

Multimedia systems and equipment — Multimedia E-Publishing and E-Books — Generic format for E-Publishing

ICS 33.160.60; 35.240.30

National foreword

This British Standard was published by BSI. It is the UK implementation of IEC 62448:2007.

The UK participation in its preparation was entrusted to Technical Committee EPL/100, Audio, video and multimedia systems and equipment.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 May 2007

© BSI 2007

ISBN 978 0 580 50818 9

Amendments issued since publication

Amd. No.	Date	Comments

INTERNATIONAL STANDARD

IEC
62448

First edition
2007-04

Multimedia systems and equipment – Multimedia E-Publishing and E-Books – Generic format for E-Publishing



CONTENTS

INTRODUCTION.....	3
1 Scope.....	4
2 Normative references	4
3 Terms and definitions	4
4 Position and requirements for generic format.....	4
4.1 Generic format in contents creation/distribution model.....	4
4.2 Requirements for generic format	5
5 Notation	5
6 Logical structure.....	5
7 Semantics	7
Annex A (normative) BBeB Xylog Format.....	8
Bibliography.....	78
Figure 1 – Contents creation/distribution model	5
Figure A.1 – Conceptual diagram of the “Block layout”	22
Figure A.2 – Coordinate system.....	23
Figure A.3 – Page layout composition.....	24
Figure A.4 – Block layout composition	25
Figure A.5 – Bgimagemode attribute.....	31
Figure A.6 – Layout attribute	32

INTRODUCTION

Markets for multimedia e-book and e-publishing require standardization of formats for e-book data interchange among associated people; authors, data preparers, publishers and readers. The formats are classified into submission format, generic format and reader's format. The submission format has to support an interaction between authors and data preparers. The reader's format depends on e-publishing equipment. The generic format has to provide an interchange format for data preparers and publishers and therefore should be e-publishing equipment independent.

MULTIMEDIA SYSTEMS AND EQUIPMENT – MULTIMEDIA E-PUBLISHING AND E-BOOKS – GENERIC FORMAT FOR E-PUBLISHING

1 Scope

This International Standard specifies a generic format for multimedia e-publishing employed for e-book data interchange among data preparers and publishers, satisfying a number of publishers requirements: revisable, extensible and heterogeneous logical structure.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 62229:2005, *Multimedia systems and equipment – Multimedia e-publishing and e-books – Conceptual model for multimedia e-publishing*

ISO/IEC 19757-2:2003 *Information technology – Document Schema Definition Language (DSDL) – Part 2: Regular-grammar based validation – RELAX NG Amendment 1* (2003)

ISO 639, *Codes for the representation of names of languages*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1

multimedia e-book

multimedia content consisting of texts, graphics, sounds and/or videos

3.2

data preparer

organization or person that prepares an e-book. An editor is an example of a preparer

3.3

publisher

organization or person that issues and distributes an e-book

3.4

generic format

format for multimedia e-book contents created and edited by a data preparer and modified by a publisher to a reader's format for e-book distribution

4 Position and requirements for generic format

4.1 Generic format in contents creation/distribution model

The conceptual model for multimedia e-publishing (IEC 62229) defines a contents creation/distribution model shown in Figure 1.

Author <--(1)--> Data preparer <--(2)--> Publisher --(3)--> Reader

IEC 2379/06

Figure 1 – Contents creation/distribution model

Between the adjacent processing steps, data preparer and publisher, e-book contents data are interchanged using the generic format. Merging original texts, graphics, sounds and/or videos provided by authors, the data preparer creates and edits the e-book data in a generic format. The e-book data are stored and forwarded to the publishers.

Publishers modify the generic format into reader's formats appropriate for distribution schemes and devices. The generic format can be used for e-book distribution as well.

4.2 Requirements for generic format

The generic format can satisfy the following requirements of the data preparer and the publisher:

- a) logical structure:
the generic format has to have a logical structure that can easily be revised;
- b) style specification:
the logical elements in the generic format should be rendered in accordance with a style specification for creating an appropriate reader's format;
- c) heterogeneous structures:
multimedia e-books may include heterogeneous structures such as mathematics, chemistry and music notes according to the genre of contents. Those specific structures should be imported to the generic format;
- d) metadata:
the generic format should support some metadata to manage the e-book contents data.

