実践！PDF5-*ML* プラグイン

牧田 敏彦

tmakita@antenna.co.jp

A Data Usability Company
ANTENNA HOUSE

ANTENNA HOUSE
Formatter V6

目次

- PDF5-*ML*プラグインまでの経緯
- PDF5-*ML*プラグインの特徴
- PDF5-*ML* プラグインは「使える」でしょうか？

目次

- PDF5-*ML*プラグインまでの経緯
- PDF5-*ML*プラグインの特徴
- PDF5-*ML* プラグインは「使える」でしょうか？

PDF5–*ML* プラグインまでの経緯

- 前身のPDF5は2008年からI18N Index LibraryのReference Implementationとして開発されました.
  - http://www.antennahouse.com/antenna1/i18n-index-library/
  - https://github.com/AntennaHouse/pdf5

- PDF5-*ML*はPDF5の拡張版として2012年から開発され、2015年8月末に公開されました.
  - https://github.com/AntennaHouse/pdf5-ml

DITA-OTのPDFプラグインの要件

- 多言語のサポート
  - 日英から始まり、多くなれば ***40*** 言語を越えることもあります
- 複数の出力タイプのサポート
  - 印刷用 (カラー, モノクロ), Web用, レビュー用など
- パラメータによる出力条件の切り替え
  - 多くなれば一つのプラグインで ***70*** を越えるパラメータ！

PDFプラグインに必要とされる機能

- 言語によるスタイルとリテラルの変更
  - 典型的にはfont-familyを@xml:langによって切り替え
- 出力タイプによる動的なスタイル割り当て
  - カラー印刷用にはCMYKカラースペース、モノクロ印刷用にはグレースケールカラースペース、WEB用にはRGBカラースペースなど
- フレキシブルなスタイル定義
  - スタイルをロジック（テンプレート）から独立させて定義、スタイルの継承、重複した記述の排除

PDF5でやりたかったこと

- スタイルとロジックを分離したい！
  - DITAからXSL-FOに変換するにはXSLTスタイルシートを書きます.
  - XSLT勧告のxsl:attribute-set を使用すると、スタイルの中にロジックを書かない限り多言語や動的なスタイルの変更が実現できません.
  - その結果スタイルシートは大抵メンテナンス不能の代物になります.
  - 外部のXMLにスタイルを定義すれば、XSLT勧告に縛られない自由なスタイル記述が可能になります.
- 一つのプラグインで、多言語のPDFを作成したい.
  - ここで一個のPDFは一言語に対応するものと考えます. 言語毎にプラグインを作る訳にはゆきません.
  - 英語ベースのスタイル定義を作り、他言語はそこからの差分だけのスタイル定義という方法が取れないか？と考えました.

PDF5の到達点

- スタイルとロジックの分離
  - xsl:attribute-set を使用せず、スタイルは外部の「スタイル定義ファイル」に記述. (config/default_style.xml)
  - 言語別のスタイルは、config/[language-tag]_style.xmlに記述.
  - XSLTライクなattribute-set/attribute, variableを導入、スタイル記述の簡略化を実現.
- 単一言語のPDF生成を実現
  - ルートのmap/@xml:langを文書の言語とみなす
  - 動的にdefault_style.xmlと[language-tag]_style.xmlを合成、言語別のスタイルを優先してXSL-FO(⇒PDF)を生成する.

目次

- PDF5-*ML*プラグインまでの経緯
- PDF5-*ML*プラグインの特徴
- PDF5-*ML* プラグインは「使える」でしょうか？

PDF5−*ML*でやりたかったこと

- 1つの文書で多言語の混在を実現する！
  - PDF5はあくまで一文書は一言語でした.
  - 緊急避難的に“ah-dita”を使えば、特定の箇所にmap/@xml:langと異なるフォントを割り当てて見かけ上の多言語を実現することもできますが本質的ではありません.
  - @xml:langを尊重して、なるべくPDF5と互換を取りながら、その言語のスタイルを割り当てるアルゴリズムを研究しました.
- 「条件付き」の変数とスタイル定義を実現したい！
  - 多くの出力条件をすべてロジックで吸収しているとコードが複雑化し、書くのが嫌になってきます.
  - 主要な出力条件をスタイルの側で「条件付き」変数とスタイルで実現すれば、テンプレート側のコードの負担を劇的に減らすことができます.

PDF5-*ML*で実装したもの

- 1つのPDFでの多言語組版の実現
- 「条件付き」の変数定義とスタイル定義
- フリーフォーマットの表紙など

１つのPDF中での多言語サポート

- 多くのPDFプラグインはルートのmap/@xml:langをその文書の言語とみなし、その値により主なフォントを選択してきました.
- PDF5-*ML* は異なった@xml:langをtopicの「どの階層」でも許容します.
- @xml:langの値により、自動的にその言語に適切なフォントとスタイルを選択します.

世界の「こんにちは」で試してみましょう

```
<p>We say "Hello" in English.</p>

<p>We say "<ph xml:lang="ja-JP">こんにちは</ph>" in
Japanese.</p>
<p>We say "<ph xml:lang="zh-CN">你好</ph>" in
Chinese.</p>
<p>We say "<ph xml:lang="ko-KR">안녕하세요</ph>" in
Korean.</p>

<p>We say "<ph xml:lang="th-TH">สวัสดีครับ</ph>" in
Thai.</p>

<p>We say "<ph xml:lang="hi-IN">नमस्ते</ph>" in
Hindi.</p>
```

組版の条件

- bookmap/@xml:lang=”en-US”とします.
- DITA-OT 2.1.1 と AH Formatter V6.2 を使用します.
- 自動フォントフォールバック（auto font fallback）を禁止します.

