



Firebird Docwriting Guide

Paul Vinkenoog

5 May 2007 – Document version 1.3

Table of Contents

Introduction	3
Purpose of this guide	3
Assumed knowledge	3
Topics discussed in this guide	3
Where the docmakers meet	4
The subproject homepage	4
The firebird-docs list	4
The Atkin news interface	4
Picking a subject	5
Preparing to write: make an outline!	5
DocBook XML – an introduction	6
A very general XML primer	6
A DocBook XML primer	9
DocBook XML authoring tools	11
Text editors	11
XML editors	11
Setting up your DocBook doc	13
Creating the document	13
Typing text	14
Elements we use frequently	15
Hierarchical elements	15
Lists	19
Links	20
Program listings, screens, literal layout, and examples	22
Tables	24
Images	27
Admonitions	28
Paragraph headers	29
Various inline elements	30
To wrap up the elements	32
Language and style	33
Language	33
Style	33
Copyright issues	35
Using material written by others	35
Your copyright and the PDL	36
Adding your document to the manual module	40
Asking for commit rights	40
Dos and don'ts if you have received commit rights	40
Committing your work	41
Publishing your document on the Firebird website	41
Naming the PDF file	41
Single-page HTML	42
Uploading the PDF	42
Uploading the multi-page HTML files	43
Updating the Firebird Documentation Index	43
Appendix A: Document History	45
Appendix B: License notice	47

Introduction

Purpose of this guide

This guide discusses the various aspects of writing documentation for Firebird. It is intended for people who want to help write documentation for the Firebird project, or who at least strongly consider to do so. After reading this guide, you'll have all the necessary knowledge to start writing Firebird docs in our chosen format DocBook XML.

Assumed knowledge

Before reading this guide, check with yourself if you know:

- What the Firebird manual module is.
- What CVS is, and how to use a CVS client to download the current manual module.
- How to build the current Firebird documentation from your downloaded manual module.

This knowledge is essential if you are going to contribute to our documentation project. If you feel unsure about one or more of these points, you should first read the [Firebird Docbuilding Howto](#), and then come back here.

Topics discussed in this guide

We start off with some short chapters about:

- The firebird-docs mailing list.
- Picking a subject.
- Making an outline for your document-to-be.

After that we'll take some time to explain the basics of DocBook XML, because that's the format we'd like you to deliver your docs in. Topics discussed include:

- DocBook XML – what is it?
- Reasons why we prefer DocBook so strongly to other formats.
- Tools you can use to produce DocBook texts.

Don't worry if DocBook doesn't mean anything to you yet: the required knowledge can be learned in less than an hour, and chances are that you will benefit from this knowledge in other projects too, whenever you have to write technical documentation.

The next part is about the actual docwriting:

- Setting up the document itself.
- Using DocBook elements.
- A word or two on language and writing style.
- Copyrights and the Public Documentation License.

Finally, we will show you how to add your finished doc to the Firebird project. Main topics in this section are:

- Committing your finished document to the manual module.
- Where to ask for commit rights if you don't have them.
- Dos and don'ts once you have received commit rights.
- Publishing HTML and PDF versions on the Firebird website.

Where the docmakers meet

The subproject homepage

The homepage of the documentation subproject is here:

<http://www.firebirdsql.org/index.php?op=devel&sub=doc>

It contains news about our activities, links to the docs we've already published, plans for the future, etc.

The firebird-docs list

If you're serious about writing docs for Firebird, the first thing you should do is subscribe to the mailing list where we discuss our plans and our work. This list is open to everybody, and subscribing commits you to nothing. Send an email to:

`<firebird-docs-request@lists.sourceforge.net>`

with the word “subscribe” either as subject or as the first and only line in the message body. Alternatively, you can fill in the form on this webpage:

<http://lists.sourceforge.net/lists/listinfo/firebird-docs>

Whichever method you choose, you'll receive an email message from the list robot within minutes. Follow the instructions in that message and you're on the list.

The Atkin news interface

There's also a news interface for this and other Firebird-related mailing lists. Sometimes it works a little problematically – or not at all, although that's rare – and I wouldn't use it to post messages if I were you, but it's great for archiving and browsing purposes. The retention is pretty long too, so if you grab all messages that are on the server you'll have a nice piece of list history from before you joined.

To access these newsgroups, point your favourite newsreader to:

`news.atkin.com`

and get the groups list. Subscribe to the groups you like. Notice that the firebird-docs list maps to the group `sourceforge.firebird-doc` (without s) on the Atkin news server.

Depending on the newsreader or browser you use, this link may also get you straight to the group:

[news://news.atkin.com/sourceforge.firebird-doc](https://news.atkin.com/sourceforge.firebird-doc)

You can post to the newsgroup even if you're not subscribed to the mailing list, but in that case your message will be held for approval by a human moderator. This means that your message will be delayed by up to a day (or more, on rare occasions).

Picking a subject

These guidelines may help you in finding a subject to write about:

- First make sure you know what's already there – nobody's waiting for three MS-SQL-to-Firebird conversion guides.
- Then ask yourself what's missing, and what may be useful for Firebird users in general, or perhaps just for a specific group.
- Also ask yourself what you would *like* to write about. The most logical choice would be a topic you are familiar with, but you can also pick a subject you'd have to learn more about first (this is much more work of course, but a great learning experience if you're willing to invest the time).
- You don't necessarily have to write an entire book, guide or article. Maybe there are already people working on a larger production, which you can contribute to. Maybe you can write one or more chapters for a book. Or maybe you can supply raw documentation material for a subject you know a lot about.
- Talk about your ideas – or your search for ideas – on the firebird-docs list. The posting frequency can be very low at times, but rest assured that if you post there, your message *will* be read, and replied to.

Preparing to write: make an outline!

It's always a good idea to make an outline before you start to write the actual text. Setting up an outline will help you to “get organized”; it reduces the chance of forgetting something vital, and it will make the actual writing job a lot easier.

You can follow these steps when making your outline:

- Define exactly what you want your readers to learn from your work.
- Divide the subject matter into logical units – chapters and/or sections and/or subsections.
- Make sure the order of the units makes sense, especially for a howto, tutorial or user's guide. That is: arrange the units in such a way that whatever the user has to do first, or understand first, also comes first in your documentation.
- Present your outline on the firebird-docs list at sourceforge.net and ask for comments.

Once you are satisfied with your outline, look it over thoroughly and decide whether you have all the (raw) information you need to start writing. Ideally, you want to have all the information ready before you start to write, because sometimes a formerly unknown piece of information may lead you to choose a different document structure. You'd better have that information while you're still in the outline phase, then.

DocBook XML – an introduction

The chosen format for the documentation in the Firebird manual module is *DocBook XML*. For those of you who are not familiar with XML and/or DocBook, short introductions to XML in general and DocBook XML in particular follow. Be aware that these introductions give a grossly oversimplified picture. But that's just fine: you don't have to be a DocBook XML expert in order to write Firebird docs. You only need some basic knowledge – which you can pick up in half an hour from the paragraphs below – and a little experience in applying DocBook XML tags to your texts (which you will gain soon enough once you start writing).

Skip the [general XML primer](#) if you know all about XML elements, tags, attributes, rendering, and multichannel publishing.

Skip both primers if you're also an experienced DocBook author.

Note

While we strongly ask that you at least *try* to deliver your work in DocBook format, we also realise that some people just won't have the time to master it (or to convert their existing docs to DocBook). If this applies to you, please talk about it on the [firebird-docs list](#). We surely don't want to refuse useful documentation just because it's not in the right format.

A very general XML primer

XML stands for *Extensible Markup Language*, which is, simply put, plain text with markup tags. A typical XML text fragment may look like this:

```
<paragraph>
<loud>'No!</loud> she screamed. <scary>But the bloody hand
<italics>kept on creeping</italics> towards her.</scary>
<picture file="bloody_hand.png"/>
</paragraph>
```

Tags and attributes

In the example given above, the words and phrases enclosed in angle brackets are the markup tags. `<italics>` is a *start tag*, `</italics>` is an *end tag*, and `<picture file="bloody_hand.png"/>` is a standalone tag, officially termed *empty-element tag*. XML tags are always formatted like this:

Table 1. Format of XML tags

Tag type	Starts with	Ends with
Start tag	<	>
End tag	</	>
Empty-element tag	<	/>

Still referring to our example, the words `paragraph`, `loud`, `scary`, `italics` and `picture` are *tag names*. In the `<picture.../>` tag, `file="bloody_hand.png"` is called an *attribute*, with `file` the *attribute name* and `bloody_hand.png` the *attribute value*. Attribute values must always be quoted; both single and double quotes are allowed.

XML allows you to define any tags you like, as long as you build them correctly. So `<thistag>`, `<thattag>`, and `<this_is_not_a_tag/>` are all well-formed XML tags. (XML that follows the standard is called *well-formed*; the term *valid* is only used for specifically defined implementations – DocBook XML, for instance.)

Clearly the tags themselves are not meant to appear in the final document (that is, the document as it is presented to the readers). Rather, they contain instructions that affect its appearance. XML, when used for writing documentation, is a typical *source format*, intended to be processed by software to produce nicely formatted output documents. This processing is usually called *rendering*.

Some tags are unmistakably makeup instructions:

```
<italics>kept on creeping</italics>
```

means of course that the words *kept on creeping* must be displayed or printed in italics. However,

```
<loud>'No!'</loud>
```

is a little less obvious. Should the word `NO!` appear in boldface? Or underlined? Or again in italics? Or maybe this text is going to be read out aloud by a speech synthesizer, and the `<loud>` tag instructs it to raise its voice? All these things are possible, and what's more: often a single XML source document is converted into several different output formats – say, a PDF document, an HTML web page, and a sound file. This is called *multichannel publishing*. With multichannel publishing, `<loud>` may be translated to boldface for the PDF document; to a bold, red-colored font for the web page; and to a 50% volume increase for the synthesizer.

Looking at the other tags, `<picture.../>` is obviously an instruction to insert the image `bloody_hand.png` into the document, and `<scary>`, well... this is even less clear than `<loud>`. Maybe the phrase between the `<scary>` tags has to drip with blood. Maybe frightening music must be played here. It all depends on the people who defined the tags, and the software they use to do the rendering.

The `<paragraph>` tag, finally, is a structural tag. It tells us something about the place that the lines have within the document's internal hierarchy. In the final document, paragraphs may or may not be separated by empty lines. Again, that depends on the rendering software and possibly also on user-configurable options. Other structural tags one might think of are e.g. `<chapter>`, `<section>`, and `<subdocument>`.

Special characters and Entities

Because the character “<” has a special meaning as the start of a tag, you can't include it directly as a literal value. Instead, if you want your readers to see an opening angle bracket, you type this:

```
&lt;
```

That's an ampersand, followed by the letters `l` and `t` (for *less than*), followed by a semicolon. You can also use `>` (*greater than*) for the closing angle bracket “>”, but you don't have to.

XML has lots of codes like this; they are called *entities*. Some represent characters, like `<` and `ä` (lower a with umlaut) and some serve totally different purposes. But they all start with an ampersand and end with a semicolon.