5 Notation

The logical structure of generic format is described by using RELAX NG schema, ISO/IEC 19757-2 and its amendment 1.

6 Logical structure

The logical structure of generic format: ebook-g, provides a simple and extensible format ebook-g-core and an existing and actually employed format bbebxlog. The structure of bbebxlog is shown in Annex A.

default namespace="http://tc100.iec.ch/2005/ebook/generic"

start = ebook-g

ebook-g = ebook-g-core

| external "bbebxlog.rnc"

ebook-g-core = meta-g & body-g

meta-g = external "meta-g.rnc"

body-g = element body-g { body }

body =

title*,

foreword*,

introduction*,
titled-clause+,
annex*,
bibliography*
title = element title {
 element main { text },
 element sub { text }?
}
foreword = element foreword { text }
introduction = element introduction { block+ }
titled-clause = element clause { id, title, clause-content }
clause-content =
 (titled-clause, titled-clause+) | untitled-clause-content
untitled-clause-content = (untitled-clause, untitled-clause+) | block+
untitled-clause = element clause { id, untitled-clause-content }
referenced-document =
 element referenced-document {
 id,
 element abbrev { text },
 element title { text },
 element field { text }*,
 element url { xsd:anyURI }
 }
annex =
 element annex {
 id,
 title,
 clause-content
 }
bibliography = element bibliography { referenced-document+ }
block = p | ol | ul | example | note | pre | float
p = element p { inline }
float = table | figure
table = external "table.rnc"
figure = external "figure.rnc"
ol =
 element ol {
 element li { id, block+ }+
 }
ul =
 element ul {
 element li { block+ }+
 }
example = element example { p+ }
note = element note { p+ }

```

pre = element pre { pre-content }
pre-content =
  (text
   | element var { pre-content })*
inline =
  (text
   | element code | b | i | var { inline }
   | ref
   | strong)*
ref =
  element ref {
    attribute to { xsd:IDREF }
  }
id = attribute id { xsd:ID }?
strong = element strong { text }

```

7 Semantics

Elements of ebook-g-core should be rendered in accordance with appropriate style specifications. Actual style specifications by style languages, for example XSL or DSSSL, are outside the scope of this standard.

The contents of meta-g.rnc, table.rnc and figure.rnc are not specified within an architecture of the ebook-g-core.

Annex A (normative)

BBeB Xylog Format

A.1 General

This annex specifies the bbebxlog format mentioned in Clause 6.

This format is intended to be used for two kinds of users, that is, the e-Book contents producer and the related tool developer. Those who produce contents can use this specification as a generic contents data storage format based on the international standard specification. The related tool developer can use it as a reference to decide on the specification for the display equipment and to define the conversion rule between this specification and the other data forms such as any types of XML or HTML.

The feature of this specification specified in this annex includes the following:

- a) "expression of book on paper" concerning two screens layout and modified characters, etc.
- b) "expression of digital" such as sound, page jump and interactive action, etc.

The elements necessary for the above-mentioned expressions and their smooth operations are defined in this specification.

A.2 Structure description

namespace a = "http://relaxng.org/ns/compatibility/annotations/1.0"

namespace sch = "http://www.ascc.net/xml/schematron"

DrawChar =

- (element.Plot
- | element.CR
- | element.Fill
- | element.CharButton
- | element.Yoko
- | element.Tate
- | element.Nekase
- | element.NoBR
- | element.DrawChar
- | element.Italic
- | element.Bold
- | SimpleChar1)*

