MySQL Guide to MySQL Documentation

MySQL Guide to MySQL Documentation

Abstract

This document describes the machinery, format, and operation of the MySQL documentation.

Document generated on: 2009-06-02 (revision: 15161)

If you want help with using MySQL, please visit either the MySQL Forums or MySQL Mailing Lists where you can discuss your issues with other MySQL users.

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML, CHM, and PDF formats, see MySQL Documentation Library.

Table of Contents

List of Tables

Chapter 1. MySQL Documentation Overview

MySQL develops all of the documentation internally. This includes the various version-specific reference manuals, the documentation for the GUI tools such as MySQL Administrator and MySQL Query Browser, and specialized guides such as the NDB API Internals manual.

These manuals are provided in multiple formats and are accessible from the main MySQL Documentation page.

This guide provides information on the format, structure, and build processes that are used to develop and produce the documentation, and information on how you can build your own custom documentation using the publicly available Subversion repository. (Note, though, that publishing MySQL documentation is subject to the limitations stated in our legal notice.) It also contains a style guide on writing proper documentation, and a DocBook guide covering mark-up.

All of the MySQL documentation is written using the DocBook XML format. Using DocBook XML enables us to easily generate the documentation in a number of different output formats, including:

- HTML, provided in both an online format and formatted, downloadable, packages. These are provided in either section-by-section, chapter-by-chapter, or single HTML formats
- PDF in both A4 and Letter page sizes as a single document
- Eclipse Documentation Plugin, which is essentially a repackaging of the HTML format
- RPM, which is another repackaging of the HTML and PDF output
- CHM, which is a compiled version of the HTML sources and the help format mostly used on Windows
- Unix Man pages
- Texinfo for use within the Info file format browsers within Linux/Unix

In addition to the standard DocBook XML source and transformations into these formats, we have built custom templates and translations to generate custom markup, and a number of tools and utilities that provide enhanced functionality. Note, though, that we have *not* modified the DocBook XML DTD. All our custom tools are written for either XSLT or Perl.

1.1. Documentation Repository and Directory Structure

The best way to understand the MySQL documentation is to download the documentation tree from the Subversion respository. To do this you will need to install Subversion (http://subversion.tigris.org). To just browse the documentation tree for a first impression you can simply go to http://svn.mysql.com/svnpublic/mysqldoc with a web browser.

The MySQL documentation repository is called `mysqldoc`. Once Subversion has been installed, you must check out the repository:

```
shell> svn checkout http://svn.mysql.com/svnpublic/mysqldoc
```

The repository contains a combination of the actual documentation source and the components and systems required to build the documentation, including tools for related tasks such as validating the XML or counting words.

Table 1.1. Core support directories in mysqldoc

Directory	Description
`dicts`	Dictionaries for the spell checker
`dtd.d`	The DocBook DTDs used to verify the structure of the DocBook XML
`make.d`	Makefile rules used to create specific targets and output formats in the documentation (examples: validation, HTML output)
`proto-doc`	A suite of basic document templates used to create chapters, sections and entire books
`tools`	A series of scripts and other tools used to help build the documentation, including tools for related tasks such as validating
`xsl.d`	The standard DocBook XSL style sheets and the custom extensions and improvements used specifically for the MySQL documentation. Note that we've modified neither the DocBook DTD nor the DocBook XSL style sheets, which enables us to update DocBook regularly.

The remaining directories hold the documentation itself, either in DocBook XML format or in a format that can be built and translated into DocBook XML for publishing. These directories also contain any additional files required for the document, such as the entity definitions (including some common files), and the graphics files used for illustrations and screenshots.

Table 1.2. Documentation directories in mysqldoc

Directory	Description
`administrator`	Files specific to the MySQL Administrator manual
`arbitrary`	Directory which holds specifications for arbitrary documentation
`common`	Files common to all documents
`dynamic-docs`	The source material for dynamically generated information, including the options/variables and operators/function definitions. These files are parsed and converted into DocBook XML for incorporation into different parts of the manual.
`falcon`	Falcon manual
`guibook`	The GUI book which bonds together the content from all the GUI tool manuals into a single book
`gui-common`	Files common to all of the GUI tool documentation
`internals`	MySQL internals manual
`migration-toolkit`	Migration Toolkit GUI tool manual
`mysqldoc-guide`	This guide
`mysqlqb`	Specially marked-up version of MySQL Manual chapters for MySQL Query Browser
`mysqltest`	Documentation for the `mysqltest` framework
`ndbapi`	Cluster/NDB API documentation
`query-browser`	MySQL Query Browser manual
`quick-guides`	Quick references for inclusion in the MySQL Manual
`refman-#.#`	DocBook XML source for documentation specific to different major versions of MySQL. The documentation here is augmented by the contents of `refman-common` and other files
`refman-common`	Documentation common to all the reference manuals. This directory contains elements such as the documentation for MySQL Connectors, graphics common to all versions, and chapters and appendices that appear in all versions of the MySQL Reference Manual
`sample-data`	Sample databases used in various parts of the documentation
`workbench`	MySQL Workbench GUI manual

Additionally, there are a few directories for translation purposes. These can be considered experimental at this point, and might or might not be moved to language-specific repositories soon.

1.2. Tools Required

To build and work with the documentation, you will need to install some additional tools, all of which are open source software with the exception of Microsoft Help Workshop (which is free but not open source). The exact list of tools required will depend on what you plan to do with the documentation.

There are some core tools required for all the basic operations:

- A text editor capable of handling plain-text files. We use a variety of editors, for example oXygen (a commercial product), vi, kate, jEdit, or Notepad++.
- **make** — you will need a standard `make` tool, either the one provided with your operating system, or preferably GNU `make`. On Windows, this might require you to install Cygwin or a similar Unix emulator that has `make` support.
- **XML tools** — the XML toolkit from http://xmlsoft.org provides a number of utilities that are required to work with the DocBook XML, including `xmllint` and `xsltproc`. You may find that your operating system already comes with the XML tools

installed.

- **Perl** — is required for some of the additional tools we use to support the production process. In addition to Perl itself, you will also need to install the Expat library, and the `XML::Parser::PerlSAX`, `IO::String`, and `IO::File` modules.

If you want to build PDF versions of the MySQL documentation, you need to install a suitable Java Runtime (v1.5 is recommended) and the Apache FOP tool (http://xmlgraphics.apache.org/fop/). At the time of writing this (December 2007) we recommend FOP 0.20.5.

For image support in PDF files you'll have to install a library such as `Jimi` or `JAI`. JAI is available from http://java.sun.com. Jimi is available from http://java.sun.com/products/jimi/.

When installing FOP under Unix/Linux/Mac OS X, install the package into a directory and then create a link to the `fop.sh` command as `fop`, for example:

```
shell> ln -s /usr/local/fop-0.20.5/bin/fop.sh /usr/local/bin/fop
```

You may also need to increase the amount of memory allocated to FOP when building the full reference manual. You can do this by adding the `-Xmx##m` command-line option to the final line of the `fop.sh` or `fop.bat` script:

```
shell> JAVACMD -Xmx2048m -classpath "$LOCALCLASSPATH" $FOP_OPTS org.apache.fop.apps.Fop "$@"
```

If you want to build CHM (Compiled HTML) documents then you must use Windows and install the HTML Workshop toolkit from Microsoft (http://www.microsoft.com/downloads/details.aspx?FamilyID=00535334-c8a6-452f-9aa0-d597d16580cc.

Chapter 2. Building Documentation Formats

The MySQL documentation can be built into a variety of different formats. The same mechanism and toolset is used to both build the documentation and build any additional files or validation steps relating to the documentation.

For information on how to build the MySQL documentation to a variety of formats, see Section 2.1, “How to Build MySQL Documentation”.

For details on what goes on behind the scenes when you build a particular target format, see Section 2.2, “How Different Formats are Built”.

Information on other targets, including processes that build, check and validate the documentation to ensure that it is ready to be built, see Section 2.3, “Other Build Targets”.

2.1. How to Build MySQL Documentation

The Documentation build process uses the `make` tool to run the various commands and steps required. We use `make` because we can specify different targets and the individual steps, and specify the required files and elements which trigger additional steps to build required documents.

By using `make` we can also automatically rebuild different components of the documentation based on whether the files have changed between the target output and the dependent files that were used to build that target.

The build system also means that you can build any documentation format by using a single command. For example, this guide is located within the `mysqldoc-guide` directory in the `mysqldoc` tree.

2.2. How Different Formats are Built

The basic process for building all documentation follows a similar sequence. You can see the sample output from building this guide into PDF format below:

```
../tools/idmap.pl mysqldoc-guide/en formats.xml
xsltproc --xinclude --novalid  \
--stringparam repository.revision "`../tools/get-svn-revision`" \
        --param map.remark.to.para 0 \
        --stringparam qandaset.style "" \
        ../xsl.d/dbk-prep.xsl mysqldoc-guide.xml | ../tools/bug-prep.pl > mysqldoc-guide-prepped.xml.tmp
../tools/idremap.pl  --srcpath=". " mysqldoc-guide-prepped.xml.tmp > mysqldoc-guide-prepped.xml.xprep
mv mysqldoc-guide-prepped.xml.xprep mysqldoc-guide-prepped.xml
rm -f mysqldoc-guide-prepped.xml.tmp
xsltproc --xinclude --novalid  \
--output - ../xsl.d/strip-remarks.xsl mysqldoc-guide-prepped.xml \
| xsltproc --xinclude --novalid  \
        --stringparam l10n.gentext.default.language en \
         \
        --output mysqldoc-guide.fo-tmp ../xsl.d/mysql-fo.xsl -
Making portrait pages on USletter paper (8.5inx11in)
lisitem encountered in itemizedlist, but no template matches.
mv mysqldoc-guide.fo-tmp mysqldoc-guide.fo
if [ -f ../xsl.d/userconfig.xml ]; then \
        ../tools/fixup-multibyte.pl mysqldoc-guide.fo > mysqldoc-guide.fo.multibyte; \
        mv mysqldoc-guide.fo.multibyte mysqldoc-guide.fo; \
        fop -q -c ../xsl.d/userconfig.xml mysqldoc-guide.fo mysqldoc-guide.pdf-tmp > mysqldoc-guide.pdf-err; \
else \
        fop -q mysqldoc-guide.fo mysqldoc-guide.pdf-tmp > mysqldoc-guide.pdf-err; \
fi
mv mysqldoc-guide.pdf-tmp mysqldoc-guide.pdf
sed -e '/hyphenation/d' < mysqldoc-guide.pdf-err
[ERROR] Unknown enumerated value for property 'span': inherit
[ERROR] Error in span property value 'inherit': org.apache.fop.fo.expr.PropertyException: No conversion defined
rm -f mysqldoc-guide.pdf-err
```

Note that all XSL stylesheets, including the MySQL specific customizations, CSS and other elements are stored within the `xsl.d` directory. For all output formats, the custom MySQL specific XSL stylesheet is used. This enables us to specify the custom setup and structure of the particular format. Each of the custom stylesheets then imports the default DocBook XSL stylesheet set.

Let's look at the steps in more detail. The relevant code is provided so that you can identify the individual steps.

1. The ID mapping system rebuilds any outdated ID maps. ID maps are used to identify whether an ID reference in a document is internal or external and allows the build system to automatically remap links it considers to be external to current document being generated. For more information, see Section 3.2, “ID Mapping System”.

   ```
   ../tools/idmap.pl mysqldoc-guide/en formats.xml
   ```

2. XSLT processor converts the base document into a “prepared” state. This builds the target `document-prepped.xml`. Included in this process is the entity expansion, the incorporation of the included XML files, and any custom DocBook XML formatting which is applied to all output formats. As part of this process, the output is also filtered through the `bug-prep.pl` command which translates Bug ID numbers into a URL that links directly to the MySQL online bugs system.

```
xsltproc --xinclude --novalid  \
--stringparam repository.revision "`../tools/get-svn-revision`" \
       --param map.remark.to.para 0 \
       --stringparam qandaset.style "" \
       ../xsl.d/dbk-prep.xsl mysqldoc-guide.xml | ../tools/bug-prep.pl > mysqldoc-guide-prepped.xml.tmp
```

3. The ID remapper is executed on the generated source. Any links that cannot be resolved internally are converted into URL links that link directly to the MySQL online HTML for the appropriate section.

```
../tools/idremap.pl  --srcpath=". " mysqldoc-guide-prepped.xml.tmp > mysqldoc-guide-prepped.xml.xprep
mv mysqldoc-guide-prepped.xml.xprep mysqldoc-guide-prepped.xml
rm -f mysqldoc-guide-prepped.xml.tmp
```

4. At this point, we now have a single DocBook XML document that is ready to be translated into the various output formats. From here on, the exact steps followed are unique for the output format being produced.

There are a number of advantages to creating the intermediate prepped XML document. The first is that because the prepped file is used by all the output format targets, we ensure that all the different forms of the documentation use the same source. If there is a change in one of the source files, the prepped version of the file has to be rebuilt, and then all the output formats need to be rebuilt too. It is impossible for the PDF and HTML versions of a document to differ in their content.

The second advantage is that because the prepped file is created before the output formats are generated, we reduce the overall time to produce a particular document in a particular format. Instead of restarting the entire process for each document, we only have to run the final transformation step for each target output format.

Although the aim is to create the full documents using this system, the same targets and systems can also be used to create a document from any sourcefile. This is very useful if you want to test the edits or corrections in one of the individual chapters or sections.

2.2.1. How HTML is Produced

You can convert DocBook XML to HTML directly using a XSL stylesheet. The stylesheet converts elements from the DocBook XML, such as a `<para>` into HTML equivalents, in this case `<p>`.

Because of the size of the documentation being produced (the reference manual is approximately 2000 printed pages long), converting the entire manual into a single HTML file would obviously produce a very large and complex document. For that reason, we support a number of different HTML output formats that split up the documentation into separate files.

These different formats are available through the same `make` interface by using the following targets:

- `make document.html` — converts the source document into a single HTML file.
- `make document.html-dir` — converts the source document into a single HTML file, placing the HTML file and any required images into a directory named `document.html-dir`.
- `make document.html-chapter` — converts the source document into HTML, creating a separate file for each chapter or appendix. The files are created in the directory `document.html-chapter`, and all the required images are copied into the directory.
- `make document.html-section` — converts the source document into HTML, creating a separate file for each section within the manual. The result is a higher number of files, but because each is smaller, they are quicker to load. The HTML files and required images are placed into the directory `document.html-section`.
- `make document.html-web` — converts the source document into HTML section by section, like the `html-section` target, but this option prepends each file with a block of PHP code which is used by MySQL when the manual is placed on the MySQL website. The files generated by this process are not usable outside of MySQL, but you may find the principles of the process useful.

You can see a sample of the process of converting to an HTML section-based format below:

In addition to the stages shown above common to all documents, making HTML in this way performs the following:

For both the online (web) and file-based HTML targets a CSS stylesheet us used to format the output for display. The CSS stylesheet used is located in `xsl.d/mysql-html.css`. For the directory-based formats this file is automatically copied into the directory as part of the package.

2.2.2. How HTML Packages are Produced

The HTML packages such as the `tar.gz`, `zip` and RPM files are produced by taking the HTML generated output from the `html-section` target and packaging them for download.

Conveniently the `html-section` and other formats also automatically copy the CSS and any images required by the document.

The same basic process is used for all the packaged HTML formats, including Zip, Tar/GZip and RPM builds. The same output files are used in each case, so that the build produces one set of files that are packaged up multiple times.

2.2.3. How Eclipse Packages are Produced

The DocBook XSL stylesheets contain a specialized output target for the Eclipse Packages. Eclipse documentation is essentially HTML documentation packaged up with a suitable Eclipse `plugin.xml` file so that the extracted directory can be dropped onto the users `plugins` directory.

The plugin registers the `toc.html` file as the root of the MySQL documentation, and adds the name and title of the documentation (and version) to the plugin details.

2.2.4. How PDFs are Produced

You cannot convert DocBook XML into PDF directly. The reason for this is that the DocBook XML standard is designed to be device independent (that is, they don't rely on fixed page or view sizes). The HTML format is also device independent, hence why the XSL stylesheet can be used to convert the source to HTML directly.

