HITACHI
Inspire the Next

COBOL2002　XML 連携機能ガイド

手引・操作書

3021-3-608-40

COBOL2002

前書き

■ 対象製品

P-1M36-1141 COBOL2002 Net Server Suite 04-41（適用 OS：AIX V7.1，AIX V7.2）

P-1M36-2141 COBOL2002 Net Server Runtime 04-41（適用 OS：AIX V7.1，AIX V7.2）

P-1M36-1241 COBOL2002 Net Server Suite(64) 04-41（適用 OS：AIX V7.1，AIX V7.2）

P-1M36-2241 COBOL2002 Net Server Runtime(64) 04-41（適用 OS：AIX V7.1，AIX V7.2）

P-2636-2344 COBOL2002 Net Developer 04-70（適用 OS：Windows 7，Windows 8.1，Windows 10，Windows 11，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2436-5344 COBOL2002 Net Server Runtime 04-70（適用 OS：Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2636-3344 COBOL2002 Net Client Runtime 04-70（適用 OS：Windows 7，Windows 8.1，Windows 10，Windows 11）

P-2436-6344 COBOL2002 Net Server Suite 04-70（適用 OS：Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2636-4344 COBOL2002 Net Client Suite 04-70（適用 OS：Windows 7，Windows 8.1，Windows 10，Windows 11）

P-2936-2344 COBOL2002 Net Developer(64) 04-70（適用 OS：Windows 7(x64)，Windows 8.1(x64)，Windows 10(x64)，Windows 11，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2936-5344 COBOL2002 Net Server Runtime(64) 04-70（適用 OS：Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2936-6344 COBOL2002 Net Server Suite(64) 04-70（適用 OS：Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-9W36-1241 COBOL2002 Net Server Suite(64) 04-60（適用 OS：Linux Server 7 (64-bit x86_64)，Linux Server 8 (64-bit x86_64)，Linux Server 9 (64-bit x86_64)）

P-9W36-2241 COBOL2002 Net Server Runtime(64) 04-60（適用 OS：Linux Server 7 (64-bit x86_64)，Linux Server 8 (64-bit x86_64)，Linux Server 9 (64-bit x86_64)）

P-2636-7344 COBOL2002 Developer Professional 04-70（適用 OS：Windows 7，Windows 8.1，Windows 10，Windows 11，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

P-2936-7344 COBOL2002 Developer Professional(64) 04-70（適用 OS：Windows 7(x64)，Windows 8.1(x64)，Windows 10(x64)，Windows 11，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019，Windows Server 2022）

これらのプログラムプロダクトのほかにもこのマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

■ 発行

2024 年 1 月　3021-3-608-40

変更内容

変更内容（3021-3-608-40） COBOL2002 Net Developer 04-70，COBOL2002 Net Server Runtime 04-70，COBOL2002 Net Client Runtime 04-70，COBOL2002 Net Server Suite 04-70，COBOL2002 Net Client Suite 04-70，COBOL2002 Net Developer(64) 04-70，COBOL2002 Net Server Runtime(64) 04-70，COBOL2002 Net Server Suite(64) 04-70，COBOL2002 Developer Professional 04-70，COBOL2002 Developer Professional(64) 04-70

追加・変更内容	変更個所
Windows COBOL2002 で，XML 連携機能が生成する COBOL 原始プログラムをコンパイルする場合の制限事項に，次のコンパイラオプションを追加した。 • -VOSCBL,AssignDataToDevice • -VOSCBL,EvaluateWhenOther	6.2.1，付録 C.2

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは，次に示すプログラムプロダクトの XML 連携機能について説明したものです。

- P-1M36-1141 COBOL2002 Net Server Suite
- P-1M36-2141 COBOL2002 Net Server Runtime
- P-1M36-1241 COBOL2002 Net Server Suite(64)
- P-1M36-2241 COBOL2002 Net Server Runtime(64)
- P-2636-2344 COBOL2002 Net Developer
- P-2436-5344 COBOL2002 Net Server Runtime
- P-2636-3344 COBOL2002 Net Client Runtime
- P-2436-6344 COBOL2002 Net Server Suite
- P-2636-4344 COBOL2002 Net Client Suite
- P-2936-2344 COBOL2002 Net Developer(64)
- P-2936-5344 COBOL2002 Net Server Runtime(64)
- P-2936-6344 COBOL2002 Net Server Suite(64)
- P-9W36-1241 COBOL2002 Net Server Suite(64)
- P-9W36-2241 COBOL2002 Net Server Runtime(64)
- P-2636-7344 COBOL2002 Developer Professional
- P-2936-7344 COBOL2002 Developer Professional(64)

■ 対象読者

COBOL2002 の XML 連携機能を使って，XML ドキュメントを操作する COBOL プログラムを作成したい方を対象としています。また，COBOL2002 と XML の知識があり，UNIX または Windows の基本操作について理解していることを前提としています。

■ このマニュアルで使用する記号

このマニュアルで使用する記号を次に示します。

記号	意味
{ }	この記号で囲まれている複数の項目のうちから一つを選択することを意味する。項目が縦に複数行にわたって記述されている場合は，そのうちの 1 行分を選択する。
[ ]	この記号で囲まれている項目は省略してもよいことを意味する。

記号	意味
	複数の項目が縦または横に並べて記述されている場合には，すべてを省略するか，記号 { } と同じく，どれか一つを選択する。
…	記述が省略されていることを意味する。 この記号の直前に示された項目を繰り返して複数個指定できる。
（下線）	括弧で囲まれた複数の項目のうち 1 項目に対して使用され，括弧内のすべてを省略したときにシステムがとる標準値を意味する。
|	横に並べられた複数の項目に対して項目間の区切りを示し，「または」を意味する。
（*斜体文字*）	ユーザが値を指定することを意味する。
[ ]	ウィンドウのメニューバーから選択するメニュー，またはコマンドを意味する。

目次

1 XML 連携機能の概要

この章では，XML 連携機能の概要，および XML 連携機能を使ったアプリケーション開発の流れについて説明します。

1.1 XML 連携機能とは

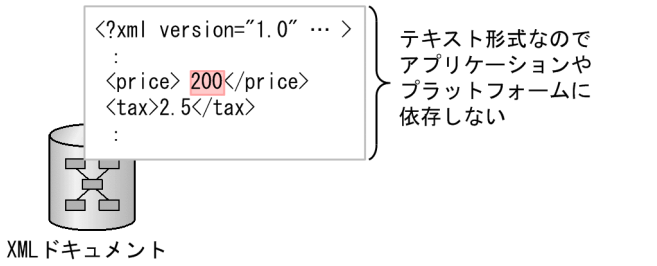
1.1.1 XML とは

XML（eXtensible Markup Language）は，World Wide Web Consortium（W3C）によって標準化されている，文章の構造を定義するための言語です。

XML では，ユーザがタグと呼ばれる文字列をドキュメントに埋め込んで，データの意味づけをします。このとき，独自のタグをドキュメントに埋め込んで意味づけし，さらに，タグ同士を入れ子にして論理的な階層構造を持つドキュメントを記述できます。

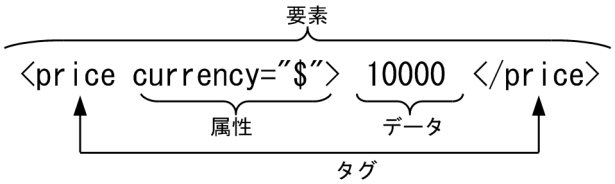
また，XML ドキュメントは，テキスト形式のデータとして作成するのでプラットフォームに依存しません。このため，インターネットを利用した企業間のデータ交換フォーマットなどに利用されています。

図 1-1 XML の特長



独自のタグによってデータを意味づけられる。
上記の例の場合，200 が価格を表しているのがタグから判断できる。

XML ドキュメントは，要素と呼ばれるタグ付きデータの集合から構成されています。おのおのの要素は，要素を意味づけるタグ，要素の内容であるデータ，および要素を修飾する属性から構成されています。



1.1.2 XML 連携機能

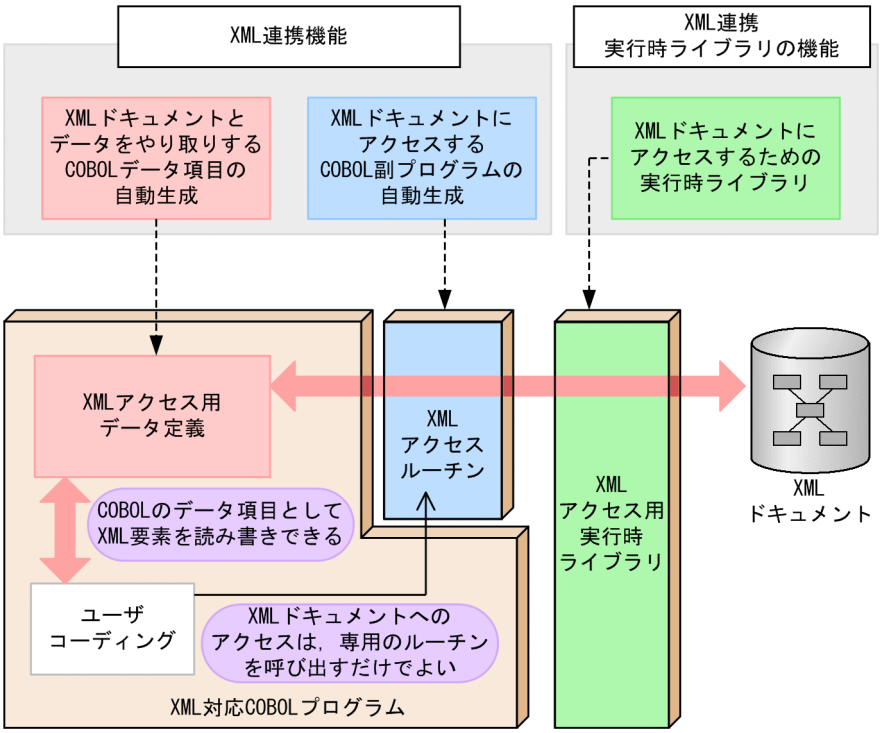
XML 連携機能は，XML ドキュメントにアクセスする COBOL プログラム（以降，XML 対応 COBOL プログラムと呼びます）を作成するために使用します。

XML 連携機能を使うと，COBOL のデータ項目と XML ドキュメント中の XML 要素を対応づけられます。COBOL プログラムでは，XML 要素に対応づけられた COBOL のデータ項目（XML アクセス用データ定義）を操作することで，XML 要素の値を読み込んだり，XML 要素に値を書き出したりできます。また，実際の XML ドキュメントへアクセスするには，XML 連携機能が生成する副プログラム（XML アク

セスルーチン）を呼び出します。XML アクセスルーチンは，XML 連携機能の実行時ライブラリを使って，XML ドキュメントへアクセスします。

図 1-2 に，XML 連携機能および XML 連携機能の実行時ライブラリの提供する機能について示します。

図 1-2　XML 連携機能および XML 連携実行時ライブラリの機能



1.1.3　処理できる XML ドキュメント

COBOL2002　XML 連携機能では，W3C で勧告された XML 1.0 に従った XML ドキュメントを処理できます。

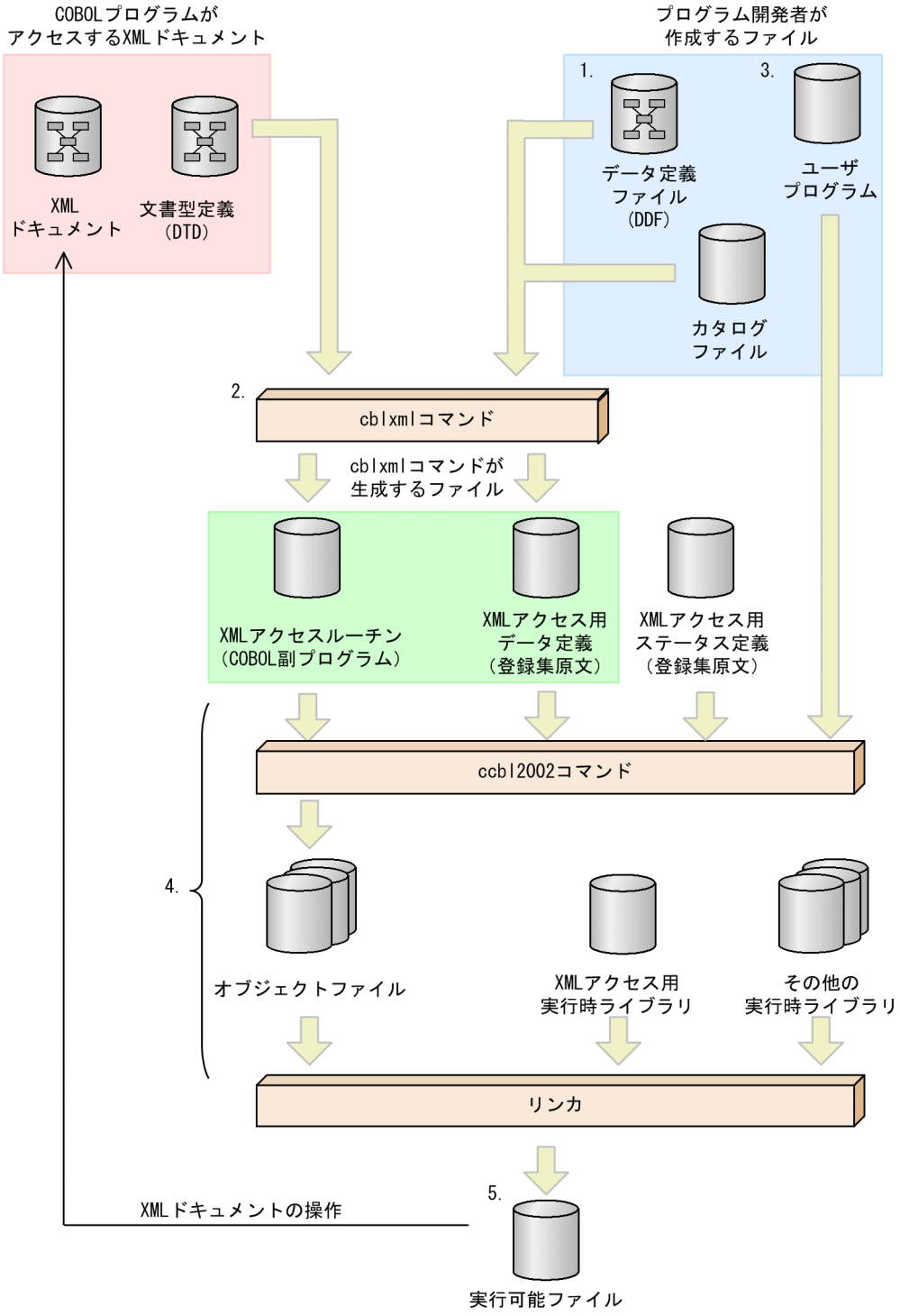
ただし，XML 連携機能で XML ドキュメントを出力する場合，XML ドキュメント内に文書の型宣言（DOCTYPE 宣言）を出力しません。

1.2　XML 連携機能を使ったプログラム開発の概要

ここでは，XML 連携機能を使って XML 対応 COBOL プログラムを作成する流れを説明します。

XML 対応 COBOL プログラムの作成手順を，図 1-3 に示します。

図 1-3　XML 対応 COBOL プログラムの作成手順



1. XML 要素と COBOL データ項目を対応づける

COBOL プログラムからアクセスしたい XML 要素を，データ定義ファイル（Data Definition File）を使って COBOL のデータ項目と対応づけます。この手順の詳細については，「2. XML 要素と COBOL データ項目の対応づけ」を参照してください。

また，入出力データ情報定義機能を使用すると，XML データに入出力する値の型，データ長などを詳細に定義できます。入出力データ情報定義機能の詳細については，「3. 入出力データ情報定義機能」を参照してください。

2. XML アクセスルーチンと XML アクセス用データ定義を生成する

cblxml コマンドを使って，XML ドキュメントの文書型定義（DTD）と Data Definition File（DDF）から XML アクセスルーチンと XML アクセス用データ定義を生成します。この手順の詳細については，「4. XML アクセスルーチンと XML アクセス用データ定義の生成」を参照してください。

3. XML アクセスルーチンを使用した COBOL プログラムを作成する

XML アクセスルーチンを呼び出して XML ドキュメントにアクセスするユーザプログラムをコーディングします。この手順の詳細については，「5. XML アクセスルーチンを使用した COBOL プログラムの作成」を参照してください。

4. コンパイル，リンケージをする

ユーザプログラム，XML アクセスルーチン，XML アクセス用データ定義のコンパイル，リンケージをして，実行可能ファイルを作成します。この手順の詳細については，「6. コンパイルとリンケージ」を参照してください。

5. プログラムを実行する

完成した実行可能ファイルを実行します。この手順の詳細については，「7. 実行」を参照してください。

Windows の場合，COBOL2002 の開発マネージャから XML アクセスルーチンと XML アクセス用データ定義の生成，および XML 対応 COBOL プログラムのコンパイル，リンケージができます。詳細については，「8. 開発マネージャ連携（Windows の場合）」を参照してください。

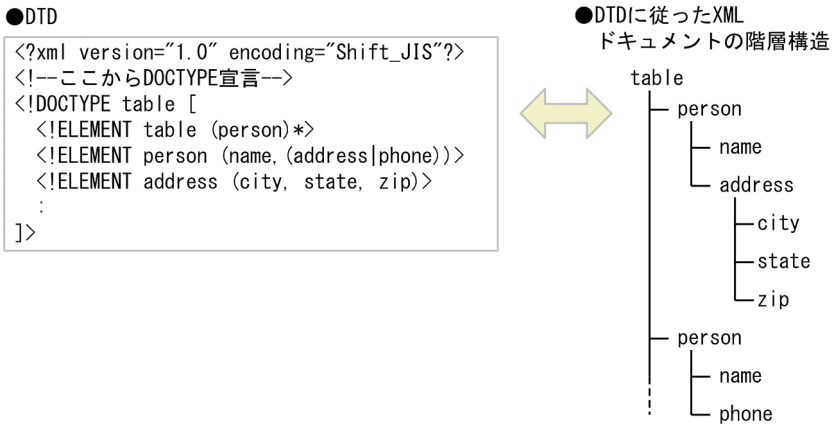
2 XML 要素と COBOL データ項目の対応づけ

XML 連携機能では，XML ドキュメントの文書型定義（DTD）中の要素を COBOL データ項目に対応づけることで，COBOL プログラムから XML ドキュメントにアクセスできるようにします。

この章では，DTD の概要，DTD から必要な要素を抽出して COBOL データ項目を生成するためのデータ定義ファイル（DDF）の概要，および DDF に記述するデータ定義言語（DDL）の文法について説明します。

2.1　文書型定義（DTD）の概要

XML では，それぞれのドキュメントに対して文書型定義（DTD）と呼ばれるドキュメントの構造定義を宣言できます。DTD を使うと，XML ドキュメントの構造の解析，検証などができます。



2.1.1　DTD の有無と XML ドキュメントの種類

XML ドキュメントは，対応する DTD が定義されているかどうかによって，次の 2 種類に分類されます。

- 正しい形式の XML ドキュメント（well-formed XML document）
  XML としてのドキュメント形式（要素の親子関係，タグの記述など）が正しい XML ドキュメントを指します。DTD の有無は問いません。
- 妥当な XML ドキュメント（valid XML document）
  正しい形式の XML ドキュメントであり，かつ DTD が定義されている XML ドキュメントを指します。

XML 連携機能では，COBOL プログラムからアクセスしたい XML ドキュメントの構造を参照するために DTD を利用します。アクセス対象となる XML ドキュメントは，DTD に対して妥当な XML ドキュメントである必要があります。

アクセス対象となる XML ドキュメントは，XML ドキュメントが正しい形式であるための制約（well-formedness constraint）を満たすために必要な場合を除き，DTD を含んでいてもいなくてもかまいません。ただし，入力 XML ドキュメントの妥当性をチェックする場合は，アクセス対象となる XML ドキュメントに DTD がなければなりません。入力 XML ドキュメントの妥当性チェック機能については，「7.2.8 入力 XML ドキュメントの妥当性チェック機能」を参照してください。

2.1.2　XML 連携機能で扱える DTD の形式

XML 連携機能では，XML Version 1.0 の規格に従った DTD を扱えます。ただし，次の点に注意する必要があります。

(1) XML 宣言

XML 宣言では，DTD ファイルの文字エンコーディングを指定できます。使用できる文字エンコーディングについては，「付録 G.1 XML ドキュメントの文字エンコーディング」を参照してください。

- 使用できる文字エンコーディング以外を指定した場合
  動作は保証しません。
- 文字エンコーディングを省略した場合
  "UTF-8"が仮定されます。

XML 宣言の例

```
<?xml version="1.0" encoding="Shift_JIS"?>
```

(2) 文書の型宣言（DOCTYPE 宣言）

文書の型宣言（DOCTYPE 宣言）は，内部サブセット，外部サブセットのどちらでも定義できます。

内部サブセット

内部サブセットは，DOCTYPE 宣言の記述されている XML ドキュメント中に，型宣言の実体を記述する形式です。

内部サブセットを使った DOCTYPE 宣言の例

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (book)>
  <!ELEMENT book  (#PCDATA)>
]>
<table/>
```

外部サブセット

外部サブセットは，外部 DTD ファイルに型宣言の実体を記述し，XML ドキュメントの DOCTYPE 宣言には外部 DTD ファイルへの参照指示を記述する形式です。

外部 DTD ファイルは，外部識別子で指定します。外部識別子については，「付録 E.3 外部識別子の解釈」を参照してください。

外部サブセットを使った DOCTYPE 宣言の例

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table SYSTEM "book-content.xml">
<table/>
```

外部 DTD ファイル（book-content.xml）の指定例

```
<!ELEMENT table (book)>
<!ELEMENT book  (#PCDATA)>
```

内部サブセットと外部サブセットが同時に指定されている場合，両方の定義が有効となります。ただし，要素が重複して定義されている場合は，内部サブセットの定義が有効となります。

(3) 要素宣言

要素宣言の内容モデルでは，#PCDATA と EMPTY だけを使用できます。

要素宣言の例

```
<!ELEMENT table (book)+>
<!ELEMENT book  (#PCDATA)>
```

(4) 属性リスト宣言

属性リスト宣言に指定する属性の型（CDATA，ID，IDREF，IDREFS，NMTOKEN，NMTOKENS，列挙）以外は使用できません。

(5) 実体宣言

実体宣言は，XML ドキュメントで実体を置換します。外部実体となる場合，テキスト宣言でエンコーディングを指定します。テキスト宣言がない場合はエンコーディングに UTF-8 が仮定されます。テキスト宣言については，「(7) テキスト宣言」を参照してください。

一般実体

パースされるテキストを置換します。

一般実体「ITEM1」を定義します。

例

```
<!ENTITY ITEM1 ”The item 01.”>
```

外部一般実体

外部にあるテキストを含んだ実体です。

公開識別子でデータを指定する例

実体「abc」を定義します。

```
<!ENTITY abc PUBLIC ”-//HITACHI//DTD test 1.0//EN” ”/home/groupX/manual.dtd”>
```

データを特定の位置として指定する例

実体「def」を定義します。

```
<!ENTITY def SYSTEM ”/home/groupX/manual.dtd”>
```

パラメタ実体

パラメタ実体は DTD 内のテキストを置換します。

実体「para1」を定義します。

例

```
<!ENTITY % para1 "(item01A | item01B )">
```

外部パラメタ実体

外部パラメタ実体は，DTD を含む実体や，外部にある DTD の一部を含む実体を示します。

公開識別子でデータを指定する例

実体「paragroup1」を定義します。

```
<!ENTITY % paragroup1 PUBLIC "-//HITACHI//DTD test 1.0//EN" "/home/groupX/table1.dtd">
```

データを特定の位置として指定する例

実体「paragroup2」を定義します。

```
<!ENTITY % paragroup2 SYSTEM "/home/groupX/table2.dtd">
```

(6) 公開識別子

外部 DTD や外部実体を識別するための文字列です。XML 連携機能では，公開識別子と，外部 DTD や外部実体を含むファイルの対応を，カタログファイルで定義することで，公開識別子を用いて外部 DTD や外部実体を参照する XML ドキュメントを処理できます。

公開識別子の解釈については，「付録 E.3　外部識別子の解釈」を参照してください。

外部 DTD の例

```
<!DOCTYPE root PUBLIC "-//HITACHI//DTD test 1.0a//EN" "/home/project1/myDTD.xml">
```

(7) テキスト宣言

テキスト宣言では，外部解析対象実体の文字エンコーディングを指定できます。

例

```
<?xml version="1.0" encoding="Shift_JIS"?>
```

(8) コメント

形式

```
<!-- コメント -->
```

機能

コメントは"<!--"で始まり，"-->"で終わる間に記述します。

規則

コメントは XML ドキュメントに出力されません。

DTD ファイルのコメントの例

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!-- コメント -->
<!DOCTYPE table [
  <!ELEMENT table (book)>
  <!ELEMENT book  (#PCDATA)>
]>
<table/>
```

(9) その他の注意事項

DTD ファイルは，それ自身が完結した XML ドキュメントである必要があります。そのため，DTD ファイルに XML 要素が記述されていない場合は，内容が空のルート要素をファイルの末尾に付ける必要があります。

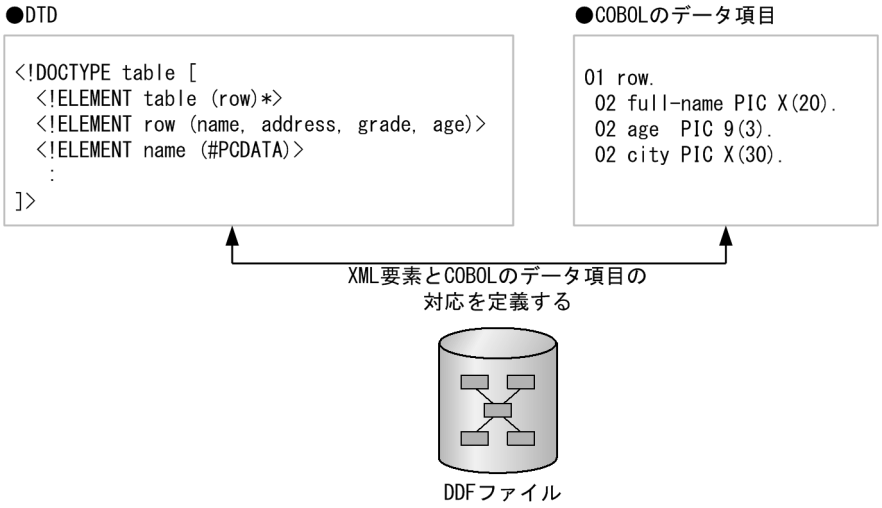
XML ドキュメントとして完結した DTD ファイルの例

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (book)>
  <!ELEMENT book  (#PCDATA)>
]>
<table/>
```

2.2　データ定義ファイル（DDF）の作成

XML ドキュメントの要素は，データを常に文字列として持っているため，DTD だけではそれぞれの要素がどの種類の COBOL データ項目に対応するかわかりません。そのため，XML 要素と COBOL データ項目とを結び付ける対応情報を定義する必要があります。

データ定義ファイル（DDF）は，XML ドキュメントの DTD で定義された XML 要素のうち，COBOL プログラムからアクセスしたい要素と COBOL データ項目との対応づけを記述するファイルです。



DDF では，データ定義言語（DDL）を使って XML 要素と COBOL データ項目とを対応づけます。DDL の記述方法については，「2.3　データ定義言語（DDL）の文法」を参照してください。

DDF の形式

<?xml version="1.0" encoding="Shift_JIS"?>

データ定義言語（DDL）

例

DTD 中の要素「row」の要素「address」に含まれる要素「name」および「age」を COBOL データ項目に対応づける DDF を次に示します。

（DTD の例）

```
<!DOCTYPE table [
  <!ELEMENT table (row)*>
  <!ELEMENT row (name, address, grade, age)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (addr1, addr2, city, state, zip)>
  <!ELEMENT addr1 (#PCDATA)>
  <!ELEMENT addr2 (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT state (#PCDATA)>
  <!ELEMENT zip (#PCDATA)>
  <!ELEMENT grade (#PCDATA)>
  <!ELEMENT age (#PCDATA)>
```

```
]>
<table/>
```

(DDF の例)

```
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="row">
    <Group cobName="row">
      <Item elemName="name" cobName="full-name"
            type="alphanumeric" size="20"/>
      <Item elemName="age" type="numeric"
            size="3"/>
      <Item elemName="city" type="alphanumeric"
            size="30"/>
    </Group>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目の例)

```
01 row.
  02 full-name PIC X(20).
  02 age  PIC 9(3).
  02 city PIC X(30).
```

2.3　データ定義言語（DDL）の文法

ここでは，データ定義ファイル（DDF）に記載するデータ定義言語（DDL）の文法について説明します。

DDL の形式

DDL は，XML の文法規則に従って記述します。DDL の記述例を次に示します。

（DDL の記述例）

```
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName=" ・・・
     :
 </BaseElement>
</Interface>
```

DDL で使用する要素

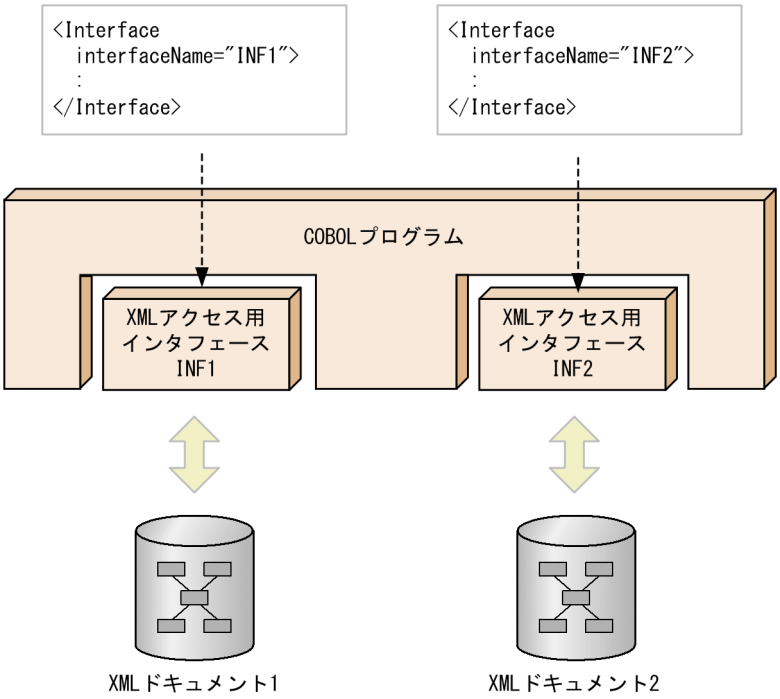
DDL で使用する要素を，表 2-1 に示します。

表 2-1　DDL で使用する要素

要素名	機能
Interface	COBOL プログラムと XML とを対応づけるインタフェースを宣言する。
BaseElement	アクセス対象とする XML 要素を指定する。
Group	XML 要素を COBOL 集団項目に対応づける。
Item	XML 要素を COBOL データ項目に対応づける。
Array	XML の繰り返し要素を，OCCURS 句付きの COBOL データ項目に対応づける。
AttrItem	XML 要素の属性を COBOL データ項目に対応づける。

2.3.1　Interface 要素（インタフェースの定義）

Interface 要素は，COBOL プログラムと XML とを対応づけるインタフェースを宣言する要素です。Interface 要素を定義すると，XML ドキュメントにアクセスするための COBOL 副プログラム（XML アクセスルーチン）や登録集原文（XML アクセス用データ定義）などのインタフェースが対応して生成されます。



Interface 要素は，ドキュメントのルート（最上位の要素）として DDL 中に 1 回だけ記述できます。また，子要素として一つ以上の BaseElement 要素を持つ必要があります。

形式

<Interface interfaceName="*インタフェース名*"［accessInfo="*yes*|*no*"］>
BaseElement 要素 …
</Interface>

(1) interfaceName 属性

形式

interfaceName="*インタフェース名*"

機能

Interface 要素に対応するインタフェース名を指定します。インタフェース名は，Interface 要素に対応して生成される XML アクセスルーチンのプログラム名の一部に使用されます。XML アクセスルーチンの名称については，「4.2.1　XML アクセスルーチンの名称形式」を参照してください。

規則

- インタフェース名称は，COBOL プログラム名に指定できる次の文字で構成する必要があります。
  英文字（A～Z，a～z），数字（0～9），ハイフン（-），下線（_），#，¥，@，日本語文字
- インタフェース名は，ほかのインタフェース名と重複しない一意な名称を指定する必要があります。
- インタフェース名と，BaseElement 要素の cobName 属性（省略時は elemName 属性）に指定した名称の長さの合計は，19 バイト以下の文字列で指定する必要があります。19 バイトを超える名称を指定した場合，動作は保証しません。

指定例

インタフェース「EXAMPLE」を宣言します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (person)*>
  <!ELEMENT person (name, address)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="person">
  <Group elemName="person">
   <Item elemName="name" cobName="FULLNAME"
         type="alphanumeric" size="30"/>
   <Item elemName="address" cobName="FULLADDRESS"
         type="alphanumeric" size="120"/>
  </Group>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目の例）

```
01 person.
 02 FULLNAME PIC X(30).
 02 FULLADDRESS PIC X(120).
```

（XML アクセスルーチンの呼び出し例）

```
  :
CALL 'CBLXML-OP-EXAMPLE'
    USING XML-FILE-NAME-POINTER XML-FILE-LENGTH
          XML-FILE-MODE XML-POINTER
    RETURNING CBLXML-RETURN-CODE.
  :
```

(2)　accessInfo 属性

形式

accessInfo="*yes*|*no*"

機能

Interface 要素の下位の要素で入出力データ情報定義機能を使用するかどうか指定します。入出力データ情報定義機能については，「3.　入出力データ情報定義機能」を参照してください。

規則

- accessInfo 属性の指定を省略した場合は，"no"が仮定されます。
- Interface 要素と BaseElement 要素の両方に accessInfo 属性を指定した場合，BaseElement 要素の accessInfo 属性値が優先されます。
- accessInfo 属性に"yes""no"以外の値を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- accessInfo 属性と emptyValue 属性を同時に指定した場合，accessInfo 属性の指定が優先されます。

指定例

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (item01, item02)>
  <!ELEMENT item01  (#PCDATA)>
  <!ELEMENT item02  (itema, itemb)*>
  <!ELEMENT itema   (#PCDATA)>
  <!ELEMENT itemb   (#PCDATA)>
]>
<root/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE"
   accessInfo="yes">
  <BaseElement elemName="root">
    <Group cobName="root">
      <Item elemName="item01" type="alphanumeric"
            size="10" />
      <Array max="10">
        <Group cobName="item02">
          <Item elemName="itema"
                type="alphanumeric" size="10" />
          <Item elemName="itemb"
                type="alphanumeric" size="10" />
        </Group>
      </Array>
    </Group>
  </BaseElement>
</Interface>
```

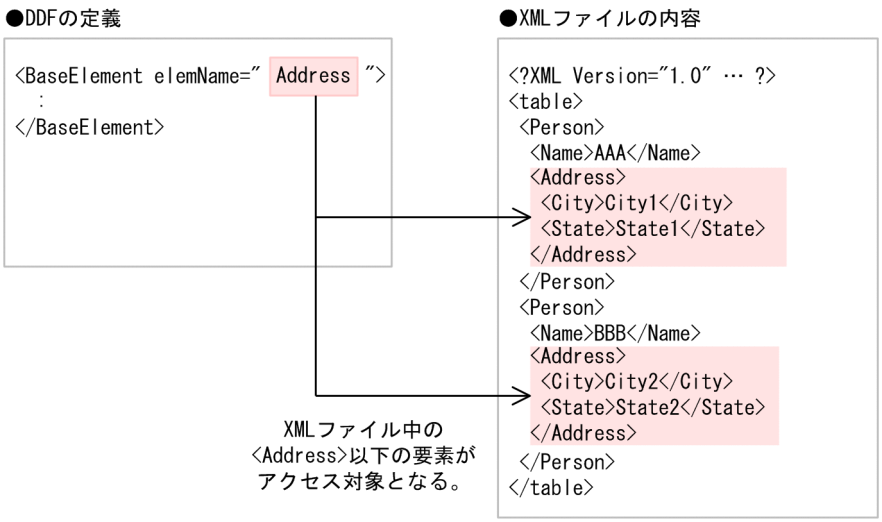
（生成される COBOL データ項目）

```
 01 root-BASE.
  02 root.
   03 item01 PIC X(10).
   03 item02 OCCURS 10.
    04 itema PIC X(10).
    04 itemb PIC X(10).
* Access Information
  02 root-GROUP.
```

```
   03 item01-FLG PIC 1(32) USAGE BIT.
   03 item01-LEN PIC 9(9) COMP.
   03 item02-TOTAL PIC 9(9) COMP.
   03 item02-COUNT PIC 9(9) COMP.
   03 item02-GROUP OCCURS 10.
    04 itema-FLG PIC 1(32) USAGE BIT.
    04 itema-LEN PIC 9(9) COMP.
    04 itemb-FLG PIC 1(32) USAGE BIT.
    04 itemb-LEN PIC 9(9) COMP.
```

2.3.2 BaseElement 要素（アクセスする要素の定義）

BaseElement 要素は，COBOL プログラムから読み出し，および書き込みをする XML 要素を指定する要素です。BaseElement 要素によって COBOL データ項目と対応づけられた XML 要素は，COBOL プログラムから XML アクセスルーチンを使ってデータにアクセスできます。XML アクセスルーチンの詳細については，「4.2　生成される XML アクセスルーチン」を参照してください。



BaseElement 要素は，子要素として Group 要素，Item 要素，または AttrItem 要素のどれか一つを持ちます。

形式

<BaseElement elemName="*XML 要素の名称*"
[cobName="*XML アクセスルーチンの名称*"]
[accessInfo="*yes*|*no*"]
[nameOfBaseVar="*XML アクセス用データ定義の名称*"] >
{*Group 要素* | *Item 要素* | *AttrItem 要素*}
</BaseElement>

規則

DTD 中の複数の要素を BaseElement 要素と対応づけた場合，対応するアクセスルーチンの呼び出し順序は，次のとおりでなくてはなりません。

（XML ドキュメントの入力および更新の場合）

BaseElement 要素と対応づけた要素が実際の XML ドキュメント中に出現する順に，対応するアクセスルーチンを使用して入力（更新）してください。

（XML ドキュメントの出力の場合）

DTD で定義された順序に従った妥当な XML ドキュメントとなるように，対応するアクセスルーチンを使用して出力してください。

(1) elemName 属性

形式

elemName="*XML の要素名*"

機能

BaseElement 要素に対応づける XML の要素名を指定します。XML の要素名は，BaseElement 要素に対応して生成される XML アクセスルーチンのプログラム名の一部に使用されます。XML アクセスルーチンの名称については，「4.2.1　XML アクセスルーチンの名称形式」を参照してください。

規則

- XML の要素名称は，XML で規定された文字で構成する必要があります。
- XML の要素名称とインタフェース名称の長さの和が 19 バイトを超える場合は，cobName 属性を使って 19 バイト以下の名称を指定する必要があります。cobName 属性を指定しないと，COBOL 原始プログラムの生成時にエラーとなります。

(2) cobName 属性

形式

cobName="*XML アクセスルーチンの名称*"

機能

elemName 属性の値の代わりに XML アクセスルーチンの COBOL プログラム名に使用する名称を指定します。elemName 属性に指定する XML 要素の名称が，COBOL プログラム名に使用できない場合などに指定します。XML アクセスルーチンの名称については，「4.2.1　XML アクセスルーチンの名称形式」を参照してください。

規則

- cobName 属性に指定する名称は，COBOL のプログラム名として使用できる名称である必要があります。COBOL のプログラム名として使用できない名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

- cobName 属性の名称は，19 バイト以下の文字列で指定する必要があります。19 バイトを超える名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

(3) accessInfo 属性

形式

accessInfo="*yes*|*no*"

機能

BaseElement 要素の下位の要素で入出力データ情報定義機能を使用するかどうかを指定します。入出力データ情報定義機能については，「3. 入出力データ情報定義機能」を参照してください。

規則

- accessInfo 属性の指定を省略した場合は，Interface 要素の accessInfo 属性値が仮定されます。
- Interface 要素と BaseElement 要素の両方に accessInfo 属性を指定した場合，BaseElement 要素の accessInfo 属性値が優先されます。
- accessInfo 属性に"yes""no"以外の値を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- accessInfo 属性と emptyValue 属性を同時に指定した場合，accessInfo 属性の指定が優先されます。

指定例

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (group01, group02)>
  <!ELEMENT group01 (item01)>
  <!ELEMENT item01  (#PCDATA)>
  <!ELEMENT group02 (item02)>
  <!ELEMENT item02  (#PCDATA)>
]>
<root/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="group01"
      accessInfo="yes">
    <Group elemName="group01">
      <Item elemName="item01" type="alphanumeric"
            size="10" />
    </Group>
  </BaseElement>
  <BaseElement elemName="group02"
      accessInfo="no">
    <Group elemName="group02">
      <Item elemName="item02" type="alphanumeric"
            size="10" />
```

```
    </Group>
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
 01 group01-BASE.
  02 group01.
   03 item01 PIC X(10).
* Access Information
  02 group01-FLG PIC 1(32) USAGE BIT.
  02 group01-GROUP.
   03 item01-FLG PIC 1(32) USAGE BIT.
    03 item01-LEN PIC 9(9) USAGE COMP.
 01 group02.
  02 item02 PIC X(10).
```

(4)　nameOfBaseVar 属性

形式

nameOfBaseVar="*XML アクセス用データ定義の名称*"

機能

入出力データ情報定義機能を使用した場合の XML アクセス用データ定義の名称を指定します。入出力データ情報定義機能については，「3.　入出力データ情報定義機能」を参照してください。

規則

- nameOfBaseVar 属性の指定を省略した場合は，cobName 属性（省略時は elemName 属性）の名称に"-BASE"を追加した名称が XML アクセス用データ定義に生成されます。

  BaseElement 要素の cobName 属性または elemName 属性の名称は，25 文字以下で指定する必要があります。25 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。

- nameOfBaseVar 属性に指定する名称は，30 文字以下で指定する必要があります。30 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。

- nameOfBaseVar 属性には，COBOL データ項目名として使用できる名称を指定する必要があります。COBOL データ項目名に使用できない文字を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

- accessInfo 属性の指定を省略した場合，または accessInfo 属性に"no"を指定した場合，nameOfBaseVar 属性の指定は無効となります。

  accessInfo 属性の指定例については，「3.3.1(2)　指定例」を参照してください。

- nameOfBaseVar 属性には，次に示す属性とは異なる名称を指定する必要があります。同じ名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

  #### nameOfBaseVar 属性と異なる名称を指定する属性

  - BaseElement 要素の cobName 属性※，nameOfBaseVar 属性

- Group 要素の cobName 属性※，nameOfGroupVar 属性
- Item 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性
- Array 要素の nameOfCountVar 属性，nameOfTotalVar 属性
- AttrItem 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性

注※

cobName 属性の指定を省略した場合は，elemName 属性になります。

指定例

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (item01, item02)>
  <!ELEMENT item01  (#PCDATA)>
  <!ELEMENT item02  (#PCDATA)>
]>
<root/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="root" accessInfo="yes"
       nameOfBaseVar="BASE">
    <Group elemName="root">
      <Item elemName="item01" type="alphanumeric"
            size="10" />
      <Item elemName="item02" type="alphanumeric"
            size="10" />
    </Group>
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
 01 BASE.
  02 root.
   03 item01 PIC X(10).
   03 item02 PIC X(10).
* Access Information
  02 root-FLG PIC 1(32) USAGE BIT.
  02 root-GROUP.
   03 item01-FLG PIC 1(32) USAGE BIT.
   03 item01-LEN PIC 9(9) USAGE COMP.
   03 item02-FLG PIC 1(32) USAGE BIT.
   03 item02-LEN PIC 9(9) USAGE COMP.
```

(5) BaseElement 要素の使い方

(a) 繰り返し要素を BaseElement 要素に指定する

繰り返しのある XML 要素を，BaseElement 要素に指定できます。

例えば，次の DTD のドキュメント構造を持つ XML ドキュメントで，繰り返し要素「row」を BaseElement 要素に指定できます。この場合，XML 要素を読み込むアクセスルーチンを実行するたびに，要素「row」のデータが順番に読み込まれます。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (row)*>
  <!ELEMENT row (name, address, age)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (addr1, addr2)>
  <!ELEMENT addr1 (#PCDATA)>
  <!ELEMENT addr2 (#PCDATA)>
  <!ELEMENT age (#PCDATA)>
]>
<table/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="row">
  <Group cobName="row">
   <Item elemName="name" cobName="FULLNAME"
         type="alphanumeric" size="30"/>
   <Item elemName="addr1" type="alphanumeric"
         size="40"/>
   <Item elemName="addr2" type="alphanumeric"
         size="40"/>
   <Item elemName="age" type="alphanumeric"
         size="3"/>
  </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目の例)

```
01 row.
 02 FULLNAME PIC X(30).
 02 addr1 PIC X(40).
 02 addr2 PIC X(40).
 02 age PIC X(3).
```

(XML アクセスルーチンの呼び出し例)

```
  :
CALL 'CBLXML-RD-EXAMPLE-row'
    USING XML-POINTER row
```

```
    RETURNING CBLXML-RETURN-CODE.
  :
```

(XML ドキュメントの例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<table>
  <row>
    <name>Mr. X</name>
    <address>
      <addr1>Kanagawa ken</addr1>
      <addr2>Yokohama shi</addr2>
    </address>
    <age>22</age>
  </row>
</table>
```

(b) 入れ子になっている繰り返し要素を BaseElement 要素に指定する

繰り返し要素の親要素が，さらに繰り返し要素になっている場合でもアクセスできます。この場合，アクセスしたい XML 要素を BaseElement 要素に指定して，XML アクセスルーチンを繰り返し呼び出して取得します。

例えば，次の DTD の場合，要素「sentence」を BaseElement に設定して，XML アクセスルーチンを繰り返し呼び出すことで，要素の値を読み書きできます。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE book [
  <!ELEMENT book (chapter)*>
  <!ELEMENT chapter (paragraph)*>
  <!ELEMENT paragraph (sentence)*>
  <!ELEMENT sentence (#PCDATA)>
]>
<book/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="sentence">
    <Item elemName="sentence" cobName="sentence1"
          type="alphanumeric" size="200"/>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 sentence1 PIC X(200).
```

また，親の繰り返し要素「paragraph」を BaseElement 要素に指定し，子の繰り返し要素「sentence」を Array 要素で COBOL データ項目に対応づける方法もあります。この場合，1 回のアクセスルーチンの

呼び出しで，一つの要素「paragraph」の下位にあるすべての要素「sentence」の値を読み書きできます。Array 要素の max 属性には，読み書きしたい要素の最大数を指定してください。

（正しい DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="paragraph">
  <Group cobName="paragraph">
   <Array max="20" nameOfCountVar="NumSentences">
    <Item elemName="sentence" cobName="sentence1"
          type="alphanumeric" size="200"/>
   </Array>
  </Group>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 paragraph.
 02 sentence1 PIC X(200) OCCURS 20.
 02 NumSentences PIC 9(9) USAGE COMP.
```

(c)　選択要素を BaseElement 要素に指定する

選択要素（複数の要素のうち，どれが出現してもよい要素）の一つを BaseElement 要素に対応づける場合は，すべての選択要素を BaseElement 要素に指定する必要があります。

例えば次の例の場合，選択要素の一方「chapter1」を BaseElement 要素に対応づける場合は，「chapter1」「chapter2」をそれぞれ BaseElement 要素に指定する必要があります。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE book [
  <!ELEMENT book (chapter1 | chapter2)*>
  <!ELEMENT chapter1 (sentence1)>
  <!ELEMENT sentence1 (#PCDATA)>
  <!ELEMENT chapter2 (sentence2)>
  <!ELEMENT sentence2 (#PCDATA)>
]>
<book/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="chapter1">
  <Group cobName="chapter1">
    <Item elemName="sentence1" type="alphanumeric"
          size="200"/>
  </Group>
 </BaseElement>
 <BaseElement elemName="chapter2">
  <Group cobName="chapter2">
```

```
    <Item elemName="sentence2" type="alphanumeric"
          size="200"/>
  </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 chapter1.
 02 sentence1 PIC X(200).
01 chapter2.
 02 sentence2 PIC X(200).
```

(XML ドキュメントの例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<book>
  <chapter2>
    <sentence2>Mr. A</sentence2>
  </chapter2>
  <chapter1>
    <sentence1>Mr. B</sentence1>
  </chapter1>
</book>
```

(XML アクセスルーチンの呼び出しの例)

```
  :
CALL 'CBLXML-RD-EXAMPLE-chapter1'
    USING XML-HANDLE chapter1
    RETURNING CBLXML-RETURN-CODE.
IF CBLXML-CANT-SKIP-BE THEN
 CALL 'CBLXML-RD-EXAMPLE-chapter2'
    USING XML-HANDLE chapter2
    RETURNING CBLXML-RETURN-CODE
END-IF.
IF CBLXML-NO-BE-LEFT THEN
  CALL 'CBLXML-CL-EXAMPLE' USING XML-HANDLE
       RETURNING CBLXML-RETURN-CODE
END-IF.
  :
```

上記の例の場合，次のように処理が実行されます。

1. 要素「chapter1」を読み込む XML アクセスルーチン（CBLXML-RD-EXAMPLE-chapter1）が実行される

   XML ドキュメント上の最初の選択要素が要素「chapter2」なので，XML アクセスルーチンの戻り値には，エラーのステータス 5（CBLXML-CANT-SKIP-BE）が返されます。

2. 要素「chapter1」の読み込みに失敗したので，次に要素「chapter2」を読み込む XML アクセスルーチン（CBLXML-RD-EXAMPLE-chapter2）が実行される

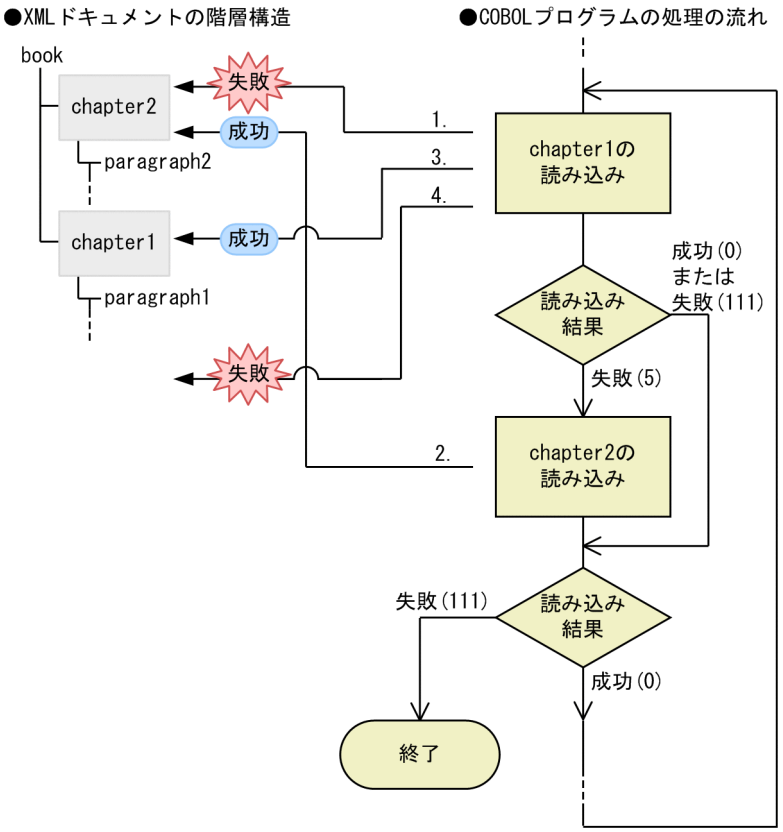
   要素「chapter2」の読み込みに成功します。XML アクセスルーチンの戻り値には，正常終了のステータス 0（CBLXML-OK）が返されます。

3. 要素「chapter1」を読み込む XML アクセスルーチン（CBLXML-RD-EXAMPLE-chapter1）が実行される

   XML ドキュメント上の次の選択要素が要素「chapter1」なので，要素の読み込みに成功します。XML アクセスルーチンの戻り値には，正常終了のステータス 0（CBLXML-OK）が返されます。

4. 要素「chapter1」を読み込む XML アクセスルーチン（CBLXML-RD-EXAMPLE-chapter1）が実行される

   XML ドキュメントには，もう読み込む要素が残っていないので，要素の読み込みに失敗します。XML アクセスルーチンの戻り値には，エラーのステータス 111（CBLXML-NO-BE-LEFT）が返されます。



(d)　異なる要素の下に共通にある要素

複数の異なる要素の下に共通にある要素を BaseElement 要素に対応づけて，要素を入力，および更新できます。ただし，要素の出力については，動作は保証しません。

次に，異なる要素の下に共通にある要素の例を示します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<!DOCTYPE root [
  <!ELEMENT root (group1 , group2 )>
  <!ELEMENT group1 (item1)>
  <!ELEMENT group2 (item1)>
  <!ELEMENT item1  (#PCDATA)>
```

```
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="item1">
  <Item elemName="item1" size="10"/>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 item1 PIC X(10).
```

(XML ドキュメントの例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
 <group1>
  <item1>dataa</item1>
 </group1>
 <group2>
  <item1>datab</item1>
 </group2>
</root>
```

(XML アクセスルーチンの呼び出しの例)

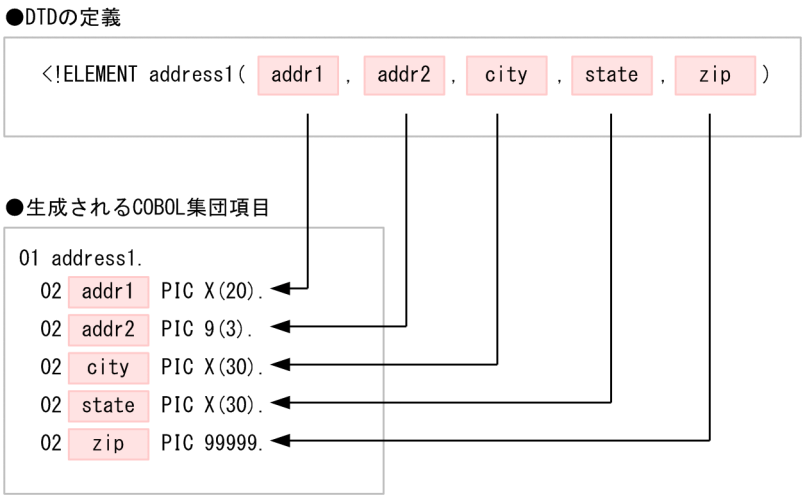
```
  :
CALL 'CBLXML-RD-EXAMPLE-item1'
    USING XML-HANDLE item1 RETURNING CBLXML-RETURN-CODE.  …1.
  :
CALL 'CBLXML-RD-EXAMPLE-item1'
    USING XML-HANDLE item1 RETURNING CBLXML-RETURN-CODE.  …2.
  :
```

(説明)

1. group1 の下の item1 を入力します。
2. group2 の下の item1 を入力します。

2.3.3 Group 要素（集団項目の定義）

Group 要素は，子要素を持つ XML 要素を COBOL の集団項目に対応づける要素です。Group 要素を使うと，DTD の階層構造に沿った COBOL データ項目を作成できます。また，DTD の階層構造とは無関係に COBOL の集団項目を作成することもできます。



Group 要素は，子要素として Group 要素，Array 要素，Item 要素，または AttrItem 要素のどれかを一つ以上持ちます。

形式

<Group
[elemName="*XML の要素名*"]
[cobName="*Group 要素に対応する名称*"]
[nameOfFlagVar="*アクセス情報フラグの名称*"]
[nameOfGroupVar="*入出力データ情報項目の名称*"]
[update="*yes*"] >
{*Group 要素* | *Array 要素* | *Item 要素* | *AttrItem 要素*} …
</Group>

(1) elemName 属性

形式

elemName="*XML の要素名*"

機能

Group 要素に対応づける XML の要素名を指定します。

elemName 属性は，DTD の階層構造と同じ構造で COBOL 集団項目を生成する場合に使用します。

規則

- Group 要素に elemName 属性を指定した場合，Group 要素の下位に存在する Group 要素すべてに elemName 属性を指定する必要があります。
- XML の要素名称は，XML で規定された文字で構成する必要があります。
- elemName 属性で指定した XML の要素名称が COBOL の集団項目名として使用できない場合，COBOL 原始プログラム生成時にエラーとなります。この場合，cobName 属性を使って集団項目名を指定する必要があります。

指定例

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<!DOCTYPE group1 [
  <!ELEMENT group1 (group21, group22)>
  <!ELEMENT group21 (item01, item02)>
  <!ELEMENT group22 (item01, item02)>
  <!ELEMENT item01 (#PCDATA)>
  <!ELEMENT item02 (#PCDATA)>
]>
<group1/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="group1">
  <Group elemName="group1">
   <Group elemName="group21">
    <Item elemName="item01" type="alphanumeric"
          size="50"/>
    <Item elemName="item02" type="alphanumeric"
          size="50"/>
   </Group>
   <Group elemName="group22">
    <Item elemName="item01" type="alphanumeric"
          size="50"/>
    <Item elemName="item02" type="alphanumeric"
          size="50"/>
   </Group>
  </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 group1.
 02 group21.
  03 item01 PIC X(50).
  03 item02 PIC X(50).
 02 group22.
  03 item01 PIC X(50).
  03 item02 PIC X(50).
```

(2) cobName 属性

形式

cobName="*Group 要素に対応する名称*"

機能

elemName 属性の値の代わりに COBOL の集団項目名を指定します。elemName 属性に指定する XML 要素の名称が，COBOL の集団項目名に使用できない場合などに指定します。

また，elemName 属性を省略して cobName 属性だけを指定した場合，COBOL プログラムからアクセスする特定の XML 要素だけを COBOL 集団項目に対応づけられます。

規則

- cobName 属性には，COBOL の集団項目名として使用できる名称を指定する必要があります。COBOL の集団項目名として使用できない名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- elemName 属性を省略して cobName 属性だけを指定した場合，DTD の中で基底要素（BaseElement タグに指定する要素）として選択した要素に含まれる要素の内容は，すべて異なっている必要があります。

  例えば，次の DTD では要素 X が要素内容として 2 回出現するため，要素 A を基底要素にできません。

  ```
  <!ELEMENT A(P,Q)>
  <!ELEMENT P(X)>
  <!ELEMENT Q(X)>
  ```

指定例（elemName 属性，cobName 属性の両方を指定する場合）

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (group21 , group22)>
  <!ELEMENT group21 (item01, item02)>
  <!ELEMENT group22 (item01, item02)>
  <!ELEMENT item01 (#PCDATA)>
  <!ELEMENT item02 (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="table">
  <Group elemName="table" cobName="GROUP1">
   <Group elemName="group21">
    <Item elemName="item01" type="alphanumeric"
          size="50"/>
    <Item elemName="item02" type="alphanumeric"
          size="50"/>
   </Group>
   <Group elemName="group22">
    <Item elemName="item01" type="alphanumeric"
          size="50"/>
    <Item elemName="item02" type="alphanumeric"
          size="50"/>
   </Group>
```

```
  </Group>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 GROUP1.
 02 group21.
  03 item01 PIC X(50).
  03 item02 PIC X(50).
 02 group22.
  03 item01 PIC X(50).
  03 item02 PIC X(50).
```

指定例（cobName 属性だけを指定する場合）

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (group21 , group22)>
  <!ELEMENT group21 (item01, item02)>
  <!ELEMENT group22 (item03, item04)>
  <!ELEMENT item01 (#PCDATA)>
  <!ELEMENT item02 (#PCDATA)>
  <!ELEMENT item03 (#PCDATA)>
  <!ELEMENT item04 (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="table">
  <Group cobName="group22">
    <Item elemName="item03" type="alphanumeric"
          size="50"/>
    <Item elemName="item04" type="alphanumeric"
          size="50"/>
  </Group>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 group22.
 02 item03 PIC X(50).
 02 item04 PIC X(50).
```

(3)　nameOfFlagVar 属性

形式

nameOfFlagVar="*アクセス情報フラグの名称*"

機能

Interface 要素または BaseElement 要素の accessInfo 属性に"yes"を指定した場合に，Group 要素に対応するアクセス情報フラグの名称を指定します。

アクセス情報フラグについては，「3.2.1　アクセス情報フラグ」を参照してください。

規則

- nameOfFlagVar 属性の指定を省略した場合は，Group 要素の cobName 属性（省略時は elemName 属性）に指定した名称に"-FLG"を追加した名称が XML アクセス用データ定義に生成されます。

  Group 要素の cobName 属性または elemName 属性の名称は，26 文字以下で指定する必要があります。26 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。

- nameOfFlagVar 属性には，COBOL データ項目名として使用できる名称を指定する必要があります。COBOL データ項目名に使用できない文字を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

- accessInfo 属性の指定を省略した場合，または accessInfo 属性に"no"を指定した場合，nameOfFlagVar 属性の指定は無効となります。

  accessInfo 属性の指定例については，「3.3.2(2)　指定例」を参照してください。

- nameOfFlagVar 属性には，次に示す属性とは異なる名称を指定する必要があります。同じ名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

  nameOfFlagVar 属性と異なる名称を指定する属性

  - BaseElement 要素の cobName 属性※，nameOfBaseVar 属性
  - Group 要素の cobName 属性※，nameOfFlagVar 属性，nameOfGroupVar 属性
  - Item 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性
  - Array 要素の nameOfCountVar 属性，nameOfTotalVar 属性
  - AttrItem 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性

注※

cobName 属性の指定を省略した場合は，elemName 属性になります。

nameOfFlagVar 属性の指定例については，「3.2.1(3)　nameOfFlagVar 属性の指定例」を参照してください。

(4)　nameOfGroupVar 属性

形式

nameOfGroupVar="*入出力データ情報項目の名称*"

機能

入出力データ情報定義機能によって生成された集団項目の名称を指定します。このとき，Interface 要素または BaseElement 要素の accessInfo 属性には"yes"を指定しておきます。

入出力データ情報定義機能については，「3.　入出力データ情報定義機能」を参照してください。

規則

- nameOfGroupVar 属性の指定を省略した場合は，Group 要素の cobName 属性（省略時は elemName 属性）に指定した名称に"-GROUP"を追加した名称が XML アクセス用データ定義に生成されます。

  Group 要素の cobName 属性または elemName 属性の名称は，24 文字以下で指定する必要があります。24 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。

- nameOfGroupVar 属性の名称は 30 文字以下で指定する必要があります。30 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。

- nameOfGroupVar 属性には，COBOL データ項目の名称に使用できる名称を指定する必要があります。COBOL データ項目名に使用できない文字を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

- accessInfo 属性の指定を省略した場合，または accessInfo 属性に"no"を指定した場合，nameOfGroupVar 属性の指定は無効となります。

  accessInfo 属性の指定例については，「3.3.2(2)　指定例」を参照してください。

- nameOfGroupVar 属性には，次に示す属性とは異なる名称を指定する必要があります。同じ名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

  #### nameOfGroupVar 属性と異なる名称を指定する属性

  - BaseElement 要素の cobName 属性※，nameOfBaseVar 属性
  - Group 要素の cobName 属性※，nameOfFlagVar 属性，nameOfGroupVar 属性
  - Item 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性
  - Array 要素の nameOfCountVar 属性，nameOfTotalVar 属性
  - AttrItem 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性

注※

cobName 属性の指定を省略した場合は，elemName 属性になります。

指定例

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (item01, item02)>
  <!ELEMENT item01  (#PCDATA)>
  <!ELEMENT item02  (#PCDATA)>
```