SimpleChar1 =

- (element.Rubi
- | element.Box
- | element.EmpDots
- | element.EmpLine
- | element.Sub
- | element.Sup

```

| element.Space
| SimpleChar0)*
SimpleChar0 = (text | element.Gaiji | element.AltString)*
SimpleChar2 = (element.Plot | SimpleChar0)*
# BBeB XML Xylog Basic
element.BBeBXylog =
  element BBeBXylog {
    attlist.version,
    element.BookInformation,
    element.Main,
    element.Style,
    element.Objects
  }
attlist.version &=
  attribute version { text }
>> a:documentation [
  ' Constraint: Current "version" attribute value is "1.0" '
]
element.BookInformation =
  element BookInformation { element.Info, element.TOC }
element.Info =
  element Info {
    attlist.version, element.BookInfo, element.DocInfo, element.Keyword*
  }
element.BookInfo =
  element BookInfo {
    element.Title,
    element.Author,
    element.BookID,
    element.Publisher,
    element.Label,
    element.Category*,
    element.Classification,
    element.FreeText
  }
element.Title = element Title { attlist.reading, text }
attlist.reading &= attribute reading { text }
element.Author = element Author { attlist.reading, text }
element.BookID = element BookID { text }
element.Publisher = element Publisher { attlist.reading, text }
element.Label = element Label { attlist.reading, text }
element.Category = element Category { text }
element.Classification = element Classification { text }
element.FreeText = element FreeText { text }
element.DocInfo =

```

```

element DocInfo {
  element.CThumbnail,
  element.Language,
  element.Creator,
  element.SumPage
}
element.CThumbnail = element CThumbnail { attlist.CThumbnail }
attlist.CThumbnail &= attribute file { text }
element.Language =
  element Language {
    xsd:string { length = "2" }
    >> a:documentation [
      " Constraint: Set two-letter primary language code which is specified in ISO 639. "
    ]
  }
element.Creator = element Creator { text }
element.SumPage = element SumPage { text }
element.Keyword = element Keyword { text }
element.TOC = element TOC { element.TocLabel* }
element.TocLabel = element TocLabel { attlist.jump, text }
element.Main = element Main { element.Page+ }
element.Page =
  element Page {
    attlist.Page,
    element.Common.Page
    >> sch:pattern [
      name = "Style Check"
      "\x{a}" ~
      "      "

      sch:rule [
        context = "Page"
        "\x{a}" ~
        "      "

        sch:assert [
          test = "@pagestyle = //Style/PageStyle/@stylelabel"
          "No PageStyle"
        ]
        "\x{a}" ~
        "      "
      ]
      "\x{a}" ~
      "      "
    ]
  }
element.Common.Page =

```

```

(element.Common.Objects
| element.BlockSpace
| element.RuledLine
| element.Wait)*
attlist.id &= attribute objid { xsd:string }
attlist.refid &= attribute refobj { xsd:string }
attlist.refstream &= attribute refstream { xsd:string }
attlist.Page &=
  attlist.id,
  attribute pagestyle { text },
  attlist.Common.Page
element.TextBlock =
  element TextBlock {
    attlist.TextBlock,
    ((element.P | element.CR)*)
  }
  >> sch:pattern [
    name = "Style Check"
    "\x{a}" ~
    "      "

    sch:rule [
      context = "TextBlock"
      "\x{a}" ~
      "      "

      sch:assert [
        test = "@blockstyle = //Style/BlockStyle/@stylelabel"
        "No BlockStyle"
      ]
      "\x{a}" ~
      "      "
    ]
    "\x{a}" ~
    "      "
  ]
  >> sch:pattern [
    name = "Style Check"
    "\x{a}" ~
    "      "

    sch:rule [
      context = "TextBlock"
      "\x{a}" ~
      "      "

      sch:assert [
        test = "@textstyle = //Style/TextStyle/@stylelabel"
        "No TextStyle"
      ]
    ]
  ]

