, Green, Blue, Yellow };
```

Red, Green, Blue and Yellow should be marked up as <constant>

<type>

Used to classify a value. In the snippet:

```
enum MyType { Red = 0, Green, Blue, Yellow };
```

MyType is a <type>

<programlisting>

Use this to wrap any source code examples in your document. You don't need to use this for short snippets that are inline in the text, but you should use it for any examples longer than a line or two, or that are a separate block of text.

<structname>,<structfield>

Not used in KDE Documentation, primarily because they are rare in KDE source code, and are almost certainly never going to require marking up.

<parameter>

Parameters can be used for commandlines as well as for code samples.

<classsynopsis>

A class synopsis

```
DCOPStub {? not sure about what goes here ? enum Status(CallSucceeded, CallFailed);}
<initializer>
```

An initializer

```
<exceptionname>
```

An exception name

Making Callouts

Callouts are difficult, so they have their own chapter. Use callouts when you want to refer from text to specific parts of an image, programlisting, or synopsis. Using callouts with graphics is currently unused, and is somewhat problematic, so they will not (yet) be described here.

```
<calloutlist>
```

A list element that contains the callouts themselves. That is, a list of the explanations that belong to the indicated areas in the item being explained.

```
<callout arearefs="">
```

The actual explanation or description of the called out area or line. The `arearefs` attribute should contain the id of the appropriate callout you are referring to.

```
<programlistingco> and <screenco>
```

Callouts applied to a `programlisting` or a screen element. Although they look more difficult than just embedding the callouts directly in the text, they really aren't too hard. The `programlistingco` contains one `areaspec`, and one `programlisting`. The `screenco` contains one `areaspec` and one screen element. The `programlisting` and screen elements are exactly as you would normally have.

```
<areaspec>
```

The `areaspec` contains a list of area elements, each of which describes one single callout.

```
<area coords="" id="" />
```

The `area` is another of the very few empty elements, so there is no `</area>`. The `id` attribute should contain a unique name for the item. The `coords` contains a pair of numbers which indicate first the line and then the column where the `co` should appear. The line and column refer to the position in relation to the container element, *not the entire document!*. That is, in a `screenco`, the line and column numbers refer to the line *within the screen element*.

Example A.13. Marking up callouts with `<screenco>`.

```

<screenco>
  <areaspec>
    <area coords="2 65" id="currentdir"/>
    <area coords="3 65" id="updir"/>
    <area coords="4 75" id="hiddenfile"/>
    <area coords="10 75" id="backupfile"/>
    <area coords="13 70" id="hiddendir"/>

<screen>
total 864
drwx-----  8 vampyr  vampyr      4096 Oct  2 18:01 ./
drwxr-xr-x  13 root    root        4096 Oct  1 16:32 ../
-rw-----  1 vampyr  vampyr        32 Sep  2 14:21 .MCOP-random-seed
-rw-----  1 vampyr  vampyr         0 Sep  2 14:42 .Xauthority
-rw-r--r--  1 vampyr  vampyr     1899 Aug  6 19:32 .Xdefaults
-rw-----  1 vampyr  vampyr     261 Sep 29 22:59 .bash_history
-rw-r--r--  1 vampyr  vampyr      24 Aug  6 19:32 .bash_logout
-rw-r--r--  1 vampyr  vampyr     285 Aug  6 19:34 .bash_profile
-rw-r--r--  1 root    root        230 Aug  6 19:32 .bash_profile~
-rw-r--r--  1 vampyr  vampyr     559 Aug  6 19:32 .bashrc
-rw-r--r--  1 vampyr  vampyr    4044 Aug  6 19:32 .emacs
drwxr-xr-x  7 vampyr  vampyr     4096 Sep 29 17:31 .kde/
</screen>
</screenco>
<calloutlist>
<callout arearefs="currentdir1"><para>The current directory.</para>
</callout>
<callout arearefs="updir1">
<para>One directory up in the tree.</para>
</callout>
<callout arearefs="hiddenfile1">
<para>A hidden file, indicated by the . beginning the name.</para>
</callout>
<callout arearefs="backupfile1">
<para>A backup or temporary file, indicated by the ~ ending the name.</para>
</callout>
<callout arearefs="hiddendir1">
<para>A hidden directory, which, like a hidden file, is indicated by the . at
the start of the name.</para>
</callout>
</calloutlist>

```

All this markup above, while it looks complicated is really quite simple if you study it closely. It would generate the following:

```

total 864
drwx-----  8 vampyr  vampyr      4096 Oct  2 18:01 ./
drwxr-xr-x  13 root    root        4096 Oct  1 16:32 ../
-rw-----  1 vampyr  vampyr        32 Sep  2 14:21 .MCOP-random-seed
-rw-----  1 vampyr  vampyr         0 Sep  2 14:42 .Xauthority
-rw-r--r--  1 vampyr  vampyr     1899 Aug  6 19:32 .Xdefaults
-rw-----  1 vampyr  vampyr     261 Sep 29 22:59 .bash_history
-rw-r--r--  1 vampyr  vampyr      24 Aug  6 19:32 .bash_logout
-rw-r--r--  1 vampyr  vampyr     285 Aug  6 19:34 .bash_profile

```

```

-rw-r--r-- 1 root    root      230 Aug  6 19:32 .bash_profile~
-rw-r--r-- 1 vampyr  vampyr    559 Aug  6 19:32 .bashrc
-rw-r--r-- 1 vampyr  vampyr   4044 Aug  6 19:32 .emacs
drwxr-xr-x 7 vampyr  vampyr   4096 Sep 29 17:31 .kde/

```

- ❶ The current directory.
- ❷ One directory up in the tree.
- ❸ A hidden file, indicated by the . beginning the name.
- ❹ A backup or temporary file, indicated by the ~ ending the name.
- ❺ A hidden directory, which, like a hidden file, is indicated by the . at the start of the name.

<CO>

Indicates where a callout is. For KDE HTML documentation, a small numbered graphic will be placed here, and also at the location of the explanation. These numbered graphics are links between the two places. It is entirely possible to embed the callout elements directly in the text you are describing, and this is perhaps the easiest way to do it. It isn't the most specific, but working out the line coordinates to use the more precise elements is difficult, so this way is acceptable for now.

Example A.14. Marking up callouts by embedding directly in text

```

<screen>
drwxr-xr-x 3 vampyr  vampyr    4096 Aug  6 19:32 .kde2/
lrwxrwxrwx 1 vampyr  vampyr     15 Sep  3 19:46
.kdeinit-whiterabbit.magicians.org->0 -> /tmp/.kinV4m2iI= <co id="symlink"/>
-rw-r--r-- 1 vampyr  vampyr   2096 Aug  6 19:32 .kderc
-r----- 1 vampyr  vampyr     21 Sep  2 14:21 .kxmlrpcd
-rw-r--r-- 1 vampyr  vampyr    185 Aug  6 19:32 .mailcap
-rw----- 1 vampyr  vampyr     31 Sep  2 14:21 .mcoprc
drwxr-xr-x 4 vampyr  vampyr   4096 Aug  6 19:32 .netscape/
-rw----- 1 vampyr  vampyr   777947 Sep  2 14:42 .xsession-errors
drwxr-xr-x 5 vampyr  vampyr    4096 Sep  2 14:42 Desktop/ <co id="dir"/>
drwx----- 2 vampyr  vampyr    4096 Aug  6 19:32 tmp/
-rw-r--r-- 1 vampyr  vampyr   3836 Oct 13 16:44 notes.txt <co id="file"/>
</screen>

```

```

<calloutlist>
<callout arearefs="symlink">
<para>A symbolic link, indicated by the ->, and showing the location it is
linked to.</para>
</callout>
<callout arearefs="dir">
<para>An ordinary directory.</para>
</callout>
<callout arearefs="file">
<para>An ordinary file.</para>
</callout>
</calloutlist>

```

Again it's really not as hard as it looks on first glance. This markup would generate the following:

```
drwxr-xr-x 3 vampyr vampyr 4096 Aug 6 19:32 .kde2/
lrwxrwxrwx 1 vampyr vampyr 15 Sep 3 19:46
.kdeinit-whiterabbit.magicians.org-:0 -> /tmp/.kinV4m2iI= ❶ -rw-r--r-- 1 vampyr vampyr 2096 Aug 6 19:32 .kderc
-r----- 1 vampyr vampyr 21 Sep 2 14:21 .kxmlrpcd
-rw-r--r-- 1 vampyr vampyr 185 Aug 6 19:32 .mailcap
-rw----- 1 vampyr vampyr 31 Sep 2 14:21 .mcoprc
drwxr-xr-x 4 vampyr vampyr 4096 Aug 6 19:32 .netscape/
-rw----- 1 vampyr vampyr 777947 Sep 2 14:42 .xsession-errors
drwxr-xr-x 5 vampyr vampyr 4096 Sep 2 14:42 Desktop/ ❷ drwx----- 2 vampyr vampyr 4096 Aug 6 19:32 tmp/
-rw-r--r-- 1 vampyr vampyr 3836 Oct 13 16:44 notes.txt ❸
```

- ❶ A symbolic link, indicated by the `->`, and showing the location it is linked to.
- ❷ An ordinary directory.
- ❸ An ordinary file.

<imageobjectco>

Currently unused in KDE Documentation.

<mediaobjectco>

Currently unused in KDE Documentation.

<areaset>

Currently unused in KDE Documentation. This and the above two elements will be used eventually (just as soon as I figure out how they work).

<graphicco>

Not to be used in KDE Documentation at all.

References, indexes, and glossaries

These elements are very underused in KDE Documentation up to this point, and we will probably make an effort to implement them more fully at some point. In the meantime, you may use them if you wish, so they are explained here.

Making a glossary

<glossterm>

Use this inline to identify words in the text that are explained further in a <glossary> or <glosslist>. When it's placed inside a <glossentry> it contains the term that glossary entry is defining (see the example below to see this in action.)

`<glossary>`

Put this where you have the glossary appearing. This is usually at the end of the document, perhaps last before the credits section, or before an index. A glossary will become a separate section in the book.

`<glosslist>`

Use this if the glossary is fairly short and simple. It can appear anywhere a normal list could appear. For KDE Documentation, a proper glossary is preferred, so keep use of `<glosslist>` to a minimum, where your glossary would only contain a small handful of entries. Use your own judgement which is most appropriate. You might use a `glosslist` for example, to explain a list of terms which only appear in one section, but are very important to understanding that section and occur several times there, so you want the explanations to appear close to the text.

`<glossdiv>`

Divides a glossary into several smaller sections. A good use of this in a very large glossary could be to break it up into separate sections for each letter in the alphabet.

`<glossentry id=" ">`

Contains the actual entries in the glossary or `glosslist`, where you explain the terms you have marked up with `glossterm` in the text. You should give these an `id`, so they can be linked to from the text, and crossreferenced between glossary entries.

A `glossentry` always contains one `<glossterm>`. It also contains one `<glossdef>`, or one `<glosssee>`, or a `<glossdef>` and a `<glossseealso>`.

Tip

I would suggest a consistent naming scheme, so glossary entries are easy to reference without having to go look them up all the time. For example, I use the form `id="gloss-word"`, where `word` is the term that is being explained.

`<glossdef>`

Contains the actual definitions of the terms

`<glosssee otherterm=" ">`

You can use this to save duplicating entries in the glossary. Instead of a `<glossdef>` you can put `<glosssee>` with the `id` of another `<glossentry>`.

`<glossseealso otherterm=" ">`

This is very similar to `<glosssee>`, but instead of replacing the `<glossdef>` it is in addition to it.

If you compare a glossary entry to a variable list entry, you'll see the structure is quite similar, with a `glossterm` taking the place of the `term`, and a `glossdef` taking the place of the `listitem`. Since variable lists get heavy use in KDE Documents, it shouldn't take you long to pick up how to do a glossary.

Example A.15. How to markup a glossary

Say you have in the text of the document the following sentence:

KWord is a graphical, wysiwyg word processor, and is part of KOffice.

You want to have the words KWord and koffice in the index, and KWord, wysiwyg, ?word processor? and KOffice explained in a glossary.

Many of these terms also need to be marked up with other tags, such as application, and acronym.

The eventual markup would look like this:

```
<para><glossterm linkend="gloss-kword">KWord</glossterm>
<indexterm><primary>KWord</primary></indexterm> is a
graphical <glossterm linkend
="gloss-wysiwyg"><acronym>WYSIWYG</acronym></glossterm>
<glossterm linkend="gloss-word-processor">word
processor</glossterm>, and is part of <glossterm
linkend="gloss-koffice">KOffice</glossterm>.
<indexterm><primary>KOffice</primary></indexterm></para>
```

The next part is shown here as a `<glosslist>`, and if there were really only this many entries in it, that could be entirely appropriate. In reality, if you are going to make a glossary, it would have many more entries and so would warrant its own `<glossary>` section. The syntax inside `<glossary>` and `<glosslist>` are otherwise the same.

```
<glosslist>
<glossentry id="gloss-kword">
<glossterm>KWord</glossterm>
<glossdef><para>The name of the KDE word
processor</para></glossdef>
</glossentry>

<glossentry id="gloss-koffice">
<glossterm>KOffice</glossterm>
<glossdef><para>A collection of office productivity tools, designed
by and for <acronym>KDE</acronym>, including presentation software,
a word processor, a spreadsheet, a <acronym>PIM</acronym>, and a
vector illustration application.</para></glossdef>
</glossentry>

<glossentry id="gloss-word-processor">
<glossterm>word processor</glossterm>
<glossdef><para>An application for handling text, typically more
concerned with formatting visually than a plain text
```

```
editor.</para></glossdef>
</glossentry>
```

```
<glossentry id="gloss-wysiwyg">
<glossterm>WYSIWYG</glossterm>
<glossdef><para>Stands for <quote>What You See Is What You
Get</quote>, indicating that you can visually format the presentation of
your data onscreen, and when you print the document, it will look exactly as you
see on the screen.</para></glossdef>
</glossentry>
</glosslist>
```

And the result of all this would be as follows:

KWord is a graphical ***WYSIWYG word processor***, and is part of ***KOffice***.

KWord

The name of the KDE word processor

KOffice

A collection of office productivity tools, designed by and for KDE, including presentation software, a word processor, a spreadsheet, a PIM, and a vector illustration application.

word processor

An application for handling text, typically more concerned with formatting visually than a plain text editor.

WYSIWYG

Stands for ?What You See Is What You Get?, indicating that you can visually format the presentation of your data onscreen, and when you print the document, it will look exactly as you see on the screen.

Making an Index

For KDE Documentation, indexes will in the future be generated automatically, so many of these elements are not to be used directly when authoring. At this stage, indexes are not generated, but if you want to you can mark up words that should be indexed with the `<indexterm>` element, to save work for later.

```
<indexterm>
```

Use this to note places in the main text of the document that should have an entry in the index. Don't over use it - not every single occurrence of a word needs to be noted in the index, but every occurrence where that term is significant should be.

`indexterm` should contain a `<primary>`, which contains the text that the entry will appear under in the index.

Place the `indexterm` directly before the word you want to index, and place the word itself inside the primary element. If the word should also be listed under a secondary heading, place that term inside a secondary element.

Example A.16. Index

Say the document contains the following sentence:

KWord is a graphical, wysiwyg word processor, and is part of KOffice.

You want KWord to have an index entry of it's own, and to also be noted under KOffice in the index.

```
<para><application>KWord</application>
<indexterm><primary>KWord</primary><secondary>KOffice</secondary></indexterm>
is a graphical, <acronym>WYSIWYG</acronym> word processor, and is part of
KOffice.</para>
```

The fact that an index entry exists is not normally indicated by a change in appearance.

If you think it should also be added under a third heading in the index, you can use tertiary to indicate this. Most terms you would find in KDE Documentation will only need a primary index heading, so use the others sparingly, if at all.

