```
namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"
namespace ctrl = "http://nwalsh.com/xmlns/schema-control/"
default namespace db = "http://docbook.org/ns/docbook"
namespace html = "http://www.w3.org/1999/xhtml"
namespace mml = "http://www.w3.org/1998/Math/MathML"
namespace rng = "http://relaxng.org/ns/structure/1.0"
namespace s = "http://purl.oclc.org/dsdl/schematron"
namespace svg = "http://www.w3.org/2000/svg"
namespace xlink = "http://www.w3.org/1999/xlink"

# This file is part of DocBook Assembly V5.2b08
#
~~#  DocBook Version 5.1~~
~~#  OASIS Standard ~~
~~#  22 November 2016~~
# Copyright ~~(c) OASIS Open 2016. All Rights Reserved.~~2008-2015 HaL Computer Systems,
Inc.,
~~#  Source: http://docs.oasis-open.org/docbook/docbook/v5.1/os/schemas/rng/~~
~~#  Link to latest version of specification: http://docs.oasis-~~
~~open.org/docbook/docbook/v5.1/docbook-v5.1.html~~
# O'Reilly & Associates, Inc., ArborText, Inc., Fujitsu Software
# Corporation, Norman Walsh, Sun Microsystems, Inc., and the
# Organization for the Advancement of Structured Information
# Standards (OASIS).
#
~~# This file is part of DocBook Assembly V5.1-OS~~
# Permission to use, copy, modify and distribute the DocBook schema
# and its accompanying documentation for any purpose and without fee
# is hereby granted in perpetuity, provided that the above copyright
# notice and this paragraph appear in all copies. The copyright
# holders make no representation about the suitability of the schema
# for any purpose. It is provided "as is" without expressed or implied
# warranty.
#
# If you modify the DocBook schema in any way, label your schema as a
# variant of DocBook. See the reference documentation
# (http://docbook.org/tdg5/en/html/ch05.html#s-notdocbook)
# for more information.
#
# Please direct all questions, bug reports, or suggestions for changes
# to the docbook-comment@lists.oasis-open.org mailing list. For more
# information, see http://www.oasis-open.org/docbook/.
#
# ======================================================================

s:ns [ prefix = "rng" uri = "http://relaxng.org/ns/structure/1.0" ]
s:ns [ prefix = "s" uri = "http://purl.oclc.org/dsdl/schematron" ]
s:ns [ prefix = "db" uri = "http://docbook.org/ns/docbook" ]
s:ns [
  prefix = "a"
  uri = "http://relaxng.org/ns/compatibility/annotations/1.0"
]
s:ns [ prefix = "ctrl" uri = "http://nwalsh.com/xmlns/schema-control/" ]
s:ns [ prefix = "html" uri = "http://www.w3.org/1999/xhtml" ]
s:ns [ prefix = "xlink" uri = "http://www.w3.org/1999/xlink" ]
s:ns [ prefix = "mml" uri = "http://www.w3.org/1998/Math/MathML" ]
s:ns [ prefix = "svg" uri = "http://www.w3.org/2000/svg" ]
div {
  db._any.attribute =

    ## Any attribute, including any attribute in any namespace.
    attribute * { text }
  db._any =
    ## Any element from almost any namespace
    element * - (db:* | html:*) {
      (db._any.attribute | text | db._any)*
    }
}
db.arch.attribute =

  ## Designates the computer or chip architecture to which the element applies
  attribute arch { text }
db.audience.attribute =

  ## Designates the intended audience to which the element applies, for example, system
administrators, programmers, or new users.
  attribute audience { text }
db.condition.attribute =

  ## provides a standard place for application-specific effectivity
  attribute condition { text }
db.conformance.attribute =

  ## Indicates standards conformance characteristics of the element
  attribute conformance { text }
db.os.attribute =

  ## Indicates the operating system to which the element is applicable
  attribute os { text }
db.revision.attribute =

  ## Indicates the editorial revision to which the element belongs
  attribute revision { text }
db.security.attribute =

  ## Indicates something about the security level associated with the element to which
it applies
  attribute security { text }
db.userlevel.attribute =

  ## Indicates the level of user experience for which the element applies
  attribute userlevel { text }
db.vendor.attribute =

  ## Indicates the computer vendor to which the element applies.
  attribute vendor { text }
db.wordsize.attribute =

  ## Indicates the word size (width in bits) of the computer architecture to which the
element applies
  attribute wordsize { text }
db.outputformat.attribute =

  ## Indicates the output format (for example, print or epub) to which the element
applies
  attribute outputformat { text }
db.effectivity.attributes =
  db.arch.attribute?
  & db.audience.attribute?
  & db.condition.attribute?
  & db.conformance.attribute?
  & db.os.attribute?
  & db.revision.attribute?
  & db.security.attribute?
  & db.userlevel.attribute?
  & db.vendor.attribute?
  & db.wordsize.attribute?
  & db.outputformat.attribute?
db.endterm.attribute =

  ## Points to the element whose content is to be used as the text of the link
  attribute endterm { xsd:IDREF }
db.linkend.attribute =

  ## Points to an internal link target by identifying the value of its xml:id attribute
  attribute linkend { xsd:IDREF }
db.linkends.attribute =

  ## Points to one or more internal link targets by identifying the value of their
xml:id attributes
  attribute linkends { xsd:IDREFS }
db.xlink.href.attribute =

  ## Identifies a link target with a URI
  attribute xlink:href { xsd:anyURI }
db.xlink.simple.type.attribute =

  ## Identifies the XLink link type
  attribute xlink:type {

    ## An XLink simple link type
    "simple"
  }
db.xlink.role.attribute =

  ## Identifies the XLink role of the link
  attribute xlink:role { xsd:anyURI }
db.xlink.arcrole.attribute =

  ## Identifies the XLink arcrole of the link
  attribute xlink:arcrole { xsd:anyURI }
db.xlink.title.attribute =

  ## Identifies the XLink title of the link
  attribute xlink:title { text }
db.xlink.show.enumeration =

  ## An application traversing to the ending resource should load it in a new window,
frame, pane, or other relevant presentation context.
  "new"
  |
    ## An application traversing to the ending resource should load the resource in the
same window, frame, pane, or other relevant presentation context in which the starting
resource was loaded.
    "replace"
  |
    ## An application traversing to the ending resource should load its presentation in
place of the presentation of the starting resource.
    "embed"
  |
    ## The behavior of an application traversing to the ending resource is
unconstrained by XLink. The application should look for other markup present in the
link to determine the appropriate behavior.
    "other"
  |
    ## The behavior of an application traversing to the ending resource is
unconstrained by this specification. No other markup is present to help the application
determine the appropriate behavior.
    "none"
db.xlink.show.attribute =

  ## Identifies the XLink show behavior of the link
  attribute xlink:show { db.xlink.show.enumeration }
db.xlink.actuate.enumeration =

  ## An application should traverse to the ending resource immediately on loading the
starting resource.
  "onLoad"
  |
    ## An application should traverse from the starting resource to the ending resource
only on a post-loading event triggered for the purpose of traversal.
    "onRequest"
  |
    ## The behavior of an application traversing to the ending resource is
unconstrained by this specification. The application should look for other markup
present in the link to determine the appropriate behavior.
    "other"
  |
    ## The behavior of an application traversing to the ending resource is
unconstrained by this specification. No other markup is present to help the application
determine the appropriate behavior.
    "none"
db.xlink.actuate.attribute =

  ## Identifies the XLink actuate behavior of the link
  attribute xlink:actuate { db.xlink.actuate.enumeration }
db.xlink.simple.link.attributes =
  db.xlink.simple.type.attribute?
  & db.xlink.href.attribute?
  & db.xlink.role.attribute?
  & db.xlink.arcrole.attribute?
  & db.xlink.title.attribute?
  & db.xlink.show.attribute?
  & db.xlink.actuate.attribute?
db.xlink.attributes =
  db.xlink.simple.link.attributes
  | (db.xlink.extended.link.attributes
     | db.xlink.locator.link.attributes
     | db.xlink.arc.link.attributes
     | db.xlink.resource.link.attributes
     | db.xlink.title.link.attributes)
db.xml.id.attribute =

  ## Identifies the unique ID value of the element
  attribute xml:id { xsd:ID }
db.version.attribute =

  ## Specifies the DocBook version of the element and its descendants
  attribute version { text }
db.xml.lang.attribute =

  ## Specifies the natural language of the element and its descendants
  attribute xml:lang { text }
db.xml.base.attribute =

  ## Specifies the base URI of the element and its descendants
  attribute xml:base { xsd:anyURI }
db.remap.attribute =

  ## Provides the name or similar semantic identifier assigned to the content in some
previous markup scheme
  attribute remap { text }
db.xreflabel.attribute =

  ## Provides the text that is to be generated for a cross reference to the element
  attribute xreflabel { text }
db.xrefstyle.attribute =
  ## Specifies a keyword or keywords identifying additional style information
  attribute xrefstyle { text }
db.revisionflag.enumeration =

  ## The element has been changed.
  "changed"
  |
    ## The element is new (has been added to the document).
    "added"
  |
    ## The element has been deleted.
    "deleted"
  |
    ## Explicitly turns off revision markup for this element.
    "off"
db.revisionflag.attribute =

  ## Identifies the revision status of the element
  attribute revisionflag { db.revisionflag.enumeration }
db.dir.enumeration =

  ## Left-to-right text
  "ltr"
  |
    ## Right-to-left text
    "rtl"
  |
    ## Left-to-right override
    "lro"
  |
    ## Right-to-left override
    "rlo"
db.dir.attribute =

  ## Identifies the direction of text in an element
  attribute dir { db.dir.enumeration }
db.rdfalite.vocab =

  ## The RDFa Lite vocab
  attribute vocab { xsd:anyURI }
db.rdfalite.typeof =

  ## The RDFa Lite typeof
  attribute typeof { text }
db.rdfalite.property =

  ## The RDFa Lite property
  attribute property { text }
db.rdfalite.resource =

  ## The RDFa Lite resource
  attribute resource { text }
db.rdfalite.prefix =

  ## The RDFa Lite prefix
  attribute prefix { text }
db.rdfalite.attributes =
  db.rdfalite.vocab?
  & db.rdfalite.typeof?
  & db.rdfalite.property?
  & db.rdfalite.resource?
  & db.rdfalite.prefix?
db.common.base.attributes =
  db.version.attribute?
  & db.xml.lang.attribute?
  & db.xml.base.attribute?
  & db.remap.attribute?
  & db.xreflabel.attribute?
  & db.revisionflag.attribute?
  & db.dir.attribute?
  & db.effectivity.attributes
  & db.rdfalite.attributes
db.common.attributes =
  db.xml.id.attribute?
  & db.common.base.attributes
  & db.annotations.attribute?
db.common.idreq.attributes =
  db.xml.id.attribute
  & db.common.base.attributes
  & db.annotations.attribute?
db.common.linking.attributes =
  (db.linkend.attribute | db.xlink.attributes)?
db.common.req.linking.attributes =
  db.linkend.attribute | db.xlink.attributes
db.common.data.attributes =

  ## Specifies the format of the data
  attribute format { text }?,
  (
   ## Indentifies the location of the data by URI
   attribute fileref { xsd:anyURI }
   |
     ## Identifies the location of the data by external identifier (entity name)
     attribute entityref { xsd:ENTITY })
db.verbatim.continuation.enumeration =

  ## Line numbering continues from the immediately preceding element with the same
name.
  "continues"
  |
    ## Line numbering restarts (begins at 1, usually).
    "restarts"
db.verbatim.continuation.attribute =

  ## Determines whether line numbering continues from the previous element or restarts.
  attribute continuation { db.verbatim.continuation.enumeration }
db.verbatim.linenumbering.enumeration =

  ## Lines are numbered.
  "numbered"
  |
    ## Lines are not numbered.
    "unnumbered"
db.verbatim.linenumbering.attribute =

  ## Determines whether lines are numbered.
  attribute linenumbering { db.verbatim.linenumbering.enumeration }
db.verbatim.startinglinenumber.attribute =

  ## Specifies the initial line number.
  attribute startinglinenumber { xsd:integer }
db.verbatim.language.attribute =

  ## Identifies the language (i.e. programming language) of the verbatim content.
  attribute language { text }
db.verbatim.xml.space.attribute =

  ## Can be used to indicate explicitly that whitespace in the verbatim environment is
preserved. Whitespace must always be preserved in verbatim environments whether this
```

```
attribute is specified or not.
  attribute xml:space {

    ## Whitespace must be preserved.
    "preserve"
  }
db.verbatim.common.attributes =
  db.verbatim.continuation.attribute?
  & db.verbatim.linenumbering.attribute?
  & db.verbatim.startinglinenumber.attribute?
  & db.verbatim.xml.space.attribute?
db.verbatim.attributes =
  db.verbatim.common.attributes & db.verbatim.language.attribute?
db.label.attribute =

  ## Specifies an identifying string for presentation purposes
  attribute label { text }
db.width.characters.attribute =

  ## Specifies the width (in characters) of the element
  attribute width { xsd:nonNegativeInteger }
db.spacing.enumeration =

  ## The spacing should be "compact".
  "compact"
  |
    ## The spacing should be "normal".
    "normal"
db.spacing.attribute =

  ## Specifies (a hint about) the spacing of the content
  attribute spacing { db.spacing.enumeration }
db.pgwide.enumeration =

  ## The element should be rendered in the current text flow (with the flow column
width).
  "0"
  |
    ## The element should be rendered across the full text page.
    "1"
db.pgwide.attribute =

  ## Indicates if the element is rendered across the column or the page
  attribute pgwide { db.pgwide.enumeration }
db.language.attribute =

  ## Identifies the language (i.e. programming language) of the content.
  attribute language { text }
db.performance.enumeration =

  ## The content describes an optional step or steps.
  "optional"
  |
    ## The content describes a required step or steps.
    "required"
db.performance.attribute =

  ## Specifies if the content is required or optional.
  attribute performance { db.performance.enumeration }
db.floatstyle.attribute =

  ## Specifies style information to be used when rendering the float
  attribute floatstyle { text }
db.width.attribute =
  ## Specifies the width of the element
  attribute width { text }
db.depth.attribute =

  ## Specifies the depth of the element
  attribute depth { text }
db.contentwidth.attribute =

  ## Specifies the width of the content rectangle
  attribute contentwidth { text }
db.contentdepth.attribute =

  ## Specifies the depth of the content rectangle
  attribute contentdepth { text }
db.scalefit.enumeration =

  ## False (do not scale-to-fit; anamorphic scaling may occur)
  "0"
  |
    ## True (scale-to-fit; anamorphic scaling is forbidden)
    "1"
db.scale.attribute =

  ## Specifies the scaling factor
  attribute scale { xsd:positiveInteger }
db.classid.attribute =

  ## Specifies a classid for a media object player
  attribute classid { text }
db.autoplay.attribute =

  ## Specifies the autoplay setting for a media object player
  attribute autoplay { text }
db.halign.enumeration =

  ## Centered horizontally
  "center"
  |
    ## Aligned horizontally on the specified character
    "char"
  |
    ## Fully justified (left and right margins or edges)
    "justify"
  |
    ## Left aligned
    "left"
  |
    ## Right aligned
    "right"
db.valign.enumeration =

  ## Aligned on the bottom of the region
  "bottom"
  |
    ## Centered vertically
    "middle"
  |
    ## Aligned on the top of the region
    "top"
db.biblio.class.enumeration =

  ## A digital object identifier.
  "doi"
  |
    ## An international standard book number.
    "isbn"
  |
    ## An international standard technical report number (ISO 10444).
    "isrn"
  |
    ## An international standard serial number.
    "issn"
  |
    ## An international standard text code.
    "istc"
  |
    ## A Library of Congress reference number.
    "libraryofcongress"
  |
    ## A publication number (an internal number or possibly organizational standard).
    "pubsnumber"
  |
    ## A Uniform Resource Identifier
    "uri"
db.biblio.class-enum.attribute =

  ## Identifies the kind of bibliographic identifier
  attribute class { db.biblio.class.enumeration }?
db.biblio.class-other.attribute =

  ## Identifies the nature of the non-standard bibliographic identifier
  attribute otherclass { xsd:NMTOKEN }
db.biblio.class-other.attributes =

  ## Identifies the kind of bibliographic identifier
  attribute class {

    ## Indicates that the identifier is some 'other' kind.
    "other"
  }
  & db.biblio.class-other.attribute
db.biblio.class.attribute =
  db.biblio.class-enum.attribute | db.biblio.class-other.attributes
db.ubiq.inlines =
  (db.inlinemediaobject
   | db.remark
   | db.link.inlines
   | db.alt
   | db.trademark
   | # below, effectively the publishing inlines (as of 5.0)
     db.abbrev
   | db.acronym
   | db.date
   | db._emphasis
   | db.footnote
   | db.footnoteref
   | db._foreignphrase
   | db._phrase
   | db._quote
   | db.subscript
   | db.superscript
   | db.wordasword)
  | db.annotation
  | (db._firstterm | db._glossterm)
  | db.indexterm
  | db.coref
db._text = (text | db.ubiq.inlines | db._phrase | db.replaceable)*
db._title = db.title? & db.titleabbrev? & db.subtitle?
db._title.req = db.title & db.titleabbrev? & db.subtitle?
db._title.only = db.title? & db.titleabbrev?
db._title.onlyreq = db.title & db.titleabbrev?
db._info = (db._title, db.titleforbidden.info?) | db.info?
db._info.title.req =
  (db._title.req, db.titleforbidden.info?) | db.titlereq.info
db._info.title.only =
  (db._title.only, db.titleforbidden.info?) | db.titleonly.info
db._info.title.onlyreq =
  (db._title.onlyreq, db.titleforbidden.info?) | db.titleonlyreq.info
db._info.title.forbidden = db.titleforbidden.info?
db.all.inlines =
  text | db.ubiq.inlines | db.general.inlines | db.domain.inlines
db.general.inlines =
  db.publishing.inlines
  | db.product.inlines
  | db.bibliography.inlines
  | db.graphic.inlines
  | db.indexing.inlines
  | db.link.inlines
db.domain.inlines =
  db.technical.inlines
  | db.math.inlines
  | db.markup.inlines
  | db.gui.inlines
  | db.keyboard.inlines
  | db.os.inlines
  | db.programming.inlines
  | db.error.inlines
db.technical.inlines =
  (db.replaceable | db.package | db.parameter)
  | db.termdef
  | db.nonterminal
  | (db.systemitem | db.option | db.optional | db.property)
db.product.inlines =
  db.trademark
  | (db.productnumber
     | db.productname
     | db.database
     | db.application
     | db.hardware)
db.bibliography.inlines =
  db.citation
  | db.citerefentry
  | db.citetitle
  | db.citebiblioid
  | db.author
  | db.person
  | db.personname
  | db.org
  | db.orgname
  | db.editor
  | db.jobtitle
db.publishing.inlines =
  (db.abbrev
   | db.acronym
   | db.date
   | db.emphasis
   | db.footnote
   | db.footnoteref
   | db.foreignphrase
   | db.phrase
   | db.quote
   | db.subscript
   | db.superscript
   | db.wordasword)
  | db.glossary.inlines
  | db.coref
db.graphic.inlines = db.inlinemediaobject
db.indexing.inlines = notAllowed | db.indexterm
db.link.inlines =
  (db.xref | db.link | db.olink | db.anchor) | db.biblioref
db.nopara.blocks =
  (db.list.blocks
   | db.wrapper.blocks
   | db.formal.blocks
   | db.informal.blocks
   | db.publishing.blocks
   | db.graphic.blocks
   | db.technical.blocks
   | db.verbatim.blocks
   | db.bridgehead
   | db.remark
   | db.revhistory)
  | db.indexterm
  | db.synopsis.blocks
  | db.admonition.blocks
db.para.blocks = db.anchor | db.para | db.formalpara | db.simpara
db.all.blocks = (db.nopara.blocks | db.para.blocks) | db.annotation
db.wrapper.blocks = db.formalgroup
db.formal.blocks = (db.example | db.figure | db.table) | db.equation
db.informal.blocks =
  (db.informalexample | db.informalfigure | db.informaltable)
  | db.informalequation
db.publishing.blocks =
  db.sidebar | db.blockquote | db.address | db.epigraph
db.graphic.blocks = db.mediaobject | db.screenshot
db.technical.blocks =
  db.procedure
  | db.task
  | (db.productionset | db.constraintdef)
  | db.msgset
db.list.blocks =
  (db.itemizedlist
   | db.orderedlist
   | db.procedure
   | db.simplelist
   | db.variablelist
   | db.segmentedlist)
  | db.glosslist
  | db.bibliolist
  | db.calloutlist
  | db.qandaset
db.verbatim.blocks =
  (db.screen | db.literallayout)
  | (db.programlistingco | db.screenco)
  | (db.programlisting | db.synopsis)
db.info.extension = db._any
db.info.elements =
  (db.abstract
   | db.address
   | db.artpagenums
   | db.author
   | db.authorgroup
   | db.authorinitials
   | db.bibliocoverage
   | db.biblioid
   | db.bibliosource
   | db.collab
   | db.confgroup
   | db.contractsponsor
   | db.contractnum
```

```
    | db.copyright
    | db.cover
    | db.date
    | db.edition
    | db.editor
    | db.issuenum
    | db.keywordset
    | db.legalnotice
    | db.mediaobject
    | db.org
    | db.orgname
    | db.othercredit
    | db.pagenums
    | db.printhistory
    | db.pubdate
    | db.publisher
    | db.publishername
    | db.releaseinfo
    | db.revhistory
    | db.seriesvolnums
    | db.subjectset
    | db.volumenum
    | db.info.extension)
   | db.annotation
   | db.extendedlink
   | (db.bibliomisc | db.bibliomset | db.bibliorelation | db.biblioset)
   | db.itermset
   | (db.productname | db.productnumber)
db.bibliographic.elements =
  db.info.elements
  | db.publishing.inlines
  | db.citerefentry
  | db.citetitle
  | db.citebiblioid
  | db.person
  | db.personblurb
  | db.personname
  | db.subtitle
  | db.title
  | db.titleabbrev
div {
  db.title.role.attribute = attribute role { text }
  db.title.attlist =
    db.title.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.title =

    ## The text of the title of a section of a document or of a formal block-level
element
    element title { db.title.attlist, db.all.inlines* }
}
div {
  db.titleabbrev.role.attribute = attribute role { text }
  db.titleabbrev.attlist =
    db.titleabbrev.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.titleabbrev =

    ## The abbreviation of a title
    element titleabbrev { db.titleabbrev.attlist, db.all.inlines* }
}
div {
  db.subtitle.role.attribute = attribute role { text }
  db.subtitle.attlist =
    db.subtitle.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.subtitle =

    ## The subtitle of a document
    element subtitle { db.subtitle.attlist, db.all.inlines* }
}
div {
  db.info.role.attribute = attribute role { text }
  db.info.attlist = db.info.role.attribute? & db.common.attributes
  db.info =

    ## A wrapper for information about a component or other block
    element info { db.info.attlist, (db._title & db.info.elements*) }
}
div {
  db.titlereq.info.role.attribute = attribute role { text }
  db.titlereq.info.attlist =
    db.titlereq.info.role.attribute? & db.common.attributes
  db.titlereq.info =

    ## A wrapper for information about a component or other block with a required title
    element info {
      db.titlereq.info.attlist, (db._title.req & db.info.elements*)
    }
}
div {
  db.titleonly.info.role.attribute = attribute role { text }
  db.titleonly.info.attlist =
    db.titleonly.info.role.attribute? & db.common.attributes
  db.titleonly.info =

    ## A wrapper for information about a component or other block with only a title
    element info {
      db.titleonly.info.attlist, (db._title.only & db.info.elements*)
    }
}
div {
  db.titleonlyreq.info.role.attribute = attribute role { text }
  db.titleonlyreq.info.attlist =
    db.titleonlyreq.info.role.attribute? & db.common.attributes
  db.titleonlyreq.info =

    ## A wrapper for information about a component or other block with only a required
title
    element info {
      db.titleonlyreq.info.attlist,
      (db._title.onlyreq & db.info.elements*)
    }
}
div {
  db.titleforbidden.info.role.attribute = attribute role { text }
  db.titleforbidden.info.attlist =
    db.titleforbidden.info.role.attribute? & db.common.attributes
  db.titleforbidden.info =

    ## A wrapper for information about a component or other block without a title
    element info { db.titleforbidden.info.attlist, db.info.elements* }
}
div {
  db.subjectset.role.attribute = attribute role { text }
  db.subjectset.scheme.attribute =
    ## Identifies the controlled vocabulary used by this set's terms
    attribute scheme { xsd:NMTOKEN }
  db.subjectset.attlist =
    db.subjectset.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.subjectset.scheme.attribute?
  db.subjectset =

    ## A set of terms describing the subject matter of a document
    element subjectset { db.subjectset.attlist, db.subject+ }
}
div {
  db.subject.role.attribute = attribute role { text }
  db.subject.weight.attribute =

    ## Specifies a ranking for this subject relative to other subjects in the same set
    attribute weight { text }
  db.subject.attlist =
    db.subject.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.subject.weight.attribute?
  db.subject =

    ## One of a group of terms describing the subject matter of a document
    element subject { db.subject.attlist, db.subjectterm+ }
}
div {
  db.subjectterm.role.attribute = attribute role { text }
  db.subjectterm.attlist =
    db.subjectterm.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.subjectterm =

    ## A term in a group of terms describing the subject matter of a document
    element subjectterm { db.subjectterm.attlist, text }
}
div {
  db.keywordset.role.attribute = attribute role { text }
  db.keywordset.attlist =
    db.keywordset.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.keywordset =

    ## A set of keywords describing the content of a document
    element keywordset { db.keywordset.attlist, db.keyword+ }
}
div {
  db.keyword.role.attribute = attribute role { text }
  db.keyword.attlist =
    db.keyword.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.keyword =

    ## One of a set of keywords describing the content of a document
    element keyword { db.keyword.attlist, text }
}
db.table.choice = notAllowed | db.cals.table | db.html.table
db.informaltable.choice =
  notAllowed | db.cals.informaltable | db.html.informaltable
db.table = db.table.choice
db.informaltable = db.informaltable.choice
div {
  db.procedure.role.attribute = attribute role { text }
  db.procedure.attlist =
    db.procedure.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.procedure.info = db._info.title.only
  db.procedure =

    ## A list of operations to be performed in a well-defined sequence
    element procedure {
      db.procedure.attlist,
      db.procedure.info,
      db.all.blocks*,
      db.step+,
      db.result?
    }
}
div {
  db.step.role.attribute = attribute role { text }
  db.step.attlist =
    db.step.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.performance.attribute?
  db.step.info = db._info.title.only
  # This content model is blocks*, step|stepalternatives, blocks* but
  # expressed this way it avoids UPA issues in XSD and DTD versions
  db.step =

    ## A unit of action in a procedure
    element step {
      db.step.attlist,
      db.step.info,
      ((db.all.blocks+,
        ((db.substeps | db.stepalternatives), db.all.blocks*)?,
        db.result?)
       | ((db.substeps | db.stepalternatives),
          db.all.blocks*,
          db.result?))
    }
}
div {
  db.stepalternatives.role.attribute = attribute role { text }
  db.stepalternatives.attlist =
    db.stepalternatives.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.performance.attribute?
  db.stepalternatives.info = db._info.title.forbidden
  db.stepalternatives =

    ## Alternative steps in a procedure
    element stepalternatives {
      db.stepalternatives.attlist, db.stepalternatives.info, db.step+
    }
}
div {
  db.substeps.role.attribute = attribute role { text }
  db.substeps.attlist =
    db.substeps.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.performance.attribute?
  db.substeps =

    ## A wrapper for steps that occur within steps in a procedure
    element substeps { db.substeps.attlist, db.step+ }
}
div {
  db.result.role.attribute = attribute role { text }
  db.result.attlist =
    db.result.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.result =

    ## A wrapper for identifying the results of a procedure or step
    element result { db.result.attlist, db.all.blocks+ }
}
div {
  db.sidebar.floatstyle.attribute = db.floatstyle.attribute
  db.sidebar.role.attribute = attribute role { text }
  db.sidebar.attlist =
    db.sidebar.role.attribute?
    & db.sidebar.floatstyle.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.sidebar.info = db._info
  db.sidebar =

    ## A portion of a document that is isolated from the main narrative flow
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:sidebar"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:sidebar)"
            "sidebar must not occur among the children or descendants of sidebar"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element sidebar {
      db.sidebar.attlist, db.sidebar.info, db.all.blocks+
    }
}
div {
  db.abstract.role.attribute = attribute role { text }
  db.abstract.attlist =
    db.abstract.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.abstract.info = db._info.title.only
  db.abstract =

    ## A summary
    element abstract {
      db.abstract.attlist, db.abstract.info, db.~~para~~all.blocks+
    }
}
div {
  db.personblurb.role.attribute = attribute role { text }
  db.personblurb.attlist =
    db.personblurb.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.personblurb.info = db._info.title.only
  db.personblurb =

    ## A short description or note about a person
    element personblurb {
      db.personblurb.attlist, db.personblurb.info, db.para.blocks+
    }
}
div {
  db.blockquote.role.attribute = attribute role { text }
  db.blockquote.attlist =
    db.blockquote.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.blockquote.info = db._info.title.only
  db.blockquote =

    ## A quotation set off from the main text
    element blockquote {
      db.blockquote.attlist,
      db.blockquote.info,
      db.attribution?,
      db.all.blocks+
    }
}
div {
  db.attribution.role.attribute = attribute role { text }
  db.attribution.attlist =
    db.attribution.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.attribution =

    ## The source of a block quote or epigraph
    element attribution {
      db.attribution.attlist,
      (db._text
       | db.person
       | db.personname
       | db.citetitle
       | db.citation)*
    }
}
div {
  db.bridgehead.renderas.enumeration =

    ## Render as a first-level section
    "sect1"
    |
      ## Render as a second-level section
      "sect2"
    |
      ## Render as a third-level section
      "sect3"
    |
      ## Render as a fourth-level section
      "sect4"
    |
      ## Render as a fifth-level section
      "sect5"
  db.bridgehead.renderas-enum.attribute =

    ## Indicates how the bridge head should be rendered
    attribute renderas { db.bridgehead.renderas.enumeration }?
  db.bridgehead.renderas-other.attribute =

    ## Identifies the nature of the non-standard rendering
    attribute otherrenderas { xsd:NMTOKEN }
  db.bridgehead.renderas-other.attributes =

    ## Indicates how the bridge head should be rendered
    attribute renderas {

      ## Identifies a non-standard rendering
      "other"
    }
    & db.bridgehead.renderas-other.attribute
  db.bridgehead.renderas.attribute =
    db.bridgehead.renderas-enum.attribute
    | db.bridgehead.renderas-other.attributes
  db.bridgehead.role.attribute = attribute role { text }
  db.bridgehead.attlist =
    db.bridgehead.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.bridgehead.renderas.attribute?
  db.bridgehead =

    ## A free-floating heading
    element bridgehead { db.bridgehead.attlist, db.all.inlines* }
}
div {
  db.remark.role.attribute = attribute role { text }
  db.remark.attlist =
    db.remark.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.remark =

    ## A remark (or comment) intended for presentation in a draft manuscript
    element remark { db.remark.attlist, db.all.inlines* }
}
div {
  db.epigraph.role.attribute = attribute role { text }
  db.epigraph.attlist =
    db.epigraph.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.epigraph.info = db._info.title.forbidden
  db.epigraph =

    ## A short inscription at the beginning of a document or component
    element epigraph {
      db.epigraph.attlist,
      db.epigraph.info,
      db.attribution?,
      (db.para.blocks | db.literallayout)+
    }
}
div {
  db.footnote.role.attribute = attribute role { text }
  db.footnote.label.attribute =

    ## Identifies the desired footnote mark
    attribute label { xsd:NMTOKEN }
  db.footnote.attlist =
    db.footnote.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.footnote.label.attribute?
  db.footnote =

    ## A footnote
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:footnote)"
            "footnote must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:example)"
            "example must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:figure)"
            "figure must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:table)"
            "table must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:equation)"
            "equation must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:sidebar)"
            "sidebar must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:task)"
            "task must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:epigraph)"
            "epigraph must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:caution)"
            "caution must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:danger)"
            "danger must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:important)"
            "important must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:note)"
            "note must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
```

```
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:tip)"
            "tip must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:footnote"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:warning)"
            "warning must not occur among the children or descendants of footnote"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element footnote { db.footnote.attlist, db.all.blocks+ }
}
div {
  db.formalpara.role.attribute = attribute role { text }
  db.formalpara.attlist =
    db.formalpara.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.formalpara.info = db._info.title.onlyreq
  db.formalpara =

    ## A paragraph with a title
    element formalpara {
      db.formalpara.attlist,
      db.formalpara.info,
      db.indexing.inlines*,
      db.para
    }
}
div {
  db.para.role.attribute = attribute role { text }
  db.para.attlist =
    db.para.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.para.info = db._info.title.forbidden
  db.para =

    ## A paragraph
    element para {
      db.para.attlist,
      db.para.info,
      (db.all.inlines | db.nopara.blocks)*
    }
}
div {
  db.simpara.role.attribute = attribute role { text }
  db.simpara.attlist =
    db.simpara.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.simpara.info = db._info.title.forbidden
  db.simpara =

    ## A paragraph that contains only text and inline markup, no block elements
    element simpara {
      db.simpara.attlist, db.simpara.info, db.all.inlines*
    }
}
div {
  db.itemizedlist.role.attribute = attribute role { text }
  db.itemizedlist.mark.attribute =

    ## Identifies the type of mark to be used on items in this list
    attribute mark { xsd:NMTOKEN }
  db.itemizedlist.attlist =
    db.itemizedlist.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.spacing.attribute?
