HITACHI
Inspire the Next

# COBOL2002　Cosminexus 連携機能ガイド

手引・操作書

4010-1J-806-10

COBOL2002

# 前書き

## ■ 対象製品

P-2636-2354 COBOL2002 Net Developer 05-00（適用 OS：Windows 10(x64)，Windows 11，Windows Server 2019，Windows Server 2022）

P-2436-5354 COBOL2002 Net Server Runtime 05-00（適用 OS：Windows Server 2019，Windows Server 2022）

P-2436-6354 COBOL2002 Net Server Suite 05-00（適用 OS：Windows Server 2019，Windows Server 2022）

P-2636-4354 COBOL2002 Net Client Suite 05-00（適用 OS：Windows 10(x64)，Windows 11）

P-2936-2354 COBOL2002 Net Developer(64) 05-00（適用 OS：Windows 10(x64)，Windows 11，Windows Server 2019，Windows Server 2022）

P-2936-5354 COBOL2002 Net Server Runtime(64) 05-00（適用 OS：Windows Server 2019，Windows Server 2022）

P-2936-6354 COBOL2002 Net Server Suite(64) 05-00（適用 OS：Windows Server 2019，Windows Server 2022）

P-9W36-1251 COBOL2002 Net Server Suite(64) 05-00（適用 OS：Linux Server 9 (64-bit x86_64)）

P-9W36-2251 COBOL2002 Net Server Runtime(64) 05-00（適用 OS：Linux Server 9 (64-bit x86_64)）

P-2636-7354 COBOL2002 Developer Professional 05-00（適用 OS：Windows 10(x64)，Windows 11，Windows Server 2019，Windows Server 2022）

P-2936-7354 COBOL2002 Developer Professional(64) 05-00（適用 OS：Windows 10(x64)，Windows 11，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

記載の会社名，製品名などは，それぞれの会社の商標もしくは登録商標です。

■ 発行

2025 年 4 月　4010-1J-806-10

# 変更内容

## 変更内容（4010-1J-806-10） COBOL2002 Net Server Suite(64) 05-00，COBOL2002 Net Server Runtime(64) 05-00

| 追加・変更内容 | 変更個所 |
| --- | --- |
| 次に示す適用 OS をサポート対象外とした。<br>• AIX V7.1<br>• AIX V7.2<br>• AIX 7.3<br>• Red Hat Enterprise Linux Server 7 (64-bit x86_64)<br>• Red Hat Enterprise Linux Server 8 (64-bit x86_64) | − |

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 Cosminexus 連携機能について説明したものです。

- P-2636-2354 COBOL2002 Net Developer
- P-2436-5354 COBOL2002 Net Server Runtime
- P-2436-6354 COBOL2002 Net Server Suite
- P-2636-4354 COBOL2002 Net Client Suite
- P-2936-2354 COBOL2002 Net Developer(64)
- P-2936-5354 COBOL2002 Net Server Runtime(64)
- P-2936-6354 COBOL2002 Net Server Suite(64)
- P-9W36-1251 COBOL2002 Net Server Suite(64)
- P-9W36-2251 COBOL2002 Net Server Runtime(64)
- P-2636-7354 COBOL2002 Developer Professional
- P-2936-7354 COBOL2002 Developer Professional(64)

上記のプログラムプロダクトの適用 OS と，COBOL2002 Cosminexus 連携機能の適用 OS には，一部相違点があります。

適用 OS の相違点，および前提ソフトウェアの詳細については，「リリースノート」を参照してください。

このマニュアルでは，次に示す製品を総称して，Cosminexus と表記します。

- Cosminexus Application Server
- Cosminexus Developer

## ■ 対象読者

Java 言語と COBOL 言語について基本的な知識を持っていて，両言語の通信を簡単に実現したい方を対象としています。

## ■ 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 |
|---|---|
| … | 記述が省略されていることを意味する。<br>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。 |

| 記号 | 意味 |
| --- | --- |
| 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。 |
| [ ] | ウィンドウのメニューバーから選択するメニュー，コマンド，またはボタンを意味する。 |

