によって追加されたことを示せます。

```
<CodeList
 OID="CL.AESEV"
 Name="Severity/Intensity Scale for Adverse Events"
 DataType="text"
 SASFormatName="$AESEV">
  <CodeListItem CodedValue="MILD" Rank="1">
    <Decode>
      <TranslatedText xml:lang="en">Grade 1</TranslatedText>
    </Decode>
    <Alias Name="C41338" Context="nci:ExtCodeID"/>
  </CodeListItem>
  <CodeListItem CodedValue="MODERATE" Rank="2">
    <Decode>
      <TranslatedText xml:lang="en">Grade 2</TranslatedText>
    </Decode>
    <Alias Name="C41339" Context="nci:ExtCodeID"/>
  </CodeListItem>
  <CodeListItem CodedValue="SEVERE" Rank="3">
    <Decode>
      <TranslatedText xml:lang="en">Grade 3</TranslatedText>
    </Decode>
    <Alias Name="C41340" Context="nci:ExtCodeID"/>
  </CodeListItem>
 <Alias Name="C66769" Context="nci:ExtCodeID"/>
</CodeList>
```

```
<CodeList
 OID="CL.VSTESTCD"
 Name="Vital Signs Test Code"
 DataType="text"
 SASFormatName="$VSTESTC">
   <EnumeratedItem CodedValue="BMI">
     <Alias Name="C16358" Context="nci:ExtCodeID"/>
   </EnumeratedItem>
   <EnumeratedItem CodedValue="WSTCRCM" def:ExtendedValue="Yes"/>
   <Alias Name="C66741" Context="nci:ExtCodeID"/>
</CodeList>
```

4-4. 計算方法の記述 <MethodDef>

計算ロジックはこのエレメントに記述され、ItemRef/@MethodOID から参照されます。詳細は 4-2 を参照して下さい。記述例は次の通りです。

```
<!-- Method Definition: Algorithm description -->
<MethodDef
 Type="Computation"
 OID="MT.ADSAMPLE.CHG"
```

```
 Name="Algorithm for ADSAMPLE.CHG">
 <Description>
  <TranslatedText xml:lang="en">CHG = AVAL - BASE</TranslatedText>
 </Description>
</MethodDef>
```

計算方法の記載を外部の文書に委ねることもできます。具体的な記載例は次の通りです。

TranslationText タグの内部に、計算方法は記述されていません。代わりに外部文書へのリンクを示すタグ（正確には、外部文書へのリンク定義への参照）が登場しています。

```
<!-- Method Definition: Algorithm included or expanded in an external file -->
<MethodDef
 OID="MT.EGDRVFL"
 Name="Algorithm to derive EGDRVFL"
 Type="Computation">
   <Description>
     <TranslatedText xml:lang="en">Equal to Y for derived EGTESTCDs QTCB and
QTCF </TranslatedText>
   </Description>
   <def:DocumentRef leafID="LF.ComplexAlgorithms">
     <def:PDFPageRef PageRefs="EG" Type="NamedDestination"/>
   </def:DocumentRef>
</MethodDef>

<def:leaf
 ID="LF.ComplexAlgorithms"
 xlink:href="complexalgorithms.pdf">
   <def:title>Complex Algorithms</def:title>
</def:leaf>
```

4-5. Value Level Metadata

Value Level Metadata とは**複数の異なるメタデータが一つの変数に集中している**ときに使われる手法です。具体例を見ましょう。

SCTESTCD	SCORRES	SCSTRESN	SCSTRESC
MARISTAT	Single		SINGLE
EYECOLOR	Brown		BROWN
FRAME	10	10	
MARISTAT	Married		MARRIED
EYECOLOR	Sliver		SILVER
FRAME	12	12	