    & db.itemizedlist.mark.attribute?
  db.itemizedlist.info = db._info.title.only
  db.itemizedlist =

    ## A list in which each entry is marked with a bullet or other dingbat
    element itemizedlist {
      db.itemizedlist.attlist,
      db.itemizedlist.info,
      db.all.blocks*,
      db.listitem+
    }
}
div {
  db.orderedlist.role.attribute = attribute role { text }
  db.orderedlist.continuation.enumeration =

    ## Specifies that numbering should begin where the preceding list left off
    "continues"
    |
      ## Specifies that numbering should begin again at 1
      "restarts"
  db.orderedlist.continuation.attribute =

    ## Indicates how list numbering should begin relative to the immediately preceding
list
    attribute continuation { db.orderedlist.continuation.enumeration }
  db.orderedlist.startingnumber.attribute =

    ## Specifies the initial line number.
    attribute startingnumber { xsd:integer }
  db.orderedlist.inheritnum.enumeration =

    ## Specifies that numbering should ignore list nesting
    "ignore"
    |
      ## Specifies that numbering should inherit from outer-level lists
      "inherit"
  db.orderedlist.inheritnum.attribute =

    ## Indicates whether or not item numbering should be influenced by list nesting
    attribute inheritnum { db.orderedlist.inheritnum.enumeration }
  db.orderedlist.numeration.enumeration =

    ## Specifies Arabic numeration (1, 2, 3, â€¦)
    "arabic"
    |
      ## Specifies upper-case alphabetic numeration (A, B, C, â€¦)
      "upperalpha"
    |
      ## Specifies lower-case alphabetic numeration (a, b, c, â€¦)
      "loweralpha"
    |
      ## Specifies upper-case Roman numeration (I, II, III, â€¦)
      "upperroman"
    |
      ## Specifies lower-case Roman numeration (i, ii, iii â€¦)
      "lowerroman"
  db.orderedlist.numeration.attribute =

    ## Indicates the desired numeration
    attribute numeration { db.orderedlist.numeration.enumeration }
  db.orderedlist.attlist =
    db.orderedlist.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.spacing.attribute?
    & (db.orderedlist.continuation.attribute
       | db.orderedlist.startingnumber.attribute)?
    & db.orderedlist.inheritnum.attribute?
    & db.orderedlist.numeration.attribute?
  db.orderedlist.info = db._info.title.only
  db.orderedlist =

    ## A list in which each entry is marked with a sequentially incremented label
    element orderedlist {
      db.orderedlist.attlist,
      db.orderedlist.info,
      db.all.blocks*,
      db.listitem+
    }
}
div {
  db.listitem.role.attribute = attribute role { text }
  db.listitem.override.attribute =

    ## Specifies the keyword for the type of mark that should be used on this
    ##  item, instead of the mark that would be used by default
    attribute override { xsd:NMTOKEN }
  db.listitem.attlist =
    db.listitem.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.listitem.override.attribute?
  db.listitem =

    ## A wrapper for the elements of a list item
    element listitem { db.listitem.attlist, db.all.blocks+ }
}
div {
  db.segmentedlist.role.attribute = attribute role { text }
  db.segmentedlist.attlist =
    db.segmentedlist.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.segmentedlist.info = db._info.title.only
  db.segmentedlist =

    ## A segmented list, a list of sets of elements
    element segmentedlist {
      db.segmentedlist.attlist,
      db.segmentedlist.info,
      db.segtitle+,
      db.seglistitem+
    }
}
div {
  db.segtitle.role.attribute = attribute role { text }
  db.segtitle.attlist =
    db.segtitle.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.segtitle =

    ## The title of an element of a list item in a segmented list
    element segtitle { db.segtitle.attlist, db.all.inlines* }
}
div {
  db.seglistitem.role.attribute = attribute role { text }
  db.seglistitem.attlist =
    db.seglistitem.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.seglistitem =

    ## A list item in a segmented list
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Cardinality of segments and titles" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:seglistitem"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "count(db:seg) = count(../db:segtitle)"
            "The number of seg elements must be the same as the number of segtitle
elements in the parent segmentedlist"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element seglistitem { db.seglistitem.attlist, db.seg+ }
}
div {
  db.seg.role.attribute = attribute role { text }
  db.seg.attlist =
    db.seg.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.seg =

    ## An element of a list item in a segmented list
    element seg { db.seg.attlist, db.all.inlines* }
}
div {
  db.simplelist.role.attribute = attribute role { text }
  db.simplelist.type.enumeration =

    ## A tabular presentation in row-major order.
    "horiz"
    |
      ## A tabular presentation in column-major order.
      "vert"
    |
      ## An inline presentation, usually a comma-delimited list.
      "inline"
  db.simplelist.type.attribute =

    ## Specifies the type of list presentation.
    [ a:defaultValue = "vert" ]
    attribute type { db.simplelist.type.enumeration }
  db.simplelist.columns.attribute =

    ## Specifies the number of columns for horizontal or vertical presentation
    attribute columns { xsd:integer }
  db.simplelist.attlist =
    db.simplelist.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.simplelist.type.attribute?
    & db.simplelist.columns.attribute?
  db.simplelist =

    ## An undecorated list of single words or short phrases
    element simplelist { db.simplelist.attlist, db.member+ }
}
div {
  db.member.role.attribute = attribute role { text }
  db.member.attlist =
    db.member.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.member =

    ## An element of a simple list
    element member { db.member.attlist, db.all.inlines* }
}
div {
  db.variablelist.role.attribute = attribute role { text }
  db.variablelist.termlength.attribute =

    ## Indicates a length beyond which the presentation system may consider a term too
long and select an alternate presentation for that term, item, or list
    attribute termlength { text }
  db.variablelist.attlist =
    db.variablelist.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.spacing.attribute?
    & db.variablelist.termlength.attribute?
  db.variablelist.info = db._info.title.only
  db.variablelist =
    ## A list in which each entry is composed of a set of one or more terms and an
associated description
    element variablelist {
      db.variablelist.attlist,
      db.variablelist.info,
      db.all.blocks*,
      db.varlistentry+
    }
}
div {
  db.varlistentry.role.attribute = attribute role { text }
  db.varlistentry.attlist =
    db.varlistentry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.varlistentry =

    ## A wrapper for a set of terms and the associated description in a variable list
    element varlistentry {
      db.varlistentry.attlist, db.term+, db.listitem
    }
}
div {
  db.term.role.attribute = attribute role { text }
  db.term.attlist =
    db.term.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.term =

    ## The word or phrase being defined or described in a variable list
    element term { db.term.attlist, db.all.inlines* }
}
div {
  db.example.role.attribute = attribute role { text }
  db.example.label.attribute = db.label.attribute
  db.example.width.attribute = db.width.characters.attribute
  db.example.pgwide.attribute = db.pgwide.attribute
  db.example.floatstyle.attribute = db.floatstyle.attribute
  db.example.attlist =
    db.example.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.example.label.attribute?
    & db.example.floatstyle.attribute?
    & (db.example.width.attribute | db.example.pgwide.attribute)?
  db.example.info = db._info.title.onlyreq
  db.example =

    ## A formal example, with a title
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:example"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:example)"
            "example must not occur among the children or descendants of example"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:example"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:figure)"
            "figure must not occur among the children or descendants of example"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:example"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:table)"
            "table must not occur among the children or descendants of example"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:example"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:equation)"
            "equation must not occur among the children or descendants of example"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
```

```
      ]
    ]
    element example {
      db.example.attlist, db.example.info, db.all.blocks+, db.caption?
    }
}
div {
  db.informalexample.role.attribute = attribute role { text }
  db.informalexample.width.attribute = db.width.characters.attribute
  db.informalexample.pgwide.attribute = db.pgwide.attribute
  db.informalexample.floatstyle.attribute = db.floatstyle.attribute
  db.informalexample.attlist =
    db.informalexample.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.informalexample.floatstyle.attribute?
    & (db.informalexample.width.attribute
       | db.informalexample.pgwide.attribute)?
  db.informalexample.info = db._info.title.forbidden
  db.informalexample =

    ## A displayed example without a title
    element informalexample {
      db.informalexample.attlist,
      db.informalexample.info,
      db.all.blocks+,
      db.caption?
    }
}
db.verbatim.inlines = (db.all.inlines | db.lineannotation) | db.co
db.verbatim.contentmodel =
  db._info.title.forbidden, (db.textobject | db.verbatim.inlines*)
div {
  db.literallayout.role.attribute = attribute role { text }
  db.literallayout.class.enumeration =

    ## The literal layout should be formatted with a monospaced font
    "monospaced"
    |
      ## The literal layout should be formatted with the current font
      "normal"
  db.literallayout.class.attribute =

    ## Specifies the class of literal layout
    attribute class { db.literallayout.class.enumeration }
  db.literallayout.attlist =
    db.literallayout.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.verbatim.attributes
    & db.literallayout.class.attribute?
  db.literallayout =

    ## A block of text in which line breaks and white space are to be reproduced
faithfully
    element literallayout {
      db.literallayout.attlist, db.verbatim.contentmodel
    }
}
div {
  db.screen.role.attribute = attribute role { text }
  db.screen.width.attribute = db.width.characters.attribute
  db.screen.attlist =
    db.screen.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.verbatim.attributes
    & db.screen.width.attribute?
  db.screen =

    ## Text that a user sees or might see on a computer screen
    element screen { db.screen.attlist, db.verbatim.contentmodel }
}
div {
  db.screenshot.role.attribute = attribute role { text }
  db.screenshot.attlist =
    db.screenshot.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.screenshot.info = db._info
  db.screenshot =

    ## A representation of what the user sees or might see on a computer screen
    element screenshot {
      db.screenshot.attlist, db.screenshot.info, db.mediaobject
    }
}
div {
  db.figure.role.attribute = attribute role { text }
  db.figure.label.attribute = db.label.attribute
  db.figure.pgwide.attribute = db.pgwide.attribute
  db.figure.floatstyle.attribute = db.floatstyle.attribute
  db.figure.attlist =
    db.figure.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.figure.label.attribute?
    & db.figure.pgwide.attribute?
    & db.figure.floatstyle.attribute?
  db.figure.info = db._info.title.onlyreq
  db.figure =

    ## A formal figure, generally an illustration, with a title
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:figure"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:example)"
            "example must not occur among the children or descendants of figure"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:figure"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:figure)"
            "figure must not occur among the children or descendants of figure"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:figure"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:table)"
            "table must not occur among the children or descendants of figure"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:figure"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:equation)"
            "equation must not occur among the children or descendants of figure"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element figure {
      db.figure.attlist, db.figure.info, db.all.blocks+, db.caption?
    }
}
div {
  db.informalfigure.role.attribute = attribute role { text }
  db.informalfigure.label.attribute = db.label.attribute
  db.informalfigure.pgwide.attribute = db.pgwide.attribute
  db.informalfigure.floatstyle.attribute = db.floatstyle.attribute
  db.informalfigure.attlist =
    db.informalfigure.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.informalfigure.label.attribute?
    & db.informalfigure.pgwide.attribute?
    & db.informalfigure.floatstyle.attribute?
  db.informalfigure.info = db._info.title.forbidden
  db.informalfigure =

    ## A untitled figure
    element informalfigure {
      db.informalfigure.attlist,
      db.informalfigure.info,
      db.all.blocks+,
      db.caption?
    }
}
db.mediaobject.content =
  (db.videoobject | db.audioobject | db.imageobject | db.textobject)
  | db.imageobjectco
div {
  db.mediaobject.role.attribute = attribute role { text }
  db.mediaobject.attlist =
    db.mediaobject.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.mediaobject.info = db._info.title.forbidden
  db.mediaobject =

    ## A displayed media object (video, audio, image, etc.)
    element mediaobject {
      db.mediaobject.attlist,
      db.mediaobject.info,
      db.alt?,
      db.mediaobject.content+,
      db.caption?
    }
}
div {
  db.inlinemediaobject.role.attribute = attribute role { text }
  db.inlinemediaobject.attlist =
    db.inlinemediaobject.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.inlinemediaobject.info = db._info.title.forbidden
  db.inlinemediaobject =

    ## An inline media object (video, audio, image, and so on)
    element inlinemediaobject {
      db.inlinemediaobject.attlist,
      db.inlinemediaobject.info,
      db.alt?,
      db.mediaobject.content+
    }
}
div {
  db.videoobject.role.attribute = attribute role { text }
  db.videoobject.attlist =
    db.videoobject.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.videoobject.info = db._info.title.forbidden
  db.videoobject =

    ## A wrapper for video data and its associated meta-information
    element videoobject {
      db.videoobject.attlist, db.videoobject.info, db.videodata+
    }
}
div {
  db.audioobject.role.attribute = attribute role { text }
  db.audioobject.attlist =
    db.audioobject.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.audioobject.info = db._info.title.forbidden
  db.audioobject =

    ## A wrapper for audio data and its associated meta-information
    element audioobject {
      db.audioobject.attlist, db.audioobject.info, db.audiodata+
    }
}
db.imageobject.content =
  db.imagedata+ | db.imagedata.mathml | db.imagedata.svg+
div {
  db.imageobject.role.attribute = attribute role { text }
  db.imageobject.attlist =
    db.imageobject.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.imageobject.info = db._info.title.forbidden
  db.imageobject =

    ## A wrapper for image data and its associated meta-information
    element imageobject {
      db.imageobject.attlist,
      db.imageobject.info,
      db.imageobject.content
    }
}
div {
  db.textobject.role.attribute = attribute role { text }
  db.textobject.attlist =
    db.textobject.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.textobject.info = db._info.title.forbidden
  db.textobject =

    ## A wrapper for a text description of an object and its associated meta-
information
    element textobject {
      db.textobject.attlist,
      db.textobject.info,
      (db.phrase | db.textdata | db.all.blocks+)
    }
}
div {
  db.videodata.role.attribute = attribute role { text }
  db.videodata.align.enumeration = db.halign.enumeration
  db.videodata.align.attribute =

    ## Specifies the (horizontal) alignment of the video data
    attribute align { db.videodata.align.enumeration }
  db.videodata.autoplay.attribute = db.autoplay.attribute
  db.videodata.classid.attribute = db.classid.attribute
  db.videodata.valign.enumeration = db.valign.enumeration
  db.videodata.valign.attribute =
    ## Specifies the vertical alignment of the video data
    attribute valign { db.videodata.valign.enumeration }
  db.videodata.width.attribute = db.width.attribute
  db.videodata.depth.attribute = db.depth.attribute
  db.videodata.contentwidth.attribute = db.contentwidth.attribute
  db.videodata.contentdepth.attribute = db.contentdepth.attribute
  db.videodata.scalefit.enumeration = db.scalefit.enumeration
  db.videodata.scalefit.attribute =

    ## Determines if anamorphic scaling is forbidden
    attribute scalefit { db.videodata.scalefit.enumeration }
  db.videodata.scale.attribute = db.scale.attribute
  db.videodata.attlist =
    db.videodata.role.attribute?
    & db.common.attributes
    & db.common.data.attributes
    & db.videodata.align.attribute?
    & db.videodata.valign.attribute?
    & db.videodata.width.attribute?
    & db.videodata.contentwidth.attribute?
    & db.videodata.scalefit.attribute?
    & db.videodata.scale.attribute?
    & db.videodata.depth.attribute?
    & db.videodata.contentdepth.attribute?
    & db.videodata.autoplay.attribute?
    & db.videodata.classid.attribute?
  db.videodata.info = db._info.title.forbidden
  db.videodata =

    ## Pointer to external video data
    element videodata {
      db.videodata.attlist, db.videodata.info, db.multimediaparam*
    }
}
div {
  db.audiodata.role.attribute = attribute role { text }
  db.audiodata.align.enumeration = db.halign.enumeration
  db.audiodata.align.attribute =

    ## Specifies the (horizontal) alignment of the video data
    attribute align { db.audiodata.align.enumeration }
  db.audiodata.autoplay.attribute = db.autoplay.attribute
  db.audiodata.classid.attribute = db.classid.attribute
  db.audiodata.contentwidth.attribute = db.contentwidth.attribute
  db.audiodata.contentdepth.attribute = db.contentdepth.attribute
  db.audiodata.depth.attribute = db.depth.attribute
  db.audiodata.scale.attribute = db.scale.attribute
  db.audiodata.scalefit.enumeration = db.scalefit.enumeration
  db.audiodata.scalefit.attribute =

    ## Determines if anamorphic scaling is forbidden
    attribute scalefit { db.audiodata.scalefit.enumeration }
  db.audiodata.valign.enumeration = db.valign.enumeration
  db.audiodata.valign.attribute =

    ## Specifies the vertical alignment of the video data
    attribute valign { db.audiodata.valign.enumeration }
  db.audiodata.width.attribute = db.width.attribute
  db.audiodata.attlist =
    db.audiodata.role.attribute?
    & db.common.attributes
    & db.common.data.attributes
    & db.audiodata.align.attribute?
    & db.audiodata.autoplay.attribute?
    & db.audiodata.classid.attribute?
    & db.audiodata.contentdepth.attribute?
    & db.audiodata.contentwidth.attribute?
    & db.audiodata.depth.attribute?
    & db.audiodata.scale.attribute?
    & db.audiodata.scalefit.attribute?
    & db.audiodata.valign.attribute?
    & db.audiodata.width.attribute?
  db.audiodata.info = db._info.title.forbidden
  db.audiodata =

    ## Pointer to external audio data
    element audiodata {
      db.audiodata.attlist, db.audiodata.info, db.multimediaparam*
    }
}
div {
  db.imagedata.role.attribute = attribute role { text }
  db.imagedata.align.enumeration = db.halign.enumeration
  db.imagedata.align.attribute =

    ## Specifies the (horizontal) alignment of the image data
    attribute align { db.imagedata.align.enumeration }
  db.imagedata.valign.enumeration = db.valign.enumeration
  db.imagedata.valign.attribute =

    ## Specifies the vertical alignment of the image data
    attribute valign { db.imagedata.valign.enumeration }
  db.imagedata.width.attribute = db.width.attribute
  db.imagedata.depth.attribute = db.depth.attribute
  db.imagedata.contentwidth.attribute = db.contentwidth.attribute
  db.imagedata.contentdepth.attribute = db.contentdepth.attribute
  db.imagedata.scalefit.enumeration = db.scalefit.enumeration
  db.imagedata.scalefit.attribute =

    ## Determines if anamorphic scaling is forbidden
    attribute scalefit { db.imagedata.scalefit.enumeration }
  db.imagedata.scale.attribute = db.scale.attribute
  db.imagedata.attlist =
    db.imagedata.role.attribute?
    & db.common.attributes
    & db.common.data.attributes
    & db.imagedata.align.attribute?
    & db.imagedata.valign.attribute?
    & db.imagedata.width.attribute?
    & db.imagedata.contentwidth.attribute?
    & db.imagedata.scalefit.attribute?
    & db.imagedata.scale.attribute?
    & db.imagedata.depth.attribute?
    & db.imagedata.contentdepth.attribute?
  db.imagedata.info = db._info.title.forbidden
  db.imagedata =

    ## Pointer to external image data
    element imagedata {
    ~~element imagedata { ~~  db.imagedata.attlist, db.imagedata.info~~ }~~,
db.multimediaparam*
    }
}
div {
  db.textdata.role.attribute = attribute role { text }
  db.textdata.encoding.attribute =

    ## Identifies the encoding of the text in the external file
    attribute encoding { text }
  db.textdata.attlist =
    db.textdata.role.attribute?
    & db.common.attributes
    & db.common.data.attributes
    & db.textdata.encoding.attribute?
  db.textdata.info = db._info.title.forbidden
  db.textdata =

    ## Pointer to external text data
    element textdata { db.textdata.attlist, db.textdata.info }
}
div {
  db.multimediaparam.role.attribute = attribute role { text }
  db.multimediaparam.name.attribute =

    ## Specifies the name of the parameter
    attribute name { text }
  db.multimediaparam.value.attribute =

    ## Specifies the value of the parameter
    attribute value { text }
  db.multimediaparam.valuetype.attribute =

    ## Specifies the type of the value of the parameter
    attribute valuetype { text }
  db.multimediaparam.attlist =
    db.multimediaparam.role.attribute?
    & db.common.attributes
    & db.multimediaparam.name.attribute
    & db.multimediaparam.value.attribute
    & db.multimediaparam.valuetype.attribute?
  db.multimediaparam =

    ## Application specific parameters for a media player
    element multimediaparam { db.multimediaparam.attlist, empty }
}
div {
  db.caption.role.attribute = attribute role { text }
  db.caption.attlist =
    db.caption.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.caption.info = db._info.title.forbidden
  db.caption =

    ## A caption
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                    "
          s:assert [
            test = "not(.//db:example)"
            "example must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
```

```
]
s:pattern [
  "\x{a}" ~
  "                "
  s:title [ "Element exclusion" ]
  "\x{a}" ~
  "                "
  s:rule [
    context = "db:caption"
    "\x{a}" ~
    "                   "
    s:assert [
      test = "not(.//db:figure)"
      "figure must not occur among the children or descendants of caption"
    ]
    "\x{a}" ~
    "                "
  ]
  "\x{a}" ~
  "             "
]
s:pattern [
  "\x{a}" ~
  "                "
  s:title [ "Element exclusion" ]
  "\x{a}" ~
  "                "
  s:rule [
    context = "db:caption"
    "\x{a}" ~
    "                   "
    s:assert [
      test = "not(.//db:table)"
      "table must not occur among the children or descendants of caption"
    ]
    "\x{a}" ~
    "                "
  ]
  "\x{a}" ~
  "             "
]
s:pattern [
  "\x{a}" ~
  "                "
  s:title [ "Element exclusion" ]
  "\x{a}" ~
  "                "
  s:rule [
    context = "db:caption"
    "\x{a}" ~
    "                   "
    s:assert [
      test = "not(.//db:equation)"
      "equation must not occur among the children or descendants of caption"
    ]
    "\x{a}" ~
    "                "
  ]
  "\x{a}" ~
  "             "
]
s:pattern [
  "\x{a}" ~
  "                "
  s:title [ "Element exclusion" ]
```

```
    "\x{a}" ~
    "                "
    s:rule [
      context = "db:caption"
      "\x{a}" ~
      "                   "
      s:assert [
        test = "not(.//db:sidebar)"
        "sidebar must not occur among the children or descendants of caption"
      ]
      "\x{a}" ~
      "                "
    ]
    "\x{a}" ~
    "             "
  ]
  s:pattern [
    "\x{a}" ~
    "                "
    s:title [ "Element exclusion" ]
    "\x{a}" ~
    "                "
    s:rule [
      context = "db:caption"
      "\x{a}" ~
      "                   "
      s:assert [
        test = "not(.//db:task)"
        "task must not occur among the children or descendants of caption"
      ]
      "\x{a}" ~
      "                "
    ]
    "\x{a}" ~
    "             "
  ]
  s:pattern [
    "\x{a}" ~
    "                "
    s:title [ "Element exclusion" ]
    "\x{a}" ~
    "                "
    s:rule [
      context = "db:caption"
      "\x{a}" ~
      "                   "
      s:assert [
        test = "not(.//db:caution)"
        "caution must not occur among the children or descendants of caption"
      ]
      "\x{a}" ~
      "                "
    ]
    "\x{a}" ~
    "             "
  ]
  s:pattern [
    "\x{a}" ~
    "                "
    s:title [ "Element exclusion" ]
    "\x{a}" ~
    "                "
    s:rule [
      context = "db:caption"
      "\x{a}" ~
      "                  "
      s:assert [
        test = "not(.//db:danger)"
        "danger must not occur among the children or descendants of caption"
      ]
      "\x{a}" ~
      "               "
    ]
    "\x{a}" ~
    "            "
  ]
  s:pattern [
    "\x{a}" ~
    "               "
    s:title [ "Element exclusion" ]
    "\x{a}" ~
    "               "
    s:rule [
      context = "db:caption"
      "\x{a}" ~
      "                  "
      s:assert [
        test = "not(.//db:important)"
        "important must not occur among the children or descendants of caption"
      ]
      "\x{a}" ~
      "               "
    ]
    "\x{a}" ~
    "            "
  ]
  s:pattern [
    "\x{a}" ~
    "               "
    s:title [ "Element exclusion" ]
    "\x{a}" ~
    "               "
    s:rule [
      context = "db:caption"
      "\x{a}" ~
      "                  "
      s:assert [
        test = "not(.//db:note)"
        "note must not occur among the children or descendants of caption"
      ]
      "\x{a}" ~
      "               "
    ]
    "\x{a}" ~
    "            "
  ]
  s:pattern [
    "\x{a}" ~
    "               "
    s:title [ "Element exclusion" ]
    "\x{a}" ~
    "               "
    s:rule [
      context = "db:caption"
      "\x{a}" ~
      "                  "
      s:assert [
        test = "not(.//db:tip)"
        "tip must not occur among the children or descendants of caption"
```

```
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:warning)"
            "warning must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element caption {
      db.caption.attlist, db.caption.info, db.all.blocks+
    }
}
div {
  db.address.role.attribute = attribute role { text }
  db.address.attlist =
    db.address.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.verbatim.attributes
  db.address =

    ## A real-world address, generally a postal address
    element address {
      db.address.attlist,
      (db._text
       | db.personname
       | db.orgname
       | db.pob
       | db.street
       | db.city
       | db.state
       | db.postcode
       | db.country
       | db.phone
       | db.fax
       | db.email
       | db.uri
       | db.otheraddr)*
    }
}
div {
  db.street.role.attribute = attribute role { text }
  db.street.attlist =
    db.street.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.street =

    ## A street address in an address
    element street { db.street.attlist, db._text }
}
div {
  db.pob.role.attribute = attribute role { text }
  db.pob.attlist =
    db.pob.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.pob =

    ## A post office box in an address
    element pob { db.pob.attlist, db._text }
}
div {
  db.postcode.role.attribute = attribute role { text }
  db.postcode.attlist =
    db.postcode.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.postcode =

    ## A postal code in an address
    element postcode { db.postcode.attlist, db._text }
}
div {
  db.city.role.attribute = attribute role { text }
  db.city.attlist =
    db.city.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.city =

    ## The name of a city in an address
    element city { db.city.attlist, db._text }
}
div {
  db.state.role.attribute = attribute role { text }
  db.state.attlist =
    db.state.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.state =

    ## A state or province in an address
    element state { db.state.attlist, db._text }
}
div {
  db.country.role.attribute = attribute role { text }
  db.country.attlist =
    db.country.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.country =

    ## The name of a country
    element country { db.country.attlist, db._text }
}
div {
  db.phone.role.attribute = attribute role { text }
  db.phone.attlist =
    db.phone.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.phone =

    ## A telephone number
    element phone { db.phone.attlist, db._text }
}
div {
  db.fax.role.attribute = attribute role { text }
  db.fax.attlist =
    db.fax.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.fax =

    ## A fax number
    element fax { db.fax.attlist, db._text }
}
div {
  db.otheraddr.role.attribute = attribute role { text }
  db.otheraddr.attlist =
    db.otheraddr.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.otheraddr =

    ## Uncategorized information in address
    element otheraddr { db.otheraddr.attlist, db._text }
}
div {
  db.affiliation.role.attribute = attribute role { text }
  db.affiliation.attlist =
    db.affiliation.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.affiliation =

    ## The institutional affiliation of an individual
    element affiliation {
      db.affiliation.attlist,
      db.shortaffil?,
      db.jobtitle*,
      (db.org? | (db.orgname?, db.orgdiv*, db.address*))
    }
}
div {
  db.shortaffil.role.attribute = attribute role { text }
  db.shortaffil.attlist =
    db.shortaffil.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.shortaffil =

    ## A brief description of an affiliation
    element shortaffil { db.shortaffil.attlist, db._text }
}
div {
  db.jobtitle.role.attribute = attribute role { text }
  db.jobtitle.attlist =
    db.jobtitle.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.jobtitle =

    ## The title of an individual in an organization
    element jobtitle { db.jobtitle.attlist, db._text }
}
div {
  db.orgname.class.enumeration =

    ## A consortium
    "consortium"
    |
      ## A corporation
      "corporation"
    |
      ## An informal organization
      "informal"
    |
      ## A non-profit organization
      "nonprofit"
  db.orgname.class-enum.attribute =

    ## Specifies the nature of the organization
    attribute class { db.orgname.class.enumeration }
  db.orgname.class-other.attributes =

    ## Specifies the nature of the organization
    attribute class {

      ## Indicates a non-standard organization class
      "other"
    },

    ## Identifies the non-standard nature of the organization
    attribute otherclass { text }
  db.orgname.class.attribute =
    db.orgname.class-enum.attribute | db.orgname.class-other.attributes
  db.orgname.role.attribute = attribute role { text }
  db.orgname.attlist =
    db.orgname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.orgname.class.attribute?