PDF5-*ML*の結果

Chapter 1 Multiple language test

1.1 "Hello" worldwide!

We say "Hello" in English.

We say "こんにちは" in Japanese.

We say "你好" in Chinese.

We say "안녕하세요" in Korean.

We say "สวัสดีครับ" in Thai.

We say "नमस्ते" in Hindi.

PDF2の結果

組版結果の考察

- PDF5-*ML* ではすべてのグリフはPDFに問題なく出力されます．組版エラーは報告されません．
- PDF2では非ラテン文字はPDFで「豆腐マーク」（グリフ無し）になってしまいます．組版では多くのエラーが報告されます．PDF2では@xml:langに基づいてフォントが選択されていないためです
- 多言語組版を必要としているならPDF5-*ML*がベターな選択でしょう．

「条件付き」変数とスタイル定義

- 次のあらかじめ定義された属性があります.
  - **@output-type**: 出力タイプ(既定値: WEB)
  - **@doc-type**: 文書タイプ (自由定義)
  - **@paper-size**: ペーパーサイズ(既定値: Letter)
- これらの属性を使用することにより. 条件付き変数、スタイルを「スタイル定義ファイル」に定義できます.
- 条件付きの定義はスタイルシートのテンプレートをこれらの条件から「透過」にし、複雑なコーディングをなくすことができます.

@output-typeによるアイコンの定義

```
<!-- WEB出力用: SVG -->
<variable name="Note_Icon"
output-type="web">url(note.svg)</variable>
<!-- カラー印刷用: フルカラーPDF -->
<variable name="Note_Icon"
 output-type="print-color">url(note.pdf)</variable>
<!-- モノクロ印刷用: モノクロのPDF -->
<variable name="Note_Icon"
 output-type="print-mono">url(note-mono.pdf)</variable>
```

テンプレートのコード

- ahf:getVarValue関数は変数値を変数名から自動的に取得します.
- @output-typeの値はxsl:paramで外部から与えます.
- テンプレートのコードは@output-typeに対し「透過的」に記述できます.

```
<!-- Noteのテンプレート -->
<xsl:template match="*[contains(@class,' topic/note ')]”>
    …
    <fo:external-graphic>
        <xsl:attribute name="src”
                      select="ahf:getVarValue('Note_Icon')”/>
    </fo:external-graphic>
    …
</xsl:template>
```