```
]>
<root/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="root" accessInfo="yes">
    <Group elemName="root"
     nameOfGroupVar="GROUP1">
      <Item elemName="item01" type="alphanumeric"
            size="10" />
      <Item elemName="item02" type="alphanumeric"
            size="10" />
    </Group>
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
 01 root-BASE.
  02 root.
   03 item01 PIC X(10).
   03 item02 PIC X(10).
* Access Information
  02 root-FLG PIC 1(32) USAGE BIT.
  02 GROUP1.
   03 item01-FLG PIC 1(32) USAGE BIT.
   03 item01-LEN PIC 9(9) USAGE COMP.
   03 item02-FLG PIC 1(32) USAGE BIT.
   03 item02-LEN PIC 9(9) USAGE COMP.
```

(5)　update 属性

形式

update="*yes*"

機能

Group 要素を更新の対象にするかどうかを指定します。更新機能については，「7.2.11　XML ドキュメントの更新」を参照してください。

規則

- Group 要素を elemName 属性で XML 要素に対応づけた場合，update 属性を指定した Group 要素に対応する XML 要素に含まれるすべての要素が更新の対象となります。Group 要素に cobName 属性だけを指定した場合，update 属性を指定した Group 要素に含まれる Item 要素，Group 要素が更新の対象となります。
- update 属性に"yes"以外の値を指定した場合は，COBOL 原始プログラムの生成時にエラーとなります。

- update 属性を指定した Group 要素の内側にある Group 要素や Item 要素に対して update 属性を指定してはなりません。指定した場合は，COBOL 原始プログラムの生成時に警告メッセージが出力され，update 属性が無視されます。

(6) Group 要素の使い方

(a) XML ドキュメントの構造と異なる COBOL の集団項目への対応づけ

Group 要素では，BaseElement 要素でアクセス対象に指定した XML 要素，およびその子要素を COBOL の集団項目に対応づけられます。このとき，COBOL の集団項目が XML ドキュメントと異なる階層構造になってもかまいません。

COBOL の集団項目と XML ドキュメントとを異なる階層構造にする場合は，次の点に注意してください。

- COBOL 集団項目に対応する Group 要素には，elemName 属性を指定できません。集団項目名は，cobName 属性で指定してください。
- DTD の中で基底要素（BaseElement タグに指定する要素）として選択する要素に含まれる要素の内容は，すべて異なっている必要があります。詳細は，「2.3.3(2) cobName 属性」の規則を参照してください。

XML 要素「name」「zipcode」「telephone」および「addr1」を COBOL の集団項目に対応づける例を次に示します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (person)*>
  <!ELEMENT person (name, address, zipcode, telephone)>
  <!ELEMENT address (addr1, addr2)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT zipcode (#PCDATA)>
  <!ELEMENT telephone (#PCDATA)>
  <!ELEMENT addr1 (#PCDATA)>
  <!ELEMENT addr2 (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="person">
  <Group cobName="person">
   <Item elemName="name" cobName="FULLNAME"
         type="alphanumeric" size="30"/>
   <Item elemName="zipcode" type="alphanumeric"
         size="8"/>
   <Item elemName="telephone" type="alphanumeric"
         size="13"/>
   <Item elemName="addr1" type="alphanumeric"
```

```
          size="120"/>
  </Group>
 </BaseElement>
</Interface>
```

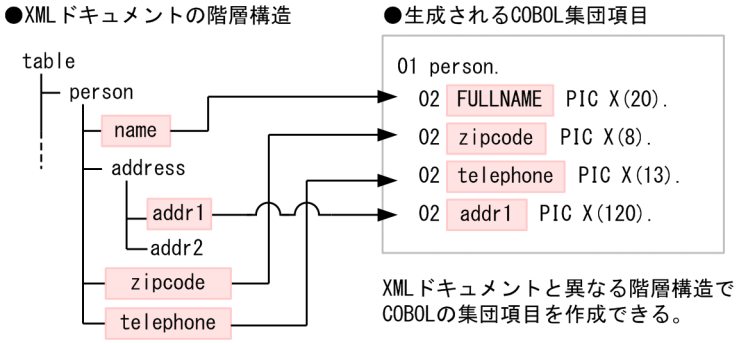
(生成される COBOL データ項目)

```
01 person.
 02 FULLNAME PIC X(30).
 02 zipcode PIC X(8).
 02 telephone PIC X(13).
 02 addr1 PIC X(120).
```

(出力される XML ドキュメントの例)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<table>
  <person>
    <name>Mr. X</name>
    <address>
      <addr1>Kanagawa ken</addr1>
      <addr2/>
    </address>
    <zipcode>XXX-XXX</zipcode>
    <telephone>XXX-XXXX-XXXX</telephone>
  </person>
</table>
```



(b) Group 要素内の Group 要素の省略

Group 要素の中に Group 要素がある場合で，子の Group 要素を COBOL プログラムからアクセスしないときは，対応づけを省略できます。このとき，省略した Group 要素のデータは，COBOL プログラムから読み込めません。また，COBOL プログラムから XML ドキュメントへの書き込みをした場合，省略した Group 要素に含まれる XML 要素には，空要素が出力されます。

次の例では，Group 要素「person」の子 Group 要素「fulladdress」への対応づけを省略しています。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (person)*>
  <!ELEMENT person (name, fulladdress, telephone)>
```

```
  <!ELEMENT telephone (#PCDATA)>
  <!ELEMENT fulladdress (address, zipcode)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT zipcode (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
]>
<table/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="person">
  <Group cobName="person">
   <Item elemName="name" cobName="FULLNAME"
         type="alphanumeric" size="30"/>
   <Item elemName="telephone" type="alphanumeric"
         size="13"/>
  </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 person.
 02 FULLNAME PIC X(30).
 02 telephone PIC X(13).
```

(出力される XML ドキュメントの例)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<table>
  <person>
    <name>Mr. X</name>
    <fulladdress>
      <address/>
      <zipcode/>
    </fulladdress>
    <telephone>XXX-XXXX-XXXX</telephone>
  </person>
</table>
```

(c) 選択要素を Group 要素に指定する場合

選択要素（複数の要素のうち，どれが出現してもよい要素）の一つを COBOL の集団項目に対応づけるには，すべての選択要素を Group 要素に指定する必要があります。

例えば，次の例の場合，選択要素「order」「invoice」のどちらか一方にアクセスしたい場合は，「order」「invoice」の両方を Group 要素に指定する必要があります。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE transaction [
  <!ELEMENT transaction (order | invoice)>
```

```
  <!ELEMENT order (goods, name1, fulladdress1,
                   telephone1)>
  <!ELEMENT goods (#PCDATA)>
  <!ELEMENT name1 (#PCDATA)>
  <!ELEMENT fulladdress1 (address1, zipcode1)>
  <!ELEMENT address1 (#PCDATA)>
  <!ELEMENT zipcode1 (#PCDATA)>
  <!ELEMENT telephone1 (#PCDATA)>
  <!ELEMENT invoice (amount, name2, fulladdress2,
                     telephone2)>
  <!ELEMENT amount (#PCDATA)>
  <!ELEMENT name2 (#PCDATA)>
  <!ELEMENT fulladdress2 (address2, zipcode2)>
  <!ELEMENT address2 (#PCDATA)>
  <!ELEMENT zipcode2 (#PCDATA)>
  <!ELEMENT telephone2 (#PCDATA)>
]>
<transaction/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
 <Interface interfaceName="EXAMPLE">
 <BaseElement elemName="transaction">
  <Group elemName="transaction"
         cobName="TRANSACTION0">
   <Group elemName="order" cobName="ORDER1">
    <Item elemName="goods" type="alphanumeric"
          size="50"/>
    <Item elemName="name1" cobName="FULLNAME1"
          type="alphanumeric" size="30"/>
    <Group elemName="fulladdress1"
          cobName="FULLADDRESS1">
     <Item elemName="address1" cobName="ADDRESS1"
          type="alphanumeric" size="120"/>
     <Item elemName="zipcode1" type="alphanumeric"
          size="8"/>
    </Group>
    <Item elemName="telephone1" type="alphanumeric"
          size="13"/>
   </Group>
   <Group elemName="invoice" cobName="INVOICE">
    <Item elemName="amount" type="alphanumeric"
          size="50"/>
    <Item elemName="name2" cobName="FULLNAME2"
          type="alphanumeric" size="30"/>
    <Group elemName="fulladdress2"
           cobName="FULLADDRESS2">
     <Item elemName="address2" cobName="ADDRESS2"
           type="alphanumeric" size="120"/>
     <Item elemName="zipcode2" type="alphanumeric"
           size="8"/>
    </Group>
    <Item elemName="telephone2" type="alphanumeric"
          size="13"/>
   </Group>
  </Group>
```

```
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 TRANSACTION0.
 02 ORDER1.
  03 goods PIC X(50).
  03 FULLNAME1 PIC X(30).
  03 FULLADDRESS1.
   04 ADDRESS1 PIC X(120).
   04 zipcode1 PIC X(8).
  03 telephone1 PIC X(13).
 02 INVOICE.
  03 amount PIC X(50).
  03 FULLNAME2 PIC X(30).
  03 FULLADDRESS2.
   04 ADDRESS2 PIC X(120).
   04 zipcode2 PIC X(8).
  03 telephone2 PIC X(13).
```

（出力される XML ドキュメントの例）

```
<?xml version="1.0" encoding="Shift_JIS"?>

<transaction>
  <order>
    <goods>DVD</goods>
    <name1>Mr. X</name1>
    <fulladdress1>
      <address1>Kanagawa ken</address1>
      <zipcode1>XXX-XXX</zipcode1>
    </fulladdress1>
    <telephone1>XXX-XXXX-XXXX</telephone1>
  </order>
</transaction>
```

なお，XML ドキュメントを書き込む場合，書き込む要素以外の選択要素は，初期化しておく必要があります。

(d)　再帰的な XML 構造

再帰的構造を持つ要素は，verbatim 属性に"yes"を指定した Item 要素に対応づけることで，XML ドキュメントの入出力，および更新ができます。

DDF で対応づけていない要素が再帰的構造を持ち，それを省略できない場合，XML ドキュメントの入力および更新はできますが，出力はできません。XML ドキュメントを出力した場合は，CBLXML-WR-*Interface-BaseElement* アクセスルーチンのステータスに 109 が返されます。

DTD に再帰的構造があっても，再帰的構造の出力を省略できる場合は，XML ドキュメントの出力ができます。次にその例を示します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<!DOCTYPE root [
  <!ELEMENT root (group1, item2) >
  <!ELEMENT group1 (item1, group1?) >    …1.
  <!ELEMENT item1 (#PCDATA)>
  <!ELEMENT item2 (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
<BaseElement elemName="item2" cobName="BE">
  <Item elemName="item2" type="alphanumeric" size="10"/>
</BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 item2 PIC X(10).
```

(出力される XML ドキュメントの例)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<root>
  <group1>
    <item1/>
                  …2.
  </group1>
  <item2>data1</item2>
</root>
```

(説明)

1. group1 要素の中で group1 要素を参照する再帰的構造です。
2. 再帰の group1 要素は省略できるので，出力しません。

(e) XML 要素の一意な参照

DTD と同じ階層を持つ COBOL データ項目を作成する場合，COBOL で明示的に一意な参照ができる対応づけをする必要があります。XML 要素の名称だけでは一意な参照ができない場合は，cobName 属性を使ってデータ項目ごとに固有の名称を指定してください。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (group21 , group11)>
  <!ELEMENT group11 (group21)>
  <!ELEMENT group21 (item01)>
```

```
  <!ELEMENT item01 (#PCDATA)>
]>
<table/>
```

(データ項目名が一意になるように対応づけされていない DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="table">
  <Group elemName="table" cobName="GROUP1">
   <Group elemName="group21">
    <Item elemName="item01" type="alphanumeric"
     size="50"/>
   </Group>
   <Group elemName="group11">
    <Group elemName="group21">
     <Item elemName="item01" type="alphanumeric"
      size="50"/>
    </Group>
   </Group>
  </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 GROUP1.
 02 group21.
  03 item01 PIC X(50).
 02 group11.
  03 group21.
   04 item01 PIC X(50).
```

上記のデータ項目の場合，"item01 OF group21 OF GROUP1"の名称が一意に特定できません。

(データ項目名が一意になるように対応づけされている DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="table">
  <Group elemName="table" cobName="GROUP1">
   <Group elemName="group21" cobName="group21A">
    <Item elemName="item01" type="alphanumeric"
     size="50"/>
   </Group>
   <Group elemName="group11">
    <Group elemName="group21">
     <Item elemName="item01" type="alphanumeric"
      size="50"/>
    </Group>
   </Group>
  </Group>
 </BaseElement>
</Interface>
```

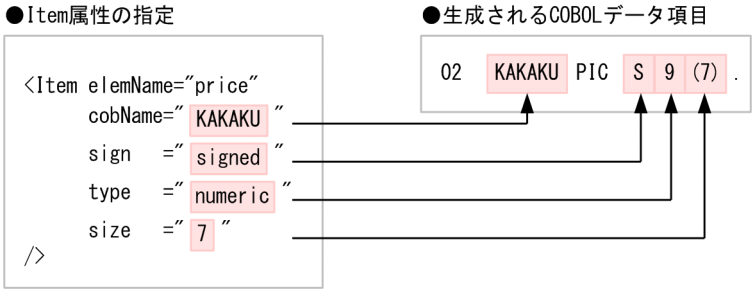
（生成される COBOL データ項目）

```
01 GROUP1.
 02 group21A.
  03 item01 PIC X(50).
 02 group11.
  03 group21.
   04 item01 PIC X(50).
```

上記のデータ項目の場合，item01 の属するデータ項目が「group21A」「group21」の 2 種類に分かれているので，名称が一意に特定できます。

2.3.4 Item 要素（要素の対応づけの定義）

Item 要素は，要素データを持つ個々の XML 要素を COBOL データ項目に対応づける要素です。



形式

<Item elemName="*XML の要素名*"
[cobName="*Item 要素に対応する名称*"]
[type="*type 属性値*"]
[size="*けた数*"]
[fractionalDigits="*けた数*"]
[emptyValue="*type 属性値に対応した COBOL の定数*"]
[sign="*符号種別*"]
[trim="*yes*|*no*"]
[nameOfFlagVar="*アクセス情報フラグの名称*"]
[nameOfLengthVar="*データ長の名称*"]
[verbatim="*yes*|*no*"]
[update="*yes*"]
[emptyContentValue="*type 属性値に対応した COBOL の定数*"]
[invalidCharValue="*type 属性値に対応した COBOL の定数*"]
[overflowValue="*type 属性値に対応した COBOL の定数*"]
/>

type 属性の値と指定できるその他の属性の関連

Item 要素に指定できる属性のうち，次の表にある属性は，type 属性の値によって指定の可否が異なります。次の表に示さない属性は，type 属性の値にかかわらず指定できます。

type 属性の値	属性の指定の可否				
	size	sign	fractionalDigits	trim	verbatim
alphanumeric	○	×	×	○	○
national	○	×	×	○	×
numeric	○	○	○	○	×
packed	○	○	○	○	×
binary	○	○	○	○	×
float	×	×	×	×	×
double	×	×	×	×	×

（凡例）

○：指定できます。

×：指定できません。

(1) elemName 属性

形式

elemName="*XML の要素名*"

機能

COBOL データ項目に対応づける XML 要素の名称を指定します。

規則

- XML の要素名称は，XML で規定された文字で構成する必要があります。
- elemName 属性に指定する XML 要素は，要素データ（#PCDATA キーワードまたは EMPTY キーワード）を持っている必要があります。
- elemName 属性で指定した XML の要素名称が COBOL データ項目名として使用できない場合，COBOL 原始プログラム生成時にエラーとなります。この場合，cobName 属性を使ってデータ項目名を指定する必要があります。

指定例

XML 要素「goods」「name」「home」および「cellular」をそれぞれ COBOL データ項目に対応づける例を，次に示します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (order)*>
```

```
  <!ELEMENT order (goods, name, telephone)>
  <!ELEMENT goods (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT telephone (home | cellular)>
  <!ELEMENT home (#PCDATA)>
  <!ELEMENT cellular (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="order">
  <Group cobName="ORDERX">
   <Item elemName="goods" type="alphanumeric"
         size="30"/>
   <Item elemName="name" cobName="FULLNAME"
         type="alphanumeric" size="30"/>
    <Group cobName="telephone">
     <Item elemName="home" type="alphanumeric"
           size="13"/>
     <Item elemName="cellular" type="alphanumeric"
           size="13"/>
    </Group>
  </Group>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 ORDERX.
 02 goods PIC X(30).
 02 FULLNAME PIC X(30).
 02 telephone.
  03 home PIC X(13).
  03 cellular PIC X(13).
```

(2)　cobName 属性

形式

cobName="*Item 要素に対応する名称*"

機能

elemName 属性の値の代わりに COBOL データ項目の名称を指定します。elemName 属性に指定する XML 要素の名称が，COBOL データ項目名に使用できない場合などに指定します。

規則

cobName 属性には，COBOL データ項目名として使用できる名称を指定する必要があります。COBOL データ項目名として使用できない名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

指定例

XML 要素「name」の COBOL データ項目名に「FULLNAME」，XML 要素「address」の COBOL データ項目名に「ADDRESS1」をそれぞれ指定する例を次に示します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (order)*>
  <!ELEMENT order (goods, name, fulladdress)>
  <!ELEMENT goods (#PCDATA)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT fulladdress (zipcode, address)>
  <!ELEMENT zipcode (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="order">
  <Group cobName="ORDER1">
   <Item elemName="goods" type="alphanumeric"
         size="30"/>
   <Item elemName="name" cobName="FULLNAME"
         type="alphanumeric" size="30"/>
    <Group cobName="FULLADDRESS">
     <Item elemName="zipcode" type="alphanumeric"
           size="8"/>
     <Item elemName="address" cobName="ADDRESS1"
           type="alphanumeric" size="120"/>
    </Group>
  </Group>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 ORDER1.
 02 goods PIC X(30).
 02 FULLNAME PIC X(30).
 02 FULLADDRESS.
  03 zipcode PIC X(8).
  03 ADDRESS1 PIC X(120).
```

(3)　type 属性

形式

type="*type 属性値*"

機能

Item 要素に対応する COBOL データ項目のデータ型を指定します。

type 属性に指定する値と，生成される COBOL データ項目の対応を表 2-2 に示します。

表 2-2　type 属性に指定する値と生成される COBOL データ項目の対応

type 属性の値	生成される COBOL データ項目
alphanumeric	英数字項目
national	日本語項目
numeric	数字項目（外部 10 進形式）
packed	数字項目（内部 10 進形式）
binary	数字項目（2 進形式）
float	数字項目（内部浮動小数点形式　4 バイト）
double	数字項目（内部浮動小数点形式　8 バイト）

規則

type 属性の指定を省略した場合は，"alphanumeric"が仮定されます。

指定例

XML 要素「description」を英数字項目に対応づける例を次に示します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (description)>
  <!ELEMENT description (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="table">
  <Item elemName="description"
        type="alphanumeric"/>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 description PIC X(100).
```

(4)　size 属性

形式

size="*けた数*"

機能

COBOL データ項目に対応する PICTURE 句のけた数を指定します。

size 属性に指定できるけた数の最小値，最大値，および size 属性の指定を省略した場合に仮定されるけた数を，表 2-3 に示します。

表 2-3　size 属性に指定できる値の範囲と省略時のけた数

type 属性の値	size 属性に指定できる値の範囲		size 属性省略時のけた数
	最小値	最大値	
alphanumeric	1	16,777,215	100
national	1	16,383	100
numeric	1	18	9
packed	1	18	9
binary	1	18	9

規則

- size 属性は，同じ Item 要素の type 属性に"alphanumeric"，"national"，"numeric"，"packed"，または"binary"を指定した場合だけ，指定できます。
- 表 2-3 に示した最小値，最大値の範囲を超える値を size 属性に指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- COBOL データ項目と XML ドキュメントとの間の転記の規則については，「7.2.7　値の入力，出力の動作」を参照してください。

指定例

XML 要素「description」に対応する COBOL データ項目の文字数を 200 に設定する例を次に示します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (description)>
  <!ELEMENT description (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="table">
  <Item elemName="description" type="alphanumeric"
        size="200"/>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 description PIC X(200).
```

(5) fractionalDigits 属性

形式

fractionalDigits="*けた数*"

機能

COBOL の数字項目に対して，小数点以下のけた数，または位取りのけた数を指定します。

規則

- けた数には，次の値を指定します。

小数点以下のけた数を指定する場合

size 属性に指定した（または仮定された）けた数以下の値を指定します。

整数部の位取りのけた数を指定する場合

値の前にハイフン（-）を付けて指定します。

小数部の位取りのけた数を指定する場合

size 属性に指定した（または仮定された）けた数より大きい値を指定します。

- fractionalDigits 属性は，同じ Item 要素の type 属性に"numeric"，"packed"，または"binary"を指定した場合だけ，指定できます。
- fractionalDigits 属性を省略した場合，小数点以下のけた数，および位取りのけた数には 0 が仮定されます。
- size 属性に指定した，または仮定されたけた数を超えたけた数を，小数点以下のけた数に指定できません。
- fractionalDigits 属性で整数部の位取りを指定する場合，size 属性の値と fractionalDigits 属性の値の絶対値（総けた数）の和が 18 けたを超えてはなりません。18 けたを超える値を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- fractionalDigits 属性で小数点以下のけた数，または小数部の位取りを指定する場合，18 けた以内の値を指定する必要があります。18 けたを超える値を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

指定例

- XML 要素「price1」に対応する COBOL データ項目の小数点以下のけた数を 3 けたに設定します。
- XML 要素「price2」に対応する COBOL データ項目の小数点以下のけた数を 6 けたに設定します。
- XML 要素「price3」に対応する COBOL データ項目の小数部の位取りのけた数を 3 けたに設定します。
- XML 要素「price4」に対応する COBOL データ項目の整数部の位取りのけた数を 3 けたに設定します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (row)>
  <!ELEMENT row (price1, price2, price3, price4)>
  <!ELEMENT price1 (#PCDATA)>
  <!ELEMENT price2 (#PCDATA)>
  <!ELEMENT price3 (#PCDATA)>
  <!ELEMENT price4 (#PCDATA)>
]>
<table/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="row">
  <Group cobName="row">
   <Item elemName="price1" type="numeric" size="6"
         fractionalDigits="3"/>
   <Item elemName="price2" type="numeric" size="6"
         fractionalDigits="6"/>
   <Item elemName="price3" type="numeric" size="6"
         fractionalDigits="9"/>
   <Item elemName="price4" type="numeric" size="6"
         fractionalDigits="-3"/>
  </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 row.
 02 price1 PIC 9(3)V9(3).
 02 price2 PIC V9(6).
 02 price3 PIC VP(3)9(6).
 02 price4 PIC 9(6)P(3).
```

(6) emptyValue 属性

形式

emptyValue="*type 属性値に対応した COBOL の定数*"

機能

次の XML 要素に対応する COBOL データ項目の値を指定します。

- 省略できる要素で，省略されている要素
- 選択要素で，選択されていない要素
- countVar 属性に"no"を指定した場合の繰り返し項目の要素

XML ドキュメントの読み込み時，読み込んだ要素が上記のどれかの場合，emptyValue 属性に指定した値が COBOL データ項目に格納されます。

emptyValue 属性に指定した値が COBOL データ項目に格納された状態で XML ドキュメントに書き込みをすると，そのデータ項目に対応する要素を出力しないようにできます。ただし，省略できる要素や選択要素が BaseElement 要素で指定された要素の場合，COBOL データ項目の値に関係なく必ず出力されます。

規則

- emptyValue 属性に指定できる要素の値，および emptyValue 属性の指定を省略した場合に仮定される要素の値を，表 2-4 に示します。

表 2-4　emptyValue 属性に指定できる値と省略時の値

type 属性の値	英字	英数字	日本語文字	SPACE	数値	ZERO	HIGH-VALUE	LOW-VALUE	省略時の値
alphanumeric	○	○	○	○	○	○	○	○	SPACE
national	×	×	○	○	×	○	○	○	SPACE
numeric	×	×	×	×	○	○	○	○	ZERO
packed	×	×	×	×	○	○	×	×	ZERO
binary	×	×	×	×	○	○	×	×	ZERO
float	×	×	×	×	○	○	×	×	ZERO
double	×	×	×	×	○	○	×	×	ZERO

（凡例）
○：指定できます。
×：指定できます。

- type 属性と emptyValue 属性に指定する値は，表 2-4 に記載された組み合わせだけが有効となります。これ以外の組み合わせを指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- emptyValue 属性に指定する文字列の長さは，同じ Item 要素の size 属性に指定する COBOL データ項目の長さを超えないようにしてください。超えた場合，size 属性に指定した長さが有効となります。
- accessInfo 属性"yes"を指定した Interface 要素，BaseElement 要素の内側にある Item 要素に emptyValue 属性は指定できません。emptyValue 属性を指定した場合は COBOL ソースの生成時に警告メッセージが表示され，emptyValue 属性は無視されます。
- emptyValue 属性に指定した文字列が 160 文字を超えた場合，先頭から 160 文字が採用されます。このとき，警告メッセージが表示されます。
- 「ZERO」「SPACE」「HIGH-VALUE」「LOW-VALUE」は，emptyValue 属性の予約語です。これらの文字列は，COBOL の英数字定数として emptyValue 属性に指定できません。また，これらの文字列は，COBOL 言語仕様の表意定数と同じように，英大文字と英小文字が等価として扱われます。
- emptyValue 属性に指定した数字が 18 けたを超える場合，COBOL 原始プログラムの生成時にエラーとなります。また，浮動小数点の仮数部が 16 けたを超える，指数部が 2 けたを超えるなど，

浮動小数点の形式でない場合も，COBOL 原始プログラムの生成時にエラーとなります。なお，浮動小数点の形式で使用する「E」の文字は，大文字，小文字の両方を使用できます。

注意事項

- ATTLIST 属性のデフォルト指定で#FIXED 値を指定した属性の入力に対しては，emptyValue 属性を指定しても無効となります。必ず，#FIXED で指定したデフォルト値を入力値として設定してください。

指定例（XML ドキュメントを出力する場合）

XML 要素「item11」「item21」「item22」を省略した場合，または選択しなかった場合に設定する COBOL の値を，それぞれ"ITEM11"，"ITEM21"，"ITEM22"に設定します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (item1, item2)>
  <!ELEMENT item1 (item11?)>
  <!ELEMENT item2 (item21 | item22)>
  <!ELEMENT item11 (#PCDATA)>
  <!ELEMENT item21 (#PCDATA)>
  <!ELEMENT item22 (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="table">
  <Group elemName="table" cobName="TABLE1">
   <Group elemName="item1">
    <Item elemName="item11" type="alphanumeric"
          size="30" emptyValue="ITEM11"/>
   </Group>
   <Group elemName="item2">
    <Item elemName="item21" type="alphanumeric"
          size="20" emptyValue="ITEM21"/>
    <Item elemName="item22" type="alphanumeric"
          size="20" emptyValue="ITEM22"/>
   </Group>
  </Group>
 </BaseElement>
</Interface>
```

（COBOL プログラムのコーディング例）

```
      :
MOVE 'ITEM11' TO ITEM11.  …1.
MOVE 'ITEM21' TO ITEM21.  …2.
MOVE 'ABCDEF' TO ITEM22.  …3.
      :
```

（説明）

1. 省略できる要素を省略したい場合には，emptyValue 属性に指定した値を設定する。

2. 選択要素のうち，選択しない要素には，emptyValue 属性に指定した値を設定する。

3. 出力する要素には，値（文字列）を設定する。

（出力される XML ドキュメント）

```
<?xml version="1.0" encoding="Shift_JIS"?>

<table>
  <item1/>  …1.
  <item2>
    <item22>ABCDEF</item22>  …2.
  </item2>
</table>
```

（説明）

1. item11 を省略したため，item1 が空要素で出力される。

2. item22 に，設定した文字列が出力される。

指定例（XML ドキュメントから入力する場合）

XML 要素「item1」～「item11」から COBOL データ項目に値を入力します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (item1?, item2?, item3?, item4?,
                   item5?, item6?, item7?, item8?,
                   item9?, item10?, item11?)>
  <!ELEMENT item1 (#PCDATA)>
  <!ELEMENT item2 (#PCDATA)>
  <!ELEMENT item3 (#PCDATA)>
  <!ELEMENT item4 (#PCDATA)>
  <!ELEMENT item5 (#PCDATA)>
  <!ELEMENT item6 (#PCDATA)>
  <!ELEMENT item7 (#PCDATA)>
  <!ELEMENT item8 (#PCDATA)>
  <!ELEMENT item9 (#PCDATA)>
  <!ELEMENT item10 (#PCDATA)>
  <!ELEMENT item11 (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="table">
  <Group elemName="table" cobName="TABLE1">
    <Item elemName="item1"  type="alphanumeric"
          size="30" emptyValue="SPACE"/>
    <Item elemName="item2"  type="alphanumeric"
          size="30" emptyValue="SPACE"/>
```

```
    <Item elemName="item3"  type="alphanumeric"
          size="30" emptyValue="SPACE"/>
    <Item elemName="item4"  type="alphanumeric"
          size="30" emptyValue="HIGH-VALUE"/>
    <Item elemName="item5"  type="alphanumeric"
          size="30" emptyValue="LOW-VALUE"/>
    <Item elemName="item6"  type="alphanumeric"
          size="30" emptyValue="COBOL"/>
    <Item elemName="item7"  type="alphanumeric"
          size="30" emptyValue="COBOL"/>
    <Item elemName="item8"  type="numeric"
          emptyValue="ZERO"/>
    <Item elemName="item9"  type="numeric"
          emptyValue="85"/>
    <Item elemName="item10" type="numeric"
          emptyValue="2001"/>
    <Item elemName="item11" type="numeric"
          emptyValue="LOW-VALUE"/>
  </Group>
 </BaseElement>
</Interface>>
```

（入力する XML ドキュメント）

```
<?xml version="1.0" encoding="Shift_JIS"?>

<table>
 <!-- 要素なし <item1> -->
 <item2></item2>
 <item3>    </item3>
 <!-- 要素なし <item4> -->
 <!-- 要素なし <item5> -->
 <!-- 要素なし <item6> -->
 <item7>COBOL</item7>
 <!-- 要素なし <item8> -->
 <!-- 要素なし <item9> -->
 <item10>2001</item10>
 <!-- 要素なし <item11> -->
</table>
```

（COBOL データ項目に格納される値）

データ項目名	値
item1	SPACE
item2	空白文字
item3	空白文字
item4	HIGH-VALUE
item5	LOW-VALUE
item6	"COBOL"
item7	"COBOL"
item8	ZERO

データ項目名	値
item9	"85"
item10	"2001"
item11	LOW-VALUE

(7) sign 属性

形式

sign="*符号種別*"

機能

COBOL の数字項目に対して PICTURE 句の演算符号"S"，および SIGN 句を指定します。

規則

- 符号種別には，次のどれかの値を指定します。これら以外の値を指定した場合，COBOL 原始プログラム生成時にエラーとなります。

unsigned

数字項目の PICTURE 句に演算符号"S"を付けません。

signed

数字項目の PICTURE 句に演算符号"S"を付けます。

leadingSeparate

数字項目に，PICTURE 句の演算符号"S"と「SIGN LEADING SEPARATE」を付けます。

- sign 属性を省略した場合，"unsigned"が仮定されます。
- sign 属性に指定できる符号種別は，type 属性に指定したデータ型によって異なります。指定できる type 属性の値と sign 属性の値の組み合わせを表 2-5 に示します。

表 2-5　指定できる type 属性の値と sign 属性の値の組み合わせ

type 属性の値	sign 属性の値		
	unsigned	signed	leadingSeparate
numeric	○	○	○
packed	○	○	×
binary	○	○	×
上記以外	×	×	×

（凡例）

○：指定できます。

×：指定できません。

指定例

XML 要素「UnsignedValue」「SignedValue」「LSSignedValue」に対応する COBOL データ項目に対して，それぞれ異なる演算符号や SIGN 句を設定する例を次に示します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (row)>
  <!ELEMENT row (UnsignedValue, SignedValue,
                 LSSignedValue)>
  <!ELEMENT UnsignedValue (#PCDATA)>
  <!ELEMENT SignedValue (#PCDATA)>
  <!ELEMENT LSSignedValue (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="row">
  <Group cobName="row">
   <Item elemName="UnsignedValue" type="numeric"
         sign="unsigned"/>
   <Item elemName="SignedValue" type="numeric"
         sign="signed"/>
   <Item elemName="LSSignedValue" type="numeric"
         sign="leadingSeparate"/>
  </Group>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 row.
 02 UnsignedValue PIC 9(9).
 02 SignedValue PIC S9(9).
 02 LSSignedValue PIC S9(9) SIGN LEADING SEPARATE.
```

(8)　trim 属性

形式

trim="*yes* | *no*"

機能

XML ドキュメントに出力されるデータ項目の値を整形するかどうかを指定します。

規則

- trim 属性には，値を整形する場合は"yes"，整形しない場合は"no"を指定します。それぞれを指定した場合の出力結果を表 2-6 に示します。

表 2-6　type 属性の値と trim 属性の値の組み合わせでの出力結果の相違

type 属性の値	出力結果	
	trim="yes"	trim="no"
alphanumeric	末尾の空白文字が削除されて，値が出力される。	size 属性で指定した長さの値が出力される。
national	末尾の全角空白文字が削除されて，値が出力される。	size 属性で指定した長さの値が出力される。
numeric packed binary	数値の前ゼロ（0）が削除されて，値が出力される。また，正の符号'＋'が削除される。	size 属性で指定した長さの値が出力される。また，sign 属性に unsigned 以外を指定した場合，値の左側に正の符号'＋'または負の符号'－'が出力される。
上記以外	trim 属性を指定できない。指定した場合，COBOL 原始プログラムの生成時にエラーとなる。	

- trim 属性に"yes"を指定した場合で符号付きの数値のとき，符号を除いた先頭から前ゼロ（0）が削除されます。また，+記号は削除されます。

（例）

+000004　→　4

-000004　→　-4

- trim 属性の指定を省略した場合，"yes"が仮定されます。
- trim 属性に"yes"または"no"以外の値を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

指定例

XML 要素「test01」,「test02」の値を，それぞれ次のように出力する例を示します。

- 「test01」の値（英数字）は，size 属性で指定した長さで出力する
- 「test02」の値（数値）は，前ゼロ（0）を削除して出力する

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (row)>
  <!ELEMENT row (test01, test02)>
  <!ELEMENT test01 (#PCDATA)>
  <!ELEMENT test02 (#PCDATA)>
]>
<table/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="row">
    <Item elemName="test01" type="alphanumeric"
          trim="no"/>
    <Item elemName="test02" type="numeric"
          trim="yes"/>
```

```
  </BaseElement>
</Interface>
```

(9)　nameOfFlagVar 属性

形式

nameOfFlagVar="*アクセス情報フラグの名称*"

機能

Interface 要素または BaseElement 要素の accessInfo 属性に"yes"を指定した場合に，Item 要素に対応するアクセス情報フラグの名称を指定します。

アクセス情報フラグについては，「3.2.1　アクセス情報フラグ」を参照してください。

規則

- nameOfFlagVar 属性の指定を省略した場合は，cobName 属性（省略時は elemName 属性）の名称に"-FLG"を追加した名称が XML アクセス用データ定義に生成されます。

  cobName 属性（省略時は elemName 属性）の名称は，26 文字以下で指定する必要があります。26 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。

- nameOfFlagVar 属性に指定する名称は，30 文字以下で指定する必要があります。30 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。

- nameOfFlagVar 属性には，COBOL データ項目名として使用できる名称を指定する必要があります。COBOL データ項目の名称に使用できない文字を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

- accessInfo 属性を省略した場合，または accessInfo 属性に"no"を指定した場合，nameOfFlagVar 属性の指定は無効となります。

  accessInfo 属性の指定例については，「3.3.3(1)　指定例」を参照してください。

- nameOfFlagVar 属性には，次に示す属性とは異なる名称を指定する必要があります。同じ名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

  nameOfFlagVar 属性と異なる名称を指定する属性

  - BaseElement 要素の cobName 属性※，nameOfBaseVar 属性
  - Group 要素の cobName 属性※，nameOfGroupVar 属性
  - Item 要素の cobName 属性※，nameOfFlagVar 属性，nameOfLengthVar 属性
  - Array 要素の nameOfCountVar 属性，nameOfTotalVar 属性
  - AttrItem 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性

注※

cobName 属性を省略した場合は，elemName 属性になります。

nameOfFlagVar 属性の指定例については，「3.2.1(3)　nameOfFlagVar 属性の指定例」を参照してください。

(10)　nameOfLengthVar 属性

形式

nameOfLengthVar="*データ長の名称*"

機能

Interface 要素または BaseElement 要素の accessInfo 属性に"yes"を指定した場合に，Item 要素に対応するデータ長の名称を指定します。

データ長については，「3.2.2　データ長」を参照してください。

規則

- nameOfLengthVar 属性の指定を省略した場合は，cobName 属性（省略時は elemName 属性）の名称に"-LEN"を追加した名称が XML アクセス用データ定義に生成されます。

  cobName 属性（省略時は elemName 属性）の名称は，26 文字以下で指定する必要があります。26 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。
- nameOfLengthVar 属性に指定する名称は，30 文字以下で指定する必要があります。30 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。
- nameOfLengthVar 属性には，COBOL データ項目名として使用できる名称を指定する必要があります。COBOL データ項目の名称に使用できない文字を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- accessInfo 属性の指定を省略した場合，または accessInfo 属性に"no"を指定した場合，nameOfLengthVar 属性の指定は無効となります。

  accessInfo 属性の指定例については，「3.3.3(1)　指定例」を参照してください。
- nameOfLengthVar 属性には，次に示す属性とは異なる名称を指定する必要があります。同じ名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

  nameOfLengthVar 属性と異なる名称を指定する属性

  - BaseElement 要素の cobName 属性※，nameOfBaseVar 属性
  - Group 要素の cobName 属性※，nameOfGroupVar 属性
  - Item 要素の cobName 属性※，nameOfFlagVar 属性，nameOfLengthVar 属性
  - Array 要素の nameOfCountVar 属性，nameOfTotalVar 属性
  - AttrItem 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性

注※

cobName 属性の指定を省略した場合は，elemName 属性になります。

nameOfLengthVar 属性の指定例については，「3.2.2(3)　nameOfLengthVar 属性の指定例」を参照してください。

(11)　verbatim 属性

形式

verbatim="*yes*|*no*"

機能

verbatim 属性に"yes"を指定した場合，elemName 属性で指定した要素の内容を文字列として，COBOL データ項目に対応づけます。指定した要素の内容に含まれるタグはすべて文字列とみなされます。

verbatim 属性に"yes"を指定した Item 要素は，Group 要素および DTD に定義した#PCDATA キーワードを持つ要素に対応づけられます。また，再帰的構造を持つ要素を XML ドキュメントに入出力できます。

規則

- verbatim 属性を指定する場合，type 属性に"alphanumeric"を指定する必要があります。type 属性に"alphanumeric"以外を指定して verbatim 属性を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- verbatim 属性を指定した場合，trim 属性は指定できません。verbatim 属性と trim 属性を同時に指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- verbatim 属性の指定を省略した場合，"no"が仮定されます。
- verbatim 属性に対応する要素の内側の空白文字と改行文字は，XML ドキュメントからそのまま入力されます。
- verbatim 属性に対応する要素の内側に実体参照文字がある場合，定義済みの実体参照は「付録 D.1　定義済み実体参照」の規則に従って変換されます。

指定例

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (item01, item02, item03)>
  <!ELEMENT item01 (#PCDATA)>
  <!ELEMENT item02 (#PCDATA)>
  <!ELEMENT item03 (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="root">
    <Item verbatim="yes" elemName="root"
          type="alphanumeric" size="200"/>
```

```
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 root PIC X(200).
```

（入力ドキュメント）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
  <item01>AAAAA</item01>
  <item02>BBBBB</item02>
  <item03>CCCCC</item03>
</root>
```

（COBOL データ項目に格納される値）

```
[改行]<item01>AAAAA</item01>[改行]<item02>BBBBB</item02>[改行]<item03>CCCCC</item03
>[改行]
```

注　[改行]は改行コード（0x0A）を表しています。

（出力ドキュメント）

```
<?xml version="1.0" encoding="Shift_JIS"?>

<root>
  <item01>AAAAA</item01>
  <item02>BBBBB</item02>
  <item03>CCCCC</item03>
</root>
```

出力ドキュメントには，COBOL データ項目の値が出力されます。COBOL データ項目に変更がない場合，入力ドキュメントと出力ドキュメントは等しくなります。

(12)　update 属性

形式

update="*yes*"

機能

Item 要素を更新の対象にするかどうかを指定します。更新機能については，「7.2.11　XML ドキュメントの更新」を参照してください。

規則

- update 属性に"yes"以外を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

(13)　emptyContentValue 属性

形式

emptyContentValue="*type 属性値に対応した COBOL の定数*"

機能

入力する XML ドキュメントの，要素の内容が空である場合，emptyContentValue 属性に指定した COBOL の定数を COBOL データ項目に設定します。また，XML ドキュメントの出力時に，emptyContentValue 属性に指定した COBOL の定数を COBOL データ項目に設定した場合，対応する要素が空要素として出力されます。

emptyContentValue 属性には，表 2-7 に示す type 属性値に対応した COBOL の値を指定できます。

入力する要素の内容と emptyContentValue 属性に指定した定数が同じ場合，要素が空であるかどうかを判定できません。必ず入力する要素の内容と emptyContentValue 属性の定数が異なる値になるように，emptyContentValue 属性の値を指定してください。

表 2-7　type 属性の指定値と emptyContentValue 属性に指定できる値

type 属性値	指定できる値							
	英字	英数字	日本語文字	SPACE	数値	ZERO	HIGH-VALUE	LOW-VALUE
alphanumeric	○	○	○	○	○	○	○	○
national	×	×	○	○	×	○	○	○
numeric	×	×	×	×	○	○	○	○
packed	×	×	×	×	○	○	×	×
binary	×	×	×	×	○	○	×	×
float	×	×	×	×	○	○	×	×
double	×	×	×	×	○	○	×	×

（凡例）

○：指定できます。

×：指定できません。

規則

- emptyContentValue 属性に指定する文字列の長さは，同じ Item 要素の size 属性に指定する COBOL データ項目の長さを超えないようにしてください。超えた場合，size 属性に指定した長さが有効となります。
- accessInfo 属性"yes"を指定した Interface 要素，BaseElement 要素の内側にある Item 要素に emptyContentValue 属性は指定できません。emptyContentValue 属性を指定した場合は COBOL 原始プログラムの生成時に警告メッセージが表示され，emptyContentValue 属性は無視されます。
- emptyContentValue 属性に指定した文字列が 160 文字を超えた場合，先頭から 160 文字が採用されます。このとき，警告メッセージが表示されます。
- 「ZERO」「SPACE」「HIGH-VALUE」「LOW-VALUE」は，emptyContentValue 属性の予約語です。これらの文字列は，COBOL の英数字定数として emptyContentValue 属性に指定できません。また，これらの文字列は，COBOL 言語仕様の表意定数と同じように，英大文字と英小文字が等価として扱われます。

- type 属性と emptyContentValue 属性に指定する値は，表 2-7 に記載された組み合わせだけが有効となります。これ以外の組み合わせを指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- emptyContentValue 属性に指定した数字が 18 けたを超える場合，COBOL 原始プログラムの生成時にエラーとなります。また，浮動小数点の仮数部が 16 けたを超える，指数部が 2 けたを超えるなど，浮動小数点の形式でない場合も，COBOL 原始プログラムの生成時にエラーとなります。

  なお，浮動小数点の形式で使用する「E」の文字は，大文字，小文字の両方を使用できます。
- emptyValue 属性と emptyContentValue 属性に同じ値を指定し，XML ドキュメントを出力した場合，emptyValue 属性が有効になります。ただし，対応する要素が省略できる要素でない場合は，その要素を削除できないため，emptyContentValue 属性が有効になります。

注意事項

- ATTLIST 属性のデフォルト指定で#FIXED 値を指定した属性の入力に対しては，emptyContentValue 属性を指定しても，無効となります。必ず，#FIXED 値で指定したデフォルト値を入力値として設定してください。
- 開始タグと終了タグ間の内容が空白文字（スペース，タブおよび改行）だけの場合でも空要素とみなしません。空要素とみなすのは，空タグの場合，および開始タグと終了タグ間の内容がない（<Item></Item>）場合だけです。

(14)　invalidCharValue 属性

形式

invalidCharValue="*type 属性値に対応した COBOL の定数*"

機能

入力する XML ドキュメントに対応する要素の内容に不当な文字がある場合，invalidCharValue 属性に指定した COBOL の定数が COBOL データ項目に返されます。

invalidCharValue 属性には，表 2-8 に示す type 属性値に対応した COBOL の値を指定できます。

入力時にチェックする文字は「9.2　入出力時に不当な文字をチェックする機能」の「表 9-3　入力時にチェックする文字」の「-chkchar オプション指定あり」を参照してください。

表 2-8　type 属性の指定値と invalidCharValue 属性に指定できる値

type 属性値	指定できる値							
	英字	英数字	日本語文字	SPACE	数値	ZERO	HIGH-VALUE	LOW-VALUE
alphanumeric	○	○	○	○	○	○	○	○
national	×	×	○	○	×	○	○	○
numeric	×	×	×	×	○	○	○	○
packed	×	×	×	×	○	○	×	×
binary	×	×	×	×	○	○	×	×

type 属性値	指定できる値							
	英字	英数字	日本語文字	SPACE	数値	ZERO	HIGH-VALUE	LOW-VALUE
float	×	×	×	×	○	○	×	×
double	×	×	×	×	○	○	×	×

（凡例）

○：指定できます。

×：指定できません。

規則

- invalidCharValue 属性に指定する文字列の長さは，同じ Item 要素の size 属性に指定する COBOL データ項目の長さを超えないようにしてください。超えた場合，size 属性に指定した長さが有効となります。
- accessInfo 属性"yes"を指定した Interface 要素，BaseElement 要素の内側にある Item 要素に invalidCharValue 属性は指定できません。invalidCharValue 属性を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示され，invalidCharValue 属性は無視されます。
- invalidCharValue 属性に指定した文字列が 160 文字を超えた場合，先頭から 160 文字が採用されます。このとき，警告メッセージが表示されます。
- 「ZERO」「SPACE」「HIGH-VALUE」「LOW-VALUE」は，invalidCharValue 属性の予約語です。これらの文字列は，COBOL の英数字定数として invalidCharValue 属性に指定できません。また，これらの文字列は，COBOL 言語仕様の表意定数と同じように，英大文字と英小文字が等価として扱われます。
- type 属性と invalidCharValue 属性に指定する値は，表 2-8 に記載された組み合わせだけが有効となります。これ以外の組み合わせを指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- invalidCharValue 属性に指定した数字が 18 けたを超える場合，COBOL 原始プログラムの生成時にエラーとなります。また，浮動小数点の仮数部が 16 けたを超える，指数部が 2 けたを超えるなど，浮動小数点の形式でない場合も，COBOL 原始プログラムの生成時にエラーとなります。

  なお，浮動小数点の形式で使用する「E」の文字は，大文字，小文字の両方を使用できます。
- invalidCharValue 属性と overflowValue 属性の両方を指定した場合で，XML ドキュメントからの入力値に不当文字があり，かつ size 属性に指定した長さを超えていると，invalidCharValue 属性が有効になります。

注意事項

invalidCharValue 属性を指定した場合でも，XML アクセスルーチンのステータスは「9.2　入出力時に不当な文字をチェックする機能」の-chkchar オプション指定の有無の条件に従ったステータスが返ります。

(15) overflowValue 属性

形式

overflowValue="*type 属性値に対応した COBOL の定数*"

機能

入力する XML ドキュメントに対応する要素の内容がオーバフローした場合，overflowValue 属性に指定した COBOL の定数が COBOL データ項目に設定されます。

overflowValue 属性には，表 2-9 に示す type 属性値に対応した COBOL の値を指定できます。

入力値のオーバフローの条件は，「9.1　入力時のオーバフローをステータスで返す機能」の表 9-1 を参照してください。

表 2-9　type 属性の指定値と overflowValue 属性に指定できる値

type 属性値	指定できる値							
	英字	英数字	日本語文字	SPACE	数値	ZERO	HIGH-VALUE	LOW-VALUE
alphanumeric	○	○	○	○	○	○	○	○
national	×	×	○	○	×	○	○	○
numeric	×	×	×	×	○	○	○	○
packed	×	×	×	×	○	○	×	×
binary	×	×	×	×	○	○	×	×
float	×	×	×	×	○	○	×	×
double	×	×	×	×	○	○	×	×

（凡例）

○：指定できます。

×：指定できません。

規則

- overflowValue 属性に指定する文字列の長さは，同じ Item 要素の size 属性に指定する COBOL データ項目の長さを超えないようにしてください。超えた場合，size 属性に指定した長さが有効となります。
- accessInfo 属性"yes"を指定した Interface 要素，BaseElement 要素の内側にある Item 要素に overflowValue 属性は指定できません。overflowValue 属性を指定した場合は COBOL 原始プログラムの生成時に警告メッセージが表示され，overflowValue 属性は無視されます。
- overflowValue 属性に指定した文字列が 160 文字を超えた場合，先頭から 160 文字が採用されます。このとき，警告メッセージが表示されます。
- 「ZERO」「SPACE」「HIGH-VALUE」「LOW-VALUE」は，overflowValue 属性の予約語です。これらの文字列は，COBOL の英数字定数として overflowValue 属性に指定できません。また，

これらの文字列は，COBOL 言語仕様の表意定数と同じように，英大文字と英小文字が等価として扱われます。

- type 属性と overflowValue 属性に指定する値は，表 2-9 に記載された組み合わせだけが有効となります。これ以外の組み合わせを指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- overflowValue 属性に指定した数字が 18 けたを超える場合，COBOL 原始プログラムの生成時にエラーとなります。また，浮動小数点の仮数部が 16 けたを超える，指数部が 2 けたを超えるなど，浮動小数点の形式でない場合も，COBOL 原始プログラムの生成時にエラーとなります。

  なお，浮動小数点の形式で使用する「E」の文字は，大文字，小文字の両方を使用できます。
- invalidCharValue 属性と overflowValue 属性の両方を指定した場合で，XML ドキュメントからの入力値に不当文字があり，かつ size 属性に指定した長さを超えていると，invalidCharValue 属性の値が設定されます。

注意事項

overflowValue 属性を指定した場合でも XML アクセスルーチンのステータスは「9.1　入力時のオーバフローをステータスで返す機能」の-chkovflow オプション指定の有無の条件に従ったステータスが返ります。

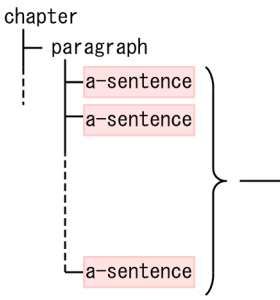
2.3.5　Array 要素（繰り返し要素の定義）

Array 要素は，繰り返し要素を OCCURS 句を持つ COBOL データ項目に対応づける要素です。

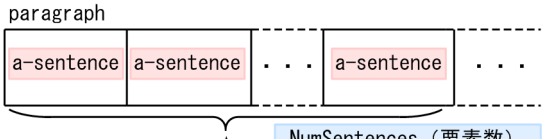
●DDFの指定例

```
<Interface interfaceName="EXAMPLE">
   <BaseElement elemName="paragraph">
     <Group cobName="paragraph">
       <Array max="20" nameOfCountVar="NumSentences">
         <Item elemName="a-sentence" type="string" size="1000"/>
       </Array>
     </Group>
   </BaseElement>
</Interface>
```

●XMLドキュメントの階層構造



●生成されるCOBOLのデータ項目




```
01 paragraph.
   02 a-sentence PIC X(1000) OCCURS 20.
   02 NumSentences  PIC 9(9) USAGE COMP.
```

Array 要素は，子要素として Group 要素，Item 要素，または AttrItem 要素のどれかを持ちます。

形式

<Array max="*要素の最大繰り返し回数*"
[nameOfCountVar="*繰り返し入出力数の名称*"]
[nameOfTotalVar="*繰り返し全要素数の名称*"]
[countVar="*yes*|*no*"] >
{*Group 要素* | *Item 要素* | *AttrItem 要素*}
</Array>

指定例

Array 要素を使って，繰り返し要素「sentence」を COBOL の OCCURS 句付きのデータ項目「a-sentence」に対応づける例を次に示します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE book [
  <!ELEMENT book (chapter)*>
  <!ELEMENT chapter (paragraph)*>
  <!ELEMENT paragraph (sentence)*>
  <!ELEMENT sentence (#PCDATA)>
]>
<book/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
   <BaseElement elemName="paragraph">
     <Group cobName="paragraph">
       <Array max="20">
         <Item elemName="sentence"
               cobName="a-sentence"
               type="alphanumeric"
               size="1000"/>
       </Array>
     </Group>
   </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 paragraph.
 02 a-sentence PIC X(1000) OCCURS 20.
 02 a-sentence-COUNT PIC 9(9) USAGE COMP.
```

(1) max 属性

形式

max="*配列の最大要素数*"

機能

要素の繰り返し回数の最大値を指定します。

規則

- max 属性に指定できる値は，1～16,777,215 の範囲です。この範囲を超える値を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- 読み込もうとした XML 要素の数が，Array 要素の max 属性に指定した値よりも小さい場合，余った COBOL データ項目（XML ドキュメントから値を入力しなかったデータ項目）は表 2-10 に示す値になります。

表 2-10　余った COBOL データ項目の値

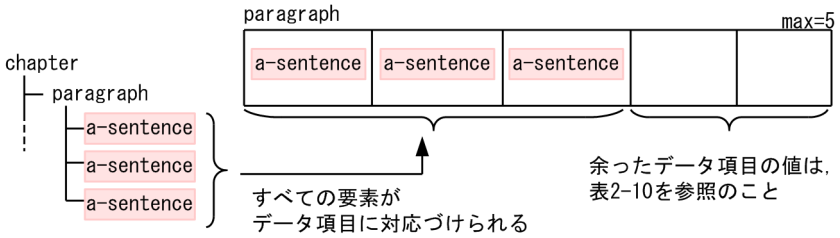
Array 要素のに対する countVar 属性の指定	余った COBOL データ項目の値
countVar="yes"または countVar を指定していない	不定となる
countVar="no"	emptyValue 属性値を設定する

- 読み込もうとした XML 要素の数が，max 属性に指定した値よりも大きい場合，最大値を超えた部分のデータは，COBOL データ項目に読み込まれません。

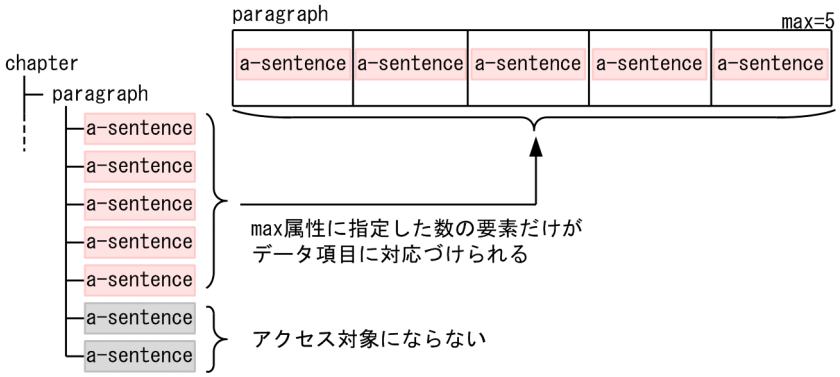
（例）

max 属性に指定した値と，XML ドキュメント中の要素の数による動作の違いを次に示します。

●（max属性に指定した値）＞（XMLドキュメントの要素数）の場合



●（max属性に指定した値）＜（XMLドキュメントの要素数）の場合



(2) nameOfCountVar 属性

形式

nameOfCountVar="*繰り返し入出力数の名称*"

機能

Array 要素に対応した繰り返し項目の入出力数を扱うための繰り返し入出力数の名称を指定します。XML ドキュメント入力時には，繰り返し項目の入力した数が設定されます。XML ドキュメント出力時には，出力する繰り返し項目の数を設定できます。

Interface 要素と BaseElement 要素の accessInfo 属性に"no"を指定した場合，要素の繰り返し入出力数を格納する COBOL データ項目は繰り返し項目と同じレベルに生成されます。accessInfo 属性に"yes"を指定した場合，入出力データ情報項目の中に生成されます。

規則

- nameOfCountVar 属性の指定を省略した場合は，Array 要素の直下に含まれる Item 要素または Group 要素の cobName 属性（省略時は elemName 属性）の名称に"-COUNT"を追加した名称が XML アクセス用データ定義に生成されます。

  cobName 属性（省略時は elemName 属性）の名称は，24 文字以下で指定する必要があります。24 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。

- nameOfCountVar 属性に指定する名称は，30 文字以下で指定する必要があります。30 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。
- nameOfCountVar 属性には，COBOL データ項目名に使用できる名称を指定する必要があります。COBOL データ項目名として使用できない名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- nameOfCountVar 属性には，次に示す属性とは異なる名称を指定する必要があります。同じ名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

  #### nameOfCountVar 属性と異なる名称を指定する属性

  - BaseElement 要素の cobName 属性※，nameOfBaseVar 属性
  - Group 要素の cobName 属性※，nameOfGroupVar 属性
  - Item 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性
  - Array 要素の nameOfCountVar 属性，nameOfTotalVar 属性
  - AttrItem 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性

注※

cobName 属性の指定を省略した場合は，elemName 属性になります。

指定例 1

Array 要素に含まれる Group 要素が elemName 属性を持つ場合の指定例を次に示します。入力時に nameOfCountVar 属性で指定した COBOL データ項目の値は，elemName 属性で指定した要素の繰り返し回数となります。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table1 [
<!ELEMENT table1 (row)*>
<!ELEMENT row (item1?,item2?)>
<!ELEMENT item1 (#PCDATA)>
<!ELEMENT item2 (#PCDATA)>
]>
<table1/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="table1">
    <Group cobName="table1">
      <Array max="100">
        <Group elemName="row">
          <Item elemName="item1"
                type="alphanumeric" size="10"/>
          <Item elemName="item2"
                type="alphanumeric" size="10"/>
        </Group>
      </Array>
    </Group>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 table1.
 02 row OCCURS 100.
  03 item1 PIC X(10).
  03 item2 PIC X(10).
 02 row-COUNT PIC 9(9) USAGE COMP.
```

(入力する XML ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<table1>
  <row>
    <item2/>
  </row>
  <row>
  </row>
  <row>
    <item1/>
  </row>
</table1>
```

この場合，Array 要素に含まれる Group 要素の elemName 属性に要素 row を指定しているので，row-COUNT の値は row 要素の出現回数である 3 になります。

指定例 2

Array 要素に含まれる Group 要素が elemName 属性を持たない場合の指定例を次に示します。入力時の nameOfCountVar 属性で指定した COBOL データ項目の値は，集団項目への要素の入力の繰り返し回数となります。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table1 [
<!ELEMENT table1 (row)*>
<!ELEMENT row (item1?,item2?)>
<!ELEMENT item1 (#PCDATA)>
<!ELEMENT item2 (#PCDATA)>
]>
<table1/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="table1">
    <Group cobName="table1">
      <Array max="100">
        <Group cobName="row">
          <Item elemName="item1"
                type="alphanumeric" size="10"/>
          <Item elemName="item2"
                type="alphanumeric" size="10"/>
        </Group>
      </Array>
    </Group>
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 table1.
 02 row OCCURS 100.
  03 item1 PIC X(10).
  03 item2 PIC X(10).
 02 row-COUNT PIC 9(9) USAGE COMP.
```

（入力する XML ドキュメント）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<table1>
  <row>
    <item2/>
  </row>
  <row>
  </row>
  <row>
    <item1/>
```

```
  </row>
</table1>
```

この場合，Array 要素に含まれる Group 要素が elemName 属性を持たないので，集団項目への要素の入力の繰り返し回数が row-COUNT の値となります。また，2 番目の row 要素には要素が含まれないため，この要素は繰り返し回数には含まれません。

要素の入力は，DDF に記述した順序で行われます。この例の場合，要素「item2」を入力した時点で 1 回目の繰り返しが終了し，要素「item1」を入力した時点で繰り返しが 2 回となるため，row-COUNT の値は 2 となります。

指定例 3

要素「sentence」の読み込み，または書き込みを繰り返した回数を格納する COBOL データ項目の名称を「NumSentences」に指定する例を次に示します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE book [
  <!ELEMENT book (chapter)*>
  <!ELEMENT chapter (paragraph)*>
  <!ELEMENT paragraph (sentence)*>
  <!ELEMENT sentence (#PCDATA)>
]>
<book/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
   <BaseElement elemName="paragraph">
     <Group cobName="paragraph">
       <Array max="20"
              nameOfCountVar="NumSentences">
         <Item elemName="sentence"
               cobName="a-sentence"
               type="alphanumeric"
               size="1000"/>
       </Array>
     </Group>
   </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 paragraph.
  02 a-sentence PIC X(1000) OCCURS 20.
  02 NumSentences PIC 9(9) USAGE COMP.
```

指定例 4

Array 要素の内側の Group 要素に elemName 属性が指定されていない場合，入力数を表すデータ項目の値（繰り返し入出力数）は，集団項目へ要素を入力する繰り返し回数となります。例えば，次のような DTD で要素「item1」を対応づけていない場合，group1 要素だけが集団項目に入力されます。

その結果，内側の Array 要素に対応する繰り返しはオーバフローし，grp0-COUNT の値は 1，group1-COUNT(1)の値は 4 となります。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root ((group1*, item1)*)>
  <!ELEMENT group1 (item2?)>
  <!ELEMENT item1 (#PCDATA)>
  <!ELEMENT item2 (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
<BaseElement elemName="root" cobName="BE">
 <Group elemName="root">
 <Array max="4">
  <Group cobName="grp0">
    <Array max="4">
    <Group elemName="group1">
      <Item elemName="item2"/>
    </Group>
    </Array>
  </Group>
 </Array>
 </Group>
</BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 root.
 02 grp0 OCCURS 4.
  03 group1 OCCURS 4.
   04 item2 PIC X(100).
  03 group1-COUNT PIC 9(9) USAGE COMP.
 02 grp0-COUNT PIC 9(9) USAGE COMP.
```

(入力する XML ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
  <group1> <item2>item2a1</item2> </group1>
  <group1> <item2>item2a2</item2> </group1>
  <group1> <item2>item2a3</item2> </group1>
  <group1> <item2>item2a4</item2> </group1>
  <item1/>
  <group1> <item2>item2b1</item2> </group1>
  <group1> <item2>item2b2</item2> </group1>
  <group1> <item2>item2b3</item2> </group1>
  <group1> <item2>item2b4</item2> </group1>
  <item1/>
</root>
```

入出力データ情報定義機能を使用した場合の nameOfCountVar 属性の指定例については，「3.2.4(1) nameOfTotalVar 属性と nameOfCountVar 属性の指定例」を参照してください。

(3) nameOfTotalVar 属性

形式

nameOfTotalVar="*繰り返し全要素数の名称*"

機能

Interface 要素または BaseElement 要素の accessInfo 属性に"yes"を指定した場合に，繰り返し全要素数の名称を指定します。

繰り返し全要素数には，Array 要素に対応づけた XML ドキュメント内の繰り返し要素の繰り返し回数が格納されます。

規則

- nameOfTotalVar 属性の指定を省略した場合は，cobName 属性（省略時は elemName 属性）の名称に"-TOTAL"を追加した名称が XML アクセス用データ定義に生成されます。

  cobName 属性（省略時は elemName 属性）の名称は，24 文字以下で指定する必要があります。24 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。
- nameOfTotalVar 属性に指定する名称は，30 文字以下で指定する必要があります。30 文字を超える名称を指定した場合，COBOL 原始プログラムの生成時に警告メッセージが表示されます。
- nameOfTotalVar 属性には，COBOL データ項目名として使用できる名称を指定する必要があります。COBOL データ項目名に使用できない文字を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- accessInfo 属性に指定がない場合，または accessInfo 属性に"no"を指定した場合，nameOfTotalVar 属性の指定は無効となります。

  accessInfo 属性の指定例については，「3.3.4　Array 要素」を参照してください。
- nameOfTotalVar 属性と，次に示す属性とは異なる名称を指定する必要があります。同じ名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

  nameOfTotalVar 属性と異なる名称を指定する属性

  - BaseElement 要素の cobName 属性※，nameOfBaseVar 属性
  - Group 要素の cobName 属性※，nameOfGroupVar 属性
  - Item 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性
  - Array 要素の nameOfCountVar 属性，nameOfTotalVar 属性
  - AttrItem 要素の cobName 属性※，nameOfLengthVar 属性，nameOfFlagVar 属性

注※

cobName 属性の指定を省略した場合は，elemName 属性になります。

入出力データ情報定義機能を使用した場合の nameOfTotalVar 属性の指定例については，「3.2.4(1) nameOfTotalVar 属性と nameOfCountVar 属性の指定例」を参照してください。

(4) countVar 属性

形式

countVar="*yes*|*no*"

機能

繰り返し入出力数に対応する COBOL データ項目を出力するかどうかを指定します。

countVar 属性に"yes"を指定した場合，繰り返し入出力数を示すデータ項目が XML アクセス用データ定義に生成されます。countVar 属性に"no"を指定した場合，繰り返し入出力数を示すデータ項目は生成されません。

規則

- countVar 属性の指定を省略した場合は，"yes"が仮定されます。
- countVar 属性に"yes""no"以外の値を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- countVar 属性に"yes"を指定した場合，Array 要素の直下にある Item 要素の emptyValue 属性は無効となります。
- countVar 属性に"no"を指定した「＋」付きの繰り返し要素で全要素を出力しない場合，CBLXML-WR-*Interface-BaseElement* アクセスルーチンでステータス 7 は返りません。
- countVar 属性に"no"を指定した場合，繰り返し要素の特定の要素を出力するかどうかを決定する規則を次に示します。

Array 要素の内側に Item 要素がある場合（accessInfo="no"指定時）

繰り返し項目の要素に含まれる項目が emptyValue 属性で指定した値に等しければ，その要素は出力されません。

Array 要素の内側に Item 要素がある場合（accessInfo="yes"指定時）

アクセス情報フラグに CBLXML-FLAG-MISSING を指定すると，その要素は出力されません。

アクセス情報フラグの設定についての詳細については，「3.6　XML ドキュメント書き込み時に設定する入出力データ情報項目」を参照してください。

Array 要素の内側に Group 要素がある場合

Array 要素の内側にある Group 要素が出力されなければ，繰り返し要素の特定の要素は出力されません。

指定例 1

countVar 属性に"no"を指定した Array 要素の内側に Item 要素がある場合の例を次に示します。

（DTD の例）

```
<?xml version=”1.0” encoding=”Shift_JIS”?>
<!DOCTYPE book [
```

```
  <!ELEMENT book (chapter)*>
  <!ELEMENT chapter (paragraph)*>
  <!ELEMENT paragraph (sentence)*>
  <!ELEMENT sentence (#PCDATA)>
]>
<book/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
   <BaseElement elemName="paragraph">
     <Group cobName="paragraph">
       <Array max="5" countVar="no">
         <Item elemName="sentence"
        cobName="a-sentence" type="alphanumeric"
        size="100" emptyValue="SPACE"/>
       </Array>
     </Group>
   </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 paragraph.
 02 a-sentence PIC X(100) OCCURS 5.
```