  db.orgname =

    ## The name of an organization
    element orgname { db.orgname.attlist, db._text }
}
div {
  db.orgdiv.role.attribute = attribute role { text }
  db.orgdiv.attlist =
    db.orgdiv.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.orgdiv =

    ## A division of an organization
    element orgdiv { db.orgdiv.attlist, db.all.inlines* }
}
div {
  db.artpagenums.role.attribute = attribute role { text }
  db.artpagenums.attlist =
    db.artpagenums.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.artpagenums =

    ## The page numbers of an article as published
    element artpagenums { db.artpagenums.attlist, db._text }
```

```
}
div {
  db.personname.role.attribute = attribute role { text }
  db.personname.attlist =
    db.personname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.personname =

    ## The personal name of an individual
    element personname {
      db.personname.attlist,
      (db._text
       | (db.honorific
          | db.firstname
          | db.surname
          | db.lineage
          | db.othername)+
       | (db.honorific
          | db.givenname
          | db.surname
          | db.lineage
          | db.othername)+)
    }
}
db.person.author.contentmodel =
  db.personname,
  (db.personblurb
   | db.affiliation
   | db.email
   | db.uri
   | db.address
   | db.contrib)*
db.org.author.contentmodel =
  db.orgname,
  (db.orgdiv
   | db.affiliation
   | db.email
   | db.uri
   | db.address
   | db.contrib)*
db.credit.contentmodel =
  db.person.author.contentmodel | db.org.author.contentmodel
div {
  db.author.role.attribute = attribute role { text }
  db.author.attlist =
    db.author.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.author =

    ## The name of an individual author
    element author { db.author.attlist, db.credit.contentmodel }
}
div {
  db.authorgroup.role.attribute = attribute role { text }
  db.authorgroup.attlist =
    db.authorgroup.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.authorgroup =

    ## Wrapper for author information when a document has multiple authors or
collaborators
    element authorgroup {
      db.authorgroup.attlist, (db.author | db.editor | db.othercredit)+
    }
}
div {
  db.collab.role.attribute = attribute role { text }
  db.collab.attlist =
    db.collab.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.collab =

    ## Identifies a collaborator
    element collab {
      db.collab.attlist,
      (db.person | db.personname | db.org | db.orgname)+,
      db.affiliation*
    }
}
div {
  db.authorinitials.role.attribute = attribute role { text }
  db.authorinitials.attlist =
    db.authorinitials.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.authorinitials =

    ## The initials or other short identifier for an author
    element authorinitials { db.authorinitials.attlist, db._text }
}
div {
  db.person.role.attribute = attribute role { text }
  db.person.attlist =
    db.person.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.person =

    ## A person and associated metadata
    element person {
      db.person.attlist,
      db.personname,
      (db.address
       | db.affiliation
       | db.email
       | db.uri
       | db.personblurb)*
    }
}
div {
  db.org.role.attribute = attribute role { text }
  db.org.attlist =
    db.org.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.org =

    ## An organization and associated metadata
    element org {
      db.org.attlist,
      db.orgname,
      (db.address | db.affiliation | db.email | db.uri | db.orgdiv)*
    }
}
div {
  db.confgroup.role.attribute = attribute role { text }
  db.confgroup.attlist =
    db.confgroup.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.confgroup =

    ## A wrapper for document meta-information about a conference
    element confgroup {
      db.confgroup.attlist,
      (db.confdates
       | db.conftitle
       | db.confnum
       | db.confsponsor
       | db.address)*
    }
}
div {
  db.confdates.role.attribute = attribute role { text }
  db.confdates.attlist =
    db.confdates.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.confdates =

    ## The dates of a conference for which a document was written
    element confdates { db.confdates.attlist, db._text }
}
div {
  db.conftitle.role.attribute = attribute role { text }
  db.conftitle.attlist =
    db.conftitle.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.conftitle =

    ## The title of a conference for which a document was written
    element conftitle { db.conftitle.attlist, db._text }
}
div {
  db.confnum.role.attribute = attribute role { text }
  db.confnum.attlist =
    db.confnum.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.confnum =

    ## An identifier, frequently numerical, associated with a conference for which a
document was written
    element confnum { db.confnum.attlist, db._text }
}
div {
  db.confsponsor.role.attribute = attribute role { text }
  db.confsponsor.attlist =
    db.confsponsor.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.confsponsor =

    ## The sponsor of a conference for which a document was written
    element confsponsor { db.confsponsor.attlist, db._text }
}
div {
  db.contractnum.role.attribute = attribute role { text }
  db.contractnum.attlist =
    db.contractnum.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.contractnum =

    ## The contract number of a document
    element contractnum { db.contractnum.attlist, db._text }
}
div {
  db.contractsponsor.role.attribute = attribute role { text }
  db.contractsponsor.attlist =
    db.contractsponsor.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.contractsponsor =

    ## The sponsor of a contract
    element contractsponsor { db.contractsponsor.attlist, db._text }
}
div {
  db.copyright.role.attribute = attribute role { text }
  db.copyright.attlist =
    db.copyright.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.copyright =

    ## Copyright information about a document
    element copyright { db.copyright.attlist, db.year+, db.holder* }
}
div {
  db.year.role.attribute = attribute role { text }
  db.year.attlist =
    db.year.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.year =

    ## The year of publication of a document
    element year { db.year.attlist, db._text }
}
div {
  db.holder.role.attribute = attribute role { text }
  db.holder.attlist =
    db.holder.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.holder =

    ## The name of the individual or organization that holds a copyright
    element holder { db.holder.attlist, db._text }
}
db.cover.contentmodel =
  (db.para.blocks
   | db.list.blocks
   | db.informal.blocks
   | db.publishing.blocks
   | db.graphic.blocks
   | db.technical.blocks
   | db.verbatim.blocks
   | db.bridgehead
   | db.remark
   | db.revhistory)
  | db.synopsis.blocks
div {
  db.cover.role.attribute = attribute role { text }
  db.cover.attlist =
    db.cover.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.cover =

    ## Additional content for the cover of a publication
    element cover { db.cover.attlist, db.cover.contentmodel+ }
}
db.date.contentmodel =
  xsd:date | xsd:dateTime | xsd:gYearMonth | xsd:gYear | text
div {
  db.date.role.attribute = attribute role { text }
  db.date.attlist =
    db.date.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.date =

    ## The date of publication or revision of a document
    element date { db.date.attlist, db.date.contentmodel }
}
div {
  db.edition.role.attribute = attribute role { text }
  db.edition.attlist =
    db.edition.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.edition =

    ## The name or number of an edition of a document
    element edition { db.edition.attlist, db._text }
}
div {
  db.editor.role.attribute = attribute role { text }
  db.editor.attlist =
    db.editor.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.editor =

    ## The name of the editor of a document
    element editor { db.editor.attlist, db.credit.contentmodel }
}
div {
  db.biblioid.role.attribute = attribute role { text }
  db.biblioid.attlist =
    db.biblioid.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.biblio.class.attribute
  db.biblioid =

    ## An identifier for a document
    element biblioid { db.biblioid.attlist, db._text }
}
div {
  db.citebiblioid.role.attribute = attribute role { text }
  db.citebiblioid.attlist =
    db.citebiblioid.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.biblio.class.attribute
  db.citebiblioid =
    ## A citation of a bibliographic identifier
    element citebiblioid { db.citebiblioid.attlist, db._text }
}
div {
  db.bibliosource.role.attribute = attribute role { text }
  db.bibliosource.attlist =
    db.bibliosource.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.biblio.class.attribute
  db.bibliosource =

    ## The source of a document
    element bibliosource { db.bibliosource.attlist, db._text }
}
div {
  db.bibliorelation.type.enumeration =

    ## The described resource pre-existed the referenced resource, which is essentially
the same intellectual content presented in another format
    "hasformat"
    |
      ## The described resource includes the referenced resource either physically or
logically
      "haspart"
    |
      ## The described resource has a version, edition, or adaptation, namely, the
referenced resource
      "hasversion"
    |
      ## The described resource is the same intellectual content of the referenced
resource, but presented in another format
      "isformatof"
    |
      ## The described resource is a physical or logical part of the referenced
resource
      "ispartof"
    |
      ## The described resource is referenced, cited, or otherwise pointed to by the
referenced resource
      "isreferencedby"
    |
      ## The described resource is supplanted, displaced, or superceded by the
referenced resource
      "isreplacedby"
    |
      ## The described resource is required by the referenced resource, either
physically or logically
      "isrequiredby"
    |
      ## The described resource is a version, edition, or adaptation of the referenced
resource; changes in version imply substantive changes in content rather than
differences in format
      "isversionof"
    |
      ## The described resource references, cites, or otherwise points to the
referenced resource
      "references"
    |
      ## The described resource supplants, displaces, or supersedes the referenced
resource
      "replaces"
    |
      ## The described resource requires the referenced resource to support its
function, delivery, or coherence of content
```

```
      "requires"
  db.bibliorelation.type-enum.attribute =

    ## Identifies the type of relationship
    attribute type { db.bibliorelation.type.enumeration }?
  db.bibliorelation.type-other.attributes =

    ## Identifies the type of relationship
    attribute type {

      ## The described resource has a non-standard relationship with the referenced
resource
      "othertype"
    }?,

    ## A keyword that identififes the type of the non-standard relationship
    attribute othertype { xsd:NMTOKEN }
  db.bibliorelation.type.attribute =
    db.bibliorelation.type-enum.attribute
    | db.bibliorelation.type-other.attributes
  db.bibliorelation.role.attribute = attribute role { text }
  db.bibliorelation.attlist =
    db.bibliorelation.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.biblio.class.attribute
    & db.bibliorelation.type.attribute
  db.bibliorelation =

    ## The relationship of a document to another
    element bibliorelation { db.bibliorelation.attlist, db._text }
}
div {
  db.bibliocoverage.spacial.enumeration =

    ## The DCMI Point identifies a point in space using its geographic coordinates
    "dcmipoint"
    |
      ## ISO 3166 Codes for the representation of names of countries
      "iso3166"
    |
      ## The DCMI Box identifies a region of space using its geographic limits
      "dcmibox"
    |
      ## The Getty Thesaurus of Geographic Names
      "tgn"
  db.bibliocoverage.spatial-enum.attribute =

    ## Specifies the type of spatial coverage
    attribute spatial { db.bibliocoverage.spacial.enumeration }?
  db.bibliocoverage.spatial-other.attributes =

    ## Specifies the type of spatial coverage
    attribute spatial {

      ## Identifies a non-standard type of coverage
      "otherspatial"
    }?,

    ## A keyword that identifies the type of non-standard coverage
    attribute otherspatial { xsd:NMTOKEN }
  db.bibliocoverage.spatial.attribute =
    db.bibliocoverage.spatial-enum.attribute
    | db.bibliocoverage.spatial-other.attributes
  db.bibliocoverage.temporal.enumeration =
    ## A specification of the limits of a time interval
    "dcmiperiod"
    |
      ## W3C Encoding rules for dates and timesâ€”a profile based on ISO 8601
      "w3c-dtf"
  db.bibliocoverage.temporal-enum.attribute =

    ## Specifies the type of temporal coverage
    attribute temporal { db.bibliocoverage.temporal.enumeration }?
  db.bibliocoverage.temporal-other.attributes =

    ## Specifies the type of temporal coverage
    attribute temporal {

      ## Specifies a non-standard type of coverage
      "othertemporal"
    }?,

    ## A keyword that identifies the type of non-standard coverage
    attribute othertemporal { xsd:NMTOKEN }
  db.bibliocoverage.temporal.attribute =
    db.bibliocoverage.temporal-enum.attribute
    | db.bibliocoverage.temporal-other.attributes
  db.bibliocoverage.coverage.attrib =
    db.bibliocoverage.spatial.attribute
    & db.bibliocoverage.temporal.attribute
  db.bibliocoverage.role.attribute = attribute role { text }
  db.bibliocoverage.attlist =
    db.bibliocoverage.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.bibliocoverage.coverage.attrib
  db.bibliocoverage =

    ## The spatial or temporal coverage of a document
    element bibliocoverage { db.bibliocoverage.attlist, db._text }
}
div {
  db.legalnotice.role.attribute = attribute role { text }
  db.legalnotice.attlist =
    db.legalnotice.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.legalnotice.info = db._info.title.only
  db.legalnotice =

    ## A statement of legal obligations or requirements
    element legalnotice {
      db.legalnotice.attlist,~~ db.legalnotice.info, db.all.blocks+~~
      db.legalnotice.info,
      db.all.blocks+,
      db.legalsection*
    }
}
div {
  db.othercredit.class.enumeration =

    ## A copy editor
    "copyeditor"
    |
      ## A graphic designer
      "graphicdesigner"
    |
      ## A production editor
      "productioneditor"
    |
      ## A technical editor
      "technicaleditor"
    |
      ## A translator
      "translator"
    |
      ## An indexer
      "indexer"
    |
      ## A proof-reader
      "proofreader"
    |
      ## A cover designer
      "coverdesigner"
    |
      ## An interior designer
      "interiordesigner"
    |
      ## An illustrator
      "illustrator"
    |
      ## A reviewer
      "reviewer"
    |
      ## A typesetter
      "typesetter"
    |
      ## A converter (a persons responsible for conversion, not an application)
      "conversion"
  db.othercredit.class-enum.attribute =

    ## Identifies the nature of the contributor
    attribute class { db.othercredit.class.enumeration }?
  db.othercredit.class-other.attribute =

    ## Identifies the nature of the non-standard contribution
    attribute otherclass { xsd:NMTOKEN }
  db.othercredit.class-other.attributes =

    ## Identifies the nature of the contributor
    attribute class {

      ## Identifies a non-standard contribution
      "other"
    }
    & db.othercredit.class-other.attribute
  db.othercredit.class.attribute =
    db.othercredit.class-enum.attribute
    | db.othercredit.class-other.attributes
  db.othercredit.role.attribute = attribute role { text }
  db.othercredit.attlist =
    db.othercredit.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.othercredit.class.attribute
  db.othercredit =

    ## A person or entity, other than an author or editor, credited in a document
    element othercredit {
      db.othercredit.attlist, db.credit.contentmodel
    }
}
div {
  db.pagenums.role.attribute = attribute role { text }
  db.pagenums.attlist =
    db.pagenums.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.pagenums =

    ## The numbers of the pages in a book, for use in a bibliographic entry
    element pagenums { db.pagenums.attlist, db._text }
}
div {
  db.contrib.role.attribute = attribute role { text }
  db.contrib.attlist =
    db.contrib.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.contrib =

    ## A summary of the contributions made to a document by a credited source
    element contrib { db.contrib.attlist, db.all.inlines* }
}
div {
  db.honorific.role.attribute = attribute role { text }
  db.honorific.attlist =
    db.honorific.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.honorific =

    ## The title of a person
    element honorific { db.honorific.attlist, db._text }
}
div {
  db.firstname.role.attribute = attribute role { text }
  db.firstname.attlist =
    db.firstname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.firstname =

    ## A given name of a person
    element firstname { db.firstname.attlist, db._text }
}
div {
  db.givenname.role.attribute = attribute role { text }
  db.givenname.attlist =
    db.givenname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.givenname =

    ## The given name of a person
    element givenname { db.givenname.attlist, db._text }
}
div {
  db.surname.role.attribute = attribute role { text }
  db.surname.attlist =
    db.surname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.surname =

    ## An inherited or family name; in western cultures the last name
    element surname { db.surname.attlist, db._text }
}
```

```
div {
  db.lineage.role.attribute = attribute role { text }
  db.lineage.attlist =
    db.lineage.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.lineage =

    ## The portion of a person's name indicating a relationship to ancestors
    element lineage { db.lineage.attlist, db._text }
}
div {
  db.othername.role.attribute = attribute role { text }
  db.othername.attlist =
    db.othername.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.othername =

    ## A component of a person's name that is not a first name, surname, or lineage
    element othername { db.othername.attlist, db._text }
}
div {
  db.printhistory.role.attribute = attribute role { text }
  db.printhistory.attlist =
    db.printhistory.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.printhistory =

    ## The printing history of a document
    element printhistory { db.printhistory.attlist, db.para.blocks+ }
}
div {
  db.pubdate.role.attribute = attribute role { text }
  db.pubdate.attlist =
    db.pubdate.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.pubdate =

    ## The date of publication of a document
    element pubdate { db.pubdate.attlist, db.date.contentmodel }
}
div {
  db.publisher.role.attribute = attribute role { text }
  db.publisher.attlist =
    db.publisher.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.publisher =

    ## The publisher of a document
    element publisher {
      db.publisher.attlist, db.publishername, db.address*
    }
}
div {
  db.publishername.role.attribute = attribute role { text }
  db.publishername.attlist =
    db.publishername.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.publishername =
    ## The name of the publisher of a document
    element publishername { db.publishername.attlist, db._text }
}
div {
  db.releaseinfo.role.attribute = attribute role { text }
  db.releaseinfo.attlist =
    db.releaseinfo.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.releaseinfo =

    ## Information about a particular release of a document
    element releaseinfo { db.releaseinfo.attlist, db._text }
}
div {
  db.revhistory.role.attribute = attribute role { text }
  db.revhistory.attlist =
    db.revhistory.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.revhistory.info = db._info.title.only
  db.revhistory =

    ## A history of the revisions to a document
    element revhistory {
      db.revhistory.attlist, db.revhistory.info, db.revision+
    }
}
div {
  db.revision.role.attribute = attribute role { text }
  db.revision.attlist =
    db.revision.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.revision =

    ## An entry describing a single revision in the history of the revisions to a
document
    element revision {
      db.revision.attlist,
      db.revnumber?,
      db.date,
      (db.authorinitials | db.author)*,
      (db.revremark | db.revdescription)?
    }
}
div {
  db.revnumber.role.attribute = attribute role { text }
  db.revnumber.attlist =
    db.revnumber.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.revnumber =

    ## A document revision number
    element revnumber { db.revnumber.attlist, db._text }
}
div {
  db.revremark.role.attribute = attribute role { text }
  db.revremark.attlist =
    db.revremark.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.revremark =
    ## A description of a revision to a document
    element revremark { db.revremark.attlist, db._text }
}
div {
  db.revdescription.role.attribute = attribute role { text }
  db.revdescription.attlist =
    db.revdescription.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.revdescription =

    ## A extended description of a revision to a document
    element revdescription { db.revdescription.attlist, db.all.blocks* }
}
div {
  db.seriesvolnums.role.attribute = attribute role { text }
  db.seriesvolnums.attlist =
    db.seriesvolnums.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.seriesvolnums =

    ## Numbers of the volumes in a series of books
    element seriesvolnums { db.seriesvolnums.attlist, db._text }
}
div {
  db.volumenum.role.attribute = attribute role { text }
  db.volumenum.attlist =
    db.volumenum.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.volumenum =

    ## The volume number of a document in a set (as of books in a set or articles in a
journal)
    element volumenum { db.volumenum.attlist, db._text }
}
div {
  db.issuenum.role.attribute = attribute role { text }
  db.issuenum.attlist =
    db.issuenum.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.issuenum =

    ## The number of an issue of a journal
    element issuenum { db.issuenum.attlist, db._text }
}
div {
  db.package.role.attribute = attribute role { text }
  db.package.attlist =
    db.package.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.package =

    ## A software or application package
    element package { db.package.attlist, db._text }
}
div {
  db.email.role.attribute = attribute role { text }
  db.email.attlist =
    db.email.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.email =

    ## An email address
    element email { db.email.attlist, db._text }
}
div {
  db.lineannotation.role.attribute = attribute role { text }
  db.lineannotation.attlist =
    db.lineannotation.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.lineannotation =

    ## A comment on a line in a verbatim listing
    element lineannotation { db.lineannotation.attlist, db._text }
}
div {
  db.parameter.class.enumeration =

    ## A command
    "command"
    |
      ## A function
      "function"
    |
      ## An option
      "option"
  db.parameter.class.attribute =

    ## Identifies the class of parameter
    attribute class { db.parameter.class.enumeration }
  db.parameter.role.attribute = attribute role { text }
  db.parameter.attlist =
    db.parameter.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.parameter.class.attribute?
  db.parameter =

    ## A value or a symbolic reference to a value
    element parameter { db.parameter.attlist, db._text }
}
db.replaceable.inlines = db._text | db.co
div {
  db.replaceable.class.enumeration =

    ## A command
    "command"
    |
      ## A function
      "function"
    |
      ## An option
      "option"
    |
      ## A parameter
      "parameter"
  db.replaceable.class.attribute =

    ## Identifies the nature of the replaceable text
    attribute class { db.replaceable.class.enumeration }
  db.replaceable.role.attribute = attribute role { text }
  db.replaceable.attlist =
    db.replaceable.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.replaceable.class.attribute?
  db.replaceable =

    ## Content that may or must be replaced by the user
    element replaceable {
      db.replaceable.attlist, db.replaceable.inlines*
    }
}
div {
  db.uri.type.attribute =

    ## Identifies the type of URI specified
    attribute type { text }?
  db.uri.role.attribute = attribute role { text }
  db.uri.attlist =
    db.uri.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.uri.type.attribute
  db.uri =

    ## A Uniform Resource Identifier
    element uri { db.uri.attlist, db._text }
}
div {
  db.abbrev.role.attribute = attribute role { text }
  db.abbrev.attlist =
    db.abbrev.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.abbrev =

    ## An abbreviation, especially one followed by a period
    element abbrev {
      db.abbrev.attlist,
      (db._text | db.superscript | db.subscript | db.trademark)*
    }
}
div {
  db.acronym.role.attribute = attribute role { text }
  db.acronym.attlist =
    db.acronym.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.acronym =

    ## An often pronounceable word made from the initial (or selected) letters of a
name or phrase
    element acronym {
      db.acronym.attlist,
      (db._text | db.superscript | db.subscript | db.trademark)*
    }
}
div {
  db.citation.role.attribute = attribute role { text }
  db.citation.attlist =
    db.citation.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.citation =

    ## An inline bibliographic reference to another published work
    element citation { db.citation.attlist, db.all.inlines* }
}
```

```
div {
  db.citerefentry.role.attribute = attribute role { text }
  db.citerefentry.attlist =
    db.citerefentry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.citerefentry =

    ## A citation to a reference page
    element citerefentry {
      db.citerefentry.attlist, db.refentrytitle, db.manvolnum?
    }
}
div {
  db.refentrytitle.role.attribute = attribute role { text }
  db.refentrytitle.attlist =
    db.refentrytitle.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.refentrytitle =

    ## The title of a reference page
    element refentrytitle { db.refentrytitle.attlist, db.all.inlines* }
}
div {
  db.manvolnum.role.attribute = attribute role { text }
  db.manvolnum.attlist =
    db.manvolnum.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.manvolnum =

    ## A reference volume number
    element manvolnum { db.manvolnum.attlist, db._text }
}
div {
  db.citetitle.pubwork.enumeration =

    ## An article
    "article"
    |
      ## A bulletin board system
      "bbs"
    |
      ## A book
      "book"
    |
      ## A CD-ROM
      "cdrom"
    |
      ## A chapter (as of a book)
      "chapter"
    |
      ## A DVD
      "dvd"
    |
      ## An email message
      "emailmessage"
    |
      ## A gopher page
      "gopher"
    |
      ## A journal
      "journal"
    |
      ## A manuscript
      "manuscript"
    |
      ## A posting to a newsgroup
      "newsposting"
    |
      ## A part (as of a book)
      "part"
    |
      ## A reference entry
      "refentry"
    |
      ## A section (as of a book or article)
      "section"
    |
      ## A series
      "series"
    |
      ## A set (as of books)
      "set"
    |
      ## A web page
      "webpage"
    |
      ## A wiki page
      "wiki"
  db.citetitle.pubwork.attribute =

    ## Identifies the nature of the publication being cited
    attribute pubwork { db.citetitle.pubwork.enumeration }
  db.citetitle.role.attribute = attribute role { text }
  db.citetitle.attlist =
    db.citetitle.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.citetitle.pubwork.attribute?
  db.citetitle =

    ## The title of a cited work
    element citetitle { db.citetitle.attlist, db.all.inlines* }
}
div {
  db.emphasis.role.attribute = attribute role { text }
  db.emphasis.attlist =
    db.emphasis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.emphasis =

    ## Emphasized text
    element emphasis { db.emphasis.attlist, db.all.inlines* }
}
div {
  db._emphasis =

    ## A limited span of emphasized text
    element emphasis { db.emphasis.attlist, db._text }
}
div {
  db.foreignphrase.role.attribute = attribute role { text }
  db.foreignphrase.attlist =
    db.foreignphrase.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.foreignphrase =
    ## A word or phrase in a language other than the primary language of the document
    element foreignphrase {
      db.foreignphrase.attlist, (text | db.general.inlines)*
    }
}
div {
  db._foreignphrase.role.attribute = attribute role { text }
  db._foreignphrase.attlist =
    db._foreignphrase.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db._foreignphrase =

    ## A limited word or phrase in a language other than the primary language of the
document
    element foreignphrase { db._foreignphrase.attlist, db._text }
}
div {
  db.phrase.role.attribute = attribute role { text }
  db.phrase.attlist =
    db.phrase.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.phrase =

    ## A span of text
    element phrase { db.phrase.attlist, db.all.inlines* }
}
div {
  db._phrase =

    ## A limited span of text
    element phrase { db.phrase.attlist, db._text }
}
div {
  db.quote.role.attribute = attribute role { text }
  db.quote.attlist =
    db.quote.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.quote =

    ## An inline quotation
    element quote { db.quote.attlist, db.all.inlines* }
}
div {
  db._quote.role.attribute = attribute role { text }
  db._quote.attlist =
    db._quote.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db._quote =

    ## A limited inline quotation
    element quote { db._quote.attlist, db._text }
}
div {
  db.subscript.role.attribute = attribute role { text }
  db.subscript.attlist =
    db.subscript.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.subscript =
~~    ## A subscript (as in H2~~
    ## A subscript (as in Hâ‚‚O, the molecular formula for water)
    element subscript { db.subscript.attlist, db._text }
}
div {
  db.superscript.role.attribute = attribute role { text }
  db.superscript.attlist =
    db.superscript.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.superscript =

~~    ## A superscript (as in x2~~
    ## A superscript (as in xÂ², the mathematical notation for x multiplied by itself)
    element superscript { db.superscript.attlist, db._text }
}
div {
  db.trademark.class.enumeration =

    ## A copyright
    "copyright"
    |
      ## A registered copyright
      "registered"
    |
      ## A service
      "service"
    |
      ## A trademark
      "trade"
  db.trademark.class.attribute =

    ## Identifies the class of trade mark
    attribute class { db.trademark.class.enumeration }
  db.trademark.role.attribute = attribute role { text }
  db.trademark.attlist =
    db.trademark.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.trademark.class.attribute?
  db.trademark =

    ## A trademark
    element trademark { db.trademark.attlist, db._text }
}
div {
  db.wordasword.role.attribute = attribute role { text }
  db.wordasword.attlist =
    db.wordasword.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.wordasword =

    ## A word meant specifically as a word and not representing anything else
    element wordasword { db.wordasword.attlist, db._text }
}
div {
  db.footnoteref.role.attribute = attribute role { text }
  db.footnoteref.label.attribute = db.label.attribute
  db.footnoteref.attlist =
    db.footnoteref.role.attribute?
    & db.common.attributes
    & db.linkend.attribute
    & db.footnoteref.label.attribute?
  db.footnoteref =
    ## A cross reference to a footnote (a footnote mark)
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Footnote reference type constraint" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:footnoteref"
          "\x{a}" ~
          "                   "
          s:assert [
            test =
              "local-name(//*[@xml:id=current()/@linkend]) = 'footnote' and namespace-
uri(//*[@xml:id=current()/@linkend]) = 'http://docbook.org/ns/docbook'"
            "@linkend on footnoteref must point to a footnote."
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element footnoteref { db.footnoteref.attlist, empty }
}
div {
  db.xref.role.attribute = attribute role { text }
  db.xref.xrefstyle.attribute = db.xrefstyle.attribute
  db.xref.endterm.attribute = db.endterm.attribute
  db.xref.attlist =
    db.xref.role.attribute?
    & db.common.attributes
    & db.common.req.linking.attributes
    & db.xref.xrefstyle.attribute?
    & db.xref.endterm.attribute?
  db.xref =

    ## A cross reference to another part of the document
    element xref { db.xref.attlist, empty }
}
div {
  db.link.role.attribute = attribute role { text }
  db.link.xrefstyle.attribute = db.xrefstyle.attribute
  db.link.endterm.attribute = db.endterm.attribute
  db.link.attlist =
    db.link.role.attribute?
    & db.common.attributes
    & db.common.req.linking.attributes
    & db.link.xrefstyle.attribute?
    & db.link.endterm.attribute?
  db.link =

    ## A hypertext link
    element link { db.link.attlist, db.all.inlines* }
}
div {
  db.olink.role.attribute = attribute role { text }
  db.olink.xrefstyle.attribute = db.xrefstyle.attribute
  db.olink.localinfo.attribute =

    ## Holds additional information that may be used by the application when resolving
the link
```

```
    attribute localinfo { text }
  db.olink.targetdoc.attribute =

    ## Specifies the URI of the document in which the link target appears
    attribute targetdoc { xsd:anyURI }
  db.olink.targetptr.attribute =

    ## Specifies the location of the link target in the document
    attribute targetptr { text }
  db.olink.type.attribute =

    ## Identifies application-specific customization of the link behavior
    attribute type { text }
  db.olink.attlist =
    db.common.attributes
    & db.olink.targetdoc.attribute?
    & db.olink.role.attribute?
    & db.olink.xrefstyle.attribute?
    & db.olink.localinfo.attribute?
    & db.olink.targetptr.attribute?
    & db.olink.type.attribute?
  db.olink =

    ## A link that addresses its target indirectly
    element olink { db.olink.attlist, db.all.inlines* }
}
div {
  db.anchor.role.attribute = attribute role { text }
  db.anchor.attlist =
    db.anchor.role.attribute? & db.common.idreq.attributes
  db.anchor =

    ## A spot in the document
    element anchor { db.anchor.attlist, empty }
}
div {
  db.alt.role.attribute = attribute role { text }
  db.alt.attlist = db.alt.role.attribute? & db.common.attributes
  db.alt =

    ## A text-only annotation, often used for accessibility
    element alt { db.alt.attlist, (text | db.inlinemediaobject)* }
}
div {
  db.formalgroup.fgstyle.attribute =

    ## Holds style of formalgroup - this can be used to specify desired layout and
positioning of subfigures
    attribute fgstyle { text }
  db.formalgroup.role.attribute = attribute role { text }
  db.formalgroup.pgwide.attribute = db.pgwide.attribute
  db.formalgroup.floatstyle.attribute = db.floatstyle.attribute
  db.formalgroup.attlist =
    db.formalgroup.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.formalgroup.floatstyle.attribute?
    & db.formalgroup.pgwide.attribute?
    & db.formalgroup.fgstyle.attribute?
  db.formalgroup.info = db._info.title.onlyreq
  db.formalgroup =

    ## A group of formal objects, for example subfigures
    element formalgroup {
      db.formalgroup.attlist,
      db.formalgroup.info,
      (db.figure+ | db.table+ | db.example+ | db.equation+)
    }
}
db.status.attribute =

  ## Identifies the editorial or publication status of the element on which it occurs
  attribute status { text }
db.toplevel.sections =
  (((db.section | db.legalsection)+, db.simplesect*)
  ~~((db.section+, db.simplesect*)~~ | db.simplesect+)
  | (db.sect1+, db.simplesect*)
  | db.refentry+
db.toplevel.blocks.or.sections =
  (db.all.blocks+, db.toplevel.sections?) | db.toplevel.sections
db.recursive.sections =
  (((db.section | db.legalsection)+, db.simplesect*)
  ~~((db.section+, db.simplesect*)~~ | db.simplesect+)
  | db.refentry+
db.recursive.blocks.or.sections =
  (db.all.blocks+, db.recursive.sections?) | db.recursive.sections
db.divisions = db.part | db.reference
db.components =
  db.dedication
  | db.acknowledgements
  | db.preface
  | db.chapter
  | db.appendix
  | db.article
  | db.colophon
db.navigation.components =
  notAllowed | db.glossary | db.bibliography | db.index | db.toc
db.component.contentmodel =
  db.navigation.components*,
  db.toplevel.blocks.or.sections,
  db.navigation.components*
db.setindex.components = notAllowed | db.setindex
db.toc.components = notAllowed | db.toc
db.set.components = db.set | db.book | db.article
div {
  db.set.status.attribute = db.status.attribute
  db.set.role.attribute = attribute role { text }
  db.set.attlist =
    db.set.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.set.status.attribute?
  db.set.info = db._info.title.req
  db.set =

    ## A collection of books
    element set {
      db.set.attlist,
      db.set.info,
      db.toc.components?,
      db.set.components+,
      db.setindex.components?