これはSCドメインの一例です。様々なデータがSCORRES・SCSTRESN・SCSTRESC変数に格納されています。婚姻状態を示すデータはSCTESTCD=MARISTATの行です。聞き取りの結果として、SINGLE・MARRIEDのいずれかが格納されます。同様に目の色の調査結果は、SCTESTCD=EYECOLORの行で、取り得る値はBROWN・SILVER・BLUEのいずれかです。SCTESTCD変数のコードリストは簡単に記述できます。具体的には、MARISTAT, EYECOLOR, FRAMEとなります。ではSCSTRESC変数のコードリストはどうでしょう。データの意味を考えれば、（婚姻状態用と目の色用の）二種類のコードリストが存在します。しかし、SCSTRESC変数は一つです。すなわち、この事例ではSCTESTCD変数の値を元に、用いられるTerminologyが変わることを記載しなければなりません。これが**Value Level Metadataが用いられる場面**です[†2,3, 4]。

†2：ここではコードリストが異なる事例を示しました。しかし、Value Level Metadataの利用範囲はもっと広いものです。例えば、桁数の違いやデータ型の違い、変数ラベルの違いも表現することが出来ます。 †3：SUPP--ドメインには、さまざまな種類のデータが格納されます。すなわちValue Level Metadataの仕組みを利用する機会が多いと思われます。 †4：ここでは種類が異なるデータが1変数に集約される事例を取り上げました。これはValue Level Metadataが利用される良い事例です。しかし、この仕組みは様々な場面に応用できます。例えば、--TEST変数と --TESTCD変数は決まったペアを作ります。これをValue Level Metadataで表現することができます。もちろん、普通のコードリストとして定義しても問題ありません。いずれのアプローチもDefine.xmlとして正しく、その判断は製作者に委ねられています。しかし、一般的にValue Level MetadataはDefine.xmlを複雑にします。どうしても必要な時に限定して利用することが望ましいでしょう。

Value Level Metadataを記述するには3種類のタグを使います。1つは、何種類のパターンがあるかを示すdef:ValueListDef、詳細なメタデータを記述するItemDef（これは変数の記述で紹介済みです）、条件を示すdef:WhereClauseDefです。先ほどの事例で、この3種類のタグ使い分けを再確認しましょう。上の事例は次のように表現できます。

『SCSTRESCには2種類の変数タイプがあり、SCTESTCDの値によって、異なるTerminologyが適用される。』

これをValue Level Metadataのタグに振り分けると次にようになります。

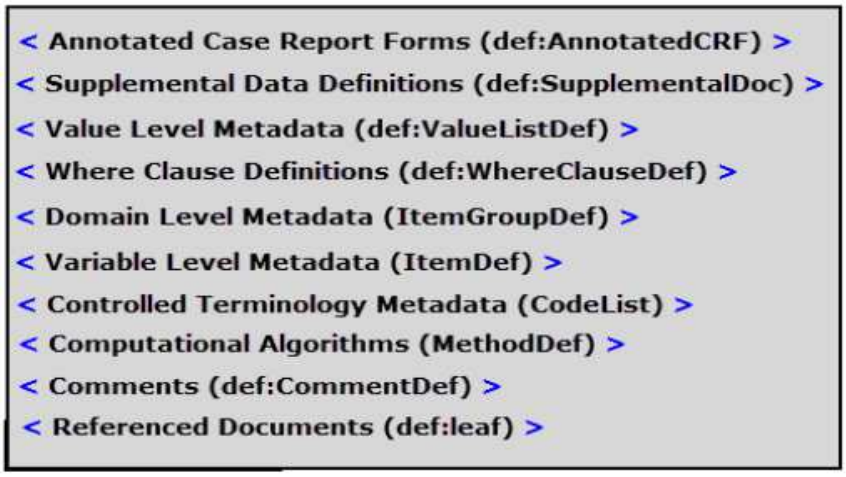
<def:ValueListDef OID="VL.SC.SCSTRESC"> <ItemRef ItemOID=VL.M> <def:WhereClauseRef WhereClauseOID="WC.M/> </ItemRef> <ItemRef ItemOID=VL.E> <def:WhereClauseRef WhereClauseOID="WC.E"/> </ItemRef>	ValueListタグの内部に二つの種類の変数タイプがある事を列挙しています。 変数タイプの詳細は記載されていません。参照先のみが提示されています。