But wait a minute... if an ampersand marks the start of an entity, how do you include a literal ampersand in your text? Well, there's an entity for that too:

```
&amp;
```

So this line of XML:

```
Kernigan &amp; Ritchie chose '&lt;' as the less-than operator for C.
```

will wind up in the final documents as:

```
Kernigan & Ritchie chose '&lt;' as the less-than operator for C.
```

And here's some good news: if you use a dedicated XML editor to author your document, you can probably just type “<” and “&” anywhere you want to use them as literals. The editor will make sure that they end up as `<` and `&` in the XML as it is saved to disk. You'll find pointers to some XML/DocBook editors later in this guide.

Elements

There's one more important XML concept you need to know about: the *element*. An element is the combination of a start tag, a matching end tag, and everything in between. This “everything in between” is called the element's *content*, and it may include other elements. Elements are named after their tags, so we can talk about paragraph elements, italics elements etc.

Note

Actually, elements are a more basic concept than tags: tags just happen to be the things that identify the elements. So it would be better to say that tags are named after their elements. But because tags are easier to recognize than entire elements, I thought I'd introduce you to them first.

This is an element:

```
<loud>'No! '</loud>
```

This is also an element:

```
<paragraph>This is an element containing <bold>another</bold>  
element!</paragraph>
```

Empty-element tags constitute an element all by themselves. These elements can have no content of course, because they don't have a *pair* of tags:

```
<picture file="bloody_hand.png" />
```

Important

Don't confuse content with attributes. Content lives *between* tags, attributes *within* tags. The empty element in the last example has an attribute, but no content.

I'm stressing the element concept here because most documentation tends to speak of “chapter elements”, “title elements” etc. rather than “chapter tags” and “title tags”. The terms are often used interchangeably, but there are cases where it's important to know the difference.

XML Conclusion

Good – that's about all you need to know about XML. By now you should have a general idea of what an XML text looks like, what tags and elements are, and what they are for. As said earlier, the picture is oversimplified but it's good enough for our purposes.

It should also be understood that just writing away in plain, self-invented XML is pretty pointless unless you have processing software that understands *your* tags. How else are you going to turn your XML source into a nicely formatted, presentable document?

Fortunately, we don't have to worry about developing our own element definitions and conversion software. There are a number of formalized XML types available, each defining a set of tags and, equally important, a set of rules on how to use them. DocBook XML is one of those types.

A DocBook XML primer

DocBook was designed to facilitate the writing of structured documents using SGML or XML (but don't worry about SGML – we use the XML strain). It is particularly fit for writing technical books and articles, especially on computer-related subjects. DocBook XML is defined in its *Document Type Definition* or *DTD*: a set of definitions and rules describing exactly how a valid DocBook document is structured. DocBook is rapidly becoming a de facto standard for computer-technical documents, and it is supported by a growing number of tools and applications.

DocBook XML Characteristics

Important characteristics of DocBook – as opposed to “general” XML – are:

- The DocBook DTD defines a limited number of tags, and it gives exact rules on how to use them: what attributes are possible for a tag A, whether element B can be nested within element C, and so on. If you use undefined tags, or if you don't follow the rules, your document isn't DocBook anymore (and DocBook-supporting processing tools may break on it).
- DocBook tags always convey structure and semantics (meaning), *never* makeup. In DocBook, you'll find structural tags like `<book>`, `<part>`, `<chapter>`, `<section>`, `<para>`, `<table>`; and semantic tags like `<filename>`, `<warning>`, `<emphasis>`, `<postcode>`; but nothing like ``, `<bold>`, `<center>`, `<indent>`, `<backgroundcolor>` – nothing that has to do with layout or makeup.
- Because of this, a decision has to be taken somewhere on how the DocBook tags are translated into presentational makeup. This decision (or rather: the rendering rules) can be hardcoded in the tools but that would make things very inflexible. That's why the rules are mostly defined in *stylesheets*. A stylesheet is a document that tells the tool stuff like:

“Print chapter titles in a 24-point black font; start each chapter on a new page; use italics for emphasis; render warnings in a bold, 12-point red font; use smallcaps for acronyms; etc. etc.”

This approach enables the user to alter the stylesheets if he or she doesn't like the appearance of the final document. It would be a lot harder – if not impossible – to alter the tools themselves.

Note

Stylesheets that are used to convert DocBook XML to other formats are called *transformation stylesheets*. They are written in yet another type of XML, called *XSLT* (eXtensible Stylesheet Language for Transformations).

Benefits of DocBook XML

DocBook has a lot of advantages for anybody writing technical documentation. These are the most important ones for us:

- A DocBook XML document consists of pure, unpolluted, *content*. You never have to worry about the presentational side of things while writing your doc; you can concentrate on structure and informational content. This practice may at first feel a little odd if you're used to writing text in e.g. Word, but I promise you: you'll soon get to love it.
- Because DocBook is all about structure and meaning, it will be surprisingly easy to transform your outline into a DocBook skeleton.
- Many people produce docs for the manual module. If they all used different formats, or even one single format like Word or HTML, their works would look very different because every contributor would make his or her own makeup decisions. Of course we could develop a set of makeup rules, but then every docwriter would have to be aware of those rules, and take care to apply them all the time. Nah... better put the rules in one central place: the stylesheets, and let the docmakers worry about documentation, not presentation. The stylesheets will ensure that all our documentation has the same look-and-feel.
- If we don't like the makeup of our documents, we can easily change it if the makeup rules are in a stylesheet. Nothing needs to be altered in the DocBook source documents; all we have to do, after changing the stylesheets, is re-render the docs. Newly developed docs will automatically get the new look. Try to achieve that if the makeup instructions are scattered all over the documents themselves!
- Another advantage is that DocBook is an open standard, not tied to any commercial application or even a particular OS. If you download the Firebird manual module, you can build the HTML and PDF docs from the DocBook source both under Linux and under Windows – and we can add support for more OS's if need be.
- A DocBook document is pure text, which is ideal for use in CVS. Yes, a CVS tree can also contain binary files, but many useful features that CVS offers (showing the difference between two versions of a file, for instance) only work with text files.

Admittedly, none of these benefits is unique to DocBook. But DocBook has them all, and it's widely supported. That makes it the perfect choice for our Firebird documentation.

DocBook documentation on the Internet

Here are some links in case you want to find out more about DocBook:

- <http://opensource.bureau-cornavin.com/crash-course/>

Writing Documentation Using DocBook – A Crash Course by David Ruge, Mark Galassi and Eric Bischoff. A very nice tutorial, even though most of the tools discussed are not the ones we use.

- <http://docbook.org/tdg/en/>

DocBook – The Definitive Guide, by Norman Walsh and Leonard Muellner. Don't expect it to be a beginner-friendly tutorial – in fact, the first part is quite intimidating if you're a DocBook newbie. The reason I mention it here is its great online element reference, which I often consult while I'm writing.

- <http://www.tldp.org/HOWTO/DocBook-Demystification-HOWTO/>

The *DocBook Demystification Howto* is interesting if you want to know a little more about XML and DocBook than what we've told you here. It also contains quite a lot of material on SGML, and – again – on tools we don't use for the Firebird documentation subproject.

- <http://sourceforge.net/projects/docbook>

The DocBook open source project at SourceForge.

If you know of some other great online resource, please let us know by posting a message to the firebird-docs list.

DocBook XML authoring tools

Text editors

Because DocBook is a non-binary format, you can use any plaintext editor like emacs, pico, Windows Notepad or vi to write your documentation. And indeed, some docmakers prefer this approach to other more sophisticated tools because it gives them full control over their text, and the hand-typed tags are always visible. But the drawback is that text editors can not *validate* your DocBook document: you'll only notice your mistakes when a build goes wrong. And the structure of your document – especially a large document – is also difficult to see in text mode, although a consistent use of multi-level indentation can do a lot of good here.

If you choose this approach or want to try it out, it would be a good idea to at least take an editor that's capable of XML syntax highlighting. A good one, and free at that, is ConText, available at <http://www.fixedsys.com/context/>. Unfortunately, ConText can't save in UTF-8 format. This is no problem for US-ASCII documents (save as DOS or Unix and you're fine), but as soon as you use diacritical marks or anything else above ASCII 127, ConText becomes as good as useless. A good, free alternative is SciTE at <http://scintilla.sourceforge.net/SciTEDownload.html>. It's less intuitive, but very powerful.

Warning

Don't save documents containing non-US-ASCII characters as 8-bit, in ConText or any other editor! Anything other than US ASCII has to be saved in a Unicode encoding, such as UTF-8 (for most languages) or UTF-16 (for some languages, if the UTF-16 file length is smaller or at least not much bigger than UTF-8). Actually, these encoding issues are an additional good reason to use an XML editor: they will usually save in the right encoding automatically.

XML editors

Dedicated XML editors often have graphical interfaces to make the tags stand out nicely (and sometimes irritatingly); many allow you to collapse and expand elements so you can view the structure of your document and at

the same time zoom in on the element you're working on; they may also let you switch between different views. Most of them can validate your document against the DocBook DTD, and some even have a special DocBook authoring mode which allows you to write more or less like in a word processor.

The author of this guide has tried out a number of these tools (free ones, cheap ones, and evaluation versions) and found XMLMind XML Editor to be the most useful. This is a personal opinion of course; your experience may differ.

Some XML editors you may want to evaluate:

- XMLMind XML Editor, or XXE for short. The Standard Edition is free.

<http://www.xmlmind.com/xmleditor/>

Runs on: Linux, Windows, Mac OS X. Requires Java, but you need Java anyway or you won't be able to build the docs from the sources – see the [Firebird Docbuilding Howto](#).

Features: Tree view (all elements collapsible) and Styled view (chapters and sections collapsible). The latter is what I usually work in: it shows the document in a basic but clear word-processor-like layout, defined in a stylesheet that comes with the program. Both views can be active simultaneously. DocBook mode won't let you enter anything non-DocBook. Element chooser. Attribute editor. Edit and Search functions. Spell checker. Special character picker. Speedbuttons to create frequently used elements like sections, lists, tables, etc. What I miss is a plaintext XML source view.

- Oxygen XML Editor. \$ 48 for non-commercial use. Free 30-day trial.

<http://www.oxygenxml.com>

Runs on: Windows, Mac OS X, Linux, Eclipse. Requires Java.

Features: XML source editor. Tree editor. Attribute editor. Outline pane. DocBook tag tooltips. XSLT debugger (a powerful tool, irrelevant to docwriting but great if you're also going to work on our transformation stylesheets). Validation, refactoring, spell-checking, etc., etc. A very good XML editor.

- epcEdit. € 89 for non-commercial use. Free 60-day evaluation.

<http://www.epcedit.com>

Runs on: Linux, Windows, Solaris. Requires Tcl/Tk 8.1 or above (included in package).

Features: Structure tree pane. Element chooser. Attribute editor. Document pane can switch between plaintext and graphic XML mode. No special DocBook mode, but can validate any XML document based upon its DTD.

- Altova XMLSpy. The Home Edition is now free.

http://www.altova.com/products_ide.html

Runs on: Windows, Eclipse. (Also reported to run on Linux using Wine, and on Mac OS X using Virtual PC 6.)