    }
}
db.book.components =
  (db.navigation.components | db.components | db.divisions)* | db.topic*
div {
  db.book.status.attribute = db.status.attribute
  db.book.role.attribute = attribute role { text }
  db.book.attlist =
    db.book.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.book.status.attribute?
  db.book.info = db._info
  db.book =

    ## A book
    element book { db.book.attlist, db.book.info, db.book.components }
}
div {
  db.dedication.status.attribute = db.status.attribute
  db.dedication.role.attribute = attribute role { text }
  db.dedication.attlist =
    db.dedication.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.dedication.status.attribute?
  db.dedication.info = db._info
  db.dedication =

    ## The dedication of a book or other component
    element dedication {
      db.dedication.attlist, db.dedication.info, db.all.blocks+
    }
}
div {
  db.acknowledgements.status.attribute = db.status.attribute
  db.acknowledgements.role.attribute = attribute role { text }
  db.acknowledgements.attlist =
    db.acknowledgements.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.acknowledgements.status.attribute?
  db.acknowledgements.info = db._info
  db.acknowledgements =

    ## Acknowledgements of a book or other component
    element acknowledgements {
      db.acknowledgements.attlist,
      db.acknowledgements.info,
      db.all.blocks+
    }
}
div {
  db.colophon.status.attribute = db.status.attribute
  db.colophon.role.attribute = attribute role { text }
  db.colophon.attlist =
    db.colophon.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.colophon.status.attribute?
  db.colophon.info = db._info
  db.colophon =

    ## Text at the back of a book describing facts about its production
    element colophon {
      db.colophon.attlist,
      db.colophon.info,
      ((db.all.blocks+, db.simplesect*)
       | (db.all.blocks*, db.simplesect+))
    }
}
db.appendix.contentmodel = db.component.contentmodel | db.topic+
div {
  db.appendix.status.attribute = db.status.attribute
  db.appendix.role.attribute = attribute role { text }
  db.appendix.attlist =
    db.appendix.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.appendix.status.attribute?
  db.appendix.info = db._info.title.req
  db.appendix =

    ## An appendix in a book or article
    element appendix {
      db.appendix.attlist, db.appendix.info, db.appendix.contentmodel?
    }
}
db.chapter.contentmodel = db.component.contentmodel | db.topic+
div {
  db.chapter.status.attribute = db.status.attribute
  db.chapter.role.attribute = attribute role { text }
  db.chapter.attlist =
    db.chapter.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.chapter.status.attribute?
  db.chapter.info = db._info.title.req
  db.chapter =

    ## A chapter, as of a book
    element chapter {
      db.chapter.attlist, db.chapter.info, db.chapter.contentmodel?
    }
}
db.part.components =
  (db.navigation.components | db.components)
  | (db.refentry | db.reference)
db.part.contentmodel = db.part.components+ | db.topic+
div {
  db.part.status.attribute = db.status.attribute
  db.part.role.attribute = attribute role { text }
  db.part.attlist =
    db.part.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.part.status.attribute?
  db.part.info = db._info.title.req
  db.part =

    ## A division in a book
    element part {
      db.part.attlist,
      db.part.info,
      db.partintro?,
      db.part.contentmodel?
    }
}
```

```
div {
  db.preface.status.attribute = db.status.attribute
  db.preface.role.attribute = attribute role { text }
  db.preface.attlist =
    db.preface.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.preface.status.attribute?
  db.preface.info = db._info.title.req
  db.preface =

    ## Introductory matter preceding the first chapter of a book
    element preface {
      db.preface.attlist, db.preface.info, db.component.contentmodel?
    }
}
div {
  db.partintro.status.attribute = db.status.attribute
  db.partintro.role.attribute = attribute role { text }
  db.partintro.attlist =
    db.partintro.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.partintro.status.attribute?
  db.partintro.info = db._info
  db.partintro =

    ## An introduction to the contents of a part
    element partintro {
      db.partintro.attlist,
      db.partintro.info,
      db.toplevel.blocks.or.sections?
    }
}
div {
  db.section.status.attribute = db.status.attribute
  db.section.role.attribute = attribute role { text }
  db.section.attlist =
    db.section.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.section.status.attribute?
  db.section.info = db._info.title.req
  db.section =

    ## A recursive section
    element section {
      db.section.attlist,
      db.section.info,
      db.navigation.components*,
      db.recursive.blocks.or.sections?,
      db.navigation.components*
    }
}
div {
  db.simplesect.status.attribute = db.status.attribute
  db.simplesect.role.attribute = attribute role { text }
  db.simplesect.attlist =
    db.simplesect.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.simplesect.status.attribute?
  db.simplesect.info = db._info.title.req
  db.simplesect =

    ## A section of a document with no subdivisions
    element simplesect {
      db.simplesect.attlist, db.simplesect.info, db.all.blocks*
    }
}
div {
  db.legalsection.role.attribute = attribute role { text }
  db.legalsection.attlist =
    db.legalsection.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
  db.legalsection.info = db._info.title.req
  db.legalsection =

    ## A recursive section of legal obligations or requirements.
    element legalsection {
      db.legalsection.attlist,
      db.legalsection.info,
      db.all.blocks*,
      db.legalsection*
    }
}
db.article.components = db.toplevel.sections
db.article.navcomponents =
  db.navigation.components
  | db.acknowledgements
  | db.dedication
  | db.appendix
  | db.colophon
div {
  db.article.status.attribute = db.status.attribute
  db.article.class.enumeration =

    ## A collection of frequently asked questions.
    "faq"
    |
      ## An article in a journal or other periodical.
      "journalarticle"
    |
      ## A description of a product.
      "productsheet"
    |
      ## A specification.
      "specification"
    |
      ## A technical report.
      "techreport"
    |
      ## A white paper.
      "whitepaper"
  db.article.class-enum.attribute =

    ## Identifies the nature of the article
    attribute class { db.article.class.enumeration }
  db.article.class-other.attribute =

    ## Identifies the nature of the non-standard article
    attribute otherclass { xsd:NMTOKEN }
  db.article.class-other.attributes =

    ## Identifies the nature of the article
    attribute class {

      ## Indicates that the identifier is some 'other' kind.
      "other"
    }
    & db.article.class-other.attribute
  db.article.class.attribute =
    db.article.class-enum.attribute | db.article.class-other.attributes
  db.article.role.attribute = attribute role { text }
  db.article.attlist =
    db.article.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.article.status.attribute?
    & db.article.class.attribute?
  db.article.info = db._info.title.req
  db.article =

    ## An article
    element article {
      db.article.attlist,
      db.article.info,
      db.article.navcomponents*,
      ((db.all.blocks+, db.article.components?)
       | db.article.components),
      db.article.navcomponents*
    }
}
db.annotations.attribute =

  ## Identifies one or more annotations that apply to this element
  attribute annotations { text }
div {
  db.annotation.role.attribute = attribute role { text }
  db.annotation.annotates.attribute =

    ## Identifies one ore more elements to which this annotation applies
    attribute annotates { text }
  db.annotation.attlist =
    db.annotation.role.attribute?
    & db.annotation.annotates.attribute?
    & db.common.attributes
  db.annotation.info = db._info.title.only
  db.annotation =

    ## An annotation
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:annotation"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:annotation)"
            "annotation must not occur among the children or descendants of annotation"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element annotation {
      db.annotation.attlist, db.annotation.info, db.all.blocks+
    }
}
db.xlink.extended.type.attribute =

  ## Identifies the XLink extended link type
  [
    s:pattern [
      "\x{a}" ~
      "            "
      s:title [ "XLink extended placement" ]
      "\x{a}" ~
      "            "
      s:rule [
        context = "*[@xlink:type='extended']"
        "\x{a}" ~
        "               "
        s:assert [
          test = "not(parent::*[@xlink:type='extended'])"
          "An XLink extended type element may not occur as the direct child of an XLink
extended type element."
        ]
        "\x{a}" ~
        "            "
      ]
      "\x{a}" ~
      "         "
    ]
  ]
  attribute xlink:type {

    ## An XLink extended link type
    "extended"
  }
db.xlink.locator.type.attribute =

  ## Identifies the XLink locator link type
  [
    s:pattern [
      "\x{a}" ~
      "            "
      s:title [ "XLink locator placement" ]
      "\x{a}" ~
      "            "
      s:rule [
        context = "*[@xlink:type='locator']"
        "\x{a}" ~
        "               "
        s:assert [
          test = "parent::*[@xlink:type='extended']"
          "An XLink locator type element must occur as the direct child of an XLink
extended type element."
        ]
        "\x{a}" ~
        "            "
      ]
      "\x{a}" ~
      "         "
    ]
  ]
  attribute xlink:type {

    ## An XLink locator link type
    "locator"
  }
db.xlink.arc.type.attribute =

  ## Identifies the XLink arc link type
  [
    s:pattern [
      "\x{a}" ~
      "            "
      s:title [ "XLink arc placement" ]
      "\x{a}" ~
      "            "
      s:rule [
        context = "*[@xlink:type='arc']"
        "\x{a}" ~
        "               "
        s:assert [
          test = "parent::*[@xlink:type='extended']"
          "An XLink arc type element must occur as the direct child of an XLink
extended type element."
        ]
        "\x{a}" ~
        "            "
      ]
      "\x{a}" ~
      "         "
    ]
  ]
  attribute xlink:type {

    ## An XLink arc link type
    "arc"
  }
db.xlink.resource.type.attribute =

  ## Identifies the XLink resource link type
  [
    s:pattern [
      "\x{a}" ~
      "            "
      s:title [ "XLink resource placement" ]
      "\x{a}" ~
      "            "
      s:rule [
        context = "*[@xlink:type='resource']"
        "\x{a}" ~
        "               "
        s:assert [
          test = "parent::*[@xlink:type='extended']"
          "An XLink resource type element must occur as the direct child of an XLink
extended type element."
        ]
        "\x{a}" ~
        "            "
      ]
      "\x{a}" ~
      "         "
    ]
  ]
  attribute xlink:type {

    ## An XLink resource link type
    "resource"
  }
db.xlink.title.type.attribute =

  ## Identifies the XLink title link type
  [
    s:pattern [
      "\x{a}" ~
      "            "
      s:title [ "XLink title placement" ]
      "\x{a}" ~
      "            "
      s:rule [
        context = "*[@xlink:type='title']"
        "\x{a}" ~
        "               "
        s:assert [
          test =
            "parent::*[@xlink:type='extended'] or parent::*[@xlink:type='locator'] or
parent::*[@xlink:type='arc']"
          "An XLink title type element must occur as the direct child of an XLink
extended, locator, or arc type element."
        ]
        "\x{a}" ~
        "            "
      ]
      "\x{a}" ~
      "         "
    ]
  ]
  attribute xlink:type {

    ## An XLink title link type
    "title"
  }
db.xlink.extended.link.attributes =
  db.xlink.extended.type.attribute
  & db.xlink.role.attribute?
  & db.xlink.title.attribute?
db.xlink.locator.link.attributes =
  db.xlink.locator.type.attribute
  & db.xlink.href.attribute
  & db.xlink.role.attribute?
  & db.xlink.title.attribute?
  & db.xlink.label.attribute?
db.xlink.arc.link.attributes =
  db.xlink.arc.type.attribute
  & db.xlink.arcrole.attribute?
  & db.xlink.title.attribute?
  & db.xlink.show.attribute?
  & db.xlink.actuate.attribute?
  & db.xlink.from.attribute?
  & db.xlink.to.attribute?
db.xlink.resource.link.attributes =
  db.xlink.resource.type.attribute
  & db.xlink.role.attribute?
  & db.xlink.title.attribute?
  & db.xlink.label.attribute?
db.xlink.title.link.attributes = db.xlink.title.type.attribute
db.xlink.from.attribute =
  ## Specifies the XLink traversal-from
  attribute xlink:from { xsd:NMTOKEN }
db.xlink.label.attribute =

  ## Specifies the XLink label
  attribute xlink:label { xsd:NMTOKEN }
db.xlink.to.attribute =

  ## Specifies the XLink traversal-to
  attribute xlink:to { xsd:NMTOKEN }
div {
  db.extendedlink.role.attribute = attribute role { text }
  db.extendedlink.attlist =
    db.extendedlink.role.attribute?
    & db.common.attributes
    &
      ## Identifies the XLink link type
      [ a:defaultValue = "extended" ]
      attribute xlink:type {

        ## An XLink extended link
        "extended"
      }?
    & db.xlink.role.attribute?
    & db.xlink.title.attribute?
  db.extendedlink =

    ## An XLink extended link
    element extendedlink {
      db.extendedlink.attlist, (db.locator | db.arc | db.link)+
    }
}
div {
  db.locator.role.attribute = attribute role { text }
  db.locator.attlist =
    db.locator.role.attribute?
    & db.common.attributes
    &
      ## Identifies the XLink link type
      [ a:defaultValue = "locator" ]
      attribute xlink:type {

        ## An XLink locator link
        "locator"
      }?
    & db.xlink.href.attribute
    & db.xlink.role.attribute?
    & db.xlink.title.attribute?
    & db.xlink.label.attribute?
  db.locator =

    ## An XLink locator in an extendedlink
    element locator { db.locator.attlist, empty }
}
div {
  db.arc.role.attribute = attribute role { text }
  db.arc.attlist =
    db.arc.role.attribute?
    & db.common.attributes
    &
      ## Identifies the XLink link type
      [ a:defaultValue = "arc" ]
      attribute xlink:type {

        ## An XLink arc link
        "arc"
      }?
    & db.xlink.arcrole.attribute?
    & db.xlink.title.attribute?
    & db.xlink.show.attribute?
    & db.xlink.actuate.attribute?
    & db.xlink.from.attribute?
    & db.xlink.to.attribute?
  db.arc =

    ## An XLink arc in an extendedlink
    element arc { db.arc.attlist, empty }
}
db.sect1.sections =
~~db.sect1.sections = (~~  ((db.sect2+, db.simplesect*) | db.simplesect+)
  | db.refentry+
div {
  db.sect1.status.attribute = db.status.attribute
  db.sect1.role.attribute = attribute role { text }
  db.sect1.attlist =
    db.sect1.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.sect1.status.attribute?
  db.sect1.info = db._info.title.req
  db.sect1 =

    ## A top-level section of document
    element sect1 {
      db.sect1.attlist,
      db.sect1.info,
      db.navigation.components*,
      ((db.all.blocks+, db.sect1.sections?) | db.sect1.sections)?,
      db.navigation.components*
    }
}
db.sect2.sections =
~~db.sect2.sections = (~~  ((db.sect3+, db.simplesect*) | db.simplesect+)
  | db.refentry+
div {
  db.sect2.status.attribute = db.status.attribute
  db.sect2.role.attribute = attribute role { text }
  db.sect2.attlist =
    db.sect2.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.sect2.status.attribute?
  db.sect2.info = db._info.title.req
  db.sect2 =

    ## A subsection within a sect1
    element sect2 {
      db.sect2.attlist,
      db.sect2.info,
      db.navigation.components*,
      ((db.all.blocks+, db.sect2.sections?) | db.sect2.sections)?,
      db.navigation.components*
    }
}
db.sect3.sections =
~~db.sect3.sections = (~~  ((db.sect4+, db.simplesect*) | db.simplesect+)
  | db.refentry+
div {
  db.sect3.status.attribute = db.status.attribute
  db.sect3.role.attribute = attribute role { text }
  db.sect3.attlist =
    db.sect3.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.sect3.status.attribute?
  db.sect3.info = db._info.title.req
  db.sect3 =

    ## A subsection within a sect2
    element sect3 {
      db.sect3.attlist,
      db.sect3.info,
      db.navigation.components*,
      ((db.all.blocks+, db.sect3.sections?) | db.sect3.sections)?,
      db.navigation.components*
    }
}
db.sect4.sections =
~~db.sect4.sections = (~~  ((db.sect5+, db.simplesect*) | db.simplesect+)
  | db.refentry+
div {
  db.sect4.status.attribute = db.status.attribute
  db.sect4.role.attribute = attribute role { text }
  db.sect4.attlist =
    db.sect4.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.sect4.status.attribute?
  db.sect4.info = db._info.title.req
  db.sect4 =

    ## A subsection within a sect3
    element sect4 {
      db.sect4.attlist,
      db.sect4.info,
      db.navigation.components*,
      ((db.all.blocks+, db.sect4.sections?) | db.sect4.sections)?,
      db.navigation.components*
    }
}
db.sect5.sections = db.simplesect+ | db.refentry+
div {
  db.sect5.status.attribute = db.status.attribute
  db.sect5.role.attribute = attribute role { text }
  db.sect5.attlist =
    db.sect5.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.sect5.status.attribute?
  db.sect5.info = db._info.title.req
  db.sect5 =

    ## A subsection within a sect4
    element sect5 {
      db.sect5.attlist,
      db.sect5.info,
      db.navigation.components*,
      ((db.all.blocks+, db.sect5.sections?) | db.sect5.sections)?,
      db.navigation.components*
    }
}
db.toplevel.refsection = db.refsection+ | db.refsect1+
db.secondlevel.refsection = db.refsection+ | db.refsect2+
db.reference.components = db.refentry
div {
  db.reference.status.attribute = db.status.attribute
  db.reference.role.attribute = attribute role { text }
  db.reference.attlist =
    db.reference.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.reference.status.attribute?
    & db.label.attribute?
  db.reference.info = db._info.title.req
  db.reference =

    ## A collection of reference entries
    element reference {
      db.reference.attlist,
      db.reference.info,
      db.partintro?,
      db.reference.components*
    }
}
div {
  db.refentry.status.attribute = db.status.attribute
  db.refentry.role.attribute = attribute role { text }
  db.refentry.attlist =
    db.refentry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.refentry.status.attribute?
    & db.label.attribute?
  db.refentry.info = db._info.title.forbidden
  db.refentry =

    ## A reference page (originally a UNIX man-style reference page)
    element refentry {
      db.refentry.attlist,
      db.indexterm*,
      db.refentry.info,
      db.refmeta?,
      db.refnamediv+,
      db.refsynopsisdiv?,
      db.toplevel.refsection
    }
}
div {
  db.refmeta.role.attribute = attribute role { text }
  db.refmeta.attlist =
    db.refmeta.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.refmeta =

    ## Meta-information for a reference entry
    element refmeta {
      db.refmeta.attlist,
      db.indexterm*,
      db.refentrytitle,
      db.manvolnum?,
      db.refmiscinfo*,
      db.indexterm*
    }
```

```
}
db.refmiscinfo.class.enumeration =

  ## The name of the software product or component to which this topic applies
  "source"
  |
    ## The version of the software product or component to which this topic applies
    "version"
  |
    ## The section title of the reference page (e.g., User Commands)
    "manual"
  |
    ## The section title of the reference page (believed synonymous with "manual" but
in wide use)
    "sectdesc"
  |
    ## The name of the software product or component to which this topic applies (e.g.,
SunOS x.y; believed synonymous with "source" but in wide use)
    "software"
db.refmiscinfo.class-enum.attribute =

  ## Identifies the kind of miscellaneous information
  attribute class { db.refmiscinfo.class.enumeration }?
db.refmiscinfo.class-other.attribute =

  ## Identifies the nature of non-standard miscellaneous information
  attribute otherclass { text }
db.refmiscinfo.class-other.attributes =

  ## Identifies the kind of miscellaneious information
  attribute class {

    ## Indicates that the information is some 'other' kind.
    "other"
  }
  & db.refmiscinfo.class-other.attribute
db.refmiscinfo.class.attribute =
  db.refmiscinfo.class-enum.attribute
  | db.refmiscinfo.class-other.attributes
div {
  db.refmiscinfo.role.attribute = attribute role { text }
  db.refmiscinfo.attlist =
    db.refmiscinfo.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.refmiscinfo.class.attribute?
  db.refmiscinfo =

    ## Meta-information for a reference entry other than the title and volume number
    element refmiscinfo { db.refmiscinfo.attlist, db._text }
}
div {
  db.refnamediv.role.attribute = attribute role { text }
  db.refnamediv.attlist =
    db.refnamediv.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.refnamediv =

    ## The name, purpose, and classification of a reference page
    element refnamediv {
      db.refnamediv.attlist,
      db.refdescriptor?,
      db.refname+,
      db.refpurpose,
      db.refclass*
    }
}
div {
  db.refdescriptor.role.attribute = attribute role { text }
  db.refdescriptor.attlist =
    db.refdescriptor.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.refdescriptor =

    ## A description of the topic of a reference page
    element refdescriptor { db.refdescriptor.attlist, db.all.inlines* }
}
div {
  db.refname.role.attribute = attribute role { text }
  db.refname.attlist =
    db.refname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.refname =

    ## The name of (one of) the subject(s) of a reference page
    element refname { db.refname.attlist, db.all.inlines* }
}
div {
  db.refpurpose.role.attribute = attribute role { text }
  db.refpurpose.attlist =
    db.refpurpose.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.refpurpose =

    ## A short (one sentence) synopsis of the topic of a reference page
    element refpurpose { db.refpurpose.attlist, db.all.inlines* }
}
div {
  db.refclass.role.attribute = attribute role { text }
  db.refclass.attlist =
    db.refclass.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.refclass =

    ## The scope or other indication of applicability of a reference entry
    element refclass { db.refclass.attlist, (text | db.application)* }
}
div {
  db.refsynopsisdiv.role.attribute = attribute role { text }
  db.refsynopsisdiv.attlist =
    db.refsynopsisdiv.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.refsynopsisdiv.info = db._info
  db.refsynopsisdiv =

    ## A syntactic synopsis of the subject of the reference page
    element refsynopsisdiv {
      db.refsynopsisdiv.attlist,
      db.refsynopsisdiv.info,
      ((db.all.blocks+, db.secondlevel.refsection?)
       | db.secondlevel.refsection)
    }
}
div {
  db.refsection.status.attribute = db.status.attribute
  db.refsection.role.attribute = attribute role { text }
  db.refsection.attlist =
    db.refsection.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.refsection.status.attribute?
    & db.label.attribute?
  db.refsection.info = db._info.title.req
  db.refsection =

    ## A recursive section in a refentry
    element refsection {
      db.refsection.attlist,
      db.refsection.info,
      ((db.all.blocks+, db.refsection*) | db.refsection+)
    }
}
db.refsect1.sections = db.refsect2+
div {
  db.refsect1.status.attribute = db.status.attribute
  db.refsect1.role.attribute = attribute role { text }
  db.refsect1.attlist =
    db.refsect1.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.refsect1.status.attribute?
  db.refsect1.info = db._info.title.req
  db.refsect1 =

    ## A major subsection of a reference entry
    element refsect1 {
      db.refsect1.attlist,
      db.refsect1.info,
      ((db.all.blocks+, db.refsect1.sections?) | db.refsect1.sections)
    }
}
db.refsect2.sections = db.refsect3+
div {
  db.refsect2.status.attribute = db.status.attribute
  db.refsect2.role.attribute = attribute role { text }
  db.refsect2.attlist =
    db.refsect2.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.refsect2.status.attribute?
  db.refsect2.info = db._info.title.req
  db.refsect2 =

    ## A subsection of a refsect1
    element refsect2 {
      db.refsect2.attlist,
      db.refsect2.info,
      ((db.all.blocks+, db.refsect2.sections?) | db.refsect2.sections)
    }
}
div {
  db.refsect3.status.attribute = db.status.attribute
  db.refsect3.role.attribute = attribute role { text }
  db.refsect3.attlist =
    db.refsect3.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.refsect3.status.attribute?
  db.refsect3.info = db._info.title.req
  db.refsect3 =

    ## A subsection of a refsect2
    element refsect3 {
      db.refsect3.attlist, db.refsect3.info, db.all.blocks+
    }
}
db.glossary.inlines =
  db.firstterm | db.glossterm | db._firstterm | db._glossterm
db.baseform.attribute =

  ## Specifies the base form of the term, the one that appears in the glossary. This
allows adjectival, plural, and other variations of the term to appear in the element.
The element content is the default base form.
  attribute baseform { text }?
div {
  db.glosslist.role.attribute = attribute role { text }
  db.glosslist.attlist =
    db.glosslist.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.glosslist.info = db._info.title.only
  db.glosslist =

    ## A wrapper for a list of glossary entries
    element glosslist {
      db.glosslist.attlist,
      db.glosslist.info?,
      db.all.blocks*,
      db.glossentry+
    }
}
div {
  db.glossentry.role.attribute = attribute role { text }
  db.glossentry.sortas.attribute =

    ## Specifies the string by which the element's content is to be sorted; if
unspecified, the content is used
    attribute sortas { text }
  db.glossentry.attlist =
    db.glossentry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.glossentry.sortas.attribute?
  db.glossentry =

    ## An entry in a glossary or glosslist
    element glossentry {
      db.glossentry.attlist,
      db.glossterm,
      db.acronym?,
      db.abbrev?,
      db.indexterm*,
      (db.glosssee | db.glossdef+)
    }
}
div {
  db.glossdef.role.attribute = attribute role { text }
  db.glossdef.subject.attribute =

    ## Specifies a list of keywords for the definition
    attribute subject { text }
  db.glossdef.attlist =
    db.glossdef.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.glossdef.subject.attribute?
  db.glossdef =

    ## A definition in a glossentry
    element glossdef {
      db.glossdef.attlist, db.all.blocks+, db.glossseealso*
    }
}
div {
  db.glosssee.role.attribute = attribute role { text }
  db.glosssee.otherterm.attribute =

    ## Identifies the other term
    attribute otherterm { xsd:IDREF }
  db.glosssee.attlist =
    db.glosssee.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.glosssee.otherterm.attribute?
  db.glosssee =

    ## A cross-reference from one glossentry
    ##  to another
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Glosssary 'see' type constraint" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:glosssee[@otherterm]"
          "\x{a}" ~
          "                   "
          s:assert [
            test =
              "local-name(//*[@xml:id=current()/@otherterm]) = 'glossentry' and
namespace-uri(//*[@xml:id=current()/@otherterm]) = 'http://docbook.org/ns/docbook'"
            "@otherterm on glosssee must point to a glossentry."
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element glosssee { db.glosssee.attlist, db.all.inlines* }
}
div {
  db.glossseealso.role.attribute = attribute role { text }
  db.glossseealso.otherterm.attribute =

    ## Identifies the other term
    attribute otherterm { xsd:IDREF }
  db.glossseealso.attlist =
    db.glossseealso.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.glossseealso.otherterm.attribute?
  db.glossseealso =
    ## A cross-reference from one glossentry to another
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Glossary 'seealso' type constraint" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:glossseealso[@otherterm]"
          "\x{a}" ~
          "                  "
          s:assert [
            test =
              "local-name(//*[@xml:id=current()/@otherterm]) = 'glossentry' and
namespace-uri(//*[@xml:id=current()/@otherterm]) = 'http://docbook.org/ns/docbook'"
            "@otherterm on glossseealso must point to a glossentry."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element glossseealso { db.glossseealso.attlist, db.all.inlines* }
}
div {
  db.firstterm.role.attribute = attribute role { text }
  db.firstterm.attlist =
    db.firstterm.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.baseform.attribute
  db.firstterm =

    ## The first occurrence of a term
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Glossary 'firstterm' type constraint" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:firstterm[@linkend]"
          "\x{a}" ~
          "                  "
          s:assert [
            test =
              "local-name(//*[@xml:id=current()/@linkend]) = 'glossentry' and
namespace-uri(//*[@xml:id=current()/@linkend]) = 'http://docbook.org/ns/docbook'"
            "@linkend on firstterm must point to a glossentry."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element firstterm { db.firstterm.attlist, db.all.inlines* }
}
div {
  db._firstterm.role.attribute = attribute role { text }
  db._firstterm.attlist =
    db._firstterm.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.baseform.attribute
  db._firstterm =

    ## The first occurrence of a term, with limited content
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Glossary 'firstterm' type constraint" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:firstterm[@linkend]"
          "\x{a}" ~
          "                   "
          s:assert [
            test =
              "local-name(//*[@xml:id=current()/@linkend]) = 'glossentry' and
namespace-uri(//*[@xml:id=current()/@linkend]) = 'http://docbook.org/ns/docbook'"
            "@linkend on firstterm must point to a glossentry."
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element firstterm { db._firstterm.attlist, db._text }
}
div {
  db.glossterm.role.attribute = attribute role { text }
  db.glossterm.attlist =
    db.glossterm.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.baseform.attribute
  db.glossterm =

    ## A glossary term
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Glossary 'glossterm' type constraint" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:glossterm[@linkend]"
          "\x{a}" ~
          "                   "
          s:assert [
            test =
              "local-name(//*[@xml:id=current()/@linkend]) = 'glossentry' and
namespace-uri(//*[@xml:id=current()/@linkend]) = 'http://docbook.org/ns/docbook'"
            "@linkend on glossterm must point to a glossentry."
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element glossterm { db.glossterm.attlist, db.all.inlines* }
}
div {
  db._glossterm.role.attribute = attribute role { text }
  db._glossterm.attlist =
    db._glossterm.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.baseform.attribute
  db._glossterm =

    ## A glossary term
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Glossary 'glossterm' type constraint" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:glossterm[@linkend]"
          "\x{a}" ~
          "                  "
          s:assert [
            test =
              "local-name(//*[@xml:id=current()/@linkend]) = 'glossentry' and
namespace-uri(//*[@xml:id=current()/@linkend]) = 'http://docbook.org/ns/docbook'"
            "@linkend on glossterm must point to a glossentry."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element glossterm { db._glossterm.attlist, db._text }
}
div {
  db.glossary.status.attribute = db.status.attribute
  db.glossary.role.attribute = attribute role { text }
  db.glossary.attlist =
    db.glossary.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.glossary.status.attribute?
  db.glossary.info = db._info
  db.glossary =

    ## A glossary
    element glossary {
      db.glossary.attlist,
      db.glossary.info,
      db.all.blocks*,
      (db.glossdiv* | db.glossentry*),
      db.bibliography?
    }
}
div {
  db.glossdiv.status.attribute = db.status.attribute
  db.glossdiv.role.attribute = attribute role { text }
  db.glossdiv.attlist =
    db.glossdiv.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.glossdiv.status.attribute?
  db.glossdiv.info = db._info.title.req
  db.glossdiv =

    ## A division in a glossary
    element glossdiv {
      db.glossdiv.attlist,
      db.glossdiv.info,
      db.all.blocks*,
      db.glossentry+
    }
}
div {
  db.termdef.role.attribute = attribute role { text }
  db.termdef.attlist =
    db.termdef.role.attribute?