# 目次

# 1 Cosminexus 連携機能の概要

この章では，COBOL で作成したプログラム（以降，COBOL UAP と呼びます）を Cosminexus アプリケーションサーバから利用する場合の Cosminexus 連携機能の特長と構成について説明します。

これ以降，Cosminexus 連携機能を COBOL アクセスと略名で表記しています。

## 1.1 Cosminexus 連携機能とは

Cosminexus 連携機能では，Cosminexus の Java プログラムから，COBOL で作成したプログラムを呼び出す機能を提供します。

Cosminexus 連携機能で使用できる Cosminexus の製品を次に示します。

- Cosminexus Application Server
- Cosminexus Developer

Cosminexus 連携機能の特長を次に示します。

- Java アプリケーションプログラムと COBOL で作成した UAP とで通信できるようになります。
  COBOL UAP を JavaBeans として動作させる機能と Enterprise Bean として動作させる機能（ステートフルセッション Bean だけ対応）の二つを提供します。
- Cosminexus 連携機能が提供するライブラリ（以降，COBOL アクセス用ライブラリと呼びます）と開発ツールで作成した COBOL アクセス用 Bean を使用することで，既存の COBOL UAP に修正を加えることなく Servlet や JSP などの Java プログラムから通信できます。
- Cosminexus アプリケーションサーバ上で，COBOL で作成した UAP を Enterprise Bean として通信できます。

**注意事項**

Java プログラム呼び出し機能を使用して呼び出した Java プログラムから，同じスレッド内で COBOL プログラムを呼び出すことはできません。COBOL プログラムを別プロセスで呼び出すなどの対処をしてください。Java プログラム呼び出し機能については，マニュアル「COBOL2002 Java プログラム呼び出し機能ガイド」を参照してください。

## 1.2　Cosminexus での位置づけ

Cosminexus での COBOL プログラムの位置づけについて説明します。

### 1.2.1　JavaBean の場合

Cosminexus 連携機能は，Java と Native 環境をつなぐ JNI （Java Native Interface）に特殊なラッピングをすることで，Java UAP から COBOL UAP への通信を実現します。

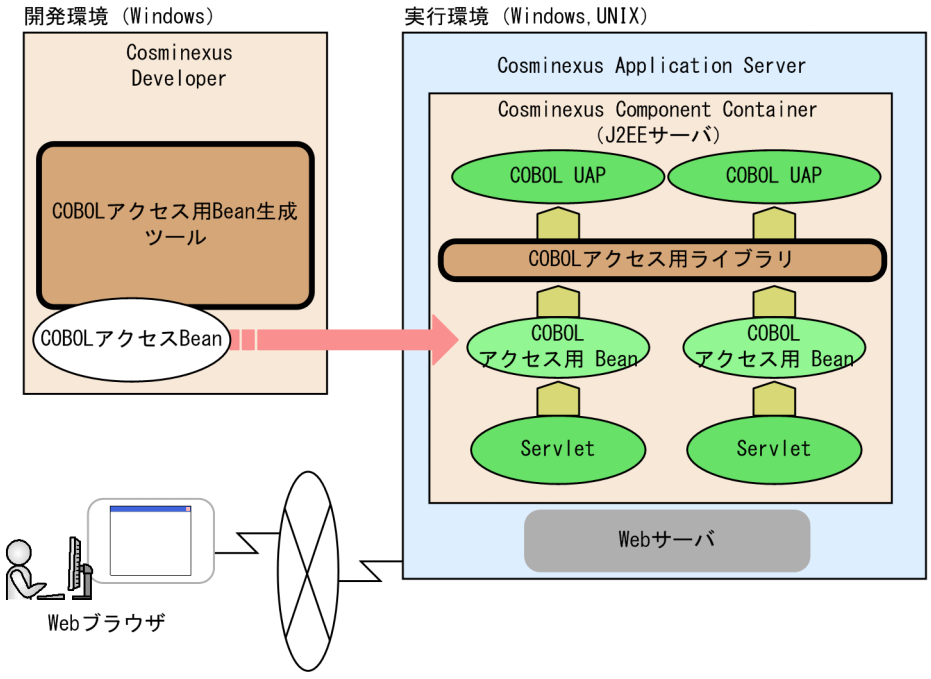
COBOL アクセス用 Bean 生成ツールは，開発環境で COBOL アクセス用 Bean を開発するときに，Java 環境から JNI を経由して COBOL UAP にアクセスするための COBOL アクセス用 Bean を生成するウィザードを提供して，簡単に開発できるよう支援します。