(COBOL データ項目の値)

```
MOVE "This is sentence 1." TO a-sentence(1).
MOVE SPACE TO a-sentence(2).
MOVE "This is sentence 3." TO a-sentence(3).
MOVE SPACE TO a-sentence(4).
MOVE SPACE TO a-sentence(5).
```

(出力ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<book>
 <chapter>
<paragraph>
 <sentence>This is sentence 1.</sentence>
 <sentence>This is sentence 3.</sentence>
</paragraph>
</chapter>
</book>
```

指定例 2

countVar 属性に"no"を指定した Array 要素の内側に Group 要素がある場合の例を次に示します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
<!ELEMENT table (person)*>
<!ELEMENT person (name?,address?)>
```

```
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
]>
<table/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="table">
    <Group cobName="table1">
      <Array max="3" countVar="no">
        <Group elemName="person">
          <Item elemName="name" cobName="name1"
          emptyValue="SPACE"/>
          <Item elemName="address"
          cobName="address1" emptyValue="SPACE"/>
        </Group>
      </Array>
    </Group>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 table1.
 02 person OCCURS 3.
  03 name1 PIC X(100).
  03 address1 PIC X(100).
```

(COBOL データ項目の値)

```
          MOVE 'name-1' TO name1(1).
          MOVE 'address-1' TO address1(1).
          MOVE SPACE TO name1(2).
          MOVE SPACE TO address1(2).
          MOVE 'name-3' TO name1(3).
          MOVE SPACE TO address1(3).
```

(出力ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<table>
  <person>
    <name>name-1</name>
    <address>address-1</address>
  </person>
  <person>
    <name>name-3</name>
  </person>
</table>
```

2.3.6 AttrItem 要素

AttrItem 要素は，XML 要素の属性を COBOL データ項目に対応づけます。

AttrItem 要素は，要素内容が EMPTY と宣言された要素に対しても指定できます。要素内容が EMPTY である要素に対して指定した場合，属性に対するデータ項目だけが生成されます。

形式

<AttrItem elemName="*XML の要素名*"
attrName="*属性名*"
[cobName="*AttrItem 要素に対応する名称*"]
[type="*type 属性値*"]
[size="*けた数*"]
[fractionalDigits="*けた数*"]
[emptyValue="*type 属性値に対応した COBOL の定数*"]
[sign="*符号種別*"]
[trim="*データ項目の整形の有無*"]
[nameOfFlagVar="*アクセス情報フラグの名称*"]
[nameOfLengthVar="*データ長の名称*"]
[emptyContentValue="*type 属性値に対応した COBOL の定数*"]
[invalidCharValue="*type 属性値に対応した COBOL の定数*"]
[overflowValue="*type 属性値に対応した COBOL の定数*"]
/>

規則

- DTD の属性に指定されたデフォルト値より小さい値を AttrItem 要素の size 属性に指定した場合，COBOL 原始プログラムの生成時にエラーとなります。
- DTD の属性に指定できるデフォルト値の最大値は，英数字項目および日本語項目の場合は 160 文字，数字項目の場合は 18 けた（符号，小数点は含まない）です。最大値を超える値を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

指定例

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (sale)*>
  <!ELEMENT sale (brand, style)>
  <!ATTLIST sale color (black | white | brown)
   #REQUIRED >
  <!ATTLIST style age (child | adult) #REQUIRED >
  <!ELEMENT brand (#PCDATA)>
  <!ELEMENT style (#PCDATA)>
]>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="sale">
    <Group elemName="sale">
      <AttrItem elemName="sale" attrName="color"
          cobName="colorname" type="alphanumeric"
           size="10"/>
      <Item elemName="style" size="100"/>
      <AttrItem elemName="style" attrName="age"
           type="alphanumeric" size="5"/>
    </Group>
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 sale.
  02 colorname PIC X(10).
  02 style PIC X(100).
  02 style-age PIC X(5).
```

(1)　elemName 属性

形式

elemName="*XML の要素名*"

機能

AttrItem 要素で対応づける属性が関連づけられている XML 要素の名称を指定します。

elemName 属性の規則，指定例については，「2.3.4(1)　elemName 属性」を参照してください。

(2)　attrName 属性

形式

attrName="*属性名*"

機能

AttrItem 要素で対応づける XML 要素の属性の名称を指定します。

規則

- AttrItem 要素に指定する属性の名称は，elemName 属性で指定した要素の属性として DTD 内で宣言されている必要があります。
- cobName 属性を省略した場合，elemName 属性の値と attrName 属性の値を'-'（ハイフン）でつないだ名称が COBOL データ項目名になります。

  elemName 属性値の長さと attrName 属性値の長さの合計は 29 文字以下でなければなりません。

(3)　cobName 属性

形式

cobName="*COBOL データ項目名*"

機能

attrName 属性の値の代わりに COBOL データ項目の名称を指定します。attrName 属性に指定する XML 要素の名称が，COBOL データ項目名に使用できない場合などに指定します。

規則

二つ以上の異なる属性を，同じ COBOL データ項目に対応づけられません。

cobName 属性の指定例については，「2.3.4(2)　cobName 属性」を参照してください。

(4)　type 属性

形式

type="*type 属性値*"

機能

AttrItem 要素に対応する COBOL データ項目の形式を指定します。

type 属性に指定できる値と，type 属性の値に対応する COBOL データ項目を表 2-11 に示します。

表 2-11　type 属性に指定できる値

type 属性の値	COBOL データ項目
alphanumeric	英数字項目
national	日本語項目
numeric	数字項目（外部 10 進形式）

規則

- type 属性を省略した場合，type="alphanumeric"が仮定されます。
- type 属性に表 2-11 に示した type 属性値以外を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

type 属性の指定例については，「2.3.4(3)　type 属性」を参照してください。

(5)　その他の属性

次に示す属性の形式，機能，規則，指定例については，「2.3.4　Item 要素（要素の対応づけの定義）」を参照してください。ただし，Item 要素と記載されている部分は AttrItem 要素と読み替えてください。

- size 属性
- fractionalDigits 属性

- emptyValue 属性
- sign 属性
- trim 属性
- nameOfFlagVar 属性
- nameOfLengthVar 属性
- emptyContentValue 属性
- invalidCharValue 属性
- overflowValue 属性

3 入出力データ情報定義機能

入出力データ情報定義機能を使用すると，XML データの入出力状態を取得または定義できます。

この章では，入出力データ情報定義機能と DDL 要素について説明します。

3.1　入出力データ情報定義機能の使用方法

入出力データ情報定義機能を使用すると，入力した XML 要素や属性の入力状態を取得できます。また，XML 要素の出力状態を設定することもできます。

使用方法

Interface 要素の下位のすべての要素で入出力データ情報定義機能を使用する場合は，Interface 要素の accessInfo 属性に"yes"を指定します。また，BaseElement 要素ごとに入出力データ情報定義機能を使用するかどうかを指定する場合は，BaseElement 要素の accessInfo 属性に"yes"または"no"を指定します。

Interface 要素の accessInfo 属性の指定例については「2.3.1(2)　accessInfo 属性」を，BaseElement 要素の accessInfo 属性の指定例については「2.3.2(3)　accessInfo 属性」を参照してください。

例

入出力データ情報定義機能を使用した場合の生成例を次に示します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (item01, group01)>
  <!ELEMENT item01  (#PCDATA)>
  <!ELEMENT group01 (item02, item03)*>
  <!ELEMENT item02  (#PCDATA)>
  <!ELEMENT item03  (#PCDATA)>
]>
<root/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE"
    accessInfo="yes">
  <BaseElement elemName="root">
    <Group cobName="root">
      <Item elemName="item01"
            type="alphanumeric" size="10" />
      <Array max="10">
        <Group cobName="group01">
          <Item elemName="item02"
                type="alphanumeric" size="10" />
          <Item elemName="item03"
                type="alphanumeric" size="10" />
        </Group>
      </Array>
    </Group>
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
 01 root-BASE.          …1.
  02 root.               …2.
```

```
   03 item01 PIC X(10).
   03 group01 OCCURS 10.
    04 item02 PIC X(10).
    04 item03 PIC X(10).
* Access Information                    …3.
  02 root-FLG PIC 1(32) USAGE BIT.
  02 root-GROUP.
   03 item01-FLG PIC 1(32) USAGE BIT.
   03 item01-LEN PIC 9(9) COMP.
   03 group01-TOTAL PIC 9(9) COMP.
   03 group01-COUNT PIC 9(9) COMP.
   03 group01-GROUP OCCURS 10.
    04 item02-FLG PIC 1(32) USAGE BIT.
    04 item02-LEN PIC 9(9) COMP.
    04 item03-FLG PIC 1(32) USAGE BIT.
    04 item03-LEN PIC 9(9) COMP.
```

（XML アクセスルーチンを呼び出す COBOL プログラム）

```
        :
    CALL 'CBLXML-WR-EXAMPLE-root'
         USING XML-POINTER root-BASE        …4.
        RETURNING CBLXML-RETURN-CODE.
        :
```

（説明）

1. 入出力データ情報定義機能使用時の先頭集団項目名を表します。アクセスルーチンの引数に指定します。
2. 入出力する XML 要素や属性に対応したデータ項目です。この例では，集団項目「root」と「root」に含まれる下位項目が対応したデータ項目となります。
3. コメント行* Access Information 以降は，入出力する XML 要素に対応した入出力データ情報項目となります。
4. XML アクセス用データ定義を指定します。

3.2　入出力データ情報項目

入出力データ情報定義機能を使用した場合，入出力する XML 要素（COBOL データ項目）に対応する入出力データ情報項目が生成されます。

入出力データ情報項目の内容を，表 3-1 に示します。

表 3-1　入出力データ情報項目の内容

入出力データ情報項目	機能
アクセス情報フラグ	入力した XML ドキュメントの要素，属性の詳細な情報を取得する。出力する XML ドキュメントの要素，属性の状態を設定する。
データ長	入力した XML ドキュメントの要素，属性のデータ長を取得する。出力する XML ドキュメントの要素，属性の出力長を設定する。
繰り返し全要素数	XML ドキュメントの要素，属性の繰り返し要素の数を取得する。
繰り返し入出力数	XML ドキュメントから入力した繰り返し要素の数を取得する。XML ドキュメントへ出力する繰り返し要素の数を設定する。

次に，入出力データ情報項目が生成される DDL の要素を表 3-2 に示します。

表 3-2　入出力データ情報項目と DDL の要素の対応づけ

入出力データ情報項目	項目が生成される DDL の要素
アクセス情報フラグ	Group 要素，Item 要素，AttrItem 要素
データ長	Item 要素，AttrItem 要素
繰り返し全要素数	Array 要素
繰り返し入出力数	Array 要素

次に，入出力データ情報項目中の各項目の詳細について説明します。

3.2.1　アクセス情報フラグ

アクセス情報フラグには，入力した要素および属性の状態が格納されます。また，出力時にアクセス情報フラグを設定すると，要素および属性の出力状態を設定できます。アクセス情報フラグは 32 けたのブール項目として，cblxml コマンドが生成する XML アクセス用データ定義に生成されます。

アクセス情報フラグに対応する COBOL のデータ項目の名称

Group 要素，Item 要素，または AttrItem 要素の nameOfFlagVar 属性に指定した名称が，アクセス情報フラグに対応する COBOL のデータ項目の名称になります。

nameOfFlagVar 属性の指定を省略した場合は，cobName 属性（省略時は elemName 属性）の名称に"-FLG"を追加した名称が，アクセス情報フラグに対応する COBOL のデータ項目の名称になります。

次に，アクセス情報フラグの値について，説明します。

(1)　入力時のアクセス情報フラグ

入力時に設定されるアクセス情報フラグの詳細を，表 3-3 に示します。なお，XML ドキュメント入力時には，複数のフラグ値が設定される場合があります。

表 3-3　入力時のアクセス情報フラグ

登録集原文"CBLXMLRC.cbl"の定義名（78 レベル）	アクセス情報フラグ値	アクセス情報フラグの意味
CBLXML-FLAG-MISSING	B'100000～0'	要素または属性がない。
CBLXML-FLAG-EMPTY	B'010000～0'	要素または属性の値が空である。※1
CBLXML-FLAG-INVAL-CHAR	B'001000～0'	要素または属性の値に Item 要素の type 属性に指定した COBOL データ形式で無効な文字が含まれている。※2
CBLXML-FLAG-OVERFLOW	B'000100～0'	要素，属性の値の長さが Item 要素の size 属性に指定した値を超えている。※3

注※1

開始タグと終了タグ間の内容が空白文字（スペース，タブおよび改行）だけの場合でも空要素とみなしません。空要素とみなすのは，空タグ，および開始タグと終了タグ間の内容がない（<Item></Item>）場合だけです。

注※2

無効な文字と判定される条件を次に示します。

- type 属性に"national"を指定して，COBOL データ項目に 1 バイト文字を入力した場合
  （不当な 1 バイト文字は，「＝」（げた記号）に置き換えられます。「＝」（げた記号）は 2 バイト文字であるため，size 属性で指定した長さを超えて CBLXML-FLAG-OVERFLOW が同時に設定される場合があります）
  不当文字については「9.2 入出力時に不当な文字をチェックする機能」の「表 9-3 入力時にチェックする文字」を参照してください。
- type 属性に"numeric"，"packed"，"binary"，"float"，または"double"を指定して，COBOL データ項目に数値以外の値を入力した場合
- sign 属性に"unsigned"を指定して，負値を入力した場合
  （ただし，-0 は+0 が仮定されるため，このフラグは設定されません）

注※3

要素，属性の値の長さと Item 要素の size 属性に指定した値を比較する場合の条件を次に示します。

- 数字項目（"numeric"，"packed"，"binary"）については，数値の左側の 0 を削除したけた数と size 属性のけた数を比較して大小を判定します。ただし，符号はけた数に含みません。

- 浮動小数点数（"float"，"double"）については，システムで定義した浮動小数点数の絶対値を超えた場合に size 属性の値を超えたと判定されます。システムで定義した浮動小数点数の絶対値については，ご使用のシステムのマニュアルを参照してください。

(2) 出力時のアクセス情報フラグ

出力時に設定するアクセス情報フラグの詳細を表 3-4 に示します。

表 3-4 出力時のアクセス情報フラグ

登録集原文"CBLXMLRC.cbl"の定義名（78 レベル）	アクセス情報フラグ値	アクセス情報フラグの意味
CBLXML-FLAG-OK	B'00000～0'	要素または属性を出力する。
CBLXML-FLAG-MISSING	B'10000～0'	要素を出力しない。※
CBLXML-FLAG-EMPTY	B'01000～0'	空要素を出力する。

注※

対応づけた要素が省略できる場合，または Array 要素の直下に Item 要素を対応づけた場合は，要素を出力しません。ただし，次の場合は要素，属性，または空要素を出力します。

- 対応づけた要素が省略できない要素である場合，または対応づけた属性に#REQUIRED を指定した場合
  （CBLXML-FLAG-OK が仮定されて，要素を出力します）
- CBLXML-FLAG-MISSING と CBLXML-FLAG-EMPTY が同時に指定されている場合
  （CBLXML-FLAG-EMPTY が仮定されて，空要素を出力します）

(3) nameOfFlagVar 属性の指定例

入出力データ情報定義機能を使用した場合の nameOfFlagVar 属性の指定例を次に示します。

指定例

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (group01)>
  <!ELEMENT group01 (item01)>
  <!ELEMENT item01  (#PCDATA)>
]>
<root/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE" accessInfo="yes">
  <BaseElement elemName="group01">
    <Group elemName="group01"
```

```
            nameOfFlagVar="G01-FLAG">
      <Item elemName="item01"
            nameOfFlagVar="I01-FLAG"
            type="alphanumeric" size="10"/>
    </Group>
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
 01 group01-BASE.
  02 group01.
   03 item01 PIC X(10).
* Access Information.
  02 G01-FLAG PIC 1(32) USAGE BIT.
  02 group01-GROUP.
   03 I01-FLAG PIC 1(32) USAGE BIT.
   03 item01-LEN PIC 9(9) USAGE COMP.
```

（COBOL 原始プログラムでのアクセス情報フラグの判定例）

```
         :
01 CHECK-FLAG PIC 1(32) BIT.
         :
    COMPUTE CHECK-FLAG = I01-FLAG AND CBLXML-FLAG-MISSING.
    IF CHECK-FLAG = CBLXML-FLAG-MISSING  THEN
       DISPLAY 'CBLXML-FLAG-MISSING'
    END-IF.
    COMPUTE CHECK-FLAG = I01-FLAG AND CBLXML-FLAG-EMPTY.
    IF CHECK-FLAG = CBLXML-FLAG-EMPTY  THEN
       DISPLAY 'CBLXML-FLAG-EMPTY'
    END-IF.
    COMPUTE CHECK-FLAG = I01-FLAG AND CBLXML-FLAG-INVAL-CHAR.
    IF CHECK-FLAG = CBLXML-FLAG-INVAL-CHAR  THEN
       DISPLAY 'CBLXML-FLAG-INVAL-CHAR'
    END-IF.
    COMPUTE CHECK-FLAG = I01-FLAG AND CBLXML-FLAG-OVERFLOW.
    IF CHECK-FLAG = CBLXML-FLAG-OVERFLOW  THEN
       DISPLAY 'CBLXML-FLAG-OVERFLOW'
    END-IF.
         :
```

3.2.2　データ長

データ長には，XML ドキュメントの入力時に要素に対応したデータ長が格納されます。また，出力時にデータ長を設定すると，出力する要素の出力長を設定できます。データ長は 9 けたの 2 進形式の数字項目として，cblxml コマンドが生成する XML アクセス用データ定義に生成されます。

データ長の COBOL データ項目名称

Item 要素または AttrItem 要素の nameOfLengthVar 属性に指定した名称が，データ長の COBOL データ項目の名称になります。

nameOfLengthVar 属性の指定がない場合は cobName 属性（省略時は elemName 属性）の名称に"-LEN"を追加した名称が，データ長の COBOL データ項目名称になります。

(1) 入力時のデータ長

XML ドキュメント入力時に，データ長に格納される値を表 3-5 に示します。

表 3-5 入力時のデータ長

XML ドキュメント	データ長に入力される値
要素（Item 要素）	XML ドキュメント内にある要素の内容のバイト数
属性（AttrItem 要素）	XML ドキュメント内にある属性値のバイト数

注

実体参照がある場合の入力時のデータ長は，実体参照に対応する文字列へ変換された値の長さになります。

(2) 出力時のデータ長

XML ドキュメント出力時に，データ長に設定する値を表 3-6 に示します。

表 3-6 出力時のデータ長

XML ドキュメント	データ長に設定する値
要素（Item 要素）	要素の内容に出力するデータのバイト数
属性（AttrItem 要素）	属性値に出力するデータのバイト数

注

- XML ドキュメント出力時に 0 または size 属性で指定した値よりも大きいバイト数をデータ長に指定した場合，size 属性で指定したけた数が最大長として出力されます。このとき，type 属性の指定に関係なく，左端からの長さが出力されます。
- 日本語項目の場合，出力される文字の長さはデータ長に格納される値のバイト数の半分になります。端数は切り捨てになります。
- 数字項目の場合，データ長には 0 が仮定されます。データ長に 0 が指定されると，size 属性で指定したけた数で出力されます。

(3) nameOfLengthVar 属性の指定例

入出力データ情報定義機能を使用した場合の nameOfLengthVar 属性の指定例を次に示します。

指定例

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (item01)>
  <!ELEMENT item01  (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE" accessInfo="yes">
  <BaseElement elemName="item01">
    <Item elemName="item01" nameOfLengthVar="I01L"
          type="alphanumeric" size="10"/>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
  01 item01-BASE.
   02 item01 PIC X(10).
* Access Information
   02 item01-FLAG PIC 1(32) USAGE BIT.
   02 I01L PIC 9(9) USAGE COMP.
```

3.2.3　繰り返し全要素数

繰り返し全要素数には，Array 要素に対応づけた繰り返し要素が，実際に入力 XML ドキュメントに出現した回数が格納されます。繰り返し全要素数は 9 けたの 2 進形式の数字項目として，cblxml コマンドが生成する XML アクセス用データ定義に生成されます。

繰り返し全要素数の COBOL データ項目の名称

Array 要素の nameOfTotalVar 属性に指定した名称が，繰り返し全要素数の COBOL データ項目の名称になります。

nameOfTotalVar 属性の指定を省略した場合は cobName 属性（省略時は elemName 属性）の名称に"-TOTAL"を追加した名称が，繰り返し全要素数の COBOL データ項目の名称になります。

nameOfTotalVar 属性の例については，「3.2.4(1)　nameOfTotalVar 属性と nameOfCountVar 属性の指定例」を参照してください。

3.2.4　繰り返し入出力数

繰り返し入出力数には，Array 要素に対応づけた繰り返し要素を実際に入力した繰り返し回数が格納されます。繰り返し入出力数は 9 けたの 2 進形式の数字項目として，cblxml コマンドが生成する XML アクセス用データ定義に生成されます。

繰り返し入出力数の COBOL データ項目名称

Array 要素の nameOfCountVar 属性に指定した名称が，繰り返し入出力数の COBOL データ項目名称になります。

nameOfCountVar 属性の指定がない場合は，cobName 属性（省略時は elemName 属性）の名称に"-COUNT"を追加した名称が，繰り返し入出力数の COBOL データ項目名称になります。

(1)　nameOfTotalVar 属性と nameOfCountVar 属性の指定例

入出力データ情報定義機能を使用した場合の nameOfTotalVar 属性と nameOfCountVar 属性の指定例を次に示します。

指定例

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (group01*)>
  <!ELEMENT group01 (item01, item02)>
  <!ELEMENT item01  (#PCDATA)>
  <!ELEMENT item02  (#PCDATA)>
]>
<root/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="root" accessInfo="yes">
    <Group cobName="root">
      <Array max="10" nameOfCountVar="ARY-CNT"
             nameOfTotalVar="ARY-TOL">
        <Group cobName="group1">
          <Item elemName="item01" type="alphanumeric"
                size="10" />
          <Item elemName="item02" type="alphanumeric"
                size="10" />
        </Group>
      </Array>
    </Group>
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
 01 root-BASE.
  02 root.
```

```
   03 group01 OCCURS 10.
    04 item01 PIC X(10).
    04 item02 PIC X(10).
* Access Information
  02 root-GROUP.
   03 ARY-TOL PIC 9(9) USAGE COMP.
   03 ARY-CNT PIC 9(9) USAGE COMP.
   03 group01-GROUP OCCURS 10.
    04 item01-FLG PIC 1(32) USAGE BIT.
    04 item01-LEN PIC 9(9) USAGE COMP.
    04 item02-FLG PIC 1(32) USAGE BIT.
    04 item02-LEN PIC 9(9) USAGE COMP.
```

3.3　入出力データ情報定義と DDL の対応づけ

入出力データ情報定義機能を使用する場合の，DDL の各要素の規則と指定例について説明します。

3.3.1　BaseElement 要素

入出力データ情報定義機能を使用した場合，XML アクセス用データ定義の名称として BaseElement 要素に対応した 01 レベルの集団項目が生成されます。

集団項目名は，BaseElement 要素の cobName 属性に指定した名称（省略時は elemName 属性）に"-BASE"を追加した名称になります。

(1)　規則

"-BASE"を追加した名称は，一意である必要があります。一意でない名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

(2)　指定例

BaseElement 要素に対応した集団項目の下位レベルには，次に示す入出力データ項目と入出力データ項目に対応する入出力データ情報項目が生成されます。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (item01, item02)>
  <!ELEMENT item01  (#PCDATA)>
  <!ELEMENT item02  (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="root" accessInfo="yes">
    <Group elemName="root">
      <Item elemName="item01" type="alphanumeric"
            size="10" />
      <Item elemName="item02" type="alphanumeric"
            size="10" />
    </Group>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
 01 root-BASE.
  02 root.
```

```
   03 item01 PIC X(10).
   03 item02 PIC X(10).
* Access Information
  02 root-FLG PIC 1(32) USAGE BIT.
  02 root-GROUP.
   03 item01-FLG PIC 1(32) USAGE BIT.
   03 item01-LEN PIC 9(9) USAGE COMP.
   03 item02-FLG PIC 1(32) USAGE BIT.
   03 item02-LEN PIC 9(9) USAGE COMP.
```

3.3.2 Group 要素

入出力データ情報定義機能を使用した場合，入出力データ情報項目の名称として Group 要素に対応した集団項目が生成されます。

集団項目名は，Group 要素の cobName 属性に指定した名称（省略時は elemName 属性）に"-GROUP"を追加した名称になります。

Group 要素に elemName 属性を使って要素を対応づけている場合は，Group 要素に対してアクセス情報フラグが生成されます。Group 要素に cobName 属性だけを指定している場合は，アクセス情報フラグが生成されません。

(1) 規則

"-GROUP"を追加した名称は，一意である必要があります。一意でない名称を指定した場合，COBOL 原始プログラムの生成時にエラーとなります。

(2) 指定例

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (group01, group02)>
  <!ELEMENT group01 (item01)>
  <!ELEMENT item01  (#PCDATA)>
  <!ELEMENT group02 (item02)>
  <!ELEMENT item02  (#PCDATA)>
]>
<root/>
```

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="group01" accessInfo="yes">
    <Group elemName="group01">
      <Item elemName="item01" type="alphanumeric"
            size="10" />
    </Group>
```

```
  </BaseElement>
  <BaseElement elemName =”group02” accessInfo=”yes”>
    <Group cobName=”group02”>
      <Item elemName=”item02” type=”alphanumeric”
            size=”10” />
    </Group>
  </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
 01 group01-BASE.
  02 group01.
   03 item01 PIC X(10).
* Access Information
  02 group01-FLG PIC 1(32) USAGE BIT.          …1.
  02 group01-GROUP.
   03 item01-FLG PIC 1(32) USAGE BIT.
   03 item01-LEN PIC 9(9) USAGE COMP.
 01 group02-BASE.
  02 group02.
   03 item02 PIC X(10).
* Access Information
  02 group02-GROUP.
   03 item02-FLG PIC 1(32) USAGE BIT.
   03 item02-LEN PIC 9(9) USAGE COMP.
```

（説明）

1. Group 要素のアクセス情報フラグが生成されるのは，elemName 属性を指定した場合だけです。

3.3.3 Item 要素

入出力データ情報定義機能を使用した場合，Item 要素に対応したアクセス情報フラグとデータ長が生成されます。

(1) 指定例

（DTD の例）

```
<?xml version=”1.0” encoding=”Shift_JIS”?>

<!DOCTYPE root [
  <!ELEMENT root (item01)>
  <!ELEMENT item01  (#PCDATA)>
]>
<root/>
```

（DDF の例）

```
<?xml version=”1.0” encoding=”Shift_JIS”?>
<Interface interfaceName=”EXAMPLE”>
  <BaseElement elemName=”root” accessInfo=”yes”>
```

```
    <Item elemName="item01" type="alphanumeric"
          size="10" />
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
 01 root-BASE.
  02 item01 PIC X(10).
* Access Information
  02 item01-FLG PIC 1(32) USAGE BIT.
  02 item01-LEN PIC 9(9) USAGE COMP.
```

3.3.4 Array 要素

入出力データ情報定義機能を使用した場合，繰り返し項目に対応した繰り返し全要素数と繰り返し入出力数が生成されます。

繰り返し要素が単一（Item 要素）の場合，Item 要素に対応するアクセス情報フラグとデータ長に OCCURS 句が付けられます。繰り返し要素が集団（Group 要素）の場合は，Group 要素に対応する集団項目に OCCURS 句が付けられます。

(1) 繰り返し要素が単一の場合

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (item01*)>
  <!ELEMENT item01  (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="root" accessInfo="yes">
    <Group cobName="root">
      <Array max="10">
        <Item elemName="item01" type="alphanumeric"
              size="10" />
      </Array>
    </Group>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
 01 root-BASE.
  02 root.
   03 item01 PIC X(10) OCCURS 10.
```

```
* Access Information
  02 root-GROUP.
   03 item01-TOTAL PIC 9(9) USAGE COMP.
   03 item01-COUNT PIC 9(9) USAGE COMP.
   03 item01-FLG PIC 1(32) USAGE BIT OCCURS 10.
   03 item01-LEN PIC 9(9) USAGE COMP OCCURS 10.
```

(2)　繰り返し要素が集団の場合

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (group01*)>
  <!ELEMENT group01 (item01, item02)>
  <!ELEMENT item01  (#PCDATA)>
  <!ELEMENT item02  (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="root" accessInfo="yes">
    <Group cobName="root">
      <Array max="10">
        <Group cobName="group1">
          <Item elemName="item01" type="alphanumeric"
                size="10" />
          <Item elemName="item02" type="alphanumeric"
                size="10" />
        </Group>
      </Array>
    </Group>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
 01 root-BASE.
  02 root.
   03 group01 OCCURS 10.
    04 item01 PIC X(10).
    04 item02 PIC X(10).
* Access Information
  02 root-GROUP.
   03 group01-TOTAL PIC 9(9) USAGE COMP.
   03 group01-COUNT PIC 9(9) USAGE COMP.
   03 group01-GROUP OCCURS 10.
    04 item01-FLG PIC 1(32) USAGE BIT.
    04 item01-LEN PIC 9(9) USAGE COMP.
    04 item02-FLG PIC 1(32) USAGE BIT.
    04 item02-LEN PIC 9(9) USAGE COMP.
```

3.4 XML アクセスルーチン使用時の注意事項

入出力データ情報定義機能を使用した場合に，XML アクセスルーチンを使用するときに注意が必要な点について説明します。

01 レベルの集団項目名を指定する場合の注意事項

CBLXML-RD-*Interface-BaseElement* アクセスルーチン，CBLXML-WR-*Interface-BaseElement* アクセスルーチンに登録集原文での 01 レベルの集団項目名を指定する場合，BaseElement 要素の名称に"-BASE"を追加した名称または nameOfBaseVar 属性で指定した名称を指定してください。

致命的なエラーステータスが返る場合の注意事項

XML ドキュメントの入力時に，CBLXML-RD-*Interface-BaseElement* アクセスルーチンが致命的なエラーステータスを返した場合は，入出力データ情報項目の値は保証しません。

3.5 XML ドキュメント読み込み時に設定される入出力データ情報項目

入出力データ情報定義機能を使用すると，XML ドキュメント入力時に入力した要素および属性の入力状態が入出力データ情報項目に設定されます。

DDL の各要素で設定される入出力データ情報項目の意味を表 3-7 に示します。

表 3-7 XML ドキュメント読み込み時の入出力データ情報項目の値

入出力データ情報項目	Array 要素	Group 要素	Item 要素	AttrItem 要素
アクセス情報フラグ	－	要素の入力状態によって設定される値が異なる。詳細については，表 3-8 を参照すること。	要素の入力状態によって設定される値が異なる。詳細については，表 3-8 を参照すること。	要素の入力状態によって設定される値が異なる。詳細については，表 3-8 を参照すること。
データ長	－	－	要素の値長が設定される（XML ドキュメント内の要素の値長）。	属性値の長さが設定される（XML ドキュメント内の属性の値長）。
繰り返し全要素数※	XML ドキュメントの繰り返し要素の数が設定される。	－	－	－
繰り返し入出力数※	入力した繰り返し要素の値の数が設定される。	－	－	－

（凡例）

－：値が設定されません。

注※

XML ドキュメントの繰り返し項目のすべてを入力していない場合，繰り返し全要素数が繰り返し入出力数より大きくなります。

次に，アクセス情報フラグについて説明します。設定されたアクセス情報フラグの種類から DDL の各要素の入力状態が判断できます。アクセス情報フラグが示す DDL の各要素の入力状態を表 3-8 に示します。

表 3-8 XML ドキュメント読み込み時のアクセス情報フラグの値

アクセス情報フラグ	Array 要素	Group 要素	Item 要素	AttrItem 要素
B'10000～0'（CBLXML-FLAG-MISSING）	－	要素と対応づいていて，Group 要素がない場合に設定される。	要素がない場合または上位の要素が空要素の場合に設定される。	属性がない場合または上位の要素が空要素の場合に設定される。
B'01000～0'（CBLXML-FLAG-EMPTY）	－	－	要素が空の場合に設定される。※1 例：<X></X>，または<X/>	属性値が空の場合に設定される。※2 例：name=""

アクセス情報フラグ	Array 要素	Group 要素	Item 要素	AttrItem 要素
B'00100～0' (CBLXML-FLAG-INVAL-CHAR)	－	－	要素の値に不当な文字が含まれる場合に設定される。	属性値に不当な文字が含まれる場合に設定される。
B'00010～0' (CBLXML-FLAG-OVERFLOW)	－	－	要素の値長が COBOL データ項目より長い場合に設定される。	属性値の長さが COBOL データ項目より長い場合に設定される。

(凡例)

－：値が設定されません。

注※1

開始タグと終了タグ間の内容が空白文字（スペース，タブおよび改行）だけの場合でも空要素とみなされません。空要素とみなされるのは空タグの場合，および開始タグと終了タグ間の内容がない（<Item></Item>）場合です。

注※2

属性値の内容が空白文字（スペース，タブおよび改行）だけの場合でも空要素とみなされません。空要素とみなされるのは，属性値がない（name=""）場合です。

3.6　XML ドキュメント書き込み時に設定する入出力データ情報項目

入出力データ情報項目の値を設定することで，XML ドキュメント出力時に要素の出力状態を設定できます。

入出力データ情報項目が示す DDL の各要素の出力状態を表 3-9 に示します。

表 3-9　XML ドキュメント書き込み時の入出力データ情報項目の値

入出力データ情報項目	Array 要素	Group 要素	Item 要素	AttrItem 要素
アクセス情報フラグ	×	要素の出力状態や出力の有無を設定する。詳細については，表 3-10 を参照すること。	要素の出力状態や出力の有無を設定する。詳細については，表 3-10 を参照すること。	要素の出力状態や出力の有無を設定する。詳細については，表 3-10 を参照すること。
データ長	×	×	出力する要素の値長を指定する。※	出力する属性の値長を指定する。※
繰り返し全要素数	無視する。	×	×	×
繰り返し入出力数	出力する繰り返し要素数を指定する。	×	×	×

（凡例）

×：値を設定できません。

注※

0 または size 属性で指定した値よりも大きい値を指定した場合は，size 属性での最大値が出力されます。

type="numeric"を指定した場合は，データ長の指定は無視されます。

次に，アクセス情報フラグについて説明します。アクセス情報フラグを設定することで，DDL の各要素の出力状態をどのように設定できるかを表 3-10 に示します。

表 3-10　XML ドキュメント書き込み時のアクセス情報フラグの値

アクセス情報フラグ	Array 要素	Group 要素	Item 要素	AttrItem 要素
B'00000～0'（CBLXML-FLAG-OK）	×	Group 要素を出力する。	Item 要素を出力する。	属性を出力する。
B'10000～0'（CBLXML-FLAG-MISSING）	×	省略できる Group 要素の場合，Group 要素を出力しない。	省略できる要素の場合，要素を出力しない。※	属性に#REQUIRED を指定していない場合，属性を出力しない。
B'01000～0'（CBLXML-FLAG-EMPTY）	×	無視する。	空要素を出力する。 例：<xyz/>	属性値が空の状態で出力する。 例：name=""

（凡例）

×：値を設定できません。

注※

countVar 属性に"no"を指定した Array 要素の直下に繰り返し要素を Item 要素で対応づけた場合は，省略できない要素であっても出力されません。

4 XMLアクセスルーチンとXMLアクセス用データ定義の生成

cblxmlコマンドは，XMLドキュメントのDTDとユーザが作成したDDFを基に，COBOLプログラムからXMLドキュメントにアクセスするために必要なXMLアクセスルーチン，およびXMLアクセス用データ定義を生成します。

XMLアクセスルーチンは，COBOLプログラムからXMLドキュメントにアクセスするときに呼び出す副プログラムです。

XMLアクセス用データ定義は，XMLドキュメントの各要素に対応づけられたCOBOLのデータ項目の定義です。

また，XMLアクセスルーチンは，ステータスコードとエラー定義名称を対応づけるためにXMLアクセス用ステータス定義と呼ばれる登録集原文を使用します。

この章では，cblxmlコマンドの使用方法と，XMLアクセスルーチン，XMLアクセス用データ定義，およびXMLアクセス用ステータス定義の概要について説明します。

4.1　cblxml コマンド

cblxml コマンドは，XML ドキュメントの文書型定義（DTD）とユーザが作成したデータ定義ファイル（DDF）を基に，COBOL プログラムから XML ドキュメントにアクセスするために必要な次のソースファイルを生成します。

- XML アクセスルーチン

  COBOL プログラムから XML ドキュメントにアクセスするときに呼び出す副プログラムです。詳細については，「4.2　生成される XML アクセスルーチン」を参照してください。

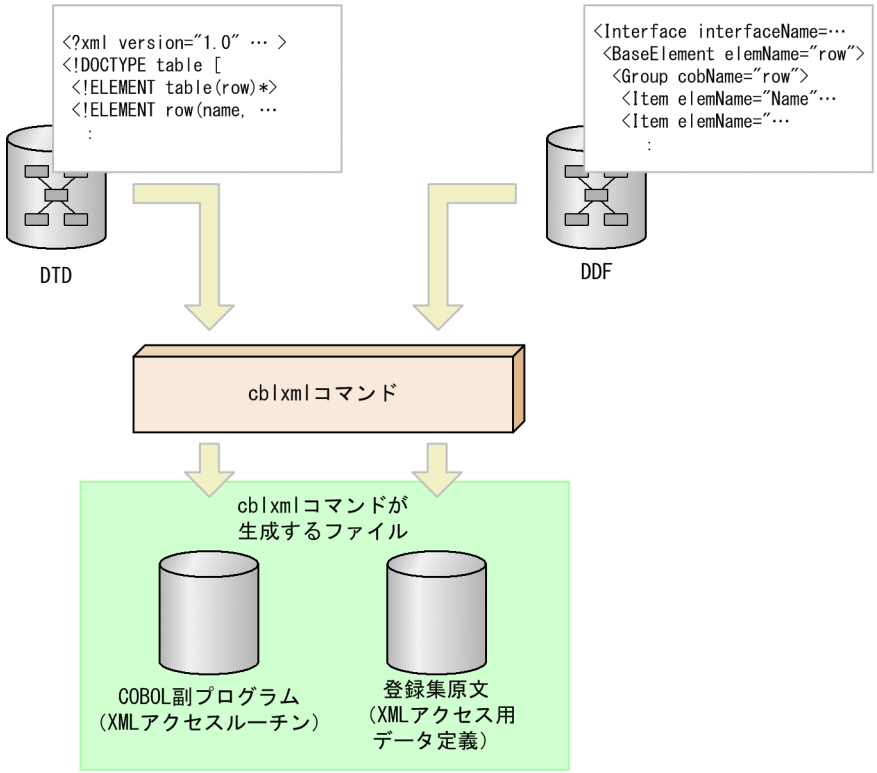
- XML アクセス用データ定義

  XML ドキュメントの各要素に対応づけられた COBOL のデータ項目の定義です。詳細については，「4.3　生成される XML アクセス用データ定義」を参照してください。

なお，DTD については「2.1　文書型定義（DTD）の概要」を，DDF については「2.2　データ定義ファイル（DDF）の作成」を参照してください。

cblxml コマンドの概要を図 4-1 に示します。

図 4-1　cblxml コマンドの概要



4.1.1　cblxml コマンドの使用方法

(1)　cblxml コマンドの実行に必要な環境変数の設定

cblxml コマンドを実行する場合は，次に示す環境変数を設定しておく必要があります。

(UNIX の場合)

- システム環境変数 PATH
  cblxml コマンドの格納パスを設定します。次の値を指定してください。
  COBOL2002のインストールディレクトリ/bin
- システム環境変数 LIBPATH（AIX の場合）
  共用ライブラリの格納パスを設定します。次の値を指定してください。
  COBOL2002のインストールディレクトリ/lib
- システム環境変数 LD_LIBRARY_PATH（Linux の場合）
  共用ライブラリの格納パスを設定します。次の値を指定してください。
  COBOL2002のインストールディレクトリ/lib:COBOL2002のインストールディレクトリ/lib/cblxml
- システム環境変数 NLSPATH
  XML パーサが出力するメッセージのパスを設定します。次の値を指定してください。
  COBOL2002のインストールディレクトリ/lib/cblxml/cat/%L/%N

(例)

sh（B シェル）の場合

```
NLSPATH=/opt/HILNGcbl2k/lib/cblxml/cat/%L/%N
export NLSPATH
```

- システム環境変数 LANG
  メッセージの言語種別を設定します。次の値を指定してください。
  AIX の場合
  Ja_JP
  ja_JP
  Linux の場合
  ja_JP.UTF-8

文字コードについては，「付録 G.2　文字コード」を参照してください。

注

COBOL2002 のインストールディレクトリは，OS によって異なります。
AIX(32)，Linux(x86)の場合
/opt/HILNGcbl2k
AIX(64)，Linux(x64)の場合

/opt/HILNGcbl2k64

（Windows の場合）

- システム環境変数 PATH

  cblxml コマンドの格納パスを設定します。次の値を指定してください。

  COBOL2002のインストールフォルダ¥bin

- システム環境変数 LIB

  共用ライブラリの格納パスを設定します。次の値を指定してください。

  COBOL2002のインストールフォルダ¥lib

（例）

```
set LIB=COBOL2002のインストールフォルダ¥lib
```

(2) cblxml コマンドの実行

cblxml コマンドの実行方法について説明します。

形式

```
cblxml DDFファイル名 -dtd DTDファイル名
       -o 生成するCOBOL副プログラムのファイル名
        〔-catalog カタログファイル名〕
        〔-nopeconv〕 〔-chkovflow〕 〔-chkchar〕
        〔-gen 生成アクセスルーチンキーワード〕
        〔-outencoding エンコーディングキーワード〕
        〔-unisrc〕 〔-bigendianbin〕 〔-bigendianfloat〕
```

DDF ファイル名

DTD に対応して作成した DDF ファイル名を指定します。DDF ファイル名の拡張子は「.cxd」でなければなりません。

DDF ファイルでは，使用する文字エンコーディングを XML 宣言で指定します。使用できる文字エンコーディングについては，「付録 G.1　XML ドキュメントの文字エンコーディング」を参照してください。使用できない文字エンコーディングを指定した場合，動作は保証しません。

DDF ファイルに文字エンコーディングの指定がない場合，「UTF-8」を仮定します。

-dtd DTD ファイル名

COBOL プログラムのアクセス対象とする XML ドキュメントの DTD ファイル名を指定します。

DTD ファイル名の拡張子は「.xml」または「.dtd」※でなければなりません。

DTD ファイルでは，使用する文字エンコーディングを XML 宣言で指定します。使用できる文字エンコーディングについては，「付録 G.1　XML ドキュメントの文字エンコーディング」を参照してください。使用できない文字エンコーディングを指定した場合，動作は保証しません。

DTD ファイルに文字エンコーディングの指定がない場合，「UTF-8」を仮定します。

注※

「.dtd」は Windows の場合だけで有効です。

-o 生成する COBOL 副プログラムのファイル名

cblxml コマンドが出力する XML アクセスルーチンのファイル名を指定します。指定するファイル名の拡張子は，「.cbl」でなければなりません。

なお，cblxml コマンドが出力する XML アクセス用データ定義のファイル名は，XML アクセスルーチンのファイル名の末尾に「-COPY」を付けた名称となります。

出力する COBOL 原始プログラムは固定形式正書法で出力されます。

-catalog カタログファイル名

公開識別子を使用するためのカタログファイル名を指定します。拡張子は「.cxc」でなければなりません。

詳細については，「9.3.3　公開識別子が指定された XML ドキュメント」を参照してください。

-nopeconv

verbatim 属性に"yes"を指定していない場合，XML ドキュメントの出力時に定義済み実体を変換しません。

詳細については，「付録 D.1　定義済み実体参照」を参照してください。

-chkovflow

XML ドキュメントの入力時にオーバフローが発生した場合，ステータス 10 を返します。

詳細については，「9.1　入力時のオーバフローをステータスで返す機能」を参照してください。

-chkchar

XML ドキュメントの入出力時に不当な文字のチェック範囲を拡張します。

詳細については，「9.2　入出力時に不当な文字をチェックする機能」を参照してください。

-gen 生成アクセスルーチンキーワード

生成アクセスルーチンキーワードに指定した XML アクセスルーチンだけを生成します。

詳細については，「4.1.2　-gen オプション」を参照してください。

-outencoding エンコーディングキーワード

XML アクセスルーチンで出力する XML ドキュメントの文字エンコーディングを指定します。指定できるエンコーディングキーワードは，sjis，euc，utf8，utf16，utf16be または utf16le です。大文字は指定できません。エンコーディングキーワードと出力する XML ドキュメントの文字エンコーディングの対応を次に示します。詳細については，「付録 G.1 表 G-3　出力する XML ドキュメントの文字エンコーディングの設定方法」を参照してください。

エンコーディングキーワード	出力する XML ドキュメントの文字エンコーディング（encoding 属性の値）
sjis	Shift_JIS
win31j	Windows-31J
euc	EUC-JP
utf8	UTF-8
utf16	Windows，Linux の場合：UTF-16，リトルエンディアン AIX の場合：UTF-16，ビッグエンディアン

エンコーディングキーワード	出力する XML ドキュメントの文字エンコーディング（encoding 属性の値）
utf16be	UTF-16，ビッグエンディアン
utf16le	UTF-16，リトルエンディアン

-unisrc

Unicode 機能に対応した XML アクセスルーチンを生成します。詳細については，「付録 H Unicode 機能」を参照してください。

-bigendianbin（Windows，Linux の場合）

XML データに対応する 2 進形式の数字項目をビッグエンディアン形式として扱う場合に指定します。詳細については，「4.1.4　-bigendianbin オプション（Windows，Linux の場合）」を参照してください。

-bigendianfloat（Windows，Linux の場合）

XML データに対応する浮動小数点形式の数字項目をビッグエンディアン形式として扱う場合に指定します。詳細については，「4.1.5　-bigendianfloat オプション（Windows，Linux の場合）」を参照してください。

cblxml コマンドの戻り値

戻り値	内容
0	正常終了した。
1	正常終了した。 ただし，警告メッセージが出力されている。
2	エラー終了した。

指定例

```
cblxml data.cxd -dtd data.xml -o subprog.cbl
```

DTD「data.xml」と DDF「data.cxd」の定義を基に，XML アクセスルーチンの副プログラムファイル「subprog.cbl」と XML アクセス用データ定義の登録集原文ファイル「subprog-COPY.cbl」が生成されます。

注意事項

- cblxml コマンドによって生成された COBOL 副プログラム（XML アクセスルーチン），および登録集原文（XML アクセス用データ定義）の内容を編集しないでください。これらのファイルを編集した場合，実行時の動作は保証しません。
- cblxml コマンドでは，生成する COBOL 副プログラム（XML アクセスルーチン），および登録集原文（XML アクセス用データ定義）の領域の大きさについて，COBOL2002 コンパイラの制限を超えたかどうかをチェックしません。コンパイラの制限値については，マニュアル「COBOL2002 使用の手引　手引編」または「COBOL2002　ユーザーズガイド」を参照してください。

4.1.2 -gen オプション

-gen オプションは，cblxml コマンドが生成する XML アクセスルーチンを指定するオプションです。

形式

```
-gen△生成アクセスルーチンキーワード
```

注

△は半角空白文字，またはタブ文字です。

生成アクセスルーチンキーワードは，表 4-1 に従って生成したい XML アクセスルーチンに対応する生成アクセスルーチンキーワードをコンマで区切って指定します。CBLXML-OB-*Interface*，CBLXML-RD-*Interface-BaseElement*，および CBLXML-CN-*Interface* アクセスルーチンを生成する場合の例を次に示します。

（例）

```
-gen△OB,RD,CN
```

注

△は半角空白文字，またはタブ文字です。

-gen オプションに指定する生成アクセスルーチンキーワードは，表 4-1 に示す 3 種類の XML ドキュメントの操作（XML ドキュメントを開く（OP，OB），XML ドキュメントを入力，出力する（RD，WR），XML ドキュメントを閉じる（CL,CN））から，生成アクセスルーチンキーワードをそれぞれ一つ以上指定しなければなりません。

表 4-1　生成アクセスルーチンキーワード

生成アクセスルーチンキーワード	生成する XML アクセスルーチン	XML ドキュメントの操作
OP	CBLXML-OP-*Interface*	XML ドキュメントを開く
OB	CBLXML-OB-*Interface*	
RD	CBLXML-RD-*Interface-BaseElement*	XML ドキュメントを入力，出力する
WR	CBLXML-WR-*Interface-BaseElement*	
CL	CBLXML-CL-*Interface*	XML ドキュメントを閉じる
CN	CBLXML-CN-*Interface*	

注意事項

- -gen オプションの生成アクセスルーチンキーワードに次の不当な指定がされた場合，XML アクセスルーチンの生成時にエラーとなります。
  - 生成アクセスルーチンキーワード以外の文字列を指定した。

- 生成アクセスルーチンキーワードの区切り文字にコンマ以外の文字や空白を指定した。
- 同じ生成アクセスルーチンキーワードを複数回指定した。
- 生成アクセスルーチンキーワードを，XML ドキュメントを開く，入出力，閉じるの構成で指定していない。
- 生成アクセスルーチンキーワードの指定がない。

- 生成アクセスルーチンキーワード"WR"を指定しない場合，警告メッセージは表示されません。
- 生成アクセスルーチンキーワードは順不同です。

4.1.3 -outencoding オプション

-outencoding オプションは，XML アクセスルーチンで出力する XML ドキュメントの文字エンコーディングを指定するオプションです。

XML データ定義ファイルの「ファイルの設定」で表示される XML データ定義ファイルタグで表示する cblxml コマンドの-outencoding オプションには，エンコーディングキーワードを一つだけ指定できます。

4.1.4 -bigendianbin オプション（Windows，Linux の場合）

type 属性値が"binary"の Item 要素に対応する COBOL データ項目をビッグエンディアン形式として扱うオプションです。生成した XML アクセス用データ定義を使用したプログラムで，コンパイル時に-BigEndian,Bin コンパイラオプションを指定して，2 進形式の数字項目をビッグエンディアン形式で扱いたい場合に指定します。

機能

1. CBLXML-RD-*Interface-BaseElement* アクセスルーチンを利用して XML ドキュメントから入力する場合，対応する COBOL データ項目へビッグエンディアン形式で格納する。
2. CBLXML-WR-*Interface-BaseElement* アクセスルーチンを利用して XML ドキュメントへ出力する場合，対応する COBOL データ項目をビッグエンディアン形式として扱う。

注意事項

- このオプションを指定して生成した XML アクセスルーチンおよび XML アクセス用データ定義を使用するプログラムをコンパイルするとき，-BigEndian,Bin コンパイラオプションと-Comp5 コンパイラオプションを指定する必要があります。指定しない場合の動作は保証しません。
- -BigEndian,Bin コンパイラオプションを指定してコンパイルした COBOL プログラムでは，XML サービスルーチンの引数で指定する数字項目の用途が COMP の場合，ビッグエンディアン形式になるため，正常に動作しません。この場合は，引数となる数字項目（2 進形式）を COMP-5 で定義してください。

4.1.5　-bigendianfloat オプション（Windows，Linux の場合）

type 属性値が"float"または"double"の Item 要素に対応する COBOL データ項目をビッグエンディアン形式として扱うオプションです。生成した XML アクセス用データ定義を使用したプログラムで，コンパイル時に-BigEndian,Float コンパイラオプションを指定して，浮動小数点形式の数字項目をビッグエンディアン形式で扱いたい場合に指定します。

機能

1. CBLXML-RD-*Interface-BaseElement* アクセスルーチンを利用して XML ドキュメントから入力する場合，対応する COBOL データ項目へビッグエンディアン形式で格納する。
2. CBLXML-WR-*Interface-BaseElement* アクセスルーチンを利用して XML ドキュメントへ出力する場合，対応する COBOL データ項目をビッグエンディアン形式として扱う。

注意事項

このオプションを指定して生成した XML アクセスルーチンおよび XML アクセス用データ定義を使用するプログラムをコンパイルするとき，-BigEndian,Float コンパイラオプションを指定する必要があります。指定しない場合の動作は保証しません。

4.1.6　cblxml コマンドのメッセージ

cblxml コマンドが出力するメッセージについて説明します。

(1)　メッセージの形式

メッセージの形式を次に示します。