    & db.glossentry.sortas.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.baseform.attribute
  db.termdef =

    ## An inline definition of a term
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Glossary term definition constraint" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:termdef"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "count(db:firstterm) = 1"
            "A termdef must contain exactly one firstterm"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element termdef { db.termdef.attlist, db.all.inlines* }
}
db.relation.attribute =

  ## Identifies the relationship between the bibliographic elemnts
  attribute relation { text }
div {
  db.biblioentry.role.attribute = attribute role { text }
  db.biblioentry.attlist =
    db.biblioentry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.biblioentry =
    ## A raw entry in a bibliography
    element biblioentry {
      db.biblioentry.attlist, db.bibliographic.elements+
    }
}
div {
  db.bibliomixed.role.attribute = attribute role { text }
  db.bibliomixed.attlist =
    db.bibliomixed.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.bibliomixed =

    ## A cooked entry in a bibliography
    element bibliomixed {
      db.bibliomixed.attlist,
      ((db._text
        | db.honorific
        | db.firstname
        | db.surname
        | db.lineage
        | db.othername
        | db.bibliographic.elements)*
       | (db._text
          | db.honorific
          | db.givenname
          | db.surname
          | db.lineage
          | db.othername
          | db.bibliographic.elements)*)
    }
}
div {
  db.biblioset.relation.attribute = db.relation.attribute
  db.biblioset.role.attribute = attribute role { text }
  db.biblioset.attlist =
    db.biblioset.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.biblioset.relation.attribute?
  db.biblioset =

    ## A raw container for related bibliographic information
    element biblioset {
      db.biblioset.attlist, db.bibliographic.elements+
    }
}
div {
  db.bibliomset.relation.attribute = db.relation.attribute
  db.bibliomset.role.attribute = attribute role { text }
  db.bibliomset.attlist =
    db.bibliomset.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.bibliomset.relation.attribute?
  db.bibliomset =

    ## A cooked container for related bibliographic information
    element bibliomset {
      db.bibliomset.attlist,
      ((db._text
        | db.honorific
        | db.firstname
        | db.surname
        | db.lineage
```

```
         | db.othername
         | db.bibliographic.elements)*
        | (db._text
           | db.honorific
           | db.givenname
           | db.surname
           | db.lineage
           | db.othername
           | db.bibliographic.elements)*)
    }
}
div {
  db.bibliomisc.role.attribute = attribute role { text }
  db.bibliomisc.attlist =
    db.bibliomisc.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.bibliomisc =

    ## Untyped bibliographic information
    element bibliomisc { db.bibliomisc.attlist, db._text }
}
div {
  db.bibliography.status.attrib = db.status.attribute
  db.bibliography.role.attribute = attribute role { text }
  db.bibliography.attlist =
    db.bibliography.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.bibliography.status.attrib?
  db.bibliography.info = db._info
  db.bibliography =

    ## A bibliography
    element bibliography {
      db.bibliography.attlist,
      db.bibliography.info,
      db.all.blocks*,
      (db.bibliodiv+ | (db.biblioentry | db.bibliomixed)+)
    }
}
div {
  db.bibliodiv.status.attrib = db.status.attribute
  db.bibliodiv.role.attribute = attribute role { text }
  db.bibliodiv.attlist =
    db.bibliodiv.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.bibliodiv.status.attrib?
  db.bibliodiv.info = db._info.title.req
  db.bibliodiv =

    ## A section of a bibliography
    element bibliodiv {
      db.bibliodiv.attlist,
      db.bibliodiv.info,
      db.all.blocks*,
      (db.biblioentry | db.bibliomixed)+
    }
}
div {
  db.bibliolist.role.attribute = attribute role { text }
  db.bibliolist.attlist =
    db.bibliolist.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.bibliolist.info = db._info.title.only
  db.bibliolist =

    ## A wrapper for a list of bibliography entries
    element bibliolist {
      db.bibliolist.attlist,
      db.bibliolist.info?,
      db.all.blocks*,
      (db.biblioentry | db.bibliomixed)+
    }
}
div {
  db.biblioref.role.attribute = attribute role { text }
  db.biblioref.xrefstyle.attribute = db.xrefstyle.attribute
  db.biblioref.endterm.attribute = db.endterm.attribute
  db.biblioref.units.attribute =

    ## The units (for example, pages) used to identify the beginning and ending of a
reference.
    attribute units { xsd:token }
  db.biblioref.begin.attribute =

    ## Identifies the beginning of a reference; the location within the work that is
being referenced.
    attribute begin { xsd:token }
  db.biblioref.end.attribute =

    ## Identifies the end of a reference.
    attribute end { xsd:token }
  db.biblioref.attlist =
    db.biblioref.role.attribute?
    & db.common.attributes
    & db.common.req.linking.attributes
    & db.biblioref.xrefstyle.attribute?
    & db.biblioref.endterm.attribute?
    & db.biblioref.units.attribute?
    & db.biblioref.begin.attribute?
    & db.biblioref.end.attribute?
  db.biblioref =

    ## A cross-reference to a bibliographic entry
    element biblioref { db.biblioref.attlist, empty }
}
db.significance.enumeration =

  ## Normal
  "normal"
  |
    ## Preferred
    "preferred"
db.significance.attribute =

  ## Specifies the significance of the term
  attribute significance { db.significance.enumeration }
db.zone.attribute =

  ## Specifies the IDs of the elements to which this term applies
  attribute zone { xsd:IDREFS }
db.indexterm.pagenum.attribute =

  ## Indicates the page on which this index term occurs in some version of the printed
document
```

```
  attribute pagenum { text }
db.scope.enumeration =

  ## All indexes
  "all"
  |
    ## The global index (as for a combined index of a set of books)
    "global"
  |
    ## The local index (the index for this document only)
    "local"
db.scope.attribute =

  ## Specifies the scope of the index term
  attribute scope { db.scope.enumeration }
db.sortas.attribute =

  ## Specifies the string by which the term is to be sorted; if unspecified, the term
content is used
  attribute sortas { text }
db.index.type.attribute =

  ## Specifies the target index for this term
  attribute type { text }
div {
  db.itermset.role.attribute = attribute role { text }
  db.itermset.attlist =
    db.itermset.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.itermset =

    ## A set of index terms in the meta-information of a document
    element itermset { db.itermset.attlist, db.indexterm.singular+ }
}
db.indexterm.contentmodel =
  (db.primary,
   (db.secondary,
    ((db.tertiary, (db.see | db.seealso+)?)?
     | (db.see | db.seealso+)?)?)?)?,
  (db.see | db.seealso+)?
div {
  db.indexterm.singular.role.attribute = attribute role { text }
  db.indexterm.singular.class.attribute =

    ## Identifies the class of index term
    attribute class {

      ## A singular index term
      "singular"
    }
  db.indexterm.singular.attlist =
    db.indexterm.singular.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.significance.attribute?
    & db.zone.attribute?
    & db.indexterm.pagenum.attribute?
    & db.scope.attribute?
    & db.index.type.attribute?
    & db.indexterm.singular.class.attribute?
  db.indexterm.singular =

    ## A wrapper for an indexed term
    element indexterm {
      db.indexterm.singular.attlist, db.indexterm.contentmodel
    }
}
div {
  db.indexterm.startofrange.role.attribute = attribute role { text }
  db.indexterm.startofrange.class.attribute =

    ## Identifies the class of index term
    attribute class {

      ## The start of a range
      "startofrange"
    }
  db.indexterm.startofrange.attlist =
    db.indexterm.startofrange.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.significance.attribute?
    & db.zone.attribute?
    & db.indexterm.pagenum.attribute?
    & db.scope.attribute?
    & db.index.type.attribute?
    & db.indexterm.startofrange.class.attribute
  db.indexterm.startofrange =

    ## A wrapper for an indexed term that covers a range
    element indexterm {
      db.indexterm.startofrange.attlist, db.indexterm.contentmodel
    }
}
div {
  db.indexterm.endofrange.role.attribute = attribute role { text }
  db.indexterm.endofrange.class.attribute =

    ## Identifies the class of index term
    attribute class {

      ## The end of a range
      "endofrange"
    }
  db.indexterm.endofrange.startref.attribute =

    ## Points to the start of the range
    attribute startref { xsd:IDREF }
  db.indexterm.endofrange.attlist =
    db.indexterm.endofrange.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.indexterm.endofrange.class.attribute
    & db.indexterm.endofrange.startref.attribute
  db.indexterm.endofrange =

    ## Identifies the end of a range associated with an indexed term
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Indexterm 'startref' type constraint" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:indexterm[@startref]"
          "\x{a}" ~
          "                    "
          s:assert [
            test =
              "local-name(//*[@xml:id=current()/@startref]) = 'indexterm' and
namespace-uri(//*[@xml:id=current()/@startref]) = 'http://docbook.org/ns/docbook'"
            "@startref on indexterm must point to an indexterm."
          ]
          "\x{a}" ~
          "                  "
          s:assert [
            test =
              "//*[@xml:id=current()/@startref]/@class='startofrange'"
            "@startref on indexterm must point to a startofrange indexterm."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element indexterm { db.indexterm.endofrange.attlist, empty }
}
div {
  db.indexterm =
    db.indexterm.singular
    | db.indexterm.startofrange
    | db.indexterm.endofrange
}
div {
  db.primary.role.attribute = attribute role { text }
  db.primary.attlist =
    db.primary.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.sortas.attribute?
  db.primary =

    ## The primary word or phrase under which an index term should be sorted
    element primary { db.primary.attlist, db.all.inlines* }
}
div {
  db.secondary.role.attribute = attribute role { text }
  db.secondary.attlist =
    db.secondary.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.sortas.attribute?
  db.secondary =

    ## A secondary word or phrase in an index term
    element secondary { db.secondary.attlist, db.all.inlines* }
}
div {
  db.tertiary.role.attribute = attribute role { text }
  db.tertiary.attlist =
    db.tertiary.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.sortas.attribute?
  db.tertiary =

    ## A tertiary word or phrase in an index term
    element tertiary { db.tertiary.attlist, db.all.inlines* }
}
div {
  db.see.role.attribute = attribute role { text }
  db.see.class.attribute =~~ attribute class { "normal" | "under" }~~

    ## Identifies the class of 'see'
    attribute class {

      ## Normal
      "normal"
      |
        ## See 'under'
        "under"
    }
  db.see.attlist =
    db.see.role.attribute?
    & db.see.class.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.see =

    ## Part of an index term directing the reader instead to another entry in the index
    element see { db.see.attlist, db.all.inlines* }
}
div {
  db.seealso.role.attribute = attribute role { text }
  db.seealso.class.attribute =~~ attribute class { "normal" | "under" }~~

    ## Identifies the class of 'seealso'
    attribute class {

      ## Normal
      "normal"
      |
        ## See 'under'
        "under"
    }
  db.seealso.attlist =
    db.seealso.role.attribute?
    & db.seealso.class.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.seealso =

    ## Part of an index term directing the reader also to another entry in the index
    element seealso { db.seealso.attlist, db.all.inlines* }
}
div {
  db.index.status.attribute = db.status.attribute
  db.index.role.attribute = attribute role { text }
  db.index.attlist =
    db.index.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.index.status.attribute?
    & db.index.type.attribute?
  db.index.info = db._info
  # Yes, db.indexdiv* and db.indexentry*; that way an <index/> is valid.
  # Authors can use an empty index to indicate where a generated index should
  # appear.
  db.index =

    ## An index to a book or part of a book
    element index {
      db.index.attlist,
      db.index.info,
      db.all.blocks*,
      (db.indexdiv* | db.indexentry* | db.segmentedlist)
    }
}
div {
  db.setindex.status.attribute = db.status.attribute
  db.setindex.role.attribute = attribute role { text }
  db.setindex.attlist =
    db.setindex.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.setindex.status.attribute?
    & db.index.type.attribute?
  db.setindex.info = db._info
  db.setindex =

    ## An index to a set of books
    element setindex {
      db.setindex.attlist,
      db.setindex.info,
      db.all.blocks*,
      (db.indexdiv* | db.indexentry*)
    }
}
div {
  db.indexdiv.status.attribute = db.status.attribute
  db.indexdiv.role.attribute = attribute role { text }
  db.indexdiv.attlist =
    db.indexdiv.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.indexdiv.status.attribute?
  db.indexdiv.info = db._info.title.req
  db.indexdiv =

    ## A division in an index
    element indexdiv {
      db.indexdiv.attlist,
      db.indexdiv.info,
      db.all.blocks*,
      (db.indexentry+ | db.segmentedlist)
    }
}
div {
  db.indexentry.role.attribute = attribute role { text }
  db.indexentry.attlist =
    db.indexentry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.indexentry =

    ## An entry in an index
    element indexentry {
      db.indexentry.attlist,
      db.primaryie,
      (db.seeie | db.seealsoie)*,
      (db.secondaryie, (db.seeie | db.seealsoie | db.tertiaryie)*)*
    }
}
div {
  db.primaryie.role.attribute = attribute role { text }
  db.primaryie.attlist =
    db.primaryie.role.attribute?
    & db.common.attributes
    & db.linkends.attribute?
  db.primaryie =

    ## A primary term in an index entry, not in the text
    element primaryie { db.primaryie.attlist, db.all.inlines* }
}
div {
  db.secondaryie.role.attribute = attribute role { text }
  db.secondaryie.attlist =
    db.secondaryie.role.attribute?
    & db.common.attributes
    & db.linkends.attribute?
  db.secondaryie =

    ## A secondary term in an index entry, rather than in the text
    element secondaryie { db.secondaryie.attlist, db.all.inlines* }
}
div {
  db.tertiaryie.role.attribute = attribute role { text }
  db.tertiaryie.attlist =
    db.tertiaryie.role.attribute?
    & db.common.attributes
    & db.linkends.attribute?
  db.tertiaryie =

    ## A tertiary term in an index entry, rather than in the text
    element tertiaryie { db.tertiaryie.attlist, db.all.inlines* }
}
div {
  db.seeie.role.attribute = attribute role { text }
  db.seeie.attlist =
    db.seeie.role.attribute?
    & db.common.attributes
    & db.linkend.attribute?
  db.seeie =

    ## A See
    ## entry in an index, rather than in the text
    element seeie { db.seeie.attlist, db.all.inlines* }
}
div {
  db.seealsoie.role.attribute = attribute role { text }
  db.seealsoie.attlist =
    db.seealsoie.role.attribute?
    & db.common.attributes
    & db.linkends.attribute?
  db.seealsoie =

    ## A See also
    ##  entry in an index, rather than in the text
    element seealsoie { db.seealsoie.attlist, db.all.inlines* }
}
db.toc.pagenum.attribute =

  ## Indicates the page on which this element occurs in some version of the printed
document
  attribute pagenum { text }
div {
  db.toc.role.attribute = attribute role { text }
  db.toc.attlist =
    db.toc.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.toc.info = db._info.title.only
  db.toc =

    ## A table of contents
    element toc {
      db.toc.attlist,
      db.toc.info,
      db.all.blocks*,
      (db.tocdiv | db.tocentry)*
    }
}
div {
  db.tocdiv.role.attribute = attribute role { text }
  db.tocdiv.pagenum.attribute = db.toc.pagenum.attribute
  db.tocdiv.attlist =
    db.tocdiv.role.attribute?
    & db.common.attributes
    & db.tocdiv.pagenum.attribute?
    & db.linkend.attribute?
  db.tocdiv.info = db._info
  db.tocdiv =

    ## A division in a table of contents
    element tocdiv {
      db.tocdiv.attlist,
      db.tocdiv.info,
      db.all.blocks*,
      (db.tocdiv | db.tocentry)+
    }
}
div {
  db.tocentry.role.attribute = attribute role { text }
  db.tocentry.pagenum.attribute = db.toc.pagenum.attribute
  db.tocentry.attlist =
    db.tocentry.role.attribute?
    & db.common.attributes
    & db.tocentry.pagenum.attribute?
    & db.linkend.attribute?
  db.tocentry =

    ## A component title in a table of contents
    element tocentry { db.tocentry.attlist, db.all.inlines* }
}
db.task.info = db._info.title.req
div {
  db.task.role.attribute = attribute role { text }
  db.task.attlist =
    db.task.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.task =

    ## A task to be completed
    element task {
      db.task.attlist,
      db.task.info,
      db.tasksummary?,
      db.taskprerequisites?,
      db.procedure+,
      db.example*,
      db.taskrelated?
    }
}
div {
  db.tasksummary.role.attribute = attribute role { text }
  db.tasksummary.attlist =
    db.tasksummary.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.tasksummary.info = db._info.title.only
  db.tasksummary =

    ## A summary of a task
    element tasksummary {
      db.tasksummary.attlist, db.tasksummary.info, db.all.blocks+
    }
}
div {
  db.taskprerequisites.role.attribute = attribute role { text }
  db.taskprerequisites.attlist =
    db.taskprerequisites.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.taskprerequisites.info = db._info.title.only
  db.taskprerequisites =

    ## The prerequisites for a task
    element taskprerequisites {
      db.taskprerequisites.attlist,
      db.taskprerequisites.info,
      db.all.blocks+
    }
}
div {
  db.taskrelated.role.attribute = attribute role { text }
  db.taskrelated.attlist =
    db.taskrelated.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.taskrelated.info = db._info.title.only
  db.taskrelated =

    ## Information related to a task
    element taskrelated {
      db.taskrelated.attlist, db.taskrelated.info, db.all.blocks+
    }
}
db.area.units.enumeration =

  ## Coordinates expressed as a pair of CALS graphic coordinates.
  "calspair"
  |
    ## Coordinates expressed as a line and column.
    "linecolumn"
  |
    ## Coordinates expressed as a pair of lines and columns.
    "linecolumnpair"
  |
    ## Coordinates expressed as a line range.
    "linerange"
db.area.units-enum.attribute =

  ## Identifies the units used in the coords attribute. The default units vary
according to the type of callout specified: calspair
  ##  for graphics and linecolumn
  ##  for line-oriented elements.
  attribute units { db.area.units.enumeration }?
db.area.units-other.attributes =

  ## Indicates that non-standard units are used for this area
  ## . In this case otherunits
```

```
    ##  must be specified.
    attribute units {

      ## Coordinates expressed in some non-standard units.
      "other"
    }?,

    ## Identifies the units used in the coords
    ##  attribute when the units
    ##  attribute is other
    ## . This attribute is forbidden otherwise.
    attribute otherunits { xsd:NMTOKEN }
  db.area.units.attribute =
    db.area.units-enum.attribute | db.area.units-other.attributes
  div {
    db.calloutlist.role.attribute = attribute role { text }
    db.calloutlist.attlist =
      db.calloutlist.role.attribute?
      & db.common.attributes
      & db.common.linking.attributes
    db.calloutlist.info = db._info.title.only
    db.calloutlist =

      ## A list of callout
      ## s
      element calloutlist {
        db.calloutlist.attlist,
        db.calloutlist.info,
        db.all.blocks*,
        db.callout+
      }
  }
  div {
    db.callout.role.attribute = attribute role { text }
    db.callout.arearefs.attribute =

      ## Identifies the areas described by this callout.
      attribute arearefs { xsd:IDREFS }
    db.callout.attlist =
      db.callout.role.attribute?
      & db.common.attributes
      & db.callout.arearefs.attribute
    db.callout =

      ## A called out
      ##  description of a marked area
      element callout { db.callout.attlist, db.all.blocks+ }
  }
  div {
    db.programlistingco.role.attribute = attribute role { text }
    db.programlistingco.attlist =
      db.programlistingco.role.attribute?
      & db.common.attributes
      & db.common.linking.attributes
    db.programlistingco.info = db._info.title.forbidden
    db.programlistingco =

      ## A program listing with associated areas used in callouts
      element programlistingco {
        db.programlistingco.attlist,
        db.programlistingco.info,
        db.areaspec,
        db.programlisting,
        db.calloutlist*
      }
```

```
}
div {
  db.areaspec.role.attribute = attribute role { text }
  db.areaspec.attlist =
    db.areaspec.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.area.units.attribute
  db.areaspec =

    ## A collection of regions in a graphic or code example
    element areaspec { db.areaspec.attlist, (db.area | db.areaset)+ }
}
div {
  db.area.role.attribute = attribute role { text }
  db.area.linkends.attribute =

    ## Point to the callout
    ## s which refer to this area. (This provides bidirectional linking which may be
useful in online presentation.)
    attribute linkends { xsd:IDREFS }
  db.area.label.attribute =

    ## Specifies an identifying number or string that may be used in presentation. The
area label might be drawn on top of the figure, for example, at the position indicated
by the coords attribute.
    attribute label { text }
  db.area.coords.attribute =

    ## Provides the coordinates of the area. The coordinates must be interpreted using
the units
    ##  specified.
    attribute coords { text }
  db.area.attlist =
    db.area.role.attribute?
    & db.common.idreq.attributes
    & db.area.units.attribute
    & (db.area.linkends.attribute | db.xlink.simple.link.attributes)?
    & db.area.label.attribute?
    & db.area.coords.attribute
  db.area =

    ## A region defined for a callout in a graphic or code example
    element area { db.area.attlist, db.alt? }
}
div {
  # The only difference is that xml:id is optional
  db.area.inareaset.attlist =
    db.area.role.attribute?
    & db.common.attributes
    & db.area.units.attribute
    & (db.area.linkends.attribute | db.xlink.simple.link.attributes)?
    & db.area.label.attribute?
    & db.area.coords.attribute
  db.area.inareaset =

    ## A region defined for a callout in a graphic or code example
    element area { db.area.inareaset.attlist, db.alt? }
}
div {
  db.areaset.role.attribute = attribute role { text }
  db.areaset.linkends.attribute = db.linkends.attribute
  db.areaset.label.attribute = db.label.attribute
  db.areaset.attlist =
    db.areaset.role.attribute?
    & db.common.idreq.attributes
    & db.area.units.attribute
    & (db.areaset.linkends.attribute | db.xlink.simple.link.attributes)?
    & db.areaset.label.attribute?
  db.areaset =

    ## A set of related areas in a graphic or code example
    element areaset { db.areaset.attlist, db.area.inareaset+ }
}
div {
  db.screenco.role.attribute = attribute role { text }
  db.screenco.attlist =
    db.screenco.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.screenco.info = db._info.title.forbidden
  db.screenco =

    ## A screen with associated areas used in callouts
    element screenco {
      db.screenco.attlist,
      db.screenco.info,
      db.areaspec,
      db.screen,
      db.calloutlist*
    }
}
div {
  db.imageobjectco.role.attribute = attribute role { text }
  db.imageobjectco.attlist =
    db.imageobjectco.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.imageobjectco.info = db._info.title.forbidden
  db.imageobjectco =

    ## A wrapper for an image object with callouts
    element imageobjectco {
      db.imageobjectco.attlist,
      db.imageobjectco.info,
      db.areaspec,
      db.imageobject+,
      db.calloutlist*
    }
}
div {
  db.co.role.attribute = attribute role { text }
  db.co.linkends.attribute = db.linkends.attribute
  db.co.label.attribute = db.label.attribute
  db.co.attlist =
    db.co.role.attribute?
    & db.common.idreq.attributes
    & db.co.linkends.attribute?
    & db.co.label.attribute?
  db.co =

    ## The location of a callout embedded in text
    element co { db.co.attlist, empty }
}
div {
  db.coref.role.attribute = attribute role { text }
  db.coref.label.attribute = db.label.attribute
  db.coref.attlist =
    db.coref.role.attribute?
    & db.common.attributes
    & db.linkend.attribute
    & db.coref.label.attribute?
  db.coref =

    ## A cross reference to a co
    element coref { db.coref.attlist, empty }
}
div {
  db.productionset.role.attribute = attribute role { text }
  db.productionset.attlist =
    db.productionset.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.productionset.info = db._info.title.only
  db.productionset =

    ## A set of EBNF productions
    element productionset {
      db.productionset.attlist,
      db.productionset.info,
      (db.production | db.productionrecap)+
    }
}
div {
  db.production.role.attribute = attribute role { text }
  db.production.attlist =
    db.production.role.attribute?
    & db.common.idreq.attributes
    & db.common.linking.attributes
  db.production =

    ## A production in a set of EBNF productions
    element production {
      db.production.attlist, db.lhs, db.rhs+, db.constraint*
    }
}
div {
  db.lhs.role.attribute = attribute role { text }
  db.lhs.attlist =
    db.lhs.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.lhs =

    ## The left-hand side of an EBNF production
    element lhs { db.lhs.attlist, text }
}
div {
  db.rhs.role.attribute = attribute role { text }
  db.rhs.attlist =
    db.rhs.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.rhs =

    ## The right-hand side of an EBNF production
    element rhs {
      db.rhs.attlist,
      (text | db.nonterminal | db.lineannotation | db.sbr)*
    }
}
div {
  db.nonterminal.role.attribute = attribute role { text }
  db.nonterminal.def.attribute =
    ## Specifies a URI that points to a production
    ## where the nonterminal
    ##  is defined
    attribute def { xsd:anyURI }
  db.nonterminal.attlist =
    db.nonterminal.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.nonterminal.def.attribute
  db.nonterminal =

    ## A non-terminal in an EBNF production
    element nonterminal { db.nonterminal.attlist, text }
}
div {
  db.constraint.role.attribute = attribute role { text }
  db.constraint.attlist =
    db.constraint.role.attribute?
    & db.common.attributes
    & db.common.req.linking.attributes
  db.constraint =

    ## A constraint in an EBNF production
    element constraint { db.constraint.attlist, empty }
}
div {
  db.productionrecap.role.attribute = attribute role { text }
  db.productionrecap.attlist =
    db.productionrecap.role.attribute?
    & db.common.attributes
    & db.common.req.linking.attributes
  db.productionrecap =

    ## A cross-reference to an EBNF production
    element productionrecap { db.productionrecap.attlist, empty }
}
div {
  db.constraintdef.role.attribute = attribute role { text }
  db.constraintdef.attlist =
    db.constraintdef.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.constraintdef.info = db._info.title.only
  db.constraintdef =

    ## The definition of a constraint in an EBNF production
    element constraintdef {
      db.constraintdef.attlist, db.constraintdef.info, db.all.blocks+
    }
}
db.char.attribute =

  ## Specifies the alignment character when align
  ##  is set to char
  ## .
  attribute char { text }
db.charoff.attribute =

  ## Specifies the percentage of the column's total width that should appear to the
left of the first occurance of the character identified in char
  ##  when align
  ##  is set to char
  ## .
  attribute charoff {
    xsd:decimal { minExclusive = "0" maxExclusive = "100" }
  }
db.frame.attribute =

  ## Specifies how the table is to be framed. Note that there is no way to obtain a
border on only the starting edge (left, in left-to-right writing systems) of the table.
  attribute frame {

    ## Frame all four sides of the table. In some environments with limited control
over table border formatting, such as HTML, this may imply additional borders.
    "all"
    |
      ## Frame only the bottom of the table.
      "bottom"
    |
      ## Place no border on the table. In some environments with limited control over
table border formatting, such as HTML, this may disable other borders as well.
      "none"
    |
      ## Frame the left and right sides of the table.
      "sides"
    |
      ## Frame the top of the table.
      "top"
    |
      ## Frame the top and bottom of the table.
      "topbot"
  }
db.colsep.attribute =

  ## Specifies the presence or absence of the column separator
  attribute colsep {

    ## No column separator rule.
    "0"
    |
      ## Provide a column separator rule on the right
      "1"
  }
db.rowsep.attribute =

  ## Specifies the presence or absence of the row separator
  attribute rowsep {

    ## No row separator rule.
    "0"
    |
      ## Provide a row separator rule below
      "1"
  }
db.orient.attribute =

  ## Specifies the orientation of the table
  attribute orient {

    ## 90 degrees counter-clockwise from the rest of the text flow.
    "land"
    |
      ## The same orientation as the rest of the text flow.
      "port"
  }
db.tabstyle.attribute =

  ## Specifies the table style
  attribute tabstyle { text }
db.rowheader.attribute =
  ## Indicates whether or not the entries in the first column should be considered row
headers
  attribute rowheader {

    ## Indicates that entries in the first column of the table are functionally row
headers (analogous to the way that a thead provides column headers).
    "firstcol"
    |
      ## Indicates that row headers are identified by use of the headers attribute on
entries in the table.
      "headers"
    |
      ## Indicates that entries in the first column have no special significance with
respect to column headers.
      "norowheader"
  }
db.align.attribute =

  ## Specifies the horizontal alignment of text in an entry.
  attribute align {

    ## Centered.
    "center"
    |
      ## Aligned on a particular character.
      "char"
    |
      ## Left and right justified.
      "justify"
    |
      ## Left justified.
      "left"
    |
      ## Right justified.
      "right"
  }
db.valign.attribute =

  ## Specifies the vertical alignment of text in an entry.
  attribute valign {

    ## Aligned on the bottom of the entry.
    "bottom"
    |
      ## Aligned in the middle.
      "middle"
    |
      ## Aligned at the top of the entry.
      "top"
  }
db.specify-col-by-colname.attributes =

  ## Specifies a column specification by name.
  attribute colname { text }
db.specify-col-by-namest.attributes =

  ## Specifies a starting column by name.
  attribute namest { text }
db.specify-span-by-spanspec.attributes =

  ## Specifies a span by name.
  attribute spanname { text }
db.specify-span-directly.attributes =
  ## Specifies a starting column by name.
  attribute namest { text }
  &
    ## Specifies an ending column by name.
    attribute nameend { text }
db.column-spec.attributes =
  db.specify-col-by-colname.attributes
  | db.specify-col-by-namest.attributes
  | db.specify-span-by-spanspec.attributes
  | db.specify-span-directly.attributes
db.colname.attribute =

  ## Provides a name for a column specification.
  attribute colname { text }
db.spanname.attribute =

  ## Provides a name for a span specification.
  attribute spanname { text }
div {
  db.tgroup.role.attribute = attribute role { text }
  db.tgroup.tgroupstyle.attribute =

    ## Additional style information for downstream processing; typically the name of a
style.
    attribute tgroupstyle { text }
  db.tgroup.cols.attribute =

    ## The number of columns in the table. Must be an integer greater than zero.
    attribute cols { xsd:positiveInteger }
  db.tgroup.attlist =
    db.tgroup.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.char.attribute?
    & db.charoff.attribute?
    & db.tgroup.tgroupstyle.attribute?
    & db.tgroup.cols.attribute
    & db.colsep.attribute?
    & db.rowsep.attribute?
    & db.align.attribute?
  db.tgroup =

    ## A wrapper for the main content of a table, or part of a table
    element tgroup {
      db.tgroup.attlist,
      db.colspec*,
      db.spanspec*,
      db.cals.thead?,
      db.cals.tfoot?,
      db.cals.tbody
    }
}
div {
  db.colspec.role.attribute = attribute role { text }
  db.colspec.colnum.attribute =

    ## The number of the column to which this specification applies. Must be greater
than any preceding column number. Defaults to one more than the number of the preceding
column, if there is one, or one.
    attribute colnum { xsd:positiveInteger }
  db.colspec.colwidth.attribute =

    ## Specifies the width of the column.
    attribute colwidth { text }
  db.colspec.attlist =
    db.colspec.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.colspec.colnum.attribute?
    & db.char.attribute?
    & db.colsep.attribute?
    & db.colspec.colwidth.attribute?
    & db.charoff.attribute?
    & db.colname.attribute?
    & db.rowsep.attribute?
    & db.align.attribute?
    & db.rowheader.attribute?
  db.colspec =

    ## Specifications for a column in a table
    element colspec { db.colspec.attlist, empty }
}
div {
  db.spanspec.role.attribute = attribute role { text }
  db.spanspec.namest.attribute =

    ## Specifies a starting column by name.
    attribute namest { text }
  db.spanspec.nameend.attribute =

    ## Specifies an ending column by name.
    attribute nameend { text }
  db.spanspec.attlist =
    db.spanspec.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.spanname.attribute
    & db.spanspec.namest.attribute
    & db.spanspec.nameend.attribute
    & db.char.attribute?
    & db.colsep.attribute?
    & db.charoff.attribute?
    & db.rowsep.attribute?