For PDF the DocBook XML is first translated into the Formatting Objects (FO) format. The FO format enables you to specify output attributes, such as the page size, and handles the flow and formatting of the source text and illustrations.

The FO file can then be converted using Apache's FOP system into the PDF format. The resulting sequence is therefore:

1. Transform the prepared XML file into an FO format file, specifying the target page size, margins and other elements (through the XSL stylesheet). The process also applies any specialized markup for the FO or PDF format.

```
xsltproc --xinclude --novalid  \
            --output - ../xsl.d/strip-remarks.xsl mysqldoc-guide-prepped.xml \
            | xsltproc --xinclude --novalid  \
            --stringparam l10n.gentext.default.language en \
            \
            --output mysqldoc-guide.fo-tmp ../xsl.d/mysql-fo.xsl -
            Making portrait pages on USletter paper (8.5inx11in)
            mv mysqldoc-guide.fo-tmp mysqldoc-guide.fo
```

2. Transform the FO file into the PDF file. To ensure that any multibyte characters (such as those in the Chinese/Japanese/Korean FAQ in the reference manual) are rendered correctly, we parse the generated FO and change the font specification for these elements. This step is only followed if the `userconfig.xml` file, which is required by FOP and defines fonts, is available. If the file is not available, we still transform the file into PDF.

```
if [ -f ../xsl.d/userconfig.xml ]; then \
     ../tools/fixup-multibyte.pl mysqldoc-guide.fo > mysqldoc-guide.fo.multibyte; \
     mv mysqldoc-guide.fo.multibyte mysqldoc-guide.fo; \
     fop -q -c ../xsl.d/userconfig.xml mysqldoc-guide.fo mysqldoc-guide.pdf-tmp > mysqldoc-guide.pdf-err; \
else \
     fop -q mysqldoc-guide.fo mysqldoc-guide.pdf-tmp > mysqldoc-guide.pdf-err; \
fi
mv mysqldoc-guide.pdf-tmp mysqldoc-guide.pdf
sed -e '/hyphenation/d' < mysqldoc-guide.pdf-err
[ERROR] Unknown enumerated value for property 'span': inherit
[ERROR] Error in span property value 'inherit': org.apache.fop.fo.expr.PropertyException: No conversion defined
rm -f mysqldoc-guide.pdf-err
```

The current documentation tree includes standard FO (and therefore PDF) output definitions for A4, Letter and 6in x 9in (standard book) sized output.

2.2.5. How CHM is Produced

The Compiled HTML (CHM) format was developed by Microsoft as a solution to the problem of distributing an online viewable version of documentation as a convenient single file.

At the core, CHM is just HTML, but some additional files are produced during the build process to generate a suitable table of contents and index file that can be indexed and updated so make browsing and searching the CHM content easier.

The core of the CHM process is therefore a suite a XSL stylesheets that generated the HTML and the associated TOC and index files required by the CHM standard. This process can be run on any machine.

To actually produce a CHM file, you need to use a Windows only application called HTMLHelp that compiles (and compresses) the individual HTML files and the table of contents/index files into the CHM file.

It is difficult to build the files directly on Windows. At MySQL we use a combination of assembly of the core files on a Unix system and a conversion of that source XML file and the source images on a Windows system to produce the CHM. The Windows system uses `xsltproc` to create the HTML files and the associated table of contents and index files, and `htmlhelp` to create the final CHM.

2.2.6. How Texinfo/Info is Produced

The Info format is supported by Emacs and many other tools as a method for browsing large and small documents. Individual chapters and sections are converted into nodes within the Info document.

The Info format is produced by calling `makeinfo` on the `texi` file. The `texi` can be produced directly from the DocBook file format using the XSL stylesheets.

To build the Info file, use the `info` target:

```
$ make manual.info
XML_CATALOG_FILES="/MySQLData/Docs/mysqldoc/trunk/catalog.xml" xsltproc --xinclude --novalid  \
                --stringparam repository.revision "`../tools/get-svn-revision`" \
                --param map.remark.to.para 0 \
                --stringparam qandaset.style "" \
                ../xsl.d/dbk-prep.xsl manual.xml > manual-prepped.xml.tmp2
../tools/bug-prep.pl < manual-prepped.xml.tmp2 > manual-prepped.xml.tmp
../tools/idremap.pl  --srcpath=". ../refman-common ../falcon ../ndbapi" manual-prepped.xml.tmp > manual-prepped.xml.tmp
mv manual-prepped.xml.tmp2 manual-prepped.xml
rm -f manual-prepped.xml.tmp
XML_CATALOG_FILES="/MySQLData/Docs/mysqldoc/trunk/catalog.xml" xsltproc --xinclude --novalid   \
                --stringparam l10n.gentext.default.language en \
                ../xsl.d/dbk-texi.xsl manual-prepped.xml > manual.texi.tmp
WARNING: FOUND 'REGISTERED' SYMBOL
WARNING: FOUND 'REGISTERED' SYMBOL
WARNING: FOUND 'REGISTERED' SYMBOL
WARNING: FOUND 'REGISTERED' SYMBOL
../tools/fixup-texi.pl manual.texi.tmp > manual.texi.tmp2
rm -f manual.texi.tmp
mv manual.texi.tmp2 manual.texi
makeinfo --no-split -I . manual.texi
```

Note that during the process we correct some problems in the `texi` output before it is processed that are known to cause problems when the file is converted to Info format.

2.3. Other Build Targets

The `Makefile` defines some additional targets which are used to help build, format and validate the DocBook XML. You can get a list of the valid targets for any directory by just running `make` in a directory. For example, within `refman-5.1`, the following list is produced:

```
You must say what you want to do.
    Some commands supported by this Makefile:
    make dba-optvars-table.xml - generate option/variable table
    make file.valid           - validate file.xml with ID map checking
    make file.validwarn       - validate file.xml with ID map checking, showing warnings
    make file.validpure       - validate file.xml without ID map checking
    make idmap                - make ID maps for the current directory
    make idmap.refs           - make ID maps for all directories referenced by main document
    make idmap.reconcile      - reconcile ID maps for the current directory
    make idmap.reconcile.refs - reconcile ID maps for all directories referenced by main document
    make file.deepcheck       - perform deep check of file.xml
    make file.ulinkcheck      - checks <ulink> URLs for file.xml
    make file.format          - put file.xml in standard format
    make file.wc              - counts the words within XML tags
    make file.wcd             - counts the words within XML tags (with tag-by-tag counts
    make file.useless         - find suboptimal constructs in file.xml
    make file-prepped.xml     - preprocess file.xml for producing output
    make file-remprepped.xml  - preprocess file.xml for remark-counting
    make file-manprepped.xml  - preprocess file.xml for producing man pages
    make file.html            - convert file.xml to HTML, single file
    make file.html-dir        - convert file.xml to HTML, single file
                                (putting output in subdirectory)
    make file.html-section    - convert file.xml to HTML, 1 file/section
                                (putting output in subdirectory)
    make file.html-chapter    - convert file.xml to HTML, 1 file/chapter
                                (putting output in subdirectory)
    make file.html-web        - convert file.xml to online manual HTML
```

```
                              (putting output in subdirectory)
    make file.eclipse         - convert file.xml to HTML for Eclipse
    make file.chm-input       - prepare input for Windows CHM build
    make file.xhtml-dir       - convert file.xml to XHTML, single file
                              (putting output in subdirectory)
    make file.pdf             - convert file.xml to file.pdf (US letter)
    make file.a4.pdf          - convert file.xml to file.pdf (A4)
    make file-toc.txt         - produce table of contents from file.xml
    make file.txt             - convert file.xml to file.txt
    make file.texi            - convert file.xml to file.texi
    make file.info            - convert file.texi to file.info
    make file.man             - generate Unix man pages from file.xml
    make file.help            - extract help-table information from file.xml
    make file.remarks         - extract <remark> elements from file.xml
    make file.remark-count    - count <remark> elements in file.xml
    make file.titles          - convert file.xml to id/titles file
    make fragments            - generate refman 'fragment' files
    make depend               - regenerate document dependencies
    make file.check-listing   - check <programlisting> line lengths
    make clean                - remove files built by other targets
    make realclean            - remove files built by other targets and metadata directory
                              (as created by ID mapping process)
```

Key additional targets include:

- `document.format` — reformat the XML in the document to help make it more readable.
- `document.valid` — validates the XML of the document using `xmllint`. As standard, `xmllint` would normally report missing IDs that could be valid in terms of the overall document (that is, those in another file). The output of `xmllint` is parsed by the ID mapping system to correctly identify these IDs. For more information, see Section 3.2, “ID Mapping System”.
- `document.deepcheck` — runs the document through a much more extensive DocBook XML checking process. This does more than validate the XML, this also checks for problems with the content, such as mismatches in the number of specified and actual columns in tables, missing IDs, duplicate IDs and other elements.

Chapter 3. Documentation Tools

In addition to the tools used to actually build and convert documentation into the various target formats, a number of additional tools and utilities have been developed that aid the build and production of documentation. Some of these tools relate to the build process and provide additional functionality or formatting. Others are used to aid in the validation.

For example, `deepcheck.pl` parses the DocBook XML file and looks for problematic constructs.

In this chapter we will look at some of the key tools that make up the documentation utilities.

3.1. Deep Checker

The Deep Checker script (`deep-check.pl`) provides additional validation on DocBook XML files. Standard validation is provided through the `valid` target on a document. However, this validation is limited to ensuring that the XML is valid XML, that all of the files included or referred to (including image files) are available, and that the different entities in the file can be parsed.

The standard validation doesn't check whether the definitions of the different elements are valid, or whether problems with specific combinations of different DocBook sequences will cause issues in certain target formats.

The Deep Checker script addresses these issues by parsing the DocBook XML and then identifying different issues and reporting on them. The script checks a for a variety of problems, including:

- Missing IDs on chapter and section elements.
- Checks for duplicate IDs in a single file.
- Checks the format and definition of tables (both `table` and `informaltable`). In particular, it checks:
  - Compares the column definitions (`colspec`), and the actual number of columns defined in the table.
  - Calculates the specified width of the table and ensures that it is not more than 100% of the width of page.
- Checks that specified images exist and that the image files available on the file system and those specified match.
- Checks the line length of `programlisting` elements to highlight potential line wrap in PDF documents. The line length can be configured on the command line.
- Checks for empty `link` references. A link reference does not automatically populate the link text, so an empty link produces a link with the link text as question marks.
- If you specifically request it, you can also check the `ulink` elements, which point to standard Internet URLs. The checking process accesses each URL and then verifies whether the URL is still reachable.

Errors reported by the Deep Checker script are reported in relation to the section in which they appear. This enbles you to identify more easily where a problem is within the DocBook XML source.

3.2. ID Mapping System

The ID mapping system was designed to provide two main areas of functionality:

- The first resolves the issue that a single source file may refer to the ID of a section outside of the current file, but which is still valid within the whole document.
- The second aspect that the ID mapping system resolves is that when building a document you may want to refer to a section that is outside of the current document, but which you still want readers to be able to access.

To achieve this, a suite of scripts is used that parse the DocBook XML source files and build a list of the IDs they contain, their parents, the titles associated with the ID (if available) and their type. This information is written into separate files, one per source file, into the `metadata` directory.

When a DocBook XML file is validated, all the ID maps for files related to the main document are loaded, and the validation output is parsed through a script, `idvalidate.pl`, which checks whether the missing IDs in the file are actually valid in the document as a whole.

When building a document, any ID cross references in the document that do not reference an ID in that document are translated to

an external URL link that links to the current document and section on the MySQL website. This enables you to add cross references in document to another document and still have the URL link to a valid resource.

Because the ID mapping system also builds a valid database of section IDs and the files in which they appear, the ID mapping system also has other potential uses, including the arbitrary documentation system. See Section 3.3, “Arbitrary Documentation”.

3.3. Arbitrary Documentation

The Arbitrary Documentation system, known internally as Arby, provides a method for building complex documents that merge together individual files and sections from multiple parts of the MySQL documentation tree.

The standard method for creating a custom document within DocBook XML relies on splitting up the source XML into multiple files which are then imported to a parent document. The MySQL Documentation tree does this for nearly all documents where individual chapters, and occasionally individual sections, are imported explicitly into the parent document.

For example, this document, the MySQL Documentation Guide, is split into four separate files, `tools.xml`, `writing.xml`, `overview.xml` and `formats.xml`. These files are merged together to provide a single guide through the `mysqldoc-guide.xml` file.

The problem with this method is that you cannot insert a single section, it has to be the whole file, unless you split up your documentation into a single file per section. This is impractical and doesn't help the logical structure of the documentation, which is better split into chapters with occasional sections for shared items, which is the approach used with the MySQL documentation.

Furthermore, using this method also means that you cannot import a section and use it as a chapter, or conversly import a chapter into an another chapter as if it was a section.

These are the issues that the arbitrary documentation system tries to resolve. The arbitrary system works in combination with the ID mapping system to enable you to create a compound document based any ID-tagged component within the documentation. The system will also remap a section to a chapter, or vice versa, and selectively include or exclude all child sections from the imported components.

Arby works through a single script combined with an XML definition of the arbitrary document you want to create, and a DocBook XML wrapper that is used to provide the base document structure.

3.3.1. Writing Arby XML

Sample files for arbitrary documentation are provided in the `arbitrary` directory within the `mysqldoc` repository. To demonstrate how the system works, we'll use the `windows.aspec` specification and the `windows.xml` wrapper file.

You can see a sample of the arbitrary specification file below:

```
<?xml version="1.0" encoding="utf-8"?>
    <book>
      <fragment id="windows">
        <include src="windows-installation" filter="none" prefix="../refman-5.1" markas="chapter">
          <include src="connector-net-installation-windows" filter="none"/>
          <include src="myodbc-installation-binary-windows" filter="none"/>
          <include src="myodbc-installation-source-windows" filter="none"/>
          <include src="installation-windows" filter="none">
            <title>Installing GUI Tools Under Windows</title>
          </include>
        </include>
        <include src="can-not-connect-to-server-on-windows" prefix="../refman-5.1" markas="appendix"/>
      </fragment>
    </book>
```

Blocks of content are organized into fragments and must have a unique ID. This ID will be used when the fragment is inserted into the wrapper document. The fragment does not have to a DocBook XML section, it is simply the name given to an arbitrary block of text that could include multiple sections or chapters.

The `include` tag is used to specify a section of the documentation that you want to incorporate into this fragment. Sections are included in the order in which they are specified. The include definitions consist of the following attributes:

- `src` — the ID of the section you want to import. This must be a valid ID from any of the documents in the documentation repository.
- `prefix` — if the ID you have specified is defined in multiple documents then you can use the prefix to select which source document to use.
- `filter` — you can choose whether to filter the subsections within a given ID. The valid options are `none`, which implies that all sections are incorporated (including any imported documents) or `all`, which filters all subsections, including imported documents.

- `markas` — if you are importing a section and want the section to be treated as a chapter, or for a chapter to be treated as a section, you can specify what the imported section should be remapped as.

For example, the first `include` tag within the Windows arbitrary document specification shows that we are importing the section `windows-installation`, that we shouldn't filter any sub-sections, that we should source the section from the `refman-5.1` directory and that the section should be remapped as a chapter.

You can close the tag at this point if this is the only section that you want to import. However, if you want to include additional sections to form subsections of the section you have just specified, you can add more `include` statements. An example of this is shown in the Windows arbitrary document.

For any included section, you can also change the title of the section when it is imported by using the `title` tag within the `include` tag. In the example specification, the `installation-windows` section title has been altered.

You can have as many `include` statements as you like, and an unlimited structure of nesting. You can also specify as many fragments as you like in the arbitrary specification file. The only limitation is that you cannot import the same section multiple times (because this would introduce duplicate IDs into the final document).

To actually generate your document, you also need the wrapper file into which the fragments will be inserted as part of the build process.

3.3.2. Writing an Arby Wrapper

An Arby wrapper is just a DocBook XML document that includes specially formatted comment tags. These comment tags will be replaced with the corresponding fragment from the arbitrary documentation specification. The wrapper document can contain any standard DocBook XML, even additional chapters, sections and statements to include other entire files into the document.