```
KCCBnnnnX-z メッセージテキスト
```

KCCB：プリフィクス

nnnn：メッセージ番号

X：XML 連携機能のメッセージであることを示す

z：メッセージレベル

E：エラー

W：警告

また，このマニュアルでは，メッセージを次の形式で記載します。

KCCBnnnnX-z

メッセージテキスト

要因：メッセージの説明を示します。

(S) システムの処置を示します。

(P) プログラム作成者の処置を示します。

メッセージ中の可変の埋め字部分は*** n ***（n は数字）で示します。

(2) メッセージの一覧

メッセージの一覧を次に示します。

KCCB0011X-E

A version of message file is disagreed.

要因：異なるバージョンの cblxml コマンドのメッセージファイルが存在する。

(S) COBOL 原始プログラムの生成を中止する。

(P) 異なるバージョンのプロダクトを削除して再実行する。

KCCB0012X-E

The file other than message file is opened.

要因：異なるシステムの cblxml コマンドのメッセージファイルが存在する。

(S) COBOL 原始プログラムの生成を中止する。

(P) 異なるシステムのプロダクトを削除して再実行する。

KCCB0013X-E

Logical error occurred. (*** 1 ***)

要因：cblxml コマンドのエラーメッセージ出力処理で論理エラーが発生した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 当社保守員に連絡する。

KCCB0014X-E

The message number "*** 1 ***" is not in the message file.

要因：cblxml コマンドで内部的に論理エラーが発生した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 当社保守員に連絡する。

KCCB0015X-E

Unable to use the message file.

要因：cblxml コマンドのメッセージファイルが存在しない，または壊れている。

(S) COBOL 原始プログラムの生成を中止する。

(P) プロダクトを再インストールする。

KCCB0205X-E

稼働するメモリが足りません。

要因：cblxml コマンドが稼働するメモリが足りない。

(S) COBOL 原始プログラムの生成を中止する。

(P) 削除できる資源を削除して再実行する。

KCCB0206X-W

不当なファイルまたはパラメタ"*** 1 ***"の指定があります。 "*** 1 ***"を無視します。

要因：不当なファイル名またはパラメタを指定した。

(S) 不当なファイル名またはパラメタを無視して実行を続ける。

(P) 不当なファイル名またはパラメタを削除する。

KCCB0207X-E

"*** 1 ***"オプションに続くファイル名の指定がありません。

要因：オプションに必要なファイル名がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) オプションに必要なファイル名を指定して実行を続ける。

KCCB0208X-W

"*** 1 ***"オプションを複数指定しました。最後に指定したオプションを有効にします。

要因：同じオプションを複数指定した。

(S) 無効なオプションを無視して実行を続ける。

(P) 同一オプションをまとめる。

KCCB0209X-W

"*** 1 ***"オプションは無効です。"*** 1 ***"オプションを無視します。

要因：無効なオプションを指定した。

(S) 無効なオプションを無視して実行を続ける。

(P) 無効なオプションを削除して再実行する。

KCCB0210X-E

"-dtd"オプションの指定がありません。"-dtd"オプションを指定してください。

要因："-dtd"オプションの指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) "-dtd"オプションを指定して再実行する。

KCCB0212X-E

"-o"オプションの指定がありません。"-o"オプションを指定してください。

要因："-o"オプションの指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) "-o"オプションを指定して再実行する。

KCCB0215X-E

DTD ファイル"*** 1 ***"の解析中に XML パーサエラーが発生しました。詳細情報を次に示します。

要因：DTD ファイルの記述に誤りがある。

(S) COBOL 原始プログラムの生成を中止する。

(P) DTD ファイルの記述を修正して再実行する。

注

cblxml コマンドのメッセージに続いて，XML パーサのエラーメッセージが出力されます。

KCCB0216X-E

DDF ファイル"*** 1 ***"の解析中に XML パーサエラーが発生しました。詳細情報を次に示します。

要因：DDF ファイルの記述に誤りがある。

(S) COBOL 原始プログラムの生成を中止する。

(P) DDF ファイルの記述を修正して再実行する。

注

cblxml コマンドのメッセージに続いて，XML パーサのエラーメッセージが出力されます。

KCCB0217X-E

DTD ファイル中に BaseElement 要素の elemName 属性に指定した"*** 1 ***"が見つかりません。

要因：DDF ファイルで BaseElement 要素の elemName 属性に指定した名称が DTD ファイルに定義されていない。

(S) COBOL 原始プログラムの生成を中止する。

(P) DTD ファイルに BaseElement 要素の elemName 属性に指定した名称の要素を追加する。または，BaseElement 要素の elemName 属性に指定した名称を変更して再実行する。

KCCB0218X-E

要素"*** 2 ***"中に要素"*** 1 ***"は存在しません。

要因：DDF ファイルで Item 要素，または Group 要素の elemName 属性に指定した名称が BaseElement 要素中に存在しない。

(S) COBOL 原始プログラムの生成を中止する。

(P) DDF ファイルに Item 要素，または Group 要素の elemName 属性に指定した名称の要素を追加する。または，Item 要素，Group 要素の elemName 属性に指定した名称を変更して再実行する。

KCCB0219X-E

cobName 属性"*** 1 ***"を持つ BaseElement 要素"*** 2 ***"と BaseElement 要素"*** 3 ***"があります。cobName 属性に一意な名称を指定してください。

要因：複数の BaseElement 要素の cobName 属性に同じ名称を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) BaseElement 要素の cobName 属性に指定する名称を一意に変更して再実行する。

KCCB0220X-E

*** 1 ***要素"*** 2 ***"は，*** 3 ***次元です。DTD ファイルでは，*** 4 ***回繰り返しています。繰り返しの次元が一致していません。

要因：DTD ファイルの繰り返し要素と DDF ファイルの Array 要素の対応付けが一致していない。

(S) COBOL 原始プログラムの生成を中止する。

(P) 繰り返し要素と Array 要素の対応付けを訂正して再実行する。

KCCB0221X-E

ファイル"*** 1 ***"に書き込めません。

要因：出力ファイルに書き込みができない。

(S) COBOL 原始プログラムの生成を中止する。

(P) ファイル，およびディレクトリ（フォルダ）のアクセス権限を見直す。

KCCB0222X-E

要素"*** 2 ***"に対応する AttrItem 要素"*** 1 ***"は，*** 3 ***次元です。DTD ファイルでは，*** 4 ***回繰り返しています。繰り返しの次元が一致していません。

要因：DTD ファイルの繰り返し要素と DDF ファイルの Array 要素の対応付けが一致していない。

(S) COBOL 原始プログラムの生成を中止する。

(P) 繰り返し要素と Array 要素の対応付けを訂正して再実行する。

KCCB0223X-E

要素"*** 1 ***"に複数の Item 要素は対応付けできません。

要因：一つの要素に複数の Item 要素を対応付けようとした。

(S) COBOL 原始プログラムの生成を中止する。

(P) 一つの要素に対応する Item 要素が一つになるように訂正して再実行する。

KCCB0224X-E

nameOfCountVar 属性"*** 1 ***"を持つ Array 要素と nameOfCountVar 属性"*** 2 ***"を持つ Array 要素が，同一の繰り返し記号"*** 3 ***"を持つ*** 4 ***要素に対応付けられています。Array 要素は一つだけ指定できます。

要因：複数の Array 要素を，XML ドキュメントの同じ繰り返し要素に対応付けようとした。

(S) COBOL 原始プログラムの生成を中止する。

(P) XML ドキュメントの一つの繰り返し要素に一つの Array 要素が対応するよう修正して再実行する。

KCCB0225X-E

DDF ファイルの不当な位置に*** 1 ***要素があります。

要因：DDF ファイル中の不当な位置に要素がある。

(S) COBOL 原始プログラムの生成を中止する。

(P) DDF ファイルの不当な位置にある要素を削除して再実行する。

KCCB0226X-W

XML アクセスルーチンの名称"*** 1 ***"が 30 バイトを超えています。

要因：XML アクセスルーチン名称を Interface 要素，および BaseElement 要素から作成する場合に，名称の長さが 30 バイトを超えた。

(S) COBOL 原始プログラムの生成を続行する。

(P) Interface 要素，および BaseElement 要素の長さを調整し 30 バイト以内で XML アクセスルーチンの条件を構成する。

KCCB0227X-E

*** 3 ***要素"*** 4 ***"の*** 2 ***属性に指定した値"*** 1 ***"は，不当です。

要因：COBOL のデータ項目の名称を指定する属性の値に空" "を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) COBOL のデータ項目名を指定できる名称に修正し，再実行する。

KCCB0228X-E

*** 1 ***属性に指定した値"*** 2 ***"に誤りがあります。

要因：elemName 属性または attrName 属性に指定した値に，XML の要素名に使えない文字を指定した。または，空を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) cobName 属性に指定した値を XML の要素名に指定できる名称に修正し，再実行する。または，elemName 属性，attrName 属性に要素名を指定する。

KCCB0229X-E

要素の子を持つ要素型宣言や混合内容宣言を Item 要素"*** 1 ***"に対応付けられません。(#PCDATA) の要素型宣言だけに対応付けできます。

要因：要素の子を持つ要素型宣言や混合内容宣言の要素を Item 要素に指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 要素の子を持つ要素型宣言，または混合内容宣言を（#PCDATA）に変更して再実行する。

KCCB0230X-W

countVar 属性"no"を指定した Array 要素に nameOfCountVar 属性を同時に指定した。

要因：countVar 属性"no"を指定した Array 要素に nameOfCountVar 属性を同時に指定した。

(S) COBOL 原始プログラムの生成を続行する。

(P) nameOfCountVar 属性を削除して再実行する。

KCCB0231X-E

Interface 要素の指定がありません。

要因：Interface 要素の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) Interface 要素を指定して再実行する。

KCCB0232X-E

nameOfCountVar 属性"*** 2 ***"を持つ Array 要素の内側に，nameOfCountVar 属性"*** 1 ***"を持つ Array 要素が指定されました。Array 要素の内側には，Item 要素，AttrItem 要素，又は Group 要素を一つだけ指定できます。

要因：Array 要素の内側に直接 Array 要素を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の内側に指定した Array 要素を削除して再実行する。

KCCB0233X-E

Interface 要素に interfaceName 属性の指定がありません。

要因：DDF で Interface 要素に interfaceName 属性の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) Interface 要素に interfaceName 属性を指定して再実行する。

KCCB0234X-E

Interface 要素"*** 1 ***"の内側に BaseElement 要素の指定がありません。

要因：Interface 要素の内側に BaseElement 要素の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) Interface 要素の内側に BaseElement 要素を指定して再実行する。

KCCB0235X-E

BaseElement 要素"*** 1 ***"の内側に Item 要素，AttrItem 要素，又は Group 要素の指定がありません。

要因：BaseElement 要素の内側に Item 要素，AttrItem 要素，または Group 要素の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) BaseElement 要素の内側に Item 要素，AttrItem 要素，または Group 要素を指定して再実行する。

KCCB0236X-E

BaseElement 要素に elemName 属性の指定がありません。

要因：BaseElement 要素に elemName 属性の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) BaseElement 要素に elemName 属性を指定して再実行する。

KCCB0238X-E

BaseElement 要素"*** 2 ***"の内側に不当な Item 要素"*** 1 ***"があります。BaseElement 要素は Item 要素，AttrItem 要素，又は Group 要素を一つだけ指定できます。

要因：BaseElement 要素の内側に不当な Item 要素がある。

(S) COBOL 原始プログラムの生成を中止する。

(P) BaseElement 要素の内側に指定した不当な Item 要素を削除して再実行する。

KCCB0239X-E

BaseElement 要素"*** 2 ***"の内側に不当な Group 要素"*** 1 ***"があります。BaseElement 要素は Item 要素，AttrItem 要素，又は Group 要素を一つだけ指定できます。

要因：BaseElement 要素の内側に不当な Group 要素がある。

(S) COBOL 原始プログラムの生成を中止する。

(P) BaseElement 要素の内側に指定した不当な Group 要素を削除して再実行する。

KCCB0240X-W

Item 要素"*** 1 ***"に不当な要素の値があります。指定した要素の値を無視します。

要因：値を指定できない Item 要素に値を指定した。

(S) COBOL 原始プログラムの生成を続行する。

(P) Item 要素を空要素に変更して再実行する。

KCCB0241X-E

DDF ファイル名の指定がありません。拡張子".cxd"を持つファイル名を指定してください。

要因：DDF ファイルの指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) DDF ファイルを指定して再実行する。

KCCB0242X-W

*** 2 ***要素"*** 3 ***"に不当な*** 1 ***属性があります。*** 1 ***属性を無視します。

要因：不当な属性を指定した。

(S) 不当な属性を無視して COBOL 原始プログラムの生成を続行する。

(P) 不当な要素を削除して再実行する。

KCCB0243X-E

DDF ファイル"*** 1 ***"が見つかりません。

要因：指定した DDF ファイルが見つからない。

(S) COBOL 原始プログラムの生成を中止する。

(P) 存在する DDF ファイルを指定して再実行する。

KCCB0244X-E

Group 要素に elemName 属性，又は cobName 属性の指定がありません。

要因：Group 要素に elemName 属性，または cobName 属性の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) Group 要素に elemName 属性，または cobName 属性を指定して再実行する。

KCCB0245X-E

DTD ファイル"*** 1 ***"が見つかりません。

要因：指定した DTD ファイルが見つからない。

(S) COBOL 原始プログラムの生成を中止する。

(P) 存在する DTD ファイルを指定して再実行する。

KCCB0246X-E

要素"*** 2 ***"の属性"*** 1 ***"に複数の AttrItem 要素は対応付けできません。

要因：一つの属性に複数の AttrItem 要素を対応付けようとした。

(S) COBOL 原始プログラムの生成を中止する。

(P) 一つの属性に対応する AttrItem 要素を一つに訂正して再実行する。

KCCB0248X-E

Group 要素"*** 1 ***"の内側に Item 要素，AttrItem 要素，Group 要素，又は Array 要素のどれかが指定されていません。

要因：Group 要素の内側に Item 要素，AttrItem 要素，Group 要素，または Array 要素の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) Group 要素の内側に Item 要素，AttrItem 要素，Group 要素，または Array 要素を追加して再実行する。

KCCB0249X-E

nameOfCountVar 属性"*** 1 ***"の Array 要素に max 属性が指定されていません。

要因：Array 要素に max 属性が指定されていない。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素に max 属性を指定して再実行する。

KCCB0250X-E

nameOfCountVar 属性"*** 2 ***"を持つ Array 要素の内側に，不当な Item 要素"*** 1 ***"があります。Array 要素は Item 要素，AttrItem 要素，又は Group 要素を一つだけ指定できます。

要因：Array 要素の内側に二つ以上の Item 要素があるか，AttrItem 要素，Item 要素と Group 要素が混在している。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の内側に指定した不当な Item 要素を削除して再実行する。

KCCB0251X-E

nameOfCountVar 属性"*** 2 ***"を持つ Array 要素の内側に，不当な Group 要素"*** 1 ***"があります。Array 要素は Item 要素，AttrItem 要素，又は Group 要素を一つだけ指定できます。

要因：Array 要素の内側に二つ以上の Group 要素があるか，Item 要素，AttrItem 要素，および Group 要素が混在している。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の内側に指定した不当な Group 要素を削除して再実行する。

KCCB0252X-E

nameOfCountVar 属性"*** 1 ***"を持つ Array 要素は，制限を超えた*** 2 ***次元にあります。最大*** 3 ***次元まで指定できます。

要因：Array 要素の次元が 7 を超えた。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の次元を 7 以下に変更して再実行する。

KCCB0253X-E

nameOfCountVar 属性"*** 1 ***"を持つ Array 要素の内側に Item 要素，AttrItem 要素，又は Group 要素の指定がありません。

要因：nameOfCountVar 属性を持つ Array 要素の内側に Item 要素，AttrItem 要素，または Group 要素の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の内側に Item 要素，AttrItem 要素，または Group 要素を追加して再実行する。

KCCB0256X-W

#REQUIRED 指定がある要素"*** 2 ***"の属性"*** 1 ***"が対応付けられていません。出力時に属性"*** 1 ***"の値に空（""）を仮定します。

要因：#REQUIRED の指定がある属性が，DDF で AttrItem 要素によって対応付けられていないため，出力時の属性値を COBOL データ項目から取得できない。

(S) その属性の値が DDF で対応付けられていないため，次の場合は属性値として空（""）を出力するものとして，COBOL 原始プログラムの生成を続行する。

- XML ドキュメント出力時に出力する値
- その属性が XML ドキュメント中の更新対象の要素（DDF で属性 update="yes"を指定した要素）に含まれる場合の，XML ドキュメント更新時に出力する値

(P) システムの処置で空（""）以外の属性値を出力する必要がある場合，その属性を，DDF で AttrItem 要素によって対応付ける。

KCCB0258X-E

Item 要素"*** 3 ***"の*** 1 ***属性に不当な値"*** 2 ***"が指定されました。

要因：Item 要素の属性値に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Item 要素の属性値を，その属性に合った値にして再実行する。

KCCB0259X-W

要素"*** 2 ***"に対応する AttrItem 要素"*** 1 ***"に不当な要素の値があります。指定した要素の値を無視します。

要因：値を指定できない AttrItem 要素に値を指定した。

(S) COBOL 原始プログラムの生成を続行する。

(P) AttrItem 要素を空要素に変更して再実行する。

KCCB0260X-E

AttrItem 要素に attrName 属性の指定がありません。

要因：AttrItem 要素に attrName 属性の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) AttrItem 要素の属性に，attrName 属性を指定して再実行する。

KCCB0261X-E

BaseElement 要素"*** 3 ***"の内側に，要素"*** 2 ***"に対応する不当な AttrItem 要素"*** 1 ***"があります。BaseElement 要素は Item 要素，AttrItem 要素，又は Group 要素を一つだけ指定できます。

要因：BaseElement 要素の内側に二つ以上の AttrItem 要素があるか，Item 要素，AttrItem 要素および Group 要素が混在している。

(S) COBOL 原始プログラムの生成を中止する。

(P) BaseElement 要素の内側に指定した不当な AttrItem 要素を削除して再実行する。

KCCB0262X-E

nameOfCountVar 属性"*** 3 ***"を持つ Array 要素の内側に，要素"*** 2 ***"に対応する不当な AttrItem 要素"*** 1 ***"があります。Array 要素は Item 要素，AttrItem 要素，又は Group 要素を一つだけ指定できます。

要因：Array 要素の内側に二つ以上の AttrItem 要素があるか，Item 要素，AttrItem 要素および Group 要素が混在している。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の内側に指定した不当な AttrItem 要素を削除して再実行する。

KCCB0263X-E

*** 3 ***要素"*** 4 ***"の*** 1 ***属性に"*** 2 ***"は指定できません。*** 1 ***属性には"yes"又は"no"を指定できます。

要因：属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 属性値に"yes"または"no"のどちらかを指定して再実行する。

KCCB0264X-E

Item 要素"*** 2 ***"の sign 属性に"*** 1 ***"は指定できません。sign 属性には"signed"又は"unsigned"を指定できます。

要因：Item 要素の sign 属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Item 要素の sign 属性に"signed"または"unsigned"のどちらかを指定して再実行する。

KCCB0265X-E

Item 要素"*** 2 ***"の sign 属性に"*** 1 ***"は指定できません。sign 属性は"signed"，"unsigned"，又は"leadingSeparate"を指定できます。

要因：Item 属性の sign 属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Item 要素の sign 属性に"signed"，"unsigned"，または"leadingSeparate"のどれかを指定して再実行する。

KCCB0267X-W

ファイル"*** 1 ***"を開くことができません。

要因：-catalog オプションに指定したカタログファイルのオープン処理中にエラーが発生した。次の要因が考えられる。

- ファイルがない。
- ファイルに対するアクセス権限がない。
- 処理中のファイル自身で入出力エラーが発生した。

(S) -catalog オプションを無視して COBOL 原始プログラムの生成を続行する。

(P) 原因を調査し，カタログファイルの指定を訂正して再実行する。

KCCB0268X-W

ファイル"*** 1 ***"の内容が不当です。

要因：-catalog オプションに指定したカタログファイルの内容が不当である。

(S) -catalog オプションを無視して COBOL 原始プログラムの生成を続行する。

(P) カタログファイルの書き方を訂正して再実行する。

KCCB0269X-E

AttrItem 要素に elemName 属性の指定がありません。

要因：AttrItem 要素に elemName 属性の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) AttrItem 要素に elemName 属性を指定して再実行する。

KCCB0271X-E

type 属性に"*** 2 ***"を持つ Item 要素"*** 3 ***"に*** 1 ***属性は指定できません。

要因：Item 要素に指定した属性が，その Item 要素の type 属性値に対して不当である。

(S) COBOL 原始プログラムの生成を中止する。

(P) Item 要素の属性に，type 属性値に対して指定可能な属性を指定する。

KCCB0272X-E

COBOL データ項目"*** 1 ***"は，COBOL データ項目"*** 2 ***"と重複しています。

要因：次に示す属性のうち，名称が重複しているものがある。

- cobName 属性
- elemName 属性（cobName 属性を省略した場合）
- elemName 属性-attrName 属性（cobName 属性を省略した場合）
- nameOfBaseVar 属性
- nameOfGroupVar 属性
- nameOfFlagVar 属性
- nameOfLengthVar 属性
- nameOfCountVar 属性
- nameOfTotalVar 属性

(S) COBOL 原始プログラムの生成を中止する。

(P) cobName 属性（省略時は elemName 属性または elemName 属性＋ attrName 属性），nameOfBaseVar 属性，nameOfGroupVar 属性，nameOfFlagVar 属性，nameOfLengthVar 属性，nameOfCountVar 属性，nameOfTotalVar 属性に指定した名称を一意に変更して，再実行する。

KCCB0273X-E

要素"*** 2 ***"に対応する AttrItem 要素"*** 1 ***"のレベルが制限を超えています。AttrItem 要素の位置を 49 レベル以内に記述してください。

要因：AttrItem 要素があるレベルが制限を超えた。

(S) COBOL 原始プログラムの生成を中止する。

(P) AttrItem 要素の位置が 49 レベル以内になるように訂正して再実行する。

KCCB0274X-E

Item 要素"*** 1 ***"の type 属性に指定した値"*** 2 ***"は不当です。type 属性に は"alphanumeric"，"national"，"numeric"，"packed"，"binary"，"float"，又は"double"を指定できます。

要因：Item 要素の type 属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Item 要素の type 属性に"alphanumeric"，"national"，"numeric"，"packed"，"binary"，"float"，または"double"のどれかを指定して再実行する。

KCCB0275X-E

Item 要素に elemName 属性の指定がありません。

要因：Item 要素に elemName 属性の指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) Item 要素に elemName 属性を指定して再実行する。

KCCB0277X-E

nameOfCountVar 属性"*** 2 ***"を持つ Array 要素の max 属性に指定した値"*** 1 ***"は不当です。

要因：Array 要素の max 属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の max 属性に 1～16,777,215 の範囲の値を指定して再実行する。

KCCB0278X-E

*** 3 ***属性"*** 4 ***"の*** 2 ***属性に"yes"を指定した場合は，*** 1 ***属性を指定できません。

要因：verbatim 属性に"yes"を指定した場合に trim 属性を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) verbatim 属性に"yes"を指定した場合，trim 属性を削除して再実行する。

KCCB0280X-E

要素"*** 3 ***"に対応する AttrItem 要素"*** 2 ***"の type 属性に指定した値と DTD のデフォルト属性値"*** 1 ***"の形式が異なります。

要因：AttrItem 要素の type 属性に指定した値と DTD のデフォルト属性値の形式が異なる。

(S) COBOL 原始プログラムの生成を中止する。

(P) AttrItem 要素の type 属性に指定した値と DTD のデフォルト属性値の形式を合わせる。

KCCB0281X-E

DTD ファイルに指定した要素"*** 1 ***"の定義が見つかりません。

要因：DTD に指定した要素の定義がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) DTD に指定した要素の定義を追加して再実行する。

KCCB0285X-E

Item 要素"*** 1 ***"のレベルが制限を超えています。Item 要素の位置を 49 レベル以内に記述してください。

要因：Item 要素があるレベルが制限を超えた。

(S) COBOL 原始プログラムの生成を中止する。

(P) Item 要素の位置が 49 レベル以内になるように訂正して再実行する。

KCCB0286X-E

Item 要素"*** 1 ***"のレベルが制限を超えています。Group 要素の位置を 49 レベル以内に記述してください。

要因：Group 要素があるレベルが制限を超えた。

(S) COBOL 原始プログラムの生成を中止する。

(P) Group 要素の位置が 49 レベル以内になるように訂正して再実行する。

KCCB0287X-E

DTD ファイル"*** 1 ***"の解析中に XML パーサで未知の例外が発生しました。

要因：DTD ファイルの XML の書き方に誤りがある。

(S) COBOL 原始プログラムの生成を中止する。

(P) DTD ファイルの XML の書き方を訂正して再実行する。

KCCB0290X-E

DTD ファイルの DOCTYPE 宣言に指定したルート要素"*** 1 ***"の定義が見つかりません。

要因：DOCTYPE 宣言に指定した要素に対応するルート要素宣言の定義がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) ルート要素の定義を追加して再実行する。

KCCB0291X-E

DTD ファイルに要素の定義がありません。

要因：DTD に要素の定義がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) DTD に要素の定義を追加して再実行する。

KCCB0292X-W

COBOL 予約語ファイル"*** 1 ***"が見つかりません。

要因：前提プログラムが正しくインストールされていない。

(S) COBOL 原始プログラムの生成を続行する。

(P) 前提プログラムを正しくインストールして再実行する。

KCCB0293X-W

COBOL 予約語ファイル"*** 1 ***"は壊れています。

要因：前提プログラムが正しくインストールされていない。

(S) COBOL 原始プログラムの生成を続行する。

(P) 前提プログラムを正しくインストールして再実行する。

KCCB0294X-E

*** 3 ***要素"*** 4 ***"の*** 2 ***属性に指定した"*** 1 ***"は，COBOL の予約語です。

要因：要素の属性値に COBOL の予約語を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 要素の属性値に COBOL の予約語ではない語を指定して再実行する。

KCCB0295X-W

*** 3 ***要素"*** 4 ***"の*** 2 ***属性に指定した"*** 1 ***"は，30 文字を超えています。

要因：次に示す属性で，30 文字を超える名称が指定されているものがある。

- cobName 属性
- nameOfBaseVar 属性
- nameOfGroupVar 属性
- nameOfFlagVar 属性
- nameOfLengthVar 属性
- nameOfCountVar 属性
- nameOfTotalVar 属性

(S) COBOL 原始プログラムの生成を続行する。

(P)「要因」で示した属性のうち，30 文字を超える名称を指定したものは，30 文字以下の名称に修正して再実行する。

KCCB0296X-E

*** 3 ***要素"*** 4 ***"の*** 2 ***属性に指定した"*** 1 ***"は，COBOL の語として不当な文字を含んでいます。

要因：要素の属性値に指定された名前が COBOL の語として不当な文字を含んでいる。

(S) COBOL 原始プログラムの生成を続行する。

(P) 要素の属性値に指定する名前が COBOL の語として使用できる文字から構成されるように修正して再実行する。

KCCB0297X-W

要素と対応付けていない Group 要素に nameOfFlagVar 属性は指定できません。Group 要素"*** 2 ***"の nameOfFlagVar 属性"*** 1 ***"を無視します。

要因：要素と対応付けていない Group 要素に不当な nameOfFlagVar 属性を指定した。

(S) COBOL 原始プログラムの生成を続行する。

(P) 要素と対応付けていない Group 要素から nameOfFlagVar 属性を削除して，再実行する。

KCCB0299X-E

Item 要素"*** 3 ***"に指定した type 属性値に対応しない*** 1 ***属性"*** 2 ***"を指定しました。

要因：Item 要素に指定した type 属性値に対応しない COBOL の定数を emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，overflowValue 属性のどれかに指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Item 要素に指定した type 属性値に対応する COBOL の定数を emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，または overflowValue 属性に指定して再実行する。

KCCB0300X-E

DDF ファイル"*** 1 ***"の解析中に XML パーサで未知の例外が発生しました。

要因：DDF ファイルの XML の書き方に誤りがある。

(S) COBOL 原始プログラムの生成を中止する。

(P) DDF ファイルの XML の書き方を訂正して再実行する。

KCCB0301X-E

要素"*** 3 ***"に対応する AttrItem 要素"*** 2 ***"の type 属性に不当な値"*** 1 ***"が指定されました。type 属性には"alphanumeric", "national"又は"numeric"を指定できます。

要因：AttrItem 要素の type 属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) AttrItem 要素の type 属性に"alphanumeric", "national"または"numeric"のどれかを指定して再実行する。

KCCB0303X-E

要素"*** 4 ***"に対応する AttrItem 要素"*** 3 ***"の*** 1 ***属性に不当な値"*** 2 ***"が指定されました。

要因：AttrItem 要素の属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) AttrItem 要素の属性値を，その属性に合った値にして再実行する。

KCCB0305X-E

要素"*** 3 ***"に対応する AttrItem 要素"*** 2 ***"の sign 属性に"*** 1 ***"は指定できません。sign 属性には"signed"，"unsigned"又は"leadingSeparate"を指定できます。

要因：AttrItem 要素の sign 属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) AttrItem 要素の sign 属性に"signed", "unsigned"または"leadingSeparate"のどれかを指定して再実行する。

KCCB0306X-E

要素"*** 4 ***"に対応する AttrItem 要素"*** 3 ***"に指定した type 属性"*** 2 ***"に対して*** 1 ***属性は指定できません。

要因：AttrItem 要素の type 属性に対して指定できない属性を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) AttrItem 要素の type 属性に対応する属性を指定して再実行する。

KCCB0307X-E

要素"*** 4 ***"に対応する AttrItem 要素"*** 3 ***"に指定した type 属性値に対応しない"*** 1 ***"属性"*** 2 ***"を指定しました。

要因：AttrItem 要素に指定した type 属性値に対応しない COBOL の定数を，emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，または overflowValue 属性に指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) AttrItem 要素に指定した type 属性値に対応する COBOL の定数を，emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，または overflowValue 属性に指定し再実行する。

KCCB0308X-W

要素"*** 4 ***"に対応する type 属性"*** 2 ***"で指定した AttrItem 要素"*** 3 ***"のデフォルト属性値"*** 1 ***"が 160 文字を超えています。

要因：ATTLIST 宣言に指定した属性のデフォルト値が 160 文字を超えている。

(S) type 属性値に従いデフォルト属性値の先頭の 160 文字だけを採用して生成を続ける。

(P) ATTLIST 宣言に指定した属性のデフォルト値を type 属性値に従い 160 文字以下に訂正して再実行する。

KCCB0309X-W

type 属性"*** 3 ***"を指定した Item 要素"*** 4 ***"の*** 1 ***属性"*** 2 ***"が 160 文字を超えています。*** 1 ***属性値の先頭の 160 文字だけを採用します。

要因：Item 要素に指定した emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，overflowValue 属性のどれかの値が 160 文字を超えている。

(S) type 属性値に従い emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，overflowValue 属性に指定した値の先頭の 160 文字だけを採用して生成を続ける。

(P) Item 要素に指定した emptyValue 属性の値を type 属性値に従い 160 文字以下に訂正して再実行する。

KCCB0310X-W

要素"*** 5 ***"に対応する type 属性"*** 3 ***"を指定した AttrItem 要素"*** 4 ***"の*** 1 ***属性"*** 2 ***"が 160 文字を超えています。*** 1 ***属性値の先頭の 160 文字だけを採用します。

要因：AttrItem 要素に指定した emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，overflowValue 属性のどれかの値が 160 文字を超えている。

(S) type 属性値に従い emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，overflowValue 属性に指定した値の先頭の 160 文字だけを採用して生成を続ける。

(P) AttrItem 要素に指定した emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，overflowValue 属性の値を type 属性値に従い 160 文字以下に訂正して再実行する。

KCCB0311X-W

accessInfo 属性に"yes"を指定した要素の内側にある要素に*** 1 ***属性は指定できません。*** 3 *** 要素"*** 4 ***"の*** 1 ***属性"*** 2 ***"を無視します。

要因：accessInfo 属性"yes"を指定した要素の内側にある要素に不当な emptyValue，emptyContentValue 属性，invalidCharValue 属性，または overflowValue 属性属性を指定した。

(S) COBOL 原始プログラムの生成を続行する。

(P) emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，または overflowValue 属性を削除して，再実行する。

KCCB0312X-E

要素"*** 4 ***"に対応する AttrItem 要素"*** 3 ***"の*** 1 ***属性に"*** 2 ***"は指定できません。*** 1 ***属性には"yes"又は"no"を指定できます。

要因：属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 属性に"yes"または"no"のどちらかを指定して再実行する。

KCCB0313X-E

要素"*** 2 ***"中に属性*** 1 ***は存在しません。

要因：属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) DTD ファイルで対応する要素の属性を宣言する。または，DDF ファイルで指定した属性を削除して再実行する。

KCCB0315X-E

DDF ファイルで再帰的な構造は対応付けできません。DDF ファイルの Group 要素"*** 2 ***"の下に*** 1 ***要素"*** 2 ***"があります。

要因：DDF ファイルに再帰的な構造を定義した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 再帰的な構造を定義しないように DDF ファイルを変更し，再実行する。

KCCB0317X-E

nameOfCountVar 属性"*** 3 ***"を持つ Array 要素の*** 1 ***属性に指定した値"*** 2 ***"は不当です。*** 1 ***属性には"yes"又は"no"を指定できます。

要因：Array 要素の属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 属性値に"yes"または"no"のどちらかを指定して再実行する。

KCCB0318X-W

要素"*** 3 ***"に対応する AttrItem 要素"*** 2 ***"に不当な*** 1 ***属性があります。*** 1 ***属性を無視します。

要因：AttrItem 要素に不当な要素を指定した。

(S) 不当な属性を無視して COBOL 原始プログラムの生成を続行する。

(P) AttrItem 要素に指定した不当な要素を削除して再実行する。

KCCB0319X-W

nameOfCountVar 属性"*** 2 ***"を持つ Array 要素に不当な*** 1 ***属性があります。*** 1 ***属性を無視します。

要因：AttrItem 要素に不当な要素を指定した。

(S) 不当な属性を無視して COBOL 原始プログラムの生成を続行する。

(P) AttrItem 要素に指定した不当な要素を削除して再実行する。

KCCB0320X-E

nameOfCountVar 属性"*** 3 ***"を持つ Array 要素の*** 2 ***属性に指定した"*** 1 ***"は，不当です。

要因：COBOL のデータ項目の名称を指定する Array 要素の属性の値に空""を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) COBOL のデータ項目名に指定できる名称に修正し，再実行する。

KCCB0321X-E

nameOfCountVar 属性"*** 3 ***"を持つ Array 要素の*** 2 ***属性に指定した"*** 1 ***"は，COBOL の予約語です。

要因：Array 要素の nameOfCountVar 属性に COBOL の予約語を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の nameOfCountVar 属性に COBOL の予約語でない語を指定して再実行する。

KCCB0322X-W

nameOfCountVar 属性"*** 3 ***"を持つ Array 要素の*** 2 ***属性に指定した"*** 1 ***"は，30 文字を超えています。

要因：Array 要素の nameOfCountVar 属性に 30 文字を超えた名称を指定した。

(S) COBOL 原始プログラムの生成を続行する。

(P) Array 要素の nameOfCountVar 属性に 30 文字以下の名称を指定して再実行する。

KCCB0323X-E

nameOfCountVar 属性"*** 3 ***"を持つ Array 要素の*** 2 ***属性に指定した"*** 1 ***"は，COBOL の語として不当な文字を含んでいます。

要因：Array 要素の nameOfCountVar 属性に COBOL の語として不当な文字を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の nameOfCountVar 属性に COBOL の予約語でない語を指定して再実行する。

KCCB0324X-W

update 属性"*** 4 ***"を指定した*** 5 ***要素"*** 6 ***"の内側にある*** 2 ***要素"*** 3 ***"に update 属性"*** 1 ***"を指定できません。内側に指定した update 属性を無視します。

要因：update 属性に"yes"を指定した Group 要素の内側にある Group 要素や Item 要素に対して，update 属性に"yes"を指定した。

(S) COBOL 原始プログラムの生成を続行する。

(P) update 属性に"yes"を指定した Group 要素の内側の要素に指定した update 属性"yes"を削除して再実行する。

KCCB0325X-E

*** 3 ***要素"*** 4 ***"の*** 1 ***属性に"*** 2 ***"は指定できません。値には"*** 5 ***"だけが指定できます。

要因：属性に不当な値を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 属性に指定した値を訂正して再実行する。

KCCB0326X-E

内部エラーが発生しました。 エラーコード = *** 1 ***, 詳細コード = *** 2 ***

要因：内部で論理エラーが発生した。

(S) COBOL 原始プログラムの生成を中止する。

(P) 当社保守員に連絡する。

KCCB0327X-E

カタログファイルに指定された公開識別子の長さが 1024 バイトを超えています。

要因：カタログファイルに 1,024 バイトを超えた公開識別子の名称を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) カタログファイルに指定する公開識別子の長さを 1,024 バイト以下に訂正し，再実行する。

KCCB0328X-E

カタログファイルに指定されたファイル名の長さが 255 バイトを超えています。

要因：カタログファイルに 255 バイトを超えたファイル名を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) カタログファイルに指定するファイル名の長さを 255 バイト以下に訂正し，再実行する。

KCCB0330X-E

DTD の要素"*** 1 ***"は無限に再帰しているため，対応付けできません。

要因：DTD で指定した要素が無限に再帰している。

(S) COBOL 原始プログラムの生成を中止する。

(P) 無限に再帰しないように DTD の要素を訂正し，再実行する。

KCCB0331X-E

*** 3 ***要素"*** 4 ***"の*** 2 ***属性に指定した"*** 1 ***"は，COBOL のプログラム名として不当な文字を含んでいます。

要因：Array 要素の nameOfCountVar 属性に COBOL の語として不当な文字を指定した。

(S) COBOL 原始プログラムの生成を中止する。

(P) Array 要素の nameOfCountVar 属性に COBOL の予約語でない語を指定して再実行する。

KCCB0332X-E

"-gen"オプションに指定した生成アクセスルーチンキーワード"*** 1 ***"は不当です。

要因："-gen"オプションに指定した生成アクセスルーチンキーワードでエラーが発生した。次の要因が考えられる。

- 生成アクセスルーチンキーワード以外の文字列を指定した。
- 生成アクセスルーチンキーワードの区切り文字にコンマ以外の文字や空白を指定した。
- 同じ生成アクセスルーチンキーワードを複数回指定した。
- 生成アクセスルーチンキーワードを，XML ドキュメントを開く，入力・出力，閉じるの構成で指定していない。

(S) COBOL 原始プログラムの生成を中止する。

(P)

- 生成アクセスルーチンキーワードに OP,OB,RD,WR,CL,CN のどれかを指定する。
- 生成アクセスルーチンキーワードの区切り文字にコンマを指定する。
- 二つ以上の同じ生成アクセスルーチンキーワードを一つにまとめる。

- XML ドキュメントを開く"OP，OB"，入力・出力"RD，WR"，閉じる"CL，CN"の生成アクセスルーチンキーワードから，それぞれ一つ以上を指定する。

KCCB0333X-E

"-outencoding"オプションに指定したエンコーディングキーワード"*** 1 ***"は不当です。"sjis"，"win31j"，"euc"，"utf8"，"utf16"，"utf16be"または"utf16le"を指定してください。

要因：-outencoding オプションに指定したエンコーディングキーワードでエラーが発生した。次の要因が考えられる。

- エンコーディングキーワードに不当な文字列を指定した。
- エンコーディングキーワードの指定がない。

(S) COBOL 原始プログラムの生成を中止する。

(P) -outencoding オプションに指定するエンコーディングキーワードを訂正し，再実行する。

KCCB0334X-E

"-unisrc"オプションを指定した場合，type 属性"*** 3 ***"を指定した Item 要素"*** 4 ***"の*** 1 *** 属性"*** 2 ***"を Unicode に変換した値の長さが size 属性の指定値を超えています。size 属性の指定値を大きくするか，または，*** 1 ***属性"*** 2 ***"の指定値を小さくしてください。

要因："-unisrc"オプションを指定した場合，Item 要素に指定した emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，overflowValue 属性のどれかの Unicode に変換した値の長さが size 属性で指定された長さを超えた。

(S) COBOL ソースの生成を中止する。

(P) Item 要素に指定した emptyValue 属性，emptyContentValue 属性，invalidCharValue 属性，overflowValue 属性の値を type 属性値に従い size 属性の設定値以下に修正するか，または，size 属性の設定値を Unicode 変換後の値が格納できる十分な長さに修正して，再実行する。

KCCB0335X-E

"-unisrc"オプションを指定した場合，Interface 要素，BaseElement 要素の cobName 属性(指定がない場合は elemName 属性)に Unicode に変換すると多バイトになる文字は指定できません。

要因："-unisrc"オプションを指定した場合，Interface 要素，BaseElement 要素の cobName 属性（指定がない場合は elemName 属性）のどちらかに，Unicode に変換すると多バイトになる文字を指定した。

(S) COBOL ソースの生成を中止する。

(P) Interface 要素，BaseElement 要素の cobName 属性（指定がない場合は elemName 属性）の，Unicode に変換すると多バイトになる文字をほかの文字に修正し，再実行する。

KCCB0336X-E

"-unisrc"オプションを指定した場合，"-o"オプションに指定する COBOL 副プログラム名に Unicode に変換すると多バイトになる文字は指定できません。

要因："-unisrc"オプションを指定した場合，"-o"オプションに指定する COBOL 副プログラム名に，Unicode に変換すると多バイトになる文字を指定した。

(S) COBOL ソースの生成を中止する。

(P) "-o"オプションに指定する COBOL 副プログラム名の，Unicode に変換すると多バイトになる文字をほかの文字に修正し，再実行する。

KCCB0337X-E

"-unisrc"オプションを指定した場合，環境変数 LANG はシフト JIS でなければなりません。

要因："-unisrc"オプションを指定した場合，環境変数 LANG にシフト JIS 以外の値が設定されている。または，環境変数 LANG が設定されていない。

(S) COBOL ソースの生成を中止する。

(P) 環境変数 LANG の値をシフト JIS に修正し，再実行する。

KCCB0338X-E

"-unisrc"オプションを指定した場合，環境変数 LANG は UTF-8 でなければなりません。

要因："-unisrc"オプションを指定した場合，環境変数 LANG に UTF-8 以外の値が設定されている。または，環境変数 LANG が設定されていない。

(S) COBOL ソースの生成を中止する。

(P) 環境変数 LANG の値を UTF-8 に修正し，再実行する。

KCCB0339X-E

環境変数 LANG に UTF-8 を指定した場合，"-unisrc"オプションを指定しなければなりません。

要因：環境変数 LANG に UTF-8 に対応する値が設定されている場合，"-unisrc"オプションが指定されていない。

(S) COBOL ソースの生成を中止する。

(P) "-unisrc"オプションを指定し，再実行する。

4.2　生成される XML アクセスルーチン

cblxml コマンドが生成する副プログラムには，COBOL プログラムから XML ドキュメントにアクセスするために必要なアクセスルーチン（XML アクセスルーチン）が格納されています。

次に，XML アクセスルーチンの一覧を示します。

- CBLXML-OP-*Interface* アクセスルーチン
  ファイル上の XML ドキュメントを開くときに呼び出すアクセスルーチンです。
- CBLXML-OB-*Interface* アクセスルーチン
  バッファ（メモリ）上の XML ドキュメントを開くときに呼び出すアクセスルーチンです。
- CBLXML-RD-*Interface-BaseElement* アクセスルーチン
  XML ドキュメントからデータを読み込むときに呼び出すアクセスルーチンです。
- CBLXML-WR-*Interface-BaseElement* アクセスルーチン
  XML ドキュメントへデータを書き込むときに呼び出すアクセスルーチンです。
- CBLXML-CL-*Interface* アクセスルーチン
  XML ドキュメントを閉じるときに呼び出すアクセスルーチンです。
- CBLXML-CN-*Interface* アクセスルーチン
  書き込みモードで開いた XML ドキュメントを閉じ，出力した XML ドキュメント長を取得するときに呼び出すアクセスルーチンです。

これらのアクセスルーチンを COBOL プログラム中から呼び出すことで，XML ドキュメントにアクセスできます。

4.2.1　XML アクセスルーチンの名称形式

XML アクセスルーチンは，次の名称形式で生成されます。

XML アクセスルーチン名の形式

CBLXML-xx-*Interface-BaseElement*

Interface

DDF 中の Interface 要素の interfaceName 属性に指定した名称が設定されます。

BaseElement

DDF 中の BaseElement 要素の cobName 属性に指定した名称が設定されます。cobName 属性を省略した場合は，elemName 属性に指定した名称が設定されます。

例

次の cblxml コマンドに入力する DDF と，生成される XML アクセスルーチン名の例を次に示します。

（DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="row" cobName="TABLEROW">
   :
 </BaseElement>
</Interface>
```

（生成される XML アクセスルーチンの名称）

- CBLXML-OP-EXAMPLE
- CBLXML-OB-EXAMPLE
- CBLXML-RD-EXAMPLE-TABLEROW
- CBLXML-WR-EXAMPLE-TABLEROW
- CBLXML-CL-EXAMPLE
- CBLXML-CN-EXAMPLE

4.2.2 CBLXML-OP-*Interface* アクセスルーチン

CBLXML-OP-*Interface* アクセスルーチンは，ファイル上の XML ドキュメントを開くためのアクセスルーチンです。このアクセスルーチンで開いた XML ドキュメントは，CBLXML-RD-*Interface-BaseElement* アクセスルーチンや CBLXML-WR-*Interface-BaseElement* アクセスルーチンを使ってデータを読み書きできます。

形式

```
CALL 'CBLXML-OP-Interface'
                USING XML-FILE-NAME
                      XML-FILE-NAME-LENGTH
                      XML-MODE
                      XML-POINTER
                RETURNING CBLXML-RETURN-CODE.
```

引数

引数のデータ型	指定	説明
01 XML-FILE-NAME USAGE POINTER.	○	開く XML ドキュメントのファイル名を指すポインタを指定する。
01 XML-FILE-NAME-LENGTH PIC 9(9) COMP.	○	開く XML ドキュメントのファイル名の長さを指定する。
01 XML-MODE PIC X(16).	○	XML ドキュメントのアクセスモード※を指定する。
01 XML-POINTER USAGE POINTER.	△	アクセスモードに'E'を指定しない場合，開いた XML ドキュメントのポインタが返される。 XML ドキュメントを開くことに失敗した場合，ポインタの値は保証しない。

引数のデータ型	指定	説明
	○	アクセスモードに'E'を指定した場合，CBLXML-CREATE-XML-POINTER サービスルーチンで作成した XML ドキュメントのポインタを指定する。
01 CBLXML-RETURN-CODE PIC 9(9) COMP.	△	ステータスが返される。詳細については，「7.3　XML アクセスルーチンが返すステータス」を参照のこと。

（凡例）

○：アクセスルーチンの呼び出し時，値を設定しておく項目

△：アクセスルーチンの完了時，値が設定される項目

注※

指定できるアクセスモード文字列を示します。括弧内の文字は省略できます。また，順不同です。

- R〔V〕〔N〕〔E〕
- W〔E〕
- U〔V〕〔N〕〔E〕

アクセスモード文字列と意味

文字列	意味
R	読み取りモード
W	書き込みモード
U	更新モード
V	妥当性チェックをする。妥当性チェック機能については，「7.2.8　入力 XML ドキュメントの妥当性チェック機能」を参照のこと。
N	外部エンティティ参照を展開しない。
E	エラー情報取得機能や公開識別子を使用するために，サービスルーチンの設定を引き継ぐ。エラー情報取得機能や公開識別子については「9.3.2　エラー情報の取得」と「9.3.3　公開識別子が指定された XML ドキュメント」を参照のこと。

規則

- 引数 XML-MODE には，1 バイト目からアクセスモードを示す文字を指定し，残りの領域には空白を指定します。
- XML ドキュメントを開くことに成功した場合，取得した XML ドキュメントのポインタを用いて CBLXML-CL-*Interface* アクセスルーチン，または CBLXML-CN-*Interface* アクセスルーチンで XML ドキュメントを閉じなければなりません。
- CBLXML-OP-*Interface* アクセスルーチンは，開こうとしている XML ドキュメントがすでに開かれているかどうかチェックしません。すでに開かれている XML ドキュメントに対して CBLXML-OP-*Interface* アクセスルーチンを実行すると，その XML ドキュメントに対する新しい XML ドキュメントのポインタが返されます。

- 読み取りモードまたは更新モードでは，アクセスモードに'V'と'N'を同時に指定できません。同時に指定したとき，あとに指定した文字列が有効となります。
- アクセスモードに'N'を指定したとき，外部エンティティ参照の該当する個所にはテキストデータがないものとして解析します。
- 環境変数 CBLXML_PARSE_NOXXE に'YES'を指定した場合，アクセスモードに'V'を指定していても，入力 XML ドキュメントの妥当性チェック機能は無効となります。詳細については，「付録 E.4　使用できる解析モードによる動作の違い」を参照してください。

注意事項

CBLXML-OP-*Interface* アクセスルーチンが返すステータスが 110 の場合は，CBLXML-GET-ERROR サービスルーチンを使用して詳細なエラー情報を取得できます。詳細については，「9.3.2　エラー情報の取得」を参照してください。

4.2.3　CBLXML-OB-*Interface* アクセスルーチン

CBLXML-OB-*Interface* アクセスルーチンは，バッファ（メモリ）上の XML ドキュメントを開くためのアクセスルーチンです。このアクセスルーチンで開いた XML ドキュメントは，CBLXML-RD-*Interface-BaseElement* アクセスルーチンや CBLXML-WR-*Interface-BaseElement* アクセスルーチンを使ってデータを読み書きできます。

形式

```
CALL 'CBLXML-OB-Interface'
                USING BUFFER
                      BUFFER-LENGTH
                      XML-MODE
                      XML-POINTER
                RETURNING CBLXML-RETURN-CODE.
```

引数

引数のデータ型	指定	説明
01 BUFFER USAGE POINTER.	○	開くバッファを指すポインタを指定する。
01 BUFFER-LENGTH PIC 9(9) COMP.	○	開くバッファの長さを指定する。
01 XML-MODE PIC X(16).	○	XML ドキュメントのアクセスモード※を指定する。
01 XML-POINTER USAGE POINTER.	△	アクセスモードに'E'を指定しない場合，開いた XML ドキュメントのポインタが返される。XML ドキュメントを開くことに失敗した場合，ポインタの値は保証しない。
	○	アクセスモードに'E'を指定した場合，CBLXML-CREATE-XML-POINTER サービスルーチンで作成した XML ドキュメントのポインタを指定する。

引数のデータ型	指定	説明
01 CBLXML-RETURN-CODE PIC 9(9) COMP.	△	ステータスが返される。詳細については，「7.3　XML アクセスルーチンが返すステータス」を参照のこと。

（凡例）

○：アクセスルーチンの呼び出し時，値を設定しておく項目

△：アクセスルーチンの完了時，値が設定される項目

注※

指定できるアクセスモード文字列を示します。括弧内の文字は省略できます。また，順不同です。

- R〔V〕〔N〕〔E〕
- W〔E〕
- U〔V〕〔N〕〔E〕

アクセスモード文字列と意味

文字列	意味
R	読み取りモード
W	書き込みモード
U	更新モード
V	妥当性チェックをする。妥当性チェック機能については，「7.2.8　入力 XML ドキュメントの妥当性チェック機能」を参照のこと。
N	外部エンティティ参照を展開しない。
E	エラー情報取得機能や公開識別子を使用するために，サービスルーチンの設定を引き継ぐ。エラー情報取得機能や公開識別子については「9.3.2　エラー情報の取得」と「9.3.3　公開識別子が指定された XML ドキュメント」を参照のこと。

規則

- 引数 XML-MODE には，1 バイト目からアクセスモードを示す文字を指定し，残りの領域には空白を指定します。
- XML ドキュメントを開くことに成功した場合，取得した XML ドキュメントのポインタを用いて CBLXML-CL-*Interface* アクセスルーチン，または CBLXML-CN-*Interface* アクセスルーチンで XML ドキュメントを閉じなければなりません。
- CBLXML-OB-*Interface* アクセスルーチンは，開こうとしている XML ドキュメントがすでに開かれているかどうかチェックしません。すでに開かれている XML ドキュメントに対して CBLXML-OB-*Interface* アクセスルーチンを実行すると，その XML ドキュメントに対する新しい XML ドキュメントのポインタが返されます。
- 読み取りモードまたは更新モードでは，アクセスモードに'V'と'N'を同時に指定できません。同時に指定したとき，あとに指定した文字列が有効となります。

- アクセスモードに'N'を指定したとき，外部エンティティ参照の該当する個所にはテキストデータがないものとして解析します。
- 環境変数 CBLXML_PARSE_NOXXE に'YES'を指定した場合，アクセスモードに'V'を指定していても，入力 XML ドキュメントの妥当性チェック機能は無効となります。詳細については，「付録 E.4　使用できる解析モードによる動作の違い」を参照してください。

注意事項

- CBLXML-OB-*Interface* アクセスルーチンでバッファ上の XML ドキュメントを開いた場合，実際に確保されているバッファ領域の大きさに関係なく，引数 BUFFER-LENGTH に指定した長さの領域をバッファとみなして入出力します。引数 BUFFER-LENGTH には，実際に確保したバッファ領域の大きさを超える長さを指定しないでください。
- 書き込みモードや更新モードで XML ドキュメントを開く場合，引数 BUFFER-LENGTH で指定する入出力領域の長さには，出力または更新後の XML ドキュメントを格納するのに十分な長さを指定してください。
- 読み取りモードで XML ドキュメントを開く場合，引数 BUFFER-LENGTH で指定した長さのバッファ全体を XML ドキュメントとみなして解析します。実際の XML ドキュメントの長さより大きな値を引数 BUFFER-LENGTH に指定すると，実際の XML ドキュメントの処理時間に加え，実際の XML ドキュメントのあとを引数 BUFFER-LENGTH で指定した長さまで解析するための処理時間が掛かります。引数 BUFFER-LENGTH で指定する長さには，実際の XML ドキュメントの長さを指定するようにしてください。
- CBLXML-OB-*Interface* アクセスルーチンが返すステータスが 110 の場合は，CBLXML-GET-ERROR サービスルーチンを使用して詳細なエラー情報を取得できます。詳細については，「9.3.2 エラー情報の取得」を参照してください。

4.2.4　CBLXML-RD-*Interface-BaseElement* アクセスルーチン

CBLXML-RD-*Interface-BaseElement* アクセスルーチンは，DDF の BaseElement 要素に指定した XML 要素の単位でデータの読み込みを実行します。1 回のアクセスルーチンの呼び出しで，DDF の Item 要素に対応するすべての要素のデータが読み込まれ，XML アクセス用データ定義に格納されます。

形式

```
CALL 'CBLXML-RD-Interface-BaseElement'
                USING XML-POINTER
                      XMLアクセス用データ定義
                RETURNING CBLXML-RETURN-CODE.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	データを読み込む XML ドキュメントのポインタを指定する。CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アク

引数のデータ型	指定	説明
		セスルーチンで XML ドキュメントを開いたときに取得したポインタを指定する。
XML アクセス用データ定義	△	cblxml コマンドで生成された XML アクセス用データ定義のデータ項目名を指定する。このデータ項目に，読み込まれたデータが格納される。
01 CBLXML-RETURN-CODE PIC 9(9) COMP.	△	ステータスが返される。詳細については，「7.3　XML アクセスルーチンが返すステータス」を参照のこと。

（凡例）

○：アクセスルーチンの呼び出し時，値を設定しておく項目

△：アクセスルーチンの完了時，値が設定される項目

規則

- 引数 XML-POINTER には，CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アクセスルーチンで XML ドキュメントを開いたときに取得したポインタを指定します。
- CBLXML-RD-*Interface-BaseElement* アクセスルーチンが回復可能エラー（ステータス 1～99）を返した場合，XML ドキュメントの入力を続行できます。
- CBLXML-RD-*Interface-BaseElement* アクセスルーチンが致命的エラー（ステータス 100～199）を返した場合，それ以上 XML ドキュメントの入力を続行できません。

  エラーが発生した XML ドキュメントに対しては，CBLXML-CL-*Interface* アクセスルーチンだけが実行できます。それ以外のアクセスルーチンを実行した場合，動作は保証しません。
- CBLXML-RD-*Interface-BaseElement* アクセスルーチンがエラーを返しても，XML ドキュメントは閉じられません。必ず CBLXML-CL-*Interface* アクセスルーチン，または CBLXML-CN-*Interface* アクセスルーチンを実行して XML ドキュメントを閉じてください。

4.2.5　CBLXML-WR-*Interface-BaseElement* アクセスルーチン

CBLXML-WR-*Interface-BaseElement* アクセスルーチンは，DDF の BaseElement 要素に指定した XML 要素の単位でデータの書き込みを実行します。1 回のアクセスルーチンの呼び出しで，DDF の Item 要素に対応するすべての要素のデータが書き込まれます。

形式

```
CALL 'CBLXML-WR-Interface-BaseElement'
                USING XML-POINTER
                      XMLアクセス用データ定義
                RETURNING CBLXML-RETURN-CODE.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	データを書き込む XML ドキュメントのポインタを指定する。CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アクセスルーチンで XML ドキュメントを開いたときに取得したポインタを指定する。
XML アクセス用データ定義	○	cblxml コマンドで生成された XML アクセス用データ定義のデータ項目名を指定する。このデータ項目に，書き込むデータを格納しておく。
01 CBLXML-RETURN-CODE PIC 9(9) COMP.	△	ステータスが返される。詳細については，「7.3　XML アクセスルーチンが返すステータス」を参照のこと。

（凡例）

○：アクセスルーチンの呼び出し時，値を設定しておく項目

△：アクセスルーチンの完了時，値が設定される項目

規則

- 引数 XML-POINTER には，CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アクセスルーチンで XML ドキュメントを開いたときに取得したポインタを指定します。
- CBLXML-WR-*Interface-BaseElement* アクセスルーチンが回復可能エラー（ステータス 1～99）を返した場合，XML ドキュメントの出力を続行できます。
- CBLXML-WR-*Interface-BaseElement* アクセスルーチンが致命的エラー（ステータス 100～199）を返した場合，それ以上 XML ドキュメントの出力を続行できません。

  エラーが発生した XML ドキュメントに対しては，CBLXML-CL-*Interface* アクセスルーチン，または CBLXML-CN-*Interface* アクセスルーチンだけが実行できます。それ以外のアクセスルーチンを実行した場合，動作は保証しません。
- CBLXML-WR-*Interface-BaseElement* アクセスルーチンがエラーを返しても，XML ドキュメントは閉じられません。必ず CBLXML-CL-*Interface* アクセスルーチン，または CBLXML-CN-*Interface* アクセスルーチンを実行して XML ドキュメントを閉じてください。CBLXML-WR-*Interface-BaseElement* アクセスルーチンが回復可能エラー，または致命的エラーを返した場合，不完全な XML ドキュメントが生成される可能性があります。
- 出力，更新した場合の XML ドキュメントの文字エンコーディングは，「付録 G.1(1)　出力する XML ドキュメントの文字エンコーディングの設定」に従って出力されます。更新機能を使用する場合，開いた XML ドキュメントの文字エンコーディングではなく，出力する XML ドキュメントの文字エンコーディングの設定に従って変更されます。
- 更新モードで開いた XML ドキュメントに対応する DDF ファイルで，update 属性に"yes"を指定していない場合，CBLXML-WR-*Interface-BaseElement* アクセスルーチンは，XML ドキュメントを更新できないことを示すステータス 13 を返します。
- 更新モードで開いた XML ドキュメントから入力をしないで出力した場合，ステータス 15 が返ります。

- 更新モードで開いた XML ドキュメントで，直前の入力に対して異なる BaseElement 属性で出力した場合，ステータス 15 が返ります。

4.2.6 CBLXML-CL-*Interface* アクセスルーチン

CBLXML-CL-*Interface* アクセスルーチンは，開いている XML ドキュメントを閉じるためのアクセスルーチンです。

形式

```
CALL 'CBLXML-CL-Interface'
                USING XML-POINTER
                RETURNING CBLXML-RETURN-CODE.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	閉じる XML ドキュメントのポインタを指定する。CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アクセスルーチンで XML ドキュメントを開いたときに取得したポインタを指定する。
01 CBLXML-RETURN-CODE PIC 9(9) COMP.	△	ステータスが返される。詳細については，「7.3 XML アクセスルーチンが返すステータス」を参照のこと。

（凡例）

○：アクセスルーチンの呼び出し時，値を設定しておく項目

△：アクセスルーチンの完了時，値が設定される項目

規則

- 引数 XML-POINTER には，CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アクセスルーチンで XML ドキュメントを開いたときに取得したポインタを指定します。
- 更新モードで開いた XML ドキュメントを更新しないで CBLXML-CL-*Interface* アクセスルーチンを呼び出した場合，CBLXML-CL-*Interface* アクセスルーチンは，XML ドキュメントが更新されていないことを示すステータス 12（CBLXML-NO-UPDATE）を返します。
- ドキュメントを書き込みモードで開いている場合，CBLXML-CL-*Interface* アクセスルーチンは，XML ドキュメントを閉じる前に終了タグを出力します。このとき，次のように必要な要素が補完されます。

ドキュメント終了時の要素の補完

- 最後に書き出された BaseElement 要素とドキュメントの終わりの間に，さらに BaseElement 要素が必要なとき，CBLXML-CL-*Interface* アクセスルーチンは，ステータス 118（CBLXML-CANT-END-DOC）を返します。このとき，XML ドキュメントは，不完全な状態となります。正しい XML ドキュメントを生成するためには，必要な BaseElement 要素を出力する必要があります。

- 最後に書き出された BaseElement 要素とドキュメントの終わりの間に，BaseElement 要素に含まれない要素が必要なときは，その要素が空要素として出力されます。このとき，CBLXML-CL-*Interface* アクセスルーチンは，ステータス 0（CBLXML-OK）を返します。CBLXML-CL-*Interface* アクセスルーチンが回復可能エラー，または致命的エラーを返した場合で，そのドキュメントが書き込みモードで開かれていたときは，不完全な XML ドキュメントが生成される可能性があります。

注意事項

XML ドキュメントの出力中にディスク容量の不足のエラーが発生したあと，その XML ドキュメントを閉じた場合，CBLXML-CN-*Interface* アクセスルーチンで次に示すステータスを返します。

（UNIX の場合）

XML アクセスルーチンに指定した引数に誤りがあることを示すステータス 114（CBLXML-INVALID-PARAMS）を返します。

（Windows の場合）

正常終了を示すステータス 0（CBLXML-OK）を返します。

4.2.7 CBLXML-CN-*Interface* アクセスルーチン

CBLXML-CN-*Interface* アクセスルーチンは，書き込みモードで開いている XML ドキュメントを閉じ，出力した XML ドキュメント長を取得するためのアクセスルーチンです。

形式

```
CALL 'CBLXML-CN-Interface'
                USING XML-POINTER XML-LENGTH
                RETURNING CBLXML-RETURN-CODE.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	閉じる XML ドキュメントのポインタを指定する。CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アクセスルーチンで XML ドキュメントを開いたときに取得したポインタを指定する。
01 XML-LENGTH PIC 9(10) COMP.	△	出力した XML ドキュメントの長さが返される。
01 CBLXML-RETURN-CODE PIC 9(9) COMP.	△	ステータスが返される。詳細については，「7.3 XML アクセスルーチンが返すステータス」を参照のこと。

（凡例）

○：アクセスルーチンの呼び出し時，値を設定しておく項目

△：アクセスルーチンの完了時，値が設定される項目

規則

- CBLXML-CN-*Interface* アクセスルーチンが正常終了すると，XML ドキュメントが閉じられ，引数 XML-LENGTH に出力した XML ドキュメントの長さが返されます。
- 引数 XML-POINTER には，CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アクセスルーチンで XML ドキュメントを開いたときに取得したポインタを指定します。
- 更新モードで開いた XML ドキュメントを更新しないで CBLXML-CN-*Interface* アクセスルーチンを呼び出した場合，CBLXML-CN-*Interface* アクセスルーチンは，XML ドキュメントが更新されていないことを示すステータス 12（CBLXML-NO-UPDATE）を返します。その場合，XML ドキュメントの長さ（XML-LENGTH）には正常終了を示すステータス 0（CBLXML-OK）が返ります。
- 読み取りモードで開かれた XML ドキュメントを CBLXML-CN-*Interface* アクセスルーチンで閉じた場合，引数 XML-LENGTH には正常終了を示すステータス 0（CBLXML-OK）が返されます。このとき，CBLXML-CN-*Interface* アクセスルーチンは，ステータス 9（CBLXML-NOT-WRITTEN）を返します。
- CBLXML-WR-*Interface-BaseElement* アクセスルーチンや CBLXML-CN-*Interface* アクセスルーチンが回復可能エラー，または致命的エラーを返した場合，引数 XML-LENGTH に返される値は不定となります。
- ドキュメントを書き込みモードで開いている場合，CBLXML-CN-*Interface* は，XML ドキュメントを閉じる前に終了タグを出力します。このとき，次のように必要な要素が補完されます。

ドキュメント終了時の要素の補完

- 最後に書き出された BaseElement 要素とドキュメントの終わりの間に，さらに BaseElement 要素が必要なとき，CBLXML-CN-*Interface* アクセスルーチンは，ステータス 118（CBLXML-CANT-END-DOC）を返します。このとき，XML ドキュメントは，不完全な状態となります。正しい XML ドキュメントを生成するためには，必要な BaseElement 要素を出力する必要があります。
- 最後に書き出された BaseElement 要素とドキュメントの終わりの間に，BaseElement 要素に含まれない要素が必要なときは，その要素が空要素として出力されます。このとき，CBLXML-CN-*Interface* アクセスルーチンは，ステータス 0（CBLXML-OK）を返します。

注意事項

XML ドキュメントの出力中にディスク容量の不足のエラーが発生したあと，その XML ドキュメントを閉じた場合，CBLXML-CN-*Interface* アクセスルーチンで次に示すステータスを返します。

（UNIX の場合）

XML アクセスルーチンに指定した引数に誤りがあることを示すステータス 114（CBLXML-INVALID-PARAMS）を返します。

（Windows の場合）

正常終了を示すステータス 0（CBLXML-OK）を返します。

4.3　生成される XML アクセス用データ定義

XML アクセス用データ定義は，COBOL プログラムからアクセスする XML 要素に対応したデータ項目が格納された登録集原文です。このデータ項目は，XML アクセスルーチンでデータの読み書きをするとき，COBOL プログラムと XML ドキュメントとのデータの受け渡しに使用します。

XML アクセス用データ定義は，DDF の BaseElement 要素ごとに 01 レベルを持つ集団項目となります。集団項目の名称は，BaseElement 要素の cobName 属性に指定した値となります。

XML アクセス用データ定義の登録集原文は，COBOL プログラムのデータ部で COPY 文を使用して取り込みます。

4.4　XML アクセス用ステータス定義

XML アクセス用ステータス定義は，XML アクセスルーチンのステータスコード（実行結果を表す値）とエラー定義名称（ステータスコードごとに定義された名称）との対応が格納された登録集原文です。この定義によって，一意のエラー定義名称を使って原始プログラムのステータスの表現を統一できます。

XML アクセス用ステータス定義は，「CBLXMLRC.cbl」という名称の登録集原文として，XML 連携機能によって提供されます。そのため，XML 対応 COBOL プログラムのコンパイル時には，XML アクセス用ステータス定義の格納ディレクトリ（フォルダ）が登録集原文の格納ディレクトリ（フォルダ）になるよう，COBOL の環境変数 CBLLIB を設定する必要があります。詳細については，「6.1.1(1)　コンパイルに必要な環境変数の設定」，「6.2.1(1)　環境変数の設定」を参照してください。

XML アクセス用ステータス定義の格納場所

（UNIX の場合）

COBOL2002のインストールディレクトリ/copy

（Windows の場合）

COBOL2002のインストールフォルダ¥copy

5 XMLアクセスルーチンを使用したCOBOLプログラムの作成

この章では，XMLアクセスルーチンを使ってXMLドキュメントにアクセスするCOBOLプログラムを作成する方法について説明します。

5.1 XML ドキュメントの読み込み

XML ドキュメントからデータを読み込むには，次のような手順の COBOL プログラムを作成します。

1. XML 要素を読み込むために，cblxml コマンドで生成された XML アクセス用データ定義を COPY 文で取り込む
2. CBLXML-OB-*Interface* アクセスルーチンを呼び出し，XML ドキュメントを定義しているメモリ領域を開く
3. CBLXML-RD-*Interface-BaseElement* アクセスルーチンを呼び出し，XML 要素に対応する COBOL データ項目にデータを読み込む
4. 読み込んだデータを処理する
5. CBLXML-CL-*Interface* アクセスルーチンを呼び出し，XML ドキュメントを閉じる
6. プログラムを終了する

5.1.1 DTD，DDF の例

次に示すコーディング例が対象とする DTD，DDF，および XML アクセス用データ定義の例です。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (row)*>
  <!ELEMENT row      (name, address, grade, age)>
  <!ELEMENT name     (#PCDATA)>
  <!ELEMENT address  (addr1, addr2, city, state,
                      postcode)>
  <!ELEMENT addr1    (#PCDATA)>
  <!ELEMENT addr2    (#PCDATA)>
  <!ELEMENT city     (#PCDATA)>
  <!ELEMENT state    (#PCDATA)>
  <!ELEMENT postcode (#PCDATA)>
  <!ELEMENT grade    (#PCDATA)>
  <!ELEMENT age      (#PCDATA)>
]>
<table/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="row">
  <Group cobName="row">
   <Item elemName="name" cobName="full-name"
         type="alphanumeric" size="20"/>
   <Item elemName="age"  type="numeric" size="3"/>
```

```
   <Item elemName="city" type="alphanumeric"
         size="30"/>
  </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL のデータ項目)

```
01 row.
 02 full-name PIC X(20).
 02 age PIC 9(3).
 02 city PIC X(30).
```

5.1.2 XML ドキュメントの読み込みのコーディング例

メモリ上の XML ドキュメントからデータを読み込むプログラムのコーディング例を，次に示します。この例では，テスト用に XML 要素を定義したメモリ空間をバッファに作成しています。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. READSAMPLE.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
         COPY 'subprog-COPY.cbl'.
       01 XML-POINTER USAGE POINTER.
       01 XML-ERROR-CODE PIC 9(9) USAGE COMP VALUE 0.
       01 BUFFER-MODE PIC X(16).
       01 BUFFER-LENGTH PIC 9(9) COMP VALUE 284.
       01 BUFFER USAGE POINTER VALUE NULL.

      * テストデータ
       01 TEST-DATA.
        02 TEST-DATA1 PIC X(147) VALUE
       '<?xml version="1.0" encoding="Shift_JIS"?><table>
      -'<!--typical values--><row><name>John Smith</name>
      -'<age>20</age><city>Irvine</city></row><row>
      -'<name>'.
        02 TEST-DATA2 PIC X(137) VALUE
       'Jane Doe</name><age>30</age><city>Los Angeles
      -'</city></row><row><name>Tim Johnson</name><age>100
      -'</age><city>San Diego</city></row></table>'.
       PROCEDURE DIVISION.
      *    プログラム開始
           DISPLAY ' COBOL Data Types - Buffer  READ'.

      *    テストデータ
           COMPUTE BUFFER = FUNCTION ADDR(TEST-DATA).

      *    XMLドキュメントを入力するための
      *    オープンモードを'READ'に設定
           MOVE 'R' TO BUFFER-MODE.

      *    XMLドキュメントを開く
           PERFORM OPEN-BUFFER.
```

```
           DISPLAY '      *****    Read 1st Row    *****'.

       *   row要素の値をXMLドキュメントから入力する
           PERFORM READ-BUFFER.

       *   入力値を確認する
           IF FULL-NAME NOT EQUAL TO 'John Smith' THEN
               DISPLAY 'ERROR! - FULL-NAME: ' FULL-NAME
           ELSE
               DISPLAY FULL-NAME
           END-IF.

           IF AGE NOT EQUAL TO 20 THEN
               DISPLAY 'ERROR! - age: ' AGE
           ELSE
               DISPLAY AGE
           END-IF.

           IF CITY NOT EQUAL TO 'Irvine' THEN
               DISPLAY 'ERROR! - city: ' CITY
           ELSE
               DISPLAY CITY
           END-IF.


           DISPLAY '      *****    Read 2nd ROW    *****'.
       *   row要素の値をXMLドキュメントから入力する
           PERFORM READ-BUFFER.

       *   入力値を確認する
           IF FULL-NAME NOT EQUAL TO 'Jane Doe' THEN
               DISPLAY 'ERROR! - FULL-NAME: ' FULL-NAME
           ELSE
               DISPLAY FULL-NAME
           END-IF.

           IF AGE NOT EQUAL TO 30 THEN
               DISPLAY 'ERROR! - age: ' AGE
           ELSE
               DISPLAY AGE
           END-IF.

           IF CITY NOT EQUAL TO 'Los Angeles' THEN
               DISPLAY 'ERROR! - city: ' CITY
           ELSE
               DISPLAY CITY
           END-IF.


           DISPLAY '      *****    Read 3rd ROW    *****'.
       *   row要素の値をXMLドキュメントから入力する
           PERFORM READ-BUFFER.

       *   入力値を確認する
           IF FULL-NAME NOT EQUAL TO 'Tim Johnson' THEN
               DISPLAY 'ERROR! - FULL-NAME: ' FULL-NAME
           ELSE
```

```
           DISPLAY FULL-NAME
       END-IF.

         IF AGE NOT EQUAL TO 100 THEN
           DISPLAY 'ERROR! - age: ' AGE
       ELSE
           DISPLAY AGE
       END-IF.

       IF CITY NOT EQUAL TO 'San Diego' THEN
           DISPLAY 'ERROR! - city: ' CITY
       ELSE
           DISPLAY CITY
       END-IF.


      *    XMLドキュメントを閉じる
           PERFORM CLOSE-BUFFER.

      *    プログラム終了
           STOP RUN.


      * XMLアクセスルーチン（メモリ空間を開く）
       OPEN-BUFFER.
           CALL 'CBLXML-OB-EXAMPLE'
               USING BUFFER BUFFER-LENGTH
                     BUFFER-MODE XML-POINTER
               RETURNING XML-ERROR-CODE.

           IF XML-ERROR-CODE NOT EQUAL 0
               MOVE XML-ERROR-CODE TO RETURN-CODE
               DISPLAY
              'Failed to open XML document (open) - Error '
                   XML-ERROR-CODE
               GO TO ENDOFPROGRAM
           END-IF.

      * XMLアクセスルーチン（入力）
       READ-BUFFER.
           CALL 'CBLXML-RD-EXAMPLE-row'
               USING XML-POINTER ROW
               RETURNING XML-ERROR-CODE.

           IF XML-ERROR-CODE NOT EQUAL 0
               MOVE XML-ERROR-CODE TO RETURN-CODE
               DISPLAY
                   'Failed to read XML document - Error '
                   XML-ERROR-CODE
               GO TO ENDOFPROGRAM
           END-IF.

      * XMLアクセスルーチン（閉じる）
       CLOSE-BUFFER.
           CALL 'CBLXML-CL-EXAMPLE'
               USING XML-POINTER
               RETURNING XML-ERROR-CODE.
```

```
           IF XML-ERROR-CODE NOT EQUAL 0
               MOVE XML-ERROR-CODE TO RETURN-CODE
               DISPLAY
                  'Failed to close XML document - Error '
                   XML-ERROR-CODE
               GO TO ENDOFPROGRAM
           END-IF.

       ENDOFPROGRAM.
           STOP RUN.