Features: Text and Browser views. All elements collapsible in Browser view. Browser view is read-only. Element chooser. Attribute picker. Edit and Search functions. Special character picker.

There's a feature matrix comparing Home, Professional and Enterprise editions at http://www.altova.com/matrix_x.html.

This list is not meant to be exhaustive, but if you know a *good* XML editor (good from the perspective of a Firebird docwriter) that you think should be in here, please let us know via the mailing list.

Setting up your DocBook doc

Hello – still with us? I know I spent quite some time explaining about XML and DocBook, but I really feel I had to do that because these are new concepts to a lot of people. Just giving them some links and telling them to go find out by themselves would probably lose us some otherwise valuable docwriters.

Anyway, here we are: finally ready to start writing our doc. This section discusses setting up your DocBook document; the next one is all about applying the right tags and attributes in the right places.

Creating the document

Every piece of documentation in our manual module is part of a `<set>`. This is the topmost element in the DocBook hierarchy. A set contains a number of `<book>`s, which in turn contain `<chapter>`s, and so on.

One advantage of placing books in a set is that they can reference each other, i.e. you can insert links in your documentation pointing to an exact spot in another book. This advantage is limited however by the fact that such links don't work across PDF file boundaries (a restriction that doesn't apply to the HTML output). Another advantage is automatic ToC (Table of Contents) generation.

Fortunately, placing books in the same set does not imply that they also have to live together in one big file. DocBook allows you to set up a main document as shown below. (Don't worry about the section starting with "`<!DOCTYPE`" – you won't have to write horrible stuff like that yourself. At the very worst you will have to copy and edit it, if you translate an existing set.)

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE set PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
  "docbookx/docbookx.dtd" [
  <!ENTITY preface SYSTEM "firebirddocs/preface.xml">
  <!ENTITY fb-intro SYSTEM "firebirddocs/firebirdintro.xml">
  <!ENTITY ...>
  <!ENTITY ...>
]>

<set id="firebird-books">
  &preface;
  &fb-intro;
  ...
  ...
</set>
```

With the main document set up like above, the various books can be in separate files: `preface.xml`, `firebirdintro.xml`, etc., which we can edit independently. Such a file – yours, for instance – is roughly structured like this:

```
<?xml version="1.0" encoding="UTF-8"?>

<book id="fbintro">
  <chapter id="fbintro-preface">
```

```
...
...
</chapter>
<chapter id="fbintro-installing-firebird">
...
...
</chapter>
...
...
</book>
```

Of course if you set up a new document it must be made known to the main set, but this is something we'll discuss with you when you're ready to start writing. (We don't give a general rule here because it depends on what you're going to write – a book, an article, a chapter, a couple of chapters... – and how your work fits in with the rest.)

Every DocBook file must start with this line:

```
<?xml version="1.0" encoding="UTF-8"?>
```

(Note: for some languages, UTF-16 will be the better choice.)

If you write your documentation “by hand”, i.e. in a text editor, you must type that line yourself. If you use a dedicated XML editor, it will be inserted automatically when you create a new document.

File locations for the different sets

Files for the English user documentation set must be placed in the `manual/src/docs/firebirddocs` directory tree. Non-English docs go in trees like `manual/src/docs/firebirddocs-fr`, `manual/src/docs/firebirddocs-es`, etc.

Since January 2006 we have the possibility of creating additional base sets, the first one added being `rlsnotes`, the Release Notes set. The same logic applies here: English Release Notes stuff goes into `manual/src/docs/rlsnotes`, French into `manual/src/docs/rlsnotes-fr`, and so on.

Each of these directory trees – `firebirddocs`, `firebirddocs-es`, `firebirddocs-nl`, `rlsnotes`, `rlsnotes-fr`, etc. – houses a separate `<set>`, with one master document and any number of include files.

Typing text

If you type your DocBook XML in a text editor like Notepad, emacs or ConText, you can use linebreaks, indentation and multiple spaces more or less as you please. Every occurrence of *whitespace* (a sequence of one or more space, tab, linefeed or formfeed characters) will be converted to a single space character in the output. So this:

```
<section><title>Firebird Architectures</title><para>Now let's have a
look at Firebird's different architectures.</para><itemizedlist>
<listitem><para>First, there's the so-called <firstterm>Classic Server
</firstterm>.</para></listitem><listitem><para>Then there is <firstterm>
Superserver</firstterm> architecture.</para></listitem><listitem><para>
And finally, with the release of Firebird 1.5 we also have the
<firstterm>embedded server</firstterm>.</para></listitem></itemizedlist>
</section>
```

will result in the same output as this:

```
<section>
  <title>Firebird Architectures</title>
  <para>Now let's have a look at Firebird's different
    architectures.</para>
  <itemizedlist>
    <listitem>
      <para>First, there's the so-called
        <firstterm>Classic Server</firstterm>.</para>
    </listitem>
    <listitem>
      <para>Then there is <firstterm>Superserver</firstterm>
        architecture.</para>
    </listitem>
    <listitem>
      <para>And finally, with the release of Firebird 1.5 we also
        have the <firstterm>embedded server</firstterm>.</para>
    </listitem>
  </itemizedlist>
</section>
```

Needless to say, the second form is much easier to read and understand for a human. So if you type your XML by hand, format the text in such a way that the structure is as clear as possible. Like the prophets said: “Indent! Indent! Indent!” (Or was that repent? No, I'm sure it was indent.)

If you use a dedicated XML editor, please be aware that hitting **Enter** may automatically close the current `<para>` and open a new one. Make sure you know how your editor behaves in this respect, and use the Enter key accordingly. Also check what happens to multiple consecutive whitespace characters, as some XML editors may use special tricks to preserve them.

Elements we use frequently

This section discusses the DocBook elements we use most in our Firebird docs. It includes lots of examples in DocBook XML format. If you use an XML authoring tool, what you see on your screen may look nothing like the examples given here, but if you open your XML file in a text editor – or choose a text view in your XML tool – you will see the actual XML. You may also have a look at the XML sources that are already in the manual module, to see how the other authors build up their docs and apply tags.

Please read the subsection on hierarchical elements even if you're a proficient DocBook writer, as it contains some guidelines specific to our project. After that, you can skip the rest of the DocBook subsections.

If you're new to DocBook, don't be discouraged by the length of this section. My advice is that you *carefully* read the subsection on hierarchical elements, and skim the others. Don't worry if there are things you don't understand at once, and by no means try to learn the material by heart! Just have this guide handy when you write your doc, and revisit the element subsections from time to time (like when you need them).

Hierarchical elements

The most common hierarchy is, starting at the top: `<set>` – `<book>` – `<chapter>` – `<section>` – `<para>`. A book may also contain `<article>`s instead of `<chapter>`s.

The next subsections will discuss some of the issues related to the document structure.

The `id` attribute

Sets, books, chapters, articles and top-level sections should always have an `id` attribute. Other elements may also have one. The `id` allows an element to be referenced from another part of the document, and even from another document in the set. Ids are not visible in the rendered docs (except in the HTML source text), but they are used to form the HTML file names.

All `id` attributes must be unique within the entire bookset. Note that the different language versions each live in their own `set`, so it's OK to keep the original `ids` in a translation.

Within a book or article, all `ids` should start with the same lowercase word, e.g. `usersguide`, followed by a dash, followed by one or more other lowercase words. Examples are `usersguide-intro` and `usersguide-download-install`. This is not a DocBook requirement, but our own convention.

The `lang` attribute on non-English sets

If you create a new set, or translate one, you must set the `lang` attribute on the root element:

```
<set id="firebird-books-fr" lang="fr">
```

This will ensure that the right captions are generated for notes, warnings etc., and that localized quotation marks are used. It's also good practice to use this attribute on the individual docs, just in case they're ever build out of the context of your set.

For English sets, the `lang` attribute is optional.

Titles

Sets, books, chapters, articles and sections must always have a `title` – either as a direct child, or within an `xxxinfo` element (see below). It is even legal to include it in both, but in that case the two `titles` *must* be the same. Unlike `id`, which is an attribute, `title` is an element. And unlike the `id`, the title will appear in the output docs.

If the `title` is long, you should add a `titleabbrev` element immediately after it, containing a shortened form of the title. The main reason for this is that each generated HTML page contains a so-called hierarchy bar or “you-are-here line” at the top and bottom. This bar shows all the steps from the topmost element (the `set`) down to the page you are on. The items are clickable so the bar doesn't only give you an insight in where you are in the hierarchy, but it also lets you navigate up to the higher-level elements easily. It looks best if all the items fit on one line, so for each item the `titleabbrev` is shown if the element in question has one; if not, the `title` is used. The same strategy is followed for the outline in the PDF documents (that's the navigation frame on the left).

Info elements

If you write a book or an article, you must include a `bookinfo` or `articleinfo` element at the start. Inside it you can put author information and more. Other `xxxinfo` elements exist, but you will rarely need them.

```
<book id='usersguide' lang='en'>
  <bookinfo>
    <title>Firebird Users Guide</title>
    <author>
```

```
<firstname>William</firstname>
<surname>Shakespeare</surname>
</author>
<edition>25 January 2006 - Document version 1.2</edition>
</bookinfo>
...
...
</book>
```

If the author is a company or other organisation, or a group you want to refer to as a collective, use `corpauthor` instead of `author`:

```
<corpauthor>IBPhoenix Editors</corpauthor>
```

If there are several authors and you want to name them separately, create an `author` (or `corpauthor`) element for each of them and wrap them together in an `authorgroup` element – all within the `xxxinfo` element.

Types of sections

Section elements are a bit different from the rest in that there are two flavors of them:

- First, the `<section>` element as mentioned earlier. It can be used recursively, i.e. you can have a `<section>` in a `<section>` in a `<section>...`. This type has the advantage that you can move entire subtrees up or down the hierarchy without having to change the tags.
- Then there's the `<sect1>`, `<sect2>` ... `<sect5>` range. These elements must be properly nested, with `<sect1>` at the top, `<sect2>` within `<sect1>` etc. You cannot put a `<sect3>` directly in a `<sect1>`. This is less flexible than `<section>`, but in practice it rarely hurts. After all, the same “rigidity” applies to `<set>`, `<book>` and `<chapter>` and we can live with that, too.

Note

In early versions of this guide, the `<sectN>` series was recommended for presentational reasons. Due to improvements in the stylesheets, this is no longer an issue. Pick whichever you want.

Appendices

You can add one or more `appendix` elements after the last chapter in a book, or after the last section in an article. Appendices can contain just about everything that a `section` can contain, including other sections.

Example structure

The following example gives you an idea of how to structure your document:

```
<?xml version="1.0" encoding="UTF-8"?>
<book id="usersguide">
  <bookinfo>
    <title>Firebird Users Guide</title>
    <author>
      <firstname>William</firstname>
```

```
<surname>Shakespeare</surname>
</author>
<edition>25 January 2006 - Document version 1.2</edition>
</bookinfo>

<chapter id="usersguide-intro">
  <title>Introduction</title>
  <para>Hello! This is the introductory text to the Firebird
    Users Guide.</para>
</chapter>

<chapter id="usersguide-download-install">
  <title>Downloading and installing Firebird</title>
  <para>In this chapter we'll demonstrate how to download and
    install Firebird.</para>
  <section id="usersguide-download">
    <title>Downloading Firebird</title>
    <para>To download Firebird from the Internet, first go to the
      following URL: etc. etc. etc.</para>
    ...more paragraphs, possibly subsections...
  </section>
  <section id="usersguide-install">
    <title>Installing Firebird</title>
    <para>Installing Firebird on your system goes like this:
      etc. etc.</para>
    ...more paragraphs, possibly subsections...
  </section>
</chapter>

...more chapters...