    & db.align.attribute?
  db.spanspec =

    ## Formatting information for a spanned column in a table
    element spanspec { db.spanspec.attlist, empty }
}
div {
  db.cals.thead.role.attribute = attribute role { text }
  db.cals.thead.attlist =
    db.cals.thead.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.valign.attribute?
  db.cals.thead =

    ## A table header consisting of one or more rows
    element thead { db.cals.thead.attlist, db.colspec*, db.row+ }
}
div {
  db.cals.tfoot.role.attribute = attribute role { text }
  db.cals.tfoot.attlist =
    db.cals.tfoot.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.valign.attribute?
  db.cals.tfoot =
    ## A table footer consisting of one or more rows
    element tfoot { db.cals.tfoot.attlist, db.colspec*, db.row+ }
}
div {
  db.cals.tbody.role.attribute = attribute role { text }
  db.cals.tbody.attlist =
    db.cals.tbody.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.valign.attribute?
  db.cals.tbody =

    ## A wrapper for the rows of a table or informal table
    element tbody { db.cals.tbody.attlist, db.row+ }
}
div {
  db.row.role.attribute = attribute role { text }
  db.row.attlist =
    db.row.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.rowsep.attribute?
    & db.valign.attribute?
  db.row =

    ## A row in a table
    element row { db.row.attlist, (db.entry | db.entrytbl)+ }
}
div {
  db.entry.role.attribute = attribute role { text }
  db.entry.morerows.attribute =

    ## Specifies the number of additional rows which this entry occupies. Defaults to
zero.
    attribute morerows { xsd:integer }
  db.entry.rotate.attribute =

    ## Specifies the rotation of this entry. A value of 1 (true) rotates the cell 90
degrees counter-clockwise. A value of 0 (false) leaves the cell unrotated.
    attribute rotate {

      ## Do not rotate the cell.
      "0"
      |
        ## Rotate the cell 90 degrees counter-clockwise.
        "1"
    }
  db.entry.scope.attribute =

    ## Specifies the scope of a header.
    attribute scope {

      ## Applies to the row
      "row"
      |
        ## Applies to the column
        "col"
      |
        ## Applies to the row group
        "rowgroup"
      |
        ## Applies to the column group
        "colgroup"
    }
  db.entry.headers.attribute =
    ## Specifies the entry or entries which serve as headers for this element.
    attribute headers { xsd:IDREFS }
  db.entry.attlist =
    db.entry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.valign.attribute?
    & db.char.attribute?
    & db.colsep.attribute?
    & db.charoff.attribute?
    & db.entry.morerows.attribute?
    & db.column-spec.attributes?
    & db.rowsep.attribute?
    & db.entry.rotate.attribute?
    & db.align.attribute?
    & db.entry.scope.attribute?
    & db.entry.headers.attribute?
  db.entry =

    ## A cell in a table
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:entry"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:table)"
            "table must not occur among the children or descendants of entry"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:entry"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:informaltable)"
            "informaltable must not occur among the children or descendants of entry"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element entry {
      db.entry.attlist, (db.all.inlines* | db.all.blocks*)
    }
}
div {
  db.entrytbl.role.attribute = attribute role { text }
  db.entrytbl.tgroupstyle.attribute =

    ## Additional style information for downstream processing; typically the name of a
style.
    attribute tgroupstyle { text }
  db.entrytbl.cols.attribute =

    ## The number of columns in the entry table. Must be an integer greater than zero.
    attribute cols { xsd:positiveInteger }
  db.entrytbl.attlist =
    db.entrytbl.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.char.attribute?
    & db.charoff.attribute?
    & db.column-spec.attributes?
    & db.entrytbl.tgroupstyle.attribute?
    & db.entrytbl.cols.attribute?
    & db.colsep.attribute?
    & db.rowsep.attribute?
    & db.align.attribute?
  db.entrytbl =

    ## A subtable appearing in place of an entry in a table
    element entrytbl {
      db.entrytbl.attlist,
      db.colspec*,
      db.spanspec*,
      db.cals.entrytbl.thead?,
      db.cals.entrytbl.tbody
    }
}
div {
  db.cals.entrytbl.thead.role.attribute = attribute role { text }
  db.cals.entrytbl.thead.attlist =
    db.cals.entrytbl.thead.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.valign.attribute?
  db.cals.entrytbl.thead =

    ## A table header consisting of one or more rows
    element thead {
      db.cals.entrytbl.thead.attlist, db.colspec*, db.entrytbl.row+
    }
}
div {
  db.cals.entrytbl.tbody.role.attribute = attribute role { text }
  db.cals.entrytbl.tbody.attlist =
    db.cals.entrytbl.tbody.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.valign.attribute?
  db.cals.entrytbl.tbody =

    ## A wrapper for the rows of a table or informal table
    element tbody { db.cals.entrytbl.tbody.attlist, db.entrytbl.row+ }
}
div {
  db.entrytbl.row.role.attribute = attribute role { text }
  db.entrytbl.row.attlist =
    db.entrytbl.row.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.rowsep.attribute?
    & db.valign.attribute?
  db.entrytbl.row =

    ## A row in a table
    element row { db.entrytbl.row.attlist, db.entry+ }
}
div {
  db.cals.table.role.attribute = attribute role { text }
  db.cals.table.label.attribute = db.label.attribute
  db.cals.table.attlist =
    db.cals.table.role.attribute?
    & db.cals.table.label.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.tabstyle.attribute?
    & db.floatstyle.attribute?
    & db.orient.attribute?
    & db.colsep.attribute?
    & db.rowsep.attribute?
    & db.frame.attribute?
    & db.pgwide.attribute?
    &
      ## Indicates if the short or long title should be used in a List of Tables
      attribute shortentry {

        ## Indicates that the full title should be used.
        "0"
        |
          ## Indicates that the short short title (titleabbrev) should be used.
          "1"
      }?
    &
      ## Indicates if the table should appear in a List of Tables
      attribute tocentry {

        ## Indicates that the table should not occur in the List of Tables.
        "0"
        |
          ## Indicates that the table should appear in the List of Tables.
          "1"
      }?
    & db.rowheader.attribute?
  db.cals.table.info = db._info.title.onlyreq
  db.cals.table =

    ## A formal table in a document
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:table"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:example)"
            "example must not occur among the children or descendants of table"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:table"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:figure)"
            "figure must not occur among the children or descendants of table"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:table"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:equation)"
            "equation must not occur among the children or descendants of table"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element table {
      db.cals.table.attlist,
      db.cals.table.info,
      (db.alt? & db.indexing.inlines* & db.textobject*),
      (db.mediaobject+ | db.tgroup+),
      db.caption?
    }
}
div {
  db.cals.informaltable.role.attribute = attribute role { text }
  db.cals.informaltable.attlist =
    db.cals.informaltable.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.tabstyle.attribute?
    & db.floatstyle.attribute?
    & db.orient.attribute?
    & db.colsep.attribute?
    & db.rowsep.attribute?
    & db.frame.attribute?
    & db.pgwide.attribute?
    & db.rowheader.attribute?
  db.cals.informaltable.info = db._info.title.forbidden
  db.cals.informaltable =

    ## A table without a title
    element informaltable {
      db.cals.informaltable.attlist,
      db.cals.informaltable.info,
      (db.alt? & db.indexing.inlines* & db.textobject*),
      (db.mediaobject+ | db.tgroup+),
      db.caption?
    }
}
db.html.coreattrs =

  ## This attribute assigns a class name or set of class names to an element. Any
number of elements may be assigned the same class name or names. Multiple class names
must be separated by white space characters.
  attribute class { text }?
  &
    ## This attribute specifies style information for the current element.
    attribute style { text }?
  &
    ## This attribute offers advisory information about the element for which it is
set.
    attribute title { text }?
db.html.i18n =

  ## This attribute specifies the base language of an element's attribute values and
text content. The default value of this attribute is unknown.
  attribute lang { text }?
db.html.events =

  ## Occurs when the pointing device button is clicked over an element.
  attribute onclick { text }?
  &
    ## Occurs when the pointing device button is double clicked over an element.
    attribute ondblclick { text }?
  &
    ## Occurs when the pointing device button is pressed over an element.
    attribute onmousedown { text }?
  &
    ## Occurs when the pointing device button is released over an element.
    attribute onmouseup { text }?
  &
    ## Occurs when the pointing device is moved onto an element.
    attribute onmouseover { text }?
  &
    ## Occurs when the pointing device is moved while it is over an element.
    attribute onmousemove { text }?
  &
    ## Occurs when the pointing device is moved away from an element.
    attribute onmouseout { text }?
  &
    ## Occurs when a key is pressed and released over an element.
    attribute onkeypress { text }?
  &
    ## Occurs when a key is pressed down over an element.
    attribute onkeydown { text }?
  &
    ## Occurs when a key is released over an element.
    attribute onkeyup { text }?
```

```
db.html.attrs =
  db.common.attributes
  & db.html.coreattrs
  & db.html.i18n
  & db.html.events
db.html.cellhalign =

  ## Specifies the alignment of data and the justification of text in a cell.
  attribute align {

    ## Left-flush data/Left-justify text. This is the default value for table data.
    "left"
    |
      ## Center data/Center-justify text. This is the default value for table headers.
      "center"
    |
      ## Right-flush data/Right-justify text.
      "right"
    |
      ## Double-justify text.
      "justify"
    |
      ## Align text around a specific character. If a user agent doesn't support
character alignment, behavior in the presence of this value is unspecified.
      "char"
  }?
  &
    ## This attribute specifies a single character within a text fragment to act as an
axis for alignment. The default value for this attribute is the decimal point character
for the current language as set by the lang attribute (e.g., the period in English and
the comma in French). User agents are not required to support this attribute.
    attribute char { text }?
  &
    ## When present, this attribute specifies the offset to the first occurrence of the
alignment character on each line. If a line doesn't include the alignment character, it
should be horizontally shifted to end at the alignment position. When charoff is used
to set the offset of an alignment character, the direction of offset is determined by
the current text direction (set by the dir attribute). In left-to-right texts (the
default), offset is from the left margin. In right-to-left texts, offset is from the
right margin. User agents are not required to support this attribute.
    attribute charoff {
      xsd:integer >> a:documentation [ "An explicit offset." ]
      | xsd:string { pattern = "[0-9]+%" }
        >> a:documentation [ "A percentage offset." ]
    }?
db.html.cellvalign =

  ## Specifies the vertical position of data within a cell.
  attribute valign {

    ## Cell data is flush with the top of the cell.
    "top"
    |
      ## Cell data is centered vertically within the cell. This is the default value.
      "middle"
    |
      ## Cell data is flush with the bottom of the cell.
      "bottom"
    |
      ## All cells in the same row as a cell whose valign attribute has this value
should have their textual data positioned so that the first text line occurs on a
baseline common to all cells in the row. This constraint does not apply to subsequent
text lines in these cells.
      "baseline"
  }?
```

```
db.html.table.attributes =

  ## Provides a summary of the table's purpose and structure for user agents rendering
to non-visual media such as speech and Braille.
  attribute summary { text }?
  &
    ## Specifies the desired width of the entire table and is intended for visual user
agents. When the value is a percentage value, the value is relative to the user agent's
available horizontal space. In the absence of any width specification, table width is
determined by the user agent.
    attribute width {
      xsd:integer >> a:documentation [ "An explicit width." ]
      | xsd:string { pattern = "[0-9]+%" }
        >> a:documentation [ "A percentage width." ]
    }?
  &
    ## Specifies the width (in pixels only) of the frame around a table.
    attribute border { xsd:nonNegativeInteger }?
  &
    ## Specifies which sides of the frame surrounding a table will be visible.
    attribute frame {

      ## No sides. This is the default value.
      "void"
      |
        ## The top side only.
        "above"
      |
        ## The bottom side only.
        "below"
      |
        ## The top and bottom sides only.
        "hsides"
      |
        ## The left-hand side only.
        "lhs"
      |
        ## The right-hand side only.
        "rhs"
      |
        ## The right and left sides only.
        "vsides"
      |
        ## All four sides.
        "box"
      |
        ## All four sides.
        "border"
    }?
  &
    ## Specifies which rules will appear between cells within a table. The rendering of
rules is user agent dependent.
    attribute rules {

      ## No rules. This is the default value.
      "none"
      |
        ## Rules will appear between row groups (see thead, tfoot, and tbody) and
column groups (see colgroup and col) only.
        "groups"
      |
        ## Rules will appear between rows only.
        "rows"
      |
        ## Rules will appear between columns only.
        "cols"
      |
        ## Rules will appear between all rows and columns.
        "all"
    }?
  &
    ## Specifies how much space the user agent should leave between the left side of
the table and the left-hand side of the leftmost column, the top of the table and the
top side of the topmost row, and so on for the right and bottom of the table. The
attribute also specifies the amount of space to leave between cells.
    attribute cellspacing {
      xsd:integer >> a:documentation [ "An explicit spacing." ]
      | xsd:string { pattern = "[0-9]+%" }
        >> a:documentation [ "A percentage spacing." ]
    }?
  &
    ## Specifies the amount of space between the border of the cell and its contents.
If the value of this attribute is a pixel length, all four margins should be this
distance from the contents. If the value of the attribute is a percentage length, the
top and bottom margins should be equally separated from the content based on a
percentage of the available vertical space, and the left and right margins should be
equally separated from the content based on a percentage of the available horizontal
space.
    attribute cellpadding {
      xsd:integer >> a:documentation [ "An explicit padding." ]
      | xsd:string { pattern = "[0-9]+%" }
        >> a:documentation [ "A percentage padding." ]
    }?
db.html.tablecell.attributes =

  ## Provides an abbreviated form of the cell's content and may be rendered by user
agents when appropriate in place of the cell's content. Abbreviated names should be
short since user agents may render them repeatedly. For instance, speech synthesizers
may render the abbreviated headers relating to a particular cell before rendering that
cell's content.
  attribute abbr { text }?
  &
    ## This attribute may be used to place a cell into conceptual categories that can
be considered to form axes in an n-dimensional space. User agents may give users access
to these categories (e.g., the user may query the user agent for all cells that belong
to certain categories, the user agent may present a table in the form of a table of
contents, etc.). Please consult an HTML reference for more details.
    attribute axis { text }?
  &
    ## Specifies the list of header cells that provide header information for the
current data cell. The value of this attribute is a space-separated list of cell names;
those cells must be named by setting their id attribute. Authors generally use the
headers attribute to help non-visual user agents render header information about data
cells (e.g., header information is spoken prior to the cell data), but the attribute
may also be used in conjunction with style sheets.
    attribute headers { text }?
  &
    ## Specifies the set of data cells for which the current header cell provides
header information. This attribute may be used in place of the headers attribute,
particularly for simple tables.
    attribute scope {

      ## The current cell provides header information for the rest of the row that
contains it
      "row"
      |
        ## The current cell provides header information for the rest of the column that
contains it.
        "col"
      |
        ## The header cell provides header information for the rest of the row group
that contains it.
        "rowgroup"
      |
        ## The header cell provides header information for the rest of the column group
that contains it.
        "colgroup"
    }?
  &
    ## Specifies the number of rows spanned by the current cell. The default value of
this attribute is one (1
    ## ). The value zero (0
    ## ) means that the cell spans all rows from the current row to the last row of the
table section (thead
    ## , tbody
    ## , or tfoot
    ## ) in which the cell is defined.
    attribute rowspan { xsd:nonNegativeInteger }?
  &
    ## Specifies the number of columns spanned by the current cell. The default value
of this attribute is one (1
    ## ). The value zero (0
    ## ) means that the cell spans all columns from the current column to the last
column of the column group (colgroup
    ## ) in which the cell is defined.
    attribute colspan { xsd:nonNegativeInteger }?
db.html.table.info = db._info.title.forbidden
db.html.table.model =
  db.html.table.info?,
  db.html.caption,
  (db.html.col* | db.html.colgroup*),
  db.html.thead?,
  db.html.tfoot?,
  (db.html.tbody+ | db.html.tr+)
db.html.informaltable.info = db._info.title.forbidden
db.html.informaltable.model =
  db.html.informaltable.info?,
  (db.html.col* | db.html.colgroup*),
  db.html.thead?,
  db.html.tfoot?,
  (db.html.tbody+ | db.html.tr+)
div {
  db.html.table.role.attribute = attribute role { text }
  db.html.table.label.attribute = db.label.attribute
  db.html.table.attlist =
    db.html.attrs
    & db.html.table.attributes
    & db.html.table.role.attribute?
    & db.html.table.label.attribute?
    & db.orient.attribute?
    & db.pgwide.attribute?
    & db.tabstyle.attribute?
    & db.floatstyle.attribute?
  db.html.table =

    ## A formal (captioned) HTML table in a document
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:table"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:example)"
            "example must not occur among the children or descendants of table"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:table"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:figure)"
            "figure must not occur among the children or descendants of table"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:table"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:equation)"
            "equation must not occur among the children or descendants of table"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element table { db.html.table.attlist, db.html.table.model }
}
div {
  db.html.informaltable.role.attribute = attribute role { text }
  db.html.informaltable.label.attribute = db.label.attribute
  db.html.informaltable.attlist =
    db.html.attrs
    & db.html.table.attributes
    & db.html.informaltable.role.attribute?
    & db.html.informaltable.label.attribute?
    & db.orient.attribute?
    & db.pgwide.attribute?
    & db.tabstyle.attribute?
    & db.floatstyle.attribute?
  db.html.informaltable =

    ## An HTML table without a title
    element informaltable {
      db.html.informaltable.attlist, db.html.informaltable.model
    }
}
div {
  db.html.caption.attlist = db.html.attrs
  db.html.caption =

    ## An HTML table caption
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:example)"
            "example must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:figure)"
            "figure must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:table)"
            "table must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:equation)"
            "equation must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:sidebar)"
            "sidebar must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:task)"
            "task must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
```

```
        "                "
      ]
      "\x{a}" ~
      "            "
    ]
    s:pattern [
      "\x{a}" ~
      "               "
      s:title [ "Element exclusion" ]
      "\x{a}" ~
      "               "
      s:rule [
        context = "db:caption"
        "\x{a}" ~
        "                  "
        s:assert [
          test = "not(.//db:caution)"
          "caution must not occur among the children or descendants of caption"
        ]
        "\x{a}" ~
        "                "
      ]
      "\x{a}" ~
      "            "
    ]
    s:pattern [
      "\x{a}" ~
      "               "
      s:title [ "Element exclusion" ]
      "\x{a}" ~
      "               "
      s:rule [
        context = "db:caption"
        "\x{a}" ~
        "                  "
        s:assert [
          test = "not(.//db:danger)"
          "danger must not occur among the children or descendants of caption"
        ]
        "\x{a}" ~
        "                "
      ]
      "\x{a}" ~
      "            "
    ]
    s:pattern [
      "\x{a}" ~
      "               "
      s:title [ "Element exclusion" ]
      "\x{a}" ~
      "               "
      s:rule [
        context = "db:caption"
        "\x{a}" ~
        "                  "
        s:assert [
          test = "not(.//db:important)"
          "important must not occur among the children or descendants of caption"
        ]
        "\x{a}" ~
        "                "
      ]
      "\x{a}" ~
      "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:note)"
            "note must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:tip)"
            "tip must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:caption"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:warning)"
            "warning must not occur among the children or descendants of caption"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element caption { db.html.caption.attlist, db.all.inlines* }
}
div {
  db.html.col.attlist =
    db.html.attrs
    &
      ## This attribute, whose value must be an integer > 0, specifies the number of
columns spanned
      ##  by the col
      ##  element; the col
      ##  element shares its attributes with all the columns it spans. The default
value for this attribute is 1 (i.e., a single column). If the span attribute is set to
N > 1, the current col
      ##  element shares its attributes with the next N-1 columns.
      attribute span { xsd:nonNegativeInteger }?
    &
      ## Specifies a default width for each column spanned by the current col
      ##  element. It has the same meaning as the width
      ##  attribute for the colgroup
      ##  element and overrides it.
      attribute width { text }?
    & db.html.cellhalign
    & db.html.cellvalign
  db.html.col =

    ## Specifications for a column in an HTML table
    element col { db.html.col.attlist, empty }
}
div {
  db.html.colgroup.attlist =
    db.html.attrs
    &
      ## This attribute, which must be an integer > 0, specifies the number of columns
in a column group. In the absence of a span attribute, each colgroup
      ##  defines a column group containing one column. If the span attribute is set to
N > 0, the current colgroup
      ##  element defines a column group containing N columns. User agents must ignore
this attribute if the colgroup
      ##  element contains one or more col
      ##  elements.
      attribute span { xsd:nonNegativeInteger }?
    &
      ## This attribute specifies a default width for each column in the current column
group. In addition to the standard pixel, percentage, and relative values, this
attribute allows the special form 0*
      ##  (zero asterisk) which means that the width of the each column in the group
should be the minimum width necessary to hold the column's contents. This implies that
a column's entire contents must be known before its width may be correctly computed.
Authors should be aware that specifying 0*
      ##  will prevent visual user agents from rendering a table incrementally. This
attribute is overridden for any column in the column group whose width is specified via
a col
      ##  element.
      attribute width { text }?
    & db.html.cellhalign
    & db.html.cellvalign
  db.html.colgroup =

    ## A group of columns in an HTML table
    element colgroup { db.html.colgroup.attlist, db.html.col* }
}
div {
  db.html.thead.attlist =
    db.html.attrs & db.html.cellhalign & db.html.cellvalign
  db.html.thead =

    ## A table header consisting of one or more rows in an HTML table
    element thead { db.html.thead.attlist, db.html.tr+ }
```

```
}
div {
  db.html.tfoot.attlist =
    db.html.attrs & db.html.cellhalign & db.html.cellvalign
  db.html.tfoot =

    ## A table footer consisting of one or more rows in an HTML table
    element tfoot { db.html.tfoot.attlist, db.html.tr+ }
}
div {
  db.html.tbody.attlist =
    db.html.attrs & db.html.cellhalign & db.html.cellvalign
  db.html.tbody =

    ## A wrapper for the rows of an HTML table or informal HTML table
    element tbody { db.html.tbody.attlist, db.html.tr+ }
}
div {
  db.html.tr.attlist =
    db.html.attrs & db.html.cellhalign & db.html.cellvalign
  db.html.tr =

    ## A row in an HTML table
    element tr { db.html.tr.attlist, (db.html.th | db.html.td)+ }
}
div {
  db.html.th.attlist =
    db.html.attrs
    & db.html.tablecell.attributes
    & db.html.cellhalign
    & db.html.cellvalign
  db.html.th =

    ## A table header entry in an HTML table
    element th {
      db.html.th.attlist, (db.all.inlines* | db.all.blocks*)
    }
}
div {
  db.html.td.attlist =
    db.html.attrs
    & db.html.tablecell.attributes
    & db.html.cellhalign
    & db.html.cellvalign
  db.html.td =

    ## A table entry in an HTML table
    element td {
      db.html.td.attlist, (db.all.inlines* | db.all.blocks*)
    }
}
div {
  db.msgset.role.attribute = attribute role { text }
  db.msgset.attlist =
    db.msgset.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msgset.info = db._info.title.only
  db.msgset =

    ## A detailed set of messages, usually error messages
    element msgset {
      db.msgset.attlist,
      db.msgset.info,
      (db.msgentry+ | db.simplemsgentry+)
    }
}
div {
  db.msgentry.role.attribute = attribute role { text }
  db.msgentry.attlist =
    db.msgentry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msgentry =

    ## A wrapper for an entry in a message set
    element msgentry {
      db.msgentry.attlist, db.msg+, db.msginfo?, db.msgexplan*
    }
}
div {
  db.simplemsgentry.role.attribute = attribute role { text }
  db.simplemsgentry.msgaud.attribute =

    ## The audience to which the message relevant
    attribute msgaud { text }
  db.simplemsgentry.msgorig.attribute =

    ## The origin of the message
    attribute msgorig { text }
  db.simplemsgentry.msglevel.attribute =

    ## The level of importance or severity of a message
    attribute msglevel { text }
  db.simplemsgentry.attlist =
    db.simplemsgentry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.simplemsgentry.msgaud.attribute?
    & db.simplemsgentry.msgorig.attribute?
    & db.simplemsgentry.msglevel.attribute?
  db.simplemsgentry =

    ## A wrapper for a simpler entry in a message set
    element simplemsgentry {
      db.simplemsgentry.attlist, db.msgtext, db.msgexplan+
    }
}
div {
  db.msg.role.attribute = attribute role { text }
  db.msg.attlist =
    db.msg.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msg.info = db._info.title.only
  db.msg =

    ## A message in a message set
    element msg {
      db.msg.attlist, db.msg.info, db.msgmain, (db.msgsub | db.msgrel)*
    }
}
div {
  db.msgmain.role.attribute = attribute role { text }
  db.msgmain.attlist =
    db.msgmain.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msgmain.info = db._info.title.only
  db.msgmain =
    ## The primary component of a message in a message set
    element msgmain { db.msgmain.attlist, db.msgmain.info, db.msgtext }
}
div {
  db.msgsub.role.attribute = attribute role { text }
  db.msgsub.attlist =
    db.msgsub.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msgsub.info = db._info.title.only
  db.msgsub =

    ## A subcomponent of a message in a message set
    element msgsub { db.msgsub.attlist, db.msgsub.info, db.msgtext }
}
div {
  db.msgrel.role.attribute = attribute role { text }
  db.msgrel.attlist =
    db.msgrel.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msgrel.info = db._info.title.only
  db.msgrel =

    ## A related component of a message in a message set
    element msgrel { db.msgrel.attlist, db.msgrel.info, db.msgtext }
}
div {
  db.msgtext.role.attribute = attribute role { text }
  db.msgtext.attlist =
    db.msgtext.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msgtext =

    ## The actual text of a message component in a message set
    element msgtext { db.msgtext.attlist, db.all.blocks+ }
}
div {
  db.msginfo.role.attribute = attribute role { text }
  db.msginfo.attlist =
    db.msginfo.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msginfo =

    ## Information about a message in a message set
    element msginfo {
      db.msginfo.attlist, (db.msglevel | db.msgorig | db.msgaud)*
    }
}
div {
  db.msglevel.role.attribute = attribute role { text }
  db.msglevel.attlist =
    db.msglevel.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msglevel =

    ## The level of importance or severity of a message in a message set
    element msglevel { db.msglevel.attlist, db._text }
}
div {
  db.msgorig.role.attribute = attribute role { text }
  db.msgorig.attlist =
    db.msgorig.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msgorig =

    ## The origin of a message in a message set
    element msgorig { db.msgorig.attlist, db._text }
}
div {
  db.msgaud.role.attribute = attribute role { text }
  db.msgaud.attlist =
    db.msgaud.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msgaud =

    ## The audience to which a message in a message set is relevant
    element msgaud { db.msgaud.attlist, db._text }
}
div {
  db.msgexplan.role.attribute = attribute role { text }
  db.msgexplan.attlist =
    db.msgexplan.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.msgexplan.info = db._info.title.only
  db.msgexplan =

    ## Explanatory material relating to a message in a message set
    element msgexplan {
      db.msgexplan.attlist, db.msgexplan.info, db.all.blocks+
    }
}
div {
  db.qandaset.role.attribute = attribute role { text }
  db.qandaset.defaultlabel.enumeration =

    ## No labels
    "none"
    |
      ## Numeric labels
      "number"
    |
      ## "Q:" and "A:" labels
      "qanda"
  db.qandaset.defaultlabel.attribute =

    ## Specifies the default labelling
    attribute defaultlabel { db.qandaset.defaultlabel.enumeration }
  db.qandaset.attlist =
    db.qandaset.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.qandaset.defaultlabel.attribute?
  db.qandaset.info = db._info.title.only
  db.qandaset =

    ## A question-and-answer set
    element qandaset {
      db.qandaset.attlist,
      db.qandaset.info,
      db.all.blocks*,
      (db.qandadiv+ | db.qandaentry+)
    }
```

```
}
div {
  db.qandadiv.role.attribute = attribute role { text }
  db.qandadiv.attlist =
    db.qandadiv.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.qandadiv.info = db._info.title.only
  db.qandadiv =

    ## A titled division in a qandaset
    element qandadiv {
      db.qandadiv.attlist,
      db.qandadiv.info,
      db.all.blocks*,
      (db.qandadiv+ | db.qandaentry+)
    }
}
div {
  db.qandaentry.role.attribute = attribute role { text }
  db.qandaentry.attlist =
    db.qandaentry.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.qandaentry.info = db._info.title.only
  db.qandaentry =

    ## A question/answer set within a qandaset
    element qandaentry {
      db.qandaentry.attlist, db.qandaentry.info, db.question, db.answer*
    }
}
div {
  db.question.role.attribute = attribute role { text }
  db.question.attlist =
    db.question.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.question =

    ## A question in a qandaset
    element question { db.question.attlist, db.label?, db.all.blocks+ }
}
div {
  db.answer.role.attribute = attribute role { text }
  db.answer.attlist =
    db.answer.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.answer =

    ## An answer to a question posed in a qandaset
    element answer { db.answer.attlist, db.label?, db.all.blocks+ }
}
div {
  db.label.role.attribute = attribute role { text }
  db.label.attlist =
    db.label.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.label =

    ## A label on a question or answer
    element label { db.label.attlist, db._text }
}
```

```
db.math.inlines = db.inlineequation
db.equation.content = (db.mediaobject+ | db.mathphrase+) | db._any.mml+
db.inlineequation.content =
  (db.inlinemediaobject+ | db.mathphrase+) | db._any.mml+
div {
  db.equation.role.attribute = attribute role { text }
  db.equation.label.attribute = db.label.attribute
  db.equation.attlist =
    db.equation.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.equation.label.attribute?
    & db.pgwide.attribute?
    & db.floatstyle.attribute?
  db.equation.info = db._info.title.only
  db.equation =

    ## A displayed mathematical equation
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:equation"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:example)"
            "example must not occur among the children or descendants of equation"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:equation"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:figure)"
            "figure must not occur among the children or descendants of equation"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:equation"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:table)"
            "table must not occur among the children or descendants of equation"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:equation"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:equation)"
            "equation must not occur among the children or descendants of equation"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element equation {
      db.equation.attlist,
      db.equation.info,
      db.alt?,
      db.equation.content,
      db.caption?
    }
}
div {
  db.informalequation.role.attribute = attribute role { text }
  db.informalequation.attlist =
    db.informalequation.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.pgwide.attribute?
    & db.floatstyle.attribute?
  db.informalequation.info = db._info.title.forbidden
  db.informalequation =

    ## A displayed mathematical equation without a title
    element informalequation {
      db.informalequation.attlist,
      db.informalequation.info,
      db.alt?,
      db.equation.content,
      db.caption?
    }
}
div {
  db.inlineequation.role.attribute = attribute role { text }
  db.inlineequation.attlist =
    db.inlineequation.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.inlineequation =

    ## A mathematical equation or expression occurring inline
    element inlineequation {
      db.inlineequation.attlist, db.alt?, db.inlineequation.content
    }
}
div {
  db.mathphrase.role.attribute = attribute role { text }
  db.mathphrase.attlist =
    db.mathphrase.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.mathphrase =

    ## A mathematical phrase that can be represented with ordinary text and a small
amount of markup
    element mathphrase {
      db.mathphrase.attlist,
      (db._text | db.ubiq.inlines | db._emphasis)*
    }
}
db.imagedata.mathml.content = db._any.mml
div {
  db.imagedata.mathml.role.attribute = attribute role { text }
  db.imagedata.mathml.attlist =
    db.imagedata.mathml.role.attribute?
    & db.common.attributes
    &
      ## Specifies that the format of the data is MathML
      attribute format {

        ## Specifies MathML.
        "mathml"
      }?
    & db.imagedata.align.attribute?
    & db.imagedata.valign.attribute?
    & db.imagedata.width.attribute?