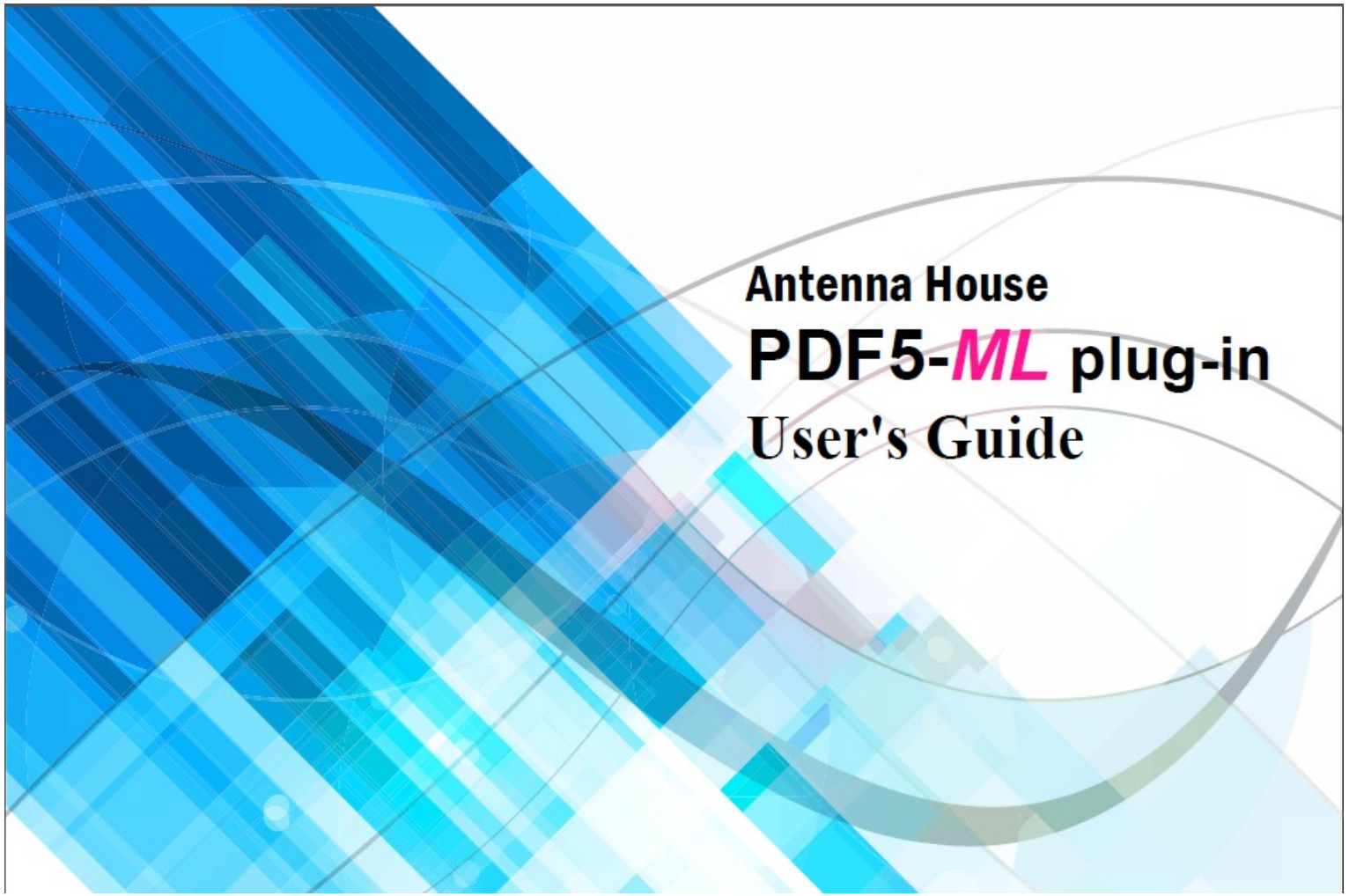
フリーフォーマットの表紙

- そもそも表紙のデータ構造はDITAのtopicの概念とはマッチしません.
- プラグインの開発では表紙を作るために多くの工数を費やしています. もしくは簡単で固定的な表紙に留めています.
- PDF5-*ML*はXSL-FOのプロパティをオーサリングレベルで書けるようにすることで実際的なソリューションを提供します.

表紙のオーサリングサンプル

```
<!--タイトルをfo:prop属性でページ内の絶対位置に置きます. -->
<conbodydiv  fo:prop="position:absolute;top:40mm;right:10mm;
                      width:40%;height:130mm;">
<!--タイトルのFOプロパティをfo:prop属性で指定します. -->
<section>
  <p fo:prop="font-family:Franklin Gothic Demi Cond;font-size:24pt;">
   Antenna House</p>
  <p fo:prop="font-family:Arial;font-weight:bold;
      font-size:32pt;">PDF5-<i fo:prop="color:deeppink;">ML</i>
     <ph fo:prop="font-size:28pt;">plug-in</ph></p>
  <p fo:prop="font-family:Times New Roman;font-weight:bold;
    font-size:28pt;">User's Guide</p>
</section>
</conbodydiv>
```

表紙の組版結果



表紙を作る条件

- 表紙のtopicにはマップのtopicref/@outputclassに”cover*N*” を指定します.
- topicではbodydivからfo:block-containerを生成する前提でオーサリングします.
- XSL-FOのプロパティをfo:prop属性にCSSの記法で指定します.
  - fo:propは「ah-dita」の特殊化で使うことができます.
  - https://github.com/AntennaHouse/ah-dita

このようなアーキテクチャの利点

- ユーザーレベルで自由に表紙を設計できます．ただし最初の雛形は開発者に作ってもらう必要があるでしょう．
- ユーザーは表紙の修正/メンテナンスにお金をかけずに済みます．自分たちで大抵のことはできてしまうからです．
- 開発者は表紙の作成のためにその都度コードを書かずに済みます．表紙のスタイルはすべてオーサリング側が保持しているためです．