COBOL アクセス用ライブラリは，Java UAP から COBOL UAP へのリクエストを JNI 経由で通信させるための実行環境です。

Java UAP の実際のユーザインタフェースは，直接 JNI にアクセスするのではなく COBOL アクセス用 Bean のメソッドによるデータの受け渡しによって COBOL UAP との通信を実現します。

COBOL UAP を JavaBeans として作成する場合の Cosminexus での位置づけを図 1-1 に示します。

図 1-1　JavaBeans 対応 COBOL アクセスの Cosminexus での位置づけ




開発環境で COBOL アクセス Bean を生成し，その生成した COBOL アクセス Bean を Cosminexus 実行環境に配置することで，COBOL UAP を呼び出すことができます。

開発環境は Windows だけで，実行環境は Windows および UNIX になります。

UNIX で COBOL アクセス用 Bean を実行させる場合でも，Windows 上で Cosminexus 連携機能が提供する COBOL アクセス用 Bean 生成ツールを使って COBOL アクセス Bean を生成する必要があります。生成後の COBOL アクセス Bean を UNIX の Cosminexus 実行環境に配置することで COBOL UAP を呼び出すことができます。COBOL アクセス用 Bean 生成ツールについては「2. COBOL アクセスを使った Web アプリケーションの開発」を参照してください。また，生成後の COBOL アクセス Bean の配置については「6. プログラムの実行」を参照してください。

## 1.2.2 EJB の場合

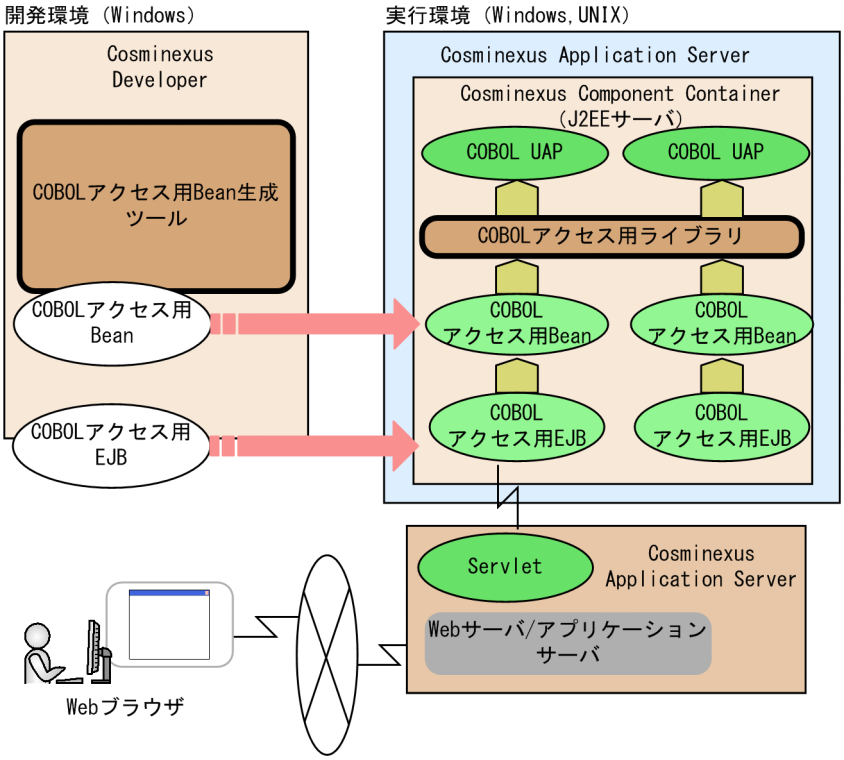
EJB 対応の COBOL アクセス機能は，COBOL アクセス用 Bean をさらに EJB クラス群（ホームインタフェース，リモートインタフェース，Enterprise Bean）でラッピングすることで実現します。