    & db.imagedata.contentwidth.attribute?
    & db.imagedata.scalefit.attribute?
    & db.imagedata.scale.attribute?
    & db.imagedata.depth.attribute?
    & db.imagedata.contentdepth.attribute?
  db.imagedata.mathml.info = db._info.title.forbidden
  db.imagedata.mathml =

    ## A MathML expression in a media object
    element imagedata {
      db.imagedata.mathml.attlist,
      db.imagedata.mathml.info,
      db.imagedata.mathml.content+
    }
}
div {
  db._any.mml =

    ## Any element from the MathML namespace
    element mml:* { (db._any.attribute | text | db._any)* }
}
db.imagedata.svg.content = db._any.svg
div {
  db.imagedata.svg.role.attribute = attribute role { text }
  db.imagedata.svg.attlist =
    db.imagedata.svg.role.attribute?
    & db.common.attributes
    &
      ## Specifies that the format of the data is SVG
      attribute format {

        ## Specifies SVG.
        "svg"
      }?
    & db.imagedata.align.attribute?
    & db.imagedata.valign.attribute?
    & db.imagedata.width.attribute?
    & db.imagedata.contentwidth.attribute?
    & db.imagedata.scalefit.attribute?
    & db.imagedata.scale.attribute?
    & db.imagedata.depth.attribute?
    & db.imagedata.contentdepth.attribute?
  db.imagedata.svg.info = db._info.title.forbidden
  db.imagedata.svg =

    ## An SVG drawing in a media object
    element imagedata {
      db.imagedata.svg.attlist,
      db.imagedata.svg.info,
      db.imagedata.svg.content+
    }
}
div {
  db._any.svg =

    ## Any element from the SVG namespace
    element svg:* { (db._any.attribute | text | db._any)* }
}
db.markup.inlines =
  db.tag
  | db.markup
  | db.token
  | db.symbol
  | db.literal
  | db.code
  | db.constant
  | db.email
  | db.uri
div {
  db.markup.role.attribute = attribute role { text }
  db.markup.attlist =
    db.markup.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.markup =

    ## A string of formatting markup in text that is to be represented literally
    element markup { db.markup.attlist, db._text }
}
div {
  db.tag.role.attribute = attribute role { text }
  db.tag.class.enumeration =

    ## An attribute
    "attribute"
    |
      ## An attribute value
      "attvalue"
    |
      ## An element
      "element"
    |
      ## An empty element tag
      "emptytag"
    |
      ## An end tag
      "endtag"
    |
      ## A general entity
      "genentity"
    |
      ## The local name part of a qualified name
      "localname"
    |
      ## A namespace
      "namespace"
    |
      ## A numeric character reference
      "numcharref"
    |
      ## A parameter entity
      "paramentity"
    |
      ## A processing instruction
      "pi"
    |
      ## The prefix part of a qualified name
      "prefix"
    |
      ## An SGML comment
      "comment"
    |
      ## A start tag
      "starttag"
    |
      ## An XML processing instruction
      "xmlpi"
  db.tag.class.attribute =

    ## Identifies the nature of the tag content
    attribute class { db.tag.class.enumeration }
  db.tag.namespace.attribute =

    ## Identifies the namespace of the tag content
    attribute namespace { xsd:anyURI }
  db.tag.attlist =
    db.tag.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.tag.class.attribute?
    & db.tag.namespace.attribute?
  db.tag =

    ## A component of XML (or SGML) markup
    element tag { db.tag.attlist, (db._text | db.tag)* }
}
div {
  db.symbol.class.attribute =

    ## Identifies the class of symbol
    attribute class {

      ## The value is a limit of some kind
      "limit"
    }
  db.symbol.role.attribute = attribute role { text }
  db.symbol.attlist =
    db.symbol.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.symbol.class.attribute?
  db.symbol =

    ## A name that is replaced by a value before processing
    element symbol { db.symbol.attlist, db._text }
}
div {
  db.token.role.attribute = attribute role { text }
  db.token.attlist =
    db.token.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.token =

    ## A unit of information
    element token { db.token.attlist, db._text }
}
div {
  db.literal.role.attribute = attribute role { text }
  db.literal.attlist =
    db.literal.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.literal =

    ## Inline text that is some literal value
    element literal { db.literal.attlist, db._text }
}
div {
  code.language.attribute =

    ## Identifies the (computer) language of the code fragment
    attribute language { text }
  db.code.role.attribute = attribute role { text }
  db.code.attlist =
    db.code.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & code.language.attribute?
  db.code =

    ## An inline code fragment
    element code {
      db.code.attlist, (db.programming.inlines | db._text)*
    }
}
div {
  db.constant.class.attribute =

    ## Identifies the class of constant
    attribute class {

      ## The value is a limit of some kind
      "limit"
    }
  db.constant.role.attribute = attribute role { text }
  db.constant.attlist =
    db.constant.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.constant.class.attribute?
  db.constant =

    ## A programming or system constant
    element constant { db.constant.attlist, db._text }
}
div {
  db.productname.role.attribute = attribute role { text }
  db.productname.class.enumeration =

    ## A name with a copyright
    "copyright"
    |
      ## A name with a registered copyright
      "registered"
    |
      ## A name of a service
      "service"
    |
      ## A name which is trademarked
      "trade"
  db.productname.class.attribute =

    ## Specifies the class of product name
    attribute class { db.productname.class.enumeration }
  db.productname.attlist =
    db.productname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.productname.class.attribute?
  db.productname =

    ## The formal name of a product
    element productname { db.productname.attlist, db._text }
}
div {
  db.productnumber.role.attribute = attribute role { text }
  db.productnumber.attlist =
    db.productnumber.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.productnumber =

    ## A number assigned to a product
    element productnumber { db.productnumber.attlist, db._text }
}
div {
  db.database.class.enumeration =

    ## An alternate or secondary key
    "altkey"
    |
      ## A constraint
      "constraint"
    |
      ## A data type
      "datatype"
    |
      ## A field
      "field"
    |
      ## A foreign key
      "foreignkey"
    |
      ## A group
      "group"
    |
      ## An index
      "index"
    |
      ## The first or primary key
      "key1"
    |
      ## An alternate or secondary key
      "key2"
    |
      ## A name
      "name"
    |
      ## The primary key
      "primarykey"
    |
      ## A (stored) procedure
      "procedure"
    |
      ## A record
      "record"
    |
      ## A rule
      "rule"
    |
      ## The secondary key
      "secondarykey"
    |
      ## A table
      "table"
    |
      ## A user
      "user"
    |
      ## A view
      "view"
  db.database.class.attribute =

    ## Identifies the class of database artifact
    attribute class { db.database.class.enumeration }
  db.database.role.attribute = attribute role { text }
  db.database.attlist =
    db.database.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.database.class.attribute?
  db.database =

    ## The name of a database, or part of a database
    element database { db.database.attlist, db._text }
}
div {
  db.application.class.enumeration =

    ## A hardware application
    "hardware"
    |
      ## A software application
      "software"
  db.application.class.attribute =

    ## Identifies the class of application
    attribute class { db.application.class.enumeration }
  db.application.role.attribute = attribute role { text }
  db.application.attlist =
    db.application.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.application.class.attribute?
  db.application =

    ## The name of a software program
    element application { db.application.attlist, db._text }
}
div {
  db.hardware.role.attribute = attribute role { text }
  db.hardware.attlist =
    db.hardware.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.hardware =

    ## A physical part of a computer system
    element hardware { db.hardware.attlist, db._text }
}
db.gui.inlines =
  db.guiicon
  | db.guibutton
  | db.guimenuitem
  | db.guimenu
  | db.guisubmenu
  | db.guilabel
  | db.menuchoice
  | db.mousebutton
div {
  db.guibutton.role.attribute = attribute role { text }
  db.guibutton.attlist =
    db.guibutton.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.guibutton =

    ## The text on a button in a GUI
    element guibutton {
      db.guibutton.attlist,
      (db._text | db.accel | db.superscript | db.subscript)*
    }
}
div {
  db.guiicon.role.attribute = attribute role { text }
  db.guiicon.attlist =
    db.guiicon.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.guiicon =

    ## Graphic and/or text appearing as a icon in a GUI
    element guiicon {
      db.guiicon.attlist,
      (db._text | db.accel | db.superscript | db.subscript)*
    }
}
div {
  db.guilabel.role.attribute = attribute role { text }
  db.guilabel.attlist =
    db.guilabel.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.guilabel =

    ## The text of a label in a GUI
    element guilabel {
      db.guilabel.attlist,
      (db._text | db.accel | db.superscript | db.subscript)*
    }
}
div {
  db.guimenu.role.attribute = attribute role { text }
  db.guimenu.attlist =
    db.guimenu.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.guimenu =

    ## The name of a menu in a GUI
    element guimenu {
      db.guimenu.attlist,
      (db._text | db.accel | db.superscript | db.subscript)*
    }
}
div {
  db.guimenuitem.role.attribute = attribute role { text }
  db.guimenuitem.attlist =
    db.guimenuitem.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.guimenuitem =

    ## The name of a terminal menu item in a GUI
    element guimenuitem {
      db.guimenuitem.attlist,
      (db._text | db.accel | db.superscript | db.subscript)*
    }
}
div {
  db.guisubmenu.role.attribute = attribute role { text }
  db.guisubmenu.attlist =
    db.guisubmenu.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.guisubmenu =

    ## The name of a submenu in a GUI
    element guisubmenu {
      db.guisubmenu.attlist,
      (db._text | db.accel | db.superscript | db.subscript)*
    }
}
div {
  db.menuchoice.role.attribute = attribute role { text }
  db.menuchoice.attlist =
    db.menuchoice.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.menuchoice =

    ## A selection or series of selections from a menu
    element menuchoice {
      db.menuchoice.attlist,
      db.shortcut?,
      (db.guibutton
       | db.guiicon
```

```
        | db.guilabel
        | db.guimenu
        | db.guimenuitem
        | db.guisubmenu)+
    }
}
div {
  db.mousebutton.role.attribute = attribute role { text }
  db.mousebutton.attlist =
    db.mousebutton.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.mousebutton =

    ## The conventional name of a mouse button
    element mousebutton { db.mousebutton.attlist, db._text }
}
db.keyboard.inlines =
  db.keycombo
  | db.keycap
  | db.keycode
  | db.keysym
  | db.shortcut
  | db.accel
div {
  db.keycap.function.enumeration =

    ## The "Alt" key
    "alt"
    |
      ## The "Alt Graph" key
      "altgr"
    |
      ## The "Backspace" key
      "backspace"
    |
      ## The "Command" key
      "command"
    |
      ## The "Control" key
      "control"
    |
      ## The "Delete" key
      "delete"
    |
      ## The down arrow
      "down"
    |
      ## The "End" key
      "end"
    |
      ## The "Enter" key
      "enter"
    |
      ## The "Escape" key
      "escape"
    |
      ## The "Home" key
      "home"
    |
      ## The "Insert" key
      "insert"
    |
      ## The left arrow
      "left"
    |
      ## The "Meta" key
      "meta"
    |
      ## The "Option" key
      "option"
    |
      ## The page down key
      "pagedown"
    |
      ## The page up key
      "pageup"
    |
      ## The right arrow
      "right"
    |
      ## The "Return" key
      "return"
    |
      ## The "Shift" key
      "shift"
    |
      ## The spacebar
      "space"
    |
      ## The "Tab" key
      "tab"
    |
      ## The up arrow
      "up"
  db.keycap.function-enum.attribute =

    ## Identifies the function key
    attribute function { db.keycap.function.enumeration }?
  db.keycap.function-other.attributes =

    ## Identifies the function key
    attribute function {

      ## Indicates a non-standard function key
      "other"
    }?,

    ## Specifies a keyword that identifies the non-standard key
    attribute otherfunction { text }
  db.keycap.function.attrib =
    db.keycap.function-enum.attribute
    | db.keycap.function-other.attributes
  db.keycap.role.attribute = attribute role { text }
  db.keycap.attlist =
    db.keycap.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.keycap.function.attrib
  db.keycap =

    ## The text printed on a key on a keyboard
    element keycap { db.keycap.attlist, db._text }
}
div {
  db.keycode.role.attribute = attribute role { text }
  db.keycode.attlist =
    db.keycode.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.keycode =

    ## The internal, frequently numeric, identifier for a key on a keyboard
    element keycode { db.keycode.attlist, db._text }
}
db.keycombination.contentmodel =
  (db.keycap | db.keycombo | db.keysym) | db.mousebutton
div {
  db.keycombo.action.enumeration =

    ## A (single) mouse click.
    "click"
    |
      ## A double mouse click.
      "double-click"
    |
      ## A mouse or key press.
      "press"
    |
      ## Sequential clicks or presses.
      "seq"
    |
      ## Simultaneous clicks or presses.
      "simul"
  db.keycombo.action-enum.attribute =

    ## Identifies the nature of the action taken. If keycombo
    ##  contains more than one element, simul
    ##  is the default, otherwise there is no default.
    attribute action { db.keycombo.action.enumeration }?
  db.keycombo.action-other.attributes =

    ## Identifies the nature of the action taken
    attribute action {

      ## Indicates a non-standard action
      "other"
    }?,

    ## Identifies the non-standard action in some unspecified way.
    attribute otheraction { text }
  db.keycombo.action.attrib =
    db.keycombo.action-enum.attribute
    | db.keycombo.action-other.attributes
  db.keycombo.role.attribute = attribute role { text }
  db.keycombo.attlist =
    db.keycombo.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.keycombo.action.attrib
  db.keycombo =

    ## A combination of input actions
    element keycombo {
      db.keycombo.attlist, db.keycombination.contentmodel+
    }
}
div {
  db.keysym.role.attribute = attribute role { text }
  db.keysym.attlist =
    db.keysym.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.keysym =
    ## The symbolic name of a key on a keyboard
    element keysym { db.keysym.attlist, db._text }
}
div {
  db.accel.role.attribute = attribute role { text }
  db.accel.attlist =
    db.accel.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.accel =

    ## A graphical user interface (GUI) keyboard shortcut
    element accel { db.accel.attlist, db._text }
}
div {
  db.shortcut.action.attrib = db.keycombo.action.attrib
  db.shortcut.role.attribute = attribute role { text }
  db.shortcut.attlist =
    db.shortcut.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.shortcut.action.attrib
  db.shortcut =

    ## A key combination for an action that is also accessible through a menu
    element shortcut {
      db.shortcut.attlist, db.keycombination.contentmodel+
    }
}
db.os.inlines =
  db.prompt
  | db.envar
  | db.filename
  | db.command
  | db.computeroutput
  | db.userinput
db.computeroutput.inlines =
  (text | db.ubiq.inlines | db.os.inlines | db.technical.inlines)
  | db.co
  | db.markup.inlines
db.userinput.inlines =
  (text | db.ubiq.inlines | db.os.inlines | db.technical.inlines)
  | db.co
  | db.markup.inlines
  | db.gui.inlines
  | db.keyboard.inlines
db.prompt.inlines = db._text | db.co
div {
  db.prompt.role.attribute = attribute role { text }
  db.prompt.attlist =
    db.prompt.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.prompt =

    ## A character or string indicating the start of an input field in a  computer
display
    element prompt { db.prompt.attlist, db.prompt.inlines* }
}
div {
  db.envar.role.attribute = attribute role { text }
  db.envar.attlist =
    db.envar.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.envar =

    ## A software environment variable
    element envar { db.envar.attlist, db._text }
}
div {
  db.filename.class.enumeration =

    ## A device
    "devicefile"
    |
      ## A directory
      "directory"
    |
      ## A filename extension
      "extension"
    |
      ## A header file (as for a programming language)
      "headerfile"
    |
      ## A library file
      "libraryfile"
    |
      ## A partition (as of a hard disk)
      "partition"
    |
      ## A symbolic link
      "symlink"
  db.filename.class.attribute =

    ## Identifies the class of filename
    attribute class { db.filename.class.enumeration }
  db.filename.path.attribute =

    ## Specifies the path of the filename
    attribute path { text }
  db.filename.role.attribute = attribute role { text }
  db.filename.attlist =
    db.filename.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.filename.path.attribute?
    & db.filename.class.attribute?
  db.filename =

    ## The name of a file
    element filename { db.filename.attlist, db._text }
}
div {
  db.command.role.attribute = attribute role { text }
  db.command.attlist =
    db.command.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.command =

    ## The name of an executable program or other software command
    element command { db.command.attlist, db._text }
}
div {
  db.computeroutput.role.attribute = attribute role { text }
  db.computeroutput.attlist =
    db.computeroutput.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.computeroutput =

    ## Data, generally text, displayed or presented by a computer
    element computeroutput {
      db.computeroutput.attlist, db.computeroutput.inlines*
    }
}
div {
  db.userinput.role.attribute = attribute role { text }
  db.userinput.attlist =
    db.userinput.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.userinput =

    ## Data entered by the user
    element userinput { db.userinput.attlist, db.userinput.inlines* }
}
div {
  db.cmdsynopsis.role.attribute = attribute role { text }
  db.cmdsynopsis.sepchar.attribute =

    ## Specifies the character that should separate the command and its top-level
arguments
    attribute sepchar { text }
  db.cmdsynopsis.cmdlength.attribute =

    ## Indicates the displayed length of the command; this information may be used to
intelligently indent command synopses which extend beyond one line
    attribute cmdlength { text }
  db.cmdsynopsis.label.attribute = db.label.attribute
  db.cmdsynopsis.attlist =
    db.cmdsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.cmdsynopsis.sepchar.attribute?
    & db.cmdsynopsis.cmdlength.attribute?
    & db.cmdsynopsis.label.attribute?
  db.cmdsynopsis.info = db._info.title.forbidden
  db.cmdsynopsis =

    ## A syntax summary for a software command
    element cmdsynopsis {
      db.cmdsynopsis.attlist,
      db.cmdsynopsis.info,
      (db.command | db.arg | db.group | db.sbr)+,
      db.synopfragment*
    }
}
db.rep.enumeration =

  ## Can not be repeated.
  "norepeat"
  |
    ## Can be repeated.
    "repeat"
db.rep.attribute =

  ## Indicates whether or not repetition is possible.
  [ a:defaultValue = "norepeat" ] attribute rep { db.rep.enumeration }
db.choice.enumeration =

  ## Formatted to indicate that it is optional.
  "opt"
  |
    ## Formatted without indication.
    "plain"
  |
    ## Formatted to indicate that it is required.
    "req"
db.choice.opt.attribute =

  ## Indicates optionality.
  [ a:defaultValue = "opt" ] attribute choice { db.choice.enumeration }
db.choice.req.attribute =

  ## Indicates optionality.
  [ a:defaultValue = "req" ] attribute choice { db.choice.enumeration }
div {
  db.arg.role.attribute = attribute role { text }
  db.arg.rep.attribute = db.rep.attribute
  db.arg.choice.attribute = db.choice.opt.attribute
  db.arg.attlist =
    db.arg.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.arg.rep.attribute?
    & db.arg.choice.attribute?
  db.arg =

    ## An argument in a cmdsynopsis
    element arg {
      db.arg.attlist,
      (db._text
       | db.arg
       | db.group
       | db.option
       | db.synopfragmentref
       | db.sbr)*
    }
}
div {
  db.group.role.attribute = attribute role { text }
  db.group.rep.attribute = db.rep.attribute
  db.group.choice.attribute = db.choice.opt.attribute
  db.group.attlist =
    db.group.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.group.rep.attribute?
    & db.group.choice.attribute?
  db.group =

    ## A group of elements in a cmdsynopsis
    element group {
      db.group.attlist,
      (db.arg
       | db.group
       | db.option
       | db.synopfragmentref
       | db.replaceable
       | db.sbr)+
    }
}
div {
  db.sbr.role.attribute = attribute role { text }
  db.sbr.attlist = db.sbr.role.attribute? & db.common.attributes
  db.sbr =

    ## An explicit line break in a command synopsis
    element sbr { db.sbr.attlist, empty }
}
div {
  db.synopfragment.role.attribute = attribute role { text }
  db.synopfragment.attlist =
    db.synopfragment.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.synopfragment =

    ## A portion of a cmdsynopsis broken out from the main body of the synopsis
    element synopfragment {
      db.synopfragment.attlist, (db.arg | db.group)+
    }
}
div {
  db.synopfragmentref.role.attribute = attribute role { text }
  db.synopfragmentref.attlist =
    db.synopfragmentref.role.attribute?
    & db.common.attributes
    & db.linkend.attribute
  db.synopfragmentref =

    ## A reference to a fragment of a command synopsis
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Synopsis fragment type constraint" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:synopfragmentref"
          "\x{a}" ~
          "                   "
          s:assert [
            test =
              "local-name(//*[@xml:id=current()/@linkend]) = 'synopfragment' and
namespace-uri(//*[@xml:id=current()/@linkend]) = 'http://docbook.org/ns/docbook'"
            "@linkend on synopfragmentref must point to a synopfragment."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element synopfragmentref { db.synopfragmentref.attlist, text }
}
db.programming.inlines =
  db.function
  | db.parameter
  | db.varname
  | db.returnvalue
  | db.type
  | db.classname
  | db.exceptionname
  | db.interfacename
  | db.methodname
  | db.modifier
  | db.initializer
  | db.buildtarget
  | db.oo.inlines
  | db.templateid
  | db.namespace
  | db.namespacename
  | db.macroname
  | db.unionname
  | db.enumname
  | db.enumvalue
  | db.enumidentifier
  | db.typedefname
db.oo.inlines = db.ooclass | db.ooexception | db.oointerface
db.synopsis.blocks =
  (db.funcsynopsis
   | db.classsynopsis
   | db.methodsynopsis
   | db.constructorsynopsis
   | db.destructorsynopsis
   | db.fieldsynopsis~~)~~
   | db.enumsynopsis
   | db.typedefsynopsis
   | db.namespacesynopsis
   | db.macrosynopsis
   | db.unionsynopsis
   | db.enumsynopsis
   | db.typedefsynopsis)
  | db.cmdsynopsis
div {
  db.synopsis.role.attribute = attribute role { text }
  db.synopsis.label.attribute = db.label.attribute
  db.synopsis.attlist =
    db.synopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.verbatim.attributes
    & db.synopsis.label.attribute?
  db.synopsis =

    ## A general-purpose element for representing the syntax of commands or functions
    element synopsis { db.synopsis.attlist, db.verbatim.contentmodel }
}
div {
  db.synopsisinfo.role.attribute = attribute role { text }
  db.synopsisinfo.attlist =
    db.synopsisinfo.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.verbatim.attributes
  db.synopsisinfo =

    ## Information supplementing synopsis
    element synopsisinfo {
      db.synopsisinfo.attlist,
      (db.verbatim.contentmodel
       | (db.programming.inlines?, db.all.blocks*))
    }
}
div {
  db.funcsynopsis.role.attribute = attribute role { text }
  db.funcsynopsis.attlist =
    db.funcsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
  db.funcsynopsis.info = db._info.title.forbidden
  db.funcsynopsis =
    ## The syntax summary for a function definition
    element funcsynopsis {
      db.funcsynopsis.attlist,
      db.funcsynopsis.info,
      (db.funcsynopsisinfo | db.funcprototype)+
    }
}
div {
  db.funcsynopsisinfo.role.attribute = attribute role { text }
  db.funcsynopsisinfo.attlist =
    db.funcsynopsisinfo.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.verbatim.attributes
  db.funcsynopsisinfo =

    ## Information supplementing the funcdefs of a funcsynopsis
    element funcsynopsisinfo {
      db.funcsynopsisinfo.attlist, db.verbatim.contentmodel
    }
}
div {
  db.funcprototype.role.attribute = attribute role { text }
  db.funcprototype.attlist =
    db.funcprototype.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.funcprototype =

    ## The prototype of a function
    element funcprototype {
      db.funcprototype.attlist,
      db.modifier*,
      db.funcdef,
      (db.void
       | db.varargs
       | ((db.paramdef | db.group.paramdef)+, db.varargs?)),
      db.modifier*
    }
}
div {
  db.funcdef.role.attribute = attribute role { text }
  db.funcdef.attlist =
    db.funcdef.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.funcdef =

    ## A function (subroutine) name and its return type
    element funcdef {
      db.funcdef.attlist,~~ (db._text | db.type | db.function)*~~
      (db._text | db.type | db.templateid | db.void | db.function)*
    }
}
div {
  db.function.role.attribute = attribute role { text }
  db.function.attlist =
    db.function.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.function =

    ## The name of a function or subroutine, as in a programming language
    element function { db.function.attlist, db._text }
```

```
}
div {
  db.void.role.attribute = attribute role { text }
  db.void.attlist =
    db.void.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.void =

    ## An empty element in a function synopsis indicating that the function in question
takes no arguments
    element void { db.void.attlist, empty }
}
div {
  db.varargs.role.attribute = attribute role { text }
  db.varargs.attlist =
    db.varargs.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.varargs =

    ## An empty element in a function synopsis indicating a variable number of
arguments
    element varargs { db.varargs.attlist, empty }
}
div {
  db.group.paramdef.role.attribute = attribute role { text }
  db.group.paramdef.choice.attribute = db.choice.opt.attribute
  db.group.paramdef.attlist =
    db.group.paramdef.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.group.paramdef.choice.attribute?
  db.group.paramdef =

    ## A group of parameters
    element group {
      db.group.paramdef.attlist, (db.paramdef | db.group.paramdef)+
    }
}
div {
  db.paramdef.role.attribute = attribute role { text }
  db.paramdef.choice.enumeration =

    ## Formatted to indicate that it is optional.
    "opt"
    |
      ## Formatted to indicate that it is required.
      "req"
  db.paramdef.choice.attribute =

    ## Indicates optionality.
    [ a:defaultValue = "opt" ]
    attribute choice { db.paramdef.choice.enumeration }
  db.paramdef.attlist =
    db.paramdef.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.paramdef.choice.attribute?
  db.paramdef =

    ## Information about a function parameter in a programming language
    element paramdef {
      db.paramdef.attlist,
      (db._text
        | db.initializer
        | db.modifier
        | db.type
        | db.templateid
        | db.parameter
        | db.funcparams)*
    }
}
div {
  db.funcparams.role.attribute = attribute role { text }
  db.funcparams.attlist =
    db.funcparams.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.funcparams =

    ## Parameters for a function referenced through a function pointer in a synopsis
    element funcparams { db.funcparams.attlist, db._text }
}
div {
  db.classsynopsis.role.attribute = attribute role { text }
  db.classsynopsis.class.enumeration =

    ## This is the synopsis of a class
    "class"
    |
      ## This is the synopsis of an interface
      "interface"
  db.classsynopsis.class.attribute =

    ## Specifies the nature of the synopsis
    attribute class { db.classsynopsis.class.enumeration }
  db.classsynopsis.attlist =
    db.classsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
    & db.classsynopsis.class.attribute?
  db.classsynopsis =

    ## The syntax summary for a class definition
    element classsynopsis {
      db.classsynopsis.attlist,
      db.template*,
      (db.oo.inlines | db.classname)+,
      db.template*,
      (db.classsynopsisinfo
        | db.synopsisinfo
        | db.methodsynopsis
        | db.constructorsynopsis
        | db.destructorsynopsis
        | db.fieldsynopsis)*,
      db.recursive.blocks.or.sections?
    }
}
div {
  db.classsynopsisinfo.role.attribute = attribute role { text }
  db.classsynopsisinfo.attlist =
    db.classsynopsisinfo.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.verbatim.attributes
  db.classsynopsisinfo =
    ## Information supplementing the contents of a classsynopsis
    element classsynopsisinfo {
      db.classsynopsisinfo.attlist, db.verbatim.contentmodel
    }
}
div {
  db.ooclass.role.attribute = attribute role { text }
  db.ooclass.attlist =
    db.ooclass.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.ooclass =

    ## A class in an object-oriented programming language
    element ooclass {
      db.ooclass.attlist, (db.package | db.modifier)*, db.classname
    }
}
div {
  db.oointerface.role.attribute = attribute role { text }
  db.oointerface.attlist =
    db.oointerface.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.oointerface =

    ## An interface in an object-oriented programming language
    element oointerface {
      db.oointerface.attlist,
      (db.package | db.modifier)*,
      db.interfacename
    }
}
div {
  db.ooexception.role.attribute = attribute role { text }
  db.ooexception.attlist =
    db.ooexception.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.ooexception =

    ## An exception in an object-oriented programming language
    element ooexception {
      db.ooexception.attlist,
      (db.package | db.modifier)*,
      db.exceptionname
    }
}
db.modifier.xml.space.attribute =

  ## Can be used to indicate that whitespace in the modifier should be preserved (for
multi-line annotations, for example).
  attribute xml:space {

    ## Extra whitespace and line breaks must be preserved.
    [
      # Ideally the definition of xml:space used on modifier would be
      # different from the definition used on the verbatim elements. The
      # verbatim elements forbid the use of xml:space="default" which
      # wouldn't be a problem on modifier. But doing that causes the
      # generated XSD schemas to be broken so I'm just reusing the existing
      # definition for now. It won't be backwards incompatible to fix this
      # problem in the future.
      #    | ## Extra whitespace and line breaks are not preserved.
      #      "default"
    ]
    "preserve"
  }
div {
  db.modifier.role.attribute = attribute role { text }
  db.modifier.attlist =
    db.modifier.xml.space.attribute?
    & db.modifier.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.modifier =

    ## Modifiers in a synopsis
    element modifier { db.modifier.attlist, db._text }
}
div {
  db.interfacename.role.attribute = attribute role { text }
  db.interfacename.attlist =
    db.interfacename.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.interfacename =

    ## The name of an interface
    element interfacename { db.interfacename.attlist, db._text }
}
div {
  db.exceptionname.role.attribute = attribute role { text }
  db.exceptionname.attlist =
    db.exceptionname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.exceptionname =

    ## The name of an exception
    element exceptionname { db.exceptionname.attlist, db._text }
}
div {
  db.fieldsynopsis.role.attribute = attribute role { text }
  db.fieldsynopsis.attlist =
    db.fieldsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
  db.fieldsynopsis =

    ## The name of a field in a class definition
    element fieldsynopsis {
      db.fieldsynopsis.attlist,
      db.modifier*,
      ~~db.type?~~(db.type | db.templateid)*,
      db.varname,
      db.initializer?
    }
}
div {
  db.initializer.role.attribute = attribute role { text }
  db.initializer.attlist =
    db.initializer.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.initializer.inlines = db._text | db.mathphrase | db.markup.inlines
  db.initializer =
    ## The initializer for a fieldsynopsis
    element initializer {
      db.initializer.attlist, db.initializer.inlines*
    }
}
div {
  db.constructorsynopsis.role.attribute = attribute role { text }
  db.constructorsynopsis.attlist =
    db.constructorsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
  db.constructorsynopsis =

    ## A syntax summary for a constructor
    element constructorsynopsis {
      db.constructorsynopsis.attlist,
      db.modifier*,
      db.methodname?,
      ((db.methodparam | db.group.methodparam)+ | db.void?),
      db.exceptionname*
    }
}
div {
  db.destructorsynopsis.role.attribute = attribute role { text }
  db.destructorsynopsis.attlist =
    db.destructorsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
  db.destructorsynopsis =

    ## A syntax summary for a destructor
    element destructorsynopsis {
      db.destructorsynopsis.attlist,
      db.modifier*,
      db.methodname?,
      ((db.methodparam | db.group.methodparam)+ | db.void?),
      db.exceptionname*
    }
}
div {
  db.methodsynopsis.role.attribute = attribute role { text }
  db.methodsynopsis.attlist =
    db.methodsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
  db.methodsynopsis =

    ## A syntax summary for a method
    element methodsynopsis {
      db.methodsynopsis.attlist,
      db.template*,
      db.modifier*,
      (db.type+ | db.templateid+ | db.void)?,
      db.methodname,
      db.template*,
      ((db.methodparam | db.group.methodparam)+ | db.void),
      db.exceptionname*,
      db.modifier*,
      db.template*,
      db.synopsisinfo*,
      db.recursive.blocks.or.sections?