```

```

        "\x{a}" ~
        "      "
    ]
    "\x{a}" ~
    "      "
]
}
attlist.TextBlock &=
    attlist.id,
    attribute textstyle { text },
    attribute blockstyle { text },
    attlist.Common.Text,
    attlist.Common.Block
element.P = element P { attlist.P, DrawChar }
attlist.P &= attribute refesound { xsd:string }?
element.Plot = element Plot { attlist.Plot, text }
attlist.xsize &= attribute xsize { xsd:unsignedShort }
attlist.ysize &= attribute ysize { xsd:unsignedShort }
attlist.Plot &=
    attlist.xsize,
    attlist.ysize,
    attlist.refid,
    [ a:defaultValue = "bottom" ]
    attribute adjustment { "center" | "baseline" | "top" | "bottom" }?
element.CR = element CR { empty }
element.Fill =
    element Fill {
        attribute code { text },
        empty
    }
element.Space = element Space { attlist.xsize, empty }
element.CharButton = element CharButton { attlist.refid, SimpleChar1 }
element.Rubi = element Rubi { (element.Oyamoji, element.Rubimoji)+ }
element.Oyamoji = element Oyamoji { SimpleChar0 }
element.Gaiji = element Gaiji { attlist.Gaiji, text }
attlist.Gaiji &=
    attlist.refid,
    attribute fontfacename { text }?,
    attribute code { text }
element.AltString = element AltString { element.Org, element.Alt }
element.Org = element Org { text }
element.Alt = element Alt { text }
element.Rubimoji = element Rubimoji { SimpleChar0 }
element.Box = element Box { attlist.Box, SimpleChar0 }
attlist.linetype &=

```

```

    attribute linetype { "solid" | "dotted" | "dashed" | "double" }
attlist.Box &= attlist.linetype?
element.EmpDots = element EmpDots { attlist.EmpDots, SimpleChar0 }
attlist.Empdotsposition &=
    attribute empdotsposition { "before" | "after" }
attlist.EmpdotsChar &=
    attribute empdotscode { text },
    attribute empdotsfontname { text },
    attribute refempdotsfont { text }
attlist.EmpDots &= attlist.Empdotsposition?, attlist.EmpdotsChar?
element.EmpLine = element EmpLine { attlist.EmpLine, SimpleChar0 }
attlist.Emplineposition &=
    attribute emplineposition { "before" | "after" }
attlist.Emplinetype &=
    attribute emlinetype {
        "none" | "solid" | "dotted" | "dashed" | "double"
    }
attlist.EmpLine &= attlist.Emplinetype?, attlist.Emplineposition?
element.Sub = element Sub { SimpleChar0 }
element.Sup = element Sup { SimpleChar0 }
element.Yoko = element Yoko { SimpleChar0 }
element.Tate = element Tate { SimpleChar2 }
element.Nekase = element Nekase { SimpleChar2 }
element.NoBR = element NoBR { SimpleChar1 }
element.DrawChar =
    element DrawChar {
        attribute line { xsd:unsignedShort },
        SimpleChar0
    }
element.Italic = element Italic { DrawChar }
element.Bold = element Bold { DrawChar }
element.ImageBlock =
    element ImageBlock { attlist.ImageBlock, text }
>> sch:pattern [
    name = "Style Check"
    "\x{a}" ~
    "    "

    sch:rule [
        context = "Page"
        "\x{a}" ~
        "    "

    sch:assert [
        test = "@blockstyle = //Style/BlockStyle/@stylelabel"
        "No BlockStyle"
    ]
]

```

```

        "\x{a}" ~
        "      "

    ]
    "\x{a}" ~
    "      "

]
attlist.ImageBlock &=
    attlist.id,
    attribute blockstyle { text },
    attlist.Common.Block,
    attlist.Common.Image
element.ButtonBlock =
    element ButtonBlock {
        attlist.ButtonBlock,
        element.BaseButton?,
        element.FocusinButton?,
        element.PushButton?,
        (element.UpButton?)
    }
    >> sch:pattern [
        name = "Style Check"
        "\x{a}" ~
        "      "

        sch:rule [
            context = "ButtonBlock"
            "\x{a}" ~
            "      "

            sch:assert [
                test = "@blockstyle = //Style/BlockStyle/@stylelabel"
                "No BlockStyle"
            ]
            "\x{a}" ~
            "      "

        ]
        "\x{a}" ~
        "      "

    ]
}
attlist.ButtonBlock &=
    attlist.id,
    attribute blockstyle { text },
    attlist.Common.Block
attlist.refimage &= attribute refimage { xsd:string }
element.BaseButton = element BaseButton { attlist.refimage?, empty }
element.FocusinButton =
    element FocusinButton {