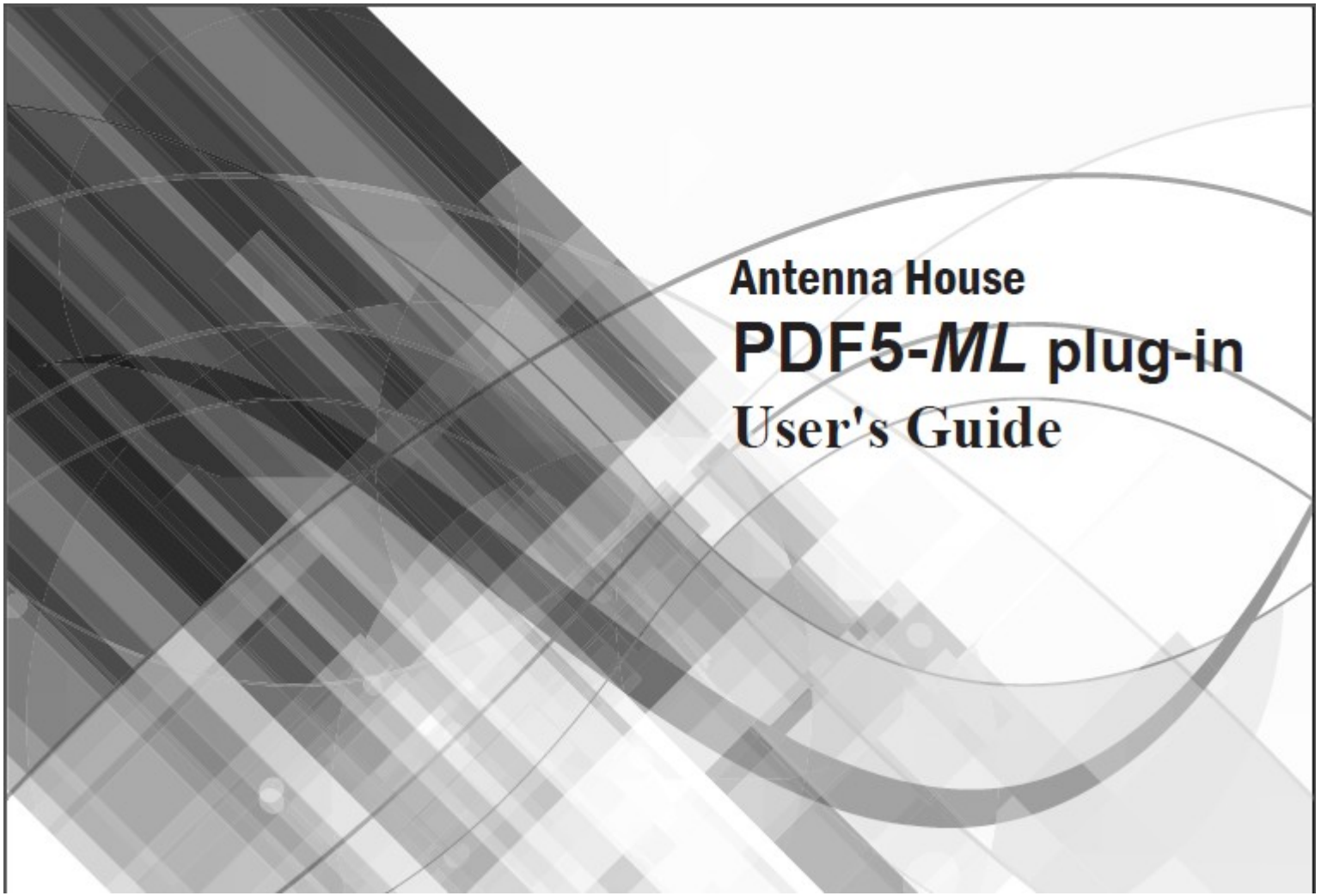
おまけ:DITA 1.3の@delivaryTarget

- @delivaryTargetを使用してフィルタリングすれば、さらにフレキシブルな表紙を作れます.

```
<!--@deliveryTargetによりcolor（文字色）を設定します-->
<p fo:prop="font-family:Arial;font-weight:bold;font-size:32pt;">PDF5-<i
fo:prop="color:deeppink;" deliveryTarget="pdf-web">ML</i><i
fo:prop="color:rgb-icc(#CMYK,0%,92%,42%,0%);"
    deliveryTarget="pdf-print-color">ML</i><i deliveryTarget="pdf-print-
mono">ML</i>
  <ph fo:prop="font-size:28pt;">plug-in</ph>
</p>
```

- このサンプルはGitHubにあります.
  - https://github.com/AntennaHouse/pdf5-ml/tree/master/samples/sample_cover

@deliveryTarget=”pdf-print-mono”



目次

- PDF5-*ML*プラグインまでの経緯
- PDF5-*ML*プラグインの特徴
- PDF5-*ML* プラグインは「使える」でしょうか？

PDF5-*ML* は「使える」でしょうか？
はい！使えます.

- PDF5/PDF5-MLは7年以上にわたって開発されてきました.
- PDF2より簡単な構造をもち、より多くの特徴があります.
- 世界でも使用されているPDF5の後継です.
  - https://groups.yahoo.com/neo/groups/dita-users/conversations/topics/38333

PDF5-*ML* の今後

- DITA 1.3で拡張された要素/属性の実装を引き続き行います.
- 実装を通じてDITA-OTに積極的にcontributeします.
- PDF5は後継のPDF5-*ML*に道を譲りバグフィックスのみといたします.

ご清聴ありがとうございました.

付録

- xsl:attribute-setの限界とPDF5のアイディア
- 言語別スタイル定義ファイル
- PDF5-*ML*の制限事項
- DITAの@xml:langの仕様と現状
- @xml:langによるスタイル切り替え
- PDF5-*ML*のテンプレートの作法
- PDF5-*ML*をカスタマイズする方法
- フォントのフォールバックの禁止
- 言語別にスタイルを変える必要があるか？
- コードポイントからのフォントマッピング

書いてはいけないコード

```
<!-- ロジックとスタイルの混在したテンプレート-->
<xsl:template match="*[contains(@class,' topic/p ')]">
    <fo:block>
        <xsl:attribute name="space-before">0.5em</xsl:attribute>
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
```

- p要素（段落）のテンプレートです.
- XSL-FOのfo:blockを生成するという「ロジック」とspace-beforeに0.5emを設定するという「スタイル設定」が混在しています.
- この方法でスタイルシートを構築すると極めてメンテナンス性の悪いものになります.

xsl:attribute-setの限界

```
<!-- スタイル定義-->
<xsl:attribute-set name="atsP">
    <xsl:attribute name="space-before">0.5em</xsl:attribute>
</xsl:attribute-set>

<!-- テンプレートでのスタイルの使用-->
<xsl:template match="*[contains(@class,' topic/p ')]">
    <fo:block xsl:use-attribute-sets="atsP">
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
```

- xsl:use-attribute-sets属性でしかxsl:attribute-setは参照できません.
- attribute()*を返します.
- xsl:variableと違い、参照される都度評価されます.

スタイルの中でのロジックの記述

```
<xsl:attribute-set name="atsP">
   <xsl:attribute name="space-before">
      <xsl:choose>
         <xsl:when test="ancestor::*[contains(@class,' topic/entry ')]">0.2em</xsl:when>
         <xsl:otherwise>0.5em</xsl:otherwise>
      </xsl:choose>
   </xsl:attribute>
</xsl:attribute-set>
```

- ロジックは子のxsl:attributeの中だけに書くことができます.
- 結果としてロジックがテンプレートとスタイル定義の両方に分散して存在することとなります.
- これがXSLT勧告で素直にコーディングした場合の限界です.