<appendix id="usersguide-dochist">
  <title>Document history</title>
  ...to be discussed later!

<appendix id="usersguide-license">
  <title>License notice</title>
  ...to be discussed later!
</book>
```

Some points to note

- First, notice again that attribute values must always be quoted. (But if you fill them in in an attribute editor, don't insert quotes: the editor will take care of them.)
- As you can see in the example, chapters and sections can start directly with one or more `para` elements. But once you include sections in a chapter, or subsections in a section, you can't add any more `para` elements after them – only within them. Good DocBook-aware XML editors simply won't let you do such a thing, but if you type your DocBook XML by hand this is something you need to be aware of.
- If you use an XML editor, chances are that you rarely have to create `para` elements explicitly. For instance, if I insert a chapter or a section in XMLMind XML Editor, a first – empty – `para` is automatically created. And when I type text in a paragraph and hit **ENTER**, that paragraph is automatically closed with a `</para>` and a next one created.

[Skip the rest of the elements subsections](#) if you already know everything about DocBook elements.

Lists

DocBook offers various list elements, of which the following are used frequently:

itemizedlist

An `itemizedlist` is used to enumerate items whose order is not (very) important:

```
<itemizedlist spacing="compact">
  <listitem><para>Oranges are juicy</para></listitem>
  <listitem><para>Apples are supposed to be healthy</para></listitem>
  <listitem><para>Most people find lemons way too sour</para>
  </listitem>
</itemizedlist>
```

The items in the list are generally marked with a bullet in the rendered output docs:

- Oranges are juicy
- Apples are supposed to be healthy
- Most people find lemons way too sour

If you leave out the `spacing` attribute, it will default to `normal`, which means that vertical whitespace (usually one line's height) will be inserted between the listitems.

orderedlist

Use an `orderedlist` when you want to stress the order of the entries:

```
<orderedlist spacing="compact" numeration="loweralpha">
  <listitem><para>Sumerians 3300 BC – 1900 BC</para></listitem>
  <listitem><para>Assyrian Empire 1350 BC – 612 BC</para></listitem>
  <listitem><para>Persian Empire 6th century BC – 330 BC</para>
  </listitem>
</orderedlist>
```

By default, arabic numerals (1, 2, 3, ...) will be placed before the items, but you can change this with the `numeration` attribute. Output:

- a. Sumerians 3300 BC – 1900 BC
- b. Assyrian Empire 1350 BC – 612 BC
- c. Persian Empire 6th century BC – 330 BC

procedure

A procedure is often rendered like an `orderedlist`, but the semantics are different: a procedure denotes a sequence of *steps* to be performed in a given order:

```
<procedure>
  <step><para>Pick the lock</para></step>
  <step><para>Rob the house</para></step>
  <step><para>Get arrested</para></step>
</orderedlist>
```

Here's how the above example is rendered:

1. Pick the lock

2. Rob the house
3. Get arrested

Within a step you can include a `substeps` element, which in turn contains more steps.

variablelist

A `variablelist` is made up of `varlistentry`s, each of which contains a term followed by a `listitem`:

```
<variablelist>
  <varlistentry>
    <term>Tag</term>
    <listitem>
      <para>A piece of text enclosed in angle brackets</para>
    </listitem>
  </varlistentry>
  <varlistentry>
    <term>Element</term>
    <listitem>
      <para>A start tag, a matching end tag, and everything in
        between</para>
    </listitem>
  </varlistentry>
  <varlistentry>
    <term>Content of an element</term>
    <listitem>
      <para>Everything between the matching tags</para>
    </listitem>
  </varlistentry>
</variablelist>
```

The list you are reading right now, enumerating the different types of lists, is a `variablelist` with the element names (`itemizedlist`, `orderedlist`, etc.) as terms. The next section – *Links* – also consists of one introductory sentence followed by a `variablelist`.

Links

You can create hyperlinks to targets in your own document, in another document in the set, or on the Internet.

link

`link` is the generic element to point to another location in the document or set. The `linkend` attribute must always be present; its value should be the `id` of the element you link to (the *link target*).

```
Click <link linkend="docwritehowto-introduction">here</link> to jump
to the introduction.
```

In the rendered document, the word “here” will be *hot text*, that is: a clickable link pointing to the introduction:

Click [here](#) to jump to the introduction.

Caution

Although you can use `link` to point to any element in the entire set, you should only do so if the link target will be in the same PDF document as the link itself. The HTML version is fully hyperlinked, but links in the PDF rendering don't work across documents. Our PDFs typically contain one book or article; if the target lies outside the current document, use a `ulink` instead (see below).

ulink

Use a `ulink` to link to an Internet resource. The `url` attribute is mandatory:

```
Click <ulink url="http://docbook.org/tdg/en/">this link</ulink> to
read The Definitive Guide on DocBook.
```

The words “this link” will be rendered as a hyperlink to `http://docbook.org/tdg/en/`, like this:

Click [this link](http://docbook.org/tdg/en/) to read The Definitive Guide on DocBook.

email

You can make an email link with a `ulink`, but it's easier to use the `email` element. This will show the email address as a clickable link in the output. This piece of XML:

```
Send mail to
<email>firebird-docs-request@lists.sourceforge.net</email> to
subscribe.
```

results in the following output:

Send mail to firebird-docs-request@lists.sourceforge.net to subscribe.

If you want the hot text to be different from the email address itself, use a `ulink` with a `mailto:` URL.

Warning

If you include links to email addresses – whether with `email` or with `ulink` – or even if you only *mention* them in your text, and your document is subsequently published on the Internet, these email addresses will be exposed to harvesting robots used by spammers. This will likely increase the amount of spam sent to such addresses. Always make sure the owner of the address agrees before publishing it!

anchor

An anchor is an empty element marking an exact spot in the document. It doesn't show up in the text that your readers see, but it can be used as a link target. This is useful if you want to link to a place somewhere in the middle of a long paragraph:

```
<para id="lost-at-sea">
  Blah blah blah...
  and some more...
  and then some...
  Now here's an interesting place in the paragraph I want to be able
  to link to:
  <anchor id="captain-haddock"/>There it is!
  Paragraph drones on...
  and on...
  and on...
</para>
```

Having placed the anchor, you can create a link to it:

```
<link linkend="captain-haddock">Go to the interesting spot</link> in  
that long, long paragraph.
```

If your link targets a short element, or the beginning of an element, it's easier to give the target element an id and use that as linkend.

Program listings, screens, literal layout, and examples

programlisting

If you include code fragments in your doc, put them in a `programlisting` element. Everything you type within a `programlisting` will be rendered verbatim, including line breaks, spaces etc. Also, a fixed-width font will be used in the rendered documents. The term “program listing” is to be interpreted loosely here: you should also use this element for SQL fragments and DocBook XML examples. This guide – and especially the section about elements, which you are reading now – is littered with `programlistings`, so you already know what they look like:

Programlistings are rendered like this.

Important

In `programlistings` you should limit the line length to around 70 characters, otherwise the text will run off the right edges of the rendered PDF documents. The same goes for other layout-preserving elements like `screen`, `literallayout`, etc.

screen

Use a `screen` element to show what a user sees or might see on a computer screen in text mode, or in a terminal window. Here too, your layout will be preserved and a fixed-width font used, but the semantics are different. It may or may not look different from a `programlisting` in the output. Here's a short example, showing what happens if you try to build a non-existing target in the manual tree:

```
<screen>  
D:\Firebird>manual_incl_howto\src\build>build ugh  
java version "1.4.2_01"  
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_01-b06)  
Java HotSpot(TM) Client VM (build 1.4.2_01-b06, mixed mode)  
  
Buildfile: build.xml  
  
BUILD FAILED  
Target `ugh' does not exist in this project.  
</screen>
```

And this is how it's rendered:

```
D:\Firebird>manual_incl_howto\src\build>build ugh  
java version "1.4.2_01"  
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_01-b06)  
Java HotSpot(TM) Client VM (build 1.4.2_01-b06, mixed mode)  
  
Buildfile: build.xml
```

```
BUILD FAILED
Target `ugh' does not exist in this project.
```

literallayout

`literallayout`, like `screen` and `programlisting`, keeps your layout intact, but it usually doesn't change the font – unless you set the `class` attribute to `monospaced`. It's also more general than the previous two in the sense that there's no meaning attached to its content: you can put any kind of text here of which you want to preserve the layout.

Example source:

```
<literallayout>
The Sick Rose

Oh Rose, thou art sick!
The invisible worm
That flies in the night,
In the howling storm,

Has found out thy bed
Of crimson joy,
And his dark secret love
Doth thy life destroy.

    – William Blake
</literallayout>
```

Output:

The Sick Rose

Oh Rose, thou art sick!
The invisible worm
That flies in the night,
In the howling storm,

Has found out thy bed
Of crimson joy,
And his dark secret love
Doth thy life destroy.

— William Blake

example

An `example` is a formal example with a title. It is usually given an `id` so it can be referred to from other places in the document. An index of examples is built automatically when the document is rendered. You'll often find `programlisting`'s in an `example`, but it may also contain `screen`'s, `para`'s, lists, etc.

Here's an example of an example:

```
<example id="docwritehowto-sql-example">
  <title>An SQL example</title>
  <para>With this command you can list all the records in the COUNTRY
    table:</para>
  <programlisting>SELECT * FROM COUNTRY;</programlisting>
```

```
</example>
```

In the output this will look like:

Example 1. An SQL example

With this command you can list all the records in the COUNTRY table:

```
SELECT * FROM COUNTRY;
```

If you want an example without a mandatory title, use an `informalexample`. Informal examples are also left out of the examples index.

Tables

If you have ever made an HTML table for a website, you won't have much difficulty creating tables in DocBook. There are differences though, and DocBook tables are vastly richer.

A table consists of a title and one or more `tgroups` – usually one. The `tgroup` element has one mandatory attribute: `cols`. You must set this attribute to the number of columns in the `tgroup`. Within a `tgroup` you can place `thead`, `tfoot` and `tbody` elements. Each of these has one or more `rows`, which in turn have as many `entry`s (cells) as you have specified in the `cols` attribute. (You can combine cells by creating spans, but we won't go into that here.)

So much for the basic structure. Now we'll show you an example; first in DocBook XML source text, and then the resulting table in the rendered output document. Don't worry about the `<colspec>`s for now; these are non-mandatory subelements used for finetuning.