```

```

    attlist.refimage?,
    (element.JumpTo
      | element.Run
      | element.SoundStop
      | element.CloseWindow)*
  }
element.JumpTo = element JumpTo { attlist.jump, empty }
attlist.jump &=
  attribute refpage { xsd:string },
  attlist.refid
element.Run = element Run { attlist.Run, empty }
attlist.Run &=
  [ a:defaultValue = "normal" ]
  attribute runoption {
    "normal" | "opposite" | "center" | "opposite-center"
  }?,
  attlist.refid
element.SoundStop = element SoundStop { empty }
element.CloseWindow = element CloseWindow { empty }
element.PushButton =
  element PushButton {
    attlist.refimage?,
    (element.JumpTo
      | element.Run
      | element.SoundStop
      | element.CloseWindow)*
  }
element.UpButton =
  element UpButton {
    attlist.refimage?,
    (element.JumpTo
      | element.Run
      | element.SoundStop
      | element.CloseWindow)*
  }
element.BlockSpace = element BlockSpace { attlist.BlockSpace, empty }
attlist.BlockSpace &=
  attribute xspace { xsd:short }?,
  attribute yspace { xsd:short }?
element.Canvas =
  element Canvas {
    attlist.Canvas, (element.Common.Canvas | element.Wait)*
  }
element.Common.Canvas =
  element.PutObj

```

```

| element.Moveto
| element.Lineto
| element.DrawBox
| element.DrawEllipse
attlist.Canvas &=
  attribute canvaswidth { xsd:unsignedShort },
  attribute canvasheight { xsd:unsignedShort },
  attribute blockrule { "block-fixed" | "block-adjustable" }?,
  attlist.id,
  attlist.Common.Canvas
attlist.Layout &= attribute layout { "LrTb" | "TbRI" }
attlist.Common.Frame &=
  attribute framewidth { xsd:unsignedShort }?,
  attribute framecolor { text }?,
  attribute framemode { "curve" | "square" }?
attlist.Common.Canvas &=
  attribute bgcolor { text }?,
  attlist.Layout?,
  attlist.Common.Frame
element.PutObj = element PutObj { attlist.PutObj, empty }
attlist.x1.unsigned &= attribute x1 { xsd:unsignedShort }
attlist.x1.signed &= attribute x1 { xsd:short }
attlist.y1.unsigned &= attribute y1 { xsd:unsignedShort }
attlist.y1.signed &= attribute y1 { xsd:short }
attlist.PutObj &=
  attlist.x1.unsigned, attlist.y1.unsigned, attlist.refid
element.Wait =
  element Wait {
    attribute time { xsd:short },
    empty
  }
element.Moveto =
  element Moveto { attlist.x1.signed, attlist.y1.signed, empty }
element.Lineto = element Lineto { attlist.Lineto, empty }
attlist.Common.line &=
  attribute linewidth { xsd:unsignedShort }?,
  attribute linecolor { text }?,
  attlist.linetype?
attlist.Lineto &=
  attlist.x1.signed,
  attlist.y1.signed,
  attlist.Common.line,
  attribute arrowtype { text }?
element.DrawBox = element DrawBox { attlist.DrawBox, empty }
attlist.DrawBox &=

```

```

attlist.x1.signed,
attlist.y1.signed,
attlist.Common.line,
attribute fillcolor { text }?
element.DrawEllipse = element DrawEllipse { attlist.DrawEllipse, empty }
attlist.DrawEllipse &=
    attlist.x1.signed,
    attlist.y1.signed,
    attlist.Common.line,
    attribute fillcolor { text }?
element.RuledLine = element RuledLine { attlist.RuledLine, empty }
attlist.RuledLine &=
    attlist.linetype,
    attribute linewidth { xsd:unsignedShort },
    attribute linelength { xsd:unsignedShort },
    attribute linecolor { text }?
element.Style =
    element Style {
        element.BookStyle,
        (element.TextStyle | element.BlockStyle | element.PageStyle)*
    }
element.BookStyle =
    element BookStyle {
        attlist.Style,
        element.SetDefault?,
        element.RegistFont*,
        element.BookSetting
    }
attlist.Style &=
    attribute stylelabel { text },
    attlist.id
element.SetDefault = element SetDefault { attlist.SetDefault, empty }
attlist.Rubyoverhang &= attribute rubyoverhang { "auto" | "none" }
attlist.Rubyalign &= attribute rubyalign { "start" | "center" }
attlist.Rubyadjust &= attribute rubyadjust { "line-edge" | "none" }
attlist.Setwaitprop &= attribute setwaitprop { "replay" | "noreplay" }
attlist.SetDefault &=
    attlist.Rubyalign,
    attlist.Rubyoverhang,
    attlist.Empdotsposition,
    attlist.Emplineposition,
    attlist.Emplinetype,
    attlist.EmpdotsChar,
    attlist.Rubyadjust,
    attlist.Setwaitprop