PDF5のアイディア

```
<!-- XSLTの名前空間を持たない外部の「スタイル定義ファイル」-->
<style-definition mlns="http://www.antennahouse.com/names/XSLT/Document/Layout">
    <attribute-set name="atsP">
        <sttribute-set name="space-before">0.5em</sttribute-set>
    </attribute-set>
    <attribute-set name="atsPInTable">
        <sttribute-set name="space-before">0.2em</sttribute-set>
    </attribute-set>
</style-definition>
```

スタイルはロジックと分離して記述します.

```
<!-- テンプレートのロジック -->
<xsl:template match="*[contains(@class,' topic/p ')]">
    <fo:block>
        <xsl:copy-of select="ahf:getAtttributeSet ("if ancestor::*[contains(@class,'
topic/entry ')]) then 'atsPInEntry' else 'atsP'"/>
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
```

PDF5のアイディア

```
<!-- XSLTの名前空間を持たない外部の「スタイル定義ファイル」-->
<style-definition mlns="http://www.antennahouse.com/names/XSLT/Document/Layout">
    <attribute-set name="atsP">
        <sttribute-set name="space-before">0.5em</sttribute-set>
    </attribute-set>
    <attribute-set name="atsPInTable">
        <sttribute-set name="space-before">0.2em</sttribute-set>
    </attribute-set>
</style-definition>
```

スタイルはロジックと分離して記述します.

```
<!-- テンプレートのロジック -->
<xsl:template match="*[contains(@class,' topic/p ')]">
    <fo:block>
        <xsl:copy-of select="ahf:getAtttributeSet ("if ancestor::*[contains(@class,'
topic/entry ')]) then 'atsPInEntry' else 'atsP'"/>
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
```

スタイルは動的にロジックの中で選択できます.

ahf:getAttributeSet関数

- ahf:getAttributeSetsは「スタイル定義ファイル」を参照してattribute()*を返します.
- XSLT勧告のxsl:use-attribute-setよりはるかに融通性があります.

```
<!-- テンプレートのロジック -->
<xsl:template match="*[contains(@class,' topic/p ')]">
    <fo:block>
        <xsl:copy-of select="ahf:getAtttributeSet ("if ancestor::*[contains(@class,'
topic/entry ')]) then 'atsPInEntry' else 'atsP'"/>
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
```

スタイルは動的にロジックの中で選択できます.

スタイル定義のなかの変数(1)

- 変数はXSLTのxsl:variableライクな記法で書かれます.
- 変数は名前に”$”をつけることで参照できます.
- この参照の記法はXSLT勧告よりはるかに簡単です.

```
<style-definition xmlns="http://www.antennahouse.com/names/XSLT/Document/Layout">
    <!-- Fonts -->
    <variable name="General_Serif_Font">serif</variable>
    <variable name="General_Sans_Serif_Font">sans-serif</variable>
    <variable name="General_Monospace_Font">monospace</variable>
    <!-- Body & Title Fonts Definition -->
    <variable name="General_Text_Font">$General_Serif_Font</variable>
    <variable name="General_Title_Font">$General_Sans_Serif_Font</variable>
</style-definition>
```

変数定義

変数参照

スタイル定義のなかの変数(2)

- 変数はスタイル定義(attribute)の中からも参照できます.
- 変数の利用で重複した値の定義を回避できます.

```
<!-- fo:rootのスタイル -->
<attribute-set name="atsRoot" use-attribute-sets="atsBaseFontSize
atsBaseLineHeight">
    <attribute name="xml:lang">en</attribute>
    <attribute name="font-family">$General_Text_Font</attribute>
    <attribute name="line-stacking-strategy">line-height</attribute>
    <attribute name="text-align">start</attribute>
</attribute-set>

<!-- タイトルのスタイル -->
<attribute-set name="atsTitle">
    <attribute name="font-family">$General_Title_Font</attribute>
</attribute-set>
```

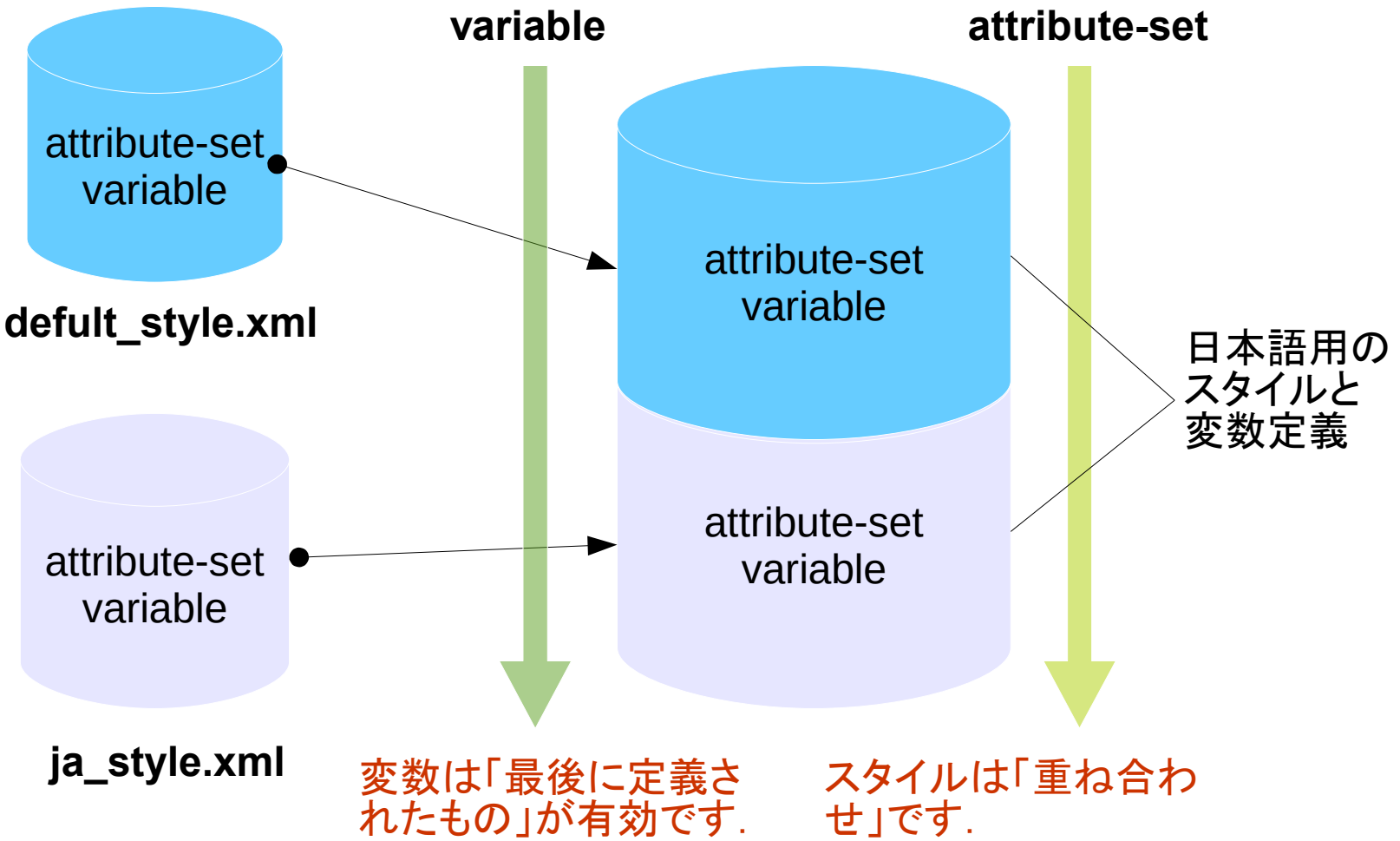
"serif"で置換されます.