    }
}
div {
  db.methodname.role.attribute = attribute role { text }
  db.methodname.attlist =
    db.methodname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.methodname =

    ## The name of a method
    element methodname { db.methodname.attlist, db._text }
}
div {
  db.methodparam.role.attribute = attribute role { text }
  db.methodparam.rep.attribute = db.rep.attribute
  db.methodparam.choice.attribute = db.choice.req.attribute
  db.methodparam.attlist =
    db.methodparam.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.methodparam.rep.attribute?
    & db.methodparam.choice.attribute?
  db.methodparam =

    ## Parameters to a method
    element methodparam {
      db.methodparam.attlist,
      db.modifier*,
      ~~db.type?~~(db.type | db.templateid)*,
      ((db.modifier*, db.parameter, db.initializer?) | db.funcparams),
      db.modifier*,
      db.recursive.blocks.or.sections?
    }
}
div {
  db.group.methodparam.role.attribute = attribute role { text }
  db.group.methodparam.choice.attribute = db.choice.opt.attribute
  db.group.methodparam.attlist =
    db.group.methodparam.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.group.methodparam.choice.attribute?
  db.group.methodparam =

    ## A group of method parameters
    element group {
      db.group.methodparam.attlist,
      (db.methodparam | db.group.methodparam)+
    }
}
div {
  db.varname.role.attribute = attribute role { text }
  db.varname.attlist =
    db.varname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.varname =

    ## The name of a variable
    element varname { db.varname.attlist, db._text }
}
div {
  db.buildtarget.role.attribute = attribute role { text }
  db.buildtarget.attlist =
    db.buildtarget.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.buildtarget =

    ## Target of a build
    element buildtarget { db.buildtarget.attlist, db._text }
}
div {
  db.returnvalue.role.attribute = attribute role { text }
  db.returnvalue.attlist =
    db.returnvalue.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.returnvalue =

    ## The value returned by a function
    element returnvalue { db.returnvalue.attlist, db._text }
}
div {
  db.type.role.attribute = attribute role { text }
  db.type.class.enumeration =

    ## Combined type is union of nested types
    "union"
    |
      ## Combined type is intersection of nested types
      "intersection"
  db.type.class.attribute =

    ## Specifies the way how are nested types combined together
    attribute class { db.type.class.enumeration }
  db.type.attlist =
    db.type.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.type.class.attribute?
  db.type =

    ## The classification of a value
    element type {~~ db.type.attlist, db._text }~~
      db.type.attlist,
      (db._text | db.type | db.templateid | db.programming.inlines)*
    }
}
div {
  db.classname.role.attribute = attribute role { text }
  db.classname.attlist =
    db.classname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.classname =

    ## The name of a class, in the object-oriented programming sense
    element classname { db.classname.attlist, db._text }
}
div {
  db.templateid.role.attribute = attribute role { text }
  db.templateid.attlist =
    db.templateid.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.templateid =

    ## The identifier for a template, in the generic programming sense
    element templateid { db.templateid.attlist, db._text }
}
div {
  db.template.role.attribute = attribute role { text }
  db.template.attlist =
    db.template.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.template =

    ## The definition of a template, in the generic programming sense
    element template {
      db.template.attlist,
      (((db.modifier | db.type | db._text)*,
        db.templateid,
        (db.modifier | db.type | db._text)*)
       | db.specializedtemplate)
    }
}
div {
  db.specializedtemplate.role.attribute = attribute role { text }
  db.specializedtemplate.attlist =
    db.specializedtemplate.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.specializedtemplate =

    ## The specialization of a template identifier, in the generic programming sense
    element specializedtemplate {
      db.specializedtemplate.attlist,
      (db.modifier | db.type | db._text)*
    }
}
div {
  db.namespace.role.attribute = attribute role { text }
  db.namespace.attlist =
    db.namespace.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.namespace =

    ## The definition of a name space, which may be more than a name
    element namespace {
      db.namespace.attlist, db.modifier*, db.namespacename, db.modifier*
    }
}
div {
  db.namespacename.role.attribute = attribute role { text }
  db.namespacename.attlist =
    db.namespacename.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.namespacename =

    ## The name of a name space
    element namespacename { db.namespacename.attlist, db._text }
}
div {
  db.namespacesynopsis.role.attribute = attribute role { text }
  db.namespacesynopsis.attlist =
    db.namespacesynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
  db.namespacesynopsis =

    ## The syntax summary for a name space definition
    element namespacesynopsis {
      db.namespacesynopsis.attlist,
      (db.namespace | db.namespacename),
      db.recursive.blocks.or.sections?,
      (db.synopsisinfo
       | db.classsynopsis
       | db.funcsynopsis
       | db.fieldsynopsis
       | db.typedefsynopsis
       | db.macrosynopsis
       | db.enumsynopsis
       | db.namespacesynopsis)*,
      db.recursive.blocks.or.sections?
    }
}
div {
  db.macroname.role.attribute = attribute role { text }
  db.macroname.attlist =
    db.macroname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.macroname =

    ## The name of a macro (a code-generating function)
    element macroname { db.macroname.attlist, db._text }
}
div {
  db.macrosynopsis.role.attribute = attribute role { text }
  db.macrosynopsis.attlist =
    db.macrosynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
  db.macrosynopsis =

    ## The syntax summary for a macro definition (code-generating function)
    element macrosynopsis {
      db.macrosynopsis.attlist,
      db.macroname,
      db.synopsisinfo*,
      db.macroprototype+,
      db.synopsisinfo*,
      db.recursive.blocks.or.sections?
    }
}
div {
  db.macroprototype.role.attribute = attribute role { text }
  db.macroprototype.attlist =
    db.macroprototype.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.macroprototype =
    ## The prototype of a macro (code-generating function)
    element macroprototype {
      db.macroprototype.attlist,
      db.modifier*,
      db.macrodef,
      (db.void
       | db.varargs
       | ((db.paramdef | db.group.paramdef)+, db.varargs?)),
      db.modifier*
    }
}
div {
  db.macrodef.role.attribute = attribute role { text }
  db.macrodef.attlist =
    db.macrodef.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.macrodef =

    ## A macro name and its return type
    element macrodef {
      db.macrodef.attlist, (db.type | db.templateid)*, db.macroname
    }
}
div {
  db.unionname.role.attribute = attribute role { text }
  db.unionname.attlist =
    db.unionname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.unionname =

    ## The name of a union of types
    element unionname { db.unionname.attlist, db._text }
}
div {
  db.union.role.attribute = attribute role { text }
  db.union.attlist =
    db.union.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.union =

    ## The definition of a union of types, which may be more than a name
    element union {
      db.union.attlist, db.unionname?, db.programming.inlines*
    }
}
div {
  db.unionsynopsis.role.attribute = attribute role { text }
  db.unionsynopsis.attlist =
    db.unionsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
    & db.enumsynopsis.ordered.attribute?
  db.unionsynopsis =

    ## The syntax summary for a union-of-types definition
    element unionsynopsis {
      db.unionsynopsis.attlist,
      db.unionname?,
      db.union?,
      db.synopsisinfo*,
      db.recursive.blocks.or.sections?
    }
}
div {
  db.enumname.role.attribute = attribute role { text }
  db.enumname.attlist =
    db.enumname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.enumname =

    ## The name of a union of types
    element enumname { db.enumname.attlist, db._text }
}
div {
  db.enumvalue.role.attribute = attribute role { text }
  db.enumvalue.attlist =
    db.enumvalue.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.enumvalue =

    ## The value an enumerated type can take
    element enumvalue { db.enumvalue.attlist, db._text }
}
div {
  db.enumitemdescription.role.attribute = attribute role { text }
  db.enumitemdescription.attlist =
    db.enumitemdescription.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.enumitemdescription =

    ## The description of a value an enumerated type can take
    element enumitemdescription {
      db.enumitemdescription.attlist, db.all.inlines*
    }
}
div {
  db.enumidentifier.role.attribute = attribute role { text }
  db.enumidentifier.attlist =
    db.enumidentifier.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.enumidentifier =

    ## The identifier of a value an enumerated type can take
    element enumidentifier { db.enumidentifier.attlist, db._text }
}
div {
  db.enumitem =

    ## A value an enumerated type can take and its description
    element enumitem {
      db.enumvalue.attlist,
      db.enumidentifier,
      db.enumvalue?,
      db.enumitemdescription?
    }
}
div {
  db.enumsynopsis.role.attribute = attribute role { text }
  db.enumsynopsis.ordered.enumeration =

    ## Value of enum is specified explicitly using enumvalue
    "0"
    |
      ## Value of enum is inferred from its position
      "1"
  db.enumsynopsis.ordered.attribute =
    attribute ordered { db.enumsynopsis.ordered.enumeration }
  db.enumsynopsis.attlist =
    db.enumsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
    & db.enumsynopsis.ordered.attribute?
  db.enumsynopsis =

    ## The syntax summary for an enumerated-type definition
    element enumsynopsis {
      db.enumsynopsis.attlist,
      db.enumname?,
      db.synopsisinfo*,
      db.enumitem+,
      db.recursive.blocks.or.sections?
    }
}
div {
  db.typedefname.role.attribute = attribute role { text }
  db.typedefname.attlist =
    db.typedefname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.typedefname =

    ## The name of a type alias
    element typedefname { db.typedefname.attlist, db._text }
}
div {
  db.typedefsynopsis.role.attribute = attribute role { text }
  db.typedefsynopsis.attlist =
    db.typedefsynopsis.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.language.attribute?
  db.typedefsynopsis =

    ## The syntax summary for a type-alias definition
    element typedefsynopsis {
      db.typedefsynopsis.attlist,
      db.typedefname,
      db.programming.inlines*,
      db.recursive.blocks.or.sections?
    }
}
div {
  db.programlisting.role.attribute = attribute role { text }
  db.programlisting.width.attribute = db.width.characters.attribute
  db.programlisting.attlist =
    db.programlisting.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.verbatim.attributes
    & db.programlisting.width.attribute?
  db.programlisting =

    ## A literal listing of all or part of a program
    element programlisting {
      db.programlisting.attlist, db.verbatim.contentmodel
    }
}
db.admonition.blocks =
  db.caution | db.danger | db.important | db.note | db.tip | db.warning
db.admonition.contentmodel = db._info.title.only, db.all.blocks+
div {
  db.caution.role.attribute = attribute role { text }
  db.caution.attlist =
    db.caution.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.caution =

    ## A note of caution
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caution"
          "\x{a}" ~
          "                    "
          s:assert [
            test = "not(.//db:caution)"
            "caution must not occur among the children or descendants of caution"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caution"
          "\x{a}" ~
          "                    "
          s:assert [
            test = "not(.//db:danger)"
            "danger must not occur among the children or descendants of caution"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caution"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:important)"
            "important must not occur among the children or descendants of caution"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caution"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:note)"
            "note must not occur among the children or descendants of caution"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caution"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:tip)"
            "tip must not occur among the children or descendants of caution"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:caution"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:warning)"
            "warning must not occur among the children or descendants of caution"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element caution { db.caution.attlist, db.admonition.contentmodel }
}
div {
  db.danger.role.attribute = attribute role { text }
  db.danger.attlist =
    db.danger.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.danger =

    ## An admonition set off from the text indicating hazardous situation
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:danger"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:caution)"
            "caution must not occur among the children or descendants of danger"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:danger"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:danger)"
            "danger must not occur among the children or descendants of danger"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:danger"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:important)"
            "important must not occur among the children or descendants of danger"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:danger"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:note)"
            "note must not occur among the children or descendants of danger"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:danger"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:tip)"
```

```
            "tip must not occur among the children or descendants of danger"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:danger"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:warning)"
            "warning must not occur among the children or descendants of danger"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element danger { db.danger.attlist, db.admonition.contentmodel }
}
div {
  db.important.role.attribute = attribute role { text }
  db.important.attlist =
    db.important.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.important =

    ## An admonition set off from the text
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:important"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:caution)"
            "caution must not occur among the children or descendants of important"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:important"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:danger)"
            "danger must not occur among the children or descendants of important"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:important"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:important)"
            "important must not occur among the children or descendants of important"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:important"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:note)"
            "note must not occur among the children or descendants of important"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
```

```
        s:rule [
          context = "db:important"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:tip)"
            "tip must not occur among the children or descendants of important"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:important"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:warning)"
            "warning must not occur among the children or descendants of important"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element important {
      db.important.attlist, db.admonition.contentmodel
    }
}
div {
  db.note.role.attribute = attribute role { text }
  db.note.attlist =
    db.note.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.note =

    ## A message set off from the text
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:note"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:caution)"
            "caution must not occur among the children or descendants of note"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:note"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:danger)"
            "danger must not occur among the children or descendants of note"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:note"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:important)"
            "important must not occur among the children or descendants of note"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:note"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:note)"
            "note must not occur among the children or descendants of note"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
```

```
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:note"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:tip)"
            "tip must not occur among the children or descendants of note"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:note"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:warning)"
            "warning must not occur among the children or descendants of note"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element note { db.note.attlist, db.admonition.contentmodel }
}
div {
  db.tip.role.attribute = attribute role { text }
  db.tip.attlist =
    db.tip.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.tip =

    ## A suggestion to the user, set off from the text
    [
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:tip"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:caution)"
            "caution must not occur among the children or descendants of tip"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:tip"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:danger)"
            "danger must not occur among the children or descendants of tip"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:tip"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:important)"
            "important must not occur among the children or descendants of tip"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:tip"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:note)"
            "note must not occur among the children or descendants of tip"
          ]
          "\x{a}" ~
          "                "
```

```
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:tip"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:tip)"
            "tip must not occur among the children or descendants of tip"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:tip"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(.//db:warning)"
            "warning must not occur among the children or descendants of tip"
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element tip { db.tip.attlist, db.admonition.contentmodel }
}
div {
  db.warning.role.attribute = attribute role { text }
  db.warning.attlist =
    db.warning.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.warning =

    ## An admonition set off from the text
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:warning"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:caution)"
            "caution must not occur among the children or descendants of warning"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:warning"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:danger)"
            "danger must not occur among the children or descendants of warning"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:warning"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:important)"
            "important must not occur among the children or descendants of warning"
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Element exclusion" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:warning"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(.//db:note)"
```

```
          "note must not occur among the children or descendants of warning"
        ]
        "\x{a}" ~
        "                "
      ]
      "\x{a}" ~
      "            "
    ]
    s:pattern [
      "\x{a}" ~
      "                "
      s:title [ "Element exclusion" ]
      "\x{a}" ~
      "                "
      s:rule [
        context = "db:warning"
        "\x{a}" ~
        "                   "
        s:assert [
          test = "not(.//db:tip)"
          "tip must not occur among the children or descendants of warning"
        ]
        "\x{a}" ~
        "                "
      ]
      "\x{a}" ~
      "            "
    ]
    s:pattern [
      "\x{a}" ~
      "                "
      s:title [ "Element exclusion" ]
      "\x{a}" ~
      "                "
      s:rule [
        context = "db:warning"
        "\x{a}" ~
        "                   "
        s:assert [
          test = "not(.//db:warning)"
          "warning must not occur among the children or descendants of warning"
        ]
        "\x{a}" ~
        "                "
      ]
      "\x{a}" ~
      "            "
    ]
  ]
  element warning { db.warning.attlist, db.admonition.contentmodel }
}
db.error.inlines =
  db.errorcode | db.errortext | db.errorname | db.errortype
div {
  db.errorcode.role.attribute = attribute role { text }
  db.errorcode.attlist =
    db.errorcode.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.errorcode =

    ## An error code
    element errorcode { db.errorcode.attlist, db._text }
}
div {
  db.errorname.role.attribute = attribute role { text }
  db.errorname.attlist =
    db.errorname.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.errorname =

    ## An error name
    element errorname { db.errorname.attlist, db._text }
}
div {
  db.errortext.role.attribute = attribute role { text }
  db.errortext.attlist =
    db.errortext.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.errortext =

    ## An error message.
    element errortext { db.errortext.attlist, db._text }
}
div {
  db.errortype.role.attribute = attribute role { text }
  db.errortype.attlist =
    db.errortype.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.errortype =

    ## The classification of an error message
    element errortype { db.errortype.attlist, db._text }
}
db.systemitem.inlines = db._text | db.co
div {
  db.systemitem.class.enumeration =

    ## A daemon or other system process (syslogd)
    "daemon"
    |
      ## A domain name (example.com)
      "domainname"
    |
      ## An ethernet address (00:05:4E:49:FD:8E)
      "etheraddress"
    |
      ## An event of some sort (SIGHUP)
      "event"
    |
      ## An event handler of some sort (hangup)
      "eventhandler"
    |
      ## A filesystem (ext3)
      "filesystem"
    |
      ## A fully qualified domain name (my.example.com)
      "fqdomainname"
    |
      ## A group name (wheel)
      "groupname"
    |
      ## A network interface (eth0)
      "interface"
    |
      ## An IP address (127.0.0.1)
      "ipaddress"
    |
      ## A library (libncurses)
      "library"
    |
      ## A macro
      "macro"
    |
      ## A netmask (255.255.255.192)
      "netmask"
    |
      ## A newsgroup (comp.text.xml)
      "newsgroup"
    |
      ## An operating system name (Hurd)
      "osname"
    |
      ## A process (gnome-cups-icon)
      "process"
    |
      ## A protocol (ftp)
      "protocol"
    |
      ## A resource
      "resource"
    |
      ## A security context (a role, permission, or security token, for example)
      "securitycontext"
    |
      ## A server (mail.example.com)
      "server"
    |
      ## A service (ppp)
      "service"
    |
      ## A system name (hephaistos)
      "systemname"
    |
      ## A user name (ndw)
      "username"
  db.systemitem.class-enum.attribute =

    ## Identifies the nature of the system item
    attribute class { db.systemitem.class.enumeration }?
  db.systemitem.class-other.attribute =

    ## Identifies the nature of the non-standard system item
    attribute otherclass { xsd:NMTOKEN }
  db.systemitem.class-other.attributes =

    ## Identifies the kind of systemitemgraphic identifier
    attribute class {

      ## Indicates that the system item is some 'other' kind.
      "other"
    }
    & db.systemitem.class-other.attribute
  db.systemitem.class.attribute =
    db.systemitem.class-enum.attribute
    | db.systemitem.class-other.attributes
  db.systemitem.role.attribute = attribute role { text }
  db.systemitem.attlist =
    db.systemitem.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.systemitem.class.attribute?
  db.systemitem =

    ## A system-related item or term
    element systemitem { db.systemitem.attlist, db.systemitem.inlines* }
}
div {
  db.option.role.attribute = attribute role { text }
  db.option.attlist =
    db.option.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.option =

    ## An option for a software command
    element option { db.option.attlist, db._text }
}
div {
  db.optional.role.attribute = attribute role { text }
  db.optional.attlist =
    db.optional.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.optional =

    ## Optional information
    element optional { db.optional.attlist, db._text }
}
div {
  db.property.role.attribute = attribute role { text }
  db.property.attlist =
    db.property.role.attribute?
    & db.common.attributes
    & db.common.linking.attributes
  db.property =

    ## A unit of data associated with some part of a computer system
    element property { db.property.attlist, db._text }
}
div {
  db.topic.status.attribute = db.status.attribute
  db.topic.role.attribute = attribute role { text }
  db.topic.type.attribute =

    ## Identifies the topic type
    attribute type { text }
  db.topic.attlist =
    db.topic.role.attribute?
    & db.topic.type.attribute?
    & db.common.attributes
    & db.common.linking.attributes
    & db.label.attribute?
    & db.topic.status.attribute?
  db.topic.info = db._info.title.req
  db.topic =

    ## A modular unit of documentation not part of any particular narrative flow
    element topic {
      db.topic.attlist,
      db.topic.info,
      db.navigation.components*,
      db.toplevel.blocks.or.sections,
      db.navigation.components*
    }
}
start =
  db.assembly
  | db.resources
  | db.relationships
  | db.transforms
  | db.module
db.grammar.attribute =

  ## Identifies the markup grammar of a resource
  attribute grammar { text }
div {
  db.assembly.role.attribute = attribute role { text }
  db.assembly.attlist =
    db.assembly.role.attribute? & db.common.attributes
  db.assembly.info = db._info
  db.assembly =

    ## Defines the hierarchy and relationships for a collection of resources
    [
      s:pattern [
        "\x{a}" ~
        "               "
        ~~rngs~~s:title [ "Root must have version" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "/db:assembly"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "@version"
            "If this element is the root element, it must have a version attribute."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element assembly {
      db.assembly.attlist,
      db.assembly.info,
      db.resources+,
      db.structure*,
      db.relationships*,
      db.transforms?
    }
}
div {
  db.resources.role.attribute = attribute role { text }
  db.resources.grammar.attribute = db.grammar.attribute
  db.resources.attlist =
    db.resources.role.attribute?
    & db.resources.grammar.attribute?
    & db.common.attributes
  db.resources.info = db._info.title.forbidden
  db.resources =

    ## Contains one or more resource objects that are managed by the assembly
    [
      s:pattern [
        "\x{a}" ~
        "               "
        ~~rngs~~s:title [ "Root must have version" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "/db:resources"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "@version"
            "If this element is the root element, it must have a version attribute."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element resources {
      db.resources.attlist,
      db.resources.info?,
      (db.description*, db.resource+)
    }
}
div {
  db.resource.role.attribute = attribute role { text }
  db.resource.href.attribute =

    ## Indentifies the location of the data by URI
    attribute href { xsd:anyURI }
  db.resource.grammar.attribute = db.grammar.attribute
  db.resource.transform.attribute =

    ## Identifies the markup grammar of a resource
    attribute transform { xsd:NMTOKEN }
  db.resource.attlist =
    db.resource.role.attribute?
    & (db.resource.transform.attribute | db.resource.grammar.attribute)?
    & db.common.attributes
  db.resource =

    ## Identifies an object managed within the assembly
    element resource {
      db.resource.attlist, db.resource.href.attribute, db.description*
    }
}
div {
  db.structure.role.attribute = attribute role { text }
  db.structure.type.attribute =

    ## Identifies the structure type of the structure
    attribute type { xsd:NMTOKEN }
  db.structure.resourceref.attribute =

    ## Indicates a single resource from which to construct this structure
    attribute resourceref { xsd:IDREF }
  db.structure.defaultformat.attribute =

    ## Identifies the default format of the structure
    attribute defaultformat { xsd:NMTOKEN }
  db.structure.renderas.attribute =

    ## Specifies the DocBook element to which this unit should be renamed
    attribute renderas { xsd:QName }
  db.structure.attlist =
    db.structure.role.attribute?
    & db.structure.type.attribute?
    & db.structure.resourceref.attribute?
    & db.structure.renderas.attribute?
    & db.structure.defaultformat.attribute?
    & db.common.attributes
  db.structure.info = db.info?
  db.structure =

    ## Describes the structure of a document
    [
      s:pattern [
        "\x{a}" ~
        "                "
        s:title [ "Specification of renderas" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "db:structure"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "not(@renderas and db:output/@renderas)"
            "The renderas attribute can be specified on either the structure or output,
but not both."
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element structure {
      db.structure.attlist,
      (db.output* & db.filterin? & db.filterout? & db.structure.info),
      db.merge?,
      db.revhistory?,
      db.module+
    }
}
div {
  db.output.role.attribute = attribute role { text }
  db.output.chunk.attribute =

    ## Specifies chunking for this module
    [ a:defaultValue = "auto" ]
    attribute chunk { db.module.chunk.enumeration }
  db.output.file.attribute =

    ## Specifies the output file for this module or structure
    attribute file { xsd:anyURI }
  db.output.renderas.attribute =

    ## Specifies the DocBook element to which this unit should be renamed
    attribute renderas { xsd:QName }
~~  db.output.grammar.attribute = db.grammar.attribute~~
  db.output.transform.attribute =

    ## Specifies the transformation that should be applied to this unit
    attribute transform { xsd:NMTOKEN }
  db.output.suppress.attribute =

    ## Indicates whether or not this unit should be suppressed
    attribute suppress { xsd:boolean }
  db.output.attlist =
    db.output.role.attribute?
    & db.common.attributes
    & db.output.chunk.attribute?
    & db.output.file.attribute?
    & db.output.renderas.attribute?
~~    & db.output.grammar.attribute?~~
    & db.output.transform.attribute?
    & db.output.suppress.attribute?
  db.output =

    ## Specify an output format and/or file name and/or renderas
    element output { db.output.attlist, empty }
}
div {
  db.merge.role.attribute = attribute role { text }
  db.merge.resourceref.attribute =

    ## Indicates a single resource from which to read merged info
    attribute resourceref { xsd:IDREF }
  db.merge.attlist =
    db.merge.role.attribute?
    & db.merge.resourceref.attribute?
    & db.common.attributes
  db.merge =

    ## A wrapper for information that a module overrides in the resource it includes
    element merge { db.merge.attlist, (db._title & db.info.elements*) }
}
div {
  db.module.role.attribute = attribute role { text }
  db.module.chunk.enumeration =

    ## This module will be in a chunk
    "true"
    |
      ## This module will not be in a chunk
      "false"
    |
      ## Chunking of this module depends on the overall chunking algorithm
      "auto"
  db.module.chunk.attribute =

    ## Specifies chunking for this module
    [ a:defaultValue = "auto" ]
    attribute chunk { db.module.chunk.enumeration }
  db.module.resourceref.attribute =

    ## ~~Indicates ~~Identifies a single resource or structure within the assembly from
which to construct this module
    attribute resourceref { xsd:IDREF }
  db.module.omittitles.attribute =

    ## Indicates if titles should be omitted when including a resource
    attribute omittitles { xsd:boolean }?
  db.module.contentonly.attribute =

    ## Indicates if the root element should be omitted when including the resource
(copying only the ~~content should be copied when including a resource~~children)
    attribute contentonly { xsd:boolean }?
  db.module.renderas.attribute =

~~    ## Specifies the DocBook element to which this unit should be renamed~~
    ## Changes the name of the root element of the included resource to the specified
name
```

```
    attribute renderas { xsd:QName }
  db.module.attlist =
    db.module.role.attribute?
    & db.module.chunk.attribute?
    & db.module.resourceref.attribute?
    & db.module.omittitles.attribute?
    & db.module.contentonly.attribute?
    & db.module.renderas.attribute?
    & db.common.attributes
  db.module.info = db.info?
  db.module =

    ## A modular component within a structure
    [
      s:pattern [
        "\x{a}" ~
        "               "
        ~~rng~~s:title [ "Root must have version" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "/db:module"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "@version"
            "If this element is the root element, it must have a version attribute."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
      s:pattern [
        "\x{a}" ~
        "               "
        s:title [ "Specification of renderas" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "db:module"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "not(@renderas and db:output/@renderas)"
            "The renderas attribute can be specified on either the structure or output,
but not both."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element module {
      db.module.attlist,
      ((db.output | db.filterin | db.filterout)*,
       db.module.info,
       db.merge?,
       db.module*)
    }
}
div {
  db.filterout.role.attribute = attribute role { text }
  db.filterout.attlist =
    db.filterout.role.attribute? & db.common.attributes
  db.filterout =

    ## Elements with effectivity attributes matching this element are suppressed
    element filterout { db.filterout.attlist, empty }
}
div {
  db.filterin.role.attribute = attribute role { text }
  db.filterin.attlist =
    db.filterin.role.attribute? & db.common.attributes
  db.filterin =

    ## Elements with effectivity attributes matching this element are allowed
    element filterin { db.filterin.attlist, empty }
}
div {
  db.relationships.role.attribute = attribute role { text }
  db.relationships.type.attribute =

    ## Identifies the type of the contained relationships
    attribute type { xsd:NMTOKENS }
  db.relationships.attlist =
    db.relationships.role.attribute?
    & db.relationships.type.attribute?
    & db.common.attributes
  db.relationships.info = db._info
  db.relationships =

    ## Groups relationship elements to define associations between resources
    [
      s:pattern [
        "\x{a}" ~
        "               "
        ~~rng~~s:title [ "Root must have version" ]
        "\x{a}" ~
        "               "
        s:rule [
          context = "/db:relationships"
          "\x{a}" ~
          "                  "
          s:assert [
            test = "@version"
            "If this element is the root element, it must have a version attribute."
          ]
          "\x{a}" ~
          "               "
        ]
        "\x{a}" ~
        "            "
      ]
    ]
    element relationships {
      db.relationships.attlist,
      db.relationships.info,
      (db.relationship | db.instance)+
    }
}
div {
  db.relationship.role.attribute = attribute role { text }
  db.relationship.type.attribute =

    ## Identifies the type of the relationship
    attribute type { xsd:NMTOKEN }
  db.relationship.attlist =
    db.relationship.role.attribute?
    & db.relationship.type.attribute?
    & db.linkend.attribute?
    & db.common.attributes
  db.relationship =

    ## A relationship associates one or more resources
    element relationship {
      db.relationship.attlist, db.association, db.instance+
    }
}
div {
  db.association.role.attribute = attribute role { text }
  db.association.attlist =
    db.association.role.attribute?
    & db.linkend.attribute?
    & db.common.attributes
  db.association =

    ## Identifies the type of relationship between one or more resources
    element association { db.association.attlist, text? }
}
div {
  db.instance.role.attribute = attribute role { text }
  db.instance.linking.attribute =

    ## Specifies the type of link for this instance
    attribute linking { xsd:NMTOKENS }
  db.instance.attlist =
    db.instance.role.attribute?
    & db.instance.linking.attribute?
    & db.common.attributes
  db.instance =

    ## Identifies a resource that is part of a relationship
    element instance {
      db.instance.attlist, db.linkend.attribute, empty
    }
}
div {
  db.transforms.role.attribute = attribute role { text }
  db.transforms.attlist =
    db.transforms.role.attribute? & db.common.attributes
  db.transforms.info = db._info
  db.transforms =

    ## List of transforms for converting from non-DocBook schemas
    [
      s:pattern [
        "\x{a}" ~
        "                "
        ~~rng~~s:title [ "Root must have version" ]
        "\x{a}" ~
        "                "
        s:rule [
          context = "/db:transforms"
          "\x{a}" ~
          "                   "
          s:assert [
            test = "@version"
            "If this element is the root element, it must have a version attribute."
          ]
          "\x{a}" ~
          "                "
        ]
        "\x{a}" ~
        "             "
      ]
    ]
    element transforms {
      db.transforms.attlist, db.transforms.info, db.transform+
    }
}
div {
  db.transform.role.attribute = attribute role { text }
  db.transform.grammar.attribute = db.grammar.attribute
  db.transform.href.attribute =

    ## Indentifies the location of the data by URI
    attribute href { xsd:anyURI }
  db.transform.name.attribute =

~~    ## Identifies the location of the data by reference~~
    ## The name of the transform
    attribute name { xsd:NMTOKEN }
  db.transform.attlist =
    db.transform.role.attribute?
    & (db.transform.grammar.attribute | db.transform.name.attribute)
    & db.transform.href.attribute
    & db.common.attributes
  db.transform =

    ## Identifies a transform for converting from a non-DocBook schema
    element transform { db.transform.attlist, empty }
}
div {
  db.description.role.attribute = attribute role { text }
  db.description.attlist =
    db.description.role.attribute? & db.common.attributes
  db.description =

    ## A description of a resource or resources
    element description { db.description.attlist, db._text }
}
```