```
<table id="docwritehowto-table-dboftheyear">
  <title>LinuxQuestions.org poll: Database of the year 2003</title>

  <tgroup cols="3">
    <colspec align="left" colname="col-dbname" colwidth="2*" />
    <colspec align="right" colname="col-votes" colwidth="1*" />
    <colspec align="right" colname="col-perc" colwidth="1*" />

    <thead>
      <row>
        <entry align="center">Database</entry>
        <entry align="center">Votes</entry>
        <entry align="center">Percentage</entry>
      </row>
    </thead>

    <tfoot>
      <row>
        <entry>Total</entry>
        <entry>1111</entry>
        <entry>99.99</entry>
      </row>
    </tfoot>

    <tbody>
      <row>
        <entry>MySQL</entry>
```

```

    <entry>405</entry>
    <entry>36.45</entry>
  </row>
  <row>
    <entry>Firebird</entry>
    <entry>403</entry>
    <entry>36.27</entry>
  </row>

  ... 5 more rows not shown here ....

</tbody>
</tgroup>
</table>

```

And here's the resulting table:

Table 2. LinuxQuestions.org poll: Database of the year 2003

Database	Votes	Percentage
MySQL	405	36.45
Firebird	403	36.27
Postgres	269	24.21
Oracle	25	2.25
Berkeley DB	4	0.36
Sybase	3	0.27
DB2	2	0.18
Total	1111	99.99

By the way, these are the actual results of a real poll at LinuxQuestions.org. As you can see, if only three more people had voted for Firebird we would have won. If you know who these three persons are, please report them to our Chief Inquisitor. He would like to have a little, er... *talk* with them :-)

Tables are automatically indexed. An `informaltable` has the same structure as a `table` but doesn't require a title and is not included in the index. If you want to nest tables, either use a `table/informaltable` *within* an entry, or an `entrytbl` *instead of* an entry.

Tables have many more features than shown here, but we'll leave those for you to explore.

HTML tables

DocBook versions 4.3 and up also allow you to fill a table the HTML way, with `trs` instead of `rows`, and `td/th` instead of `entry` elements. Why would you want to do that? There are two situations where it may be advantageous to use an HTML table:

- You already have the HTML table available, and you'd rather not spend time converting it;

- You want to use several different background colors in the table. This can be done in a DocBook table too, but only with *processing instructions* – one for each target for every child element that needs an explicit color. In an HTML table you can use the children's `bgcolor` attributes.

An HTML table can't have `tgroups`; you put the `tr`s either directly in the table or in `thead`/`tfoot`/`tbody` elements which are direct children of the table. Also, it has a `caption` instead of a `title`. (An `informaltable` has neither `caption` nor `title`.)

Here is the source of an HTML table:

```
<table bgcolor="blue" border="1">
  <caption align="bottom">An HTML-style table</caption>

  <tr bgcolor="#FFE080">
    <th>First column</th>
    <th bgcolor="#FFFF00">Second column</th>
  </tr>
  <tr align="center">
    <td bgcolor="orange" colspan="2">Table cell spanning two
      columns</td>
  </tr>
  <tr>
    <td bgcolor="#00FFC0">Yes, here I am</td>
    <td align="right" bgcolor="#E0E0E0" rowspan="2" valign="bottom">And
      there I go!</td>
  </tr>
  <tr>
    <td bgcolor="#FFA0FF">Another row...</td>
  </tr>
</table>
```

And here's the result:

First column	Second column
Table cell spanning two columns	
Yes, here I am	And there I go!
Another row...	

Table 3. An HTML-style table

Not all HTML table elements and attributes are supported by our stylesheets. For instance, properties specified in `col` and `colgroup` elements won't be picked up. Specify them in the `td`/`th` elements instead – or extend the stylesheets!

Note

In XMLMind, you can only create an HTML table from the menu opened by the “Add table” button on the toolbar. From the Edit pane you can only add regular DocBook tables.

PDF rendering of large tables

DocBook tables belong to a group called *formal elements*. Formal elements are included in automatically generated indices (list of tables, list of figures etc.); if a formal element doesn't have an `id` attribute, the stylesheets

assign one. The templates that generate the XSL-FO output (this is the intermediate stage for the PDF) also give each formal object the attribute `keep-together.within-page="always"` to prevent page breaks to occur within the object. This is usually fine, but what if the object doesn't fit on one page? Until recently, we used Apache FOP 0.20.5 to render the XSL-FO output to PDF. This processor simply ignored the `keep-together` attribute if the object was too large. But the current version (0.93 or higher) *always* enforces it. The result is that if the object is too large, it is truncated (or wrecked in some other way) to make it fit on the page. This is a feature, not a bug, so there's no use complaining about it.

There are two ways to work around this problem if a table grows too large to fit on a single page:

1. If the table doesn't need a title and you don't mind that it won't be included in the List of Tables, use an `informaltable` instead.
2. Insert a *processing instruction* at the beginning of the table:

```
<table frame="all" id="ufb-about-tbl-features">
  <?dbfo keep-together='auto'?>
  <title>Summary of features</title>
```

In XMLMind, this is done as follows:

1. Place the cursor somewhere in the title or select the entire title element.
2. Choose *Edit -> Processing Instruction -> Insert Processing Instruction Before* from the menu. A green line will appear above the title.
3. Type `keep-together='auto'` on that line.
4. With the cursor still on the green line, choose *Edit -> Processing Instruction -> Change Processing Instruction Target* from the menu. A dialogue box pops up.
5. In the dialogue box, change `target` to `dbfo` and click OK.

Of course you can do the same for smaller tables if you want them to be breakable. The opposite instruction, `<?dbfo keep-together='always'>`, will prevent page breaks in `informaltables`. Make sure that the element fits on one page before using this!

Images

To include an image, use a `mediaobject` containing an `imageobject` containing an `imagedata` element:

```
<mediaobject>
  <imageobject>
    <imagedata align="center" fileref="images/services.png"
      format="PNG" />
  </imageobject>
</mediaobject>
```

You may wonder why you need three nested elements to include a simple image. There's a good reason for this, but I'm not going to tell you ;-) — it's of no concern to us. All we have to know is that this is how it's done.

Regardless of the location of the image relative to the DocBook source, the `fileref` should *always* be of the form `images/filename.ext`. This is because, both for the HTML and the FO output, the image files will be

copied from their source locations to a subdirectory called `images` under the output directory. (The FO output is an intermediate form. Once converted to PDF, the image will be included in the file itself.)

If the `fileref` is not “correct” from the source file's point of view, you won't see the image in XMLMind. If this bothers you, create a symlink to the images folder (Linux) or copy the images folder into the same folder as the source file (Windows). Creating a shortcut under Windows doesn't seem to do the trick. Only do this in your local copy – don't commit duplicated image folders to CVS!

A `mediaobject` is formatted as a separate block. If you want the image inlined with the text, use an `inline-mediaobject` instead; the nested elements remain the same.

Note for translators

Translators: Any images that you don't edit or replace by a localised version should not be copied into your language set. As from January 2006, the build tools first look in your language's image folder (e.g. `manual/src/docs/firebirddocs-fr/images`), and after that in `manual/src/docs/firebirddocs/images`. So, if you use the original image, there's no need to waste CVS space by duplicating it.

The same behaviour applies to other base sets: if an image referenced from, say, the Spanish Release Notes sources is not in `rlsnotes-es/images`, the one in `rlsnotes/images` is used. It doesn't work *across* base sets, though.

Admonitions

DocBook has several tags to mark a block of text as a note, a warning, a tip, etc. In the output documents such blocks typically appear indented, and marked with an icon or a word to denote their purpose. These tags are, in alphabetical order:

`<caution>`, `<important>`, `<note>`, `<tip>`, and `<warning>`

I will give you a `<tip>` as an example; the others are used in exactly the same way:

```
<tip>
  <para>If you insert a caution, important, note, tip, or warning
    element in your text, don't start it with the word caution,
    important, note, tip, or warning, because these words are usually
    automatically generated by the rendering engine.</para>
</tip>
```

And this is the result:

Tip

If you insert a `<caution>`, `<important>`, `<note>`, `<tip>`, or `<warning>` element in your text, don't start it with the word `caution`, `important`, `note`, `tip`, or `warning`, because these words are usually automatically generated by the rendering engine.

You may have noticed that the words `caution`, `important` etc. look different from the rest of the tip's text. How come? Well, to tell you the truth, I've surrounded them with special tags (first with `<sgmltag>`s, the second time with `<literal>`s) to make them look like that. But this made the source XML look very noisy, so I decided to remove those tags from the example source I presented to you.

You can optionally give the admonition a `title`. If you don't, a default header (in the document language) will be generated in the output.

If you want to set off a block of text from its surroundings without marking it as a tip or whatever, use a `<blockquote>`.

Paragraph headers

If you want a paragraph header or title without creating a subsection, there are a few possibilities.

bridgehead

A `bridgehead` is a free-floating title between paragraphs, not associated with the start of a chapter or section. The `renderas` attribute determines how it will be rendered.

```
<para>You may remember that Mr. Hardy started with this firm as
  elevator boy and with grim determination worked his way up to
  the top. And after the wedding today he becomes General Manager
  of this vast organisation.</para>

<bridgehead renderas="sect5">Mr. Laurel's comments</bridgehead>

<para>We also spoke to his lifetime friend and companion Mr. Laurel.
  Mr. Laurel says that after viewing the situation from all sides,
  he is thoroughly reconciled to the fact that the moving picture
  industry is still in its infancy. Mr. Laurel also states that
  technology, whilst it may appear to be the center of all—</para>
```

The above source is rendered as:

You may remember that Mr. Hardy started with this firm as elevator boy and with grim determination worked his way up to the top. And after the wedding today he becomes General Manager of this vast organisation.

Mr. Laurel's comments

We also spoke to his lifetime friend and companion Mr. Laurel. Mr. Laurel says that after viewing the situation from all sides, he is thoroughly reconciled to the fact that the moving picture industry is still in its infancy. Mr. Laurel also states that technology, whilst it may appear to be the center of all—

You are free in your choice of `renderas` level, but the logical choice would normally be the current section level plus (at least) one.

formalpara

A `formalpara` is a paragraph with a title. Our stylesheets render the title as a run-in head.

```
<formalpara>
  <title>Motherly love:</title>
  <para>This is the love your mother has for you, not to be
    confused with brotherly or otherly love.</para>
</formalpara>
```

In the output this looks like:

Motherly love: This is the love your mother has for you, not to be confused with brotherly or otherly love.

A period will be appended to the title, unless it already ends with a punctuation character.

Various inline elements

To conclude the subsection on DocBook elements I will now briefly introduce a number of *inline elements*. They are called “inline” because they don't interrupt the flow of the text. If I use e.g. an `emphasis` element:

```
Don't <emphasis>ever</emphasis> call me fat again!
```

the result is this:

Don't *ever* call me fat again!

The word “ever” is emphasized, but it keeps its place in the sentence. We've already encountered some inline elements before: the various link types. Other elements – like `table`, `warning`, `blockquote` and `program-listing` – are always displayed as a block, set apart from the surrounding text (even if you “inline” them in your XML source). Not surprisingly, these are called *block elements*. Block elements often contain inline elements; the reverse is not possible.

OK, let's get started with those inline elements. I'll include examples – both XML source and rendered output – for most of them:

filename – command – application – envar

Use the `filename` tag to mark file names in the broadest sense. Attributes can optionally indicate that the file is a header file, a directory, etc.