```
<tertiary>
```

```
    tertiary
```

```
<seealso>
```

```
    seealso
```

The following elements are used to create the actual index, but they are automatically generated, if required. You should not use them when authoring documents.

- `<index>`
- `<indexdiv>`
- `<indexentry>`
- `<primaryie>`
- `<secondaryie>`
- `<see>`
- `<seealsoie>`

- `<seeie>`
- `<tertiaryie>`

Other Reference Sections

`<firstterm>`

Mark up the first occurrence of a technically significant term with this element. If you are creating a glossary or an index, the first occurrence of a term will probably also warrant being an entry in one or both.

`<refsynopsisdivinfo>`

`refsynopsisdivinfo`

`<refnamediv>`

`refnamediv`

`<refclass>`

`refclass`

`<refmeta>`

`refmeta`

`<refsect1>`, `<refsect2>` and `<refsect3>`

`refsect1`, `refsect2` and `refsect3`

`<refmiscinfo>`

`refmiscinfo`

`<refsect1info>`, `<refsect2info>` and `<refsect3info>`

`refsect1info`, `refsect2info` and `refsect3info`

`<refdescriptor>`

`refdescriptor`

`<setindex>`

Not Used in KDE Documentation

<refpurpose>

refpurpose

<reference>

reference

<refentrytitle>

refentrytitle

<refname>

refname

<refentry>

refentry

<refsynopsisdiv>

refsynopsisdiv

Tags we do not use

These are tags that are available for DocBook XML, but we have decided they will not (at this time) be used for KDE Documentation. They are included here for completeness, and so nobody can say "I didn't know I wasn't supposed to use that!"

They fall into two categories: Tags we have definitely decided to not use, in which case we have made a decision to use another tag instead, and tags that are just irrelevant to the documentation we are doing, which you hopefully will never want. Should we write new documentation that can sensibly be marked up with any of these elements, this list will be revised.

If you think you have a use for one of these elements, please, check with the DocBook team first, and be prepared to justify your case.

- <ackno>
- <alt>
- <appendixinfo>
- <arthead>
- <article>

- <articleinfo>
- <artpagenums>
- <audiodata>
- <audioobject>
- <authorblurb>
- <authorinitials>
- <beginpage>
- <bibliodiv>
- <biblioentry>
- <bibliography>
- <bibliographyinfo>
- <bibliomisc>
- <bibliomixed>
- <bibliomset>
- <biblioset>
- <bookbiblio>
- <bridgehead>
- <chapterinfo>
- <citation>
- <citerefentry>
- <citetitle>
- <city>
- <collab>
- <collabname>

- <colophon>
- <colspect>
- <comment>
- <confdates>
- <confgroup>
- <confnum>
- <confsponsor>
- <conftitle>
- <contractnum>
- <contractsponsor>
- <corpname>
- <country>
- <database>
- <dedication>
- <docinfo>
- <edition>
- <editor>
- <entrytbl>
- <fax>
- <figure>
- <formalpara>
- <sgmltag>
- <graphic>
- <highlights>

- <honorific>
- <indexinfo>
- <informalequation>
- <informalfigure>
- <inlineequation>
- <inlinegraphic>
- <interfacedefinition>
- <interfacename>
- <invpartnumber>
- <isbn>
- <issn>
- <issuenum>
- <itermset>
- <jobtitle>
- <lineage>
- <lot>
- <lotentry>
- <manvolnum>
- <medialabel>
- <modespec>
- <msg>
- <msgaud>
- <msgentry>
- <msgexplan>

- <msginfo>
- <msglevel>
- <msgmain>
- <msgorig>
- <msgrel>
- <msgset>
- <msgsub>
- <objectinfo>
- <olink>
- <orgdiv>
- <orgname>
- <otheraddr>
- <pagenums>
- <part>
- <partintro>
- <phone>