```

```

element.RegistFont = element RegistFont { attlist.RegistFont, empty }
attlist.FontEncoding &= attribute encoding { "TTF" | "OTF" | "BF" }
attlist.RegistFont &=
    attribute fontfilename { text },
    attribute file { text },
    attribute fontname { text },
    attlist.FontEncoding
element.BookSetting = element BookSetting { attlist.BookSetting, empty }
attlist.BookSetting &=
    attribute bindingdirection { "Lr" | "RI" },
    attribute dpi { xsd:unsignedShort },
    attribute screenwidth { xsd:unsignedShort },
    attribute screenheight { xsd:unsignedShort },
    [ a:defaultValue = "24" ] attribute colordepth { xsd:unsignedByte }?
element.TextStyle = element TextStyle { attlist.TextStyle, empty }
attlist.TextStyle &= attlist.Style, attlist.Common.Text
attlist.Common.Text &=
    attlist.EmpDots,
    attlist.EmpLine,
    attribute fontsize { xsd:short }?,
    attribute fontwidth { xsd:short }?,
    attribute fontescapement { xsd:short }?,
    attribute fontorientation { xsd:short }?,
    attribute fontfacename { text }?,
    attribute textcolor { text }?,
    attribute textbgcolor { text }?,
    attribute wordspace { xsd:short }?,
    attribute letterspace { xsd:short }?,
    attribute baselineskip { xsd:short }?,
    attribute linespace { xsd:short }?,
    attribute parindent { xsd:short }?,
    attribute parskip { xsd:short }?,
    attlist.Rubyalign?,
    attlist.Rubyoverhang?,
    attribute column { text }?,
    attribute columnsep { text }?,
    attribute align { "head" | "center" | "foot" }?,
    attribute linecolor { text }?,
    attribute charspace { xsd:short }?,
    attribute fontweight { xsd:unsignedShort }?,
    attlist.Rubyadjust?,
    attribute textlinewidth { xsd:unsignedShort }?
element.BlockStyle = element BlockStyle { attlist.BlockStyle, empty }
attlist.BlockStyle &= attlist.Style, attlist.Common.Block
attlist.Common.Block &=

```

```

attribute blockwidth { xsd:unsignedShort }?,
attribute blockheight { xsd:unsignedShort }?,
attribute blockrule {
    "horz-fixed"
    | "horz-adjustable"
    | "vert-fixed"
    | "vert-adjustable"
    | "block-fixed"
    | "block-adjustable"
}?,
attribute bgcolor { text }?,
attlist.Layout?,
attlist.Common.Frame,
attribute topskip { xsd:unsignedShort }?,
attribute sidemargin { xsd:unsignedShort }?,
attribute footskip { xsd:unsignedShort }?,
attribute refbgimage { text }?,
attlist.Bgimagemode?
element.PageStyle = element PageStyle { attlist.PageStyle, empty }
attlist.PageStyle &=
    attlist.Style,
    attlist.Common.Page,
    attribute setemptyview { "show" | "empty" }?,
    attlist.Setwaitprop?
attlist.Bgimagemode &=
    attribute bgimagemode { "fix" | "tile" | "centering" }
attlist.Common.Page &=
    (attribute refbgimage { text },
    attlist.Bgimagemode)?,
    attribute evenfooterid { xsd:string }?,
    attribute evenheaderid { xsd:string }?,
    attribute oddfooterid { xsd:string }?,
    attribute oddheaderid { xsd:string }?,
    attribute pageposition { "upper" | "lower" | "any" }?,
    attribute topmargin { xsd:short }?,
    attribute headheight { xsd:short }?,
    attribute headsep { xsd:short }?,
    attribute oddsidemargin { xsd:unsignedShort }?,
    attribute evensidemargin { xsd:unsignedShort }?,
    attribute textheight { xsd:unsignedShort }?,
    attribute textwidth { xsd:unsignedShort }?,
    attribute footheight { xsd:short }?,
    attlist.Layout?,
    attribute footspace { xsd:short }?
element.Common.Objects &=