When you want to insert a fragment definition from the arbitrary specification you use an XML comment of the form:

```
<!--SRCFILE fragment-id-->
```

Where `fragment-id` is the ID of the corresponding fragment in the specification.

These specifications should be placed in the appropriate place within your document. For example, if your fragment specification creates a chapter, then the specification can be placed at top level in the document. If the specification is a section, then it must be included in a parent chapter.

Also be aware that you could, if you wanted, generated multiple arbitrary documents which are themselves imported and inserted into existing documents.

3.3.3. Building an Arby Document

To build an arbitrary document you use the same method as you would any other document, `make` with a specific target. The `-arbitrary` after the base name of the source files indidicates that we should build using the arbitrary system. For example, to build the Windows arbitrary document as a PDF you would use:

```
shell> make windows-arbitrary.pdf
```

Behind the scenes, the following happens:

1. The `genarbelements.pl` script is executed. This accepts the arbitrary specification file and wrapper file as arguments.
2. The arbitrary specification is parsed and validated.
3. For each each fragment, the script then:
    a. Parses each `include` statement, first loading the file that contains the specified ID.
    b. The DocBook XML block (be it section, chapter, or appendix) is extracted from the file.
    c. If subsections are filtered, these sections are removed.
    d. If the section is to be remapped, the correesponding tags are updated.
    e. If the `include` section has additional subsections defined, these are appended to the end of the section before the closing tag for that section.

4. Once a fragment specification is complete, the corresponding import specification in the wrapper XML is located and replaced with the fragment text.

5. The updated wrapper document is written out as a file with name *`filename`*`-arbitrary.xml`.

The generated file is then processed as normal by the rest of the documentation build process into the desired format.

Chapter 4. Conventions for Writing Proper DocBook XML ("Formatting Guide")

This chapter provides guidelines for writing MySQL documents using DocBook:

- Introduction to DocBook
- General document-writing conventions
- How to begin a new document
- How to structure a DocBook document
- Which DocBook XML elements should be used for which purpose
- How to use IDs, links, and quotes
- What to do when you begin using a DocBook element that we have not been using before
- Graphics markup and special markup

A few basic hints for editing before we start:

- **svn update:** Update your copy of the SVN repository before you start editing files in that repository. That reduces the chance of having to do manual merges because of conflicts.
- **Spaces, not tabs:** Use spaces, not tabs. Use 2 spaces to indent.
- **Unix line endings:** Make sure your editor saves your edits using Unix-style line endings (LF), rather than Windows-style (CR/LF) or Mac OS-style (CR) line endings.

4.1. Introduction to DocBook

DocBook contains a large number of XML elements from which we only use a subset. Our subset, however, uses other elements than the subset used by Simplified DocBook; that is why we are not using Simplified DocBook (which would otherwise be a nice idea, because many editors support Simplified DocBook only).

Be aware that XML has two special characters that cannot be used literally in text: `<` (the opening angle bracket) and `&` (the ampersand sign). Whenever you need to use those two characters literally, you have to use entities, like in this example: `<literal> & <programlisting>`

We structure our documentation using nested `section` elements. A `book` as DocBook defines it would look like this:

```
<book>
 <chapter>
  <section>
   <section>
    ...
   </section>
  </section>
 </chapter>
 <appendix>
  <section>
   ...
  </section>
 </appendix>
</book>
```

4.2. Document-Writing Conventions

Most of our documents are written using DocBook, a "dialect" of XML. We use the following conventions:

- Our DocBook source files use UTF-8 encoding, so they begin with an XML prolog that looks like this:

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

- We validate documents against the DocBook DTD. Currently, we use version 4.3 of the DTD.
- We process documents into various output formats such as HTML and PDF using the DocBook XSL stylesheets. Currently, we use version 1.69.1 of the stylesheets. For some output formats not supported by the DocBook stylesheets, we have implemented the necessary transforms by writing our own stylesheets. The DocBook-to-Texinfo transform is one of these.
- Generally, we set up each document in its own directory. The content for a short document usually is written in a single DocBook file. For larger documents, we use XInclude so that we can modularize content using multiple XML files. The usual modularization unit is the chapter, so the document consists of a "main" document file that refers to other parts of the document using XInclude directives.

  Sometimes the files for a document are located in multiple directories. This occurs when two or more documents need to share files. In that case, we put the shared files in their own directory. Examples:

  - There are several versions of the *Reference Manual*, all of which use files from the `refman-common` directory.
  - The manuals for the MySQL Administrator and MySQL Query Browser GUI tools both use files that are located in the `gui-common` directory.
- Each document has an "info" section that includes certain standard elements: Markup that causes the current date and revision number to be inserted into formatted output, a legal notice, copyright information, and (for translated documents) a disclaimer that points out that the translation might not be as up to date as the English version.
- If a document uses images, they'll usually be located in an `images` directory under the document directory. If multiple documents share images, they'll be located in the `images` directory under the directory that holds shared files.
- Each document directory contains a `Makefile` that provides the `make` target rules for document-processing operations such as validating the document or producing formatted output from it.

4.3. How to Begin a New Document

The easiest way to begin a new document is to copy the directory for an existing document and then make the necessary changes for the new document. To facilitate this process, the `proto-doc` directory of the `mysqldoc` repository can be used as a basis for a new document. Use it like this (assuming that you are located in the root directory of the `mysqldoc` repository and want to create a new document directory named `my-doc` there):

1. Make a copy of the `proto-doc` directory named `my-doc`:

   ```
   shell> cp -r proto-doc my-doc
   ```

2. Change location into the new directory:

   ```
   shell> cd my-doc
   ```

3. Rename the `proto-book.xml` file to the name that you'll use for your document:

   ```
   shell> mv proto-book.xml my-doc.xml
   ```

4. Look through `my-doc.xml` and edit as necessary. For example, change the title to something meaningful.
5. Edit `Makefile` to change instances of `proto-book` to `my-doc` and instances of `PROTO_BOOK` to `MY_DOC`. Also read the other comments in the `Makefile` and make any appropriate changes.
6. Update the `Makefile` to list the correct dependencies for the document:

   ```
   shell> make depend
   ```

7. Update `legalnotice.lang.xml` with the actual copyright date.
8. Change location a level up to the parent directory, add the new directory to the value of the `SUBDIRS` variable in the parent directory `Makefile`. Then add the new directory to the repository and commit your changes:

   ```
   shell> cd ..
   shell> vi Makefile (edit SUBDIRS)
   shell> svn add my-doc
   shell> svn commit -m "Add my-doc document directory"
   ```

At this point, you can develop the document further to create its content. Use Section 4.4, "General Document Organization", as a guide to structuring it.

4.4. General Document Organization

MySQL documents vary somewhat in their structural characteristics, but the overall layout generally is similar to the following example. The document shown uses `<book>` and `<bookinfo>`, which is appropriate for a longer document. Shorter documents might use `<article>` and `<articleinfo>` instead.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd"
[
  <!ENTITY % all.entities SYSTEM "all-entities.ent">
  %all.entities;
]>
<book id="top" lang="en">

  <title>... document title ...</title>

  <bookinfo>

    <abstract>

      <para>
        ... optional high-level description of document ...
      </para>

      <para>
        Document generated on:
        <?dbtimestamp format="Y-m-d"?>
        <remark role="repository.revision"/>
      </para>

    </abstract>

    <xi:include href="legalnotice.en.xml"
      xmlns:xi="http://www.w3.org/2001/XInclude"/>

    <releaseinfo>... disclaimer (for translations only) ...</releaseinfo>

  </bookinfo>

  ... document content goes here; chapters, sections, etc. ...

  <index/>

</book>
```

The first line is the XML declaration, which indicates the character encoding. This is almost always `utf-8` for document files. Exceptions might occur for files containing licensing text containing special characters in some other character set such as `iso-8859-1`.

The `DOCTYPE` declaration names the root element contained in the file and identifies the relevant DTD. If a document file needs to use entity references, those entities should be defined in the `DOCTYPE` declaration. Our convention is to place all definitions for the entities needed by the XML files in a given directory in a file named `all-entities.ent`. Then the `DOCTYPE` declaration for each XML file that needs entities defines and references the single entity file `all-entities.ent`. This makes the `DOCTYPE` declaration easier to write. Also, if the set of required entities changes, the change needs to be made only in `all-entities.ent`. For additional information about using entity files, see Section 4.4.4, "Entities and Entity Files".

Each DocBook document has a root element and a title. For longer documents, the root element is `<book>`. For a shorter document, the root element might be `<article>`. The root element must be the same as specified in the `DOCTYPE` declaration. The opening tag of the root element should have an `id` attribute, and a `lang` attribute that indicates the document language.

Following the title, the document contains an "info" element that includes some standard elements. The "info" element is `<bookinfo>` if the root element is `<book>`, and `<articleinfo>` if the root element is `<article>`. The "info" element contains several parts:

- An `<abstract>` element. The abstract includes an optional high-level description of the document and a paragraph containing special markup that causes the current date and repository revision number to be inserted when formatted output is generated from the document. The date and revision number are useful for determining how up to date a document is. For example, when you're looking at an online manual on `dev.mysql.com`, you can tell when changes that you have committed have propagated to that site by noting when the revision number changes. The special markup paragraph must be written exactly as shown, except that the amount of whitespace between its elements does not matter.
- A legal notice that contains the copyright date and the conditions under which the document can be copied. This is a boilerplate file, so its content is stored in a separate file that is referenced via XInclude from the main document file. The legal notice file is named `legalnotice.`*`lang`*`.xml`, where *`lang`* is the code for the document language. An English legal notice file (`leg-`

`alnotice.en.xml`) looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE legalnotice PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd">
<legalnotice>

<!--
  The role attribute for each top-level legalnotice element determines
  the context in which the element is used: 1) no role attribute: part
  of the default legalnotice, used for regular documentation that is
  not released under GPL; 2) role = legalnotice-gpl: for man page
  legalnotices, which are released under GPL; 3) role =
  legalnotice-all: included in legalnotice for all documentation.
-->

  <para>
    Copyright 2007-2008 MySQL AB, 2009 Sun Microsystems, Inc.
  </para>

  <para>
    This documentation is NOT distributed under a GPL license. Use of
    this documentation is subject to the following terms: You may create
    a printed copy of this documentation solely for your own personal
    use. Conversion to other formats is allowed as long as the actual
    content is not altered or edited in any way. You shall not publish
    or distribute this documentation in any form or on any media, except
    if you distribute the documentation in a manner similar to how MySQL
    disseminates it (that is, electronically for download on a Web site
    with the software) or on a CD-ROM or similar medium, provided
    however that the documentation is disseminated together with the
    software on the same medium. Any other use, such as any
    dissemination of printed copies or use of this documentation, in
    whole or in part, in another publication, requires the prior written
    consent from an authorized representative of MySQL AB. MySQL AB
    reserves any and all rights to this documentation not expressly
    granted above.
  </para>

  <para role="legalnotice-gpl">
    Copyright 2007-2008 MySQL AB, 2009 Sun Microsystems, Inc.
  </para>

  <para role="legalnotice-gpl">
    This documentation is free software; you can redistribute it and/or
    modify it only under the terms of the GNU General Public License as
    published by the Free Software Foundation; version 2 of the License.
  </para>

  <para role="legalnotice-gpl">
    This documentation is distributed in the hope that it will be
    useful, but WITHOUT ANY WARRANTY; without even the implied warranty
    of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
    General Public License for more details.
  </para>

  <para role="legalnotice-gpl">
    You should have received a copy of the GNU General Public License
    along with the program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
    02110-1301 USA or see http://www.gnu.org/licenses/.
  </para>

  <para role="legalnotice-all">
    For more information on the terms of this license, for details on
    how the MySQL documentation is built and produced, or if you are
    interested in doing a translation, please contact the
    <ulink url="http://www.mysql.com/company/contact/">Documentation
    Team</ulink>.
  </para>

</legalnotice>
```

Observe that the legal notice has text for different kinds of licenses. MySQL documentation is under copyright, so for most output formats we use the non-GPL text. The exception is that we release Unix man pages under the GPL to make it easier for organizations that produce GPL operating system distributions. These organizations often include MySQL, and having the man pages under GPL enables them to include some documention for the MySQL programs in the distribution.

For MySQL Reference Manual documents, the legal notice also contains a paragraph pointing to the "Preface, Notes, License" section for additional licensing information. The `preface.xml` file then has subsections for a more complete copyright notice and for licensing terms of any third-party products used in MySQL that require their licensing to be reproduced in the documentation of any including products. The files for these subsection are found in the `refman-common` directory.

- For translated documents, the "info" element includes a `<releaseinfo>` element. The content of this element is a disclaimer that points out that the translation is based on the English version but might not be as up to date. The German disclaimer for the *MySQL 5.1 Reference Manual* looks like this:

```
<releaseinfo>
  Dies ist eine Übersetzung des MySQL-Referenzhandbuchs, das sich auf
```

```
  <ulink url="http://dev.mysql.com/doc/mysql/en">dev.mysql.com</ulink>
  befindet. Das ursprüngliche Referenzhandbuch ist auf Englisch,
  und diese Übersetzung ist nicht notwendigerweise so aktuell wie
  die englische Ausgabe. <emphasis role="bold">Das vorliegende
  deutschsprachige Handbuch behandelt MySQL bis zur Version 5.1. Es
  ist derzeit noch unvollständig (Teile sind noch nicht
  übersetzt). Geplanter Fertigstellungstermin: 30. Juni
  2006</emphasis>
</releaseinfo>
```

Following the "info" element, write the document's content. This content can be written directly (inline) within the root element, or placed in other files and referenced from the main document file using XInclude directives. For more information about using multiple content files, see Section 4.4.1, "Partitioning Document Content into Multiple Files".

If the document should have an index, place an `<index/>` element just before the root element closing tag. For the index to have any content, you must place `<indexterm>` elements at appropriate places within the document.