       END PROGRAM READSAMPLE.
```

5.2 XML ドキュメントの書き込み

XML ドキュメントへデータを書き込むには，次のような手順の COBOL プログラムを作成します。

1. XML 要素を書き込むために，cblxml コマンドで生成された XML アクセス用データ定義を COPY 文で取り込む
2. CBLXML-OB-*Interface* アクセスルーチンを呼び出し，XML ドキュメントを定義しているメモリ領域を開く
3. XML 要素に書き込むためのデータを作成する
4. CBLXML-WR-*Interface-BaseElement* アクセスルーチンを呼び出し，COBOL データ項目から対応する XML 要素にデータを書き込む
5. CBLXML-CL-*Interface* アクセスルーチンを呼び出し，XML ドキュメントを閉じる
6. プログラムを終了する

5.2.1 XML ドキュメントの書き込みのコーディング例

メモリ上の XML ドキュメントへデータを書き込むプログラムのコーディング例を，次に示します。なお，このコーディング例が対象にしている DTD，DDF については，「5.1.1 DTD，DDF の例」を参照してください。

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. WRITESAMPLE.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
         COPY 'subprog-COPY.cbl'.
       01 BUFFER USAGE POINTER VALUE NULL.
       01 BUFFER-DATA PIC X(500).
       01 BUFFER-LENGTH PIC 9(9) COMP VALUE 500.
       01 BUFFER-MODE PIC X(16).
       01 XML-POINTER USAGE POINTER.
       01 XML-ERROR-CODE PIC 9(9) COMP VALUE 0.
       PROCEDURE DIVISION.
      *    プログラム開始
           DISPLAY ' COBOL Data Types - WRITE'.

      *    XMLドキュメントを入力するための
      *    オープンモードを'WRITE'に設定
           MOVE 'W' TO BUFFER-MODE.

      *    XMLドキュメントを開く
           PERFORM OPEN-BUFFER.

      *    row要素の値を初期化
           MOVE SPACES TO ROW.
```

```
     DISPLAY '      *****    Write 1st Row    *****'.

     MOVE 'John Smith'   TO FULL-NAME.
     MOVE 20             TO AGE.
     MOVE 'Irvine'       TO CITY.

*    row要素の値をXMLドキュメントに出力する
     PERFORM WRITE-BUFFER.


     DISPLAY '      *****    Write 2nd ROW    *****'.

     MOVE 'Jane Doe'     TO FULL-NAME.
     MOVE 30             TO AGE.
     MOVE 'Los Angeles'  TO CITY.

*    row要素の値をXMLドキュメントに出力する
     PERFORM WRITE-BUFFER.


     DISPLAY '      *****    Write 3rd ROW    *****'.

     MOVE 'Tim Johnson'  TO FULL-NAME.
     MOVE 100            TO AGE.
     MOVE 'San Diego'    TO CITY.

*    row要素の値をXMLドキュメントに出力する
     PERFORM WRITE-BUFFER.


*    XMLドキュメントを閉じる
     PERFORM CLOSE-BUFFER.
     STOP RUN.


* XMLアクセスルーチン（メモリ空間を開く）
 OPEN-BUFFER.
     COMPUTE BUFFER = FUNCTION ADDR(BUFFER-DATA).
     CALL 'CBLXML-OB-EXAMPLE'
         USING BUFFER BUFFER-LENGTH
               BUFFER-MODE XML-POINTER
         RETURNING XML-ERROR-CODE.

     IF XML-ERROR-CODE NOT EQUAL 0
         MOVE XML-ERROR-CODE TO RETURN-CODE
         DISPLAY
        'Failed to open XML document (open) - Error '
             XML-ERROR-CODE
         GO TO ENDOFPROGRAM
     END-IF.

* XMLアクセスルーチン（出力する）
 WRITE-BUFFER.
     CALL 'CBLXML-WR-EXAMPLE-row'
         USING XML-POINTER ROW
         RETURNING XML-ERROR-CODE.
```

```
           IF XML-ERROR-CODE NOT EQUAL 0
               MOVE XML-ERROR-CODE TO RETURN-CODE
               DISPLAY
                   'Failed write base element - Error '
                   XML-ERROR-CODE
                   GO TO ENDOFPROGRAM
           END-IF.

       * XMLアクセスルーチン（閉じる）
        CLOSE-BUFFER.
           CALL 'CBLXML-CL-EXAMPLE'
               USING XML-POINTER
               RETURNING XML-ERROR-CODE.

           IF XML-ERROR-CODE NOT EQUAL 0
               MOVE XML-ERROR-CODE TO RETURN-CODE
               DISPLAY
                   'Failed to close XML document - Error '
                   XML-ERROR-CODE
               GO TO ENDOFPROGRAM
           END-IF.

        ENDOFPROGRAM.
           STOP RUN.

        END PROGRAM WRITESAMPLE.
```

6 コンパイルとリンケージ

この章では，XML 対応 COBOL プログラムのコンパイル，およびリンケージを実行して，実行可能ファイルを作成する方法について説明します。

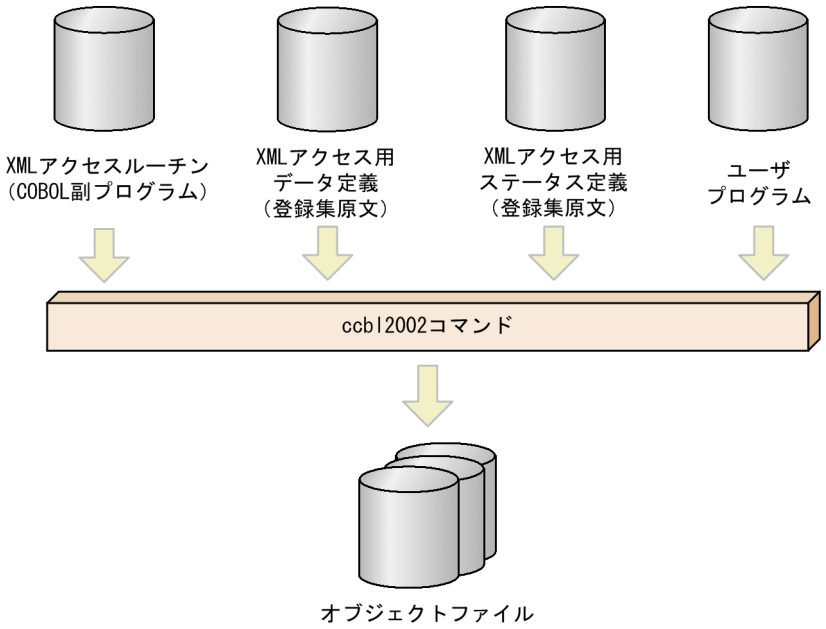
COBOL プログラムのコンパイル，およびリンケージの詳細は，マニュアル「COBOL2002　使用の手引　手引編」またはマニュアル「COBOL2002　ユーザーズガイド」を参照してください。

6.1 UNIX で作成した COBOL プログラムの UNIX でのコンパイルとリンケージ

UNIX で作成した COBOL プログラムを，UNIX でコンパイル，リンケージする方法について説明します。

6.1.1 コンパイル

ccbl2002 コマンドでコンパイルを実行して，オブジェクトファイル（.o）を生成します。



(1) コンパイルに必要な環境変数の設定

XML 対応 COBOL プログラムをコンパイルする場合は，次に示す環境変数を設定しておく必要があります。

- システム環境変数 PATH

  ccbl2002 コマンドの格納パスを設定します。次の値を指定してください。

  COBOL2002のインストールディレクトリ/bin

- COBOL の環境変数 CBLLIB

  XML ステータス定義（登録集原文 CBLXMLRC.cbl）の格納パスを設定します。次の値を指定してください。

  COBOL2002のインストールディレクトリ/copy

  XML ステータス定義の詳細については，「4.4 XML アクセス用ステータス定義」を参照してください。

指定例

sh（B シェル）の場合

（AIX(32)，Linux(x86)の場合）

```
CBLLIB=/opt/HILNGcbl2k/copy
export CBLLIB
```

（AIX(64)，Linux(x64)の場合）

```
CBLLIB=/opt/HILNGcbl2k64/copy
export CBLLIB
```

(2) コンパイルの指定例

コンパイルの指定例を次に示します。

指定例

（AIX の場合）

```
ccbl2002 -Compile,NoLink -Main,System mainprog.cbl subprog.cbl
```

（Linux の場合）

```
ccbl2002 -Compile,NoLink -UniObjGen -Main,System mainprog.cbl subprog.cbl
```

mainprog.cbl

作成した COBOL 主プログラムファイルの名称です。

subprog.cbl

cblxml コマンドによって生成された COBOL 副プログラムファイル（XML アクセスルーチン）の名称です。

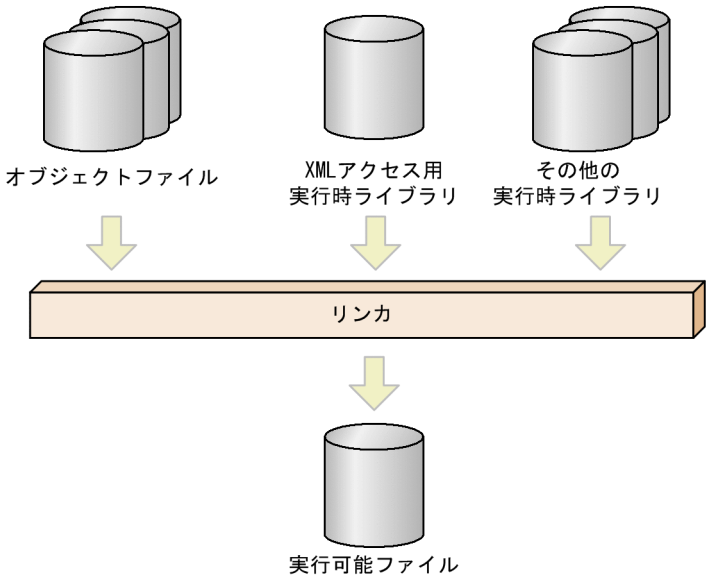
注意事項

XML 連携機能が生成する COBOL ソースプログラムをコンパイルする場合，ccbl2002 コマンドに次のオプションを指定できません。指定した場合，動作は保証しません。

-SimMain -Bin1Byte -EquivRule,NotAny -EquivRule,NotExtend -EquivRule,StdCode -V3Rec,Fixed -V3Rec,Variable -V3RecFCSpace -JPN,Alnum -JPN,V3JPN -JPN,V3JPNSpace -CompatiV3 -VOSCBL,OccursKey -VOSCBL,ReportControl -VOSCBL,DataComm -DigitsTrunc -V3Spec -V3Spec,CopyEased -LowerAsUpper

6.1.2 リンケージ

オブジェクトファイルと XML 連携機能の実行時ライブラリを，リンカ（AIX の場合は xlC コマンド，Linux の場合は g++コマンド）でリンクします。なお，Linux(x86)の場合，x64 向け Linux 環境では，32bit アプリケーションとしてリンクするために，-m32 オプションの指定が必要です。



(1) 一般的なリンケージ

指定例

(AIX(32)の場合)

```
xlC -o mainprog mainprog.o subprog.o
    -L/opt/HILNGcbl2k/lib
    -lcblxmlrt -lcbl2k -lcbl2kml -lm
```

(AIX(64)の場合)

```
xlC -q64 -o mainprog mainprog.o subprog.o
    -L/opt/HILNGcbl2k64/lib
    -lcblxmlrt64 -lcbl2k64 -lcbl2kml64 -lm
```

(Linux(x86)の場合)

```
g++ -o mainprog mainprog.o subprog.o
    -lcblxmlrt -lcbl2k -lcbl2kml -lm -ldl
```

(Linux(x86)の場合（x64 向け Linux 環境))

```
g++ -m32 -o mainprog mainprog.o subprog.o
    -lcblxmlrt -lcbl2k -lcbl2kml -lm -ldl
```

(Linux(x64)の場合)

```
g++ -o mainprog mainprog.o subprog.o
    -lcblxmlrt -lcbl2k -lcbl2kml -lm -ldl
```

引数

-o mainprog

作成する実行可能ファイルの名称です。

-lcblxmlrt

XML アクセス用実行時ライブラリのリンケージ指定です。

-lcbl2k -lcbl2kml

COBOL2002 実行時ライブラリのリンケージ指定です。

-lm

数学ライブラリのリンケージ指定です。

(2) テストデバッグ機能を使用する場合のリンケージ

-TDInf，-CVInf，-TestCmd,Full，または-TestCmd,Sim オプションを指定して作成したオブジェクトファイルをリンケージするとき，AIX では，COBOL2002 テストデバッガのライブラリを指定しないでください。また，Linux では，COBOL2002 テストデバッガのライブラリを指定しないで，「-ldl」オプションを指定してください。

指定例

(AIX(32)の場合)

```
xlC -o mainprog mainprog.o subprog.o
    -L/opt/HILNGcbl2k/lib
    -lcblxmlrt -lcbl2k -lcbl2kml -lm
```

(AIX(64)の場合)

```
xlC -q64 -o mainprog mainprog.o subprog.o
    -L/opt/HILNGcbl2k64/lib
    -lcblxmlrt64 -lcbl2k64 -lcbl2kml64 -lm
```

(Linux(x86)の場合)

```
g++ -o mainprog mainprog.o subprog.o
    -lcblxmlrt -lcbl2k -lcbl2kml -lm -ldl
```

(Linux(x86)の場合（x64 向け Linux 環境))

```
g++ -m32 -o mainprog mainprog.o subprog.o
    -lcblxmlrt -lcbl2k -lcbl2kml -lm -ldl
```

(Linux(x64)の場合)

```
g++ -o mainprog mainprog.o subprog.o
    -lcblxmlrt -lcbl2k -lcbl2kml -lm -ldl
```

6.1.3 マルチスレッドに対応した COBOL プログラムの作成

マルチスレッド対応 COBOL プログラムをコンパイルする場合は，-MultiThread オプションを指定します。

指定するライブラリ

(AIX(32)の場合)

COBOL2002 スレッド関数インターフェースライブラリ「-lcbl2kmp」，スレッド関数ライブラリ「-lpthreads」，およびスレッド対応システムライブラリ「-lC_r」

(AIX(64)の場合)

COBOL2002 スレッド関数インターフェースライブラリ「-lcbl2kmp64」，スレッド関数ライブラリ「-lpthreads」，およびスレッド対応システムライブラリ「-lC_r」

(Linux の場合)

COBOL2002 スレッド関数インターフェースライブラリ「-lcbl2kmp」，およびスレッド関数ライブラリ「-lpthread」

実行可能プログラムの作成方法

(AIX(32)の場合)

```
ccbl2002 -Compile,NoLink -MultiThread subprog.cbl
xlC_r -o mainprog mainprog.o subprog.o
    -L/opt/HILNGcbl2k/lib -L/opt/HILNGcbl2k/lib/cblxml
    -lcblxmlrt -lcbl2k -lcbl2kml -lcbl2kmp -lpthreads -lC_r -lm
```

(AIX(64)の場合)

```
ccbl2002 -Compile,NoLink -MultiThread subprog.cbl
xlC_r -q64 -o mainprog mainprog.o subprog.o
    -L/opt/HILNGcbl2k64/lib -L/opt/HILNGcbl2k64/lib/cblxml
    -lcblxmlrt64 -lcbl2k64 -lcbl2kml64 -lcbl2kmp64 -lpthreads -lC_r
    -lm
```

(Linux(x86)の場合)

```
ccbl2002 -Compile,NoLink -MultiThread -UniObjGen subprog.cbl
g++ -o mainprog mainprog.o subprog.o
    -L/opt/HILNGcbl2k/lib
    -lcblxmlrt -lcbl2k -lcbl2kml -lcbl2kmp -lpthread -lm
```

(Linux(x86)の場合（x64 向け Linux 環境))

```
ccbl2002 -Compile,NoLink -MultiThread -UniObjGen subprog.cbl
g++ -m32 -o mainprog mainprog.o subprog.o
    -L/opt/HILNGcbl2k/lib
    -lcblxmlrt -lcbl2k -lcbl2kml -lcbl2kmp -lpthread -lm
```

(Linux(x64)の場合)

```
ccbl2002 -Compile,NoLink -MultiThread -UniObjGen subprog.cbl
g++ -o mainprog mainprog.o subprog.o
    -L/opt/HILNGcbl2k64/lib
    -lcblxmlrt -lcbl2k -lcbl2kml -lcbl2kmp -lpthread -lm
```

注意事項

- XML ドキュメントの出力では排他制御しません。そのため，マルチスレッドでの動作時に，複数のスレッドから同じ XML ドキュメントのファイルや同じ XML ドキュメントのバッファへは，出力できません。

6.1.4 ダイナミックリンクに対応した COBOL プログラムの作成

COBOL2002 のダイナミックリンク機能を使用する場合，次のように実行可能ファイル，および副プログラムを作成してください。

(1) 実行可能ファイル（主プログラム）の作成方法

(AIX(32)の場合)

```
ccbl2002 -Compile,NoLink -DynamicLink,Call -Main,System mainprog.cbl
xlC -o mainprog mainprog.o -L/opt/HILNGcbl2k/lib
    -L/opt/HILNGcbl2k/lib/cblxml -lcblxmlrt -lcbl2k -lcbl2kml
    -lm
```

(AIX(64)の場合)

```
ccbl2002 -Compile,NoLink -DynamicLink,Call -Main,System mainprog.cbl
xlC -q64 -o mainprog mainprog.o -L/opt/HILNGcbl2k64/lib
    -L/opt/HILNGcbl2k64/lib/cblxml -lcblxmlrt64 -lcbl2k64 -lcbl2kml64
    -lm
```

(Linux(x86)の場合)

```
ccbl2002 -Compile,NoLink -DynamicLink,Call -UniObjGen
         -Main,System mainprog.cbl
g++ -o mainprog mainprog.o
    -L/opt/HILNGcbl2k/lib -lcblxmlrt -lcbl2k -lcbl2kml
    -lm -ldl
```

(Linux(x86)の場合（x64 向け Linux 環境))

```
ccbl2002 -Compile,NoLink -DynamicLink,Call -UniObjGen
         -Main,System mainprog.cbl
g++ -m32 -o mainprog mainprog.o
    -L/opt/HILNGcbl2k/lib -lcblxmlrt -lcbl2k -lcbl2kml
    -lm -ldl
```

(Linux(x64)の場合)

```
ccbl2002 -Compile,NoLink -DynamicLink,Call -UniObjGen
         -Main,System mainprog.cbl
g++ -o mainprog mainprog.o
    -L/opt/HILNGcbl2k64/lib -lcblxmlrt -lcbl2k -lcbl2kml
    -lm -ldl
```

(2) 副プログラムの作成方法

(AIX(32)の場合)

```
ccbl2002 -PIC,Std subprog.cbl
xlC -o libsubprog.a subprog.o -qmkshrobj
    -L/opt/HILNGcbl2k/lib -lcbl2k -lcbl2kml -lcblxmlrt -lm
```

(AIX(64)の場合)

```
ccbl2002 -PIC,Std subprog.cbl
xlC -q64 -o libsubprog.a subprog.o -qmkshrobj
    -L/opt/HILNGcbl2k64/lib -lcbl2k64 -lcbl2kml64 -lcblxmlrt64 -lm
```

(Linux(x86)の場合)

```
ccbl2002 -PIC,Std -UniObjGen subprog.cbl
g++ -shared -o libsubprog.so subprog.o
    -L/opt/HILNGcbl2k/lib -Bstatic -lcbl2kml
```

(Linux(x86)の場合（x64 向け Linux 環境))

```
ccbl2002 -PIC,Std -UniObjGen subprog.cbl
g++ -m32 -shared -o libsubprog.so subprog.o
    -L/opt/HILNGcbl2k/lib -Bstatic -lcbl2kml
```

(Linux(x64)の場合)

```
ccbl2002 -PIC,Std -UniObjGen subprog.cbl
g++ -shared -o libsubprog.so subprog.o
    -L/opt/HILNGcbl2k64/lib -Bstatic -lcbl2kml
```

注意事項

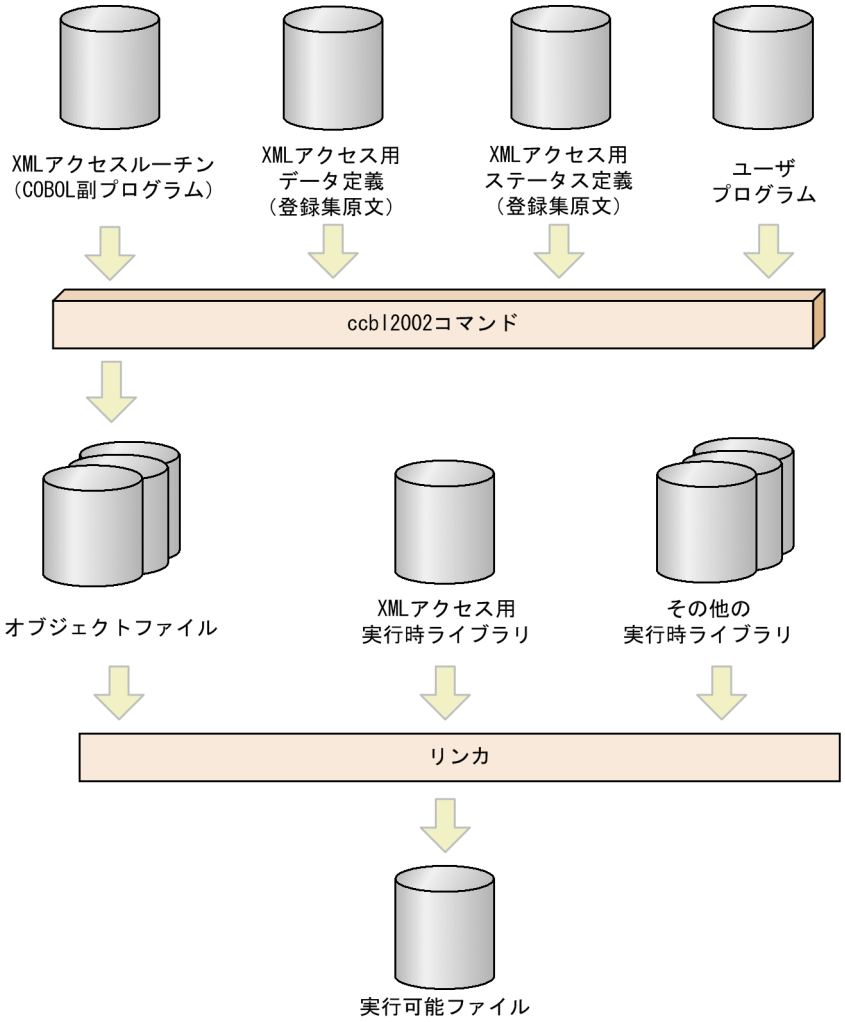
- プログラムの実行前に，COBOL2002 の環境変数 CBLLPATH または環境変数 CBLLSLIB に検索する共用ライブラリのディレクトリ名またはライブラリ名を設定しておく必要があります。Linux では，環境変数 CBLLSLIB にライブラリ名を設定するとき，環境変数 LD_LIBRARY_PATH に検索する共用ライブラリのディレクトリ名を設定しておく必要があります。

6.2 Windows で作成した COBOL プログラムの Windows でのコンパイルとリンケージ

Windows で作成した COBOL プログラムを，Windows でコンパイル，リンケージする方法について説明します。

6.2.1 コンパイルとリンケージ

ccbl2002 コマンドでコンパイル，リンケージを実行して，実行可能ファイル（.exe）を生成します。



(1) 環境変数の設定

XML 対応 COBOL プログラムをコンパイルする場合は，次に示す環境変数を設定しておく必要があります。

- システム環境変数 PATH
  ccbl2002 コマンドの格納パスを設定します。次の値を指定してください。
  COBOL2002のインストールフォルダ¥bin

- COBOL の環境変数 CBLLIB

  XML ステータス定義（登録集原文 CBLXMLRC.cbl）の格納パスを設定します。次の値を指定してください。

  COBOL2002のインストールフォルダ¥copy

  XML ステータス定義の詳細については，「4.4　XML アクセス用ステータス定義」を参照してください。

（例）

COBOL の環境変数 CBLLIB の設定

```
set CBLLIB=COBOL2002インストールフォルダ¥copy
```

(2)　コンパイルとリンケージの指定例

cblxml コマンドが生成した COBOL 原始プログラムをコンパイルしたオブジェクトファイルでリンケージする場合，ccbl2002 コマンドに XML アクセス用実行時ライブラリ（cblxmlrt.lib）を指定します。コンパイルとリンケージの指定例を次に示します。

指定例

```
ccbl2002 -OutputFile mainprog.exe mainprog.cbl subprog.cbl
         cblxmlrt.lib
```

mainprog.cbl

作成した COBOL 主プログラムファイルの名称です。

subprog.cbl

cblxml コマンドによって生成された COBOL アクセスルーチンの副プログラムファイルの名称です。

cblxmlrt.lib

XML アクセス用実行時ライブラリのリンケージ指定です。

注意事項

XML 連携機能が生成する COBOL 原始プログラムをコンパイルする場合，ccbl2002 コマンドには次のオプションを指定できません。オプションを指定した場合，動作は保証しません。

-SimMain -Bin1Byte -EquivRule,NotAny -EquivRule,NotExtend -EquivRule,StdCode -V3Rec,Fixed -V3Rec,Variable -V3RecFCSpace -JPN,Alnum -JPN,V3JPN -JPN,V3JPNSpace -CompatiV3 -VOSCBL,OccursKey -VOSCBL,ReportControl -VOSCBL,DataComm -VOSCBL,RedefinesData -VOSCBL,AssignDataToDevice -VOSCBL,EvaluateWhenOther -DigitsTrunc -V3Spec -V3Spec,CopyEased -LowerAsUpper

6.2.2　マルチスレッドに対応した COBOL プログラムの作成

マルチスレッド対応 COBOL プログラムをコンパイルする場合は，-MultiThread オプションを指定します。ccbl2002 コマンドでリンクをします。

実行可能プログラムの作成方法

```
ccbl2002 -Compile,NoLink -MultiThread subprog.cbl
ccbl2002 -OutputFile mainprog.exe mainprog.obj subprog.obj cblxmlrt.lib
```

注意事項

XML ドキュメントの出力での排他制御はしていません。そのため，マルチスレッドでの動作時に，複数のスレッドから同じ XML ドキュメントのファイルや同じ XML ドキュメントのバッファへは，出力できません。

6.2.3 ダイナミックリンクに対応した COBOL プログラムの作成

COBOL2002 のダイナミックリンク機能を使用する場合，静的なリンクのときは，副プログラムを先にコンパイルし，作成されたインポートライブラリを実行可能ファイル作成時に指定してください。

動的なリンクのときは，インポートライブラリを実行可能プログラム作成時に指定しないでください。

定数指定の CALL 文の呼び出しを動的にリンクするときは，-DynamicLink,Call オプションを指定します。

(1) 副プログラムの作成方法

Windows(x86)の場合

```
ccbl2002 -OutputFile subprog.dll -Dll,Stdcall subprog.cbl cblxmlrt.lib
```

Windows(x64)の場合

```
ccbl2002 -OutputFile subprog.dll -Dll subprog.cbl cblxmlrt.lib
```

(2) 実行可能ファイル（主プログラム）の作成方法

(a) 静的なリンクの場合

Windows(x86)の場合

```
ccbl2002 -OutputFile mainprog.exe -Stdcall -Main,System mainprog.cbl subprog.lib
```

Windows(x64)の場合

```
ccbl2002 -OutputFile mainprog.exe -Main,System mainprog.cbl subprog.lib
```

注意事項

COBOL2002 の-StdCall オプションで読み込む.cbw ファイル（stdcall 呼び出し指示ファイル）に XML アクセスルーチンを指定する場合，XML アクセスルーチン名に含まれる特殊文字は，COBOL2002 のプログラム名の変換規則に従い，変換して指定してください。例えば，ハイフン（-）は下線（_）に変換した XML アクセスルーチン名を指定してください。

(b) 動的なリンク（定数指定の CALL 文での呼び出し）の場合

Windows(x86)の場合

```
ccbl2002 -DynamicLink,Call -OutputFile mainprog.exe -Stdcall -Main,System mainprog.cbl
```

Windows(x64)の場合

```
ccbl2002 -DynamicLink,Call -OutputFile mainprog.exe -Main,System mainprog.cbl
```

注意事項

- 動的なリンクを使用する場合，実行時に呼び出し先プログラムの検索方法などを指定してください。動的なリンクについては，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。
- cblxml コマンドで生成した XML アクセスルーチン（COBOL 副プログラム）を，-DynamicLink,Call コンパイラオプションを指定してコンパイルしたとき，プログラム実行時に環境変数 CBLLDLL に cblxmlrt.dll を指定する必要があります。

7 実行

この章では，XML 対応 COBOL プログラムの実行方法，実行時の動作で注意が必要な点，および実行時に XML アクセスルーチンが返すステータスについて説明します。また，実行時のメモリ所要量について説明します。

7.1.1　XML 対応 COBOL プログラム実行時に必要な環境変数の設定

XML 対応 COBOL プログラムを実行する場合は，次に示す環境変数を設定しておく必要があります。

（UNIX の場合）

- システム環境変数 PATH
  cblxml コマンドの格納パスを設定します。次の値を指定してください。
  COBOL2002のインストールディレクトリ/bin
- システム環境変数 LIBPATH（AIX の場合）
  共用ライブラリの格納パスを設定します。次の値を指定してください。
  COBOL2002のインストールディレクトリ/lib
- システム環境変数 LD_LIBRARY_PATH（Linux の場合）
  共用ライブラリの格納パスを設定します。次の値を指定してください。
  COBOL2002のインストールディレクトリ/lib:COBOL2002のインストールディレクトリ/lib/cblxml
- システム環境変数 NLSPATH
  XML パーサが出力するメッセージのパスを設定します。次の値を指定してください。
  COBOL2002のインストールディレクトリ/lib/cblxml/cat/%L/%N

（指定例）

sh（B シェル）の場合

```
NLSPATH=/opt/HILNGcbl2k/lib/cblxml/cat/%L/%N
export NLSPATH
```

- システム環境変数 LANG
  メッセージの言語種別を設定します。次の値を指定してください。
  AIX の場合
  Ja_JP
  ja_JP
  Linux の場合
  ja_JP.UTF-8
  文字コードについては，「付録 G.2　文字コード」を参照してください。

注

COBOL2002 のインストールディレクトリは，OS によって異なります。
AIX(32)，Linux(x86)の場合
/opt/HILNGcbl2k

AIX(64)，Linux(x64)の場合

/opt/HILNGcbl2k64

（Windows の場合）

- システム環境変数 PATH

  DLL ファイルの格納パスを設定します。次の値を指定してください。

  COBOL2002のインストールフォルダ¥bin

（指定例）

```
set PATH=COBOL2002インストールフォルダ¥bin
```

7.1.2 実行時に指定できる環境変数

実行時に指定できる環境変数を次に示します。

- 環境変数 CBLXML_CHKUFLOW

  小数点以下のけた落ちを判定します。詳細については「9.4 小数点以下のけた落ちを判定する機能」を参照してください。

- 環境変数 CBLXML_ENTITYLIMIT

  XML ドキュメントの入力時に，エンティティ参照回数を制限します。詳細については「付録 E.8(2) エンティティ参照の参照回数を制限する」を参照してください。

- 環境変数 CBLXML_PARSE_NOXXE

  XML ドキュメントの入力時に，外部エンティティ参照を展開しないで解析するかどうかを指定します。詳細については「付録 E.8(1) 外部エンティティ参照の展開を無効にする」を参照してください。

7.1.3 XML 対応 COBOL プログラムの実行

XML 対応 COBOL プログラムは，通常の COBOL プログラムと同様にコマンドラインから実行します。

（例）

実行可能ファイル wsample を実行する例を次に示します。

（UNIX の場合）

```
wsample
```

（Windows の場合）

```
wsample.exe
```

7.2　実行時の動作に関する注意事項

XML 対応 COBOL プログラムの実行時の動作で，注意が必要な点について説明します。

7.2.1　省略可能な要素へのアクセス

XML では，省略可能な要素を定義できます。省略可能な要素を定義するには，要素の名称の最後に（?）を追加します。

例えば，次のような DTD が定義された XML ドキュメントでは，要素「group1」の下位に要素「item1」は必要ですが，要素「item2」はあってもなくてもかまいません。

（省略可能な要素を含む DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
 <!DOCTYPE table [
  <!ELEMENT table (group1)>
  <!ELEMENT group1 (item1, item2?)>
  <!ELEMENT item1  (#PCDATA)>
  <!ELEMENT item2  (#PCDATA)>
]>
<table/>
```

省略可能な要素にアクセスする場合，DDF は次のようにすべての要素にアクセスできるように記述する必要があります。

（省略可能な要素にアクセスする DDF の例）

- emptyValue 属性および accessInfo 属性の指定がない場合

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE" >
 <BaseElement elemName="group1">
  <Group elemName="group1">
   <Item elemName="item1" type="alphanumeric"
         size="30"/>
   <Item elemName="item2" type="alphanumeric"
         size="30"/>
  </Group>
 </BaseElement>
</Interface>
```

- emptyValue 属性の指定がある場合

```
<?xml version="1.0" encoding="Shift_JIS"?>
 <Interface interfaceName="EXAMPLE" >
 <BaseElement elemName="group1">
  <Group elemName="group1">
   <Item elemName="item1" type="alphanumeric"
         size="30"/>
   <Item elemName="item2" type="alphanumeric"
```

```
        size="30" emptyValue="EP"/>
  </Group> </BaseElement></Interface>
```

- accessInfo 属性の指定がある場合

```
<?xml version="1.0" encoding="Shift_JIS"?>
 <Interface interfaceName="EXAMPLE" accessInfo="yes">
 <BaseElement elemName="group1">
  <Group elemName="group1">
   <Item elemName="item1" type="alphanumeric"
         size="30"/>
   <Item elemName="item2" type="alphanumeric"
         size="30"/>
  </Group>
 </BaseElement>
</Interface>
```

また，省略可能な要素は，XML 対応 COBOL プログラムでは次のように扱われます。

(1) 省略可能な要素の入力処理

省略可能な要素を省略した XML ドキュメントを入力した場合，Item 要素に指定した属性によって，対応する COBOL データ項目に入力される値，およびアクセス情報フラグに設定される値が異なります。省略可能な要素を持つ XML ドキュメントを入力した場合の，Item 要素に指定した属性と COBOL データ項目に入力される値の関係を表 7-1 に示します。

表 7-1　省略可能な要素の入力処理

省略可能な要素に対応する Item 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			入力 XML ドキュメントの省略可能な要素	COBOL データ項目（アクセス情報フラグ）の入力値
emptyValue	emptyContentValue	accessInfo="yes"		
○	×	×	要素値あり	要素値
			空要素	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1
			要素なし	emptyValue 属性の指定値※2
○	○	×	要素値あり	要素値
			空要素	emptyContentValue 属性の指定値
			要素なし	emptyValue 属性の指定値※2
×	×	○	要素あり	要素値（CBLXML-FLAG-OK）

省略可能な要素に対応する Item 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			入力 XML ドキュメントの省略可能な要素	COBOL データ項目（アクセス情報フラグ）の入力値
emptyValue	emptyContentValue	accessInfo="yes"		
			空要素	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1 （CBLXML-FLAG-EMPTY）
			要素なし	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1 （CBLXML-FLAG-MISSING）

（凡例）

○：指定あり

×：指定なし

注

- emptyValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyValue 属性の指定は無効となります。
- emptyContentValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyContentValue 属性の指定は無効となります。

注※1

type 属性の指定が"alphanumeric"または"national"の場合は SPACE が設定されます。

type 属性の指定が"numeric"，"packed"，"binary"，"float"，または"double"の場合は ZERO が設定されます。

注※2

emptyValue 属性の指定がない場合，emptyValue 属性の省略値が設定されます。

(2) 省略可能な要素の出力処理

省略可能な要素に対応した COBOL データ項目，アクセス情報フラグに特定の値を設定することで，省略可能な要素を XML ドキュメントに出力します。Item 要素の属性の指定と COBOL データ項目，アクセス情報フラグに設定する値によって出力する XML ドキュメントの省略可能な要素の関係を表 7-2 に示します。

表 7-2　省略可能な要素の出力処理

省略可能な要素に対応する Item 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			COBOL データ項目（アクセス情報フラグ）の値	出力 XML ドキュメント
emptyValue	emptyContentValue	accessInfo="yes"		
○	×	×	emptyValue 属性の指定値以外の値	要素を出力する
			emptyValue 属性の指定値※	要素を出力しない
○	○	×	emptyValue 属性，emptyContentValue 属性の指定値以外の値	要素を出力する
			emptyContentValue 属性の指定値	空要素を出力する
			emptyValue 属性の指定値※	要素を出力しない
			emptyContentValue 属性と emptyValue 属性が同じ指定値	要素を出力しない
×	×	○	値（CBLXML-FLAG-OK）	要素を出力する
			値（CBLXML-FLAG-EMPTY）	空要素を出力する
			値（CBLXML-FLAG-MISSING）	要素を出力しない

（凡例）

○：指定あり

×：指定なし

注

- emptyValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyValue 属性の指定は無効となります。
- emptyContentValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyContentValue 属性の指定は無効となります。

注※

emptyValue 属性の指定がない場合，emptyValue 属性の省略値を設定します。

7.2.2　省略可能な選択要素

省略可能な選択要素は，次の設定で XML ドキュメントの出力時に削除できます。

DTD の条件	設定方法
accessInfo="no"	すべての選択要素に emptyValue 値を設定する。
accessInfo="yes"	すべての選択要素に CBLXML-FLAG-MISSING を設定する。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root   (group1?)>
  <!ELEMENT group1 (item1 | item2)>
  <!ELEMENT item1  (#PCDATA)>
  <!ELEMENT item2  (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="root">
  <Group elemName="root">
   <Group elemName="group1">
    <Item elemName="item1" size="10" emptyValue="A"/>
    <Item elemName="item2" size="10" emptyValue="B"/>
   </Group>
  </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 root.
02 group1.
 03 item1 PIC X(10).
 03 item2 PIC X(10).
```

(COBOL プログラムのコーディング例)

```
       :
    MOVE 'A' TO item1.         …1.
    MOVE 'B' TO item2.         …2.
    CALL 'CBLXML-WR-EXAMPLE-root'
        USING XML-POINTER   root
        RETURNING CBLXML-RETURN-CODE.
      :
```

(説明)

1. item1 を出力しないように emptyValue 値を設定してください。
2. item2 を出力しないように emptyValue 値を設定してください。

7.2.3　＋繰り返しを持つ選択要素

選択する要素に＋繰り返し付きの要素がある場合，次の設定で＋繰り返し付きの要素を出力しないようにできます。

DTD の条件	設定方法
countVar 指定なし countVar="yes"	＋繰り返し付き要素の countVar の値に 0 を設定する。
countVar="no" accessInfo="no"	すべての繰り返し項目に emptyValue 値を設定する。
countVar="no" accessInfo="yes"	すべての繰り返し項目に CBLXML-FLAG-MISSING を設定する。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (item1 | item2+ )>
  <!ELEMENT item1  (#PCDATA)>
  <!ELEMENT item2  (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="root">
  <Group elemName="root">
   <Item elemName="item1" size="10"/>
   <Array max="2">
    <Item elemName="item2" size="10"/>
   </Array>
  </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 root.
 02 item1 PIC X(10).
 02 item2 PIC X(10) OCCURS 2.
 02 item2-COUNT PIC 9(9) USAGE COMP.
```

(COBOL プログラムのコーディング例)

```
       :
    MOVE 0 TO item2-COUNT.        …1.
    MOVE 'data1' TO item1.
    CALL 'CBLXML-WR-EXAMPLE-root'
        USING XML-POINTER   root
```

```
        RETURNING CBLXML-RETURN-CODE.
      :
```

（説明）

1. item2 を出力しないように繰り返し数 0 を設定してください。

7.2.4　選択要素へのアクセス

XML では，選択要素（複数の項目のどれかが存在する要素）が定義できます。例えば，次のような DTD が定義された XML ドキュメントでは，aaa 要素の下位に xxx，yyy，zzz のどれかの要素が存在する可能性があります。

（選択要素を含む DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (aaa)*>
  <!ELEMENT aaa (xxx | yyy | zzz)>
  <!ELEMENT xxx (#PCDATA)>
  <!ELEMENT yyy (#PCDATA)>
  <!ELEMENT zzz (#PCDATA)>
]>
<table/>
```

選択要素にアクセスする場合，DDF は次のようにすべての要素にアクセスできるように記述する必要があります。

（選択要素にアクセスする DDF の例）

- accessInfo 属性の指定がない場合

```
<?xml version="1.0" encoding="Shift_JIS"?>
 <Interface interfaceName="EXAMPLE">
 <BaseElement elemName="aaa">
  <Group cobName="AAA">
   <Item elemName="xxx" type="alphanumeric"
         size="30"/>
   <Item elemName="yyy" type="alphanumeric"
         size="30"/>
   <Item elemName="zzz" type="alphanumeric"
         size="30"/>
  </Group>
 </BaseElement>
</Interface>
```

- accessInfo 属性の指定がある場合

```
<?xml version="1.0" encoding="Shift_JIS"?>
 <Interface interfaceName="EXAMPLE" accessInfo="yes">
 <BaseElement elemName="aaa">
  <Group cobName="AAA">
   <Item elemName="xxx" type="alphanumeric"
```

```
        size="30"/>
   <Item elemName="yyy" type="alphanumeric"
        size="30"/>
   <Item elemName="zzz" type="alphanumeric"
        size="30"/>
  </Group>
 </BaseElement>
</Interface>
```

また，選択要素は，XML 対応 COBOL プログラムでは次のように扱われます。

(1)　選択要素の入力処理

選択要素からの入力処理で，選択された要素以外の要素は，対応する COBOL データ項目に空白が設定されます。

入出力データ情報定義機能を使用する場合，値がある選択要素に対応する COBOL データ項目のアクセス情報フラグの CBLXML-FLAG-MISSING，CBLXML-FLAG-EMPTY，CBLXML-FLAG-INVAL-CHAR，および CBLXML-FLAG-OVERFLOW にすべて 0 が設定されます。それ以外の選択要素に対応する COBOL データ項目のアクセス情報フラグには，CBLXML-FLAG-MISSING が設定されます。

(2)　選択要素の出力処理

選択要素への出力処理では，出力したい要素に対応する COBOL データ項目だけにデータを格納してください。二つ以上の要素にデータを出力した場合や，選択要素のすべてに空白を格納した場合，XML アクセスルーチンからエラーのステータスが返されます。

入出力データ情報定義機能を使用する場合，出力したい要素に対応する COBOL データ項目のアクセス情報フラグだけに CBLXML-FLAG-OK を設定し，その他のアクセス情報フラグには CBLXML-FLAG-MISSING を設定しなければなりません。選択要素中の二つ以上の COBOL データ項目のアクセス情報フラグに CBLXML-FLAG-OK を設定した場合，CBLXML-WR-*Interface-BaseElement* アクセスルーチンの戻り値にエラーのステータスが返されます。また，選択要素に対応するすべての COBOL データ項目に対応するアクセス情報フラグに CBLXML-FLAG-MISSING を設定した場合も，CBLXML-WR-*Interface-BaseElement* アクセスルーチンの戻り値にエラーのステータスが返されます。

7.2.5　対応づけしない要素の扱い

DTD の要素のうち，COBOL データ項目に対応づけなかった要素の出力動作について説明します。

対応づけしない要素のうち 0 回以上の繰り返し要素（*）と省略可能な要素（?）は，BaseElement 要素の内側にある場合は出力されません。その他の要素の場合，空要素が出力されます。

また，BaseElement 要素の外側にある要素はすべて空要素が出力されます。BaseElement 要素の内側および外側にある対応づけしない要素の出力例を次に示します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [  <!ELEMENT root (group1, group2, group3)>
  <!ELEMENT group1 (item1a,  item1b, item1c*)>
  <!ELEMENT item1a  (#PCDATA)>
  <!ELEMENT item1b  (#PCDATA)>
  <!ELEMENT item1c  (#PCDATA)>
  <!ELEMENT group2 (item2a?, item2b+)>
  <!ELEMENT item2a  (#PCDATA)>
  <!ELEMENT item2b  (#PCDATA)>
  <!ELEMENT group3 (item3a | item3b)>
  <!ELEMENT item3a  (#PCDATA)>
  <!ELEMENT item3b  (#PCDATA)>]><root/>
```

item1a 要素を BaseElement 要素と対応づける場合の例を次に示します。item1a 要素に値を代入し，XML ドキュメントを出力します。この場合，BaseElement 要素の外側にある要素はすべて空要素として出力します。

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="item1a" cobName="BE">
    <Item elemName="item1a" type="alphanumeric" size="10"/>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 item1a PIC X(10).
```

(出力した XML ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
  <group1>
    <item1a>AAAAAAAAAA</item1a>
    <item1b/>
    <item1c/>
  </group1>
  <group2>
    <item2a/>
    <item2b/>
  </group2>
  <group3>
    <item3a/>
  </group3>
</root>
```

root 要素を BaseElement 要素と対応づける場合の例を次に示します。item1a 要素に値を代入し，XML ドキュメントを出力します。この場合，BaseElement 要素の内側にある要素で 0 回以上の繰り返し要素 (*) と省略可能な要素 (?) は出力しません。その他の要素は空要素を出力します。

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="root" cobName="BE">
    <Group elemName="root" cobName="BASE">
      <Group elemName="group1">
        <Item elemName="item1a" type="alphanumeric"
              size="10"/>
      </Group>
    </Group>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 BASE.
 02 group1.
  03 item1a PIC X(10).
```

(出力した XML ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
  <group1>
    <item1a>AAAAAAAAAA</item1a>
    <item1b/>
  </group1>
  <group2>
    <item2b/>
  </group2>
  <group3>
    <item3a/>
  </group3>
</root>
```

7.2.6 要素に囲まれた要素の扱い

DTD で定義された要素のうち，一部の要素だけを DDF で対応づけた場合，対応づけした要素以外は空要素として出力されます。また，対応づけた要素の親要素を省略した場合，親要素が補われて出力されます。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE table [
  <!ELEMENT table (row)+>
  <!ELEMENT row (name, address, grade, age)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (addr1, addr2, city, state, zip)>
  <!ELEMENT addr1 (#PCDATA)>
  <!ELEMENT addr2 (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT state (#PCDATA)>
  <!ELEMENT zip (#PCDATA)>
```

```
  <!ELEMENT grade (#PCDATA)>
  <!ELEMENT age (#PCDATA)>
]>
<table/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="row">
    <Group cobName="row">
      <Item elemName="name" cobName="full-name"
            type="alphanumeric" size="20"/>
      <Item elemName="age" type="numeric" size="3"/>
      <Item elemName="city" type="alphanumeric"
            size="30"/>
    </Group>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 row.
 02 full-name PIC X(20).
 02 age PIC 9(3).
 02 city PIC X(30).
```

(出力される XML ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<table>
  <row>
    <name>Mr. X</name>
    <address>
      <addr1/>
      <addr2/>
      <city>Tokyo</city>
      <state/>
      <zip/>
    </address>
    <grade/>
    <age>103</age>
  </row>
</table>
```

7.2.7 値の入力，出力の動作

XML ドキュメントと COBOL データ項目間のデータの読み込み，書き込みには，それぞれ次の規則が適用されます。

(1) 数値の場合

数値データの入力規則

- XML のデータに「E」が含まれる場合，数値を浮動小数点数とみなします。このとき，「E」の左側が仮数部，右側が指数部となります。また，「E」の右側がない場合は，指数を 0 とみなします。
- XML のすべてのデータが数値以外（スペース，タブおよび改行も含む）の場合，0 が読み込まれます。
- 入力する数値の中に数値以外（スペース，タブおよび改行も含む）がある場合，不当な数値であるため結果は不定となります。ただし，invalidCharValue 属性が指定されている場合，その値となります。

  （入力例）

  <Item>1△2</Item>

  注

  △は，数値の中に空白文字（スペース，タブおよび改行）がある場合を表します。
- 入力する数値の前後に空白文字（スペース，タブおよび改行）がある場合，空白文字（スペース，タブおよび改行）を無視して数値だけが読み込まれます。

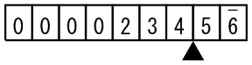
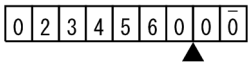
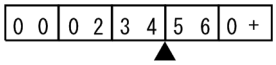
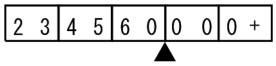
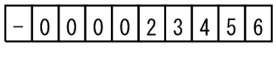
  （入力例）

  <Item>△12△</Item>

  注

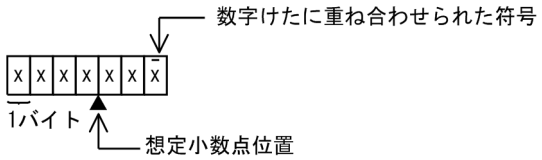
  △は，数値の前後に空白文字（スペース，タブおよび改行）がある場合を表します。
- 浮動小数点数の入力で負符号を持つ 0 を入力した場合，正の 0 として入力されます。

数値データの入力例

DDF の定義例	COBOL データ項目	XML データ	COBOL に 格納される値
type="numeric" sign="signed" size="9" fractionalDigits="2"	S9(7)V9(2)	-234.56	
		-234.56E3	
type="packed" sign="unsigned" size="9" fractionalDigits="3"	9(6)V9(3) USAGE PACKED-DECIMAL	-234.56	
		-234.56E3	
type="numeric" sign="leadingSeparate"	S9(9) SIGN LEADING SEPARATE	-23456	
type="binary" size="9"	9(9) USAGE COMP	-234.56	'+000000234'
		-234.56E3	'+000234560'
type="float"	COMP-1	12345.E4	' .12345000E 09'

DDF の定義例	COBOL データ項目	XML データ	COBOL に 格納される値
		234.56	' .23456000E 03'
type="double"	COMP-2	12345.E4	' .12345000000000000E 09'
		234.56	' .23456000000000000E 03'

（凡例）



数値データの出力規則

- DDF で，Item 要素の type 属性に"numeric"，"packed"，または"binary"を指定した値は，size 属性，sign 属性，fractionalDigits 属性，trim 属性に従って値が出力されます。これらの数値は，size 属性のけた数に従って数値の先頭部分に 0 が挿入されます。
- DDF で，Item 要素の type 属性に"float"，または"double"を指定した値は，次の例に示すような浮動小数点形式で出力されます。

  数値データの出力例

DDF の定義例	COBOL データ項目	COBOL データ項目の値	XML データに出力される値
type="numeric" sign="signed" size="9" fractionalDigits="2" trim="yes"	S9(7)V9(2)	-0.05	<nu1> -0.05 </nu1>
type="numeric" sign="signed" size="9" fractionalDigits="2" trim="yes"	S9(7)V9(2)	-123.45	<nu1> -123.45 </nu1>
type="numeric" sign="signed" size="9" fractionalDigits="2" trim="no"	S9(7)V9(2)	-123.45	<nu1> -0000123.45 </nu1>
type="numeric" sign="leadingSeparate"	S9(9) SIGN LEADING SEPARATE	-12345	<nu1> -12345 </nu1>

DDF の定義例	COBOL データ項目	COBOL データ項目の値	XML データに出力される値
type="packed" sign="unsigned" size="9" fractionalDigits="3" trim="no"	9(6)V9(3) USAGE PACKED-DECIMAL	123.45	<pa1> 000123.450 </pa1>
type="binary" size="9" trim="no"	9(9) USAGE COMP	123	<bi1> 0000123 </bi1>
type="float"	COMP-1	123.45E2	<fl1> 1.2345000E+04 </fl1>
type="double"	COMP-2	123.45E2	<do1> 1.2345000000000000E+04 </do1>

無限大（Infinity）と非数（Not-a-Number）の浮動小数点数の出力結果

- 無限大や非数の浮動小数点数を出力した場合，無限大や非数を示す文字列を出力します。無限大や非数を示す文字列は入力できないため，無限大や非数の浮動小数点数は出力しないでください。

(2)　文字列の場合

文字列データの入力規則

- DDF で，Item 要素の type 属性に"national"，または"alphanumeric"を指定した値を XML ドキュメントから入力する場合，入力した文字列の長さより COBOL データ項目の方が長いときは，余った領域に空白文字が埋められます。
- 入力する文字列中に不当な文字があった場合，動作は保証しません。
- type 属性に"national"を指定した値を入力した場合，データ中の 1 バイト文字は，すべて 2 バイト文字「＝」（げた記号）に置き換えられます。不当文字については「9.2 入出力時に不当な文字をチェックする機能」の「表 9-3 入力時にチェックする文字」を参照してください。
- 空要素を入力した場合，空白が格納されます。
- XML ドキュメントから入力した文字列がスペースまたはタブの場合，そのまま COBOL データ項目に読み込まれます。
- 改行コードは，UNIX の場合は，LF（0x0A）に置き換えられます。Windows の場合は，CR/LF(0x0D0A)に置き換えられます。

  文字列データの入力例

DDF の定義例	COBOL データ項目	XML データ	COBOL に 格納される値
type="alphanumeric" size="10"	X(10)	ABC	'ABC△△△△△△△'
		ABCDEFGHIJKL	'ABCDEFGHIJ'
type="national" size="10"	N(10)	あいう	'あいう▲▲▲▲▲▲▲'
		あいうえおかきくけこさし	'あいうえおかきくけこ'
		あ AB い	'あ == い▲▲▲▲▲▲'

（凡例）

△：半角空白文字

▲：全角空白文字

文字列データの出力規則

- DDF で，Item 要素の type 属性に"national"，または"alphanumeric"を指定した値を XML ドキュメントへ出力する場合，データの末尾の空白文字は削除されます。
- 出力する文字列中に不当な文字があった場合，動作は保証しません。
- 値がすべて空白のデータを出力した場合，空要素が出力されます。
- COBOL データ項目にタブ文字が含まれる場合，XML ドキュメントにそのまま出力されます。
- 改行コードは，UNIX の場合は，LF（0x0A）で出力されます。Windows の場合は，CR/LF(0x0D0A)で出力されます。

  文字列データの出力例

DDF の定義例	COBOL データ項目	COBOL データ項目の値	XML データに出力される値
type="alphanumeric" size="10" trim="yes"	X(10)	'ABC'	<an1>ABC</an1>
		SPACE	<an1></an1>
type="alphanumeric" size="10" trim="no"	X(10)	'ABC'	<an1>ABC△△△△△△△</an1>
		SPACE	<an1>△△△△△△△△△△</an1>
type="national" size="5" trim="yes"	N(5)	N'あいう'	<na1>あいう</na1>
		SPACE	<na1></na1>
type="national" size="5" trim="no"	N(5)	N'あいう'	<na1>あいう▲▲</na1>
		SPACE	<na1>▲▲▲▲▲</na1>

（凡例）

△：半角空白文字

▲：全角空白文字

7.2.8　入力 XML ドキュメントの妥当性チェック機能

CBLXML-OP-*Interface* または CBLXML-OB-*Interface* アクセスルーチンの XML-MODE 引数（アクセスモードを指定する引数）に"V"を指定すると，XML パーサのチェック機能によって，入力する XML ドキュメントの DTD と本文の妥当性がチェックされます。

妥当性チェックを受ける XML ドキュメントは，DTD がなければなりません。DTD がない場合は，妥当性チェックでエラーになります。妥当性チェックでエラーが検出された場合，CBLXML-OP-*Interface* または CBLXML-OB-*Interface* アクセスルーチンのステータスとして 110（CBLXML-XML-PARSE-FAIL）が返されます。

妥当性チェックの有無による動作の違いについては，「付録 E.4　使用できる解析モードによる動作の違い」を参照してください。

7.2.9　属性の入出力

属性の入出力動作は，属性のデフォルト指定によって異なります。

この節では，属性のデフォルト指定と属性の入出力動作の関係を表にして説明します。また，属性のデフォルト指定および属性値の入力動作についての注意事項を説明したあとに，属性の入力例と出力例を示します。

(1)　属性のデフォルト指定値と属性の入力

表 7-3　属性のデフォルト指定値と属性の入力動作

ATTLIST 属性のデフォルト指定	AttrItem 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			入力 XML ドキュメントの属性値	COBOL データ項目（アクセス情報フラグ）の入力値
	emptyValue	emptyContentValue	accessInfo="yes"		
#REQUIRED	○	×	×	属性あり（値あり）	属性値
				属性あり（値なし）	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1
				属性なし	emptyValue 属性の指定値※2
	○	○	×	属性あり（値あり）	属性値
				属性あり（値なし）	emptyContentValue 属性の指定値

ATTLIST 属性のデフォルト指定	AttrItem 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			入力 XML ドキュメントの属性値	COBOL データ項目（アクセス情報フラグ）の入力値
	emptyValue	emptyContentValue	accessInfo="yes"		
				属性なし	emptyValue 属性の指定値※2
	×	×	○	属性あり（値あり）	属性値 （CBLXML-FLAG-OK）
				属性あり（値なし）	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1 （CBLXML-FLAG-EMPTY）
				属性なし	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1 （CBLXML-FLAG-MISSING）
#IMPLIED	○	×	×	属性あり（値あり）	属性値
				属性あり（値なし）	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1
				属性なし	emptyValue 属性の指定値※2
	○	○	×	属性あり（値あり）	属性値
				属性あり（値なし）	emptyContentValue 属性の指定値
				属性なし	emptyValue 属性の指定値※2
	×	×	○	属性あり（値あり）	属性値 （CBLXML-FLAG-OK）
				属性あり（値なし）	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1 （CBLXML-FLAG-EMPTY）
				属性なし	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1 （CBLXML-FLAG-MISSING）
値	○	×	×	属性あり（値あり）	属性値
				属性あり（値なし）	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1
				属性なし	ATTLIST 属性に指定したデフォルト値
	○	○	×	属性あり（値あり）	属性値
				属性あり（値なし）	emptyContentValue 属性の指定値

ATTLIST 属性のデフォルト指定	AttrItem 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			入力 XML ドキュメントの属性値	COBOL データ項目（アクセス情報フラグ）の入力値
	emptyValue	emptyContentValue	accessInfo="yes"		
				属性なし	ATTLIST 属性に指定したデフォルト値
	×	×	○	属性あり（値あり）	属性値 （CBLXML-FLAG-OK）
				属性あり（値なし）	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1 （CBLXML-FLAG-EMPTY）
				属性なし	※3
#FIXED 値	○	×	×	属性あり（値あり）	ATTLIST 属性に指定したデフォルト値
				属性あり（値なし）	
				属性なし	
	○	○	×	属性あり（値あり）	ATTLIST 属性に指定したデフォルト値
				属性あり（値なし）	
				属性なし	
	×	×	○	属性あり（値あり）	ATTLIST 属性に指定したデフォルト値 （CBLXML-FLAG-OK）
				属性あり（値なし）	※4
				属性なし	※5

（凡例）

○：指定あり

×：指定なし

注

- emptyValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyValue 属性の指定は無効となります。
- emptyContentValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyContentValue 属性の指定は無効となります。

注※1

type 属性の指定が"alphanumeric"または"national"の場合は SPACE が設定されます。

type 属性の指定が"numeric"，"packed"，"binary"，"float"，または"double"の場合は ZERO が設定されます。

注※2

emptyValue 属性の指定がない場合，emptyValue 属性の省略値が設定されます。

注※3

DTD あり

ATTLIST 属性に指定したデフォルト値
（CBLXML-FLAG-OK）

DTD なし

cblxml コマンドに指定した XML ドキュメントの ATTLIST 属性に指定したデフォルト値
（CBLXML-FLAG-MISSING）

注※4

DTD あり

ATTLIST 属性に指定したデフォルト値
（CBLXML-FLAG-EMPTY）

DTD なし

cblxml コマンドに指定した XML ドキュメントの ATTLIST 属性に指定したデフォルト値
（CBLXML-FLAG-EMPTY）

注※5

DTD あり

ATTLIST 属性に指定したデフォルト値
（CBLXML-FLAG-OK）

DTD なし

cblxml コマンドに指定した XML ドキュメントの ATTLIST 属性に指定したデフォルト値
（CBLXML-FLAG-MISSING）

(2) 属性のデフォルト指定値と属性の出力

表 7-4 属性のデフォルト指定値と属性の出力動作

ATTLIST 属性のデフォルト指定	AttrItem 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			COBOL データ項目（アクセス情報フラグ）の出力値	出力 XML ドキュメントでの属性
	emptyValue	emptyContentValue	accessInfo="yes"		
#REQUIRED	○	×	×	emptyValue 属性の指定値以外の値	指定した値の属性を出力する
				emptyValue 属性の指定値※	空の属性を出力する

ATTLIST 属性のデフォルト指定	AttrItem 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			COBOL データ項目（アクセス情報フラグ）の出力値	出力 XML ドキュメントでの属性
	emptyValue	emptyContentValue	accessInfo="yes"		
	○	○	×	emptyValue 属性，emptyContentValue 属性の指定値以外の値	指定した値の属性値
				emptyContentValue 属性の指定値	空の属性値
				emptyValue 属性の指定値※	空の属性値
				emptyContentValue 属性と emptyValue 属性が同じ指定値	空の属性値
	×	×	○	値（アクセス情報フラグに CBLXML-FLAG-OK が設定される）	指定した値の属性値
				値（アクセス情報フラグに CBLXML-FLAG-EMPTY が設定される）	空の属性値
				値（アクセス情報フラグに CBLXML-FLAG-MISSING が設定される）	空の属性値
#IMPLIED	○	×	×	emptyValue 属性の指定値以外の値	指定した値の属性値
				emptyValue 属性の指定値※	出力しない
	○	○	×	emptyValue 属性，emptyContentValue 属性の指定値以外の値	指定した値の属性値
				emptyContentValue 属性の指定値	空の属性値
				emptyValue 属性の指定値	出力しない
				emptyContentValue 属性と emptyValue 属性が同じ指定値	出力しない
	×	×	○	値（アクセス情報フラグに CBLXML-	指定した値の属性値

ATTLIST 属性のデフォルト指定	AttrItem 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			COBOL データ項目（アクセス情報フラグ）の出力値	出力 XML ドキュメントでの属性
	emptyValue	emptyContentValue	accessInfo="yes"		
				FLAG-OK が設定される）	
				値（アクセス情報フラグに CBLXML-FLAG-EMPTY が設定される）	空の属性値
				値（アクセス情報フラグに CBLXML-FLAG-MISSING が設定される）	出力しない
値	○	×	×	emptyValue 属性の指定値以外の値	指定した値の属性値
				emptyValue 属性の指定値※	出力しない
	○	○	×	emptyValue 属性，emptyContentValue 属性の指定値以外の値	指定した値の属性値
				emptyContentValue 属性の指定値	空の属性値
				emptyValue 属性の指定値※	出力しない
				emptyContentValue 属性と emptyValue 属性が同じ指定値	出力しない
	×	×	○	値（アクセス情報フラグに CBLXML-FLAG-OK が設定される）	指定した値の属性値
				値（アクセス情報フラグに CBLXML-FLAG-EMPTY が設定される）	空の属性値
				値（アクセス情報フラグに CBLXML-FLAG-MISSING が設定される）	出力しない
#FIXED 値	○	×	×	emptyValue 属性の指定値以外の値	属性のデフォルト指定値

ATTLIST 属性のデフォルト指定	AttrItem 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			COBOL データ項目（アクセス情報フラグ）の出力値	出力 XML ドキュメントでの属性
	emptyValue	emptyContentValue	accessInfo="yes"		
				emptyValue 属性の指定値※	出力しない
	○	○	×	emptyValue 属性，emptyContentValue 属性の指定値以外の値	属性のデフォルト指定値
				emptyContentValue 属性の指定値	空の属性値
				emptyValue 属性の指定値※	出力しない
				emptyContentValue 属性と emptyValue 属性が同じ指定値	出力しない
	×	×	○	値（アクセス情報フラグに CBLXML-FLAG-OK が設定される）	属性のデフォルト指定値
				値（アクセス情報フラグに CBLXML-FLAG-EMPTY が設定される）	空の属性値
				値（アクセス情報フラグに CBLXML-FLAG-MISSING が設定される）	出力しない

（凡例）

○：指定あり

×：指定なし

注

- emptyValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyValue 属性の指定は無効となります。
- emptyContentValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyContentValue 属性の指定は無効となります。

注※

emptyValue 属性の指定がない場合，emptyValue 属性の省略値を設定します。

(3)　属性が DDF で対応づけされていない場合の属性の入出力

表 7-5　属性が DDF で対応づけされていない場合

ATTLIST 属性のデフォルト指定	XML ドキュメントの入力	XML ドキュメントの出力
#REQUIRED※	無視する。	空の属性値（attr = ""）を出力する。
#IMPLIED	無視する。	属性を出力しない。
値	無視する。	属性を出力しない。
#FIXED 値	無視する。	属性を出力しない。

注※

属性値のデフォルト指定が#REQUIRED の場合は，必ずその属性を DDF で対応づけてください。DDF で対応づけていない場合，適正な属性値が出力されません。

DDF で対応づけた属性のデフォルト指定に#REQUIRED を使用した場合の指定例については，「(4) 属性のデフォルト指定時の注意事項」を参照してください。

(4)　属性のデフォルト指定時の注意事項

属性のデフォルト指定を使用する場合の注意事項を次に示します。

- 属性値の入出力は，属性の型には依存しません。属性値として記述された文字列は属性の型には関係なく COBOL データ項目に入出力されます。
- ATTLIST 属性のデフォルト指定では，英数字項目および日本語項目の場合は 160 文字，数字項目の場合は 18 けた（符号，小数点は含まない）を指定してください。
- XML ドキュメント妥当性チェック機能で属性値が適正かどうかをチェックできるのは，属性値の入力時だけです。属性値の出力時に値が適正かどうかをチェックすることはできません。

  XML ドキュメント妥当性チェック機能については，「7.2.8　入力 XML ドキュメントの妥当性チェック機能」を参照してください。
- 属性のデフォルト指定に#REQUIRED を使用して属性値に空文字列が出力されると，出力された属性値が不正と判断される属性があります。常に適正な属性値を出力するために，#REQUIRED でデフォルト指定をする場合は，その属性を DDF で対応づける必要があります。

  DDF で対応づけた属性を#REQUIRED で指定した場合の指定例を次に示します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
 <!DOCTYPE table [
    <!ELEMENT table (item1, item2, item3, item4, item5)>
  <!ELEMENT item1 EMPTY>
  <!ELEMENT item2 EMPTY>
  <!ELEMENT item3 EMPTY>
  <!ELEMENT item4 EMPTY>
  <!ELEMENT item5 EMPTY>
  <!ATTLIST item1 attr11 CDATA #REQUIRED>
```