"sans-serif"で置換されます.

言語別スタイル定義ファイル（1）

- 既定スタイルは英語を想定して、**config/default_style.xml**に記述します.
- 言語別スタイル定義ファイルは**config/[language_tag]_style.xml**に記述します.
- 「PDF5」の場合、ある文書の言語タグは、**bookmap/@xml:lang**から取得します.
- ２つのファイルは **default_style.xml**に対して言語別スタイル定義ファイルに優先度を与えてマージされます.

言語別スタイル定義ファイル (2)



言語別スタイル定義ファイル（3）

- 次の例の場合、bookmap/@xml:lang=”ja”ならば ja_style.xmlに書かれた変数値が有効となります.

```
<!-- default_style.xmlのフォント定義 -->
<variable name="General_Serif_Font">serif</variable>
<variable name="General_Sans_Serif_Font">sans-serif</variable>
<variable name="General_Monospace_Font">monospace</variable>

<variable name="General_Text_Font">$General_Serif_Font</variable>
<variable name="General_Title_Font">$General_Sans_Serif_Font</variable>

<!-- ja_style.xmlのフォント定義 -->
<variable name="General_Serif_Font">Times New Roman,MS 明朝</variable>
<variable name="General_Sans_Serif_Font">Arial,MS ゴシック</variable>
<variable name="General_Monospace_Font">Courier New,MS ゴシック</variable>
```

言語別スタイル定義ファイル（4）

- 次の例の場合、スタイルは合成され、font-weight =”normal”になります.

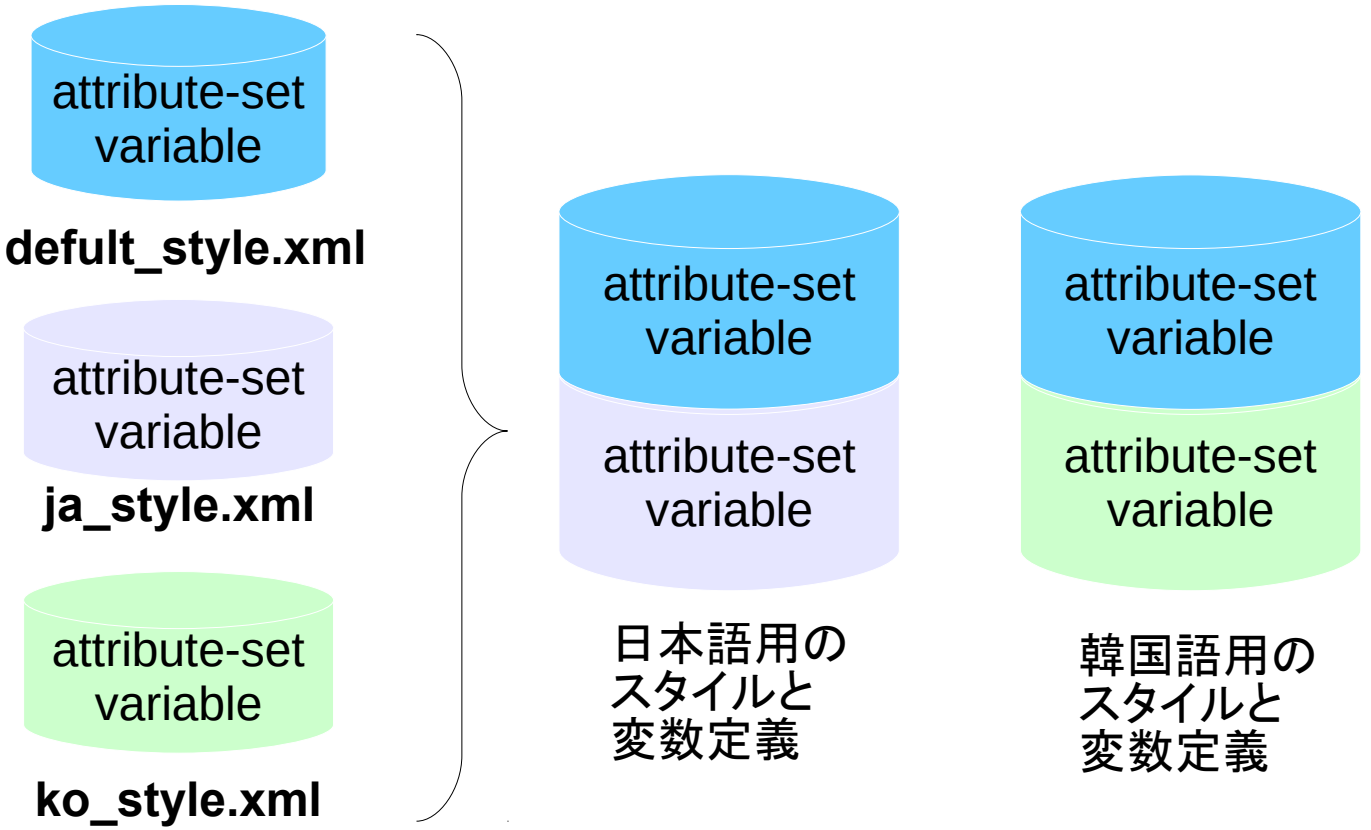
```
<!-- default_style.xmlのフォント定義 -->
<attribute-set name="atsB">
  <attribute name="font-weight">bold</attribute>
</attribute-set>

<!-- ja_style.xmlのフォント定義 -->
<attribute-set name="atsB">
  <attribute name="font-weight">normal</attribute>
</attribute-set>
```