4.4.1. Partitioning Document Content into Multiple Files

A document file can include document content directly (inline), or it can reference its parts by using XInclude directives. For a longer document, the usual structuring arrangement uses a "main" document file and a separate file per chapter. The main file contains the root element, the document title, the "info" element, and XInclude directives that refer to the chapter files. For example, the *MySQL Test Framework Manual* has a main file named `mysqltest.xml` that looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd"
[
  <!ENTITY % all.entities SYSTEM "all-entities.ent">
  %all.entities;
]>
<book id="mysql-test-framework" lang="en">

  <title>The MySQL Test Framework</title>

  <bookinfo>
    ... standard info element parts ...
  </bookinfo>

  <xi:include href="preface.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  <xi:include href="introduction.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  <xi:include href="components.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  <xi:include href="tutorial.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  <xi:include href="programs.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  <xi:include href="command-reference.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>

  <index/>

</book>
```

XInclude directives are used for files that have XML (or perhaps text) content, but not for files that define entities. Entity files should be listed in the `DOCTYPE` declaration. (See Section 4.4.4, "Entities and Entity Files".)

A DocBook content file that is referenced via an XInclude directive must itself be a valid DocBook file. That is, it must begin with an XML declaration and `DOCTYPE` declaration, and it must have a single root element. For example, you cannot have two `<chapter>` elements in the file. You must have two files, each containing one `<chapter>` element.

Sometimes document content is stored in a file and referenced via XInclude for reasons that have nothing to do with document length:

- Some pieces of content are so stereotypical that they are the same for every document (or at least for several documents). The `legalnotice.en.xml` file is like this. In such cases, we put the content in a file, copy it to every document directory where it's needed, and refer to it from within each document via XInclude.
- If content is used by multiple documents, put it in a separate file and include it in those documents by using XInclude. Files in the `refman-common` directory are instances of this concept. For example, the `copyright.en.xml` file contains the full *Reference Manual* copyright notice.
- Some documents contain program-generated content. In such cases, it's usually easiest to use a separate file for this content so that you can simply run the program again to update the file. Examples of this are the *Reference Manual* files that list SQL reserved words and error messages. The reserved words and error messages vary per version of the manual, but are generated for

each manual by running programs that parse server source files in the appropriate source tree.

The *MySQL Reference Manual* is an example where multiple documents share files. There are actually several versions of the manual, which use directories located in the `mysqldoc` repository as follows:

```
mysqldoc/
        |common/
               |fixedchars.ent
        |refman-common/
                       |all-entities.ent
                       |urls.ent
                       |...
        |refman-6.0/
                   |all-entities.ent
                   |versions.ent
                   |manual.xml
                   |preface.xml
                   |...
        |refman-5.1/
                   |all-entities.ent
                   |versions.ent
                   |manual.xml
                   |preface.xml
                   |...
        |refman-5.0/
                   |all-entities.ent
                   |versions.ent
                   |manual.xml
                   |preface.xml
                   |...
        |refman-4.1/
                   |all-entities.ent
                   |versions.ent
                   |manual.xml
                   |preface.xml
                   |...
        |...
```

The files in the `refman-common` directory are shared by all versions of the *Reference Manual*. For each version of the manual, chapter structure (and order) is governed by the `manual.xml` file that uses XInclude directories to reference the chapter files:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd"
[
  <!ENTITY % all.entities SYSTEM "all-entities.ent">
  %all.entities;
]>
<book id="refman-5-1" lang="en">

  <title>MySQL 5.1 Reference Manual</title>

  <bookinfo>
    ... standard info element parts ...
  </bookinfo>

  <xi:include href="preface.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  <xi:include href="introduction.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  <xi:include href="installing.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  <xi:include href="tutorial.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  ...
  <xi:include href="../refman-common/ha.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  ...

  <index/>

</book>
```

4.4.2. Referring to Files in Other Directories

For a file referenced via an XInclude directive or `ENTITY` definition, refer to the file using a path name that is relative to the location of the referencing file. Suppose that a *Reference Manual* document file in the `refman-5.1` directory refers to these files:

- The entity file `all-entities.ent` located in the `refman-5.1` directory
- The XML content file `preface.xml` located in the `refman-5.1` directory
- The XML content file `ha.xml` located in the `../refman-common` directory

The references to the entity and content files should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd"
[
  <!ENTITY % all.entities SYSTEM "all-entities.ent">
  %all.entities;
]>
<book id="refman-5-1" lang="en">
  ...
  <xi:include href="preface.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  ...
  <xi:include href="../refman-common/ha.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude"/>
  ...
</book>
```

Similar principles apply to entity files that reference other entity files. Suppose that the `all-entities.xml` entity file in the `refman-5.1` directory refers to several other entity files:

- `fixedchars.ent`, located in the `../common` directory
- `urls.ent`, located in the `../refman-common` directory
- `versions.ent`, located in the `refman-5.1` directory

`all-entities.xml` can be written like this:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  This file names all the entity files needed by .xml files in the
  current directory.  All ENTITY declarations should be given
  first, followed by references to the those entities.
-->
<!ENTITY % fixedchars.entities SYSTEM "../common/fixedchars.ent">
<!ENTITY % urls.entities       SYSTEM "../refman-common/urls.ent">
<!ENTITY % versions.entities   SYSTEM "versions.ent">
%fixedchars.entities;
%urls.entities;
%versions.entities;
```

> **Note**
>
> The `ENTITY` lines that define the entity files should all precede the lines that refer to the entity definitions. This is required because the parser used by our dependency generator `xmldepend.pl` gets confused otherwise and notices only the first definition. The result is that dependency information is incomplete.

There are certain ways in which you should *not* refer to files in other directories:

- Do not use absolute path names to name entity files or files in XInclude directives. The path names might be correct for your system, but they won't be correct for other people.
- Do not set up symbolic links to the files in the current directory and then refer to the symlinks as though the files are located in the current directory. This strategy works on Unix, but fails on Windows.

4.4.3. Content of Chapter Files

The term "chapter file" as used here applies to files that contain chapters, but also to those that contain a preface or appendix. All appear at the same level in a table of contents.

Chapter files have an XML declaration and a `DOCTYPE` declaration that contains references to XML entity files, just like the main document. The root element should be appropriate for the file content (either `<preface>`, `<chapter>`, or `<appendix>`), and the root element named in the `DOCTYPE` must match. For example, a file that contains an appendix might begin like this:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE appendix PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd"
[
... references to entity files ...
]>
<appendix id="restrictions">
...
```

The file then continues with its title and all the other elements that make up its content (paragraphs, tables, lists, and so forth).

All levels below the preface/chapter/appendix level are sections, written using `<section>` elements. Nested sections also use `<section>`. Here is an example of how to nest sections within a chapter:

```
<chapter>
<section>
(This is a section)
<section>
 (This is a subsection)
 <section>
  (This is a subsubsection)
 </section>
</section>
</section>
<section>
(This is a section)
</section>
</chapter>
```

We don't use the `<sect1>` through `<sect6>` elements because `<section>` can be handled more flexibly. For example, it's easier to promote or demote a section a level when all sections use the `<section>` tag rather than numbered section tags. Also, there is no limit to the depth of `<section>` element nesting.

Chapter files are “standalone.” That is, they are well-formed XML files, even if not included into the main document. Among other things, this means you can validate a chapter file separately, and you might generate output using just a particular chapter file (not the entire document). For example, we could create a PDF version of just the Cluster chapter from the *Reference Manual*.

4.4.4. Entities and Entity Files

We use entities for several purposes. Some entities are fixed and do not vary between documents. Some entities are version-specific (for documents such as the *Reference Manual* for which there are several versions). Examples of these entity types:

- Character entities are fixed but enable us to refer to special characters by name rather than by knowing their numeric encoding. An example of an entity defined in this file is `<!ENTITY ouml "&mp;#246;">`, which defines an entity for the letter `ö`.
- Some terms (such as URLs) occur multiple times but must be the same each time. By using an entity for a URL, it appears uniformly throughout a document. If the URL changes, it is easy to make the change by modifying the entity definition.
- Version numbers change occasionally. By using entities, we can update the entity definition so that the change takes effect throughout a document. This type of entity is used for the current release number in the various *Reference Manuals*. We update these when the release number changes.

To make it easy to refer to these entities within XML files, we group them into entity files (named with an `.ent` extension) and refer to the files via system entities in `DOCTYPE` declarations. The location of an entity file depends on the document for which it is used. If a document's files are all in a single directory, its entity files will be there as well. If a document's files are located in multiple directories, the location of its entity files depends on the particular directory layout.

Our convention is to place all definitions for the entities needed by the XML files in a given directory in a file named `all-entities.ent`. Then the `DOCTYPE` declaration for those XML files defines and references the single entity file `all-entities.ent`. For example, the beginning of a chapter file is written like this:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.3//EN"
"http://www.oasis-open.org/docbook/xml/4.3/docbookx.dtd"
[
  <!ENTITY % all.entities SYSTEM "all-entities.ent">
  %all.entities;
]>
```

The use of `all-entities.ent` makes the `DOCTYPE` declaration easier to write, Also, if the set of required entities changes, the change needs to be made only in `all-entities.ent`.

`all-entities.ent` can refer to other entity files. Suppose that the XML files in the directory need the character entities defined in the `fixedchars.ent` file of the `common` directory, the URLs defined in the `urls.ent` file of the `../refman-common` directory, and the directory-specific version numbers defined in the `versions.ent` file of the main document directory. In that case, `all-entities.ent` can be written like this:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  This file names all the entity files needed by .xml files in the
  current directory.  All ENTITY declarations should be given
  first, followed by references to the those entities.
-->
```

```
<!ENTITY % fixedchars.entities SYSTEM "../common/fixedchars.ent">
<!ENTITY % urls.entities       SYSTEM "../refman-common/urls.ent">
<!ENTITY % versions.entities   SYSTEM "versions.ent">
%fixedchars.entities;
%urls.entities;
%versions.entities;
```

> **Note**
>
> The `ENTITY` lines that define the entity files should all precede the lines that refer to the entity definitions. This is required because the parser used by our dependency generator `xmldepend.pl` gets confused otherwise and notices only the first definition. The result is that dependency information is incomplete.

The following sections provide additional detail about individual entity files.

4.4.4.1. `fixedchars.ent`

The `fixedchars.ent` file contains entity definitions for special characters such as accented characters or math symbols. If you have an editor that enables you to enter such characters directly, you can do so. (But remember that you should be using UTF-8 when editing document files.) If your editor does not have this capability, write special characters using the entity definitions as listed in `fixedchars.ent`. For example, to write a math minus character or a left single curly quote, use a `&minus;` or `&lsquo;` entity, which appear in formatted output as – or ‘.

Sample of `fixedchars.ent` content:

```
<!ENTITY auml "&#228;">
<!ENTITY ouml "&#246;">
<!ENTITY uuml "&#252;">
<!ENTITY Auml "&#196;">
<!ENTITY Ouml "&#214;">
<!ENTITY Uuml "&#220;">
```

> **Note**
>
> Most of our documents use the `fixedchars.ent` file, so it is located in the `common` directory. Should you need to add a new entity, add it to the master copy of `fixedchars.ent` in the `mysqldoc` repository, and then use the master copy to update the `fixedchars.ent` file in the `common` directory of any other documentation repositories.

4.4.4.2. `urls.ent`

This file contains entity definitions for commonly used URLs. These URLs are not likely to change, but that does happen on occasion. If that does occur, we update the definition in the `urls.ent` file and no edits to `.xml` files are necessary.

Sample of `urls.ent` content:

```
<!ENTITY base-url-docs "http://dev.mysql.com/doc/">
<!ENTITY base-url-downloads "http://dev.mysql.com/downloads/">
<!ENTITY base-url-refman "http://dev.mysql.com/doc/refman">
<!ENTITY base-url-forum-list "http://forums.mysql.com/list.php">
<!ENTITY base-url-uploads "ftp://ftp.mysql.com/pub/mysql/upload/">
```

The entity names begin with “base” because they are often used for the first part of a longer URL. For example, to refer to the changelog section of the online *Reference Manual*, write this:

```
<ulink url="&base-url-docs;mysql/en/news.html"/>
```

Which expands to this:

```
<ulink url="http://dev.mysql.com/doc/mysql/en/news.html"/>
```

4.4.4.3. `versions.ent`

The `versions.ent` file exists for documents such as the *Reference Manual* that exist in multiple versions (for MySQL 5.0, MySQL 5.1, and so on). Every version-specific document directory will have a `versions.ent` file. Each `versions.ent` file contains definitions for the same set of entities, but the entity values are specific to one version of the document. For example, the `versions.ent` file for the *MySQL 5.1 Reference Manual* looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  This file contains only entities for which the value varies per
  version of the manual.
-->
<!ENTITY title-refman-previous "MySQL 5.0 Reference Manual">
```

```
<!ENTITY current-series "5.1">
<!ENTITY previous-series "5.0">
<!ENTITY next-series "5.2">
<!ENTITY current-version "5.1.18-beta">
<!ENTITY current-maturity "beta">
```

As the current version of MySQL 5.1 changes, we update the value for the `current-version` entity, and possibly the `current-maturity` entity.

4.5. IDs Within a Document

In DocBook, you may assign an ID to virtually every element used. Although this might be useful in some cases, it also adds a lot of overhead to document, as those IDs need to be managed and handled. Therefore, we use IDs only for a few elements, particularly for the structural elements like `book`, `chapter`, `appendix`, and `section`.

IDs have to be unique within a document. Note that the term "document" may refer to more than just one file, for example when you include many XML files in one "master" file.

ID names may not contain spaces or special characters. They may (and should), however, contain dashes, to make them more readable.

4.6. Links (or References) Within a Document

We use the `<xref>`, `<link>`, and `<ulink>` elements to refer (or link) to other resources. Note that we prefer `<xref>` over `<link>`.

Normally, we use the `<xref>` element, like this:

```
...
<para>
Here is a link that points to <xref linkend="some-id"/>.
</para>
<section id="some-id">
<title>Important Section</title>
</section>
```

In this example, the link description is taken from the title of the referenced section, so that in most output formats, it will look like this:

```
Here is a link that points to Important Section.
```

In output formats that can display the link description as "hot" (or clickable), "Important Section" becomes a hotspot. This is, for example, true for HTML and PDF. In output formats where this is not possible (like printed books), our XSL stylesheets take care of adding additional information, so that the above example might look like this:

```
Here is a link that points to Section 10.5 Important Section.
```

You should use the `<link>` element only if you have to use text other than the linked element's title as the link description, like in the following example:

```
...
<para>
Here is a link that points to an <link linkend="some-id">important section</link>.
</para>
<section id="some-id">
<title>Important Section</title>
</section>
```

For both `<xref>` and `<link>`, *do not use numbers to refer to elements!* For example, do not use something like this:

```
See  <link linkend="some-id">10.5</link>.
```

For information about (automatic) numbering, see Section 5.2, "Guidelines for Numbering".

Also, do not add additional information like the words "chapter" or "section" to your link. For example, do not use something like this:

```
See section <link linkend="some-id">Important Section</link>.
```

Our XSL stylesheets take care of adding such additional information *in the correct language* (bear in mind translations) when appropriate.

For links referring to resources other than XML documents, you should use the `<ulink>` element, which has the following syntaxes:

```
<ulink url="target-resource">descriptive text</ulink>
<ulink url="target-resource"/>
```

The second form can be used if the content of the element is the same as the URL attribute. Here is an example:

```
The MySQL Reference Manual is
<ulink url="http://dev.mysql.com/doc/mysql/">available online</ulink>
or can be downloaded in a number of formats. For example, you can find the
PDF version here:
<ulink url="http://dev.mysql.com/get/Downloads/Manual/manual.pdf/from/pick"/>
```

4.7. Quotes

The rules for entering quote characters are pretty straightforward:

- Double quotes: `<quote>example</quote>`
- Double quotes in program listings: `"` (entered verbatim)
- Single quotes: `'` (entered verbatim)
- Backticks: `` ` `` (entered verbatim)

See Section 4.8.33, “quote”.

4.8. DocBook Elements We Use

4.8.1. answer

Used for answers to a question posed in a `quandaset`.

http://docbook.org/tdg/en/html/answer.html

4.8.2. caution

Used where a procedure may result in data loss if not carried out correctly. Used to indicate great care should be taken.

http://docbook.org/tdg/en/html/caution.html

```
<caution>
  <para>
    The following procedure should be carried out with care...
  </para>
</caution>
```

4.8.3. colspec

Used to denote column widths within an `informaltable`.

```
<informaltable>
<tgroup cols="2">
<colspec colwidth="30*"/>
<colspec colwidth="50*"/>
<tbody>
...
```

http://docbook.org/tdg/en/html/colspec.html

4.8.4. command

Used for program names.

```
To dump all databases, you can use <command>mysqldump <option>- -
all-databases</option></command>.
```

http://docbook.org/tdg/en/html/command.html

4.8.5. email

Used for email addresses.

```
<email>stefan@mysql.com</email>
```

http://docbook.org/tdg/en/html/email.html

4.8.6. emphasis

Used to emphasize important parts of text. If you want boldface, add a `role` attribute:

```
<emphasis role="bold">Note:</emphasis> Remember that ...
```

http://docbook.org/tdg/en/html/emphasis.html.

4.8.7. entry

Used for column entries within rows of a table.

4.8.8. figure

Used as a wrapper to include graphics (and possibly other media).

```
<figure>
 <title>Caption of the graphic</title>
 <mediaobject>
   <imageobject>
     <imagedata fileref="screenshot.en.png" lang="en"/>
   </imageobject>
   <imageobject>
     <imagedata fileref="screenshot.de.png" lang="de"/>
   </imageobject>
   <textobject>
     <phrase lang="en">A screenshot</phrase>
   </textobject>
   <textobject>
     <phrase lang="de">Ein Mattscheiben-Foto</phrase>
   </textobject>
 </mediaobject>
</figure>
```

Every figure should be preceded by a specific in-text reference (for example: `see Figure figure-title`, `Figure figure-title shows`, etc.). Figures should not be introduced with colons or phrases like "in the following figure," or "as shown in this figure." Figures float, so the lack of specific in-text references may cause incorrect placement of figures.

As for capitalization of figure captions, see Guidelines for Capitalization of Terms. Do not use periods at the end of captions.