```
  <!ATTLIST item1 attr12 CDATA #IMPLIED>
  <!ATTLIST item1 attr13 CDATA "abc" >
  <!ATTLIST item1 attr14 CDATA #FIXED "def">
  <!ATTLIST item2 attr21 ID #IMPLIED >
  <!ATTLIST item2 attr22 IDREF #IMPLIED>
  <!ATTLIST item2 attr23 IDREFS #IMPLIED>
  <!ATTLIST item2 attr24 NMTOKEN #IMPLIED>
  <!ATTLIST item2 attr25 NMTOKENS #IMPLIED>
  <!ATTLIST item3 attr31 (blue|red|yellow) #IMPLIED>
  <!ATTLIST item4 attr41 CDATA #IMPLIED>
  <!ATTLIST item5 attr51 CDATA #REQUIRED>
  <!ATTLIST item5 attr52 CDATA #IMPLIED>
]>
<table/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="table">
    <Group cobName="grp1">
      <AttrItem elemName="item1" attrName="attr11"
      type="alphanumeric" size="10"/>
      <AttrItem elemName="item1" attrName="attr12"
      type="alphanumeric" emptyValue="xxx" size="10"/>
      <AttrItem elemName="item1" attrName="attr13"
      type="alphanumeric" emptyValue="yyy" size="10"/>
      <AttrItem elemName="item1" attrName="attr14"
      type="alphanumeric" size="10"/>
      <AttrItem elemName="item2" attrName="attr21"
      type="alphanumeric" size="10"/>
      <AttrItem elemName="item2" attrName="attr22"
      type="alphanumeric" size="10"/>
      <AttrItem elemName="item2" attrName="attr23"
      type="alphanumeric" size="10"/>
      <AttrItem elemName="item2" attrName="attr24"
      type="alphanumeric" size="10"/>
      <AttrItem elemName="item2" attrName="attr25"
      type="alphanumeric" size="10"/>
      <AttrItem elemName="item3" attrName="attr31"
      type="alphanumeric" size="10"/>
      <AttrItem elemName="item4" attrName="attr41"
      type="numeric" trim="no" size="4"/>
    </Group>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 grp1.
 02 item1-attr11 PIC X(10).
 02 item1-attr12 PIC X(10).
 02 item1-attr13 PIC X(10).
 02 item1-attr14 PIC X(10).
 02 item2-attr21 PIC X(10).
 02 item2-attr22 PIC X(10).
 02 item2-attr23 PIC X(10).
 02 item2-attr24 PIC X(10).
```

```
 02 item2-attr25 PIC X(10).
 02 item3-attr31 PIC X(10).
 02 item4-attr41 PIC 9(4).
```

(5) 属性値の入力動作についての注意事項

- type 属性に"alphanumeric"を指定し，入力する値の空白文字（スペース，タブおよび改行）はスペース(X'20')に置き換えて COBOL データ項目に読み込まれます。
- type 属性に"numeric"，"packed"，"binary"，"float"，"double"を指定し，入力する数値の前後に空白文字（スペース，タブおよび改行）がある場合，空白文字（スペース，タブおよび改行）を無視して数値だけが読み込まれます。

  （入力例）

  <Item>△12△</Item>

  注

  △は，数値の前後に空白文字（スペース，タブおよび改行）がある場合を表します。
- type 属性に"numeric"，"packed"，"binary"，"float"，"double"を指定し，入力する数値の中に空白文字（スペース，タブおよび改行）がある場合，不当な数値であるため結果は不定となります。

  （入力例）

  <Item>1△2</Item>

  注

  △は，数値の中に空白文字（スペース，タブおよび改行）がある場合を表します。

(6) 属性の入力例

次に示す XML ドキュメントを入力した場合に COBOL データ項目に格納される値を表 7-6 に示します。

（入力ドキュメント）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<table>
 <item1 attr11="aaa" attr14="ddd"/>
 <item2 attr21="id1" attr22="id1" attr23="id1 id1"
        attr24="1cm" attr25="1cm 2cm"/>
 <item3 attr31="red"/>
 <item4 attr41="nan"/>
 <item5 attr51="foo"/>
</table>
```

表 7-6　COBOL データ項目に格納される値

COBOL データ項目	値	備考
attr11	aaa	−
attr12	xxx	属性が存在しないため，emptyValue で指定した値が格納される。
attr13	abc	属性が存在しないため，DTD のデフォルト値が格納される。

COBOL データ項目	値	備考
attr14	def	#FIXED が指定されているため，ドキュメント上の属性値に関係なく DTD のデフォルト値が格納される。
attr21	id1	−
attr22	id1	−
attr23	id1△id1	−
attr24	1cm	−
attr25	1cm△2cm	−
attr31	red	−
attr41	0000	数字項目に文字列が入力されたため，0 が格納される。

（凡例）

△：半角空白文字

(7)　属性の出力例

表 7-7　COBOL データ項目に設定した出力値

COBOL データ項目	値
attr11	aaa
attr12	xxx
attr13	zzz
attr14	bbb
attr21	id1
attr22	id1
attr23	id1△id1
attr24	1cm
attr25	1cm△2cm
attr31	blue
attr41	0012

（凡例）

△：半角空白文字

表 7-7 に示した値を COBOL データ項目に設定した場合，次のような XML ドキュメントが出力されます。

(出力ドキュメント)

```
<table>
 <item1 attr11=”aaa”attr13=”zzz”attr14=”def”/>    …1.
        attr13=”zzz”
        attr14=”def”/>           …2.
 <item2 attr21=”id1”
        attr22=”id1”
        attr23=”id1 id1”
        attr24=”1cm”
        attr25=”1cm 2cm”
 <item3 attr31=”blue”/>
 <item4 attr41=”0012”/>           …3.
 <item5 attr51=””                 …4.
 />
                                  …5.
</table>
```

(説明)

1. COBOL データ項目の値が emptyValue の値に等しいため，属性は出力されません。
2. #FIXED が指定されているため，COBOL データ項目の値に関係なく"def"が出力されます。
3. trim="no"が指定されているため，前に 0 が付いて出力されます。
4. DDF で対応づけてはいませんが，#REQUIRED が指定されているため，空文字列が出力されます。
5. DDF で対応づけていないため，attr52 は出力されません。

7.2.10　EMPTY を指定した要素の入出力

ELEMENT 属性に EMPTY を指定した要素と Item 要素を対応づける場合の入出力を次に示します。

(1)　EMPTY を指定した要素の入力

ELEMENT 属性のキーワードに「EMPTY」を指定した場合の要素の入力内容を表 7-8 に示します。

表 7-8　EMPTY を指定した要素の入力

Item 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			入力 XML ドキュメント	COBOL データ項目（アクセス情報フラグ）の入力値
emptyValue	emptyContentValue	accessInfo="yes"		
○	×	×	空要素	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1
			要素なし	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1

Item 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			入力 XML ドキュメント	COBOL データ項目（アクセス情報フラグ）の入力値
emptyValue	emptyContentValue	accessInfo="yes"		
○	○	×	空要素	emptyContentValue 属性の指定値
			要素なし	emptyValue 属性の指定値※2
×	×	○	空要素	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1 （CBLXML-FLAG-EMPTY）
			要素なし	type 属性の指定によって，英数字と日本語は SPACE，数値は ZERO※1 （CBLXML-FLAG-MISSING）

（凡例）

○：指定あり

×：指定なし

注

- emptyValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyValue 属性の指定は無効となります。
- emptyContentValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyContentValue 属性の指定は無効となります。

注※1

type 属性の指定が"alphanumeric"または"national"の場合は SPACE が設定されます。

type 属性の指定が"numeric"，"packed"，"binary"，"float"，または"double"の場合は ZERO が設定されます。

注※2

emptyValue 属性の指定がない場合，emptyValue 属性の省略値が設定されます。

(2) EMPTY を指定した要素の出力

ELEMENT 属性のキーワードに「EMPTY」を指定した場合の要素の出力内容を表 7-9 に示します。

表 7-9　EMPTY を指定した要素の出力

Item 要素の emptyValue 属性，emptyContentValue 属性，accessInfo 属性			COBOL データ項目（アクセス情報フラグ）の出力値	出力 XML ドキュメント
emptyValue	emptyContentValue	accessInfo="yes"		
○	×	×	emptyValue 属性の指定値以外の値	要素を出力する。
			emptyValue 属性の指定値※3	省略可能な要素の場合は要素を出力しない。 省略できない要素の場合は emptyValue 属性の指定値を出力する。※1
○	○	×	emptyValue 属性，emptyContentValue 属性の指定値以外の値	COBOL データ項目に指定した値
			emptyContentValue 属性の指定値	空要素
			emptyValue 属性の指定値	省略可能な要素の場合は要素を出力しない。 省略できない要素の場合は emptyValue 属性の指定値を出力する。※1
			emptyContentValue 属性と emptyValue 属性が同じ指定値	省略可能な要素の場合は要素を出力しない。 省略できない要素の場合は空要素を出力する。
×	×	○	属性値（アクセス情報フラグに CBLXML-FLAG-OK を設定する）	要素を出力する。
			属性値（アクセス情報フラグに CBLXML-FLAG-EMPTY を設定する）	空要素を出力する。
			属性値（アクセス情報フラグに CBLXML-FLAG-MISSING を設定する）	省略可能な要素の場合は要素を出力しない。 省略できない要素の場合は要素値を出力する。※2

（凡例）

○：指定あり

×：指定なし

注

- emptyValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyValue 属性の指定は無効となります。
- emptyContentValue 属性と accessInfo 属性を同時に指定した場合，accessInfo 属性の指定値が優先され，emptyContentValue 属性の指定は無効となります。
- EMPTY に属性があり，その属性だけを AttrItem 要素で対応づけて XML ドキュメントを出力する場合，空要素を出力する。

注※1

空要素タグの例

```
<ABC/>
```

注※2

countVar 属性に"no"を指定した Array 要素の直下に繰り返し要素を Item 要素で対応づけて，アクセス情報フラグに CBLXML-FLAG-MISSING を指定した場合は，省略できない要素であっても出力されません。

注※3

emptyValue 属性の指定がない場合，emptyValue 属性の省略値を設定します。

7.2.11 XML ドキュメントの更新

XML ドキュメントの更新をする場合の XML アクセスルーチンの呼び出し順を次に示します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<!DOCTYPE root [
  <!ELEMENT root (X)>
  <!ELEMENT X (A,B?)>
  <!ELEMENT A (#PCDATA)>
  <!ELEMENT B (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="X">
  <Group elemName="X" update="yes">
   <Item elemName="A" size="5"/>
   <Item elemName="B" size="5"/>
  </Group>
 </BaseElement>
</Interface>
```

（生成される COBOL データ項目）

```
01 X.
 02 A PIC X(5).
 02 B PIC X(5).
```

（COBOL 原始プログラム）

```
                :
* open
     MOVE 'U' TO XML-MODE.
     MOVE 'data.xml' TO FILE-NAME.
     MOVE 8 TO XML-FILE-NAME-LENGTH.
     COMPUTE XML-FILE-NAME = FUNCTION ADDR(FILE-NAME).
     CALL 'CBLXML-OP-EXAMPLE'        …1.
         USING XML-FILE-NAME
               XML-FILE-NAME-LENGTH
               XML-MODE
               XML-POINTER
         RETURNING CBLXML-RETURN-CODE.

* read
     CALL 'CBLXML-RD-EXAMPLE-X'      …2.
         USING XML-POINTER X
         RETURNING CBLXML-RETURN-CODE.
     MOVE 'new A' TO A.
     MOVE 'new B' TO B.

* write
     CALL 'CBLXML-WR-EXAMPLE-X'      …3.
         USING XML-POINTER X
         RETURNING CBLXML-RETURN-CODE.

* close
     CALL 'CBLXML-CL-EXAMPLE'        …4.
         USING XML-POINTER
         RETURNING CBLXML-RETURN-CODE.
                :
```

XML ドキュメントを更新する場合，次の順序で XML アクセスルーチンを呼び出す必要があります。

1. アクセスモード"U"（更新）を指定して，CBLXML-OP-EXAMPLE アクセスルーチンで XML ドキュメントを開く。
2. CBLXML-RD-EXAMPLE-X アクセスルーチンで更新対象を入力する。
3. CBLXML-WR-EXAMPLE-X アクセスルーチンで更新対象を更新する。
4. CBLXML-CL-EXAMPLE アクセスルーチンで XML ドキュメントを閉じる。

XML ドキュメント

（更新前）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
  <X>
    <A>AAA01</A>
```

```
    <B>BBB01</B>
  </X>
</root>
```

（更新後）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
  <X>
    <A>new A</A>
    <B>new B</B>
  </X>
</root>
```

繰り返し要素の内側に BaseElement 要素を対応づけた場合は，更新したい位置まで CBLXML-RD-*Interface-BaseElement* アクセスルーチンを繰り返し呼び出し，更新したい項目に値を代入して，CBLXML-WR-*Interface-BaseElement* アクセスルーチンを呼び出すことで更新してください。3 回目の row 要素を更新する場合の XML アクセスルーチンの呼び出し例を次に示します。

（例）

```
CBLXML-OP-xyz                    ("U"更新モードでXMLドキュメントを開く)
CBLXML-RD-xyz-row                (1回目のrow要素を入力する)
CBLXML-RD-xyz-row                (2回目のrow要素を入力する)
CBLXML-RD-xyz-row                (3回目のrow要素を入力する)
ADD 1 TO QUANTITY OF ROW         (更新したい要素に値を指定する)
CBLXML-WR-xyz-row                (3回目のrow要素を更新する)
CBLXML-CL-xyz                    (XMLドキュメントを閉じる)
```

update 属性に"yes"を指定した Group 要素を elemName 属性で対応づけた場合，Group 要素以下が更新の対象になります。cobName 属性だけで対応づけた場合は，Group 要素の直下の Group 要素，および Item 要素以下が更新の対象となります。

次に，elemName 属性で対応づけた場合と，cobName 属性で対応づけた場合の例を示します。

（DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>

<!DOCTYPE root [
  <!ELEMENT root (X)>
  <!ELEMENT X (A,B)>
  <!ELEMENT A (#PCDATA)>
  <!ELEMENT B (#PCDATA)>
]>
<root/>
```

（DDF の例）

指定例 1

（elemName 属性で対応づけた場合）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="X">
 <Group elemName="X" update="yes">
   <Item elemName="A" size="5"/>
   <Item elemName="B" size="5"/>
 </Group>
 </BaseElement>
</Interface>
```

指定例 2

(cobName 属性で対応づけた場合)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="X">
 <Group cobName="X" update="yes">
   <Item elemName="A" size="5"/>
   <Item elemName="B" size="5"/>
 </Group>
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 X.
 02 A PIC X(5).
 02 B PIC X(5).
```

(更新される XML ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
  <X>
    <A>AAA</A>
    <!-- COMMENT -->
    <B>BBB</B>
  </X>
</root>
```

(更新後の XML ドキュメント)

指定例 1 の更新結果

elemName 要素で対応づけた場合，X 以下の要素が更新の対象となり更新時にコメントは削除されます。

```
<?xml version="1.0" encoding="Shift_JIS"?>

<root>
  <X>
    <A>New A</A>
    <B>New B</B>
  </X>
</root>
```

指定例 2 の更新結果

cobName 要素だけで対応づけた場合，X の直下の A 要素と B 要素が更新の対象となり，X の直下のコメントは，更新されません。

<?xml version="1.0" encoding="Shift_JIS"?>

```
<root>
  <X>
    <A>New A</A>
    <!-- COMMENT -->
    <B>New B</B>
  </X>
</root>
```

7.2.12　XML ドキュメントの更新機能の注意事項

- XML ドキュメントの更新処理では，更新対象外の部分であっても，更新前後で異なって出力される部分があります。詳細については，「付録 E.7　入力ドキュメントとの相違点」を参照してください。
- 更新対象以外の要素の値や属性の値に，定義済み実体参照や文字参照がある場合，「付録 D.1　定義済み実体参照」の規則に従って変換して更新されます。更新対象の要素の値や属性の値に，定義済み実体参照や文字参照がある場合は，cblxml コマンドの-nopeconv オプション（実体参照を変換しない）の指定に従って更新されます。
- XML ドキュメントを更新した場合，XML ドキュメントの XML 宣言は，属性「version="1.0"」，および encoding 属性だけの XML 宣言に変更されます。更新後の文字エンコーディングは，「付録 G.1(1)　出力する XML ドキュメントの文字エンコーディングの設定」に従って出力されます。更新機能を使用する場合，開いた XML ドキュメントの文字エンコーディングではなく出力する XML ドキュメントの文字エンコーディングの設定に従って変更されます。
- 更新前の XML ドキュメントに文書型宣言（DOCTYPE 宣言）がある場合，更新後の XML ドキュメントに，更新前と等価な文書型宣言（DOCTYPE 宣言）を復元します。次の点は更新前と異なって出力されます。

  1. 文書型宣言の開始を示す'<!DOCTYPE'，ルート要素名，公開識別子，システム識別子，内部サブセットおよび文書型宣言の終了を示す'>'を区切る空白文字（スペース，タブおよび改行）は，一つのスペースに置き換えられます。
  2. 公開識別子，およびシステム識別子を囲む引用符は，必ず二重引用符(")で出力されます。

- 省略可能な要素（?）や，繰り返し要素（*，＋）を BaseElement 要素に対応づけ，属性 update="yes"を指定して更新の対象とした場合，BaseElement 要素に入出力した要素の値を更新できますが，追加，および削除はできません。

  例えば，BaseElement 要素に対応づけた省略可能な要素が入力する XML ドキュメントに存在しない場合，その省略可能な要素の値を指定しても，その要素を出力できません。また，BaseElement 要素に対応づけた省略可能な要素が入力する XML ドキュメントに存在する場合，emptyValue 属性やアクセス情報フラグで要素を出力しないようにしても，削除できません。その場合，空要素が出力されます。

- 要素を更新の対象とした場合，その要素の属性も更新の対象とみなされるため，更新の対象とした要素の属性は，DDF で対応づけなければなりません。対応づけない場合は，属性の指定に従い，値が空に更新されるか，または属性が削除されます。
- XML ドキュメントを更新する場合，更新する要素や属性とそれらの値は，数値や文字列の出力規則に従って更新されます。詳細については「7.2.7　値の入力，出力の動作」を参照してください。
- 更新する XML ドキュメントの要素が次に示すように改行がなく，横に並んでいても更新する要素の内容だけが更新されます。

  B 要素を更新した場合

  （更新前）

```
<A>123</A><B>456</B><C>789</C>
```

  （更新後）

```
<A>123</A><B>777</B><C>789</C>
```

- 更新モードでの出力時に，属性 update="yes"を指定していない要素の値を変更しても，更新されません。
- バッファの XML ドキュメントを更新する場合で，更新前の XML ドキュメントの長さより更新後の XML ドキュメントが短くなったときは，短くなった領域は不定になります。そのため，CBLXML-CN-*Interface* アクセスルーチンを使用して更新後の XML ドキュメントの長さを取得し，更新後の XML ドキュメントの部分だけを参照してください。

  （更新前）

```
<row><root><I1>item1</I1><I1>item1</I1><I2>item2</I2></root></row>
```

  （更新後）

```
<row><root><I1></I1><I1></I1><I2>item2</I2></root></row>·····
```

- 省略可能な要素で複数の子要素を持つ場合，その子要素の一つだけに update="yes"を指定して，更新時に追加や削除をすると妥当でない XML ドキュメントが出力されます。このような省略可能な要素で複数の子要素を持つ場合は，すべての子要素に update="yes"を指定してください。次に一つの Item 要素だけに update="yes"を指定する不当な書き方の例を示します。

  （DTD の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>

<!DOCTYPE root [
  <!ELEMENT root (group1)>
  <!ELEMENT group1 (item1, item2)? >
  <!ELEMENT item1 (#PCDATA)>
  <!ELEMENT item2 (#PCDATA)>
]>
<root/>
```

  （DDF の例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
```

```
 <BaseElement elemName="group1">
  <Group elemName="group1">
   <Item elemName="item1" size="10" update="yes"/>
   <Item elemName="item2" size="10"/>
  </Group>
 </BaseElement>
</Interface>
```

emptyValue 値"SPACE"を COBOL データ項目に設定することで item1 要素を削除したとき，次のように更新後のドキュメントは DTD に対して妥当でなくなります。item1 要素と item2 要素の両方に update="yes"属性を指定し，item1 要素を削除した場合は，item2 要素も削除してください。

(更新前)

```
<?xml version="1.0" encoding="Shift_JIS"?>

<root>
  <group1>
    <item1>data1</item1>
    <item2></item2>
  </group1>
</root>
```

(更新後の不当な XML ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
  <group1>
    <item2/>
  </group1>
</root>
```

- ファイル上の XML ドキュメントを更新した場合，更新対象となる要素の前後にある空白文字（スペース，タブおよび改行）の個数や位置が更新の前後で異なって出力されることがあります。
- 属性だけを更新することはできません。属性を更新する場合，属性に対応する要素も対応づけてください。
- XML ドキュメントを更新した場合，XML 宣言と文書型宣言（DOCTYPE 宣言）の間のコメントは削除されます。
- バッファ上の XML ドキュメントを更新する際にバッファ長を超えた場合，CBLXML-CL-*Interface*，CBLXML-CN-*Interface* アクセスルーチンで XML ドキュメントを閉じるときにバッファ長を超えた旨のステータス 115 が返されます。CBLXML-WR-*Interface-BaseElement* アクセスルーチンではバッファ長を超えた旨のステータス 115 が返されませんので注意してください。
- DTD を持つ XML ドキュメントを更新した場合，DTD 上でデフォルト値を持つ属性は，更新対象外の部分であっても，更新後の XML ドキュメントに出力されます。
- バッファの XML ドキュメントを更新する場合で，更新する XML ドキュメントを超えるバッファの領域は更新する XML ドキュメントの文字エンコーディングに従った空白文字（スペース，タブおよび改行）でなければなりません。

文字エンコーディング	スペース	タブ	改行
Shift_JIS，Windows-31J，EUC-JP	0x20	0x09	（Windows の場合）0x0a0d, （UNIX の場合）0x0a
UTF-8	0x20	0x09	（Windows の場合）0x0a0d （UNIX の場合）0x0a
UTF-16，リトルエンディアン	0x0020	0x0009	（Windows の場合）0x000a000d （UNIX の場合）0x000a
UTF-16，ビッグエンディアン	0x2000	0x0900	（Windows の場合）0x0a000d00 （UNIX の場合）0x0a00

7.3　XML アクセスルーチンが返すステータス

7.3.1　ステータスの概要

XML 対応 COBOL プログラムの実行時，XML アクセスルーチンから戻り値としてステータスが返されます。

ステータスの値

ステータスは，0～199 の範囲の値で返されます。

0：

正常終了を表します。

1～99 の範囲：

回復可能なエラーを表します。アクセス中の XML ドキュメントに対する操作を続行できます。

100～199 の範囲：

回復できない致命的エラーであり，アクセス中の XML ドキュメントに対する操作をそれ以上続行できないことを表します。ただし，CBLXML-RD-*Interface-BaseElement* アクセスルーチンおよび CBLXML-WR-*Interface-BaseElement* アクセスルーチンで致命的エラーが発生した場合は，必ず CBLXML-CL-*Interface* または CBLXML-CN-*Interface* アクセスルーチンを呼び出して XML ドキュメントを閉じてください。

ステータスに対応するデータ項目

cblxml コマンドが生成する XML アクセス用データ定義には，それぞれのステータスと同じ値が格納されたデータ項目が定義されています。

ステータスごとに切り分け処理をする場合は，ステータス値に対応するステータス定義名称を使用することを推奨します。ステータス定義名称の詳細については，「4.4　XML アクセス用ステータス定義」を参照してください。

7.3.2　ステータスの一覧

このマニュアルでは，ステータスの一覧を次の形式で記載します。

ステータス番号（エラー定義名称）【ステータスを返すアクセスルーチン】

ステータスの説明

（要因）

ステータスが返される要因を示す。

（対策）

プログラム作成者の処置を示す。

ステータス番号

XML アクセスルーチンが戻り値として返す番号を示します。

エラー定義名称

ステータス番号に対応した COBOL のデータ名です。エラー定義名称は，XML アクセス用ステータス定義によって定義されています。

ステータスを返すアクセスルーチン

ステータスを返すアクセスルーチンの種類を示します。

OP：CBLXML-OP-Interface アクセスルーチン

OB：CBLXML-OB-Interface アクセスルーチン

RD：CBLXML-RD-Interface-BaseElement アクセスルーチン

WR：CBLXML-WR-Interface-BaseElement アクセスルーチン

CL：CBLXML-CL-Interface アクセスルーチン

CN：CBLXML-CN-Interface アクセスルーチン

ステータスの一覧を次に示します。

0（CBLXML-OK）【OP】【OB】【RD】【WR】【CL】【CN】

XML アクセスルーチンは正常終了しました。

1（CBLXML-ELEM-NOT-DTD）【WR】

要素が DTD ファイルの構成と一致しません。

（要因）

出力した BaseElement 要素が DTD で定義した XML ドキュメントの構成と一致していない。

（対策）

DTD で定義した XML ドキュメントの構成と一致するように BaseElement 要素を出力する。

3（CBLXML-ONLY-WRITE-BE）【WR】

BaseElement 要素を見つけることができません。

（要因）

BaseElement 要素が見つからない。原因として，cblxml コマンドの出力プログラムが改変されたことが考えられる。

（対策）

cblxml コマンドで XML アクセスルーチン（COBOL 副プログラム）を生成し直す。

4（CBLXML-CANT-FIND-BE）【WR】

BaseElement 要素に対応する要素名が見つかりません。

（要因）

BaseElement が DTD 中に定義されていない。原因として，cblxml コマンドの出力プログラムが改変されたことが考えられる。

（対策）

XML サービスルーチンのインタフェース名称を確認し，呼び出し順序を見直す。または，cblxml コマンドで XML アクセスルーチン（COBOL 副プログラム）を生成し直す。

5（CBLXML-CANT-SKIP-BE）【RD】

BaseElement 要素を飛ばして，次の BaseElement 要素を入力できません。

（要因）

読み込み中の XML ドキュメントの現在位置には，読み込もうとした BaseElement 要素とは異なる BaseElement 要素が存在する。

（対策）

XML ドキュメントの現在位置に存在する BaseElement 要素に対応した CBLXML-RD-*Interface-BaseElement* ルーチンを実行する。

6（CBLXML-MUTUAL-VIOLATE）【WR】

選択要素に対応づけられたすべてのデータ項目が空です。または選択要素の二つ以上の COBOL データ項目に値が設定されているため，出力できません。または，入出力データ情報定義機能を使用している場合に，選択要素の二つ以上のアクセス情報フラグに要素を出力するフラグを設定したため，出力できません。

（要因）

選択要素に対応づけたすべてのデータ項目に emptyValue 属性の値を設定しているか，二つ以上のデータ項目に emptyValue 属性の値以外の値を設定しているため，要素を出力できない。または，入出力データ情報定義機能を使用している場合，選択要素の二つ以上のアクセス情報フラグに要素を出力するフラグを設定しているため，要素を出力できない。

（対策）

出力したい要素以外の要素に対応するデータ項目に，emptyValue 属性の値を設定して出力する。入出力データ情報定義機能を使用している場合，出力する要素に対応する COBOL データ項目のアクセス情報フラグだけに要素を出力するフラグを設定する。

7（CBLXML-PLUS-EMPTY-WR）【WR】

'+'の繰り返し要素と対応づけられた COBOL データ項目の内容を出力するとき，出力数が 0 に設定されています。

（要因）

DTD で'+'の繰り返し記号で表される繰り返し要素を出力しようとしたが，nameOfCountVar 属性に設定されているデータ項目の値が 0 になっている。

（対策）

nameOfCountVar 属性に設定されているデータ項目に 1 以上の値を指定して出力する。

8（CBLXML-ILLEGAL-CHAR）【RD】

XML ドキュメントの入力時に不当な文字が見つかりました。

（要因）

cblxml コマンドに-chkchar オプションを指定して生成した COBOL 原始プログラムを用いて，XML ドキュメントから要素や属性を入力した場合，type="national"を指定したデータ項目に 1 バイト文字が入力された。数字項目や浮動小数点項目に数値以外の値や sign="unsigned"指定時に負値が入力された。または，type 属性に"numeric"，"packed"，"binary"，"float"，"double"を指定した場合に 320 けたを超えた数値が入力された。

（対策）

入力する値が，type 属性に指定した値の型に対し有効な文字から構成されるようにする。

9（CBLXML-NOT-WRITTEN）【CN】

アクセスモード"R"で開いた XML ドキュメントに対して，CBLXML-CN-*Interface* アクセスルーチンを呼び出しました。XML ドキュメントを閉じる処理は成功しますが，XML ドキュメントの長さには 0 が設定されます。

（要因）

読み取りモードで開いた XML ドキュメントに対して，CBLXML-CN-*Interface* アクセスルーチンを呼び出した。

（対策）

読み取りモードで開いた XML ドキュメントを閉じる場合は，CBLXML-CL-*Interface* アクセスルーチンを呼び出す。

10（CBLXML-VALUE-OVERFLOW）【RD】

XML ドキュメントから入力した要素，属性の値がオーバフローしました。

(要因)

cblxml コマンドに-chkovflow オプションを指定して生成した COBOL 原始プログラムを用いて，XML ドキュメントから要素や属性を入力した場合で，要素や属性の値の長さが size 属性に指定したけた数を超えた。ただし，数字項目は左側の 0 を削除した数値のけた数が size 属性に指定したけた数を超えた。また，浮動小数点数はシステムで定義した浮動小数点数の絶対値を超えた。ただし，type 属性に"numeric"，"packed"，"binary"，"float"，"double"を指定した場合に入力する数値が 320 けたを超えたときは，ステータスに 8（CBLXML-ILLEGAL-CHAR）を返す。

(対策)

size 属性のけた数を大きくする。または，XML ドキュメントの要素，属性値の長さが size 属性に指定したけた数を超えないようにする。

11（CBLXML-ILLEGAL-CHAR-WRITE）【WR】

XML ドキュメントの出力時に不当な文字が指定されました。

(要因)

XML ドキュメントを出力する場合に type="alphabetic"，"alphanumeric"，"national"，"numeric"，"packed"に対応する COBOL データ項目に 0x00～0x08，0x0b，0x0c，0x0e～0x1f，0x80，0xa0，0xfd～0xff，シフト JIS コード，日本語 EUC の多バイト文字以外を指定して出力しようとした。

(対策)

出力する文字列から制御文字や不当な多バイト文字文字を削除して出力する。

12（CBLXML-NO-UPDATE）【CL】【CN】

更新モードで開いた XML ドキュメントを 1 回も更新していません。

(要因)

アクセスモード"U"で開いた XML ドキュメントに対して CBLXML-WR-*Interface-BaseElement* アクセスルーチンを 1 回も呼び出さずに CBLXML-CL-*Interface*，CBLXML-CN-*Interface* アクセスルーチンを呼び出した。

(対策)

アクセスモード"U"で開いた XML ドキュメントを更新する。

13（CBLXML-NO-UPDATE-BE）【WR】

更新モードで開いた XML ドキュメントに対して更新の対象を指定しないで（update 属性を指定していない）出力しました。

(要因)

アクセスモード"U"で開いた XML ドキュメントに対して更新の対象を指定していない DDF を用いて更新しようとした。

(対策)

DDF で更新の対象を設定する。

15（CBLXML-NOT-READ）【WR】

更新モードで開いた XML ドキュメントを入力しないで出力しました。または，開いた XML ドキュメントに対して直前に同じ BaseElement 要素単位で入力しないで出力しました。

(要因)

更新モードで開いた XML ドキュメントを入力しないで出力した。または，更新モードで開いた XML ドキュメントで，直前の入力に対して異なる BaseElement 要素単位を出力した。

(対策)

同じ BaseElement 要素単位の CBLXML-RD-*Interface-BaseElement* アクセスルーチンで入力したあとに同じ BaseElement 要素単位の CBLXML-WR-*Interface-BaseElement* アクセスルーチンを呼び出す。

101（CBLXML-NOT-ENOUGH-MEM）【OP】【OB】【RD】【WR】【CL】【CN】

操作を続けるメモリを確保できません。

(要因)

操作を続けるためのメモリが不足した。

(対策)

使用可能なメモリを増やして再実行する。

103（CBLXML-FILE-OPEN-FAIL）【OP】

ファイル上の XML ドキュメントを開けません。

(要因)

ファイルを書き込みモード，または更新モードで開けなかった（例：書き込み権限がない)。または，不当なファイル名称を指定した。

(対策)

ファイルのアクセス権限，ディスク容量などを見直す。または，ファイル名称を見直す。

104（CBLXML-FILE-CLOSE-FAIL）【CL】【CN】

ファイル上の XML ドキュメントを閉じられません。

(要因)

書き込みモード，または更新モードで開かれたファイルを閉じられなかった（例：ディスクがいっぱいになった)。

(対策)

ディスク容量などのファイル環境を見直す。

105（CBLXML-BAD-SESSION）【OP】【OB】

ファイルモードに不当なモードが指定されました。

(要因)

XML ドキュメントのアクセスモードに"R"，"W"，"U"，"V"，"E"以外の文字を指定した。または，アクセスモードの左端に"R"，"W"，"U"以外の文字を指定した。

(対策)

XML ドキュメントを開く際のアクセスモードに正しい文字列を指定する。

106（CBLXML-CANT-WRITE）【WR】【CL】【CN】

ファイル上の XML ドキュメントに出力できません。

(要因)

ディスク容量不足，アクセス権がない，ネットワーク障害などの要因で，ファイルに出力できなかった。

(対策)

ディスク容量，ファイルのアクセス権，ネットワークの状態などを見直す。

109（CBLXML-RECURSIVE-DTD）【WR】【CL】【CN】

DTD ファイルに再帰的な要素を見つけました。

(要因)

DTD に再帰的な要素が定義されている。

(対策)

DTD を再帰しない構造で定義する。

110（CBLXML-XML-PARSE-FAIL）【OP】【OB】

XML ドキュメントの解析でエラーが発生しました。

(要因)

XML ドキュメントの解析に失敗した。次の要因が考えられる。

- XML ドキュメントが整形式でない。
- 構造が DTD の定義と一致しない。
- エンティティ参照回数が制限した値を超えた。

CBLXML-GET-ERROR サービスルーチンを使用すると，詳細なエラー情報を取得できる。詳細については，「9.3.2　エラー情報の取得」を参照のこと。

エンティティ参照回数と制限値については，「付録 E.8　エンティティ参照の扱い」を参照のこと。

（対策）

XML ドキュメントのエラー部分を修正する。

111（CBLXML-NO-BE-LEFT）【RD】

XML ドキュメントに BaseElement 要素が見つかりません。

（要因）

入力 XML ドキュメントに読み込む BaseElement 要素が残っていない。

（対策）

すべての BaseElement 要素を読み終わった場合は，XML ドキュメントを閉じる。

114（CBLXML-INVALID-PARAMS）【OP】【OB】【RD】【WR】【CL】

XML アクセスルーチンに指定した引数に誤りがあります。

（要因）

XML アクセスルーチンに指定した引数が，各アクセスルーチンの引数仕様を満たしていない。

（対策）

XML アクセスルーチンの引数を見直す。

115（CBLXML-BUF-OVERFLOW）【WR】【CL】【CN】

バッファへの出力中にバッファ長を超えました。

（要因）

バッファへの書き込み時に，書き込む XML ドキュメントの長さがバッファ長を超えた。

（対策）

バッファ長を書き込む XML ドキュメント長よりも大きくとる。

117（CBLXML-FILE-IO）【OP】【OB】【RD】【WR】【CL】【CN】

システムファイル I／O の操作に失敗しました。

（要因）

ファイル I／O のシステム関数の実行に失敗した。次の要因が考えられる。

- XML ドキュメントを出力するファイルシステムの容量が不足した。
- ファイルに対するアクセス権がない。

- メモリが不足した。
- 出力する一つの XML ドキュメントに複数から同時にアクセスした。

（対策）

ファイルのエラー原因を調査する。

118（CBLXML-CANT-END-DOC）【CL】【CN】

XML ドキュメントを閉じるタグを出力できません。

（要因）

書き込みモードで開いた XML ドキュメントに要素を出力しないで，XML ドキュメントを閉じた。

（対策）

XML ドキュメントを閉じる前に，要素を出力する。

119（CBLXML-ALREADY-OPEN）【OP】【OB】

一つの XML ポインタに対して複数の XML ドキュメントを開こうとしました。

（要因）

アクセスモード"E"を指定して一つの XML ポインタに対して複数の XML ドキュメントを開こうとした。

（対策）

XML ドキュメントを開いている XML ポインタにアクセスモード"E"を指定して CBLXML-OP-*Interface*， CBLXML-OB-*Interface* アクセスルーチンを呼び出さないように処理を訂正する。

121（CBLXML-INTERNAL）【OP】【OB】【RD】【WR】【CL】【CN】

内部で論理エラーが発生しました。

（要因）

XML アクセスルーチンで内部的に論理エラーが発生した。または，XML アクセスルーチンを生成した cblxml コマンドに対して XML アクセス用実行時ライブラリの方が古い。

（対策）

当社保守員に連絡する。または，XML アクセスルーチンを生成した cblxml コマンドが含まれるプロダクトと，同じまたは新しいバージョンのプロダクトをインストールする。

122（CBLXML-INVALIDCBLLANG）【OP】【OB】

-unisrc オプションの指定と環境変数 CBLLANG の設定に矛盾があります。

（要因）

cblxml コマンドで-unisrc オプションを指定して生成した Unicode 機能に対応した XML アクセスルーチンを実行する場合，環境変数 CBLLANG が設定されていないか，環境変数 CBLLANG に UNICODE

以外の値が指定されている。または，-unisrc オプションを指定しないで生成した XML アクセスルーチンを実行する場合に環境変数 CBLLANG が UNICODE に設定されている。

（対策）

-unisrc オプションを指定した Unicode 機能に対応した XML アクセスルーチンを実行する場合，環境変数 CBLLANG=UNICODE に設定する。-unisrc オプションを指定していない XML アクセスルーチンを実行する場合，環境変数 CBLLANG を設定しない。

123（CBLXML-INVALIDLANG）【OP】【OB】

-unisrc オプションの指定と環境変数 LANG の設定に矛盾があります。

（要因）

- （AIX の場合）cblxml コマンドで-unisrc オプションを指定して生成した Unicode 機能に対応した XML アクセスルーチンを実行する場合，環境変数 LANG が設定されていないか，環境変数 LANG にシフト JIS 以外の値が設定されている。
- （Linux の場合）cblxml コマンドで-unisrc オプションを指定して生成した Unicode 機能に対応した XML アクセスルーチンを実行する場合，環境変数 LANG が設定されていないか，環境変数 LANG に UTF-8 以外の値が設定されている。

（対策）

- （AIX の場合）cblxml コマンドで-unisrc オプションを指定している場合，環境変数 LANG をシフト JIS に設定する。
- （Linux の場合）cblxml コマンドで-unisrc オプションを指定している場合，環境変数 LANG を UTF-8 に設定する。

なお，文字コードについては，「付録 G.2　文字コード」を参照のこと。

124（CBLXML-PROG-FILE-TOO-OLD）【OP】【OB】

互換性がない古い XML アクセスルーチンを使用しています。

（要因）

実行したバージョンと互換性がない古いバージョンの cblxml コマンドを使用して生成した XML アクセスルーチンを使用した。

（対策）

実行したバージョンと同じバージョンの cblxml コマンドを使用して，XML アクセスルーチンおよび XML アクセス用データ定義を再生成する。

7.4　実行時のメモリ所要量

XML 連携機能を使用して XML ドキュメントをアクセスする場合，XML ドキュメントによっては使用するメモリ所要量が増大することがあります。ここでは，メモリ所要量の概算式について説明します。

この式は，メモリ所要量の上限を求めるものです。XML ドキュメントでタグが占める割合が低いほど，また，繰り返しが多い（同じ名前のタグや属性が多い）ほど，実際のメモリ所要量は小さくなります。

特に記載しないかぎり，概算式の結果および所要量を示す値の単位はバイトです。

7.4.1　XML ドキュメントを入出力，更新する場合のメモリ所要量

XML ドキュメントを入出力，または更新する場合のメモリ所要量を求める概算式を次に示します。

```
上限値=DT＋MA＋(NS＋2×(XC×2.6＋MC×5))×1.2＋MC＋ML＋UP
```

概算式で使用している変数について説明します。

(1)　DT

文書型定義（DTD）がある場合に，この DTD の情報を保持するために必要なメモリ領域です。DTD がない場合には，この項の値は 0 としてください。

DTD の情報を保持するために必要なメモリ所要量の詳細は，「7.4.2　文書型定義（DTD）の情報を保持するためのメモリ所要量」を参照してください。

(2)　MA

XML 連携機能内で使用する情報を管理するために必要なメモリ領域です。

この値は固定値で，220 キロバイトです。

(3)　NS

XML ドキュメントでの各構成要素の種類ごとのメモリ領域の累計です。

各構成要素の単価と構成要素数の積の総和になります。各構成要素の単価を次に示します。

表 7-10　XML ドキュメントでの各構成要素の単価

構成要素	システム	
	Windows(x86)，AIX(32)，Linux(x86)	Windows(x64)，AIX(64)，Linux(x64)
要素※1	240	420

構成要素	システム	
	Windows(x86)，AIX(32)，Linux(x86)	Windows(x64)，AIX(64)，Linux(x64)
属性※2	160	1,190
連続するテキスト※3	60	90
CDATA セクション	60	90
コメント	60	90
エンティティ宣言※4	870	980
エンティティ参照※4	140	220
記法宣言	680	720
処理命令	60	80
文書型宣言※5	920	1,310
XML ドキュメント※6	260	520

注※1

タグ間のテキストや子要素は含みません。

注※2

属性値は含みません。

注※3

タグを間に含まない連続するテキストデータです。

タグの前後の改行や空白文字もテキストデータとして扱われることに注意してください。

注※4

エンティティの内容のテキストは含みません。

注※5

DTD の内容（要素型宣言やエンティティ宣言など）は含みません。

注※6

XML ドキュメント 1 つに対して 1 つ生成されます。XML ドキュメント内の DTD，タグ，およびテキストデータは含みません。

(4) XC

XML ドキュメント内の文字数です。多バイト文字が混在している場合は，次の計算方式で算出してください。

シフト JIS または日本語 EUC で多バイト文字が混在している場合

```
XC＝XS×(100-DR／2)／100　　(切り上げ)
```

各変数は，次のとおりです。

- XS：XML ドキュメントのサイズ
- DR：XML ドキュメントの多バイト文字の占有率（%）

UTF-16 でエンコードされた XML ドキュメントの場合

```
XC＝XS／2　　（切り上げ）
```

変数は，次のとおりです。

- XS：XML ドキュメントのサイズ

UTF-8 でエンコードされた XML ドキュメントの場合

```
XC＝XS×(100-DR2／2-2×DR3／3)／100　　（切り上げ）
```

各変数は，次のとおりです。

- XS：XML ドキュメントのサイズ
- DR2：XML ドキュメントの 2 バイトコード占有率（%）
- DR3：XML ドキュメントの 3 バイトコード占有率（%）

(5) MC

連続するテキストデータ（タグ間テキスト，CDATA セクション，およびコメントを含む）の最大文字数です。

文書系の XML ドキュメント※の場合，この値の影響を大きく受けます。

伝票系の XML ドキュメント※の場合，最大文字数が XML ドキュメント全体の文字数に比べて小さいので，ほとんどの場合でこの項の影響を受けません。

注※

文書系の XML ドキュメントとは，タグが少なくテキストデータの割合の大きい XML ドキュメントです。伝票系の XML ドキュメントとは，タグが多くタグ間テキストが短い XML ドキュメントです。

(6) ML

マッピングした Item 要素のタグ名を管理するメモリ領域です。次の計算方式で算出してください。

```
Σ(MNL×2＋MPL)
```

各変数は，次のとおりです。

- MNL：Item 要素のタグ名の長さ
- MPL：Item 要素の BaseElement 要素からのパス名の長さ
  このパス名は BaseElement 要素から子要素をたどって，Item 要素に至るまでのタグ名をスラント（/）でつなげた文字列です。

Σ はデータ定義ファイル（DDF）でマッピングしたすべての Item 要素の合計を示します。

(7)　UP

XML ドキュメントを更新するときに，更新の情報などを保持するために必要なメモリ領域です。

XML ドキュメントを更新しない場合は，この項の値は 0 としてください。XML ドキュメントを更新のために必要なメモリ所要量の詳細は「7.4.3　XML ドキュメントを更新するためのメモリ所要量」を参照してください。

7.4.2　文書型定義（DTD）の情報を保持するためのメモリ所要量

XML ドキュメントに DTD がある場合に，DTD の情報を保持するために必要なメモリ領域を求める概算式を次に示します。

```
DT＝DS＋CM
```

概算式で使用している変数について説明します。

(1)　DS

DTD 内に現れる各宣言の種類ごとの累計です。

各宣言の単価と宣言数の積の総和になります。各宣言の単価を次に示します。

表 7-11　DTD 内の各宣言の単価

構成要素	単価	
	Windows(x86)，AIX(32)，Linux(x86)	Windows(x64)，AIX(64)，Linux(x64)
要素	720	860
属性	720	890
エンティティ宣言	1,400	1,670
記法宣言	900	1,270

(2)　CM

要素型宣言内に含まれる内容モデルの情報を格納するために必要なメモリ領域です。

入力 XML ドキュメントの妥当性チェック機能を使用する場合は，次の概算式で求めます。

```
CM＝4×(EL^2)＋1280×EL
```

各変数は，次のとおりです。

- ^：べき乗を表します。
- EL：要素型宣言数

入力 XML ドキュメントの妥当性チェック機能を使用しない場合，0 となります。

7.4.3　XML ドキュメントを更新するためのメモリ所要量

XML ドキュメントを更新する場合，追加で必要なメモリ所要量を求める概算式を次に示します。

```
UP＝UDT＋UMA＋USS＋8×MC
```

概算式で使用している変数について説明します。

(1)　UDT

DTD がある場合に，DTD の情報を保持するために必要なメモリ領域です。

DTD がない場合には，この項の値は 0 としてください。DTD の情報を保持するために必要なメモリ所要量の詳細は，「7.4.2　文書型定義（DTD）の情報を保持するためのメモリ所要量」を参照してください。

(2)　UMA

XML 連携機能内で使用する情報を管理するために必要なメモリ領域です。

この値は固定値で，80 キロバイトです。

(3)　USS

XML ドキュメント内に現れる要素，および属性の名前ごとの累計です。各構成要素の概算式の算出結果の総和になります。

同じ名前の要素や属性が XML ドキュメント内に繰り返し現れるほど，XML ドキュメントのサイズに対して，メモリ所要量が一定値に近づきます。

各構成要素の概算式を次に示します。

表 7-12　更新時の XML ドキュメントでの各構成要素の概算式

構成要素	概算式	
	Windows(x86)，AIX(32)，Linux(x86)	Windows(x64)，AIX(64)，Linux(x64)
要素	180 ＋ 3×NC	500 ＋ 3×NC
属性	400 ＋ 3×NC	560 ＋ 3×NC

変数は，次のとおりです。

- NC：各構成要素の名前の文字数

各構成要素の名前の文字数に大きなばらつきがなければ平均値で近似し，定数として扱ってもかまいません。

(4)　MC

連続するテキストデータの最大文字数です。

詳細は，「7.4.1　XML ドキュメントを入出力，更新する場合のメモリ所要量」の「(5)　MC」を参照してください。

7.4.4　概算式の計算例

次に示す XML ドキュメントでの概算式の計算例を示します。

この XML ドキュメントは，サイズが 7,284 バイトで，シフト JIS の 2 バイトコードが 4,000 バイト（2,000 文字）あります。2 バイトコードの占有率が約 55%となっています。

この例では，改行文字を「↓」と表記しています。

(XML ドキュメント)

```
<?xml version="1.0" encoding="Shift_JIS"?>↓
<root>↓                                                    …1
    <e1>abcdefghij</e1>↓                                   …2
    <e2>klmnopqrst<e3>uvw</e3>xyz</e2>↓                    …3
<!--comment-->↓                                            …4
    <e4></e4>↓                                             …5
    <e5 a1="abc" a2=""/>↓                                  …6
    <e5 a1="12345"/>↓                                      …7
    <e5>↓
 <?pi-target pi-data?>↓
       <e6>xxxxxx ... (3000文字) ... xxxxxx<e6/>↓
       <e6>あああ ... (2000文字) ... あああ<e6/>↓
    </e5>↓
</root>
```

データ定義ファイル（DDF）は BaseElement 要素を root に対応づけし，e1 を Item 要素に対応づけします。

(DDF)

```
<?xml version="1.0" encoding="Shift_JIS"?>↓
<Interface interfaceName="EXAMPLE">↓
 <BaseElement elemName="root" cobName="BE">↓
  <Item elemName="e1" size="10"/>↓
 </BaseElement>↓
</Interface>
```

この XML ドキュメントを XML 連携機能で解析する場合のメモリ所要量の概算値は次のようになります。ここでは，Windows(x86)の場合の例を示します。ほかのプラットフォームの場合は，それぞれの概算式に各単価を当てはめて算出してください。

(計算例)

```
上限値＝DT＋MA＋(NS＋2×(XC×2.6＋MC×5))×1.2＋MC＋ML＋UP
```

上限値は，302,670 バイトです。

```
DT＝0                 …DTDがない場合，0バイト。
MA＝220×1,024         …固定値（220キロバイト）
NS＝240×10            …要素数＝10（root,e1,e2,e3,e4,e5×3,e6×2）
    ＋160×3           …属性数＝3（a1×2,a2）
    ＋60×21           …連続するテキストの数＝21※
    ＋60×1            …コメント数＝1
    ＋60×1            …処理命令数＝1
    ＋260             …XMLドキュメント
    ＝4,520
XC＝XS×(100-DR／2)／100
    ＝7,284×(100-55／2)／100
    ≒5,281            …XMLドキュメントの文字数。
                         XMLドキュメントのサイズ7,284バイト，
                         2バイトコード占有率55%として換算。
MC＝3,000             …最大長のテキストデータxxx...xxx（3,000文字）
ML＝Σ(2×2＋8)         …DDFでBaseElement要素をrootに対応づけし，
    ＝12                 e1をItem要素に対応づけした場合，12バイト。
                         マッピングしたタグ名のサイズ（e1）＝2
                         BaseElementからのパス名のサイズ（/root/e1）＝8
UP＝0                 …更新しない場合，0バイト。
```

注※

タグの前後の改行や空白文字もテキストデータとして扱われます。

- 1.の改行と，続く 2.のインデントは合わせて 1 つの連続するテキストデータとして扱われます。
- 4.の改行と 5.のインデントも合わせて 1 つの連続するテキストデータとして扱われます。
- 3.の改行も 1 つのテキストデータとして扱われます。
- 3.の klmnopqrst，uvw，xyz はそれぞれ独立したテキストデータとして扱われます。
- 5.の空のタグ間テキストはテキストデータとして扱われません。
- 6.，7.の各属性の値は独立したテキストデータとして扱われます。
- 6.の属性 a2 の属性値は空文字列ですが，5.の空のタグ間テキストと異なり，テキストデータとして扱われます。

8 開発マネージャ連携（Windows の場合）

この章では，Windows COBOL2002 の開発マネージャを使用して，XML 対応 COBOL プログラムを開発する方法について説明します。

8.1　開発マネージャ上でのファイルの表示名

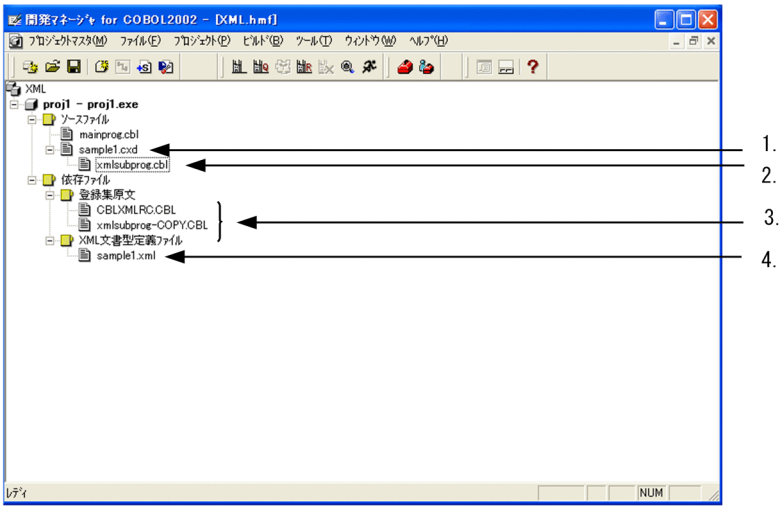
XML 対応 COBOL プログラムで使用するファイルは，開発マネージャ上では次の名称で表示されます。

ファイル種別	拡張子	開発マネージャ上での表示名
データ定義ファイル （DDF ファイル）	.cxd	XML データ定義ファイル
文書型定義ファイル （DTD ファイル）	.dtd .xml	XML 文書型定義ファイル

DDF ファイルから生成される COBOL 原始プログラム（XML アクセスルーチン）については，通常の COBOL 原始プログラムと同じ表示になります。

8.2　開発マネージャ上でのイメージ図

開発マネージャ上で XML 対応 COBOL プログラムを作成しているプロジェクトのイメージ図を次に示します。



1. データ定義ファイル（DDF ファイル）
2. XML データ定義ファイルから生成される COBOL 原始プログラム（XML アクセスルーチン）
3. COBOL 原始プログラムのコンパイル時に登録される登録集原文（XML アクセス用データ定義，XML アクセス用ステータス定義）
4. 文書型定義ファイル（DTD ファイル）

8.3　開発マネージャの操作

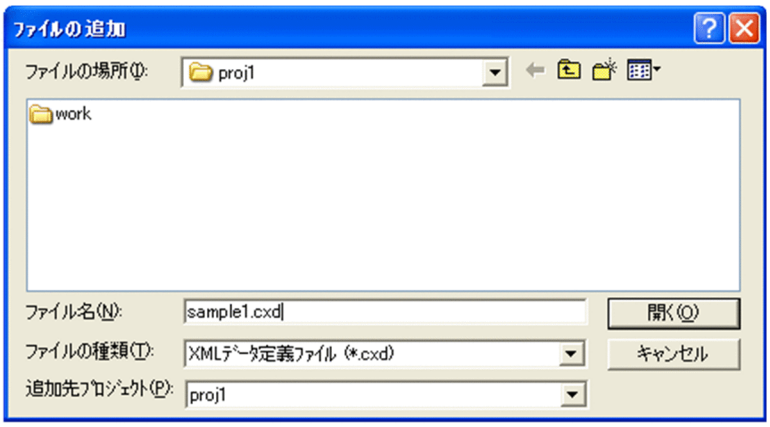
8.3.1　DDF ファイルの登録

DDF ファイルは，開発マネージャがサポートしている，OLE プロジェクトを除くプロジェクトの種類で登録できます。

DDF ファイルを登録する手順を次に示します。

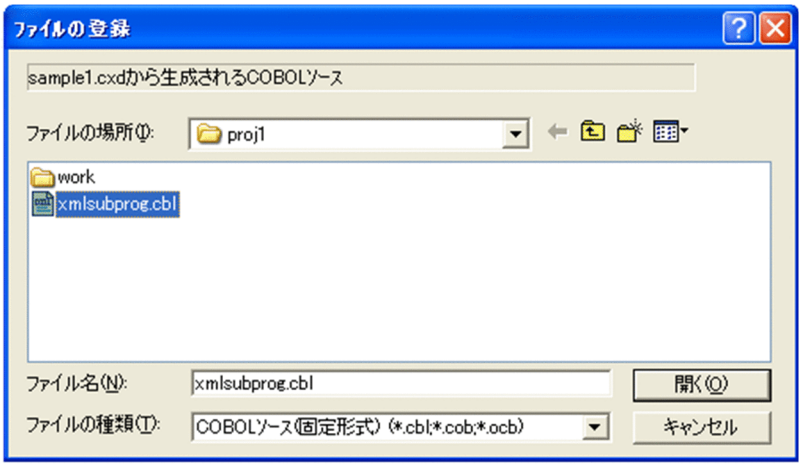
1. DDF ファイルの登録
2. 生成される COBOL 原始プログラムの登録
3. DTD ファイルの登録

1. ［プロジェクト］メニューから［ソースファイルの追加］を選ぶ
   ファイルの追加ダイアログボックスが表示されます。



2. ファイルの種類ドロップダウンリストで「XML データ定義ファイル（*.cxd)」を指定する

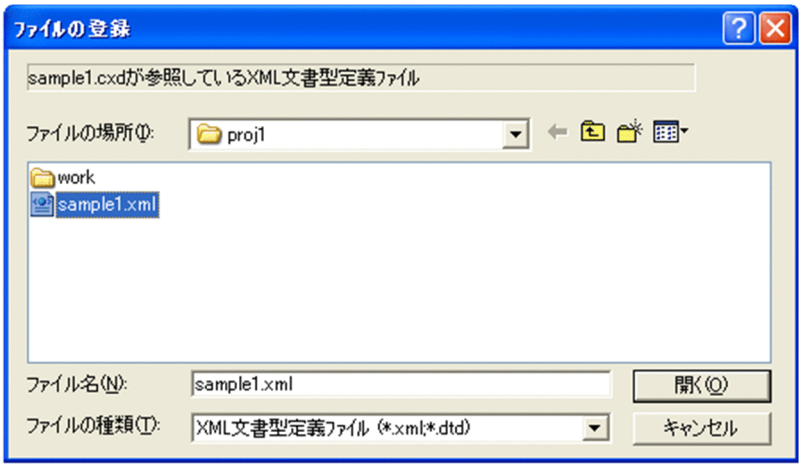
3. ファイル名を指定して［開く］ボタンを選ぶ
   ファイルの登録ダイアログボックスが表示されます。



4. ファイルの種類ドロップダウンリストで「COBOL ソース（固定形式）（*.cbl,*.cob,*ocb)」を指定する

5. DDF ファイルから生成される COBOL ソースファイル名を指定して［開く］ボタンを選ぶ

   再度ファイルの登録ダイアログボックスが表示されます。



6. ファイルの種類ドロップダウンリストで「XML 文書型定義ファイル（*.xml,*.dtd)」を指定する

7. DDF ファイルが参照している DTD ファイルを指定して［開く］ボタンを選ぶ

   DDF ファイル，生成される COBOL ソースファイル，および DTD ファイルがプロジェクトに登録されます。なお，DTD ファイルは DDF ファイルの依存ファイルとして登録されます。

注意事項

DDF ファイルから生成される COBOL ソースファイルは，プロジェクトのメインファイルにはできません。

8.3.2　DDF ファイルと DTD ファイルの除外

DDF ファイルと DTD ファイルを除外する手順を次に示します。

1. DDF ファイルは開発マネージャの画面上で，該当するファイルを選択する

2. ［ファイル］メニューの［削除］を選ぶ

   DDF ファイルが除外されます。

注意事項

- DDF ファイルを除外すると，DDF ファイルの依存となっている DTD ファイルも除外されます。ただし，DTD ファイルが複数の DDF ファイルの依存ファイルになっている場合は，DTD ファイルを参照している DDF ファイルがすべて除外されたときに，DTD ファイルが開発マネージャ上から除外されます。
- DTD ファイルは単独では除外できません。

8.3.3　DTD ファイルの変更

DDF ファイルが参照する DTD ファイルを変更する手順を次に示します。

1. 開発マネージャのウィンドウの［プロジェクト］メニューから［プロジェクトの設定］を選ぶ
   プロジェクト設定ダイアログボックスが表示されます。

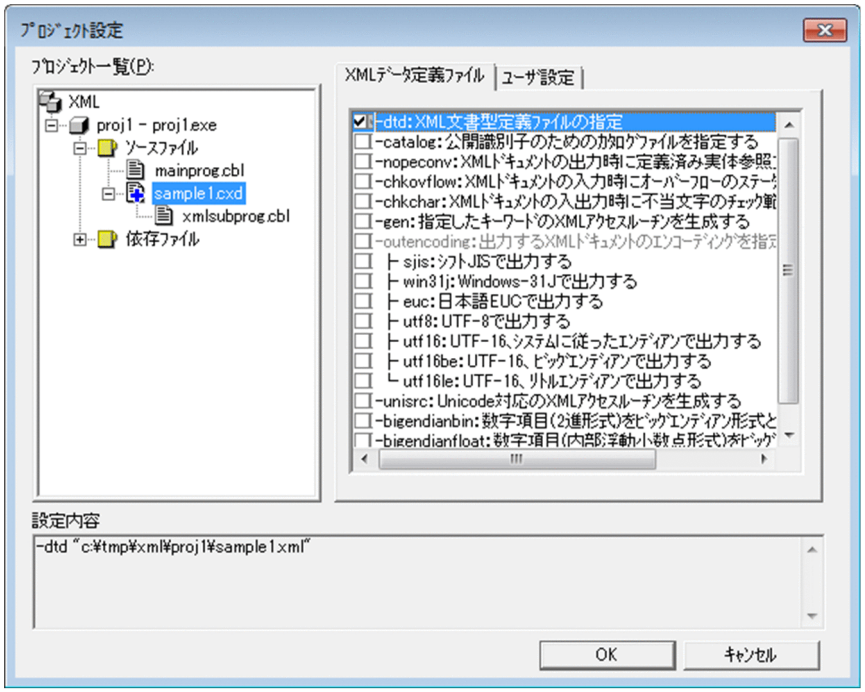
2. プロジェクト設定ダイアログボックスのプロジェクト一覧ツリービューで，DDF ファイルを選ぶ
   ［XML データ定義ファイル］タブが表示されます。

3. -dtd オプションのチェックボックスを選び，DTD ファイルを変更する

4. ［OK］ボタンを選ぶ
   設定した内容がプロジェクトに反映されます。

注意事項

- DTD ファイルは必須のため，-dtd オプションを外すことはできません。
- ［依存ファイルの設定］メニューからは登録・変更できません。



8.3.4　生成される COBOL ソースファイルの変更

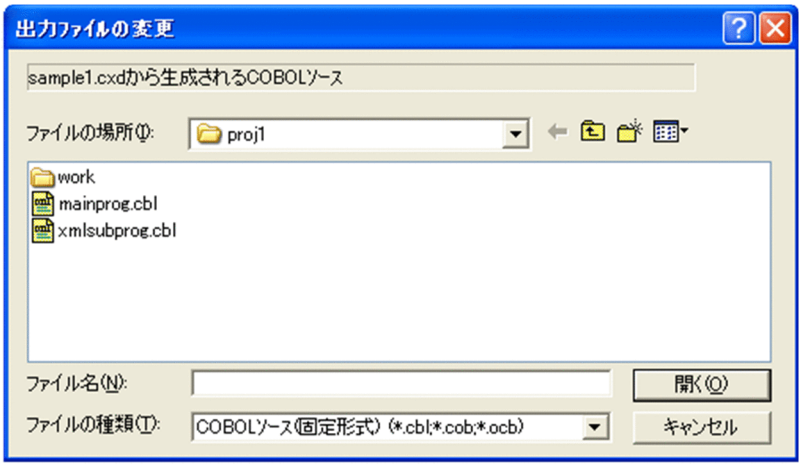
DDF ファイルから生成される COBOL ソースファイルを変更する手順を次に示します。

1. 開発マネージャのツリービューウィンドウで，出力先ファイルを変更したい DDF ファイルを選ぶ

2. ［ファイル］メニューから［出力ファイルの変更］を選ぶ

出力ファイルの変更ダイアログボックスが表示されます。

なお，ツリービューウィンドウの DDF ファイルを右クリックして［出力ファイルの変更］を選んでも，出力ファイルの変更ダイアログボックスが表示されます。



3. ファイルの種類ドロップダウンリストで「COBOL ソース（固定形式）（*.cbl,*.cob,*.ocb)」を指定する

4. DDF ファイルから生成される COBOL ソースファイル名を指定して［開く］ボタンを選ぶ

注意事項

変更した場合，出力ファイルに設定されていたオプションはクリアされます。

8.4　ビルド

開発マネージャでビルドするときの注意点を説明します。

8.4.1　生成される COBOL ソースのコンパイル

ビルドの前に，XML 連携機能が提供する登録集原文のあるフォルダを COBOL の環境変数 CBLLIB に設定しておく必要があります。

8.4.2　リンク

DDF ファイルを含むプログラムでは，XML 連携機能が提供するライブラリ cblxmlrt.lib をリンク時に指定する必要があります。開発マネージャはプロジェクトに DDF ファイルが含まれている場合リンク時に cblxmlrt.lib を指定してリンクするため，明示的に cblxmlrt.lib を指定する必要はありません。

8.5　新規作成

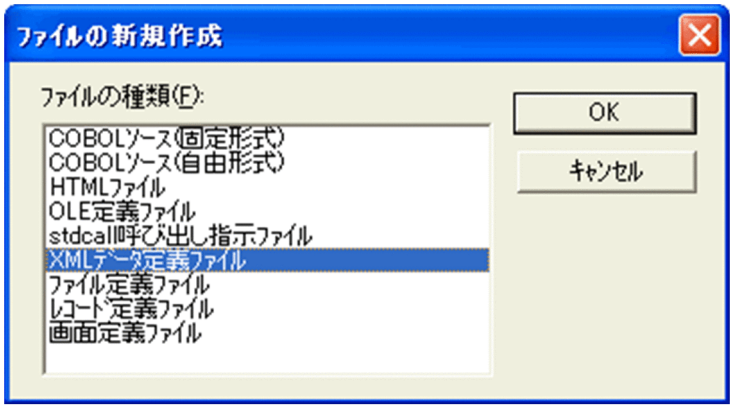
開発マネージャ上から DDF ファイルを新規に作成する手順を次に示します。

1. ［ファイル］メニューの［新規作成］を選ぶ

   ファイルの新規作成ダイアログボックスが表示されます。

2. 「XML データ定義ファイル」を選ぶ

   エディタが起動されます。



注意事項

エディタは，標準では COBOL エディタが起動されます。使用するエディタを変更する場合は，マニュアル「COBOL2002　操作ガイド」の開発マネージャのカスタマイズの説明を参照してください。