```
Place your doc in the <filename  
class="directory">src/docs/firebirddocs</filename> subdirectory.
```

The output reads:

Place your doc in the `src/docs/firebirddocs` subdirectory.

`command` and `application` are both used for executable programs. `command` is usually chosen for smaller programs and internal commands; its content should be the exact command as given on a command line; `application` is generally used for bigger programs and need not be the name of the executable file. Both can refer to the same program:

```
Type <command>netscape&amp;</command> in a terminal window to start  
<application>Netscape Navigator</application>.
```

This is rendered as:

Type **netscape&** in a terminal window to start Netscape Navigator.

`envar` denotes an environment variable.

subscript – superscript

These two do the expected thing:

```
After inventing the formula e = mc<superscript>2</superscript>, I  
really felt like a glass of liquid H<subscript>2</subscript>O !
```

Output: After inventing the formula $e = mc^2$, I really felt like a glass of liquid H₂O !

varname – constant – database

The use of `varname` and `constant` should be obvious. The `<database>` tag is not only meant for databases, but also for database objects:

```
The <database class="table">COUNTRY</database> table has two fields:
<database class="field">COUNTRY</database> and
<database class="field">CURRENCY</database>.
```

Output: The COUNTRY table has two fields: COUNTRY and CURRENCY.

function – parameter – returnvalue

These three speak for themselves, I trust.

```
The <function>log</function> function takes parameters
<parameter>a</parameter> and <parameter>b</parameter>.
```

Output: The `log` function takes parameters `a` and `b`.

prompt – userinput – computeroutput

`prompt` is used for a string inciting the user to enter some text; `userinput` refers to text entered by the user (not necessarily at a prompt!); `computeroutput` is text displayed by the computer:

```
Type <userinput>guest</userinput> at the <prompt>login:</prompt>
prompt and the server will greet you with a <computeroutput>Welcome,
guest user</computeroutput>.
```

Output: Type **guest** at the `login:` prompt and the server will greet you with a `Welcome, guest user`.

keycap

The text on a keyboard key, or its common name:

```
Hit the <keycap>Del</keycap> key to erase the message, or
<keycap>SPACE</keycap> to move on.
```

Output: Hit the **Del** key to erase the message, or **SPACE** to move on.

sgmltag

This element is used extensively throughout this guide: it marks SGML *and* XML tags, elements, attributes, entities etc.:

```
If it concerns a directory, set the
<sgmltag class="attribute">class</sgmltag> attribute of the
<sgmltag class="element">filename</sgmltag> element to
<sgmltag class="attvalue">directory</sgmltag>.
```

Output: If it concerns a directory, set the `class` attribute of the `filename` element to `directory`.

Other possible values for `sgmltag.class` are: `starttag`, `endtag`, `emptytag`, and `genentity` (for an entity).

emphasis – citetitle – firstterm

Use `emphasis` to stress words in general, `citetitle` for book titles etc., and `firstterm` if you introduce a new word or concept to your readers:

```
We use <firstterm>DocBook XML</firstterm> for our Firebird
documentation. A short introduction follows;
<emphasis>please</emphasis> read it carefully! If you want to know
more about the subject, buy <citetitle>DocBook – The Definitive
Guide</citetitle>.
```

Output: We use *DocBook XML* for our Firebird documentation. A short introduction follows; *please* read it carefully! If you want to know more about the subject, buy *DocBook – The Definitive Guide*.

quote – literal

Use `quote` for an inline quotation (as opposed to a `blockquote`). Quotation marks will be inserted automatically. Using `quote` instead of typing the quote characters yourself (which is also perfectly legal) has the advantage that we can alter the type of quotation marks through stylesheets if we want to. Also, quotes differ per language:

```
<para>An <quote lang="en">English quote</quote>
and a <quote lang="fr">French quote</quote>.</para>
```

Output: An “English quote” and a « French quote ».

Please note that you shouldn't use the `lang` attribute on `quotes` in your own docs. Your root element's `lang` attribute will ensure that the right type of quotes are used. If someone translates your document – and changes the root `lang` attrib – it will be rendered with the quotation marks for the target language. Of course I had to use the attribute here to show the difference, and to make sure that the different quotation marks survived any translation.

A `literal` is a word or text fragment to be taken literally. It is a rather general element, often used to make certain words stand out typographically:

```
At all costs avoid using the word <literal>humongous</literal> in
your documentation.
```

Output: At all costs avoid using the word *humongous* in your documentation.

Should you always use these inline elements wherever you can? Well, if you do, you will certainly make your document richer; you'll make it easier to scan for filenames for instance, or to generate an index of all the applications mentioned in your document. On the other hand, there are so many of these semantic elements (in fact we've only discussed a *few* here) that if you apply them everywhere you can, you'll probably wind up in a straightjacket before you can finish your doc. This is not what we want: if you really have to go mad, please do so *after* you've committed your document :-)

So, as a general advice: go a bit easy on those inlines; use them wherever you think it makes sense, but don't overdo it.

To wrap up the elements

You may have noticed that in the rendered documents (you're reading one now, unless you opened the XML version) many different elements have the same appearance: a `filename`, a `literal` and an `application` may have the exact same typography; the same goes for `emphasis`, `firstterm` and `citetitle`.

So what's the point of all these different tags? Why not use just a few, like `emphasis` and `literal`, if they're going to look the same anyway? Well, there are two very good reasons not to:

- First, if we dropped most of our inlines in favor of say, `emphasis` and `literal`, the semantics would be lost. Remember that DocBook XML is all about structure and semantics. `firstterm` and `citetitle` may *look* the same as `emphasis` once rendered, but they *are* not the same thing. The XML source knows that, even if it doesn't always show. This information is useful, and we don't want to lose it.
- Further, we can adapt our stylesheets for each type of element individually. As soon as we decide that a `firstterm` should look different from a `citetitle`, we can arrange for that – but *only* if they are indeed marked with different tags, not if they are both `emphasis`'s in the XML source.

This concludes the sections on DocBook. With the knowledge presented above, you should now be able to author DocBook XML documents for the Firebird project. Of course if you use a dedicated XML editor – which, again, is highly advisable – you must also consult its documentation to learn how to use it; that's one thing this guide doesn't cover.

Language and style

After the flood of DocBook information in the previous sections, we now turn our attention to some other important docwriting aspects: language and style (in this section), and copyrights (in the next section).

Language

The Firebird community is a very diverse one, and made up of people with many different mother tongues. If you write your documentation in a language other than your own, you'll probably make some mistakes. This is not catastrophic, but you should at least try to reduce the number of errors. Some strategies to help you with this are:

- Use a dictionary! Simple, effective, and blissfully non-hightech.
- When hesitating between two spellings of a word, or between several possible versions of an expression, google for the alternatives and look at their frequencies. Also follow some of the result links to see how native speakers use the word or expression in their texts.
- Have a native speaker look over your text and correct it where necessary.

Style

Don't expect a Style Guide here – I wouldn't know how to write one anyway. Just some guidelines and tips:

- Try to write in plain, everyday language wherever possible. Avoid difficult words if there's a familiar, simple alternative.
- Avoid long sentences (over 25 words) if you can; especially avoid two or more long sentences immediately after each other.
- Be careful with constructs like double or triple negatives (“I can't deny that I'm not displeased”) and passive voice (“Care should be taken...”). You don't have to avoid them at all costs, but they can make a sentence harder to understand. To prevent that, use the positive (“I am pleased”) and the active voice (“Take care...”).

- Use lists to enumerate a number of parallel items, for instance:
 - A collection of hints and tips.
 - A sequence of examples (like this one).
 - Steps to be followed in a procedure.
 - Alternative solutions to a problem.

But if there's only a small number of short items, use a plain sentence instead: “My mother loves three men: John, Dick, and Dave.”

- Don't overuse exclamation marks. Never use multiple exclamation marks or question marks. This is annoying!!!! Don't you agree???

Docwriter's block

Sometimes you know what you want to write, and you have all the words ready, but you can't get the sentence started – you just don't get it to *flow*. This is very frustrating and it can sometimes block the advance of your text for many minutes. And it's all the more frustrating because you *do* know what you want to tell your readers, but you don't seem to be able to produce a decent sentence. After many painful experiences of this kind, I've developed the following strategy (not that I think I'm the first):

1. Write down what you have to say in loose sentences and chunks of words. Never mind about style, never mind if it looks ugly. Just write down what you want to tell the reader; make sure it's all there, and in the right order. If, while doing this, you notice that you feel unsure about something, include a remark at exactly that point. Make your remarks stand out from the surrounding text, e.g. <<like this>> or !LIKE THAT!

This may result in a text like:

CVS means Concurrent Versions System (<<check!>>). Purpose: managing versions of software. You can use it alone or with a group. You need a CVS client to use it. A CVS client is a program with which you can access a CVS repository (<<explain this term?>>). To find out if a CVS client is installed on your system, type “cvs” on the command line. If it's not there, go to this URL to download it.... [etc., etc.]

2. If you have included any remarks, handle them first. *Check* if CVS really means Concurrent Versions System (it does). *Decide* whether you should really explain the term “CVS repository” at this point (you should).
3. Now, go over the paragraph again and try to make the text flow more naturally wherever you can. Chances are that this will be a lot easier than you expected!
4. If it still looks a little clumsy, never mind – better clumsy and clear than smooth-flowing and fuzzy. Maybe you can revisit this passage later and see if you can nice it up some more.

This approach works well for me. So if you're stuck in this way, try it out; hopefully it will help you too.

Copyright issues

Many people find legal issues boring, but this is an important section. Please read it thoroughly.

Using material written by others

As we write our manuals, we can consult all kinds of other documentation – and so we should, because we want to achieve the best possible result. Any information we find in publicly available third-party manuals, user's guides, tutorials etc. can be freely used in our own docs, but it is important not to confuse *information* with *literal text*. We cannot copy-and-paste text from other works into our own documentation, unless the author explicitly permits us to do so.

If you would like to use a piece of text written by somebody else, check the copyright notice of the work in question. If there isn't one, the work is automatically copyrighted under the Berne convention and you must assume that it's *illegal* to copy it – even partially. This is also true if the work is freely available! Not having to pay for a document does not imply that you can freely copy portions of text and republish them in a work of your own.

Borland InterBase manuals

The Borland InterBase 6 beta docs – although free – are not part of the InterBase package that was open-sourced in July 2000. We have asked Borland several times if we could use these docs “as if they fell under the InterBase Public License”, but they didn't even bother to answer. So feel free to use this documentation set as a source of information, but don't copy text from it.

PostgreSQL docs

PostgreSQL is another major open source database, with (not surprisingly) many similarities to Firebird, but also many differences. Depending on the kind of documentation you are going to write, it may be beneficial to base it on existing PostgreSQL docs. Be aware though that if you use PostgreSQL material, you **MUST** include their copyright notice in your document!

The PostgreSQL documentation homepage is here:

<http://www.postgresql.org/docs/>

The most recent PostgreSQL license is currently at:

<http://www.postgresql.org/about/licence>

One nice thing about the PostgreSQL docs is that they are authored in DocBook, just like ours. However, they use DocBook SGML instead of XML, so some tweaking may be necessary. The DocBook SGML sources can be found here:

<http://developer.postgresql.org/cvsweb.cgi/pgsql-server/doc/src/sgml/>

Or check out the entire CVS tree, docs and all. For instructions, go to:

<http://developer.postgresql.org/docs/postgres/cvs.html>

Your copyright and the PDL

If you contribute to the Firebird documentation subproject, your work will be included in the Open Source repository at SourceForge. In January 2005, the Firebird doc team decided to release the documentation it develops under the *Public Documentation License*. Licensing your work under the PDL means that you retain the copyright, but you grant others certain rights:

- *Free use*: everyone may use and distribute your work, for free or for money, as long as the license notice is kept intact.
- *Right to modify*: everyone may modify and redistribute your work, as long as any modified versions are PDL-licensed too, the original license notice is kept intact, and the modifications are documented.
- *Larger works*: everyone may incorporate your documentation (modified or not) in a larger work. The larger work as a whole need not be released under the PDL, but the license requirements must be fulfilled for the PDL-licensed parts.

What's so nice about the PDL is that it provides the same rights and restrictions on the usage of our docs as the IPL and IDPL (Firebird's code licences) do for the Firebird source code. For the complete license text, see the links in the License Notice below; the DocBook source is in `src/docs/firebirddocs/licenses.xml`

How to apply the PDL to your work

In order to release your work under the PDL, add an appendix titled *License Notice*, with this text:

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the "License"); you may only use this Documentation if you comply with the terms of this License. Copies of the License are available at <http://www.firebirdsql.org/pdfmanual/pdl.pdf> (PDF) and <http://www.firebirdsql.org/manual/pdl.html> (HTML).

The Original Documentation is *_TITLE OF THE WORK_*.

The Initial Writer of the Original Documentation is *_INITIAL AUTHOR'S NAME_*.

Copyright (C) *_YEAR(S)_*. All Rights Reserved. Initial Writer contact(s): *_EMAIL OR OTHER CONTACT ADDRESS(ES)_*.

Everything that looks *_LIKE THIS_* must of course be replaced. If you are not the original author, you should leave his or her notice intact and append the following:

Contributor(s): *_NAME(S) + SHORT DESCRIPTION (COUPLE OF WORDS) OF CONTRIBUTION_*.

Portions created by *_CONTRIBUTOR'S NAME_* are Copyright (C) *_YEAR(S)_*. All Rights Reserved. Contributor contact(s): *_EMAIL OR OTHER CONTACT ADDRESS(ES)_*.

There may be several Contributor's sections in the License Notice.

Including a Document History

If your contribution consists of more than a simple change or addition in one spot, also include an appendix called *Document History* before or after the License Notice. If such an appendix already exists, always enter a description of your modification(s) in it. Please note that even if there's a Document History, you must still add a contributor's section to the License Notice – but then you can fill in “see Document History” in place of the short description.

If you're the original author, it's also perfectly OK to include a Document History in the first version of a document, to serve as a starting point for future revisions. See the first `revision` element in the example below.

Centerpiece of the Document History is the `revhistory` element with its children:

```
<revhistory>
  <revision>
    <revnumber>1.0</revnumber>
    <date>12 Sep 2005</date>
    <authorinitials>PV</authorinitials>
    <revdescription>
      <para>First version</para>
    </revdescription>
  </revision>
  <revision>
    <revnumber>1.1</revnumber>
    <date>5 Dec 2005</date>
    <authorinitials>PV</authorinitials>
    <revdescription>
      <para>Added information on COALESCE</para>
      <para>Corrected some spelling errors</para>
    </revdescription>
  </revision>
</revhistory>
```

Please abbreviate the month name in the `date` element, as the date column in the PDF output is rather narrow.

Below is a Document History example (output view, not source!) that uses a `revhistory` element. Notice the referral to the CVS tree: we are legally obliged to identify and date all changes. But since CVS already does that, we can simply alert the user to it and give a less extensive but nicer-to-read history in the document itself.

The exact file history is recorded in the `manual` module in our CVS tree; see http://sourceforge.net/cvs/?group_id=9028

Revision History

1.0	2003	IBP	First publication of the free Quick Start Guide.
1.x	June 2004	IBP	Donated to Firebird Project by IBPhoenix.
2.0	2004	PV	Downgraded to Firebird 1.0 Added Classic vs. Superserver section. Reorganised and corrected Disk Locations Table. Added (new) screenshots. Updated and completed information on Control Panel applets. Added extra examples to “Expressions involving NULL”. Various other corrections and additions.

If you open the DocBook source of this Guide (`src/docs/firebirddocs/docwriting-howto.xml`) in your favourite XML editor, you can easily copy-and-paste the Document History and License Notice into your own document. Don't copy the examples given above; copy the real appendices at the end of the document, and edit them to fit your work.

A copyright notice at the start

License Notice and Document History both appear at the end of the document. If you want to make your copyright obvious right from the start, you may also include a short copyright notice in the document's `xxxinfo`, like this:

```
<bookinfo>
  <title...
  <author...
  <edition...
  <copyright>
    <year>2003</year>
    <year>2004</year>
    <holder>Tootsie Griff</holder>
  </copyright>
</bookinfo>
```

Such a notice does not replace the License Notice and/or Document History – it's an extra.

Attaching the entire Pubic Documentation License

Instead of providing the URL, you can also attach the entire PDL to your document. This may especially be useful if your work is a book or long article and you expect (or hope) that people will print it and distribute hardcopies. On a short document the PDL may be a little heavy, but it's your call.

You can get the PDL's DocBook source from `src/docs/firebirddocs/licenses.xml`. Please note that only the section with the license text itself (including the generic license notice) belongs to the PDL proper. The Introduction is not part of the license.

If you include the PDL in your document, you can fill in the blanks in section 5.2 of the license. But you may also leave them as they are (provided your name is in the License Notice) or just fill in “the Initial Writer” or “the Copyright holder”.

Translator's notices

Translating a document is a form of modification. So, as a translator, you should:

- List yourself as a Contributor in the License Notice, with a contribution description like e.g. "Translation into Russian". You may translate the License Notice into the target language if you wish, but you can also leave it in English or include it in both languages.
- Add a `revision` element – in the target language – to the `revhistory` in the Document History. For the `revnumber`, you use the number of the revision that you've translated, followed by a hyphen and your language code, e.g. “2.0-es” or “1.1-fr”:

```
<revhistory>
  ...previous revisions...
```

```
<revision>
  <revnumber>1.1</revnumber>
  <date>5 Dec 2005</date>
  <authorinitials>PV</authorinitials>
  <revdescription>
    <para>Added information on COALESCE</para>
    <para>Corrected some spelling errors</para>
  </revdescription>
</revision>
<revision>
  <revnumber>1.1-fr</revnumber>
  <date>13 Déc 2005</date>
  <authorinitials>AM</authorinitials>
  <revdescription>
    <para>Traduction en français</para>
  </revdescription>
</revision>
</revhistory>
```

- Add an `othercredit` element to the `xxxinfo` at the beginning of the document, like this:

```
<articleinfo>
  <title>Guía de NULL en Firebird</title>
  <author>
    <firstname>Paul</firstname>
    <surname>Vinkenoog</surname>
  </author>
  <othercredit>
    <firstname>V́ctor</firstname>
    <surname>Zaragoza</surname>
    <contrib>Traducción al castellano</contrib>
  </othercredit>
  <edition>22 de julio de 2005 - Versión de documento 2.0-es</edition>
</articleinfo>
```

The `contrib` element contains the same information as the contribution description in the License Notice, but it should always be in the target language.

Also notice the document version in the `edition` element – make sure it's the same as in the Document History.

Translating the PDL

You don't have to translate the PDL itself. But if you do:

- Add it as an independent document to your language's docset, in a book called *Licenses* (but translate “Licenses” into *your* language).
- In the translated Introduction to the PDL, explain that only the English version is legally binding, and include a link to the English version.
- In any License Notice where you link to the translated PDL, also provide a link to the original PDL and make clear that this is the one that's legally binding.

You can optionally also attach the translated PDL to the document itself, if you don't mind the extra load and bloat.

Adding your document to the manual module

When your doc is finished, and you have verified that it builds correctly, you want it added to the manual module. If this is your first contribution to the documentation project you'll probably have agreed with the coordinators that you first submit it to them for review, or that you temporarily put up the HTML version on a website so that it can be discussed on the list. After that – and maybe after some corrections are made – the document can be committed to the module. If you have commit rights you can do this yourself; if not, one of the coordinators will do it for you.

Asking for commit rights

To receive commit rights you first need a SourceForge user account. If you haven't got one, register at <http://sourceforge.net/account/register.php>. Then post a message to the firebird-docs mailing list stating your SF user name and asking to be added to the Firebird project. The manual subproject leader and several Firebird project admins follow the list; they will consider your request. As a general rule you should ask for commit rights *after* your first contribution, because the people who decide on your request need something to go by.

The following phrases currently all mean the same, by the way:

- Being a project member.
- Having commit rights.
- Having read-write access to the repository.

Dos and don'ts if you have received commit rights

Once you are accepted as a project member, you have write access to the entire Firebird repository, not only to the manual module. There is no technical barrier to keep you from committing changes to other modules – the `firebird2` core module for instance, or even the `CVSROOT` module where important project information is stored.

You may already have guessed that this is *NOT* the idea. Keep to the following rules:

- Don't *ever* commit to other modules unless the people in charge of those modules explicitly ask you to do so.
- Only commit work to the manual module if it concerns a task assigned to you. Even then, it's good practice to announce your changes and additions on the mailing list first, so the other doccers have a chance to comment on it. After all, this is a collective effort.
- If you think a new document or directory should be added, don't just create and commit it, but propose it on the list.

In practice, things may be a bit more relaxed than stated here, especially where it concerns your own tasks. We don't want you to feel unfree and you certainly shouldn't get the feeling that you have to ask permission for every minor change you make. But we do want you to act responsibly, and we want to know from each other what we are doing. Besides, keeping in touch with each other is often inspirational. Together we can make this thing work!

Committing your work

Even if you are a project member, you can only commit changes from a local copy if it was checked out with your SF login name. If you're still working with a copy you've checked out anonymously you must first make a fresh SSH checkout, and then re-apply your changes and commit them. Refer to the [Docbuilding Howto](#) if you don't remember how to perform an SSH checkout.

If some time has passed since your last checkout or update, perform an update before committing. This will get your local copy in sync with the repository and reduce the possibility of conflicts.

Once you are ready to commit, go to the manual directory. If you use command-line CVS, type:

```
cvs update -d [ only if you want to update first ]
```

```
cvs add path/to/mydocument.xml [ only if it concerns a new document not yet in CVS ]
```