```

```

    element.ImageBlock
    | element.TextBlock
    | element.ButtonBlock
    | element.Canvas
element.Objects =
    element Objects {
        (element.Common.Objects
        | element.Window
        | element.PopUpWin
        | element.Sound
        | element.SoundStream
        | element.ImageStream
        | element.Header
        | element.Footer
        | element.eSound
        | element.Font
        | element.Image
        | element.Button)*
    }
element.Window = element Window { attlist.Window, element.Common.Page }
attlist.Window &=
    attribute windowwidth { xsd:unsignedShort }?,
    attribute windowheight { xsd:unsignedShort }?,
    attlist.Layout?,
    attlist.id
element.PopUpWin =
    element PopUpWin {
        attlist.PopUpWin, (element.TextBlock | element.ImageBlock)
    }
attlist.PopUpWin &= attlist.id
element.Sound = element Sound { attlist.Sound, empty }
attlist.Sound &=
    [ a:defaultValue = " 1" ] attribute times { xsd:unsignedShort }?,
    [ a:defaultValue = "sync" ] attribute playmode { "sync" | "async" }?,
    attlist.refstream,
    attlist.id
element.SoundStream = element SoundStream { attlist.SoundStream, empty }
attlist.SoundStream &=
    attribute encoding { "PCM" | "MP3" },
    attribute file { text },
    attlist.id
element.ImageStream = element ImageStream { attlist.ImageStream, text }
attlist.ImageStream &=
    attribute encoding { "JPEG" | "GIF" | "BMP" | "PNG" },
    attribute file { text },

```

```

attlist.id
element.Header =
    element Header {
        attlist.id, attlist.Common.Canvas, (element.Common.Canvas)*
    }
element.Footer =
    element Footer {
        attlist.id, attlist.Common.Canvas, (element.Common.Canvas)*
    }
element.eSound = element eSound { attlist.eSound, empty }
attlist.eSound &= attlist.id, attlist.refstream
element.Font = element Font { attlist.Font, empty }
attlist.Font &=
    attribute file { text },
    attribute fontfilename { text },
    attribute fontname { text },
    attlist.id,
    attlist.FontEncoding
element.Image = element Image { attlist.Image, text }
attlist.Image &= attlist.id, attlist.Common.Image
attlist.Common.Image &=
    attribute x0 { xsd:unsignedShort },
    attribute y0 { xsd:unsignedShort },
    attlist.x1.unsigned,
    attlist.y1.unsigned,
    attlist.xsize,
    attlist.ysize,
    attlist.refstream
element.Button =
    element Button {
        attlist.Button,
        element.BaseButton?,
        element.FocusinButton?,
        element.PushButton?,
        element.UpButton?
    }
attlist.Button &= attlist.id
start = element.BBeBXylog

```

A.3 Block layout

A.3.1 Layout

The “BBeB Book Format” utilizes the “Block layout” concept. In the “Block layout”, the content to be shown is displayed in an area called “Block”. The layout is composed of multiple “Block”s. Each “Block” is placed in a relative location. The “BBeB XML Xylog” file format defines the implementation rules based on this “Block layout” concept.