- <pob>
- <postcode>
- <preface>
- <prefaceinfo>
- <printhistory>
- <productname>
- <productnumber>
- <property>

- <pubdate>
- <publisher>
- <publishername>
- <pubsnumber>
- <qandadiv>
- <refentryinfo>
- <referenceinfo>
- <remark>
- <revdescription>
- <revhistory>
- <revision>
- <revnumber>
- <revremark>
- <secondary>
- <section>
- <sectioninfo>
- <seriesinfo>
- <seriesvolnums>
- <set>
- <setindexinfo>
- <setinfo>
- <shortaffil>
- <sidebar>
- <sidebarinfo>

- `<simpara>`
- `<simplemsgentry>`
- `<simplesect>`
- `<spanspec>`
- `<state>`
- `<street>`
- `<subject>`
- `<subjectset>`
- `<subjectterm>`
- `<subtitle>`
- `<systemitem>`
- `<titleabbrev>`
- `<toc>`
- `<tocback>`
- `<tocchap>`
- `<tocentry>`
- `<tocfront>`
- `<toclevel1>`
- `<toclevel2>`
- `<toclevel3>`
- `<toclevel4>`
- `<toclevel5>`
- `<tocpart>`
- `<videodata>`

- `<videoobject>`
- `<volumenum>`
- `<wordasword>`

Alphabetical List of all elements

This is a list of all the markup elements contained in DocBook XML 4.1.2. Choose the element you are interested in to go directly to the section of this document which describes it.

Note

We don't use all these elements in KDE Documentation - they are here for completeness. Elements we don't use are listed in [the section called ?Tags we do not use?](#).

- `<authorinitials>`
- `<beginpage>`
- `<bibliodiv>`
- `<biblioentry>`
- `<bibliographyinfo>`
- `<bibliomset>`
- `<bibliomisc>`
- `<bibliomixed>`
- `<biblioset>`
- `<bibliography>`
- `<blockquote>`
- `<book>`
- `<bookbiblio>`
- `<bookinfo>`
- `<bridgehead>`
- `<co>`

- <callout>
- <calloutlist>
- <caption>
- <caution>
- <chapter>
- <chapterinfo>
- <citation>
- <citerefentry>
- <citetitle>
- <city>
- <classname>
- <classsynopsis>
- <classsynopsisinfo>
- <cmdsynopsis>
- <colspec>
- <collab>
- <collabname>
- <colophon>
- <command>
- <comment>
- <computeroutput>
- <confdates>
- <confgroup>
- <confnum>

- <confsponsor>
- <conftitle>
- <constant>
- <constructorsynopsis>
- <contractnum>
- <contractspnosor>
- <contrib>
- <copyright>
- <corpauthor>
- <corpname>
- <country>
- <database>
- <date>
- <dedication>
- <destructorsynopsis>
- <docinfo>
- <edition>
- <editor>
- <email>
- <emphasis>
- <envar>
- <entry>
- <entrytbl>
- <epigraph>

- <equation>
- <errorcode>
- <errorname>
- <errortype>
- <example>
- <exceptionname>
- <fax>
- <figure>
- <fieldsynopsis>
- <filename>
- <firstterm>
- <footnote>
- <footnoteref>
- <foreignphrase>
- <formalpara>
- <funcdef>
- <funcparams>
- <funcprototype>
- <funcsynopsis>
- <funcsynopsisinfo>
- <function>
- <guibutton>
- <guiicon>
- <guilabel>

- <guimenu>
- <guimenuitem>
- <guisubmenu>
- <glossdef>
- <glossdiv>
- <glossentry>
- <glosslist>
- <glosssee>
- <glossseealso>
- <glossterm>
- <glossary>
- <glossaryinfo>
- <graphic>
- <graphicco>
- <group>
- <hardware>
- <highlights>
- <holder>
- <honorific>
- <isbn>
- <issn>
- <itermset>
- <imagedata>
- <imageobject>

- <imabeobjectco>
- <important>
- <index>
- <indexdiv>
- <indexentry>
- <indexinfo>
- <indexterm>
- <informalequation>
- <informalexample>
- <informalfigure>
- <informaltable>
- <initializer>
- <inlineequation>
- <inlinegraphic>
- <inlinemediaobject>
- <interface>
- <interfacedefinition>
- <interfacename>
- <invpartnumber>
- <issuenum>
- <itemizedlist>
- <jobtitle>
- <keycap>