```
cvs commit -m "Short informational message here"
```

After the `-m`, and within quotes, you type a short message about this commit, e.g. "Added new functions to API Reference" or "Errors in isql tutorial fixed".

Give your SF password when prompted, and all the changes you have made – including those in subdirectories – will be committed. Your CVS client knows which server to contact; this and other information is stored in the CVS subdirectories that were created upon checkout.

If you use another CVS client, refer to its documentation.

Important

After adding a new document, you must still perform a separate commit. This goes for command-line CVS and most (if not all) other CVS clients.

Publishing your document on the Firebird website

In order to publish your document, you first have to build the HTML and PDF output. This is documented in the [Firebird Docbuilding Howto](#). In the remainder of this section it is assumed that you have successfully built the HTML and PDF files.

Naming the PDF file

The build tools automatically name each file after the ID of the topmost DocBook element it contains. We don't change the names of the multi-page HTML output – these pages are primarily intended for online browsing, and changing even a single file name would immediately break a number of links contained in the other pages. But PDFs are often downloaded by the reader, and having files called `qsg2.pdf` or `ubusetup.pdf` in a download

directory or on ones desktop doesn't *really* help to identify them as Firebird manuals. So here are some guidelines for the file names:

- Make sure the name contains the word `Firebird`, preferably at the beginning;
- Try to make it resemble the document title, but keep it short;
- Use hyphens (“-”) to separate words;
- If the title is long, omit parts like “manual”, “guide”, “howto” etc., unless leaving them out would cause confusion;
- Use the language of the document, but ASCII-only (no accents etc.)
- If (and *only* if) applying the above rules leads to a file name that already exists in another language, add the document language (or an abbreviation thereof) to the name.

To illustrate these guidelines, some of our existing file names are listed below:

- `Firebird-2.0-QuickStart.pdf`
- `Firebird-Security.pdf`
- `MSSQL-to-Firebird.pdf`
- `Firebird-Generator-Guide.pdf`
- `Firebird-nbackup.pdf`
- `Firebird-2.0-Schnellanleitung.pdf`
- `Firebird-1.5-Arranque.pdf`
- `Firebird-et-Null.pdf`
- `Firebird-nbackup-fr.pdf`
- `Firebird-su-Ubuntu.pdf`
- `Firebird-nbackup-nl.pdf`
- `Guia-Escrita-Firebird.pdf`
- `Firebird-1.5-BystryjStart.pdf`
- `Firebird-Perehod-s-MSSQL.pdf`

Single-page HTML

If and when we start publishing single-page HTML files on the website – produced with **build monohtml** – we should give them the same name as the corresponding PDF, but of course with an `.html` extension.

Uploading the PDF

If you have write access to the Firebird web server, make an SFTP connection to `web.firebirdsql.org` and upload your properly named file(s) to:

- `/srv/www/htdocs/pdfmanual` (English docs)
- `/srv/www/htdocs/pdfmanual/fr` (French docs)
- `/srv/www/htdocs/pdfmanual/ja` (Japanese docs)
- etc.

Release Notes however go to:

- `/srv/www/htdocs/rlsnotes`
- `/srv/www/htdocs/rlsnotes/fr`
- etc.

If you don't have access to the server, ask someone else to upload the document(s) for you, or – if you are a project member – ask for a user name and password on the server.

Uploading the multi-page HTML files

Make sure you upload all the necessary files: the HTML files that together form your manual(s), the stylesheet `firebirddocs.css` (if it has changed since the last upload), as well as the subdirectory `images` with any content that has changed or has been added. To keep all the links working, it may also be necessary to build and upload the parent `book` of the document you have created or updated (or even the entire `set`). Upload the whole shebang to:

- `/srv/www/htdocs/manual` (English docs)
- `/srv/www/htdocs/manual/fr` (French docs)
- etc.

Warning

If the pages in question belong to another base set than the default `firebirddocs` (e.g. `papers` or `rl-snotes`) do not place them in the directories mentioned here. We haven't made any clear rules for this yet, but multi-page HTML builds from different sets should not be mixed. If this situation arises, bring it up on the `firebird-docs` list.

Updating the Firebird Documentation Index

The Firebird Documentation Index at <http://www.firebirdsql.org/?op=doc> is a PHP script that picks up most of its content from data files on the server. If you have updated existing documents that are already in the Index, you don't have to do anything here, unless you've changed the file name. But if you have created a new document or a new translation, you must add it to the Index. Here's how:

If you have created a completely new document

1. Look at the Documentation Index and decide which category is best suited for your document. (Categories are indicated with orange headers.)
2. Connect to the server, `cd` to `/srv/www/htdocs/doc` and look at the files starting with `Cat_`. Open the one that corresponds to the chosen category.
3. Read the instructions at the top of the file.
4. Create a new section starting with the document title in English, followed by a specially formatted line for each available version. For a new document, such a section could look like this:

```
Firebird Uninstallation Howto
en:/manual/fb-uninstall.html
en:/pdfmanual/Firebird-Uninstall.pdf
```

Each version line starts with the language code, followed by a colon, followed by a URL. For documents on our own server, this URL is simply the “absolute” path from the server root. Sections are separated by empty lines. The order of the sections in the file determines the listing order of the documents within their category on the Documentation Index web page. The order of the version lines within a section is irrelevant.

5. Save the file. If you've edited it on your own computer, upload it back to the server. Now refresh the Firebird Documentation Index page in your web browser and check if the document is listed where it should be, and if the links work well. Also verify that the links are in the right columns (HTML in the middle column, PDF and anything else in the rightmost column).
6. The PHP script does a pretty good job of auto-determining the document type, but there are cases where it gets it wrong. If this happens, add the file type – between curly braces – immediately after the URL on the version line:

```
en:http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_60_sqlref{html}
```

7. Once everything works fine, commit the updated category file to CVS. If you've checked out the Firebird web module from SourceForge, you'll find the category files (and more) in the folder `web/website/doc`. Use your SF user name and password to check out, otherwise you won't be able to commit your changes. Working with CVS is described in the [Firebird Docbuilding Howto](#).

If you have translated an existing document into a new language, or added a new document type for it

1. Consult the Documentation Index to see which category the document belongs to. (Categories are indicated with orange headers.)
2. Connect to the server, `cd` to `/srv/www/htdocs/doc` and look at the files starting with `Cat_`. Open the one that corresponds to the category.
3. Read the instructions at the top of the file.
4. Find the section for the document in question and add the version line(s) for your additions, e.g.:

```
Firebird Uninstallation Howto
en:/manual/fb-uninstall.html
en:/pdfmanual/Firebird-Uninstall.pdf
fr:/manual/fr/fb-uninstall-fr.html
fr:/pdfmanual/fr/Deinstaller-Firebird.pdf
```

The order of the version lines within a section is irrelevant, but the title must stay on top.

5. Steps 5, 6, and 7 are the same as for new documents.

Appendix A: Document History

The exact file history is recorded in the `manual` module in our CVS tree; see http://sourceforge.net/cvs/?group_id=9028

Revision History

0.1	17 Jan 2004	PV	First incomplete draft published under the title <i>Writing Documentation for Firebird</i> (aka <i>Firebird Docwriting Howto</i>).
0.2	27 Jan 2004	PV	First complete version. (Entered into CVS 31 Jan 2004)
1.0	8 Mar 2004	PV	First official release on Firebird website.
1.1	26 Feb 2005	PV	<p><i>The following changes have accumulated between March 2004 and Feb. 2005:</i></p> <p>Changed title to <i>Firebird Docwriting Guide</i>. Added section on PostgreSQL docs. Added note on non-DocBook contributions. Explained term well-formed XML. Made DocBook benefits list more concise. Changed recommendation on section vs. sectN elements. Dropped <code>xref</code> and some other rarely-used stuff from element reference; added <code>procedure</code>. Updated info on non-monospaced literallayout. Added section on PDL and how to include a License Notice and Document History. Numerous minor improvements. Added document history and revision number. Licensed this work under the Public Documentation License.</p>
1.1.1	8 April 2005	PV	Added paragraph on <code>titleabbrev</code> elements.
1.2	10 Feb 2006	PV	<p>Changed all <code><sectN></code> elements in the source structure to <code><section></code>.</p> <p>Changed <code>docbuildhowto</code> links to <code>ulinks</code>, as the articles will be in separate PDFs from now on.</p> <p><i>DocBook XML Characteristics</i>: removed “plaintext” remark. Added note about XSLT.</p> <p><i>DocBook XML authoring tools</i>: divided into two subsections; warned against ConText UTF-8 issue; added info on SciTE; added warning about saving as 8-bit; altered first para on dedicated XML tools; added Oxygen; removed Altova Authentic; updated/altered Altova XMLSpy information.</p> <p><i>Writing your DocBook doc</i>: renamed to <i>Setting up your DocBook doc</i>; changed 2nd para; moved 3rd para (“Please read the subsection...”) to <i>Elements we use frequently</i>; changed “subsection on hierarchical elements” link to normal text in the relocated para.</p>

Creating the Document: changed set/book introduction; updated master doc example; added UTF-16 note; added information on placement of files belonging to alternative base sets.

Typing text: minor changes to first and last para.

Elements we use frequently: promoted to top-level section, following *Setting up your DocBook doc*; changed tip before first subsection to normal para, altering its first sentence; split *Hierarchical elements* in subsections, and edited/added LOTS of stuff; added subsection on HTML tables; heavily edited the “quote - literal” section; added subsections on images and paragraph headers.

Non-DocBook aspects of the writing process: disappeared, all subsections have been promoted to top level; its first para is now in *Language and style*, and edited.

Copyrights: renamed *Copyright issues* and added an introductory para.

Respecting others' copyrights: renamed *Using material written by others*. The first para is split in two, and edited. The para about Borland docs is now in a subsection, with the first words removed. *Using PostgreSQL docs* is now also a subsection of *Using material written by others*, and renamed *PostgreSQL docs*.

Your copyright and the PDL: extensive editing, reorganisation of subsections, and additions.

Committing your work: included **cvs add** command line and “Important” note about committing after adding.

1.2.1	11 May 2006	PV	Corrected start tag in bridgehead example (removed /).
1.2.2	25 Jan 2007	PV	<i>Elements we use frequently</i> : Mentioned <code>title</code> option for admonitions. Moved instructions for translators regarding images into a note.
1.3	5 May 2007	PV	<p><i>Topics discussed in this guide</i>: Added new item to the last list.</p> <p><i>Links</i>: Removed note about offset hot zones (fixed in FOP 0.93).</p> <p><i>Program listings, screens, literal layout, and examples</i>: Removed note about non-monospaced <code>literallayout</code>. Wrapped <code>example</code> output in a blockquote.</p> <p><i>HTML tables</i>: Assigned id. Wrapped <i>processing instructions</i> in a <code>firstterm</code>.</p> <p><i>PDF rendering of large tables</i>: New section.</p> <p><i>Style</i>: Slight rewording in 3rd list item.</p> <p><i>Publishing your document on the Firebird website</i>: New section.</p> <p><i>License notice</i>: (C) 2004–2006 -> 2004–2007.</p>

Appendix B: License notice

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the “License”); you may only use this Documentation if you comply with the terms of this License. Copies of the License are available at <http://www.firebirdsql.org/pdfmanual/pdl.pdf> (PDF) and <http://www.firebirdsql.org/manual/pdl.html> (HTML).

The Original Documentation is titled *Firebird Docwriting Guide*.

The Initial Writer of the Original Documentation is: Paul Vinkenoog.

Copyright (C) 2004–2007. All Rights Reserved. Initial Writer contact: paulvink at users dot sourceforge dot net.