See also Section 4.8.23, "mediaobject". As for file names of graphics, see Section 4.10, "File Names and Guidelines for Graphics".

http://docbook.org/tdg/en/html/figure.html.

4.8.9. filename

Used for files, directories, and paths.

```
Change the server configuration using the <filename>my.cnf</filename> option file.
```

http://docbook.org/tdg/en/html/filename.html

4.8.10. graphic

DEPRECATED — DO NOT USE: Formerly used to include graphics.

http://docbook.org/tdg/en/html/graphic.html

4.8.11. guibutton

Used for buttons in a graphical user interface (GUI).

```
Click <guibutton>OK</guibutton> to select that option.
```

http://docbook.org/tdg/en/html/guibutton.html

4.8.12. guilabel

Used for labels in a graphical user interface (GUI).

```
Those options are listed in the group labeled <guilabel>Server Settings</guilabel>.
```

http://docbook.org/tdg/en/html/guibutton.html

4.8.13. guimenu

Used for menus and menu items (we do not use `<guimenuitem>` and `<guimenusubmenu>` yet) in a graphical user interface (GUI).

```
To open another program window, select <guimenu>File</guimenu>, <guimenu>New Instance
Connection ...</guimenu>.
```

http://docbook.org/tdg/en/html/guimenu.html

4.8.14. imagedata

See Section 4.8.23, “mediaobject”.

4.8.15. imageobject

See Section 4.8.23, “mediaobject”.

4.8.16. indexterm

Used to mark index entries.

```
<para>
The Tiger
<indexterm>
 <primary>Big Cats</primary>
 <secondary>Tigers</secondary>
</indexterm>
is a very large cat indeed.
</para>
```

See also primary and secondary.

http://docbook.org/tdg/en/html/indexterm.html

4.8.17. informaltable

Used for tables.

```
<informaltable>
 <tgroup cols="2">
  <colspec colwidth="30*"/>
  <colspec colwidth="50*"/>
  <tbody>
   <row>
    <entry>
     <emphasis role="bold">Operating System</emphasis>
    </entry>
    <entry>
     <emphasis role="bold">File-size Limit</emphasis>
    </entry>
   </row>
   <row>
    <entry>
     Linux-Intel 32-bit
    </entry>
    <entry>
     2GB, much more when using LFS
    </entry>
   </row>
  </tbody>
 </tgroup>
</informaltable>
```

See also tgroup, tbody, row, and entry.

http://docbook.org/tdg/en/html/informaltable.html

4.8.18. itemizedlist

Used for lists in which each entry is marked with a bullet or other dingbat.

```
<itemizedlist>
 <listitem><para>
  This is a list element.
 </para></listitem>
 <listitem><para>
  This is another list element.
 </para></listitem>
</itemizedlist>
```

See also listitem.

http://docbook.org/tdg/en/html/itemizedlist.html

4.8.19. keycap

Used to wrap the text printed on a key on a keyboard.

```
Press <keycap>CTRL</keycap>, <keycap>ALT</keycap>, and <keycap>DELETE</keycap> to invoke
the Windows Task Manager.
```

http://docbook.org/tdg/en/html/keycap.html

4.8.20. link

Used to reference resources within a document. Note that we preferably use the `xref` element for this.

4.8.21. listitem

Used for items of a list. Note that a `listitem` should always contain a `para` (paragraph) element. This is needed for better rendering in various output formats.

See also itemizedlist and orderedlist.

http://docbook.org/tdg/en/html/listitem.html

4.8.22. literal

Used for a variety of purposes, like character set names (`<literal>latin1</literal>`) or SQL statements and SQL functions (`<literal>SELECT * FROM tbl_name</literal>`).

Here is a list of types of terms that should be wrapped in the `<literal>` element. This list is by Pearson. The way Pearson describes these terms is that they are “monospaced” or “mono.” In some cases, we use elements other than `<literal>` (this is noted in parens). When producing output for Pearson, we map *all* these terms onto mono. For other types of output, we may display some of them differently. For example, the DocBook PDF stylesheets display <command> in bold and <filename> in italic.

- arguments
- arrays
- class names
- code commands (in some cases it is more appropriate to use `programlisting`)
- constructors
- data types
- directives
- DOS programs used as commands (we use `command`, and `option` for program options)

- email addresses (we use `email`)
- events
- fields
- flags
- functions
- HTML tags (remember to properly encode instances of `<` and `&`)
- Internet addresses (we use `ulink`)
- keywords
- logical operators
- loops
- methods
- newsgroup names (we use `ulink`)
- objects
- onscreen messages
- parameters
- pointers
- procedures
- properties
- shell scripts
- statements
- structures
- symbolic constants
- system prompts
- switches (and options)
- units
- URLs (we use `ulink`)
- values
- variables

http://docbook.org/tdg/en/html/literal.html

4.8.23. mediaobject

Used for referencing images and other media. This element serves as a wrapper around these elements:

- `imageobject`: Used to wrap image data. An imageobject can wrap exactly one image file. If more than one image file should be included in the mediaobject, you have to specify more than one imageobject. As in the following example, this could be done for multiple languages. An imageobject contains exactly one `imagedata` tag.
- `textobject`: Used to wrap a description that shows up as the `alt` attribute to the HTML `img` tag. `textobject` contains exactly one `phrase`. If more `phrase`s are needed, you need to specify more than one `textobject`.

Apart from `imageobject` and `textobject`, `mediaobject` may contain other XML elements for media files, such as `videoobject` and `audioobject`.

```
<mediaobject>
  <imageobject>
    <imagedata fileref="screenshot.en.png" lang="en"/>
  </imageobject>
  <imageobject>
    <imagedata fileref="screenshot.de.png" lang="de"/>
  </imageobject>
  <textobject>
    <phrase lang="en">A screenshot</phrase>
  </textobject>
  <textobject>
    <phrase lang="de">Ein Mattscheiben-Foto</phrase>
  </textobject>
</mediaobject>
```

http://docbook.org/tdg/en/html/mediaobject.html

4.8.24. option

Used for options of software commands.

```
To dump all databases, use the <option>--all-databases</option> option of the
<command>mysqldump</command> program.
```

http://docbook.org/tdg/en/html/option.html

4.8.25. orderedlist

Used for lists in which each entry is marked with a sequentially incremented label.

```
<orderedlist>
 <listitem><para>
  This is a list element. It is prefixed with 1.
 </para></listitem>
 <listitem><para>
  This is another list element. It is prefixed with 2.
 </para></listitem>
</orderedlist>
```

See also listitem and itemizedlist.

http://docbook.org/tdg/en/html/orderedlist.html

4.8.26. para

Used for paragraphs.

```
<para>This is a paragraph.</para><para>This is another paragraph.</para>
```

http://docbook.org/tdg/en/html/para.html

4.8.27. phrase

See Section 4.8.23, “mediaobject”.

4.8.28. primary

Used for primary index entries. See also <indexterm> and <secondary>.

http://docbook.org/tdg/en/html/primary.html

4.8.29. programlisting

Used for SQL examples and other code listings.

In the output created from the XML (HTML, PDF, CHM, and so forth), everything between `<programlisting>` and `</programlisting>` is displayed literally. When you indent a program listing the indentation also shows literally.

Within `programlisting` elements, make sure that code lines do not run longer than 72 characters. Code lines should be inden-

ted using spaces, not tabs.

http://docbook.org/tdg/en/html/programlisting.html

4.8.30. qandaentry

Used for question/answer sets within a quandaset.

http://docbook.org/tdg/en/html/qandaentry.html

4.8.31. quandaset

Used for lists consisting of questions and answers, and can be divided into sections. Every entry in a `<qandaset>` must contain a `question`, but `answer` elements are optional (some questions have no answers), and may be repeated (some questions have more than one answer). Common uses for this element include reader questionnaires and lists of “Frequently Asked Questions” (FAQs).

```
<qandaset defaultlabel='qanda'>
 <qandaentry>
  <question>
   <para>
    To be, or not to be?
   </para>
  </question>
  <answer>
   <para>
    That is the question.
   </para>
  </answer>
 </qandaentry>
</qandaset>
```

The `defaultlabel` attribute identifies the default label that should be used for `question` and `answer` elements:

- `quanda`: Questions are labeled “Q:” and Answers are labeled “A:”. Other similar labels may be substituted, for example, the words might be spelled out, “Question:” and “Answer:”, and the actual characters or words used are dependent on the language.
- `number`: The entries are enumerated.
- `none`: No distinguishing label precedes questions or answers.

If no value is specified, the implied presentation may be any one of these, as defined by the stylesheet. Note that each question and answer can explicitly define a label, regardless of the default label specified.

See also quandaentry, question, and answer.

http://docbook.org/tdg/en/html/qandaset.html

4.8.32. question

Used for questions in a quandaset.

http://docbook.org/tdg/en/html/question.html

4.8.33. quote

Double quotes (“example”) are entered like this:

```
<quote>example</quote>
```

Exception: In program listings, they're written verbatim:

```
"example"
```

The difference between “example” and "example" might not be visible in every output format (particularly in HTML). For printed material, however, it's important to use the `<quote>` element wherever appropriate.

For writing curly-quoted single literal characters, you should use the `&lsquo;` and `&rsquo;` XML entities. Example:

```
Terminate an SQL statement with a &lsquo;<literal>;</literal>&rsquo; character.
```

That example will look like this:

```
Terminate an SQL statement with a “;” character.
```

Curly single quotes is a pretty common convention with O'Reilly and Pearson.

Single quotes (`'`) and **backticks** (`` ` ``) are always entered verbatim.

http://docbook.org/tdg/en/html/quote.html

4.8.34. remark

Used for a remark (or comment) intended for presentation in a draft manuscript.

```
<remark>[SH] This is a remark.</remark>
```

Remarks may be placed almost everywhere; for a complete list of elements that may contain remarks, see http://docbook.org/tdg/en/html/remark.html.

Remarks can be processed by XSL stylesheets in a way that they are not visible in generated output. Furthermore, it is possible to hide remarks in HTML and CHM output by means of CSS stylesheets (in HTML, the remark will still be visible in the page source code), without even processing them using XSLT. This means that there is no reason to remove remarks, unless those are not needed any more.

We start remarks with the initials of the author, put in square brackets. If you are commenting on a remark, you should not use another remark, but rather do it like this: `<remark>[SH] To be, or not to be? [WS] That is the question.</remark>`

http://docbook.org/tdg/en/html/remark.html

4.8.35. replaceable

Used for content that may or must be replaced by the user.

```
SELECT * FROM <replaceable>tbl_name</replaceable>
```

Here, the user is expected to replace *`tbl_name`* with the name of an actual table when issuing the `SELECT` statement. Pearson notes: All placeholders should be in mono and italics. A placeholder is pseudo text you put in code (a file name, or parameter) where the reader will need to fill in their actual info to complete.

http://docbook.org/tdg/en/html/replaceable.html

4.8.36. row

Used for rows in tables.

http://docbook.org/tdg/en/html/row.html

4.8.37. secondary

Used to mark secondary index entries.

See also primary and indexterm (for an example).

http://docbook.org/tdg/en/html/secondary.html

4.8.38. tbody

Used to mark the table body in tables.

http://docbook.org/tdg/en/html/tbody.html

4.8.39. tgroup

Used to mark column groups in tables.

http://docbook.org/tdg/en/html/tgroup.html

4.8.40. textobject

See Section 4.8.23, "mediaobject".

4.8.41. title

Used for the text of the title of a formal block-level element (for example `book`, `chapter`, or `figure`).

```
<section id="my-section"><title>The Title</title> ...
```

http://docbook.org/tdg/en/html/title.html

4.8.42. ulink

Used for referencing resources external to the document, particularly for URLs.

```
My favorite literature is the <ulink url="http://dev.mysql.com/doc/mysql/en">MySQL Ref-
erence Manual</ulink>.
```

See also Section 4.6, "Links (or References) Within a Document".

http://docbook.org/tdg/en/html/ulink.html

4.8.43. userinput

Used for data entered by the user.

```
At the system prompt, enter <userinput>xyzzy</userinput> to gain access to the system.
```

http://docbook.org/tdg/en/html/userinput.html

4.8.44. xref

Used to refer to other resources. See Section 4.6, "Links (or References) Within a Document" for information on how to use this element.

http://docbook.org/tdg/en/html/xref.html

4.9. Using DocBook Elements That We Have Not Used Before

After you have worked with our documentation for a while, you'll have a pretty good idea of which DocBook elements we use. If you're unsure, one way to see which DocBook elements we are actually using is to check Section 4.8, "DocBook Elements We Use". Another way to check is to look in the `tools/xmlformat.conf` file. In most cases, if we use an element, it is listed in this file.

Before using a formerly unused element in documents, you must take some steps to ensure that the element will not cause problems:

1. List the element in `tools/xmlformat.conf` according to how it should be formatted during reformatting operations.
2. Handle the new element in those XSL transforms that need to know about it. For output formats that are based on the standard DocBook XSL stylesheets, there should be no problems with the new element. However, some formats are produced with "local" transforms that we have written ourselves, and these transforms typically handle only those elements that we are already using. Examples: Transforms to produce Texinfo, plain text, or help-table content.

   If you use a new element in a document without first modifying these local transforms, they will either throw up their hands and die when they encounter the element, or ignore it (and thus lose content). If the element is significant for any of these transforms, they must be changed to handle it properly.
3. After you incorporate the new element into document content, verify that it is handled properly for various output formats. The way *not* to do this is to commit the document changes and then wait to see if they blow up the auto-build processes on the documentation server! Instead, run `make` locally on your own machine to verify that output looks okay. Generate at least one of the HTML formats, one of the PDF formats, plain text format, and the formats that use our own local transforms.

4.10. File Names and Guidelines for Graphics

Graphics should have a file name that relates to the chapter or section where the graphic appears. At least, they should contain the name of the document (the `<book>`), like in this example:

```
<graphic fileref="cygwin-packetmanager.png" format="PNG" lang="en"/>
```

If the document is likely to become translated, and if we need different graphics for each individual language, then the file name should also include the language, like in this example:

```
<graphic fileref="cygwin-packetmanager.en.png" format="PNG" lang="en"/>
```

Note that the language code is prefixed using dots (`.en`) before the file extension (`.png` in this example). The following two-character ISO language codes should be used:

- `ar`: Arabic
- `de`: German
- `en`: English/American
- `es`: Spanish
- `fr`: French
- `hi`: Hindi
- `it`: Italian
- `ja`: Japanese
- `pt`: Portuguese
- `ru`: Russian
- `zh`: Chinese

For a complete list, see the ISO 639 2-letter codes.

Figures must be 640 by 480 resolution, 256 color or 24-bit color, pcx format. This is a Pearson requirement.

4.11. Special Markup

This section describes special markup that we use in the *MySQL Reference Manual*:

- Markup that distinguishes sections of a document to use for generating Unix man pages.
- Markup that distinguishes *Reference Manual* content to be loaded into help tables in the `mysql` database. The server uses these tables to support its server-side help capability.
- Markup for functions, operators, SQL statements, and so for, for which links to the appropriate descriptions can be generated automatically.

4.11.1. Markup for Unix Man Pages

Some sections of the *MySQL Reference Manual* describe programs, such as the sections for `mysql` and `mysqladmin`. To avoid writing descriptions for the programs both in the manual and in separate Unix-style man pages (which are written using `troff`/`groff` markup), we generate man pages directly from the manual DocBook source. This strategy can be used for other documents as well (for example, the *MySQL Test Framework Manual*).

Formerly, MySQL distributions contained a set of handwritten man page files, but these were contributed by third parties and rarely were updated, so they became obsolete. Also, there were some programs for which no man pages had ever been written. By generating the man pages from the *Reference Manual*, we can keep them up to date easily, and we have a man page for every program described in the manual.

Note

> If the set of man pages generated from the manual changes, it's necessary to let the build team know about it. (For example, they need to update the `.spec` files used for the RPM distributions.) Examples of changes requiring notification of the build team:

- You mark up a new man page in the manual.
- You change the "volume" for a man page. For example, `mysqld(1)` becomes `mysqld(8)`.
- You remove a section from which a man page formerly was created. For example, `safe_mysqld` is no longer present in MySQL 5.1.