COBOL プログラムと COBOL アクセス用 Bean および EJB クラス群は，J2EE サーバまたは Java EE サーバに配置します。

この機能は，COBOL アクセス用 Bean，EJB クラス群およびデプロイ情報（DD ファイル）を生成する開発環境と EJB を動作させるための実行環境を提供します。

COBOL UAP を Enterprise Bean として作成する場合の Cosminexus での位置づけを図 1-2 に示します。

図 1-2 EJB 対応 COBOL アクセスの Cosminexus での位置づけ




開発環境で COBOL アクセス用 EJB（COBOL アクセス用 Bean 含む）を生成し，その生成した COBOL アクセス用 EJB（COBOL アクセス用 Bean 含む）を Cosminexus 実行環境に配置することで，COBOL UAP を呼び出すことができます。

開発環境は Windows だけで，実行環境は Windows および UNIX です。

UNIX で COBOL アクセス用 EJB（COBOL アクセス用 Bean 含む）を実行させる場合でも，Windows 上で Cosminexus 連携機能が提供する COBOL アクセス用 Bean 生成ツールを使って COBOL アクセス用 EJB（COBOL アクセス用 Bean 含む）を生成する必要があります。生成後の COBOL アクセス用 EJB（COBOL アクセス用 Bean 含む）を UNIX の Cosminexus 実行環境に配置することで COBOL UAP を呼び出すことができます。COBOL アクセス用 EJB の生成については，「2. COBOL アクセスを使った Web アプリケーションの開発」を参照してください。また，生成後の COBOL アクセス Bean の配置については「6. プログラムの実行」を参照してください。

## 1.3　COBOL アクセスの概要

COBOL アクセスは，Cosminexus 上の Java アプリケーションオブジェクトとして動作します。

COBOL アクセスは，COBOL UAP との通信を可能にする API（Application Program Interface）を Cosminexus 上の Java UAP のために JavaBeans または Enterprise Bean として提供します。

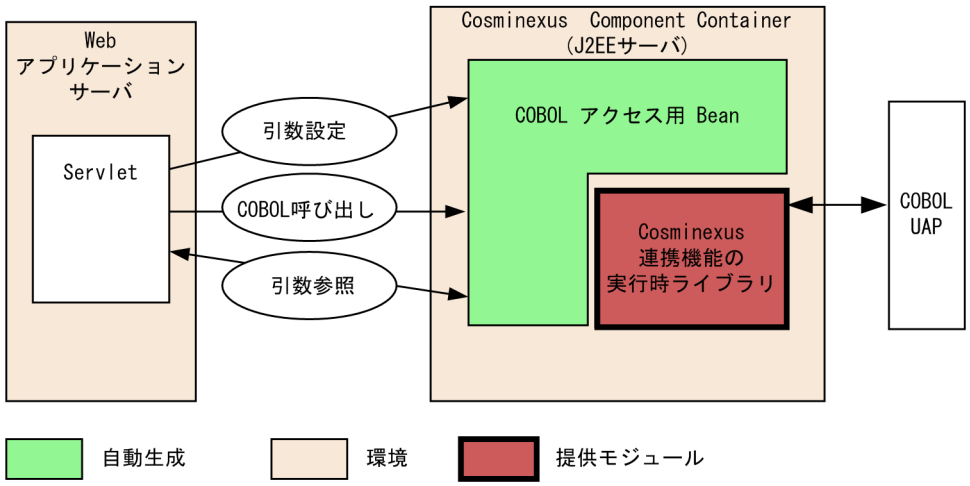
### 1.3.1　JavaBeans 対応 COBOL アクセス

JavaBeans 対応 COBOL アクセスでは，次の機能を提供します。

1. COBOL アクセス用 Bean 生成ツールの指定で，次の Java プログラムを生成する開発環境
   - COBOL アクセス用 Bean
2. JavaBeans 対応 COBOL アクセス機能を動作させるための実行環境
   - Java クラスおよびライブラリを配置して実行

JavaBeans 対応 COBOL アクセスの処理概要を図 1-3 に示します。

図 1-3　JavaBeans 対応 COBOL アクセスの処理概要




Servlet（Java UAP）は COBOL アクセス用 Bean に対して次のようなリクエストを発行します。

1. 入力データの設定のために，任意の基本項目に対して生成した COBOL アクセス用 Bean の各 setter メソッドの呼び出し。
2. 生成した COBOL アクセス用 Bean の COBOL UAP 呼び出しメソッドの実行。
3. 生成した COBOL アクセス用 Bean のデータの参照は任意の getter メソッドの呼び出し。