言語別スタイル定義ファイル（5）

- 「PDF5-*ML*」の場合、文書中のすべての言語タグを集めて集約し、必要な言語別スタイル定義ファイルを選択します.
- これらのファイルは **default_style.xml**に対して言語別スタイル定義ファイルに優先度を与えてマージされます.

言語別スタイル定義ファイル (2)



※ あくまで概念的な図です.

PDF5-*ML*の制限事項

- 索引ページは日英2言語のサポートのみです.
- PDFへの変換はAH Formatterしか想定していません.
- mapから入れ子のtopicへの参照はサポートされません.
- <fn> からはXSL-FOの <fo:footnote>を生成しません.
- <image>の絶対パス指定はサポートしていません.

DITAの@xml:langの仕様と現状

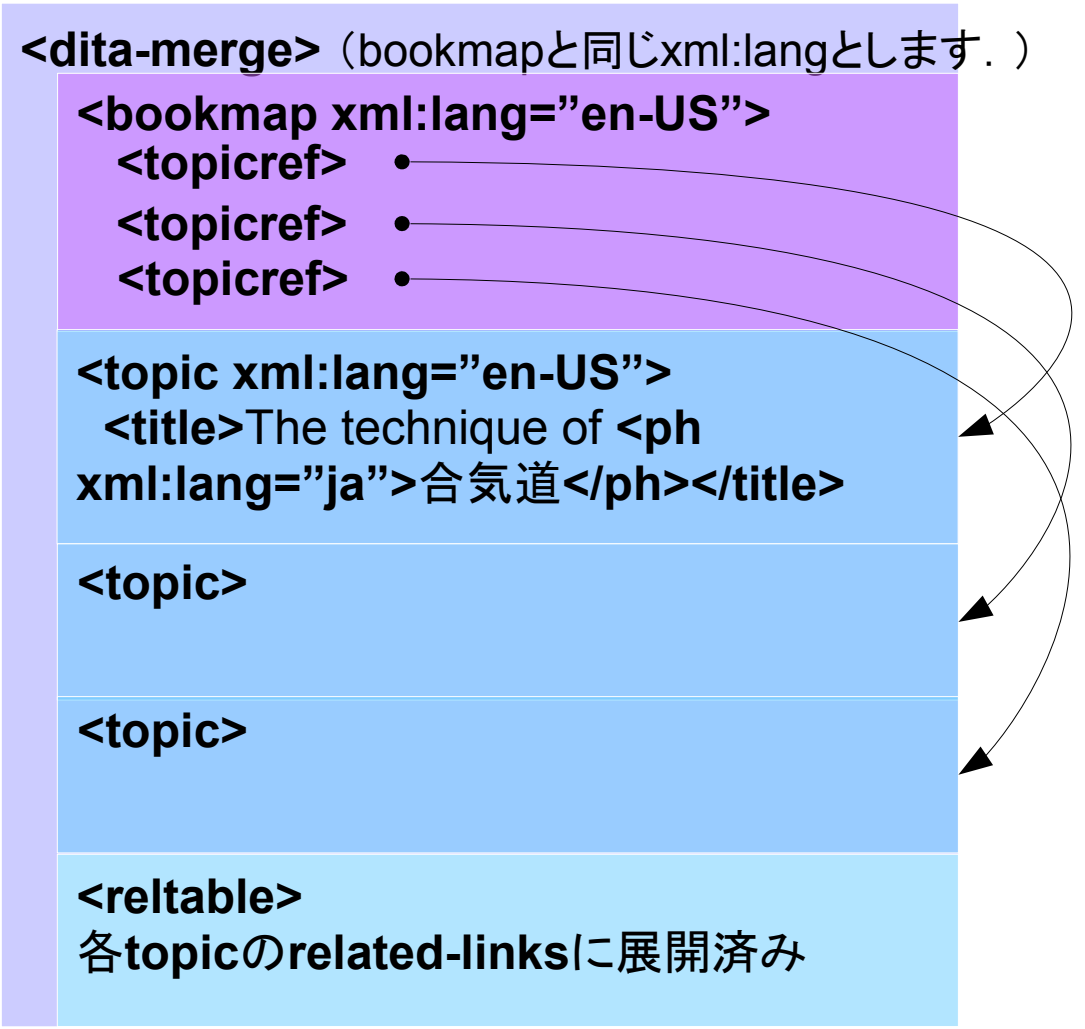
- 2.1.3.9.1 The @xml:lang attribute
  - @xml:langは要素内容の言語を指定します.
  - mapやtopicには必ず@xml:langを指定すべきです.
    - topicに指定しないとDITA-OTは”en-US”を付けてしまいます.
  - 言語が混在する場合は@xml:langで書き分けます.
  - @xml:langはmapからtopicへは継承しません.
- PDF5-*ML* 以外では、このような@xml:langの実装はされてきませんでした. PDF2では
  - 言語リテラルを@xml:langに合わせて出せるくらいです.
  - 言語別のスタイル適用は夢のまた夢でしょう.