8.6 注意事項

- XML 連携機能では，生成される COBOL ソースについて，幾つかのコンパイラオプションを指定した場合，動作は保証しません。開発マネージャでは，これらはチェックされません。これらのコンパイラオプションが指定された場合，動作は保証しません。なお，動作を保証しないオプションは「6.2.1 コンパイルとリンケージ」の注意事項を参照してください。
- DDF ファイルのコンパイルでエラーが発生した場合，メッセージウィンドウに表示されているエラーメッセージをダブルクリックしてもエラーの発生したファイルを参照できません。開発マネージャのツリービューウィンドウから DDF ファイルを参照してください。
- 外部参照の XML ファイルは，ビルド後に開発マネージャのツリー上に自動的に登録されません。また，ユーザによる登録もできません。
- プロジェクトに同名の DDF ファイルを複数登録できません。また，すでにプロジェクトに登録されている COBOL ソース，およびほかのファイルの出力になっている COBOL ソースは，DDF ファイルから生成される COBOL ソースに指定できません。
- OLE プロジェクトには，DDF ファイルを登録できません。また，すでに DDF ファイルが登録されているプロジェクトを OLE プロジェクトに変更した場合，DDF ファイルはビルド対象になりません。

9 入出力時の拡張機能

この章では，XML ドキュメント入出力時の拡張機能について説明します。

9.1　入力時のオーバフローをステータスで返す機能

入力時のオーバフローをステータスで返す機能は，XML ドキュメントを入力する場合に，表 9-1 の条件に従って，オーバフローであれば CBLXML-RD-*Interface-BaseElement* アクセスルーチンでステータス 10 を返す機能です。

この機能を使用する場合は，COBOL 原始プログラムの生成時，cblxml コマンドに-chkovflow オプションを指定します。cblxml コマンドに-chkovflow オプションを指定しないで生成した COBOL 原始プログラムでは，XML ドキュメントの入力時にオーバフローの条件に該当した場合でもオーバフローのステータスを返しません。

表 9-1　入力時のオーバフローの条件

type 属性値	オーバフローの条件
alphanumeric national	size 属性に指定したけた数を超えた場合。
numeric packed binary	左側の 0 を削除した数値が size 属性に指定したけた数を超えた場合。
float double	システムで定義された浮動小数点の絶対値を超えた場合※。

注※

システムで定義された浮動小数点の絶対値については，ご使用のシステムのマニュアルを参照してください。

9.2　入出力時に不当な文字をチェックする機能

XML ドキュメントの入出力時に不当な文字をチェックする機能です。cblxml コマンドの-chkchar オプションを指定することでチェックされる不当な文字のチェック範囲を拡張できます。

XML ドキュメント出力時，表 9-2 に従って不当な文字をチェックします。XML ドキュメント入力時，表 9-3 に従って不当な文字をチェックします。

CBLXML-WR-*Interface-BaseElement* アクセスルーチンで XML ドキュメント出力時に不当な文字を見つけた場合，ステータス 11 を返します。該当部分の要素，属性には空の値が出力されます。CBLXML-RD-*Interface-BaseElement* アクセスルーチンで，XML ドキュメントの入出力時に不当な文字を見つけた場合，ステータス 8 を返します。

type 属性の値が「numeric」，「packed」，「binary」，「float」，および「double」のとき，数字，符号，小数点，"E"，および"e"以外の文字は無視して入力されます。なお，XML として不正な文字は，XML ドキュメントを開く際にチェックされ，XML アクセスルーチンのエラーステータス 110（CBLXML-XML-PARSE-FAIL）が返されます。XML 不正文字については，「付録 E.2(1)　XML 標準仕様での文字の範囲」を参照してください。

表 9-2　出力時にチェックする文字

type 属性値	チェックする文字（-chkchar オプション指定あり）	不当な文字・数値があった場合の出力結果
alphanumeric	• シフト JIS の場合 0x00～0x08，0x0b，0x0c，0x0e～0x1f，0x80，0xa0，0xfd～0xff とシフト JIS の多バイト文字以外 • 日本語 EUC の場合 0x00～0x08，0x0b，0x0c，0x0e～0x1f，0x80，0xa0，0xff と日本語 EUC コードの多バイト文字以外 • Unicode の場合（環境変数 CBLLANG が UNICODE） 0x00～0x08，0x0b，0x0c，0x0e～0x1f とコード範囲（1 バイト～4 バイト）以外	空の要素，属性を出力する。
national	• シフト JIS の場合 0x00～0x08，0x0b，0x0c，0x0e～0x1f，0x80，0xa0，0xfd～0xff とシフト JIS の多バイト文字以外 • 日本語 EUC の場合 0x00～0x08，0x0b，0x0c，0x0e～0x1f，0x80，0xa0，0xff と日本語 EUC コードの多バイト文字以外 • Unicode の場合（環境変数 CBLLANG が UNICODE） UCS-2（0x0009，0x000a，0x000d，0x0020～0x4dff，0x4e00～0x9fff，0xe000～0xf8ff，0xf900～0xfffd）の範囲以外	空の要素，属性を出力する。

type 属性値	チェックする文字（-chkchar オプション指定あり）	不当な文字・数値があった場合の出力結果
numeric packed	数字・符号以外（COBOL の字類条件での数字検査（IS NUMERIC）を満たさない文字）	空の要素，属性を出力する。
binary float double	－	結果は保証しない。

（凡例）

－：チェックしない。

表 9-3　入力時にチェックする文字

type 属性の値	チェックする文字		不当な文字・数値があった場合の入力結果
	-chkchar オプション指定なし	-chkchar オプション指定あり	
alphanumeric	－	－	結果は保証しない。
national	• Unicode 機能を使用しない場合：1 バイトの文字 • Unicode 機能を使用する場合：　－	• Unicode 機能を使用しない場合：1 バイトの文字 • Unicode 機能を使用する場合：　－	• 1 バイト文字は「＝」（げた記号）※に置き換える。 • その他の不当な文字の結果は保証しない。
numeric packed binary	－	• 数値以外 • sign="unsigned"指定時に負値を入力した場合 • けた数が 320 を超えた場合	結果は保証しない。
float double	－	• 数値以外 • けた数が 320 を超えた場合	結果は保証しない。

（凡例）

－：チェックしない。

注※　「＝」（げた記号）は次に示すコードになります。

- シフト JIS：0x81ac
- 日本語 EUC（UNIX の場合）：0xa2ae
- UTF-16（Unicode 機能を使用した場合）：0x3013

9.3　XML サービスルーチンを使用した機能

XML 連携機能には次のようなサービスルーチンが用意されています。

次に，XML サービスルーチンの一覧を示します。

- CBLXML-CREATE-XML-POINTER サービスルーチン
  空の XML ドキュメントのポインタを割り当てるサービスルーチンです。
- CBLXML-FREE-XML-POINTER サービスルーチン
  割り当てた XML ドキュメントのポインタを解放するサービスルーチンです。
- CBLXML-GET-ERROR サービスルーチン
  直前に実行した XML アクセスルーチンの詳細なエラー情報を取得するサービスルーチンです。
- CBLXML-READ-CATALOG-FILE サービスルーチン
  公開識別子で利用するカタログファイルを入力するサービスルーチンです。
- CBLXML-GET-NEXT-BE サービスルーチン
  入力または更新モードで開いた XML ドキュメントに対し，次に入力する BaseElement 要素の XML ドキュメント上での位置情報を取得するサービスルーチンです。
- CBLXML-SET-ENTITYLIMIT サービスルーチン
  XML ドキュメントの入力時に，エンティティ参照回数を制限するサービスルーチンです。

注意事項（Windows(x86)の場合）

- XML サービスルーチンを stdcall 呼び出し規約で呼び出すことはできません。
- XML サービスルーチンは C（cdecl）呼び出し規約で呼び出してください。

注意事項（Windows，Linux の場合）

-bigendianbin オプションを使用して XML アクセスルーチンを生成する場合，-BigEndian,Bin コンパイラオプションを指定してコンパイルしなければなりません。このとき，XML サービスルーチンの引数で指定する数字項目（2 進形式）がビッグエンディアン形式になるため，正常に動作しません。この場合は，引数となる数字項目（2 進形式）を COMP-5 で定義してください。

9.3.1　XML サービスルーチンの初期処理と終了処理

XML サービスルーチンを使用する機能のエラー情報取得機能や公開識別子では，XML ドキュメントを開く（CBLXML-OP-*Interface*，CBLXML-OB-*Interface*）アクセスルーチンの前に，XML サービスルーチンの初期処理として CBLXML-CREATE-XML-POINTER サービスルーチンを呼び出して空の XML ドキュメントのポインタを作成し，XML ドキュメントを閉じる（CBLXML-CL-*Interface*，CBLXML-CN-*Interface*）アクセスルーチンのあとに CBLXML-FREE-XML-POINTER サービスルーチンで XML ドキュメントのポインタを解放する必要があります。

次に，XML サービスルーチンの初期処理と終了処理となる CBLXML-CREATE-XML-POINTER サービスルーチンと CBLXML-FREE-XML-POINTER サービスルーチンを説明します。

(1) CBLXML-CREATE-XML-POINTER サービスルーチン

CBLXML-CREATE-XML-POINTER サービスルーチンは，空の XML ドキュメントのポインタを割り当てます。このサービスルーチンは，エラー情報取得機能や公開識別子のカタログファイルの入力で使用します。

形式

```
CALL 'CBLXML-CREATE-XML-POINTER' USING XML-POINTER.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	XML ドキュメントのポインタを受け取るポインタ項目を指定する。

（凡例）

○：サービスルーチンの呼び出し時，値を設定しておく項目

戻り値

戻り値	内容
0	正常終了した。
-1	操作を続けるためのメモリが不足した。

注

戻り値は，RETURN-CODE 特殊レジスタに設定されます。

規則

- XML ドキュメントのポインタ割り当てに成功した場合，CBLXML-FREE-XML-POINTER サービスルーチンで XML ドキュメントのポインタを解放しなければなりません。
- CBLXML-CREATE-XML-POINTER サービスルーチンは，XML ドキュメントを開くアクセスルーチン（CBLXML-OP-*Interface*，CBLXML-OB-*Interface*）の前に呼び出さなければなりません。
- CBLXML-CREATE-XML-POINTER サービスルーチンを呼び出すたびに，異なる空の XML ドキュメントのポインタを返します。

(2) CBLXML-FREE-XML-POINTER サービスルーチン

CBLXML-FREE-XML-POINTER サービスルーチンは，割り当てた XML ドキュメントのポインタを解放します。

形式

```
CALL 'CBLXML-FREE-XML-POINTER' USING XML-POINTER.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	CBLXML-CREATE-XML-POINTER サービスルーチンで割り当てた XML ドキュメントのポインタを受け取るポインタ項目を指定する。

（凡例）

○：サービスルーチンの呼び出し時，値を設定しておく項目

戻り値

戻り値	内容
0	正常終了した。
-1	CBLXML-FREE-XML-POINTER サービスルーチンの引数の値が不正である。
-2	XML ドキュメントを閉じていない状態で，CBLXML-FREE-XML-POINTER サービスルーチンを実行した。

注

戻り値は，RETURN-CODE 特殊レジスタに設定されます。

9.3.2 エラー情報の取得

エラー情報取得機能は，CBLXML-GET-ERROR サービスルーチンによって XML アクセスルーチンの詳細なエラー情報を取得する機能です。

XML アクセスルーチンのエラー情報を取得するには，XML ドキュメントを開く XML アクセスルーチンの前に CBLXML-CREATE-XML-POINTER サービスルーチンを呼び出して作成した空のポインタを用いて，XML ポインタを入力するアクセスモード"E"を指定し，XML ドキュメントを開く XML アクセスルーチンを呼び出さなければなりません。また，XML ドキュメントを閉じる XML アクセスルーチンのあとに CBLXML-FREE-XML-POINTER サービスルーチンを呼び出して CBLXML-CREATE-XML-POINTER サービスルーチンで作成した XML ポインタを解放しなければなりません。これらを行い，各 XML アクセスルーチンの直後に CBLXML-GET-ERROR サービスルーチンで呼び出すことでエラー情報を取得できます。

次に XML ドキュメントを開く CBLXML-OP-*Interface* アクセスルーチンのエラー情報の取得例を示します。

(エラー情報の取得)

```
  :
01 CBLXML-RETURN-CODE PIC 9(9) COMP.
01 FILE-NAME PIC X(255).
01 XML-FILE-NAME USAGE POINTER VALUE NULL.
01 XML-FILE-NAME-LENGTH PIC 9(9) COMP VALUE 0.
01 XML-MODE PIC X(16).
```

```
01 XML-POINTER USAGE POINTER VALUE NULL.
01 MESSAGE-BUFFER PIC X(255).
01 MESSAGE-BUFFER-LENGTH PIC 9(9) COMP VALUE 255.
01 MESSAGE-LENGTH PIC 9(9) COMP VALUE 0.
  :
PROCEDURE DIVISION.
START-MAIN.
  :
    CALL 'CBLXML-CREATE-XML-POINTER'
        USING XML-POINTER.
    IF RETURN-CODE NOT = 0 THEN
        EXIT PROGRAM
    END-IF.
*open
    MOVE 'WE' TO XML-MODE.
    MOVE 'output.xml' TO FILE-NAME.
    MOVE 10 TO XML-FILE-NAME-LENGTH.
    COMPUTE XML-FILE-NAME = FUNCTION ADDR(FILE-NAME).
    CALL 'CBLXML-OP-EXAMPLE'
        USING XML-FILE-NAME
              XML-FILE-NAME-LENGTH
              XML-MODE
              XML-POINTER
        RETURNING CBLXML-RETURN-CODE.

    IF CBLXML-RETURN-CODE NOT = 0 THEN
        CALL 'CBLXML-GET-ERROR'
            USING XML-POINTER
                  MESSAGE-BUFFER
                  MESSAGE-BUFFER-LENGTH
                  MESSAGE-LENGTH
        DISPLAY 'ERROR = ' MESSAGE-BUFFER
    ELSE
*write
        CALL 'CBLXML-WR-EXAMPLE-BE'
            USING XML-POINTER root
            RETURNING CBLXML-RETURN-CODE
*close
        CALL 'CBLXML-CL-EXAMPLE'
            USING XML-POINTER
            RETURNING CBLXML-RETURN-CODE
    END-IF.

    CALL 'CBLXML-FREE-XML-POINTER'
        USING XML-POINTER.
  :
```

(1) CBLXML-GET-ERROR サービスルーチン

直前に実行した XML アクセスルーチンの詳細なエラー情報を取得します。エラー情報の取得対象となる XML アクセスルーチンを次に示します。

(エラー情報の取得対象となる XML アクセスルーチン)

- CBLXML-OP-*Interface*

- CBLXML-OB-*Interface*
- CBLXML-RD-*Interface-BaseElement*
- CBLXML-WR-*Interface-BaseElement*
- CBLXML-CL-*Interface*
- CBLXML-CN-*Interface*

形式

```
CALL ’CBLXML-GET-ERROR’ USING XML-POINTER MESSAGE-BUFFER MESSAGE-BUFFER-LENGTH MESSAGE-LENGTH.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	XML ドキュメントのポインタを受け取るポインタ項目を指定する。
01 MESSAGE-BUFFER PIC X(n).	○	エラー情報を受け取る領域を英数字項目で指定する。
01 MESSAGE-BUFFER-LENGTH PIC 9(9) COMP.	○	エラー情報を受け取る領域の長さを 4 バイトの 2 進項目で指定する。
01 MESSAGE-LENGTH PIC 9(9) COMP.	△	受け取ったエラー情報の長さが設定される。この項目は 4 バイトの 2 進項目で指定する。

（凡例）

○：サービスルーチンの呼び出し時，値を設定しておく項目

△：サービスルーチンの完了時，値が設定される項目

戻り値

戻り値	内容
1	エラー情報の長さがエラー情報を受け取る領域の長さを超えた。
0	正常終了した。
-1	CBLXML-GET-ERROR サービスルーチンの引数の値が不正である。
-2	操作を続けるためのメモリが不足した。

注

戻り値は，RETURN-CODE 特殊レジスタに設定されます。

規則

- 受け取るエラー情報がない場合，MESSAGE-LENGTH に 0 が返されます。
- MESSAGE-BUFFER-LENGTH に指定した長さより，受け取るエラー情報の長さが長い場合は，MESSAGE-BUFFER-LENGTH に指定した長さでエラー情報を切り捨てます。ただし，切り捨て位置が多バイト文字の途中の場合は，MESSAGE-BUFFER-LENGTH に指定した長さより 1 バイト短くなって MESSAGE-LENGTH に設定されます。MESSAGE-BUFFER の多バイト文字で切り捨てられる領域には半角空白文字が設定されます。

- 各アクセスルーチンのエラー情報を取得する場合，CBLXML-OP-*Interface* アクセスルーチンとCBLXML-OB-*Interface* アクセスルーチンでアクセスモード"E"を指定しなければなりません。アクセスモード"E"を指定しない場合，CBLXML-OP-*Interface* アクセスルーチンと CBLXML-OB-*Interface* アクセスルーチンのステータスが 0 で正常終了しても CBLXML-GET-ERROR サービスルーチンでエラー情報を取得できません。
- Unicode 機能を設定した場合，引数 2 で受け取る XML パーサのエラーメッセージは，文字コードが UTF-8 の文字列が返ります。引数 3 に指定した長さ（1 バイト 1 文字）より，エラー情報の文字列が長い場合，引数 3 に指定した長さで文字列を切り捨てます。切る位置が文字を構成する多バイト列の途中の場合，該当する文字以降を文字単位に切り捨てます。

(2) XML アクセスルーチンのエラー情報

CBL-GET-ERROR サービスルーチンによって得られるエラー情報を表 9-4 に示します。また，エラー情報の***番号***は，表 9-5 に示す埋め字情報に従って埋め字に置き換わります。

表 9-4 エラー情報

ステータス	エラー情報	埋め字情報
0	（ステータス 0 はエラー情報を出力しない）	−
1	[1]△要素"***1***"を出力できない。	*** 1 ***: DTD
3	[3]△"***1***"を出力できない。	*** 1 ***: DTD
4	[4]△BaseElement 要素に対応する要素を見つけられない。	−
5	[5]△BaseElement 要素"***1***"を飛ばして，次の BaseElement 要素を入力できない。	*** 1 ***: DTD
6	[6]△要素"***2***"の選択要素に対して，***1***個選択したために出力を決定できない。	***1***:選択要素で出力しようとした個数 ***2***:選択要素の直下にある要素名
7	[7]△出力の＋付き要素の繰り返し回数に 0 が設定されている。	−
8	[8]△"***1***"に不当な文字がある。	*** 1 ***: Document
9	[9]△出力時のファイルまたはバッファを開けない。	−
10	[10]△"***1***"に入力した値がデータ項目の長さを超えた。	*** 1 ***: Document
11	[11]△"***1***"に不当な文字が設定されている。	*** 1 ***: DTD
12	[12]△XML ドキュメントを 1 回も更新していない。	−
13	[13]△更新の対象を指定しないで XML ドキュメントを更新した。	−
15	[15]△直前に同じ BaseElement 要素単位の入力を呼び出していない。	−
101	[101]△メモリを確保できない。	−
103	[103]△XML ドキュメントを開けない。	−
104	[104]△XML ドキュメントを閉じることができない。	−

ステータス	エラー情報	埋め字情報
105※1	[105]△アクセスモードに不当な値が設定された。	－
106	[106]△ファイルに出力できない。	－
109	[109]△再帰的に定義された要素は出力できない。	－
110	[110]△XML ドキュメントの解析でエラーが発生した。;***1***	*** 1 ***: XML パーサのメッセージ
111	[111]△BaseElement 要素に対応した要素が見つからない。	－
114	[114]△引数に無効なパラメタが指定された。	－
115	[115]△要素の出力中に指定したバッファ領域の長さに達した。	－
117	[117]△出力でファイル I／O エラーが発生した。	－
118	[118]△ドキュメントを閉じることができない。	－
119	[119]△一つの XML ポインタに対して複数の XML ドキュメントを開こうとした。	－
121	[121]△内部エラーが発生した。	－
122	[122]△-unisrc オプションの指定と環境変数 CBLLANG の設定に矛盾がある。	－
123※2	[123]△-unisrc オプションの指定と環境変数 LANG の設定に矛盾がある。	－
124※3	[124]△互換性がない古い XML アクセスルーチンを使用している。	－

（凡例）

△：半角空白文字

－：埋め字情報はありません。

注※1

アクセスモード"E"を指定した場合だけ取得できます。

注※2

AIX，Linux の場合だけ出力されます。

注※3

64bit 版 COBOL2002 の場合だけ出力されます。

表 9-5　埋め字情報

埋め字の種類	埋め字に出力する情報
DTD	埋め字に示される要素は「/」文字を区切り文字として，階層でルート要素から「ルート要素/要素/要素」を出力する。 属性は「@」文字を区切り文字として「要素@属性」で出力する。
Document	埋め字に示される要素は「/」文字を区切り文字として，階層でルート要素から「ルート要素/要素/要素」を出力する。

埋め字の種類	埋め字に出力する情報
	要素が繰り返しである場合，繰り返し順序を示す"[番号]"を要素に付ける。DTD で"*"または"+"によって繰り返しがあることが指定されている要素でも，XML ドキュメント中に 1 回しか現れない要素については，繰り返し順序を示す"[番号]"は付け加えられない。番号は 1 から開始し，取得できる要素の繰り返しの番号は，2,147,483,647 までである。それを超えた場合は，保証しない。属性は"@"文字を区切り文字として"要素@属性"で出力する。
XML パーサのメッセージ	XML パーサが返すメッセージを出力する。

9.3.3 公開識別子が指定された XML ドキュメント

XML 連携機能では，カタログファイルに公開識別子とファイル名を対応づけることで，外部 DTD や外部実体を取り込めます。

(1) カタログファイル

カタログファイルは，公開識別子とファイル名を対応づけするファイルです。カタログファイルは cblxml コマンドの-catalog オプション，または実行時に CBLXML-READ-CATALOG-FILE サービスルーチンで入力できます。

次に，カタログファイルの形式を示します。

(a) カタログファイルの形式

公開識別子を対応づける場合，区切り文字「->」の左側に公開識別子，右側にファイル名を記述します。

```
公開識別子->ファイル名
```

開始文字「{」と終了文字「}」の間の文字列をコメントとみなします。

```
{ コメント }
```

DTD の公開識別子とカタログファイルを対応づけた例を次に示します。

(UNIX の場合)

(DTD の例)

```
          :
<!DOCTYPE root   PUBLIC "-//HITACHI//DTD test//EN" "/home/project1/myDTD.xml">
<!ENTITY  letter PUBLIC "-//HITACHI//ENT letter 1.0//EN" "/home/project1/letter.xml">
          :
```

(カタログファイルの例)

```
-//HITACHI//DTD test//EN->/home/local1/abc.xml
-//HITACHI//ENT letter 1.0//EN->/home/local1/xyz.xml
```

（Windows の場合）

（DTD の例）

```
          :
<!DOCTYPE root   PUBLIC "-//HITACHI//DTD test//EN" "C:¥home¥project1¥myDTD.xml">
<!ENTITY  letter PUBLIC "-//HITACHI//ENT letter 1.0//EN" "C:¥home¥project1¥letter.xml"
>
          :
```

（カタログファイルの例）

```
-//HITACHI//DTD test//EN->C:¥home¥local1¥abc.xml
-//HITACHI//ENT letter 1.0//EN->C:¥home¥local1¥xyz.xml
```

(b) 公開識別子

公開識別子は，次の文字や数字で構成できます。

```
#x20  #xD  #xA  a～z  A～Z  0～9  -  '  (  )  +  ,  .  /  :  =  ?  ;  !  *  #  @  $  _  %
```

カタログファイルで指定できる公開識別子は，1,024 バイト以下で指定してください。1,024 バイトを超えた場合，cblxml コマンドによる COBOL 原始プログラム生成時にエラーとなります。CBLXML-READ-CATALOG-FILE サービスルーチンは，戻り値として-2 を返します。

(c) ファイル名

ファイル名として使用できる文字を指定できます。

ファイル名は絶対パスまたは相対パスで指定します。ファイル名を相対パスで指定した場合，その相対パスはプロセスのカレントディレクトリ（フォルダ）からの相対パスとなります。

ファイル名は 255 バイト以下で指定してください。255 バイトを超えた場合，cblxml コマンドによる COBOL 原始プログラム生成時にエラーとなります。CBLXML-READ-CATALOG-FILE サービスルーチンは，戻り値として-2 を返します。

(d) 注意事項

- 公開識別子の前後にある空白文字（スペース，タブ，および改行）は削除されます。
- 公開識別子の中に現れる連続する空白文字（スペース，タブ，および改行）は正規化され一つの 1 バイトスペースとみなします。
- 公開識別子を対応づけた行の右側にコメントを記述できません。
- コメントの開始文字「{」と終了文字「}」を示す文字は，複数行にわたって指定できます。
- （Windows の場合）カタログファイルはシフト JIS で記述します。
  （UNIX の場合）カタログファイルは環境変数 LANG に従って，シフト JIS，日本語 EUC，または UTF-8 で記述します。

(2) カタログファイルの使い方

公開識別子を使用する場合，カタログファイルで公開識別子とファイル名の対応づけを解決します。次に，cblxml コマンド，および実行時でのカタログファイルの指定方法を示します。

(a) cblxml コマンド

cblxml コマンドに指定する DTD に公開識別子がある場合，cblxml コマンドの-catalog オプションでカタログファイルを指定します。カタログファイルには相対パス，絶対パスも指定できます。次に例を示します。

cblxml コマンドでのカタログファイルの指定例

```
cblxml a.cxd -dtd a.xml -o a.cbl -catalog mycatalog.cxc
```

注意事項

- カタログファイルの拡張子には「.cxc」を指定します。
- カタログファイルが見つからない場合は，-catalog オプションに指定したカタログファイルのオープン処理中にエラーとなります。
- カタログファイルのフォーマット不正などで入力できない場合は，エラーとなります。
- -catalog オプションで入力したカタログファイルの情報は，実行時には有効となりません。

(b) 実行時

公開識別子を有効にするには，XML ドキュメントを開く前にカタログファイルを読み込む必要があります。そのため，CBLXML-READ-CATALOG-FILE サービスルーチンを，CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アクセスルーチンの前に呼び出さなければなりません。

次にカタログファイルを使う場合の XML アクセスルーチン，および XML サービスルーチンの呼び出し順序の例を示します。

カタログファイルを使う場合の呼び出し順序

1. CBLXML-CREATE-XML-POINTER
   XML ポインタを作成する。
2. CBLXML-READ-CATALOG-FILE
   カタログファイルの情報を入力する。
3. CBLXML-OP-*Interface*
   アクセスモード「E」を追加して XML ドキュメントを開く。
4. CBLXML-RD-*Interface*-BaseElement
   XML ドキュメントを入力する。
5. CBLXML-CL-*Interface*
   XML ドキュメントを閉じる。

6. CBLXML-FREE-XML-POINTER

XML ポインタを解放する。

注意事項

カタログファイルの入力に失敗した場合，CBLXML-OP-*Interface* アクセスルーチンは DTD に記述した公開識別子に対応するシステム識別子を公開識別子の代わりに参照します。システム識別子のファイルを入力できない場合は，CBLXML-OP-*Interface* アクセスルーチンがステータス 110 を返します。

(3) CBLXML-READ-CATALOG-FILE サービスルーチン

CBLXML-READ-CATALOG-FILE サービスルーチンは，公開識別子で利用するカタログファイルを設定します。

形式

```
CALL ’CBLXML-READ-CATALOG-FILE’ USING XML-POINTER CATALOG-FILE CATALOG-FILE-LENGTH.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	XML ドキュメントのポインタを受け取るポインタ項目を指定する。
01 CATALOG-FILE PIC X(n).	○	カタログファイル名の領域を英数字項目で指定する。
01 CATALOG-FILE-LENGTH PIC 9(9) COMP.	○	カタログファイル名の長さを 4 バイトの 2 進項目で指定する。

（凡例）

○：サービスルーチンの呼び出し時，値を設定しておく項目

戻り値

戻り値	内容
0	正常終了した。
-1	カタログファイルが見つからない。または，カタログファイルが読み込めない。
-2	カタログファイルの形式が不正。
-3	操作を続けるためのメモリが不足した。

注

戻り値は，RETURN-CODE 特殊レジスタに設定されます。

規則

- CBLXML-READ-CATALOG-FILE サービスルーチンは XML ドキュメントを開くアクセスルーチン（CBLXML-OP-*Interface*，CBLXML-OB-*Interface*）より前に呼び出してください。
- カタログファイル名を相対パスで指定した場合，その相対パスはプロセスのカレントディレクトリ（フォルダ）からの相対パスとなります。

- 環境変数 CBLLANG に UNICODE を設定した場合，CATALOG-FILE に指定されたファイル名は，UTF-8 をシフト JIS に変換した文字列でアクセスします。ただし，Linux の場合は変換しないで，UTF-8 の文字列でアクセスします。

9.3.4　次に入力する BaseElement 要素の位置を取得する機能

CBLXML-GET-NEXT-BE サービスルーチンは，入力または更新モードで開いた XML ドキュメントに対し，次に入力する BaseElement 要素の XML ドキュメント上での位置情報を取得できます。位置情報は"/"文字を区切り文字としての階層で，ルート要素から BaseElement 要素までを設定します。BaseElement 要素が繰り返しである場合，繰り返し順序を示す"[番号]"が要素に付け加えられます。DTD で"*"または"+"によって繰り返しがあることが指定されている要素でも，XML ドキュメント中に 1 回しか現れない要素については，繰り返し順序を示す"[番号]"は付け加えられません。番号は 1 から開始し，2,147,483,647 までです。番号の範囲を超えた場合は，保証しません。

（位置情報の例）

```
/ルート要素/要素/BaseElement要素[番号]
```

形式

```
CALL ’CBLXML-GET-NEXT-BE’ USING XML-POINTER BE-LOCALE-BUFFER BE-LOCALE-BUFFER-LENGTH BE-LOCALE-LENGTH.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	XML ドキュメントのポインタを受け取るポインタ項目を指定する。
01 BE-LOCALE-BUFFER PIC X(n).	○	BaseElement 要素の位置情報を受け取る領域を英数字項目で指定する。
01 BE-LOCALE-BUFFER-LENGTH PIC 9(9) COMP.	○	BaseElement 要素の位置情報を受け取る領域の長さを 4 バイトの二進項目で指定する。
01 BE-LOCALE-LENGTH PIC 9(9) COMP.	△	受け取った位置情報の長さが設定される。この項目は 4 バイトの二進項目で指定する。

（凡例）

○：サービスルーチンの呼び出し時，値を設定しておく項目

△：サービスルーチンの完了時，値が設定される項目

戻り値

戻り値	内容
3	次に入力する BaseElement 要素がない。
2	XML ドキュメントを開いていない状態で位置情報を取得した。または出力モードで開いた XML ドキュメントの位置情報を取得した。

戻り値	内容
1	BaseElement 要素の位置情報の長さが位置情報を受け取る領域の長さを超えた。
0	正常終了した。
-1	CBLXML-GET-NEXT-BE サービスルーチンの引数の値が不正。
-2	操作を続けるためのメモリが不足した。

注

戻り値は，RETURN-CODE 特殊レジスタに設定されます。

規則

- BE-LOCALE-BUFFER-LENGTH に指定した長さより，受け取る BaseElement 要素の位置情報の長さが長い場合は，BE-LOCALE-BUFFER-LENGTH に指定した長さで位置情報を切り捨てます。ただし，切り捨て位置が多バイト文字の途中に当たる場合は，BE-LOCALE-BUFFER-LENGTH に指定した長さより 1 バイト短くなって BE-LOCALE-LENGTH に設定されます。BE-LOCALE-BUFFER の多バイト文字で切り捨てられる領域には半角空白文字が設定されます。
- 環境変数 CBLLANG に UNICODE を設定した場合，BE-LOCALE-BUFFER で受け取る BaseElement 要素の位置情報は，シフト JIS を UTF-8 に変換後の文字列が返ります。BE-LOCALE-BUFFER-LENGTH に指定した長さ（1 バイト 1 文字）より，位置情報の文字列が長い場合，BE-LOCALE-BUFFER-LENGTH に指定した長さで文字列を切り捨てます。切り捨て位置が多バイト文字を構成するバイト列の途中に当たる場合は，該当する文字以降を文字単位に切り捨て，1～3 バイト短くなって BE-LOCALE-LENGTH に設定されます。BE-LOCALE-BUFFER の多バイト文字で切り捨てられる領域には半角空白文字が設定されます。

次に入力する BaseElement 要素の位置を取得し，該当する BaseElement 要素を入力する例を次に示します。

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
  <!ELEMENT root (group1 | group2 | group3)>
  <!ELEMENT group1 (item1)>
  <!ELEMENT group2 (item1)>
  <!ELEMENT item1 (#PCDATA)>
  <!ELEMENT group3 (item3)>
  <!ELEMENT item3 (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
 <BaseElement elemName="item1">
  <Item elemName="item1" size="10"/>
 </BaseElement> <BaseElement elemName="item3">
  <Item elemName="item3" size="10"/>
```

```
 </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 item1 PIC X(10).
01 item3 PIC X(10).
```

次の XML ドキュメント data.xml を入力し，item3 要素の内容"ABC"を読み込みます。

(入力データ)

```
  <?xml version="1.0" encoding="Shift_JIS"?>
  <root>
    <group3>
      <item3>ABC</item3>
    </group3>
  </root>
```

(COBOL プログラムのコーディング例)

```
    :
 01 CBLXML-RETURN-CODE PIC 9(9) COMP.
 01 FILE-NAME PIC X(255).
 01 XML-FILE-NAME USAGE POINTER.
 01 XML-FILE-NAME-LENGTH PIC 9(9) COMP VALUE 255.
 01 XML-MODE PIC X(16).
 01 XML-POINTER USAGE POINTER.
 01 BE-LOCALE-BUFFER PIC X(255).
 01 BE-LOCALE-BUFFER-LENGTH PIC 9(9) COMP VALUE 255.
 01 BE-LOCALE-LENGTH PIC 9(9) COMP VALUE 0.
 PROCEDURE DIVISION.
 START-MAIN.
    :
* open
     MOVE 'R' TO XML-MODE.
     MOVE 'data.xml' TO FILE-NAME.
     MOVE 8 TO XML-FILE-NAME-LENGTH.
     COMPUTE XML-FILE-NAME = FUNCTION ADDR(FILE-NAME).
     CALL 'CBLXML-OP-EXAMPLE'
          USING XML-FILE-NAME
                XML-FILE-NAME-LENGTH
                XML-MODE
                XML-POINTER
          RETURNING CBLXML-RETURN-CODE.
     IF CBLXML-RETURN-CODE NOT = 0 THEN
         DISPLAY 'ERROR - CBLXML-OP-EXAMPLE, STATUS = '
         CBLXML-RETURN-CODE
     END-IF.
* read
     CALL 'CBLXML-GET-NEXT-BE'     …1.
         USING XML-POINTER
               BE-LOCALE-BUFFER
               BE-LOCALE-BUFFER-LENGTH
               BE-LOCALE-LENGTH.
     IF RETURN-CODE = 0 THEN
        IF BE-LOCALE-BUFFER = '/root/group1/item1' OR
           BE-LOCALE-BUFFER = '/root/group2/item1' THEN
```

```
            CALL 'CBLXML-RD-EXAMPLE-item1'    …2.
                USING XML-POINTER   item1
                RETURNING CBLXML-RETURN-CODE
            IF CBLXML-RETURN-CODE NOT = 0 THEN
               DISPLAY 'ERROR - READ item1, STATUS = '
               CBLXML-RETURN-CODE
            END-IF

            DISPLAY 'BE-LOCALE-BUFFER = ' BE-LOCALE-BUFFER
            DISPLAY 'item1 = ' item1
         ELSE
         IF BE-LOCALE-BUFFER = '/root/group3/item2' THEN
            CALL 'CBLXML-RD-EXAMPLE-item3    …3.
                USING XML-POINTER   item3
                RETURNING CBLXML-RETURN-CODE
            IF CBLXML-RETURN-CODE NOT = 0 THEN
                DISPLAY 'ERROR - READ item3, STATUS = '
                CBLXML-RETURN-CODE
            END-IF

            DISPLAY 'BE-LOCALE-BUFFER = ' BE-LOCALE-BUFFER
            DISPLAY 'item3 = ' item3
        END-IF
     END-IF.

* close
       CALL 'CBLXML-CL-EXAMPLE'
           USING XML-POINTER
           RETURNING CBLXML-RETURN-CODE.
       IF CBLXML-RETURN-CODE NOT = 0 THEN
           DISPLAY 'ERROR - CBLXML-CL-EXAMPLE, STATUS = '
           CBLXML-RETURN-CODE
       END-IF.
     :
```

(説明)

1. 次に入力する BaseElement 要素の位置を取得します。
2. item1 要素の内容を入力します。
3. item3 要素の内容を入力します。

9.3.5 文字エンコーディングが指定された XML ドキュメント

(1) CBLXML-SET-ENCODING サービスルーチン

出力する XML ドキュメントの文字エンコーディングは，CBLXML-SET-ENCODING サービスルーチンで指定できます。

形式

```
CALL 'CBLXML-SET-ENCODING' USING XML-POINTER ENCODING-CODE.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	XML ドキュメントのポインタを受け取るポインタ項目を指定する。
01 ENCODING-CODE PIC 9(9) COMP.	○	文字エンコーディングフラグを設定する領域を 4 バイトの 2 進項目で指定する。指定する値は，XML アクセス用データ定義（CBLXMLRC.cbl）に登録されている文字エンコーディングフラグの値でなければならない。

（凡例）

○：サービスルーチンの呼び出し時，値を設定しておく項目

戻り値

戻り値	内容
1	文字エンコーディングが無効である。
0	正常終了した。
-1	CBLXML-SET-ENCODING サービスルーチンの引数の値が不正である。
-2	文字エンコーディングを変えることができない。

注

戻り値は，RETURN-CODE 特殊レジスタに設定されます。

表 9-6　文字エンコーディングフラグ

登録集原文"CBLXMLRC.cbl"の定義名（78 レベル）	文字エンコーディングフラグ値	文字エンコーディングの意味
CBLXML-EUC-ENCODING	1	EUC-JP
CBLXML-SJIS-ENCODING	2	Shift_JIS
CBLXML-UTF8-ENCODING	3	UTF-8
CBLXML-UTF16-ENCODING	4	Windows，Linux の場合：UTF-16，リトルエンディアン AIX の場合：UTF-16，ビッグエンディアン
CBLXML-UTF16BE-ENCODING	5	UTF-16，ビッグエンディアン
CBLXML-UTF16LE-ENCODING	6	UTF-16，リトルエンディアン
CBLXML-WIN31J-ENCODING	7	Windows-31J

規則

- CBLXML-SET-ENCODING サービスルーチンは，CBLXML-CREATE-XML-POINTER サービスルーチン，CBLXML-OP-*Interface* アクセスルーチン，または CBLXML-OB-*Interface* アクセスルーチンで XML ドキュメントのポインタを作成後，CBLXML-WR-*Interface*-*BaseElement* アクセスルーチンを呼び出す前に実行してください。

- XML アクセスルーチンのアクセスモード'R'を指定して開いた XML ドキュメントに対しては，CBLXML-SET-ENCODING サービスルーチンの呼び出しは無効となり，戻り値として 1 を返します。
- CBLXML-SET-ENCODING サービスルーチンを複数回呼び出した場合，最後の呼び出しが有効となります。
- XML アクセスルーチンのアクセスモード'W'，または'U'を指定して開いた XML ドキュメントに対して，CBLXML-WR-*Interface-BaseElement* アクセスルーチンを呼び出したあと，CBLXML-CL-*Interface* アクセスルーチン，または CBLXML-CN-*Interface* アクセスルーチンを呼び出す前に CBLXML-SET-ENCODING サービスルーチンを実行した場合は出力エンコーディングを変えることはできません。この場合，CBLXML-SET-ENCODING サービスルーチンは戻り値として-2 を返します。

9.3.6　エンティティ参照回数を制限する機能

(1)　CBLXML-SET-ENTITYLIMIT サービスルーチン

XML ドキュメントの入力時にエンティティ参照回数を制限するときは，CBLXML-SET-ENTITYLIMIT サービスルーチンを呼び出して制限値を設定します。

形式

```
CALL 'CBLXML-SET-ENTITYLIMIT' USING XML-POINTER ENTITY-LIMIT.
```

引数

引数のデータ型	指定	説明
01 XML-POINTER USAGE POINTER.	○	XML ドキュメントのポインタを受け取るポインタ項目を指定します。
01 ENTITY-LIMIT PIC 9(9) COMP.	○	最大のエンティティ参照回数を指定する領域を 4 バイトの 2 進項目で指定します。指定できる値の範囲は，0～2,147,483,647 です。

（凡例）

○：サービスルーチンの呼び出し時，値を設定しておく項目

戻り値

戻り値	内容
0	正常終了した。
-1	CBLXML-SET-ENTITYLIMIT サービスルーチンの引数の値が不正である。

規則

- CBLXML-SET-ENTITYLIMIT サービスルーチンは，CBLXML-CREATE-XML-POINTER サービスルーチンで XML ドキュメントのポインタを作成したあとで，呼び出してください。

- CBLXML-SET-ENTITYLIMIT サービスルーチンの設定は，XML ドキュメントの入力時に有効になります。CBLXML-SET-ENTITYLIMIT サービスルーチンを呼び出した XML ドキュメントのポインタを指定して，XML ドキュメントを開くアクセスルーチン（CBLXML-OP-*Interface*，CBLXML-OB-*Interface*）を呼び出してください。
- 引数 ENTITY-LIMIT の値が範囲外の場合，制限値を設定していないとみなし，参照回数は制限されません。CBLXML-SET-ENTITYLIMIT サービスルーチンの戻り値は-1 になります。
- 環境変数 CBLXML_ENTITYLIMIT が指定されている状態で CBLXML-SET-ENTITYLIMIT サービスルーチンを呼び出した場合，CBLXML-SET-ENTITYLIMIT サービスルーチンに指定した値が有効になります。

9.4 小数点以下のけた落ちを判定する機能

XML ドキュメントの入力時に数字項目に対応づけた要素および属性の入力する値の小数点以下のけた数がDDF で指定した COBOL データ項目の小数点以下のけた数より長く，けたの切り落としが発生したときにオーバフローが発生したとみなしたい場合は環境変数 CBLXML_CHKUFLOW に YES を指定します。オーバフローを判定するためには環境変数 CBLXML_CHKUFLOW と同時に-chkovflow オプション，overflowValue 属性，または accessInfo 属性のどちらかを指定しなければなりません。

数字項目とは，DDF で Item 要素の type 属性に"numeric"，"packed"，または"binary"のどれかを指定した項目です。

形式

(UNIX の場合)

sh（B シェル）の場合

```
CBLXML_CHKUFLOW=YES
export CBLXML_CHKUFLOW
```

(Windows の場合)

```
set CBLXML_CHKUFLOW=YES
```

けたの切り落としが発生する例

(DTD の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
<!ELEMENT root (item1)>
<!ELEMENT item1 (#PCDATA)>
]>
<root/>
```

(DDF の例)

```
<?xml version="1.0" encoding="Shift_JIS"?>
<Interface interfaceName="EXAMPLE">
  <BaseElement elemName="root">
  <Item elemName="item1" type="numeric" size="9" fractionalDigits="3"/>
  </BaseElement>
</Interface>
```

(生成される COBOL データ項目)

```
01 item1 PIC 9(6)V9(3).
```

(入力される XML ドキュメント（けたの切り落としが発生する値))

```
<?xml version="1.0" encoding="Shift_JIS"?>

<root>
```

```
  <item1>123.4567</item1>※
</root>
```

注※

DDF で指定した COBOL データ項目の小数点以下のけた数を超えています。

注意事項

- 環境変数 CBLXML_CHKUFLOW を指定しても-chkovflow オプション，overflowValue 属性，または accessInfo 属性のどちらかを指定しない場合は，それぞれの条件で-chkovflow オプション指定時の CBLXML-RD-*Interface-BaseElement* アクセスルーチンのステータス 10（CBLXML-VALUE-OVERFLOW）や overflowValue 属性の設定値，accessInfo 属性でのアクセス情報フラグの CBLXML-FLAG-OVERFLOW が設定されません。小数点以下のけた落ちを判定する場合は環境変数 CBLXML_CHKUFLOW と同時に-chkovflow オプション，overflowValue 属性，または accessInfo 属性のどちらかを必ず指定してください。
- 小数点以下のけた落ちの判定は，有効数字以外の 0 を削除した値で判定します。

（例）

COBOL データ項目 9(4)V9(1)に対して，XML の入力値が「123.400」である場合，右端の「00」を削除した有効数字 123.4 で判定を行います。このため，けたの切り落としが発生しません。

付録

付録 A　データ定義言語（DDL）の文法形式

(1) Interface 要素

<Interface interfaceName="*インタフェース名*"
[accessInfo="*yes*|*no*"] >
BaseElement 要素 …
</Interface>

(2) BaseElement 要素

<BaseElement elemName="*XML 要素の名称*"
[cobName="*XML アクセスルーチンの名称*"]
[accessInfo="*yes*|*no*"]
[nameOfBaseVar="*XML アクセス用データ定義の名称*"] >
{*Group 要素* | *Item 要素* | *AttrItem 要素*}
</BaseElement>

(3) Group 要素

<Group
[elemName="*XML の要素名*"]
[cobName="*Group 要素に対応する名称*"]
[nameOfFlagVar="*アクセス情報フラグの名称*"]
[nameOfGroupVar="*入出力データ情報項目の名称*"]
[update="*yes*"] >
{*Group 要素* | *Array 要素* | *Item 要素* | *AttrItem 要素*} …
</Group>

(4) Item 要素

<Item elemName="*XML の要素名*"
[cobName="*Item 要素に対応する名称*"]
[type="*type 属性値*"]
[size="*けた数*"]
[fractionalDigits="*けた数*"]
[emptyValue="*type 属性値に対応した COBOL の定数*"]
[sign="*符号種別*"]
[trim="*yes* | *no*"]
[nameOfFlagVar="*アクセス情報フラグの名称*"]
[nameOfLengthVar="*データ長の名称*"]
[verbatim="*yes*|*no*"]
[update="*yes*"]

[emptyContentValue="*type 属性値に対応した COBOL の定数*"]
[invalidCharValue="*type 属性値に対応した COBOL の定数*"]
[overflowValue="*type 属性値に対応した COBOL の定数*"]
/>

(5) Array 要素

<Array max="*要素の最大繰り返し回数*"
[nameOfCountVar="*繰り返し入出力数の名称*"]
[nameOfTotalVar="*繰り返し全要素数の名称*"]
[countVar="*yes*|*no*"] >
{*Group 要素* | *Item 要素* | *AttrItem 要素*}
</Array>

(6) AttrItem 要素

<AttrItem elemName="*XML の要素名*"
attrName="*属性名*"
[cobName="*AttrItem 要素に対応する名称*"]
[type="*type 属性値*"]
[size="*けた数*"]
[fractionalDigits="*けた数*"]
[emptyValue="*type 属性値に対応した COBOL の定数*"]
[sign="*符号種別*"]
[trim="*データ項目の整形の有無*"]
[nameOfFlagVar="*アクセス情報フラグの名称*"]
[nameOfLengthVar="*データ長の名称*"]
[emptyContentValue="*type 属性値に対応した COBOL の定数*"]
[invalidCharValue="*type 属性値に対応した COBOL の定数*"]
[overflowValue="*type 属性値に対応した COBOL の定数*"]
/>

付録 B　XML 連携機能，XML 連携機能の実行ライブラリで使用するファイル

使用するファイルの一覧を表 B-1 に示します。

表 B-1　XML 連携機能，XML 連携機能の実行ライブラリで使用するファイル

ファイル種別	拡張子	内容	出力元	入力先
共用ライブラリ	.so※1 .sl※1 .a※4	オブジェクトプログラムのライブラリを格納するファイル。	－	リンカ
DLL ファイル	.dll※2	DLL（ダイナミックリンクライブラリ）を格納するファイル。	リンカ	－
ライブラリファイル	.lib	インポートライブラリファイル。	－	リンカ
COBOL ソースファイル	.cbl ほか※3	COBOL 原始プログラムを格納するファイル。	－	コンパイラ
XML アクセス用データ定義	.cbl	XML ドキュメントへのアクセスに必要な COBOL のデータ定義を格納するファイル。 拡張子は.cbl でなければならない。	cblxml	コンパイラ
XML アクセス用ステータス定義	.cbl	XML アクセスルーチンのステータス名称を定義する登録集原文。	－	コンパイラ
XML アクセスルーチン	.cbl	XML ドキュメントにアクセスする COBOL 副プログラムを格納するファイル。 拡張子は.cbl でなければならない。	cblxml	コンパイラ
カタログファイル	.cxc	公開識別子とファイルの対応づけを定義するファイル。 拡張子は.cxc でなければならない。	－	cblxml，実行時
データ定義ファイル（DDF ファイル）	.cxd	COBOL プログラムからアクセスする XML ドキュメント中の要素と，COBOL データ項目との対応づけを指定するファイル。 拡張子は.cxd でなければならない。	－	cblxml
外部 DTD ファイル	.xml	DTD の DOCTYPE 宣言の実体を記述したファイル。	－	cblxml
オブジェクトファイル	.o※1 .obj※2	コンパイルの結果であるオブジェクトプログラムを格納するファイル。	コンパイラ	リンカ
文書型定義ファイル（DTD ファイル）	.xml .dtd※2	XML ドキュメントで使用するマークづけ要素と要素の構成を定義するファイル。	－	cblxml

ファイル種別	拡張子	内容	出力元	入力先
		拡張子は.xml または.dtd でなければならない。		
XML ドキュメント	.xml	XML 対応 COBOL プログラムのアクセス対象となる XML ファイル。	実行時	実行時
実行可能ファイル	.exe	コンパイル，リンクをして実行可能になったプログラムを格納するファイル。	リンカ	－

（凡例）

－：該当しない。

注

出力元，入力先の欄の略称の意味は次のとおりです。

cblxml：cblxml コマンド

コンパイラ：COBOL2002 コンパイラ（ccbl2002 コマンド）

リンカ：リンカ（AIX の場合 xlC コマンド，Linux の場合 g++コマンド，Windows の場合 LINK コマンド）

実行時：XML アクセス用実行時ライブラリ

注※1

UNIX の場合だけで有効

注※2

Windows の場合だけで有効

注※3

目的に応じて次の拡張子を使用します。

- 固定形式正書法で書かれた原始プログラムをコンパイルする場合
  .cbl，.cob，.ocb，または COBOL の環境変数 CBLFIX で指定した拡張子
- 自由形式正書法で書かれた原始プログラムをコンパイルする場合
  .cbf，.ocf，または COBOL の環境変数 CBLFREE で指定した拡張子

注※4

AIX の場合だけで有効

付録 C　制限事項

制限事項について説明します。

付録 C.1　XML 連携機能，XML 連携機能の実行ライブラリの制限事項

(1)　DTD，DDF の制限事項

(a)　XML の仕様

- 名前空間（ネームスペース）は使用できません。
- XML 宣言，文書の型宣言，要素宣言，属性リスト宣言，および実体宣言以外の宣言は，使用できません。それぞれの宣言の注意事項は，「2.1.2　XML 連携機能で扱える DTD の形式」を参照してください。
- elemName 属性に指定した名称について，XML 規格のチェックはしていません。

(b)　COBOL の仕様

- 再帰的な構造の XML 要素を，COBOL 集団項目に対応づけることはできません。
- 次のような構成の混合内容は，DDL で対応づけられません。

```
<!ELEMENT 要素A (#PCDATA | 要素B)*>
```

- DTD で，グループ内に同じ XML 要素を記述できません。

```
<!ELEMENT 要素A (要素B, 要素C, 要素B)>
```

(2)　製品の制限事項

- XML アクセスルーチンの引数に指定するポインタ項目に対して，NULL 値や不当なアドレスを指定した場合，動作は保証しません。
- XML 対応 COBOL プログラムでは，2GB を超える XML ドキュメントを扱えません。
- XML ドキュメントにアクセスすると，メモリ所要量は実際のファイルサイズより大幅に大きくなる場合があります。32bit 版アプリケーションでは，一つのプロセスで扱える仮想アドレス空間の制限が 2GB のため，実際のファイルサイズが 2GB 未満であっても，XML 連携機能が使用するメモリ所要量が 2GB を超えると，メモリ不足で実行時エラーとなります。アクセスする XML ドキュメントを分割するなど，メモリ所要量が 2GB を超えないようにしてください。

付録 C.2　COBOL2002 との連携での制限事項

- COBOL2002 コンパイラは，COBOL 原始プログラムが例外の伝播を抑止する条件に該当している場合，例外の伝播を自動的に無効とします。XML 連携機能によって生成された COBOL 原始プログラム

（XML アクセスルーチン）は，COBOL2002 コンパイラが例外の伝播を抑止する条件に該当しているので，例外の伝播は無効になります。したがって，-Compati85,NoPropagate オプションを指定する必要はありません。例外の伝播を抑止する条件の詳細については，マニュアル「COBOL2002　使用の手引　手引編」または「COBOL2002　ユーザーズガイド」を参照してください。

- XML 連携機能が生成する COBOL 原始プログラムをコンパイルする場合，次に示すオプションは指定できません。これらのオプションを指定したときの動作は保証しません。

  -SimMain，-Bin1Byte，-EquivRule,NotAny，-EquivRule,NotExtend，-EquivRule,StdCode，-V3Rec,Fixed，-V3Rec,Variable，-JPN,Alnum，-JPN,V3JPN，-CompatiV3，-DigitsTrunc，-V3Spec，-V3Spec,CopyEased，-LowerAsUpper，-V3RecFCSpace，-JPN,V3JPNSpace，-VOSCBL,OccursKey，-VOSCBL,ReportControl，-VOSCBL,DataComm，-VOSCBL,RedefinesData，-VOSCBL,AssignDataToDevice，-VOSCBL,EvaluateWhenOther

付録 C.3　COBOL85／COBOL2002 共存環境での制限事項

COBOL85／COBOL2002 共存環境での制限事項を次に示します。

- COBOL adapter for XML，および COBOL 拡張 Run Time System for XML は，COBOL2002 と連携して使用できません。

(UNIX の場合)

- COBOL 拡張 Run Time System for XML を使用する共用ライブラリと，COBOL2002　XML 連携機能を使用する共用ライブラリを一つのプログラムに混在させることはできません。

付録 C.4　64bit 版 COBOL2002　XML 連携機能の制限事項

64bit 版 COBOL2002 XML 連携機能では，32bit 版の cblxml コマンド※を使用して生成した XML アクセス用データ定義，および XML アクセスルーチンは使用できません。

32bit 版の cblxml コマンド※を使用して生成した XML アクセス用データ定義，および XML アクセスルーチンを使用してオブジェクトを作成した場合，アクセスルーチンの実行時にステータス 124(CBLXML-PROG-FILE-TOO-OLD)が返されます。この場合，該当するバージョンの 64bit 版 cblxml コマンドを使用して再生成してください。

注※

COBOL adapter for XML，または 32bit 版 COBOL2002 XML 連携機能の cblxml コマンドです。Windows(x86) COBOL2002 03-00 以降，AIX(32) COBOL2002 03-02 以降または Linux(x86) COBOL2002 で生成した場合は使用できます。

付録 C.5　文字コードの制限事項

(1)　文字コード

XML 連携機能では，使用する XML ドキュメントの文字エンコーディングに関連して，使用できる文字コードに相違があります。XML 連携機能で使用できる文字コードについては，「付録 G　XML 連携機能の XML ドキュメントの文字エンコーディングと文字コード」を参照してください。また，XML ドキュメントで使用できる文字エンコーディングについては，「付録 E.1　使用できる文字エンコーディング」を参照してください。

付録 C.6　Windows OS 固有の注意事項

Windows で XML 連携機能を使用する場合は，次の点に注意してください。

- フォルダ名，ファイル名，プログラムへの入力文字列，および環境変数に指定できる文字は，シフト JIS の範囲だけです。JIS X0213 の第 3 水準漢字，および第 4 水準漢字を含む Unicode の文字は使用できません。
- cblxml コマンドが生成する XML アクセスルーチンや XML アクセス用データ定義の出力先，XML 連携機能を使用した XML ドキュメントのファイルの出力先や更新先として，Windows リソース保護（WRP）対象のフォルダを指定しないでください。指定した場合，意図しないフォルダにリダイレクトされるか，または出力できません。WRP については，各 OS のヘルプなどを参照してください。

付録 D　実体参照

付録 D.1　定義済み実体参照

入力する XML ドキュメントの定義済み実体参照は表 D-1 に従って COBOL データ項目に入力されます。例えば，"&"の定義済み実体参照は"&"文字に変換され COBOL データ項目に入力されます。verbatim 属性に"yes"を指定した Item 要素に対応づけた要素や属性は，定義済み実体参照を示す文字列で入力されます。

cblxml コマンドに-nopeconv オプションを指定して生成した COBOL 原始プログラムを使用して，XML ドキュメントを入力した場合は，COBOL adapter for XML 01-01 までと互換のある動作で入力されます。

表 D-1　XML ドキュメント入力時の定義済み実体と対応する文字

入力する XML ドキュメントの定義済み実体参照とそれに対応する文字参照		COBOL データ項目に入力される文字				
		-nopeconv オプション指定なし		-nopeconv オプション指定あり（COBOL adapter for XML 01-01 までの動作互換）		
		verbatim 指定なし (verbatim="no") の要素，属性の値	verbatim="yes" 指定ありの要素の値	verbatim 指定なし (verbatim="no") の要素，属性の値	verbatim="yes"指定ありの要素（Group 要素の子要素や子要素の属性を含む）	
					要素	属性
&lt;	&#60;	<	&lt;	<	&lt;	&lt;
&gt;	&#62;	>	&gt;	>	>	>
&amp;	&#38;	&	&amp;	&	&amp;	&amp;
&apos;	&#39;	'	&apos;	'	'	'
&quot;	&#34;	"	&quot;	"	"	"※

注※

Group 要素に verbatim="yes"指定の Item 要素を対応づけたとき，Group 要素の子要素に属性が宣言され，その値に"""，"""がある場合。

定義済み実体参照に対応する文字を XML ドキュメントに出力する場合，表 D-2 に従って要素，属性の値を変換して XML ドキュメントに出力します。verbatim 属性に"yes"を指定した Item 要素に対応づけた場合は，変換しないで出力されます。

cblxml コマンドに-nopeconv オプションを指定して生成した COBOL 原始プログラムを使用して，XML ドキュメントを出力した場合は，COBOL adapter for XML 01-01 までと互換のある動作で出力されます。

表 D-2　XML ドキュメント出力時に変換する文字

対応づけた項目	COBOL データ項目の文字	XML ドキュメントに出力する文字		
		-nopeconv オプション指定なし		-nopeconv オプション指定あり (COBOL adapter for XML 01-01 までの動作互換)
		verbatim 指定なし (verbatim="no")の要素，属性の値	verbatim="yes"指定ありの要素の値	
要素 属性	<	<	<	<
	>	>	>	>
	&	&	&	&
	'	'	'	'
	"	"	"	"

注意事項

実体参照を入出力する場合，属性 type="alphanumeric"を指定した項目だけで有効です。その他の type 属性値を指定した場合，動作は保証しません。

付録 D.2　実体参照

入力する XML ドキュメントの DTD に ENTITY 宣言がある場合，宣言した実体参照を表 D-3 に従って入力してください。

表 D-3　XML ドキュメント入力時の実体参照

対応づけた項目	verbatim 指定なし，または verbatim="no"	verbatim="yes"指定あり
要素	変換した実体が入力される	変換しないで入力する
属性	変換した実体が入力される	変換した実体が入力される

verbatim 属性の指定なしで対応づけた場合の，実体参照"&xxx;"の入力例を次に示します。

(入力される XML ドキュメントの例)

```
<?xml version=”1.0” encoding=”Shift_JIS”?>

<!DOCTYPE table [
  <!ELEMENT table (item1, item2)>
  <!ELEMENT item1 (#PCDATA)>
   <!ELEMENT item2 (#PCDATA)>
   <!ENTITY  xxx ”AAA”>
]>
<table>
  <item1>a&xxx;b</item1>
  <item2>abc</item2>
</table>
```

表 D-4　COBOL データ項目の入力値

COBOL データ項目	入力値
01 item1 PIC X(10)	aAAAb
01 item2 PIC X(10)	abc

注意事項

- 実体参照を入力する場合，type 属性値に"alphanumeric"を指定しなければなりません。その他の type 属性値を指定した場合，動作は保証しません。
- XML ドキュメント出力時に実体参照で定義した名称は，実体参照に変換されません。

付録 E　XML ドキュメントの解析に関する仕様

XML 連携機能の XML ドキュメント解析処理（XML パーサ）に関する仕様について説明します。

なお，XML ドキュメントの解析処理は Cosminexus Interschema-Parsing Kit の XML パーサ機能を使用して実現しています。

付録 E.1　使用できる文字エンコーディング

XML 連携機能で処理する XML ドキュメントでは，次に示す文字エンコーディングを使用してください。

- Shift_JIS
- Windows-31J
- EUC-JP
- UTF-8
- UTF-16

文字エンコーディングの指定を省略した場合は UTF-8 を仮定します。

(1)　シフト JIS でのコード範囲と Unicode への変換表

シフト JIS コードの XML ドキュメントを解析する場合は，シフト JIS コードから Unicode へ変換して解析したあと，値を COBOL データ項目に入力するときに，Unicode からシフト JIS コードへ変換します。

XML 不正文字は，Unicode に変換したあとにチェックされ，エラーとして通知されます。XML 不正文字については，「付録 E.2(1)　XML 標準仕様での文字の範囲」を参照してください。シフト JIS コードから Unicode への変換表を表 E-1 に示します。

表 E-1　シフト JIS でのコード範囲と Unicode への変換表

項番	説明	コード範囲（16 進数）	備考
1	制御コード	00-20	※1
2	ASCII/JIS ローマ字	21-7e	x-sjis-cp932※2
3	半角かたかな	a1-df	
4	JIS X 0208-1990（第 1 バイト）	81-9f e0-ec	
5	JIS X 0208-1990（第 2 バイト）	40-7e 80-fc	
6	ユーザ定義文字（第 1 バイト）	f0-f9	ユーザ外字※3
7	ユーザ定義文字（第 2 バイト）	40-7e 80-fc	
8	NEC 選定 IBM 拡張漢字（第 1 バイト）	ed-ee	x-sjis-cp932※2

項番	説明	コード範囲（16 進数）	備考
9	NEC 選定 IBM 拡張漢字（第 2 バイト）	40-7e 80-fc	
10	IBM 拡張漢字（第 1 バイト）	fa-fc	
11	IBM 拡張漢字（第 2 バイト）	40-7e 80-fc	
12	項番 1～11 以外の範囲		未定義※4

注※1

制御コード 0x00～0x20 は，Unicode の制御コード 0x0000～0x0020 に変換されます。

注※2

変換規則 x-sjis-cp932 については，「付録 E.2(2)　文字コード変換表」を参照してください。

注※3

シフト JIS のユーザ定義文字の領域は，Unicode の外字領域（Private Use Area）に変換されます。使用できる外字の範囲については，「付録 E.1(6)　使用できる外字コード」を参照してください。

注※4

Unicode の空白（0x0020）や中点（0x30fb）など適当な文字に置き換わります。XML ドキュメントの入力動作は保証しません。

(2)　日本語 EUC でのコード範囲と Unicode への変換表

日本語 EUC の XML ドキュメントを解析する場合は，日本語 EUC から Unicode へ変換して解析したあと，値を COBOL データ項目に入力するときに，Unicode から日本語 EUC へ変換します。

日本語 EUC では，半角かな文字に半角英数字 2 文字分の領域が必要です。このため，XML ドキュメントの入出力で半角かな文字を使用する場合は，COBOL プログラム中で長さの扱いに注意してください。

XML 不正文字は，Unicode に変換したあとにチェックされ，エラーとして通知されます。XML 不正文字については，「付録 E.2(1)　XML 標準仕様での文字の範囲」を参照してください。日本語 EUC から Unicode への変換表を表 E-2 に示します。

表 E-2　日本語 EUC でのコード範囲と Unicode への変換表

項番	説明	コード範囲（16 進数）	備考
1	制御コード	00-20	※1
2	ASCII/JIS ローマ字	21-7E	x-eucjp-open-19970715-ms※2
3	JIS X 0208-1990（第 1 バイト）	A1-FE	
4	JIS X 0208-1990（第 2 バイト）	A1-FE	
5	半角かたかな（第 1 バイト）	8E	
6	半角かたかな（第 2 バイト）	A1-DF	
7	項番 1～6 以外の範囲		未定義※3

注※1

制御コード 0x00～0x20 は，Unicode の制御コード 0x0000～0x0020 に変換されます。

注※2

変換規則 x-eucjp-open-19970715-ms については，「付録 E.2(2)　文字コード変換表」を参照してください。

注※3

JIS X 0212 補助漢字の範囲では変換は保証しません。それ以外の範囲については，Unicode の空白文字（0x0020），中点（0x30FB）など適当な文字に置き換えられます。