</def:ValueListDef>	def:WhereClauseRef タグは、変数定義が適用される条件を示します。こちらも詳細は記述されていません。参照先のみが示されています。
<ItemDef OID="VL.M" Name="MARISTAT" DataType="text" Length="10" SASFieldName="Martial Status"> <Description> <TranslatedText xml:lang="en">Martial Status </TranslatedText> </Description> <CodeListRef CodeListOID="CL.MARI"/> </ItemDef> <ItemDef OID="VL.E" Name="EYE COLOR" DataType="text" Length="15" SASFieldName="Eye Color"> <Description> <TranslatedText xml:lang="en">Eye Color </TranslatedText> </Description> <CodeListRef CodeListOID="CL.EC"/> </ItemDef>	二つの変数定義の詳細が記述されています。通常の変数メタデータの記載方法が用いられます。 赤い文字の部分に注目してください。コードリスト以外にも変数長やラベルに対して、異なる定義が与えられています。
<def:WhereClauseDef OID="WC.M"> <RangeCheck SoftHard="Soft" def:ItemOID="SC.SCTESTCD" Comparator="EQ"> <CheckValue>MARISTAT</CheckValue> </RangeCheck> </def:WhereClauseDef> <def:WhereClauseDef OID="WC.E"> <RangeCheck SoftHard="Soft" def:ItemOID="SC.SCTESTCD" Comparator="EQ"> <CheckValue>EYECOLOR</CheckValue> </RangeCheck> </def:WhereClauseDef>	def:WhereClauseDef タグの中で、変数定義がスイッチされる条件を記述します。それぞれの条件には名前が付けられています。この定義は、def:ValueListRef タグの内部から参照されています。
<ItemDef OID="SC.SCSTRESC"	大元の変数である SCSTRESC 変数のメタ

Name="SCSTRESC" DataType="text" Length="200" SASFieldName="SCSTRESC"> <Description> <TranslatedText xml:lang="en">Result or Finding in Standard Units </TranslatedText> </Description> <def:ValueListRef ValueListOID="VL.SC.SCSTRESC"/> </ItemDef>	データ定義です。def:ValueListRef タグに注目してください。ここから、多重定義の情報へリンクが張られています。

Value Level Metadata は上記のような構造をしています。これらの要素の Define.xml 内での配置についても触れておきましょう。

def:ValueListDef は Define.xml 内の先頭のブロックに記載されます。XML の性質上、他のタグより前に記載されなければならないためです。続いて def:WhereClauseDef タグが記述されます。変数定義は、通常の規則に従います。参考のために、Define.xml のブロック構造の図を示しておきます。



Value Level Metadata と Variable Level Metadata の関係

以上が Value Level Metadata の記述方法です。Value Level Metadata は、個々の値に対してメタデータを定義することでした。メタデータとは、コードリスト・変数ラベル・変数長などの情報です（蛇足ですが、みなさんは条件を付けて個々の値を区別する方法も知っています）。一方、変数に対するメタデータも定義することができます。これは通常の方法です。例えば『STUDYID 変数は文字列で長さが xx である』といった具合です。もし、”変数レベルのメタデータ”と”値レベルのメタデータ”が同時に定義されたら、その結果はどのように解釈されるのでしょう？

その答えはシンプルです。値レベルのメタデータ の定義は変数レベルのメタデータ情報を上書き（正確にはオーバーライド）します。このルールは個々のメタデータに対して独立に適用されます。例え