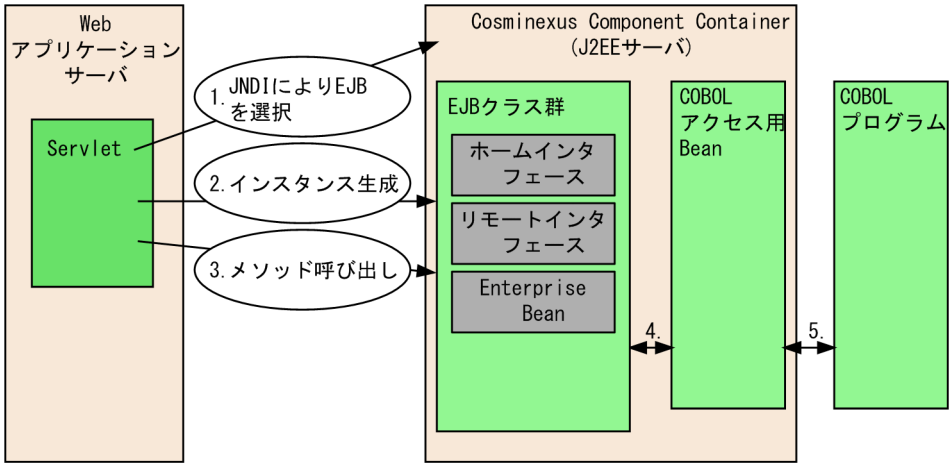
## 1.3.2　EJB 対応 COBOL アクセス

EJB 対応 COBOL アクセスでは，次の機能を提供します。

1. COBOL アクセス用 Bean 生成ツールの指定で，EJB 対応の次のプログラムおよび各種ファイルを生成する開発環境
   - COBOL アクセス用 Bean
   - ホームインタフェース
   - リモートインタフェース
   - Enterprise Bean
   - デプロイ情報（DD ファイル）
2. EJB 対応の COBOL アクセス機能を動作させるための実行環境
   - Java クラスおよびライブラリを配置して実行

   EJB 対応 COBOL アクセスの処理概要を図 1-4 に示します。

図 1-4　EJB 対応 COBOL アクセスの処理概要




次に COBOL UAP を起動するまでの流れを示します。

1. Servlet から JNDI（Java Naming and Directory Interface）の lookup メソッドに EJB の名称（EJB 配置時に指定する名称）を指定してホームインタフェースのインスタンスを取得する。
2. ホームインタフェースを経由して Enterprise Bean の create メソッドを実行してリモートインタフェースのインスタンスを取得する。
3. リモートインタフェースを経由して Enterprise Bean の COBOL アクセス用 Bean の各メソッドに対応するメソッドを呼び出す。
4. Enterprise Bean から COBOL アクセス用 Bean の該当メソッドを呼び出す。
5. COBOL アクセス用 Bean から COBOL UAP を呼び出す。

### 注意事項

COBOL アクセス用 Bean は，JNI を使って COBOL のライブラリ※を呼び出します。このため，Java の EJB から COBOL アクセス用 Bean を呼び出すことはできますが，COBOL のライブラリ※は，Cosminexus アプリケーションサーバで実行するトランザクション管理の対象外となりますので，注意してください。

注※

Windows の場合：ダイナミックリンクライブラリ（*.dll）

AIX の場合：共用ライブラリ（*.a）

Linux の場合：共用ライブラリ（*.so）

## 1.4 COBOL アクセス用 Bean の役割と処理

COBOL アクセス用 Bean は，ユーザのリクエストを COBOL UAP の API に変換します。それらは，COBOL アクセス用 Bean のメソッドとして提供されます。COBOL UAP の引数は，COBOL アクセスのプロパティとして提供され，プロパティアクセス（getter と setter）でプロパティの参照や設定をして各関数の引数への参照および設定処理を実現します。