- <keycode>

- <keycombo>
- <keysym>
- <keyword>
- <keywordset>
- <legalnotice>
- <lineannotation>
- <lineage>
- <link>
- <listitem>
- <literal>
- <literallayout>
- <lot>
- <lotentry>
- <manvolnum>
- <markup>
- <medialabel>
- <mediaobject>
- <mediaobjectco>
- <member>
- <menuchoice>
- <methodname>
- <methodparam>
- <methodsynopsis>
- <modespec>

- <modifier>
- <mousebutton>
- <msg>
- <nmsgaud>
- <msgentry>
- <msgexplan>
- <msginfo>
- <msglevel>
- <msgmain>
- <msgorig>
- <msgrel>
- <msgset>
- <msgsub>
- <msgtext>
- <note>
- <olink>
- <objectinfo>
- <option>
- <optional>
- <orderedlist>
- <orgdiv>
- <orgname>
- <otheraddr>
- <othercredit>

- <othername>
- <pob>
- <pagenums>
- <para>
- <paramdef>
- <parameter>
- <part>
- <partintro>
- <phone>
- <phrase>
- <postcode>
- <preface>
- <prefaceinfo>
- <primary>
- <primaryie>
- <printhistory>
- <procedure>
- <productname>
- <productnumber>
- <programlistingco>
- <prompt>
- <property>
- <pubdate>
- <publisher>

- <publishername>
- <pubsnumber>
- <qandadiv>
- <qandaentry>
- <qandaset>
- <question>
- <quote>
- <refclass>
- <refdescriptor>
- <refentry>
- <refentryinfo>
- <refentrytitle>
- <referenceinfo>
- <refmeta>
- <refmiscinfo>
- <refname>
- <refnamediv>
- <refpurpose>
- <refsect1>
- <refsect1info>
- <refsect2>
- <refsect2info>
- <refsect3info>
- <refsect3info>

- <refsynopsisdiv>
- <refsynopsisdivinfo>
- <reference>
- <releaseinfo>
- <remark>
- <replaceable>
- <returnvalue>
- <revdescription>
- <revhistory>
- <revnumber>
- <revremark>
- <revision>
- <row>
- <sbr>
- <sgmltag>
- <screen>
- <screenco>
- <screeninfo>
- <screenshot>
- <secondary>
- <secondaryie>
- <sect1>
- <sect1info>
- <sect2>

- <sect2info>
- <sect3>
- <sect3info>
- <sect4>
- <sect4info>
- <sect5>
- <sect5info>
- <section>
- <sectioninfo>
- <see>
- <seealso>
- <seealsoie>
- <seeie>
- <seq>
- <seqlistitem>
- <segmentedlist>
- <seriesinfo>
- <seriesvolnums>
- <set>
- <setindex>
- <setindexinfo>
- <setinfo>
- <shortaffil>
- <shortcut>

- <sidebar>
- <sidebarinfo>
- <simpara>
- <simplelist>
- <simplemsgentry>
- <simplesect>
- <spanspec>
- <state>
- <step>
- <street>
- <structfield>
- <structname>
- <substeps>
- <subject>
- <subjectset>
- <subjectterm>
- <subscript>
- <subtitle>
- <superscript>
- <surname>
- <symbol>
- <synopfragment>
- <synopsis>
- <systemitem>

- <tbody>
- <tfoot>
- <tgroup>
- <thead>
- <table>
- <term>
- <tertiary>
- <tertiaryie>
- <textobject>
- <tip>
- <title>
- <titleabbrev>
- <toc>
- <tocback>
- <tocchap>
- <tocentry>
- <tocfront>
- <toclevel1>
- <toclevel2>
- <toclevel3>
- <toclevel4>
- <toclevel5>
- <tocpart>
- <token>

- <trademark>
- <type>
- <ulink>
- <userinput>
- <varargs>
- <varlistentry>
- <varname>
- <variablelist>
- <videodata>
- <videoobject>
- <void>
- <volumenum>
- <warning>
- <wordasword>
- <xref>
- <year>

Credits and License

Document copyright 2000, 2001 Lauri Watts <lauri@kde.org>

This reference was written with substantial help and input from the following people who definitely deserve credit:

- Frederik Fouvry
- Eric Bischoff
- Michael McBride
- Lee Wee Tiong

- Philip Rodrigues
- Eyal Lotem <GNUPeaker@yahoo.com>
- Malte Starostik <malte.starostik@t-online.de>
- Antonio Larossa Jiminez

Appendix B. Widget Names

Steal from (and extend) the ?Visual Guide to KDE?.