The DocBook XSL stylesheets have a man page mode that can be used for generating man page output. The markup described here requires at least version 1.69.0 of the stylesheets, because man page support was improved markedly in that version, and we use features not supported in older versions.

The DocBook XSL stylesheets man page mode requires that man page sections be marked up using `<refentry>` elements. One problem with using `<refentry>` is that the DocBook DTD prohibits `<refentry>` and `<section>` from appearing at the same level within a document if the document is to be considered valid. To work around this, we wrap `<refentry>` elements within "fake" `<section>` elements so that the source refman files pass validation. For output generation, we strip off these fake element tags, and then either pass through the `<refentry>` elements unchanged (for man page output), or map them onto regular `<section>` elements (for all other output formats).

The following template shows how to mark up a section so that a man page can be generated from it. The `<ref...>` tags and their subelements should be used in the order shown, because the DTD doesn't allow much flexibility and variations from the order shown typically result in validation errors.

There must be at least one `<refsection>` element per `<refentry>`. By convention, the first should be titled "Description". If there are further subdivisions to be made, those should also be marked up as `<refsection>` elements. It is permissible to nest `<refsection>` elements.

Here is a prototype for a manual section that generates a man page:

```
<section id="fake-id-for-PROGNAME-manpage-section-wrapper">

  <title>fake title for PROGNAME man page section wrapper</title>

  <refentry id="PROGNAME">

  <!-- initial index terms go here -->

    <refmeta>
      <refentrytitle>PROGNAME</refentrytitle>
      <manvolnum>VOLNUM</manvolnum>
      <refmiscinfo class="manual">MySQL Database System</refmiscinfo>
      <refmiscinfo class="source">MySQL</refmiscinfo>
      <refmiscinfo class="version">VERSION</refmiscinfo>
      <refmiscinfo class="refman">PURPOSE-FOR-TITLE</refmiscinfo>
    </refmeta>

    <refnamediv>
      <refname>PROGNAME</refname>
      <refpurpose>PURPOSE-FOR-MANPAGE</refpurpose>
    </refnamediv>

    <refsynopsisdiv>
      <cmdsynopsis>
        <command>PROGNAME ARGUMENT-SYNTAX</command>
      </cmdsynopsis>
    </refsynopsisdiv>

    <refsection id="PROGNAME-description">

      <title>Description</title>

      <para>
        Description of the program, how to use it, its allowable
        command-line options, and so forth.
      </para>

    </refsection>

  </refentry>

</section>
```

In the template, make the following substitutions:

- `PROGNAME` = program name

If the program name occurs within an element attribute value, underscores in the name should be written as dashes. For example, `mysql_config` becomes `mysql-config`.

- `VOLNUM` = Unix man page "volume" number

  Most of our man pages are part of volume 1. The `mysqld` and `mysqlmanager` man pages are in volume 8.

- `PURPOSE-FOR-TITLE`, `PURPOSE-FOR-MANPAGE` = short phrase describing program purpose

  The purpose string will appear with the program name in the output, but which of the two strings to use depends on the output format. Man pages typically begin with a name section that looks like this, where words in the description are not title-capped:

  ```
  .SH "NAME"
  myprog \- a program for doing this and that
  ```

  Here, "a program for doing this and that" is the text to use for `PURPOSE-FOR-MANPAGE`.

  In non-man page output, the name and description become a section title, so words in the description should be title-capped:

  ```
  <title>myprog - A Program for Doing This and That<title>
  ```

  Here, "A Program for Doing This and That" is the text to use for `PURPOSE-FOR-TITLE`.

  To accommodate both output formats, we include both description strings in the man page markup. The title-capped version is included using a `<refmiscinfo>` element (with a `class` attribute of `"refman"`) that does not appear in man page output. The `<refmiscinfo>` element does not allow much internal markup. In particular, `<literal>` and `<command>` are not allowed.

  In some cases, a manual page describes more than one program. For example, the same material in the *MySQL 4.1 Reference Manual* describes both `myisamchk` and `isamchk`. To handle this, include one `<refnamediv>` element for each program. The first such element should be for the primary program described by the man page. For example:

  ```
  <refnamediv>
    <refname>myisamchk</refname>
    <refpurpose>MyISAM table-maintenance utility</refpurpose>
  </refnamediv>

  <refnamediv>
    <refname>isamchk</refname>
    <refpurpose>ISAM table-maintenance utility</refpurpose>
  </refnamediv>
  ```

  When the file is processed, the man page stylesheets will produce a file named `myisamchk.1` that contains the man page contents, and another "stub" file named `isamchk.1` that contains a `.so` directive that simply includes the other page:

  ```
  .so man1/myisamchk.1
  ```

- `VERSION` = version of MySQL to which the manual applies (4.1, 5.0, 5.1, and so forth)

  If the document is unversioned, omit the version element.

- `ARGUMENT-SYNTAX` = syntax for any arguments that the program takes

  The `<cmdsynopsis>` value describes program invocation syntax. It appears in the `SYNOPSIS` section of man page output. It does not appear in non-man page output.

- Where the "initial index terms" comment appears, you may add `<indexterm>` elements for index terms that apply to the program as a whole. Index terms for specific features probably are better placed near the text that describes the feature.

Output generation always begins with a preprocessing step. For non-man page output from a source document `manual.xml`, we generate `manual-prepped.xml`, and then run that through the appropriate DocBook XSL stylesheets. For man page output, we generate `manual-manprepped.xml`, and run that through the man page DocBook XSL stylesheets. The preprocessing in both cases is done using the `xsl.d/dbk-prep.xsl` transform, where the type of `<refentry>` handling to be done is handled by setting the value of the `$map.refentry.to.section` parameter. See the comments in `dbk-prep.xsl` for more information.

For man page generation, we also have a postprocessing step that acts on the set of mapages as a whole. The postprocessing script is `tools/fixup-manpages.pl`. For each man page file, it adds this information:

- A `SEE ALSO` section that names every *other* man page and provides the URL to the online *Reference Manual*.
- An `AUTHOR` section that names MySQL AB and indicates that the software comes with no warranty.

For the `SEE ALSO` section to be generated properly, `fixup-manpages.pl` must be invoked with the names of *all* man pages as its arguments. `fixup-manpages.pl --help` shows the help message that describes the invocation syntax.

4.11.2. Help Tag Markup

Beginning with MySQL 4.1, the server supports a server-side help capability. This can be accessed from the `mysql` client by means of its `HELP` command. For example, if you enter the following command, the server returns a description for the `ISNULL()` function:

```
mysql> HELP ISNULL;
Name: 'ISNULL'
Description:
Syntax:
ISNULL(expr)

If expr is NULL, ISNULL() returns 1, otherwise it returns 0.
Examples:
mysql> SELECT ISNULL(1+1);
        -> 0
mysql> SELECT ISNULL(1/0);
        -> 1
```

The server gets help content from a set of tables in the `mysql` directory. These tables are loaded from information that is extracted from the *MySQL Reference Manual*.

Originally, help tags were placed in the Texinfo manual, based on an implementation devised by Victor Vagim. (The original Texinfo implementation and notes about the conversion to DocBook are described in another document, *MySQL Documentation Team Historical Notes*.)

To support this capability, *Reference Manual* documents contain embedded tags that mark content to be extracted and used for generating the contents of the help tables. These tags are based on `<remark>` elements, where the `role` attribute has a value of `help-xxx` that indicates tag type. Some tags also have a `condition` attribute to specify additional information. ("condition" has no real meaning in the context of help tagging. It was simply the most "generic" of the allowable `<remark>` attributes, so we press it into service for general use.)

The following discussion sets forth the allowable help tags and the rules for using them. You can examine the existing markup in the *Reference Manual* to see how the tags are used in practice.

These are the allowable help tags:

- Begin a category:

  ```
  <remark role="help-category" condition="CATEGORY_NAME"/>
  ```

- Begin a topic within the current category:

  ```
  <remark role="help-topic" condition="TOPIC_NAME"/>
  ```

- Specify keywords for the current topic:

  ```
  <remark role="help-keywords"> keyword1 keyword2 ... </remark>
  ```

- Specify syntax for the current topic:

  ```
  <remark role="help-syntax"/>
  ```

  Or:

  ```
  <remark role="help-syntax-begin"/>
  <remark role="help-syntax-end"/>
  ```

- Specify descriptive text for the current topic:

  ```
  <remark role="help-description"/>
  ```

Or:

```
<remark role="help-description-begin"/>
<remark role="help-description-end"/>
```

- Provide an example for the current topic:

```
<remark role="help-example"/>
```

Or:

```
<remark role="help-example-begin"/>
<remark role="help-example-end"/>
```

For any chapter that will contain help topics, you must include a `help-category` tag preceding any of those topics:

```
<remark role="help-category" condition="CATEGORY_NAME"/>
```

The *CATEGORY_NAME* value indicates the help category, and optionally the parent category. If the parent category is present, separate it from the category by an '@' character. Examples:

```
<remark role="help-category" condition="Functions"/>
<remark role="help-category" condition="String Functions@Functions"/>
```

The category tag sets the context for any following help topics in the chapter until the next `help-category` tag.

Within a category, you mark up each topic using a small set of tags. Each topic must begin with a `help-topic` tag:

```
<remark role="help-topic" condition="TOPIC_NAME"/>
```

The *TOPIC_NAME* value indicates the topic. Examples:

```
<remark role="help-topic" condition="ISNULL"/>
<remark role="help-topic" condition="DELETE"/>
<remark role="help-topic" condition="DROP DATABASE"/>
```

The topic tag sets the context for the following keyword, syntax, description, and example tags.

To associate keywords with the topic, use a `help-keywords` tag and list the keywords as the content of the remark:

```
<remark role="help-keywords"> keyword1 keyword2 ... </remark>
```

Syntax, descriptive text, and examples are marked up with `help-syntax`, `help-description`, and `help-example` tags.

If the content for syntax, a description, or an example comprises only a single XML element, you can just precede that element with the appropriate single tag just described. If the content requires multiple elements, you should use a pair of tags, one with a `-begin` tag and one with an `-end` tag. For example, a multiple-paragraph description must be tagged using paired tags that mark the beginning and end of the description:

```
<remark role="help-description-begin"/>

<para>Para 1 of description</para>
<para>Para 2 of description</para>
<para>Para 3 of description</para>

<remark role="help-description-end"/>
```

However, a single-paragraph description can be tagged in two ways. You can use either a single (unpaired) tag, like this:

```
<remark role="help-description"/>

<para>Para 1 of description</para>
```

Or you can use paired tags, like this:

```
<remark role="help-description-begin"/>

<para>Para 1 of description</para>

<remark role="help-description-end"/>
```

Here is a complete example of a marked-up topic:

```
<remark role="help-topic" condition="PURGE MASTER LOGS"/>

<remark role="help-keywords">
  BEFORE TO
</remark>

<remark role="help-syntax"/>

<programlisting>
PURGE {MASTER | BINARY} LOGS TO '<replaceable>log_name</replaceable>'
PURGE {MASTER | BINARY} LOGS BEFORE '<replaceable>date</replaceable>'
</programlisting>

<remark role="help-description-begin"/>

<para>
  Deletes all the binary logs listed in the log index prior to
  the specified log or date. The logs also are removed from the
  list recorded in the log index file, so that the given log
  becomes the first.
</para>

<remark role="help-description-end"/>

<para>
  Example:
</para>

<remark role="help-example"/>

<programlisting>
PURGE MASTER LOGS TO 'mysql-bin.010';
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
</programlisting>
```

For a given help topic, the `help-topic` tag must be present. The other tags (for keywords, syntax, description, or examples), all are optional, although it is rare for there to be no description.

Within help topic content, only the following XML elements are allowable:

- `<programlisting>`
- `<para>`
- `<itemizedlist>`
- `<orderedlist>`

In particular, table markup cannot be used. (DocBook table markup is complex and can be difficult to deal with.) Perhaps eventually we should try to handle tables. For now, we forbid them.

Do not put help markup in the changelogs. If a topic is worth marking up for help-table content, the content is worth stating in the main chapter files.

4.11.3. Markup for Automatic Link Generation

For references to certain objects that can be associated with a section or paragraph in a manual, we generate links automatically. The processing for this is handled in the `dbk-prep.xsl` transform to add `<link>` elements. In this way, we get auto-linking of object references to their descriptions, which is useful for those output formats that support hyperlinking (HTML, PDF). Auto-linking is done for these objects:

- We use the `<literal>` element with a `role` attribute to indicate the object type. We do this for SQL functions, operators, data types, and statements, and for C API functions.
- For `<command>` elements, a link is generated if a section with an ID matching the element content can be found.
- For `<option>` elements with a `role` attribute to indicate which program to which the option belongs, a link is generated to the option description.

For auto-link generation to occur, the `Makefile` for the directory containing the document must have the following line:

```
GENERATE_AUTOLINK = 1
```

For `<literal>` elements, the attributes that identify an object as one for which a link should be generated automatically are:

- The `role` attribute, if present, identifies the object type. The value should be `"func"` for an SQL function, `"op"` for an SQL operator, `"type"` for an SQL data type, `"stmt"` for an SQL statement, `"sysvar"` for a system variable, `"statvar"` for a status variable, `"sqlmode"` for a SQL mode, `"isolevel"` for a transaction isolation level, `"cfunc"` for a C API function, `"jointype"` for an optimizer join type (as seen in `EXPLAIN` output), or `"priv"` for an access privilege.
- A `condition` attribute may be added as a hint. The `dbk-prep.xsl` transform tries to determine the ID to use for the link, using the rules described shortly. If it cannot determine the ID, the `condition` attribute indicates which ID to use.

When `dbk-prep.xsl` determines the ID for the `linkend` attribute when generating `<link>` elements, the name-to-ID mapping rule depends on the function type.

- SQL function: Use element content up to the leading open parenthesis, with underscores converted to dashes and letters converted to lowercase. Add a `function_` prefix. Some functions can be written without parentheses, so if no opening parenthesis is found, use the entire element content.

  These references both will link to an ID of `function_current-timestamp`:

  ```
  <literal role="func">CURRENT_TIMESTAMP()</literal>
  <literal role="func">CURRENT_TIMESTAMP</literal>
  ```

- SQL operator: Use the entire element content, with underscores and spaces converted to dashes and letters converted to lowercase. Add an `operator_` prefix.

  This reference will link to an ID of `operator_binary`:

  ```
  <literal role="op">BINARY</literal>
  ```

- SQL data type: Use the entire element content, with underscores and spaces converted to dashes and letters converted to lowercase.

  This reference will link to an ID of `datetime`:

  ```
  <literal role="type">DATETIME</literal>
  ```

  Data type sections do not always correspond to the type name, so the link generator actually uses a lookup table to map types to section IDs.

- SQL statement: Use the entire element content, with underscores and spaces converted to dashes and letters converted to lowercase.

  This reference will link to an ID of `create-table`:

  ```
  <literal role="stmt">CREATE TABLE</literal>
  ```

- `INFORMATION_SCHEMA` table: Use the entire element content (stripping off any `INFORMATION_SCHEMA.` qualifier), with underscores converted to dashes and letters converted to lowercase. Add a `-table` suffix.

  Either of these references will link to an ID of `triggers-table`:

  ```
  <literal role="is">TRIGGERS</literal>
  <literal role="is">INFORMATION_SCHEMA.TRIGGERS</literal>
  ```

- System variable or status variable: Use the first word of the content, without lettercase conversion. Add a `sysvar_` or `statvar_` prefix.

  ```
  <literal role="sysvar">autocommit</literal>
  <literal role="statvar">Key_read_requests</literal>
  ```

- SQL mode: Use the entire element content, with underscores converted to dashes and letters converted to lowercase.

  This reference will link to an ID of `sqlmode_ansi_quotes`:

  ```
  <literal role="sqlmode">ANSI_QUOTES</literal>
  ```

- Transaction isolation level: Use the entire element content, with spaces converted to dashes and letters converted to lowercase. Add an `isolevel_` prefix.