COBOL アクセス用 Bean は，使用する COBOL UAP ごとに作成してください。作成するときは，COBOL アクセス用 Bean 生成ツールを使用します。

また，COBOL アクセスで提供するライブラリはスレッドセーフなので，Java UAP で複数のスレッドを実行して COBOL UAP へのリクエストを制御できます。

# 2 COBOL アクセスを使った Web アプリケーションの開発

この章では，COBOL アクセスを使った Web アプリケーションの開発に必要な操作を説明します。

開発環境は，Windows で提供しています。

## 2.1　Web アプリケーションの開発手順

### 2.1.1　JavaBean の場合

JavaBean として COBOL アクセスを使ったアプリケーションの開発の流れは次のようになります。

1. COBOL UAP の作成
2. JavaBeans 対応 COBOL アクセス用 Bean の作成
3. Servlet の作成
4. 入力用 HTML と結果出力用 JSP の作成

### 2.1.2　EJB の場合

Enterprise Bean として COBOL アクセスを使ったアプリケーションの開発の流れは次のようになります。

1. COBOL UAP の作成
2. EJB 対応 COBOL アクセス用 Bean の作成
3. jar ファイルの作成と配置
4. Servlet の作成
5. 入力用 HTML の作成

## 2.2 COBOL UAP の作成

COBOL アクセスを使用する場合に，COBOL UAP を作成する際の前提条件や注意事項について説明します。

### 2.2.1 COBOL UAP プログラムソースの構成

COBOL UAP を作成する場合，次の構成で作成することを推奨します。また，既存の COBOL UAP がある場合，ここで説明している仕様に準拠していれば，そのまま Java のクライアントから COBOL アクセスを経由して実行できる COBOL UAP として利用できます。

#### (1) COBOL プログラムのソース構成

1. COBOL プログラムソース
2. COBOL-Java 間で渡す引数の登録集原文
   （COBOL アクセス用 Bean 生成時に，必要な登録集原文）

#### (2) COBOL プログラムの作成例

COBOL プログラムの作成例を次に示します。

```
  IDENTIFICATION  DIVISION.
  PROGRAM-ID.     SEARCHCBL.
  DATA            DIVISION.
  LINKAGE SECTION.
    COPY   SEARCHCBLCOPY.     ·····  ※
*>  01 PERSONAL-DATA.                    ┐
*>    05 P-NUMBER   PIC 9(9) USAGE COMP.  │
*>    05 P-NAME     PIC X(50).            ├ 登録集原文の展開
*>    05 P-ADDRESS  PIC X(100).           │
*>    05 P-GIF      PIC X(50).            ┘
 PROCEDURE DIVISION   USING PERSONAL-DATA.
     IF P-NUMBER = 100001  THEN
       MOVE  '日立 一郎'       TO  P-NAME
       MOVE  '日立市'          TO  P-ADDRESS
       MOVE  '/ICHIRO.GIF'     TO  P-GIF
     ELSE
                   :
    END-IF.
    EXIT  PROGRAM.
  END PROGRAM  SEARCHCBL.
```

注※

LINKAGE SECTION の各データ名定義は，登録集にすることを推奨します。

COBOL アクセス用 Bean 生成時に，この登録集を使用できます。

## 2.2.2 COBOL アクセスを使用するための COBOL UAP の引数の規則

COBOL アクセスを使用するためには，COBOL UAP の引数は次の規則に従っている必要があります。

**【書き方】**

```
レベル番号 ［データ名］
[REDEFINES データ名2]
[{PICTURE|PIC} IS 文字列 ]

[ [USAGE IS ] { ADDRESS | BINARY | COMPUTATIONAL | COMP | COMPUTATIONAL-1 | COMP-1 |
                COMPUTATIONAL-2 | COMP-2 | COMPUTATIONAL-3 | COMP-3 | COMPUTATIONAL-4 |
                COMP-4 | COMPUTATIONAL-5 | COMP-5 | DISPLAY | NATIONAL | PACKED-DECIMAL |
                POINTER } ]

[OCCURS 整数 TIMES]
[ [SIGN IS] {LEADING|TRAILING} [SEPARATE CHARACTER] ]
```