XML ドキュメントの入出力動作は保証しません。

(3)　UCS 系エンコードでの利用可能なコード範囲

表 E–3　UTF-8 でのコード範囲

項番	説明	UTF-8 のコード範囲※1（1 バイト目）	UTF-8 のコード範囲※1（2 バイト目）	UTF-8 のコード範囲※1（3 バイト目）	UTF-8 のコード範囲※1（4 バイト目）	ISO-10646-UCS-4 のコード範囲※1	備考
1	1 バイト形式	00-7f				00000000-0000007f	
2	2 バイト形式	c2-df	80-bf			00000080-000007ff	
3	3 バイト形式	e0-ef	80-bf	80-bf		00000800-0000ffff	
4	4 バイト形式	f0-f4	80-bf	80-bf	80-bf	00010000-0010ffff※2	
5	項番 1～4 以外の範囲						未定義※3

注※1

コード範囲は 16 進数で表しています。

注※2

UTF-8 の 4 バイト形式は ISO-10646-UCS-4 で表現する 0x00010000～0x001fffff の範囲ですが，UTF-16 のサロゲートペア（surrogate pairs）で表す範囲の上限値（0x0010ffff）より大きい範囲については未定義です。

また，JIS X0213 の第 3，4 水準の文字は使用できません。

注※3

未定義の範囲については，空白(0x0020)など適当な文字に置き換わります。XML ドキュメントの入力動作は保証しません。

表 E–4　UTF-16 でのコード範囲

項番	説明	UTF-16 のコード範囲※1	ISO-10646-UCS-4 のコード範囲※1※2	備考
1	サロゲートペア以外のコード範囲※2	0000-d7ff	00000000-0000d7ff	
2		e000-ffff	0000e000-0000ffff	
3	上位のサロゲートのコード範囲※3	d800-dbff	00010000-0010ffff	
4	下位のサロゲートのコード範囲※3	dc00-dfff		

項番	説明	UTF-16 のコード範囲※1	ISO-10646-UCS-4 のコード範囲※1※2	備考
5	項番 1～4 以外の範囲			未定義※4

注※1

コード範囲は 16 進数で表しています。

注※2

XML 連携機能では UCS-2 範囲でサポートしています。

ただし，JIS X0213 の第 3，4 水準の文字は使用できません。

注※3

XML 連携機能では使用できません。

注※4

未定義の範囲については，空白（0x0020）など適当な文字に置き換わります。 XML ドキュメントの入出力動作は保証しません。

表 E–5　ISO-10646-UCS-2 でのコード範囲

項番	説明	コード範囲※1	備考
1	アルファベット，かな，記号など	0000-4dff	
2	CJK 統一文字	4e00-9fff	
3	ユーザ定義外字領域	e000-f8ff	
4	全角アルファベット，半角かな文字	f900-fffd	
5	項番 1～4 以外の範囲		未定義※2

注※1

コード範囲は 16 進数で表しています。

注※2

未定義の範囲については，空白(0x0020)など適当な文字に置き換わります。XML ドキュメントの入出力動作は保証しません。

(4)　Unicode からシフト JIS への変換

Unicode からシフト JIS への変換は，シフト JIS のコード範囲で，x-sjis-cp932 の逆変換となります。ただし，シフト JIS から Unicode への変換で，同じ字形を表す複数のコードから，Unicode の一つのコードへ対応づけられているものについては，対応するコードの一つが逆変換に使用されるため，完全な逆変換とはなりません。

(5)　Unicode から日本語 EUC への変換

Unicode から日本語 EUC への変換は，日本語 EUC のコード範囲で，x-eucjp-open-19970715-ms の逆変換となります。ただし，日本語 EUC から Unicode への変換で，同じ字形を表す複数のコードから，Unicode の一つのコードへ対応づけられているものについては，対応するコードの一つが逆変換に使用されるため，完全な逆変換とはなりません。

(6)　使用できる外字コード

XML 連携機能，および XML 連携機能の実行ライブラリで使用できる外字コードを，次に示します。なお，外字コードは，要素の値だけに使用できます。

シフト JIS，Unicode 以外の外字コードは使用できません。

表 E-6　使用できる外字コード

使用できるシフト JIS での外字コード（16 進数）	Unicode のユーザエリアでの文字コード（16 進数）
f040～f07e，f080～f0fc	e000～e0bb
f140～f17e，f180～f1fc	e0bc～e177
f240～f27e，f280～f2fc	e178～e233
f340～f37e，f380～f3fc	e234～e2ef
f440～f47e，f480～f4fc	e2f0～e3ab
f540～f57e，f580～f5fc	e3ac～e467
f640～f67e，f680～f6fc	e468～e523
f740～f77e，f780～f7fc	e524～e5df
f840～f87e，f880～f8fc	e5e0～e69b
f940～f97e，f980～f9fc	e69c～e757

付録 E.2　使用できる文字の範囲

(1)　XML 標準仕様での文字の範囲

XML 標準仕様では，XML の解析対象実体で使用できる文字の範囲を，Unicode で次のように定めています。

```
#x9, #xa, #xd, [#x20-#xd7ff], [#xe000-#xfffd], [#x10000-#x10ffff]
```

XML 連携機能では，XML 不正文字として，#xfeff および#x110000 以降を除く上記以外の文字が現れた場合，エラーとします。#x110000 以降の文字は#x20 に置き換えられます。#xfeff はエラーとしないで削除されます。

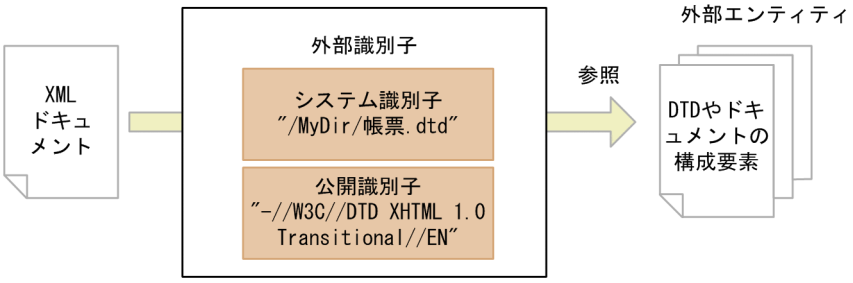
(2)　文字コード変換表

文字コード変換表は下記の URL に定義されています。

http://www.w3.org/TR/japanese-xml/

付録 E.3　外部識別子の解釈

XML ドキュメントでは，外部のファイルにある DTD など，外部エンティティの場所を示すために外部識別子が使用されます。外部識別子には，システム識別子と公開識別子があります。XML パーサは，この外部識別子を解釈して外部エンティティの内容を参照します。



システム識別子には，ファイルの位置を示すパス名を直接記述します。

XML パーサで使用するシステム識別子は，標準的な URI 形式の仕様と次の点で異なります。

- 非 ASCII 文字（多バイト文字）を指定できます。このため，漢字を使用したファイル名を指定できます。
- %HH 形式（HH:16 進数）のエスケープは最初に展開されるため，特殊記号のエスケープとしては使用できません。

公開識別子には，外部エンティティを識別する文字列を記述します。XML 連携機能では，カタログファイルを用いて公開識別子と外部エンティティの対応づけができます。公開識別子とカタログファイルの詳細については，「9.3.3　公開識別子が指定された XML ドキュメント」を参照してください。

XML パーサは，公開識別子を解釈する場合，XML 仕様に従った公開識別子の正規化を行います。

公開識別子を指定する場合には，システム識別子も同時に指定します。公開識別子の解決ができない場合（カタログファイルを指定していない場合，公開識別子が示す外部エンティティが存在しない場合など），システム識別子が外部エンティティの参照に用いられます。

(1)　システム識別子の解決

通常のファイルを指定する際に絶対パスや相対パスを指定するように，システム識別子でも絶対パスや相対パスでの指定ができます。

- システム識別子を絶対パスで指定した場合，指定されたパスがそのままシステム識別子として解釈されます。
- システム識別子を相対パスで指定した場合，相対パスは次のように解釈されます。

（アクセス対象の XML ドキュメントがファイル上にある場合）

アクセス対象の XML ドキュメントが存在するディレクトリ（フォルダ）からの相対パスとなります。

なお，アクセス対象の XML ドキュメントのパスが相対パスで指定された場合は，その相対パスはプロセスのカレントディレクトリ（フォルダ）が基点になります。

（アクセス対象の XML ドキュメントがバッファ（メモリ）上にある場合）
　プロセスのカレントディレクトリ（フォルダ）からの相対パスとなります。

相対パスの場合のシステム識別子の解決例を次に示します。

アクセス対象の XML ドキュメントのパス	参照する外部エンティティのシステム識別子	解決されたシステム識別子
"dir1/MyDoc.xml"	"dir2/MyDtd.dtd"	"dir1/dir2/MyDtd.dtd"
"/home/dir1/MyDoc1.xml"	"dir2/MyDtd.dtd"	"/home/dir1/dir2/MyDtd.dtd"

(2)　使用できるシステム識別子

XML パーサで使用できるシステム識別子の例を示します。

(a)　共通の記述形式

OS に依存しない XML ドキュメントを記述したい場合は，次に示す形式で記述するようにしてください。

- 絶対パス指定のシステム識別子の例
  共通の記述形式はありません。
- 相対パス指定のシステム識別子の例
  - MyDirectory/MyXMLFile.xml

(b)　UNIX で使用できる記述形式

- 絶対パス指定の識別子の例
  - /usr/MyDirectory/MyXMLFile.xml
- 相対パス指定のシステム識別子の例
  「(a)　共通の記述形式」に示す形式で記述してください。

(c)　Windows で使用できる記述形式

- 絶対パス指定のシステム識別子の例
  - C:¥usr¥MyDirectory¥MyXMLFile.xml
  - C:/usr/MyDirectory/MyXMLFile.xml
- 相対パス指定のシステム識別子の例
  - MyDirectory¥MyXMLFile.xml
  - MyDirectory/MyXMLFile.xml

付録 E.4　使用できる解析モードによる動作の違い

XML 連携機能では，CBLXML-OP-*Interface* アクセスルーチン，CBLXML-OB-*Interface* アクセスルーチンの引数で，XML ドキュメントの解析モードを指定できます。

解析モードによって，XML ドキュメントの解析時に実行したいデータ構造の検証や外部エンティティの読み込みに関する動作が異なります。

表 E-7　解析モード

環境変数 CBLXML_PARSE_NOXXE の値	解析モード	妥当性制約の検証	外部エンティティの読み込み
'YES'以外 または指定なし	アクセスモード'V'および'N'指定なし	×	○
	アクセスモード'V'指定あり	○	○
	アクセスモード'N'指定あり	×	×
'YES'	任意	×	×

（凡例）

○：該当する。

×：該当しない。

解析モードの詳細を次に示します。

- アクセスモード'V'および'N'指定なし

  DTD によるドキュメント構成の検証が必要ない場合に使用するモードです。

  外部 DTD サブセットを含むすべての DTD が読み込まれ，エンティティ宣言，および属性宣言を認識できます。したがって，エンティティ参照や属性の初期値は次のように処理されます。

  - エンティティ参照は解決されます。
  - 属性の初期値は供給されます。
  - 属性値を正規化するとき，属性の型に従って処理されます。

  なお，XML ドキュメントが正しい形式であるための制約（well-formedness constraint）を満たすために必要な場合を除き，DTD は必須ではありません。

- アクセスモード'V'指定あり

  DTD を使用して XML ドキュメントのタグの順序や属性などのドキュメント構成を検証したい場合に使用するモードです。外部 DTD サブセットを含むすべての DTD が検証され，エンティティ宣言，および属性宣言を認識できます。

- 環境変数 CBLXML-PARSE-NOXXE に'YES'を指定，またはアクセスモード'N'指定あり

  外部エンティティを読み込まない場合に使用するモードです。DTD によるドキュメント構成の検証はしません。DTD を読み込み，エンティティ宣言，および属性宣言を認識します。ただし，DTD 内で外部 DTD サブセットを使用している場合，外部 DTD サブセットは読み込まれないので，DTD 内の

以後のマークアップ宣言を認識できません。また，XML 文書内で外部エンティティが参照された場合も，外部エンティティは読み込まれません。

付録 E.5　文字参照・実体参照の扱い

1. XML ドキュメントの構文要素「文字参照」を通常の文字と区別しません。XML ドキュメントの入力・更新時，文字参照は対応する文字に変換されます。
2. XML ドキュメント上で，属性値にエンティティ参照や文字参照が含まれる場合，すべて展開された状態で入力・更新されます。
3. 次に示す二つの条件に該当する場合，エンティティは空白となります。
   - 参照されたエンティティが宣言されていない。
   - 参照しているエンティティの宣言を記述した外部 DTD サブセットが読み込まれていない。

付録 E.6　重複する宣言の扱い

XML パーサは重複する宣言に対する扱いが宣言によって異なります。

- エンティティ宣言
  同一の実体が 1 回以上宣言されている場合，最初の宣言を用います。
- 要素型宣言
  - 妥当性制約の検証を実行する場合
    一つの要素型に対して 2 回以上宣言されている場合，エラーとなります。
  - 妥当性制約の検証を実行しない場合
    一つの要素型に対して 2 回以上宣言されている場合，最後の宣言を用います。
- 属性リスト宣言
  ある要素型の同じ属性に複数の定義を与える場合には，最初の宣言を有効とし，他の宣言は無視します。

付録 E.7　入力ドキュメントとの相違点

XML ドキュメント更新機能では，出力ドキュメントは入力ドキュメントと次の点で異なります。

- 文字参照，定義済み実体参照は展開されて出力されます。
- 属性値中のエンティティ参照は展開された値となります。
- DTD でデフォルト値を設定した属性は追加されます。
- 改行コード（CR+LF，LF）は，UNIX の場合，LF で出力されます。Windows の場合，CR+LF(0x0D0A)で出力されます。

- XML ドキュメントを更新した場合，「<」，「>」と要素名の間の空白文字（スペース，タブ，および改行）は削除されます。また，タグの要素や属性の間の空白文字（スペース，タブおよび改行）は，一つのスペースに変換されます。

（例）

```
<ABC△><ABC▲a="xx"▲b="yy"△>
```

注

△：削除します。

▲：一つのスペースにまとめます。

付録 E.8　エンティティ参照の扱い

XML 連携機能では，エンティティ参照の扱いを指定できます。

- 外部エンティティ参照の展開を無効にする
- エンティティ参照の参照回数を制限する

(1)　外部エンティティ参照の展開を無効にする

CBLXML-OP-*Interface* または CBLXML-OB-*Interface* アクセスルーチンの引数（アクセスモード），または環境変数 CBLXML_PARSE_NOXXE で，入力時の外部エンティティ参照の展開を無効にするかどうかを指定します。

環境変数 CBLXML_PARSE_NOXXE に YES を指定すると，外部エンティティ参照を展開しないようにできます。このとき，アクセスモードで"N"（外部エンティティ参照を展開しないモード）を指定していなくても，外部エンティティ参照を展開しません。

外部エンティティ参照を展開しないとき，該当する個所にはテキストデータがないものとして解析します。

（例）外部エンティティ参照がある XML ドキュメント

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root [
 <!ENTITY var SYSTEM URI>
]>
<root>
 <tag1>&var;</tag1>
</root>
```

上記の外部エンティティ参照を展開しないように指定して読み込んだ結果は，次を読み込んだ場合と同じになります。

```
<?xml version="1.0" encoding="Shift_JIS"?>
<root>
```

```
 <tag1></tag1>
</root>
```

注意事項

- XML 連携機能の既定の動作では，すべての外部エンティティ参照を展開します（互換性のため）。システム全体で外部エンティティ参照を展開しない設定にするためには，環境変数 CBLXML_PARSE_NOXXE に YES を指定してください。
- XML ドキュメントを更新モードでオープンするとき，外部エンティティ参照を展開しないように設定すると出力結果にも展開されません。更新モードを使用する XML ドキュメントには，外部エンティティ参照を使用しないことをお勧めします。
- 外部エンティティ参照を展開しないモードで入力するとき，入力 XML ドキュメントの妥当性チェック機能は無効となります。妥当性チェック機能については，「付録 E.4　使用できる解析モードによる動作の違い」を参照してください。
- 内部エンティティ参照を展開しないように指定する機能はありません。

(a)　アクセスルーチンのアクセスモードで無効にする

CBLXML-OP-*Interface* アクセスルーチンで外部エンティティ参照の展開を無効にする例を次に示します。この例は，アクセスモードで「N」を指定した場合です。

CBLXML-OP-*Interface* アクセスルーチンについては「4.2.2　CBLXML-OP-Interface アクセスルーチン」を，CBL XML-OB-*Interface* アクセスルーチンについては「4.2.3　CBLXML-OB-Interface アクセスルーチン」を参照してください。

```
      *> 外部エンティティ参照の展開を無効にして
      *> XMLドキュメントを入力する設定
           MOVE 'RN' TO XML-MODE.

      *> XMLドキュメントを開く
           CALL 'CBLXML-OP-Interface'
                USING XML-FILE-NAME
                      XML-FILE-NAME-LENGTH
                      XML-MODE
                      XML-POINTER
                RETURNING CBLXML-RETURN-CODE.
```

(b)　環境変数 CBLXML_PARSE_NOXXE で展開しないように設定する

環境変数 CBLXML_PARSE_NOXXE で外部エンティティ参照を展開しないように設定します。

形式

```
CBLXML_PARSE_NOXXE=YES
```

規則

環境変数を省略，または'YES'以外を指定したときは，XML ドキュメントを開くアクセスルーチン（CBLXML-OP-*Interface*，CBLXML-OB-*Interface*）に指定したアクセスモードで XML ドキュメントを入力します。

(2)　エンティティ参照の参照回数を制限する

CBLXML-SET-ENTITYLIMIT サービスルーチンを呼び出すか，または環境変数 CBLXML_ENTITYLIMIT に値を指定して，入力時のエンティティ参照回数の制限値を指定します。再帰的なエンティティ参照などで参照回数が膨大になったときのメモリ不足を抑止できます。使用できるメモリ量に合わせて，有効な制限値を指定してください。XML ドキュメントを入力する場合のメモリ所要量については，「7.4　実行時のメモリ所要量」を参照してください。

エンティティ参照回数が指定した値を超えた場合，CBLXML-OP-*Interface* または CBLXML-OB-*Interface* アクセスルーチンのステータスとして 110（CBLXML-XML-PARSE-FAIL）が返されます。

エンティティ参照回数の数え方

XML 連携機能では，入力中に検出した内部エンティティおよび外部エンティティの個数をエンティティ参照回数としてカウントします。定義済み実体参照や文字参照はカウントしません。

外部エンティティ参照を展開しない場合でも，外部エンティティ参照自身は検出したエンティティとしてカウントされます。また，その外部エンティティの参照先に含まれるエンティティは解析対象にならないため，カウントされません。

カウントの例を次に示します。

（XML ドキュメントの例）

```
<?xml version="1.0" encoding="Shift_JIS"?>
<!DOCTYPE root SYSTEM "sample.dtd" [
<!ENTITY xxe10 SYSTEM "xxes/xxe10.ent">
<!ENTITY var1 "12345">
<!ENTITY var2 "67890">
<!ENTITY varX "XXXXX&var1;">
]>
<root>
 &xxe10;
 <tag1>&var1;&var2;</tag1>
</root>
```

（XML ドキュメントで参照する外部エンティティ xxes/xxe10.ent の例）

```
<tag1>&varX;</tag1>
```

説明

- 環境変数 CBLXML_PARSE_NOXXE に YES を指定したときなど，入力時の外部エンティティ参照の展開を無効にした場合，入力する XML ドキュメントの「&xxe10;」は検出したエンティティとしてカウントされますが，"xxes/xxe10.ent"に含まれる「&varX;」は解析されないため，カウントされません。

参照回数は，入力した XML ドキュメントの「&var1;&var2;」を加えて 3 になります。

- 外部エンティティ参照を展開する場合，「&varX;」は「&var1;」を展開するので，"xxes/xxe10.ent"の内容の参照回数は 2 になります。このとき，「&xxe10;」も検出したエンティティとしてカウントされるため，参照回数は全体で 5 となります。

注意事項

- XML 連携機能の既定の動作では，エンティティ参照回数を制限しません（互換性のため）。システム全体でエンティティ参照回数を制限するためには，環境変数 CBLXML_ENTITYLIMIT に制限値を指定してください。
- 入力する XML ドキュメントでエンティティ参照を使用する場合，十分に妥当な値を検証して設定してください。制限値に 0 を指定した場合，1 件でもエンティティ参照が見つかったときは解析を中断します。

(a) CBLXML-SET-ENTITYLIMIT サービスルーチンで制限値を指定する

エンティティ参照回数を 512 回までに制限して入力する例を次に示します。

```
       01 XML-POINTER USAGE POINTER VALUE NULL.
       01 XML-ENTITY-LIMIT PIC 9(9) USAGE COMP VALUE 512.

      *> 空のXMLドキュメントのポインタを作成する
           CALL 'CBLXML-CREATE-XML-POINTER'
               USING XML-POINTER.
           IF RETURN-CODE NOT = 0 THEN
               EXIT PROGRAM
           END-IF.

      *> エンティティ参照回数の制限値を指定する
           CALL 'CBLXML-SET-ENTITYLIMIT'
               USING XML-POINTER XML-ENTITY-LIMIT.

      *> XMLドキュメントを開く（READ）
           MOVE 'RE' TO XML-MODE.
           CALL 'CBLXML-OP-Interface'
                USING XML-FILE-NAME
                      XML-FILE-NAME-LENGTH
                      XML-MODE
                      XML-POINTER
                RETURNING CBLXML-RETURN-CODE.
```

注意事項

- CBLXML-SET-ENTITYLIMIT サービスルーチンの指定は，CBLXML-CREATE-XML-POINTER サービスルーチンで割り当てた XML ドキュメントのポインタごとに有効です。CBLXML-OP-*Interface* または CBLXML-OB-*Interface* アクセスルーチンで XML ドキュメントを入力する前に呼び出してください。CBLXML-SET-ENTITYLIMIT サービスルーチンについては，「9.3.6　エンティティ参照回数を制限する機能」を参照してください。
- CBLXML-SET-ENTITYLIMIT サービスルーチンの指定は，環境変数 CBLXML_ENTITYLIMIT の指定よりも優先されます。

(b) 環境変数 CBLXML_ENTITYLIMIT で制限値を指定する

XML ドキュメントの入力時に，エンティティ参照回数を制限します。

形式

```
CBLXML_ENTITYLIMIT=参照回数
```

規則

- 参照回数に指定できる値は，0～2,147,483,647 の範囲です。
- 環境変数を省略，または参照回数に範囲外の値を指定したとき，エンティティの参照回数は制限されません。

付録 F　XML 連携機能サービスルーチンファイル（Windows の場合）

XML 連携機能では，次のサービスルーチンファイルを提供しています。

表 F-1　XML 連携機能で提供するサービスルーチンファイル

ファイル名	登録内容
COBOL2002XML 連携機能サービスルーチン.svw	COBOL2002　XML 連携機能のサービスルーチンファイル名

サービスルーチンファイルは，次のフォルダに格納されます。

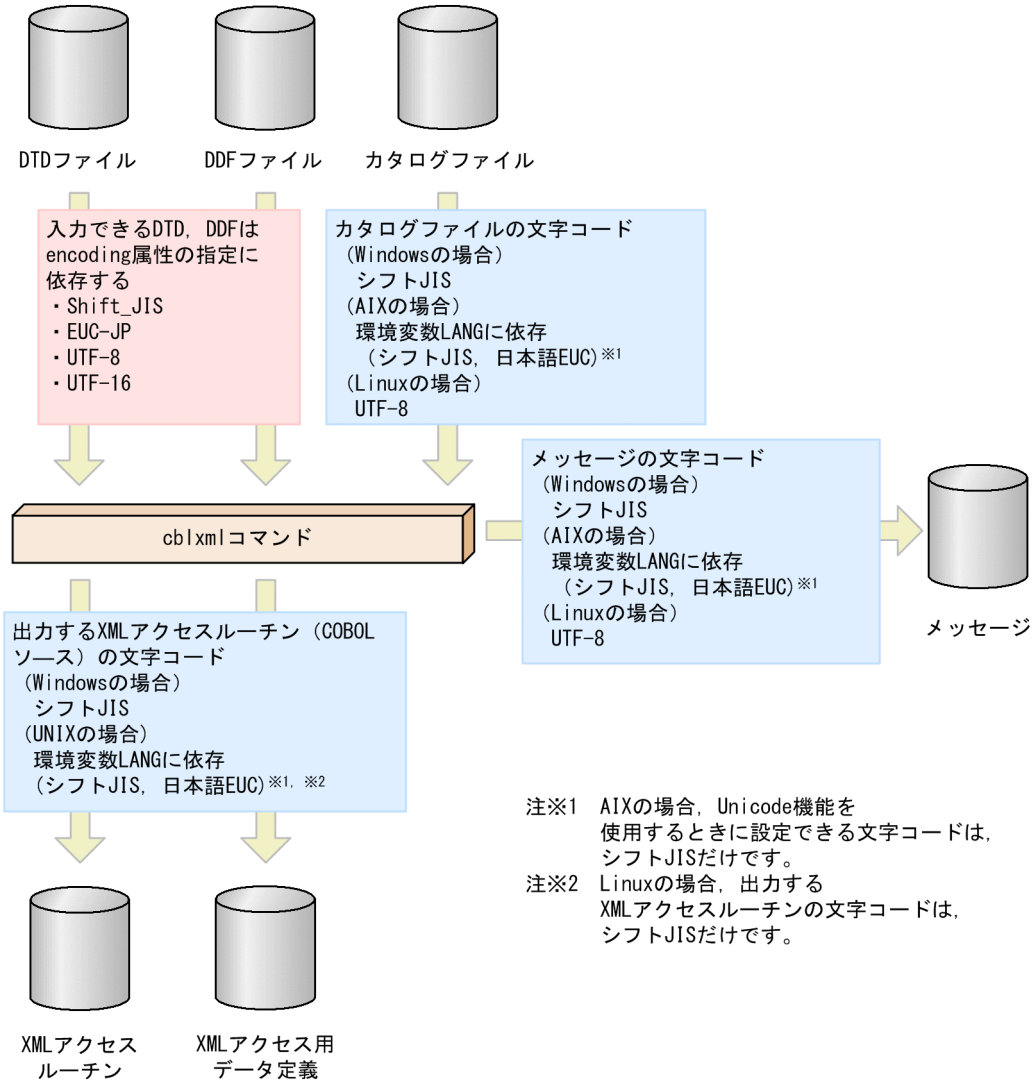
```
COBOL2002のインストールフォルダ¥template
```

COBOL エディタのカスタマイズでこのサービスルーチンファイルを登録すると，XML 連携機能で提供しているサービスルーチンのキーワードを入力したとき，色分け表示や補完ができます。登録方法の詳細，色分け表示，およびキーワード補完については，マニュアル「COBOL2002　操作ガイド」を参照してください。

付録 G　XML 連携機能の XML ドキュメントの文字エンコーディングと文字コード

XML 連携機能では，複数の XML ドキュメントの文字エンコーディングと複数の文字コードをサポートします。cblxml コマンドと XML アクセス用実行時ライブラリが使用する XML ドキュメントの文字エンコーディングとファイルの文字コードの関連を図 G-1，図 G-2 に示します。

図 G-1　cblxml コマンドが入出力するファイルとメッセージ

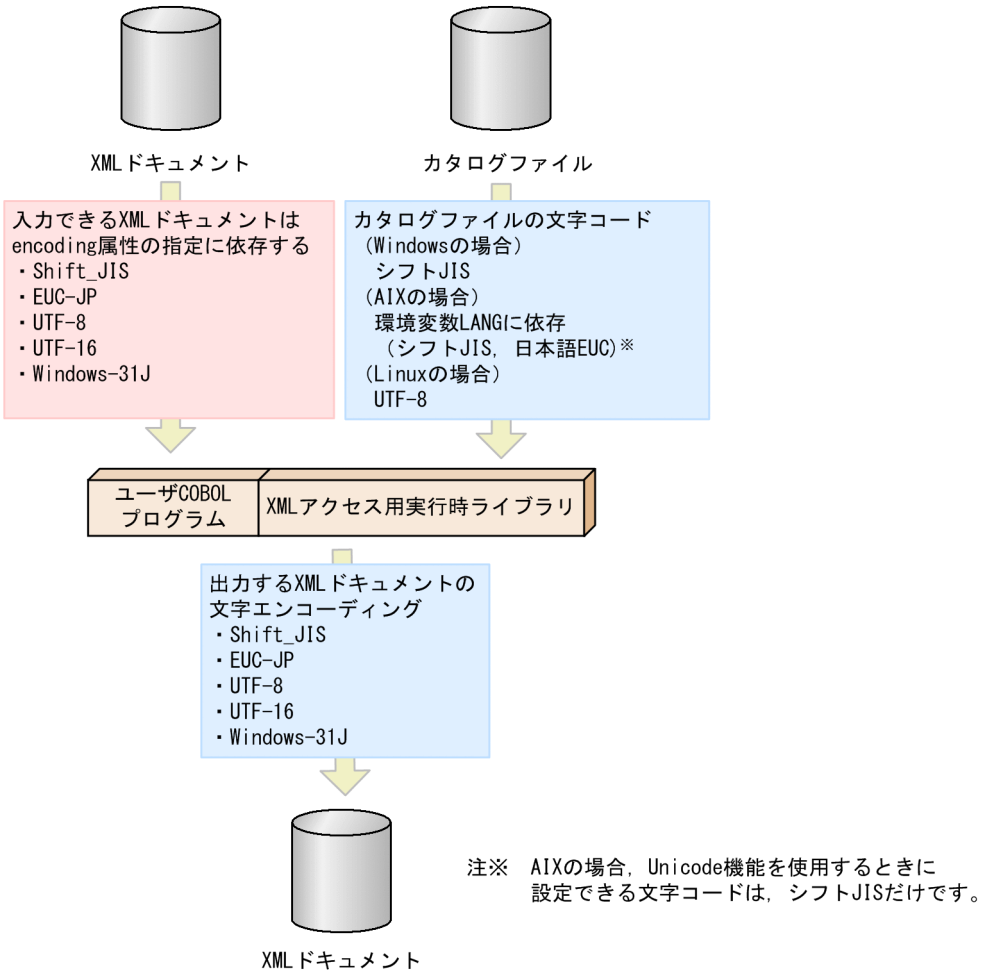


注意事項

1. （UNIX の場合）cblxml コマンドによる XML アクセスルーチン（COBOL ソース）を生成する場合と，XML 連携機能を使用した実行可能ファイルを実行する場合の環境変数 LANG の設定値は同じでなければなりません。もし，環境変数 LANG の設定値が異なる場合は動作は保証しません。
2. （AIX の場合）Unicode 機能を使用する場合，環境変数 LANG はシフト JIS を設定しなければなりません。シフト JIS 以外を設定した場合はエラーが出力されます。

3. (Linux の場合) 環境変数 LANG は UTF-8 を設定しなければなりません。UTF-8 以外を設定した場合はエラーが出力されます。

図 G-2　XML ライブラリが入出力するファイルとメッセージ



注意事項

1. (Windows の場合) 入力した XML ドキュメントの内容をシフト JIS に変更して COBOL のデータ項目に格納します。

   (UNIX の場合) 入力した XML ドキュメントの内容を環境変数 LANG の設定値に従って，文字コードを変更して COBOL のデータ項目に格納します。

   Unicode 機能を使用する場合，英数字項目を UTF-8，日本語項目を UTF-16 に変更して COBOL のデータ項目に格納します。

2. (Windows の場合) COBOL データ項目に格納された値をシフト JIS とみなし，XML ドキュメントの文字エンコーディングに合わせて変更して出力します。

   (UNIX の場合) COBOL データ項目に格納された値を環境変数 LANG の設定値に従った文字コードとみなし，XML ドキュメントの文字エンコーディングに合わせて変更して出力します。

   Unicode 機能を使用する場合，英数字項目を UTF-8，日本語項目を UTF-16 とみなし，XML ドキュメントの文字エンコーディングに合わせて変更して出力します。

付録 G.1　XML ドキュメントの文字エンコーディング

cblxml コマンドが入力できる DTD，DDF ファイルや，XML アクセスルーチンが入力，出力，更新する XML ドキュメントには次に示す文字エンコーディングが使用できます。

表 G–1　入力する XML ドキュメントに使用できる文字エンコーディング

項番	文字エンコーディング
1	Shift_JIS
2	Windows-31J※
3	EUC-JP
4	UTF-8
5	UTF-16 （エンディアンを自動判定し入力します）

注※

XML アクセスルーチンが入力する XML ドキュメントでだけ使用できます。DTD や DDF ファイルでは使用できません。

表 G–2　出力，更新する XML ドキュメントに使用できる文字エンコーディング

項番	文字エンコーディング
1	Shift_JIS
2	Windows-31J
3	EUC-JP
4	UTF-8
5	UTF-16 （出力時ビッグエンディアン，リトルエンディアンの指定ができます）

注意事項

- XML ドキュメントに指定した文字エンコーディングと文字コードは同一でなければなりません。例えば，文字エンコーディングに Shift_JIS を指定した場合，文字コードはシフト JIS でなければなりません。
- UTF-16 の XML ドキュメントを出力した場合，XML ドキュメントの先頭には BOM(Byte Order Mark)を出力します。UTF-8 の XML ドキュメントを出力した場合，XML ドキュメントの先頭には BOM を出力しません。また，BOM の有無を指定することもできません。

(1)　出力する XML ドキュメントの文字エンコーディングの設定

XML アクセスルーチンを使用して出力する XML ドキュメントの文字エンコーディングは，CBLXML-SET-ENCODING サービスルーチン，cblxml コマンドの-outencoding オプションのどちらかを用いて指定します。優先順位は次の 1.，2.の順になります。

1. CBLXML-SET-ENCODING サービスルーチン
2. cblxml コマンドの-outencoding オプション

1.，2.のどちらも指定がない場合は，文字エンコーディングに次の値が設定されます。

- UNIX の場合：環境変数 LANG の値
  環境変数 LANG の値が文字エンコーディングに設定されます。環境変数 LANG の設定がない場合は動作は保証しません。
- Windows の場合：Shift_JIS
  文字エンコーディングに Shift_JIS が設定されます。

表 G-3　出力する XML ドキュメントの文字エンコーディングの設定方法

CBLXML-SET-ENCODING サービスルーチンの指定	cblxml コマンドの-outencoding オプションの値	環境変数 LANG の値（コード系）	出力する XML ドキュメントの文字エンコーディング（encoding 属性の値）
CBLXML-SJIS-ENCODING フラグ	−	−	Shift_JIS
CBLXML-WIN31J-ENCODING フラグ	−	−	Windows-31J
CBLXML-EUC-ENCODING フラグ	−	−	EUC-JP
CBLXML-UTF8-ENCODING フラグ	−	−	UTF-8
CBLXML-UTF16-ENCODING フラグ	−	−	• Windows，Linux の場合：UTF-16，リトルエンディアン • AIX の場合：UTF-16，ビッグエンディアン
CBLXML-UTF16BE-ENCODING フラグ	−	−	UTF-16，ビッグエンディアン
CBLXML-UTF16LE-ENCODING フラグ	−	−	UTF-16，リトルエンディアン
CBLXML-SET-ENCODING サービスルーチンを呼び出さない	sjis	−	Shift_JIS
	win31j	−	Windows-31J
	euc	−	EUC-JP
	utf8	−	UTF-8
	utf16	−	• Windows，Linux の場合：UTF-16，リトルエンディアン • AIX の場合：UTF-16，ビッグエンディアン

CBLXML-SET-ENCODING サービスルーチンの指定	cblxml コマンドの-outencoding オプションの値	環境変数 LANG の値（コード系）	出力する XML ドキュメントの文字エンコーディング（encoding 属性の値）
	utf16be	－	UTF-16，ビッグエンディアン
	utf16le	－	UTF-16，リトルエンディアン
	指定なし	シフト JIS	Shift_JIS
		日本語 EUC	EUC-JP
		UTF-8（Linux の場合）	UTF-8
		シフト JIS，日本語 EUC，UTF-8 以外	• Windows，AIX の場合：Shift_JIS • Linux の場合：EUC-JP

（凡例）

－：指定を無視します。

CBLXML-SET-ENCODING サービスルーチンについては「9.3.5　文字エンコーディングが指定された XML ドキュメント」を，cblxml コマンドの-outencoding オプションについては「4.1.1(2)　cblxml コマンドの実行」を参照してください。

付録 G.2　文字コード

(1)　cblxml コマンド

（UNIX の場合）

cblxml コマンドは環境変数 LANG に指定された値に従った文字コードの XML アクセスルーチンと XML アクセス用データ定義の COBOL ソースを生成します。また，入力するカタログファイルは環境変数 LANG に対応する文字コードで記述しなければなりません。

cblxml コマンドが対応する環境変数 LANG の値を表 G-4 に示します。

表 G-4　cblxml コマンドが対応する環境変数 LANG の値

OS	環境変数 LANG の値	対応する文字コード
AIX	Ja_JP	シフト JIS
	ja_JP	日本語 EUC
Linux	ja_JP.UTF-8	• COBOL ソースの場合：シフト JIS • カタログファイルの場合：UTF-8

（Windows の場合）

cblxml コマンドは，シフト JIS の文字コードの XML アクセスルーチンと XML アクセス用データ定義の COBOL ソースを作成します。また，入力するカタログファイルの文字コードはシフト JIS です。

(2) XML アクセス用実行時ライブラリ

（UNIX の場合）

XML 連携機能を使用するプログラムは，環境変数 LANG に設定した文字コードに従って動作します。そのため，CBLXML-READ-CATALOG サービスルーチンで入力するカタログファイルは環境変数 LANG に従った文字コードでなければなりません。表 G-5 に対応する環境変数 LANG の値を示します。

表 G-5　XML アクセス用実行時ライブラリが対応する環境変数 LANG の値

OS	環境変数 LANG の値	対応する文字コード
AIX	Ja_JP	シフト JIS
	ja_JP	日本語 EUC
Linux	ja_JP.UTF-8	UTF-8

（Windows の場合）

XML 連携機能を使用するプログラムは，文字コードがシフト JIS として動作します。そのため，CBLXML-READ-CATALOG サービスルーチンで入力するカタログファイルの文字コードはシフト JIS でなければなりません。

ただし，Unicode 機能を使用した場合，COBOL のデータ項目に入出力する文字コードは Unicode になります。

付録 H　Unicode 機能

COBOL2002 の Unicode 機能を次に示します。

- コンパイル時に，シフト JIS で記述された COBOL ソースから，コード系が Unicode のオブジェクトを生成します。その際，英数字定数を UTF-8 の文字コードへ変換し，日本語定数を UTF-16 に変換します。
- 実行時に，用途が DISPLAY の項目の文字コードを UTF-8 として扱い，用途が NATIONAL の項目の文字コードを UTF-16 として扱います。
- 日本語定数や用途が NATIONAL の項目のバイトオーダを設定できます。

XML 連携機能では，COBOL2002 の Unicode 機能に対応した XML 対応 COBOL プログラムが作成できます。

付録 H.1　COBOL2002 の Unicode 機能に対応した XML 連携機能を使用する COBOL プログラムの作成

COBOL2002 の Unicode 機能に対応した XML 連携機能を使用する場合，Unicode 機能に対応したデータ定義ファイル(DDF)を作成し，cblxml コマンドで Unicode 機能に対応した XML アクセスルーチンを生成する必要があります。

また，生成した XML アクセスルーチン(COBOL ソース)や XML アクセス用データ定義(登録集原文)，それを呼び出すユーザプログラムの COBOL ソースは，コンパイル時に-UniObjGen オプションを指定して，Unicode 機能に対応したコンパイルを行う必要があります。

次に Unicode 機能に対応したプログラムの作成手順を示します。

(1)　Unicode 機能に対応した COBOL ソースの作成

Unicode 機能に対応した COBOL ソースの作成については，マニュアル「COBOL2002　使用の手引　手引編」または「COBOL2002　ユーザーズガイド」を参照してください。

(2)　Unicode 機能に対応したデータ定義ファイル(DDF)の作成

COBOL2002 の Unicode 機能を使用する場合，XML ドキュメントから入力する値や，DDF で Item 要素の emptyValue 属性などに指定する値が，Unicode に変換されることを考慮した設計が必要です。

例えば，COBOL の英数字項目(UTF-8)に，文字エンコーディングが Shift_JIS(シフト JIS)の XML ドキュメントから値'日立'を入力する場合，表 H-1 に示すように文字コードによって 1 文字を構成するバイト数が異なるため，DDF に指定する size 属性のけた数は，UTF-8 の文字列として格納できるけた数を設定する必要があります。この例では UTF-8 に変換されることを考慮し，DDF での size 属性には 6 を指定する必要があります。

表 H-1　文字コードによる長さの違いの例

文字	日	立
シフト JIS	0x93FA	0x97A7
UTF-8	0xE697A5	0xE7AB8B

次の仕様に従ってデータ定義ファイル(DDF)を作成してください。

- Item 要素，AttrItem 要素に指定する size 属性には，XML ドキュメントから入力した値を Unicode(英数字項目の場合 UTF-8，日本語項目の場合 UTF-16)に変換した文字列を格納するのに十分な大きさを指定してください。
- Item 要素，AttrItem 要素に指定した次に示す属性の値は，Unicode に変換した文字列になります。そのため，size 属性に指定するけた数は，おのおのの値を Unicode に変換した文字列の長さが収まるけた数を指定してください。
  - emptyValue 属性
  - emptyContentValue 属性
  - invalidCharValue 属性
  - overflowValue 属性
- Interface 要素，BaseElement 要素の cobName 属性(省略時は elemName 属性)に指定する値に，UTF-8 に変換すると多バイトになる文字は指定できません。

(3)　Unicode 機能に対応した XML アクセスルーチンや XML アクセス用データ定義の生成

Unicode 機能に対応した XML アクセスルーチンや XML アクセス用データ定義を生成するため，cblxml コマンドに-unisrc オプションを指定してください。-unisrc オプションを指定した場合，cblxml コマンドの-o オプションに Unicode に変換すると多バイトになる文字は指定できません。

(4)　Unicode 機能に対応した COBOL ソースのコンパイル

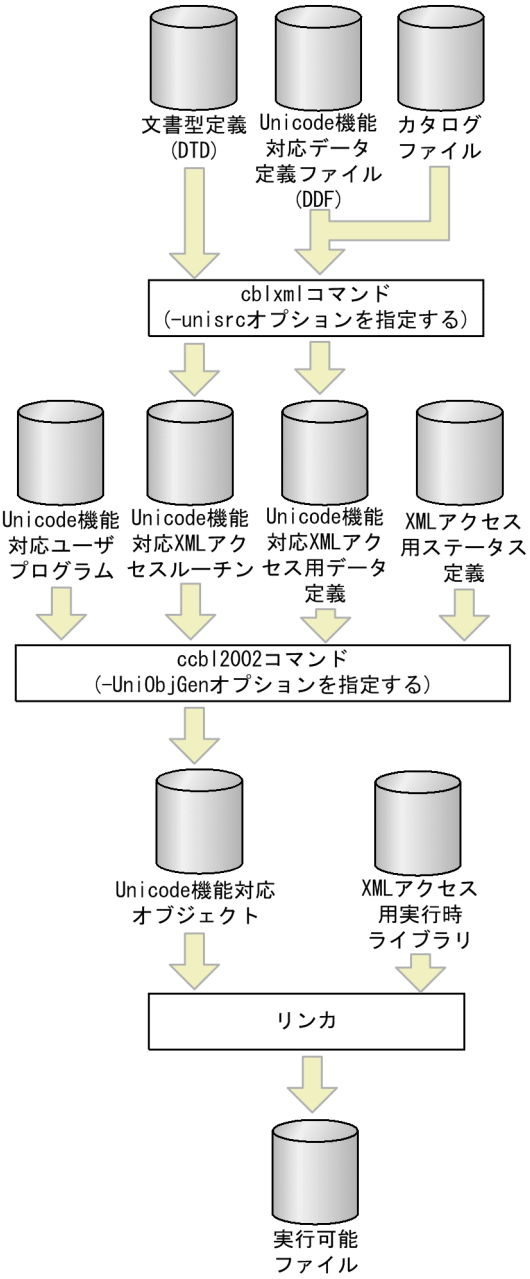
-unisrc オプションを指定した Unicode 機能に対応した XML アクセスルーチンや XML アクセス用データ定義，Unicode 機能に対応したユーザプログラムの COBOL ソースをコンパイルする場合，ccbl2002 コマンドに-UniObjGen オプションを指定します。また，日本語項目の定数のバイトオーダを指定したい場合は-UniEndian オプションを指定します。

(5)　オブジェクトのリンク

-UniObjGen オプションを指定して，生成されたオブジェクトをリンクします。

図 H-1 に XML 連携機能を使用する Unicode 機能に対応した COBOL プログラムの作成の概略を示します。

図 H-1　Unicode 機能に対応した XML 連携機能を使用する COBOL プログラムの作成概略



付録 H.2　COBOL2002 の Unicode 機能に対応した XML 連携機能を使用する COBOL プログラムの実行

(1)　Unicode 機能の実行時環境変数

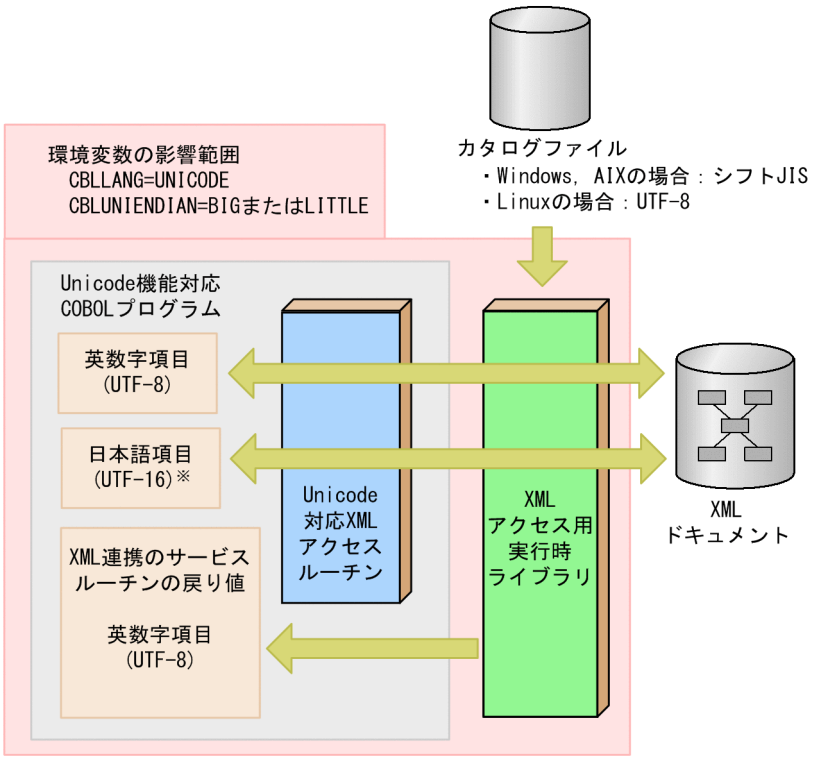
Unicode 機能の COBOL プログラムを実行する場合，COBOL2002 の実行時環境変数 CBLLANG=UNICODE を指定する必要があります。また，日本語項目のバイトオーダを指定したい場合，COBOL2002 の実行時環境変数 CBLUNIENDIAN を指定してください。詳細についてはマニュアル「COBOL2002　使用の手引　手引編」または「COBOL2002　ユーザーズガイド」を参照してください。

(2)　XML 連携機能の動作

XML 連携機能の Unicode 機能に対応した XML アクセスルーチンは，英数字項目の文字コードを UTF-8，日本語項目の文字コードを UTF-16(UCS-2 範囲)として扱います。日本語項目のバイトオーダは COBOL2002 の実行時環境変数 CBLUNIENDIAN の指定値に従います。また，XML 連携機能が提供するサービスルーチン(CBLXML-GET-ERROR, CBLXML-READ-CATALOG-FILE, CBLXML-GET-NEXT-BE)は，Unicode へ変換した結果を返します。

図 H-2 に Unicode 機能に対応した XML 連携機能を使用する COBOL プログラムの文字コードを示します。

図 H-2　Unicode 機能に対応した XML 連携機能を使用する COBOL プログラムでの文字コード



注※

バイトオーダは COBOL2002 の実行時環境変数 CBLUNIENDIAN の指定に従います。

注意事項

- cblxml コマンドで-unisrc オプションを指定した Unicode 機能に対応した XML アクセスルーチンの実行時に，COBOL2002 の実行時環境変数 CBLLANG の値が UNICODE 以外の場合は，CBLXML-OP-*Interface* と CBLXML-OB-*Interface* アクセスルーチンで実行環境が不正であるステータス 122 を返します。また，Unicode 機能に対応していない XML アクセスルーチンの実行時に，COBOL2002 の実行時環境変数 CBLLANG の値が UNICODE の場合も CBLXML-OP-*Interface* と CBLXML-OB-*Interface* アクセスルーチンで実行環境が不正であるステータス 122 を返します。
- (Windows，AIX の場合) Unicode 機能を使用する場合，カタログファイルの文字コードはシフト JIS でなければなりません。

- （Linux の場合）Unicode 機能を使用する場合，カタログファイルの文字コードは UTF-8 でなければなりません。
- （UNIX の場合）CBLXML-OP-*Interface* と CBLXML-OB-*Interface* アクセスルーチンの実行時に，cblxml コマンドの-unisrc オプションの指定の有無と環境変数 LANG の設定値の整合性チェックを行い，整合性に矛盾がある場合はステータス 123 を返します。

付録 H.3　Unicode 機能に対応した XML 連携機能の入出力ファイルの文字コード

Unicode 機能に対応した XML 連携機能の入出力ファイルの文字コードを次に示します。

分類	入出力	入出力ファイル	文字コード
cblxml コマンド	入力	DTD ファイル	encoding 属性に依存
		DDF ファイル	encoding 属性に依存
		カタログファイル	• Windows，AIX の場合：シフト JIS • Linux の場合：UTF-8
		cblxml コマンドのコマンドラインの引数	• Windows，AIX の場合：シフト JIS • Linux の場合：UTF-8
	出力	XML アクセスルーチン（COBOL ソース）	シフト JIS
		XML アクセス用データ定義(登録集原文)	シフト JIS
		エラーメッセージ	• Windows，AIX の場合：シフト JIS • Linux の場合：UTF-8
実行時	入力	XML ドキュメント	encoding 属性に依存
		カタログファイル	• Windows，AIX の場合：シフト JIS • Linux の場合：UTF-8
	出力	XML ドキュメント	encoding 属性に依存

付録 I　各バージョンの変更内容

各バージョンの変更内容を示します。

変更内容（3021-3-608-30）　COBOL2002 Net Server Suite 04-41，COBOL2002 Net Server Runtime 04-41，COBOL2002 Net Server Suite(64) 04-41，COBOL2002 Net Server Runtime(64) 04-41

追加・変更内容
XML 連携機能が生成する COBOL 原始プログラムをコンパイルする場合の制限事項に，-VOSCBL,RedefinesData オプションを追加した。

変更内容（3021-3-608-20）　COBOL2002 Net Developer 04-10，COBOL2002 Net Server Runtime 04-10，COBOL2002 Net Client Runtime 04-10，COBOL2002 Net Server Suite 04-10，COBOL2002 Net Client Suite 04-10，COBOL2002 Net Developer(64) 04-10，COBOL2002 Net Server Runtime(64) 04-10，COBOL2002 Net Server Suite(64) 04-10，COBOL2002 Developer Professional 04-10，COBOL2002 Developer Professional(64) 04-10

追加・変更内容
次の製品の適用 OS に「Windows Server 2019」を追加した。 • P-2636-2344 COBOL2002 Net Developer • P-2436-5344 COBOL2002 Net Server Runtime • P-2436-6344 COBOL2002 Net Server Suite • P-2936-2344 COBOL2002 Net Developer(64) • P-2936-5344 COBOL2002 Net Server Runtime(64) • P-2936-6344 COBOL2002 Net Server Suite(64) • P-2636-7344 COBOL2002 Developer Professional • P-2936-7344 COBOL2002 Developer Professional(64)
Windows COBOL2002 で，XML ドキュメントの文字エンコーディングとして Windows-31J をサポートした。 これに伴い，開発マネージャのプロジェクト設定ダイアログボックスで選択する XML ドキュメントの文字エンコーディングに，Windows-31J を追加した。
XML 連携機能が生成する COBOL 原始プログラムをコンパイルする場合の制限事項に，-VOSCBL,DataComm オプションを追加した。

変更内容（3021-3-608-10）　COBOL2002 Net Server Suite 04-00，COBOL2002 Net Server Runtime 04-00，COBOL2002 Net Server Suite(64) 04-00，COBOL2002 Net Server Runtime(64) 04-00

追加・変更内容
次に示す適用 OS がサポート対象外となった。 • AIX V6.1 • Red Hat Enterprise Linux 5 (AMD/Intel 64) • Red Hat Enterprise Linux 5 Advanced Platform (AMD/Intel 64) • Red Hat Enterprise Linux Server 6 (64-bit x86_64)

追加・変更内容
XML ドキュメントの文字エンコーディングとして Windows-31J をサポートした。

付録 J　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 J.1　関連マニュアル

このマニュアルは次のマニュアルと関連があります。必要に応じてお読みください。

- COBOL2002　ユーザーズガイド　（3021-3-600）
- COBOL2002　操作ガイド　（3021-3-601）
- COBOL2002　使用の手引　手引編　（3021-3-602）
- COBOL2002　使用の手引　操作編　（3021-3-603）
- COBOL2002　言語　標準仕様編　（3021-3-604）
- COBOL2002　言語　拡張仕様編　（3021-3-605）
- COBOL2002　Java プログラム呼び出し機能ガイド　（3021-3-607）
- COBOL2002　メッセージ　（3021-3-609）

付録 J.2　このマニュアルでの表記

このマニュアルでは，マイクロソフト製品の名称を次のように表記しています。

マニュアルでの表記			製品名
Windows	Windows 7	Windows 7(x86)	Microsoft Windows 7 Professional 日本語版(32 ビット版)
			Microsoft Windows 7 Enterprise 日本語版(32 ビット版)
			Microsoft Windows 7 Ultimate 日本語版(32 ビット版)
		Windows 7(x64)	Microsoft Windows 7 Professional 日本語版(64 ビット版)
			Microsoft Windows 7 Enterprise 日本語版(64 ビット版)
			Microsoft Windows 7 Ultimate 日本語版(64 ビット版)
	Windows 8.1	Windows 8.1(x86)	Windows 8.1 Pro 日本語版(32 ビット版)
			Windows 8.1 Enterprise 日本語版(32 ビット版)
		Windows 8.1(x64)	Windows 8.1 Pro 日本語版(64 ビット版)
			Windows 8.1 Enterprise 日本語版(64 ビット版)
	Windows 10	Windows 10(x86)	Windows 10 Pro 日本語版(32 ビット版)
			Windows 10 Enterprise 日本語版(32 ビット版)

マニュアルでの表記			製品名
		Windows 10(x64)	Windows 10 Pro 日本語版(64 ビット版)
			Windows 10 Enterprise 日本語版(64 ビット版)
	Windows 11		Windows 11 Pro 日本語版
			Windows 11 Enterprise 日本語版
	Windows Server 2008 R2		Microsoft Windows Server 2008 R2 Standard 日本語版
			Microsoft Windows Server 2008 R2 Enterprise 日本語版
			Microsoft Windows Server 2008 R2 Datacenter 日本語版
	Windows Server 2012		Microsoft Windows Server 2012 Standard 日本語版
			Microsoft Windows Server 2012 Datacenter 日本語版
	Windows Server 2012 R2		Microsoft Windows Server 2012 R2 Standard 日本語版
			Microsoft Windows Server 2012 R2 Datacenter 日本語版
	Windows Server 2016		Microsoft Windows Server 2016 Standard 日本語版
			Microsoft Windows Server 2016 Datacenter 日本語版
	Windows Server 2019		Windows Server 2019 Standard 日本語版
			Windows Server 2019 Datacenter 日本語版
	Windows Server 2022		Windows Server 2022 Standard 日本語版
			Windows Server 2022 Datacenter 日本語版

このマニュアルは，製品種別によって相違点があります。本文中での製品種別ごとの表記を，次に示します。

マニュアルでの表記			該当する製品の形名
Windows	Windows(x86)		P-2636-2344 P-2436-5344 P-2636-3344 P-2436-6344 P-2636-4344 P-2636-7344
	Windows(x64)		P-2936-2344 P-2936-5344 P-2936-6344 P-2936-7344
UNIX	AIX	AIX(32)	P-1M36-1141 P-1M36-2141
		AIX(64)	P-1M36-1241 P-1M36-2241

マニュアルでの表記			該当する製品の形名
	Linux	Linux(x86)	−※
		Linux(x64)	P-9W36-1241 P-9W36-2241

注※

該当する製品の形名の詳細は，「リリースノート」でご確認ください。

- Windows(x64) COBOL2002，AIX(64) COBOL2002，Linux(x64) COBOL2002 で機能差がない場合，64bit 版 COBOL2002 と表記しています。
- Windows(x86) COBOL2002，AIX(32) COBOL2002，Linux(x86) COBOL2002 で機能差がない場合，32bit 版 COBOL2002 と表記しています。

また，このマニュアルでは各製品を次のように表記しています。

マニュアルでの表記			製品名
UNIX	AIX		AIX V7.1
			AIX V7.2
	Linux	Linux Server 7 (64-bit x86_64)	Red Hat Enterprise Linux Server 7 (64-bit x86_64)
		Linux Server 8 (64-bit x86_64)	Red Hat Enterprise Linux Server 8 (64-bit x86_64)
		Linux Server 9 (64-bit x86_64)	Red Hat Enterprise Linux Server 9 (64-bit x86_64)

- 日立 COBOL2002 のことを日立 COBOL2002，または単に COBOL2002 と表記しています。また，使用できる機能が異なるため，プラットフォームを明確にする必要がある場合は，「(UNIX の場合)」または「UNIX COBOL2002」，「(Windows の場合)」または「Windows COBOL2002」のように表記しています。
- コンパイラオプションの説明では，次の表記を使用します。

  「XXX オプション」，または単に「XXX」とオプション名が表記されている場合

  　XXX オプションについて，サブオプションの組み合わせを含む，すべての場合を意味します。

  「XXX,YYY オプション」，または単に「XXX,YYY」とサブオプションを含めたオプション名が表記されている場合

  　XXX,YYY オプションだけの場合を意味します。

  「XXX コンパイラオプション」と表記されている場合

  　リンカオプションなど，ほかのオプションと明確に区別する必要がある場合を意味します。

（例 1）

「-Compile オプション」または「-Compile」と記載している場合，-Compile オプションのサブオプションの組み合わせすべて（-Compile,CheckOnly／-Compile,NoLink）を意味します。

（例 2）

「-Compile,CheckOnly オプション」または「-Compile,CheckOnly」と記載している場合，-Compile,CheckOnly だけを意味します。

付録 J.3　KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

付録 K　用語解説

(英字)

BOM（バイトオーダマーク）

ファイルの先頭に付加された，Unicode の表現形式を表す情報。COBOL2002 では，テキスト編成ファイルに対してこの情報を付加します。本文中では，Unicode シグニチャと表記します。

DDF（Data Definition File）

XML ドキュメントの DTD で定義された要素のうち，COBOL プログラムからアクセスしたい要素と COBOL のデータ項目との対応づけを記述するファイルです。

DDF は，データ定義ファイルともいいます。

DDL（Data Definition Language）

DDF に記述するマークアップ言語です。XML 要素に対応する COBOL のデータ項目の構造，データ名，データ型などを定義できます。

DDL は，データ定義言語ともいいます。

DOCTYPE 宣言

DTD に記載する，文書の型を宣言する記述のことです。DOCTYPE 宣言は，「<!DOCTYPE ～」の形式で宣言されます。

DTD（Document Type Definition）

XML ドキュメントの構造を定義するものです。DTD を使うと，XML ドキュメントの構造の解析，検証などができます。XML 連携機能では，DTD を基に COBOL プログラムからアクセスする XML 要素を指定します。

DTD は，文書型定義ともいいます。

IVS

漢字を表す Unicode の直後に Variation Selector と呼ばれるコードを付加し，漢字の「異体字」を表現する方法のことです。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目で使用します。

UCS-2（Universal multi-octet Character Set 2）

符号化文字集合の一つの形式です。1 文字を 2 バイトで表現します。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目でサポートします。

UCS-4（Universal multi-octet Character Set 4）

符号化文字集合の一つの形式です。1 文字を 4 バイトで表現します。

COBOL2002 では，用途が DISPLAY および NATIONAL の項目で UCS-4 の範囲をサポートします。

Unicode

1 バイトでは表現できない文字セットを含めあらゆる文字セットをサポートした文字コード。代表的な符号化文字集合として UCS-2，UCS-4 があります。代表的なエンコーディングスキーマとして UTF-8，UTF-16 があります。

COBOL2002 では，UCS-4 の範囲（UCS-2 の範囲を含む）をサポートします。

本文中では，符号化文字集合，およびエンコーディングスキーマを文字コードと表記します。

URI

Uniform Resource Identifier の略です。URL のように Web 上のデータの位置を指定する方法と，Web 上の特定のデータを固有の名前で指定する方法を持ち合わせた規格です。

URL

Uniform Resource Locator の略です。Web 上にある特定のデータの位置を指定する規格です。

UTF-16（16-bit UCS Transformation Format）

エンコーディングスキーマの一つの形式です。1 つのコード単位を 2 バイトとし，1 文字を 1 コード単位（2 バイト），または 2 コード単位（4 バイト）で表現します。UTF-16 では 2 バイトのコード単位の 1 バイト目を先に書くビッグエンディアン形式（UTF-16BE）と 1 バイト目を後に書くリトルエンディアン形式（UTF-16LE）があります。

COBOL2002 では，用途が NATIONAL の項目で UCS-4 の範囲（UCS-2 の範囲を含む）をサポートします。

UTF-8（8-bit UCS Transformation Format）

エンコーディングスキーマの一つの形式です。ASCII 文字を 1 バイト，日本語文字を 3～8 バイト，半角かたかなを 3 バイトで表現します。

COBOL2002 では，用途が DISPLAY の項目で使用します。

XML（eXtensible Markup Language）

World Wide Web Consortium（W3C）によって標準化が進められている，文章の構造を定義するための言語です。XML では，テキストファイル中の各データをタグ（<>）で囲むことによって，データの構造を定義します。

XML アクセス用データ定義

XML アクセスルーチンの戻り値や，アクセスする XML 要素に対応したデータ項目を登録集原文として提供するものです。XML アクセス用データ定義は，XML ドキュメントの DTD とユーザが作成した DDF を基に，cblxml コマンドを使って生成します。

XML アクセスルーチン

COBOL プログラムから XML ドキュメントにアクセスするときに呼び出す副プログラムです。XML アクセスルーチンは，XML ドキュメントの DTD とユーザが作成した DDF を基に，cblxml コマンドを使って生成します。

XML 要素

XML ドキュメントで，XML タグ，タグに付随する属性，およびタグに囲まれたデータのすべてのことです。

（ア行）

エンティティ

XML 文書中で使用される画像データや PDF ファイルなどのファイルや，置換する文字列などのデータを保持する実体です。

（カ行）

解析モード

XML パーサが XML 文書を解析するときの動作方式です。解析モードには，検証モードと非検証モードがあります。

公開識別子

外部 DTD や外部実体を識別するための文字列です。XML 連携機能では，公開識別子と，外部 DTD や外部実体を含むファイルの対応を，カタログファイルで定義することで，公開識別子を用いて外部 DTD や外部実体を参照する XML ドキュメントを処理できます。

（サ行）

サロゲート

UTF-16 の拡張で，2 つのコード単位（4 バイト）で 1 文字を表す機能です。この 2 つのコード単位の組み合わせのことをサロゲートペアと呼びます。

COBOL2002 では，用途が NATIONAL の項目で使用します。

システム識別子

システム識別子とは，解析中のエンティティ（XML 文書）を指す識別子のことです。

属性

XML 要素のうち，タグに情報を追加するための修飾部分のことです。

（タ行）

タグ

XML 要素のうち，データを意味づけするために囲んだ「<…>」部分のことです。

データ

XML 要素のうち，XML タグに囲まれた内容のことです。

登録集原文

COBOL プログラム中でよく利用される標準化した手続き，ファイル記述，レコード記述，または完全な一つのプログラムなどを，コンパイルするプログラムとは別のファイルに登録したものです。XML 連携機能では，XML アクセス用データ定義が登録集原文として生成されます。

（ハ行）

パーサ

パーサとは，XML 文書を解析するプログラムのことです。

バイトオーダ

2 バイト以上のデータの記録を行う順序のことです。例えば，0x1234 のデータを 0x1234 のように最上位のバイトから順番に記録する方式をビッグエンディアン，0x3412 のように最下位のバイトから順番に記録する方式をリトルエンディアンと呼びます。2 バイトの UTF-16 は，バイトオーダを意識します。

（マ行）

見た目幅

Unicode 機能の組み込み関数で使用します，文字の見た目の幅です。半角文字の幅は 1，全角文字の幅は 2 として扱います。

Unicode 機能の組み込み関数で，文字を見た目幅で数える場合に使用します。

索引

記号

数字

A

B

C

D

E

F

G

I

L

M

N

O

P