  Either of these references will link to an ID of `isolevel_read-committed`:

  ```
  <literal role="isolevel">READ COMMITTED</literal>
  <literal role="isolevel">READ-COMMITTED</literal>
  ```

- C API function: Use element content up to the leading open parenthesis, with underscores converted to dashes.

  This reference will link to an ID of `mysql-real-connect`:

  ```
  <literal role="cfunc">mysql_real_connect()</literal>
  ```

- Optimizer join type: Use the entire element content, with letters converted to lowercase. Add a `jointype_` prefix.

  This reference will link to an ID of `jointype_eq_ref`:

  ```
  <literal role="jointype">eq_ref</literal>
  ```

- Access privilege: Use the entire element content, with letters converted to lowercase and spaces converted to dashes. Add a `priv_` prefix.

  This reference will link to an ID of `priv_show-view`:

  ```
  <literal role="priv">SHOW VIEW</literal>
  ```

In some cases, the content of the `<literal>` element is unsuitable for constructing an ID. In such cases, include a `condition` attribute as a hint about the ID to use. It is not necessary to add the leading `function_` or `operator_` for the `condition` attribute value. That can be determined automatically as needed because the `role` attribute value determines whether the element refers to a function or an operator.

A `condition` attribute is needed when the element contains any of the following:

- Non-alphanumeric text:

  ```
  Wrong: <literal role="op">*</literal>

  Right: <literal role="op" condition="times">*</literal>
  ```

- Extra text within the element that confuses the content-to-ID conversion:

  ```
  Wrong: <literal role="op">expr REGEXP pat</literal>

  Right: <literal role="op" condition="regexp">expr REGEXP pat</literal>
  ```

- Synonyms, when a function or operator that has multiple names but the description for all is given under a single ID. For example, `AsWKT()` is a synonym for `AsText()`, but the description for both is listed under an ID of `function_astext`:

  ```
  Wrong: <literal role="func">AsText()</literal>
         <literal role="func">AsWKT()</literal>

  Right: <literal role="func">AsText()</literal>
         <literal role="func" condition="astext">AsWKT()</literal>
  ```

If a function reference is to some function that is not described in the manual, no attributes should be present in the `<literal>` tags so that no auto-linking will be attempted. For example, the `printf()` function in the C standard I/O library should be written as `<literal>printf()</literal>`.

For `<command>` elements, there is no special `role` attribute required. If the element content matches the `id` attribute for a `<section>` or `<refsection>` element, the link is generated. The first word within the element is used, minus any `.exe` suffix that might be present.

For example, the following elements all generate links to the section that describes the `mysqladmin` program:

```
<command>mysqladmin</command>
<command>mysqladmin.exe</command>
<command>mysqladmin -h 127.0.0.1 password</command>
<command>mysqladmin.exe -h 127.0.0.1 password</command>
```

For `<option>` elements, the `role` attribute indicates the program to which the option belongs. The element content provides the option name (possibly after stripping leading dashes and a following `=value`. A `condition` attribute can be provided to specify the option name explicitly, which can be useful for short-form options.

For example, the following elements all generate links to the description for the `--host` option of the `mysqld` program, which has an ID of `option_mysqld_host`:

```
<option role="mysqld">--host</option>
<option role="mysqld">--host=host_name</option>
<option role="mysqld" condition="host">-h</option>
<option role="mysqld" condition="host">-h host_name</option>
```

System variables can be passed as command-line options as well, so `<option>` can be used for those. In such cases, the `role` attribute is `sysvar` and the option content provides the variable name, possible with dashes mapped to underscores. The following elements all generate links to the description for the `max_allowed_packet` system variable, which has an ID of `sysvar_max_allowed_packet`:

```
<option role="sysvar">--max_allowed_packet=1G</option>
<option role="sysvar">--max-allowed-packet=1G</option>
<option role="sysvar">max_allowed_packet=1G</option>
<option role="sysvar">max-allowed-packet=1G</option>
```

Chapter 5. Conventions for Writing and Editing MySQL Documentation (“Styleguide”)

5.1. Principles and Miscellaneous

This section covers general guidelines.

- *Acronyms* should generally be spelled out the first time they appear in a book, as in: “Computer Development Environment (CDE).” After the acronym has been defined, you should generally use the acronym only (not the whole term, unless it makes more sense contextually to use the whole term). Usually, acronyms are defined only one per book. But if you prefer, you can also define them the first time they appear in each chapter.

- *Change notes*: In change notes, references to bug numbers should be written after the sentence describing the bugfix, like this: `This sentence describes the fix. (Bug #nnnn)`, where *`nnnn`* is the number of the bug assigned in the MySQL bug tracking system. The process that converts the Manual to HTML finds these and transforms them to hyperlinks that take the reader to the proper report in the bug system at http://bugs.mysql.com.

  If there are multiple bugs, write them as `(Bug #mmmm, Bug #nnnn)` rather than as `(Bug #mmmm, #nnnn)`. (The word “Bug” preceding the number is important.)

- *Code lines* should run no longer than 72 characters. This is particularly important for printed books. Code lines should be indented using spaces, not tabs.

- *Commas*: To write a list of items, add commas after all items preceding the last one. Example: `Features, products, and services`, not `Features, products and services`

- *Dates and numbers*:

  - Spell out numbers under 10, unless the same object appears in a sentence with an object 10 or over (five apples; 5 apples and 100 oranges).

  - Use numerals for versions (`Version 5`), if it is an actual value (set `long_query_time` to `2`), for bits (`32-bit`), and for dates (`the 1980s`).

  - Phone numbers should appear in this format, using either spaces or dashes: *`+country code city code telephone number`*

    Example: `+49 30 82702940` or `+49-30-8270294-0`

- *Ellipsis*: The character entity for an ellipsis (…) causes problems when transforming a DocBook document to PDF output. Use the literal `...` instead.

- *Footnotes*: Do not use footnotes at all. Footnotes work well for print or PDF, but we use our documentation in many other formats such as HTML, where footnotes require a lot of navigation on the reader's part. It can take some thought to determine how best to incorporate into a passage the text that you might otherwise put in a footnote. Make the effort. Sometimes in doing so you will become aware that you were using a footnote as a means of being lazy, and that having to rethink a passage reveals a much better way of organizing what you are trying to say.

- *Language*: US English, not UK English. (optimize, not optimise; center, not centre; color, not colour; transportation, not transport; etc.)

- *Lists*: We use two types of lists only: numbered lists and bulleted lists. For the former, use the `<orderedlist>`, for the latter, use the `<itemizedlist>` element.

  No period after list items unless one item forms a complete sentence (then use periods for all items within that list, even fragments).

- *Quotes*: For which kind of quotes to use under which circumstances, see Section 4.7, “Quotes”.

- *Spaces, not tabs*: Do not use tab characters at all, use spaces instead.

- *URL*s are considered breakable at punctuation characters. This is how Pearson handles it in printed materials.

  In examples, only use URLs that are either owned by MySQL, or (preferably) URLs that are recommended for documentation in general:

  - `example.com`

- `example.net`
- `example.org`

Example:

```
You have reached this web page by typing “example.com”,
“example.net”, or “example.org” into the
address bar of your web browser.
```

These domain names are reserved for use in documentation and are not available for registration. See RFC 2606, Section 3.

- We recommend using binary distributions, rather than source distributions.

  If a section contains instructions for both types of distributions, put those for binary distributions first. If you find a section that still has source instructions first, see if it can be rewritten to put binary instructions first.

- We now have more Windows users than Unix users. So, if a section contains instructions for both platforms, put those for Windows first, then Unix. If you find a section that still has the Unix instructions first, see if it can be rewritten to put Windows first.

  Example of allowable exception: The `--socket` option is used for specifying a Unix domain socket file name or a Windows named pipe name. The name of the option itself obviously has its roots in the Unix usage, so it can make sense to mention its Unix meaning first and then the Windows meaning. Such exceptions tend to be rare.

5.2. Guidelines for Numbering

Do not number elements by hand. The numbering for chapters, appendices, and sections is done automatically when output formats like HTML or PDF are created. This is also true for figures and graphics. If you need numbered lists, you should use the `<orderedlist>` element.

5.3. Guidelines for Capitalization of Terms

This subsection covers the capitalization of words and terms for certain elements, like headings.

- *Figure captions* are initial-capped on first word only, with the exception of proper nouns. There is no period after figure captions. Example: `Switching to configure-service mode`

  Figure captions may end with a period. This makes sense if most figure captions are full sentences. Currently, we don't use periods, but should we start using them we should use periods even for captions that are not full sentences (consistency overrules grammatical correctness).

- *SQL function names* should use all capitals, followed by `()`; for example, `DATABASE()`, and the `<literal>` element. The `<literal>` element must not be used when an SQL function is inside a `<programlisting>` element.

- *SQL keywords and statements* should use all capitals. In paragraph text, enclose the terms within `<literal>`. In program listings (`<programlisting>...</programlisting>`), do not use `<literal>`. `<replaceable>` is allowable within `<literal>` and `<programlisting>`. `<userinput>` and `<replaceable>` are allowable within `<programlisting>`.

- *Titles* of chapters and sections should use initial capitals following these rules: Cap the first letter of each word, with the exception of articles, conjunctions, prepositions of four letters or less, and program names or technical words that are always lowercase. Hyphenated words in titles or captions should both be capped if the second word is a main word, but only the first should be capped if the second word isn't too important (it is a bit of a judgment call). Example: `Big-Endian, Built-in` *Prepositions* with four letters or fewer should be lowercase: `from, with, Within, Between`. Also, the word `to` should be lowercase.

- *Product names* should be capitalized. Sometimes it's not easy to determine whether a term is a product name or not. Here's an example:

  “MySQL Server” is a product name, while “MySQL server” is for phrases like “the MySQL server”. Use of the product name is a concept that gets a lot of love from marketing or something. However, use of it everywhere makes text sound still and stilted. We should use “MySQL Server” when talking about the product in general (for example, discussing features) but “MySQL server” (or just “the server”) when talking about specific uses (for example, “the MySQL server performs character set conversion when necessary”).

5.4. Guidelines for Using Denominations

This section gives advice how names and other denominations should be used.

5.4.1. Denominations in General

- *CLI*: Do not use CLI as an abbreviation for Command Line Interface. It is a standard abbreviation for Call-Level Interface.
- *Connection*: It is true that connections and sessions are associated, but the word “connection” should not be used when “session” is more accurate. For example: not “a transaction during this connection” but “a transaction during this session.” Indeed, Paul has noted a tendency to use connection, session, client, and thread all to mean roughly the same thing. And with the advent of thread-pooling for handing client connections in MySQL 6.0, it no longer necessarily true that a given connection will be serviced by a given thread. This means the rough synonymy that historically has held true is no longer valid.
- *Cursor*: It is difficult to pin this one down, but it seems that some people might be saying “create a cursor” when they really mean “allocate a statement (stmt).” The word cursor must be reserved for the named object which is the subject of open/fetch/close.
- *Database*: This appears even in the MySQL Reference Manual: “MySQL, the most popular Open Source database ...” Correct is: Database Management System, or DBMS.
- *Derived table*: It is true that, in the following statement, the bit in the `FROM` clause is a “derived table.”

  ```
  SELECT * FROM (SELECT * FROM t) AS X;
  ```

  However, so are many other things. For example, a view is a [named] derived table. Therefore, if you really want to specify that you are using a query in the `FROM` clause, correct is: Query in the `FROM` clause.
- *Field*: People are using “field” when they clearly must mean “column”, as in “a field of a table.” Correct is: Column. This will take a while, since the misuse of “field” occurs in MySQL functions and statements.

  Paul breaks this rule in the following context: Some `SHOW` statements display information about table columns. To discuss this output, it's necessary to refer to table columns and columns in the result set of the `SHOW` statement, which can easily become confusing. It can help to disambiguate this by referring to the fields of the result set and the columns of the table.
- *Key*: The word KEY may refer to a particular value in an index (an “index key”) or to a set of columns which are useful for lookup (a “primary key” or “foreign key”). However, influenced perhaps by MySQL's “KEY(column)” clause in CREATE TABLE, some people are saying “create a key on column1.” They should say “create an index on column1.”
- *MySQL*: Write MySQL without adornment, that is, not as <emphasis>MySQL</emphasis> or <literal>MySQL</literal>.
- *MySQL Reference Manual*: The name of the reference Manual is MySQL Reference Manual. Subsequent references in close proximity can be just Reference Manual or *Manual*, as long as it is clear what the text is referring to.
- *Query*: A query is a question, look in your dictionary. `SELECT` may be called a query, but it is not appropriate to call `UPDATE` a “query.” Also incorrect is: command. Correct is: statement.
- *Relation*: A relation is a set. That is why a relational database is called relational. It isn't because of some “relationship between tables.”
- *ROWID*: The word “rowid” has been used once, by one person, as an apparent synonym for Primary Key. Wrong. We should use rowid only for a physical location.
- *Subselect*: This was okay at one time, but we want to standardize. Correct is: Subquery.
- Write Unix, not UNIX.
- *Unix and Linux*: Unless necessary for some specific reason, we treat Unix and Linux as synonymous, with both being covered by the term “Unix”.
- URLs ending in a domain name or directory should not have a slash (`/`) at the end.

5.4.2. Denominations With Regards to Collations

This section contains a list of terms and definitions used in the context of collations.

- *allkeys.txt*: An example of a series of collating-table entries as defined by UCA (Unicode Collation Algorithm).

Actually UCA says `allkeys.txt` is a “collation element table”, that is, it is the part of a collating table which shows collating elements.

- *COLLATING ELEMENT*: The unit which linguistically-aware users perceive as the minimal building block in string comparisons.

  Usually there is a one-to-one relation between characters and collating elements, for example in English there is a character “A” and a collating element for “A”. More rarely there is a many-to-one relation, for example in traditional Spanish the two-character combination “LL” is a single collating element.

  Usually there is a one-to-many relation between collating elements and weights (because there are multiple levels); however, for an ignorable character, one collating element has zero weights.

- *COLLATING TABLE*: A table which describes all the rules for a collation, including Posix-like “Locale” declarations and a list of collating elements.

  Here are entries for collating elements from two sources, ISO 14651 and `allkeys.txt`:

  ```
  [From ISO 14651]
  <U0024> <S2C4>;<BASE>;<MIN>;<U0024> % DOLLAR SIGN
  <UFF04> <S2C4>;<BASE>;<WIDE>;<UFF04> % FULLWIDTH DOLLAR SIGN
  <UFE69> <S2C4>;<BASE>;<SMALL>;<UFE69> % SMALL DOLLAR SIGN
  ```

  ```
  [From allkeys.txt 4.0]
  0024  ; [.0E0F.0020.0002.0024] # DOLLAR SIGN
  FF04  ; [.0E0F.0020.0003.FF04] # FULLWIDTH DOLLAR SIGN; QQK
  FE69  ; [.0E0F.0020.000F.FE69] # SMALL DOLLAR SIGN; QQK
  ```

  Clearly these are the same thing, but ISO 14651 uses names (e.g. “BASE”) where `allkeys.txt` uses numbers (e.g. `0020`). So ISO 14651 had to define earlier in its table `BASE = 0020; MIN = 0002; WIDE = 0003; SMALL= 000F` etc.

- *COLLATION ELEMENT*: Do not use. Use *collating element*.

- *COLLATION TABLE*: Do not use. Use *collating table*.

- *COLLATING TABLE ENTRY*: A line in a collating table, representing one fact.

  Each “line” in `allkeys.txt` (which is a subset of a collating table) is an entry for one collating element.

- *CONTRACTION*: A mapping from *`N`* characters to less-than-*`N`* collation elements.

  Contraction is rare, for example the character “C” has one collation element “C”. But take an example from traditional Spanish: “LL” is a single collation element between “L” and “M”. Contraction also occurs when there has been decomposition. For example here are two collating element entries (from `allkeys.txt` 5.0.0):

  ```
  0622  ; [.15E2.0020.0002.0622] # ARABIC LETTER ALEF WITH MADDA ABOVE
  0627 0653 ; [.15E2.0020.0002.0622] # ARABIC LETTER ALEF WITH MADDA ABOVE
  ```

  Notice that there is one collation element labelled `0627 0653`, which clearly is the result of mapping from two characters `U+0627 ARABIC LETTER ALEF` and `U+0653 ARABIC MADDAH ABOVE`, with the same weights as the composed character `U+0622 ARABIC LETTER ALEF WITH MADDA ABOVE`.