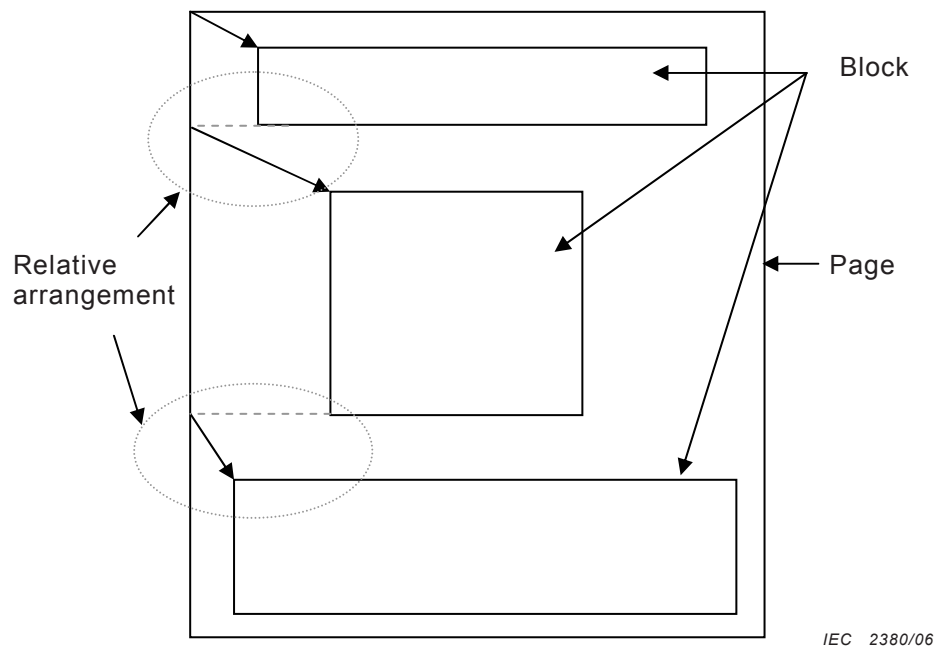


Figure A.1 – Conceptual diagram of the “Block layout”

A.3.2 Block

This specification defines the **TextBlock**, **ImageBlock**, **ButtonBlock**, and **Canvas** elements. These are positioned on a “Page” as a “Block”. These are described as sub-elements of the **Page** element with the **BlockSpace** element that indicates the relative arrangement position.

It is possible to specify background image, background color and frame in the “Block”.

A.3.3 Coordinate system

This specification uses a two-dimensional coordinate system (X,Y) that depends on the vertical layout or the horizontal layout.

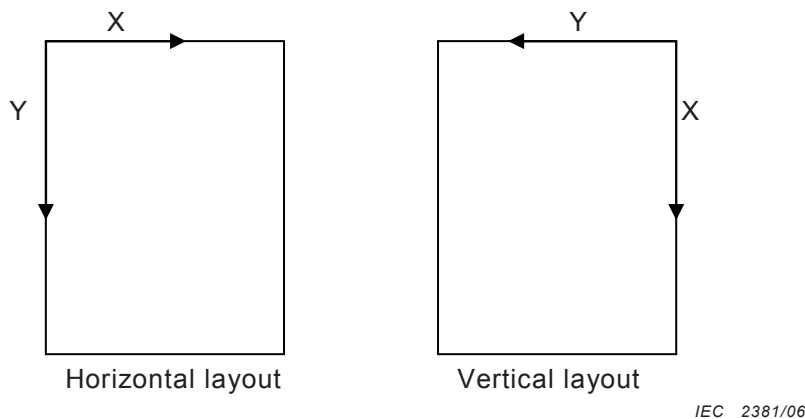


Figure A.2 – Coordinate system

A.4 Terminology

A.4.1 Page layout

For the page layout area, the following terms are defined (see Figure A.3):

Screen size:

screen size of page content which is expected to be viewed by a specific display device and specified by the value of the **screenwidth** and **screenheight** attribute in the **BookSetting** element (see A.5.71)

topmargin:

the space between the top edge of screen and Header area

Header area:

area where header(s) is/are placed

headheight:

the height of Header area

headsep:

the space between Header area and Main text area

Main text area:

area where main content is placed

textheight:

the height of Main text area

textwidth:

the width of Header area, Main text area and Footer area

footspace:

the distance between the bottom of Main text area and the bottom edge of Footer area

footheight:

the height of Footer area

oddsidemargin:

the margin between the left-hand side edge of screen and left-hand side edge of Main text area in the odd page