ば、変数レベルで次のメタデータが宣言されています。

変数 Hoge: ラベル=xxx　変数タイプ＝文字　変数長＝20

これに対して、変数 Hoge の値が foo であるとき、次の値レベルのメタデータを宣言します。

変数 Hoge（Hoge=foo）：変数長＝10

この時、メタデータは次のように解釈されます。特に宣言されていないメタデータは、変数レベルのメタデータから自動的に継承されます。

変数 Hoge（Hoge=foo）： ラベル=xxx　変数タイプ＝文字　変数長＝10

4-6. コメント <def:CommentDef>

コメントは特定のブロックにまとめて記述されます。コメントにはユニークな名称が与えられ、それを目印にして Define.xml 内から参照されます。

```
<def:CommentDef OID="COM.AE.AEENDY">
  <Description>
    <TranslatedText xml:lang="en">Calculation: = AEENDTC -RFSTDTC+1 if AEENDTC
is on or after RFSTDTC. AEENDTC - RFSTDTC if AEENDTC precedes RFSTDTC
</TranslatedText>
   </Description>
</def:CommentDef>
```

外部ファイルを参照する記述も可能です。

```
<!-- Comment Definition: Long Comment, included in a PDF file -->
<def:CommentDef OID="COM.DOMAIN.QS">
 <Description>
 <TranslatedText xml:lang="en"> See attached document</TranslatedText>
 </Description>
 <def:DocumentRef leafID="LF.ReviewersGuide"/>
</def:CommentDef>

<def:leaf ID="LF.ReviewersGuide" xlink:href="reviewersguide.pdf">
 <def:title>Reviewers Guide</def:title>
</def:leaf>
```

4-7. Annotated CRF <def:AnnotatedCRF>

Annotated CRF の記述は最も最初ブロックで行われます。Annotated CRF は独立したファイルで作成されることが一般的です。そのため、def:AnnotatedCRF タグの内部から、リンク先の定義を参照します。この構造は計算方法の記述でも見られました。以下にサンプルを示します。

```
<def:AnnotatedCRF>
```

```
  <def:DocumentRef leafID="LF.blankcrf"/>
</def:AnnotatedCRF>

<def:leaf
 ID="LF.blankcrf"
 xlink:href="blankcrf.pdf">
  <def:title>Annotated Case Report Form</def:title>
</def:leaf>
```

4-8. 補足文書 <def:SupplementalDoc>

その他の補助文書の情報を格納するタグです。補助文書とはReviewer's Guideなどを指します。このブロックはAnnotated CRFのブロックの直後に配置されます。記述例は次の通りです。ここでも、外部文書を参照する形式が採られます。

```
<def:SupplementalDoc>
  <def:DocumentRef leafID="LF.FeviewersGuide"/>
</def:SupplementalDoc>

<def:leaf
 ID="LF.ReviewersGuide"
 xlink:href="reviewersguide.pdf">
  <def:title>Reviewers Guide</def:title>
</def:leaf>
```

5 あとがき

1 版あとがき

ここまでお読みいただきありがとうございます。今回は SDTM から離れ、define.xml 2.0 のドラフトを用いて解説書を作成しました。FDA は define.xml を重要なコンポーネントと位置づけています。しかし、define.xml 1.0 は規格としてクソであるばかりでなく、CDISC 公式サイトに掲載されたガイドラインに誤記があるという酷い状態にあります。2.0 になり、なんとか使えそうな仕組みになった印象です。CDISC 標準が Interoprability を実現するためには、define.xml の完成度が重要です。今後の更なる進化が期待されます。

最後にお付き合いいただいた読者の皆さんに感謝しつつ、筆を置きたいと思います。

2012 年 11 月、秋深い秋葉原にて

構成担当：TKD

執筆担当：SMZ

2 版あとがき

Define.xml 2.0.0 のドキュメントが正式にリリースされ、当サークルも解説書のバージョンアップを行いました。対応に時間がかかりましたが、趣味の活動であることを鑑みれば十分にタイムリーな更新だと考えています。2 版の作成では、導入部分を強化し、いくつかの誤記を修正しました。このテキストはサンプルの XML を取り入れたため、文字の量が多くなりました。その分、誤記も多くなっているようです。実用的な XML のサンプルは元資料を参照するのがよいでしょう。この解説書は、Define.xml の基本構造・設計概念を理解するためのものと理解いただければ幸いです。

2013 年 4 月、秋葉原にて

構成担当：TKD

執筆担当：SMZ

Define-XML 2.0　解説