- *EXPANSION*: A one-to-many mapping from collating element to weighting levels.

  For example, German Sharp S may be treated as “ss”, so the `allkeys.txt` entry for collating element `00DF` (Sharp S) is:

  ```
  00DF  ; [.11AF.0020.0004.00DF][.0000.0199.0004.00DF][.11AF.0020.001F.00DF] # LATIN SMALL LETTER SHARP S;
  ```

  The entry for “s” alone is:

  ```
  0073  ; [.11AF.0020.0002.0073] # LATIN SMALL LETTER S
  ```

  Since `0000` means ignorable, two-level weight strings are:

  ```
  11AF 11AF 0020 0199 0020      /* for SHARP S */
  11AF 11AF 0020 0020           /* for 'ss' */
  ```

- *IGNORABLE CHARACTER*: A character which has one collating element which has no significance for comparison. One ignorable character has one collating element, but zero weights at all levels. For example (from allkeys.txt): 0591 ; [.0000.0000.0000.0591] # HEBREW ACCENT ETNAHTA This is ignorable for three levels but not four levels. Therefore it is an “ignorable character” when you produce a weight string for one, two, or three levels. “Ignorable at level 1” means the level-1 weight is ignorable, as represented by 0000 in allkeys.txt. “Fully ignorable” means ignorable for all levels.

- *ISO 14651*: The ISO/IEC 14651 “International String Ordering” standard.

  Draft documents:

  - http://std.dkuug.dk/jtc1/sc22/open/n2933.pdf
  - http://ra.dkuug.dk/jtc1/sc22/wg20/docs/n731-fdis14651.pdf
  - http://www.dulug.dk/JTC1/SC2/WG2/docs/n2691.pdf

- *LEVEL*: A prioritization order for weights.

  Each level has a name “level + number”, for example “level 1”, “level 2”, “level 3”, “level 4”. (Do not use, or rarely use, equivalent terms “primary”, “secondary”, “tertiary”, “quaternary”.) Typically level 1 is the character-differs level for `WHERE` clauses, levels 2 and following are case-differs or accent-differs something-minor-differs levels which might be useful for `ORDER BY` clauses. For example, from `allkeys.txt` 5.0.0:

  ```
  0061  ; [.0FD0.0020.0002.0061] # LATIN SMALL LETTER A
  24D0  ; [.0FD0.0020.0006.24D0] # CIRCLED LATIN SMALL LETTER A;
  0041  ; [.0FD0.0020.0008.0041] # LATIN CAPITAL LETTER A
  ```

  There are four levels here. Level 1 is always `0FD0` for “A”. Level 2 is always `0020`. Level 3 is `0002` for `SMALL`, `0006` for `CIRCLED`, `0008` for `CAPITAL`. Level 4 is the same as the Unicode code point value. Do not confuse “weight level” with “weighting level”.

- *ORDERING KEY*: Do not use. Use “weight string”.

- *SORTKEY*: Do not use. Use “weight string”.

- *SUBKEY*: A sequence of weights for a single level.

- *UCA*: Unicode Collation Algorithm as described in Unicode Technical Standard #10, http://www.unicode.org/reports/tr10.

- *WEIGHT*: A positive numeric value used for comparisons.

  Weights come from collating tables and go to weight strings. Often weight appears as a 4-digit number in collating tables. For example (from `allkeys.txt`):

  ```
  0062  ; [.0FE6.0020.0002.0062] # LATIN SMALL LETTER B
  ```

  This is the entry for collating element `0062`, and there are 4 weights: `0FE6` and `0020` and `0002` and `0062`.

- *WEIGHT STRING*: A binary string, sometimes called a “sortkey” or an “ordering key”, produced by taking a series of weights from a collating table for a certain number of levels, ordering them by position and level, and outputting.

  For example: starting with a character string `ABC`, and knowing that the number of levels is 2, look up the collating elements for `A` and `B` and `C` in `allkeys.txt` 5.0.0:

  ```
  0041  ; [.0FD0.0020.0008.0041] # LATIN CAPITAL LETTER A
  0042  ; [.0FE6.0020.0008.0042] # LATIN CAPITAL LETTER B
  0043  ; [.0FFE.0020.0008.0043] # LATIN CAPITAL LETTER C
  ```

  Result: `0FD0.0FE6.0FFE.0020.0020.0020`. MySQL's `weight_string()` function produces a weight string.

- *WEIGHTING ELEMENT*: A sequence of weights, in ascending order by level.

  For example, from `allkeys.txt` 5.0.0:

  ```
  00DF  ; [.11AF.0020.0004.00DF][.0000.0199.0004.00DF][.11AF.0020.001F.00DF] # LATIN SMALL LETTER SHARP S
  ```

  There are three weighting elements in this example, each is surrounded by square brackets:

  ```
  [.11AF.0020.0004.00DF]
  [.0000.0199.0004.00DF]
  [.11AF.0020.001F.00DF]
  ```

  Often one collating element has only one weighting element (which has many weights), but `SHARP S` is an example of expansion.

- *ZERO WEIGHTS*: The meaning is “an empty sequence of weights” (the ISO 14651 definition), not “weights with value 0000” (the UCA definition).

  For example (from `allkeys.txt`):

  ```
  0591  ; [.0000.0000.0000.0591] # HEBREW ACCENT ETNAHTA
  ```

There are three “empty sequences of weights” here, all of which look like `0000`, which we interpret as code for “empty”.

5.5. Guidelines on Grammar and Style

This section covers a number of common grammar and style issues.

Do not use contructs like this:

```
How to use SELECTs.
```

This is not just bad English, but makes it unnecessarily hard for translators. You could rewrite it like this:

```
How to use SELECT statements.
```

Another problem with constructs such as “SELECTs” is that it involves an ugly within-word font change. Constructs such as non-`NULL` do not have this problem because there is an intervening hyphen.

Under no circumstance allow instance of things like “SELECT's” or “SELECT:s” to go unchanged whenever you encounter them. Writing “SELECT's” is incorrect because it is not a possessive. Writing “SELECT:s” just looks weird. You are most likely to see these things in the Change Notes sections of the Reference Manual, because developers tend to be sloppier in their language there for some reason. As documenters, we have no such luxury. It is required that we expend the thought necessary to figure out how to rewrite the text properly.

Do not use *emoticons* such as :-), ;-), etc.

Do not end sentences with an exclamation mark (!). There can be exceptions, but they are rare. On occasion you might come across a passage consisting of three or four sentences in a row that end with exclamation marks. Invariably, you can change most of these to periods with no damage to the meaning, and the passage no longer will look like it's hyperventilating.

Avoid *expletives* such as “simple”, “easy”, or “just”. Leave it to the reader or user to decide, for example, whether or not something is simple. Here's an example how to not write a sentence:

```
Using this feature is easy; simply click the red button.
```

Make sure you do not confuse `its` with `it's`. This can best be accomplished by following the next rule.

Apostrophes: Avoid using apostrophes. Print publishers don't (uh, do not) like them. Write `it is`, rather than `it's`. Note that this rule may change due to an editor's preference.

Abbreviations: Avoid using abbreviations. Print publishers do not like these, either. Write “and so forth”, rather than “etc.”. Use “that is” and “for example”, rather than “i.e.” or “e.g.” If you absolutely must use i.e. or e.g., do *not* use i.e, ie, e.g, or eg.

The word *data* is problematic. It is commonly used both in plural and in singular form. The Manual uses it as plural, which means you use `data are` rather than `data is`. It is unfortunate that no matter which form we use, it will look incorrect to some people. But we can at least be internally consistent. If you can substitute another word such as “information,” that makes the problem go away entirely. “Metadata” is especially problematic; there is a shortage of good synonyms. Sometimes you can rewrite a sentence to avoid the singluar/plural issue entirely. “The data are stored in the table” and “The data is stored in the table” can be rewritten as “The table stores the data.” (This also has the benefit of changing the sentence from passive to active.)

When reproducing program output, reproduce it exactly, even if it contains typos. Do not “fix” it. (If the output is produced by a MySQL program, then fix the source for the program to the output correctly without the typo, then update the Manual to match.)

To refer to ASCII codes, use `ASCII` *`n`*, not `ASCII(`*`n`*`)`, unless you are referring to the `ASCII()` function, in which case you use `<function>ASCII()</function>` . For example, write `ASCII 13`.

The rules for hyphenated words are:

- Use a hyphen for a combination adjective preceding its noun as in “case-sensitive collation”.
- Do not use a hyphen for a combination adjective following a noun as in “this collation is case sensitive”.
- Do not use a hyphen for a combination noun as in “case sensitivity”.
- There is no hyphen after words ending with “ly”, for example “statically linked”, not “statically-linked”.
- *Do not use hyphens (or spaces) for prefixes* such as “sub”, “pre”, or “re”, unless this is necessary to disambiguate words. For example, it's “subroutine”, “preamble”, or “reorder”, rather than “sub routine”, “pre-amble”, or “re-order”.

  To disambiguate words, though, some people use hyphens, for example to disambiguate “recreation” (leisure) from

“re-creation” (creating something anew). Others use hyphens when the stem of a word begins with a vowel. For example, they would use “re-introduced”, rather than “reintroduced”.

- *The usage of hyphens may depend on how naturalized a word is* in English, and (this is related) to what extent it's regarded a proper noun. Here are some examples:
  - Precambrian
  - pre-Columbian

Avoid terms that imply spatial direction when you really mean preceding or following. Use “mentioned previously” instead of “above” when making relative references in your text. Something that is “above” in a single-page presentation format such as a Web page will not necessarily be above in a multiple-page format such as a printed book. Similarly, use “later in this section” instead of “below.”

For a long dash, use `&mdash;`.

Comments: If a `(comment) in parentheses` is at the end of a sentence, start the comment with lowercase and put the period (`.`) after the closing parens (`)`), such as `(like this).`. If a comment is separate, start with uppercase and put the period inside the closing parens, `(Like this.)`.

After a semicolon, don't use uppercase. It is *not* the start of a new sentence.

After a colon, capitalize the next word if the following text is a complete sentence in itself. Do not capitalize if it is not a complete sentence.

5.6. Guidelines on Spelling

This subsection covers a number of common spelling issues.

- 32-bit, not 32 bit or 32 bits. “32 bit result” should be “32-bit result”. However, “the result has 32 bits” has no hyphen.
- 4KB, 4MB, 2GB, 4TB -- not 4 kilobytes, not 4 Kilobytes, not 4 KB; the same is true for MB (not megabytes), GB (not gigabytes), TB (not terabytes).
- Megahertz and gigahertz are MHz, GHz, not Mhz, Ghz.
- B means bytes, b means bits; use appropriately. The usual place for lowercase b is in network rate indicators, such as 100Mb/s to mean 100 megabits per second.
- administer, not administrate
- a lot requires two words; alot is wrong
- application-related, not application related. In titles, use Application-Related, not Application-related.
- authenticate, not authentify
- authentication, not authentification
- authorize, not authorise
- automatically, not automaticly
- backup is a noun or adjective (as in `a backup file`), back up is a verb (as in `to back up a database`)
- backward, not backwards
- backward-compatible, not backward compatible or backwards compatible
- behavior, not behaviour
- byte-swapping, not byte swapping
- cannot, not can not
- client-side, not client side

- client/server, not client-server
- color, not colour
- command-line is an adjective, command line is a noun. Example: “You should use a command-line tool if you prefer entering commands on the command line.”
- compliance, not compliancy
- core dump is a noun or a verb (as in `a core dump file` or `a program core dumps when it fails`). In the latter case, however, it is better to say `a program dumps core when it fails`
- CPU time, not CPU (in phrases like `uses CPU time`; unless you are referring the processor itself.)
- CPU, not cpu
- data file, data set, data type, not datafile, dataset, datatype
- deprecate, not depreciate (depreciate is a word, but not the one you want when you are writing about features that we discourage people from using, and which may be removed in later releases)
- deprecated, not depricated
- different from *`something`*, not different than *`something`*
- dynamically, not dynamicly
- email, not e-mail
- file-size, not file size
- file name, file system, not filename, filesystem
- floating-point, not floating point
- following some, not something [shown] below
- forward, not forwards
- full-text, not fulltext (unless you are referring to a `FULLTEXT` index)
- hand-held, not hand held
- heavy-duty, not heavy duty
- heavy-load production systems (used as an adjective), but used under heavy load (used on its own).
- high-priority *`something`* (when used as an adjective), not high priority
- host name, not hostname
- indexes, not indices; exception: when referring to array elements, use “indices”
- installation, not install (as a noun)
- lettercase, not letter case
- long-awaited, not long awaited
- long-time *`something`* (when used as an adjective), not long time
- lowercase, not lower case
- low-volume *`something`* (when used as an adjective)
- master/slave, not master-slave
- memory-based, not memory based
- multiple CPU, not multiple-CPU
- multi-byte, not multi byte

- multi-thread(ed), not multithread(ed)
- multi-user, not multi user
- natural-language, not natural language
- Net (capitalized), if referring to the Internet in that way
- NetWare, not Netware
- Note:, not NOTE:
- “Note that...” When a sentence begins this way, consider whether the "Note that" is useless noise that can be deleted or whether it is a necessary part of the sentence. If the reader really needs to be brought to a full stop, use a `<note>` element.
- Object-oriented. It is hyphenated.
- okay, not ok or Ok or OK. Exceptions: When describing instructions for a GUI with buttons that say OK, then use OK. That is, use the label that the GUI uses. When showing the output from a program, show the output exactly; do not change ok to okay, and so forth.
- online, not on-line
- onward, not onwards
- Open Source, not open source
- optimize, not optimise
- otherwise is followed by a comma at the start of a sentence
- percent, not per cent
- platform-dependent, not platform dependent
- PostScript, not Postscript
- power-start, not power start
- press Enter, or press the Enter key, not hit Return or hit Enter
- publicly, not publically
- re-issue(ing), not reissue(ing)
- replication-safe, not replication safe
- rewriting, not re-writing
- rollback is a noun or adjective (as in `a rollback operation`), `roll back` is a verb (as in `roll back a transaction`)
- runtime, not run time
- schemas, not schemata
- server-side, not server side
- shown here, not shown below
- single-CPU, not single CPU
- statically, not staticly
- “sub” is a prefix, not a word, so it's subfunction, rather than sub function, or subroutine, rather than sub routine (or sub-routine)
- third-party, not third party
- “this only”, not “only this”
- thread-safe, not thread safe

- toward, not towards
- transaction-safe, not transaction safe
- turnkey, not turn-key
- uncorrelated subquery, not non-correlated subquery
- unstable, not instable
- uppercase, not upper case
- user-defined, not user defined
- user name, not username
- web page, web site, not webpage, website. Note: Pearson prefers Web page, Web site.
- whether, not whether or not
- wildcard, not wild card or wild-card

More words and terms can be found on the O'Reilly Default Stylesheet and Word List (on the bottom of that page).

5.7. Guidelines for GUI Documentation

This subsection covers how to use terms for GUI documentation.

- To indicate that a checkbox is available for the user, use *active*.
- To indicate that a checkbox is unavailable for the user, that is it's greyed out, use *inactive*.
- For checkboxes, terms such as *enable/disable*, *on/off*, or *selected/de-selected* can be used to indicate the value of the control, as determined by what the user does with it.
- When referring to menu actions, use one of the following forms:
  - Menu, menu item (, submenu item, ...), ...; for example: `Select <guimenu>Edit</guimenu>, <guimenu>Options</guimenu>, <guimenu>Connections</guimenu>.`
  - Select ... from menu (item); for example: `Select <guimenu>Connections</guimenu> after choosing the <guimenu>Options</guimenu> item from the <guimenu>Edit</guimenu> menu.` (Obviously, the first form is suited better for longer actions, whereas the second form might be good for shorter actions.)

  Do not use forms like `Select Edit > Options > Connections`, or `Edit => Options => Connections`.