@xml:langによるスタイル切り替え

- @xml:langの変化の検出
  - 自分と祖先の要素の@xml:langを順に取得します.
  - [言語コード(小文字)]-[国コード(大文字)]に正規化します.
  - 最後の@xml:langとその直前の@xml:langが異なっていた場合、言語が切り替えられたと判定します.
- @xml:langが切り替えられたとき
  - 自分と祖先の要素のに対して最後の@xml:langで示された言語に対するスタイルの集合を求めます.
  - このスタイルを該当の要素に適用します.

スタイル切り替えの例(1)

- DITA-OTの中間ファイル
  - PDFにpublishする際、”TopicMerge”というモジュールで作り出されます.
  - bookmapとtopicが一つのファイルにマージされます.
  - ルート要素にdita-mergeが生成されます.
- この例では
  - 英文の文書のtopicのタイトルに”合気道”という日本語のph要素が含まれています.



スタイル切り替えの例(2)

- “合気道”のphから上の@xml:langを調べる
  - “ja”→”en-US”→”en-US”の順となります.
  - “ja”≠”en-US”なので言語が切り替えられていると判定します.
- 適用するスタイルを求めます.
  - ルートから順に”ja”のスタイル(attribute()*)を取得します.
    ① dita-mergeのスタイル(fo:root用、たいてい明朝体を使用します)
    ② topicのスタイル(たぶん何もなし)
    ③ titleのスタイル(たいていゴシック体を使用します).
    ④ phのスタイル(phは汎用のインラインなので何もなし)
  - これを合成することにより、タイトル用の日本語のスタイルが”合気道”に適用されます.(例えばMS ゴシック)

テンプレートの作法（1）

- PDF5とは作法が変わります.
  - PDF5は単に名前指定（atsP）でスタイルを取得しているだけでした.

```
<!--PDF5のp要素（段落）のテンプレート-->
<xsl:template match="*[contains(@class, ' topic/p ')]">
   <fo:block>
      <xsl:copy-of select="ahf:getAttributeSet('atsP')"/>
      <xsl:call-template name="ahf:getUnivAtts"/>
      <xsl:copy-of select="ahf:getFoStyleAndProperty(.)"/>
      <xsl:apply-templates/>
   </fo:block>
</xsl:template>
```

テンプレートの作法（2）

- PDF5-*ML*ではスタイル名を取得するテンプレートを分離します.
  - これにより、目的の要素から先祖に遡ってスタイルを取得ことが可能になります.

```
<!--PDF5-MLのp要素（段落）のテンプレート-->
<xsl:template match="*[contains(@class, ' topic/p ')]" mode="MODE_GET_STYLE"
as="xs:string*">
    <xsl:sequence select="'atsP'"/>
</xsl:template>

<xsl:template match="*[contains(@class, ' topic/p ')]">
    <fo:block>
        <xsl:call-template name="getAttributeSetWithLang"/>
        <xsl:call-template name="ahf:getUnivAtts"/>
        <xsl:copy-of select="ahf:getFoStyleAndProperty(.)"/>
        <xsl:apply-templates/>
    </fo:block>
</xsl:template>
```

PDF5-*ML*カスタマイズの方法

- プラグインにcustomizationフォルダがあります.
  - ここに自分専用のカスタマイズコードを書くこともできますが、「あまり」お勧めではありません.
  - configフォルダのスタイル定義のカスタマイズに難があり、また複数の用途に使えないためです.
- お勧めは自分用のプラグインを作ることです.
  - 自分のプラグインからPDF5-*ML*のテンプレートやスタイルをimportします.

    ① テンプレートはxsl:importする側がプライオリティ（import precedence）を持ちます.

    ② スタイルはinclude要素でPDF5-*ML*の定義をインポートし、その後にカスタマイズしたスタイルを書きます. 順番が後のものが優先されます.
  - この例はGitHubにあります.
    - https://github.com/AntennaHouse/pdf5-ml/tree/master/jp.acme-corporation.pdf.ml

フォントのフォールバックの禁止

- 実はPDF2でもフォールバックを許容すればたいていの文字は出てしまいます.
  - AH Formatterがスクリプト毎に既定のフォント指定を持っているためです.
  - 特にWindows(≠10)はフォントが豊富なので、エラーメッセージさえ気にしなければ「豆腐マーク」は消えます.
- 商業出版物から見てそれで良いのでしょうか？
  - フォールバックの発生原因はたいてい、① オーサリングの誤りか、② スタイルシートのフォント指定の誤りです.
  - 従ってプロダクション用途にはフォールバックを禁止するのが本来でしょう.

言語別にスタイルを変える必要があるのでしょうか？

- あります．例えばタイ語は子音に母音や声調記号が「載り」ます．英文フォントよりフォントサイズを大きくしてバランスを取ります．

コードポイントからのフォントマッピング

- グリフは言語毎のフォントにより異なります.
- フォントはxml:langにより選択されるべきでしょう.
- つまりコードポイントから単純に一意のフォントは求められません.