注

【書き方】の規則（括弧や下線などの意味）は，マニュアル「COBOL2002　言語　標準仕様編」の「1. 記述技法」に従っています。

**【構文規則】**

1. レベル番号，データ名，REDEFINES 句，PICTURE 句，USAGE 句，OCCURS 句，SIGN 句だけが記述できます。
2. レベル番号は，1 けたから 2 けたの符号なし整数で 1 から 49 までの範囲内，または 77 でなければなりません。
3. データ名を省略した場合は，FILLER を仮定します。
4. USAGE 句は，【書き方】に記述しているものだけ使用できます。それ以外の属性は使用できません。
5. OCCURS 句は，【書き方】に記述しているものだけ使用できます。整数の値は 1 以上でなければなりません。
6. PICTURE 句の文字列に P を含んではなりません。
7. PICTURE 句の小数点を表す文字は常にピリオドです。
8. レベル番号，データ名，REDEFINES 句，PICTURE 句，USAGE 句，OCCURS 句，SIGN 句の構文規則で上記の構文規則で規定されている規則以外はマニュアル「COBOL2002　言語　標準仕様編」および「COBOL2002　言語　拡張仕様編」の構文規則に従ってください。

【一般規則】

1. REDEFINES 句，USAGE 句，OCCURS 句，SIGN 句は，【書き方】で記す表現形式の範囲では，マニュアル「COBOL2002　言語　標準仕様編」および「COBOL2002　言語　拡張仕様編」の一般規則に準拠します。

【使用上の注意事項】

1. SIGN 句に SEPARATE CHARACTER 指定を書かない場合は，符号の表現形式は処理系によって異なることがあります。

   このシステムではマニュアル「COBOL2002　言語　標準仕様編」および「COBOL2002　言語　拡張仕様編」の表現形式を仮定します。

2. 外部浮動小数点項目および外部ブール項目などは使用できません。使用できない COBOL 定義項目一覧については「付録 A.1　COBOL アクセス用 Bean 生成時の COBOL 定義注意事項／制限事項」を参照してください。付録 A.1 では回避方法も記載しています。

3. REDEFINES 句を使用して被再定義項目と異なるデータ属性で領域を再定義した場合，格納されているデータの形式にあったデータ項目の getter を使用しなかった場合，データは正しく取得できません。

4. USAGE 句に ADDRESS，POINTER を指定した場合は，値渡し（BY VALUE 指定）で受け取ってください。

## 2.2.3　COBOL UAP の引数を含む登録集原文の記述規則

1. COBOL UAP の引数の定義は構文的に正しいものでなければなりません。
2. COBOL UAP の引数の定義はすべての引数が一つの登録集原文中で完結していなければなりません（登録集原文中に COPY 文を記述できます。詳細は「2.2.5　COPY 文」を参照してください）。
3. 登録集原文の書式は固定形式を標準としています。

   登録集原文の固定形式・自由形式の切り替えは入力ファイルの拡張子とします。

   拡張子が「.cbf」および環境変数 CBLFREE に設定した拡張子のファイルを自由形式，それ以外は固定形式として扱います。

   次に固定形式と自由形式の記述規則を示します。

   **(固定形式規則)**

   - 1～6 カラム目および 73 カラム目以降の記述を無視します。
   - 1～6 カラム目は行番号としては使用されません。また，エラー情報に出力される行番号としても使用されません。
   - 7 カラム目が「*」，「/」の行をコメント行として扱います。
   - デバッグ行（7 カラム目が「D」，「d」の行）をコメント行として扱います。
   - コメント行以外の行の 72／73 カラム目に全角文字が記述された場合，解析エラーとなります（ただし，全角空白の場合を除く）。

- 7 カラム目に「*」,「/」，空白，タブ,「D」,「d」以外の文字が書かれた場合，解析エラーとなります。
- 6／7 カラム目，7／8 カラム目に全角文字が記述された場合，解析エラーとなります（ただし，全角空白の場合を除く）。

(自由形式規則)

- 1 文字目が「*」,「/」,「＊」（全角),「／」（全角）の行をコメントとして扱います。
- 1 文字目が「D」,「d」,「D」（全角),「ｄ」（全角）で，かつ 2 文字目が半角空白，全角空白，改行(¥n または¥r¥n)，タブ，EOF の行をコメントとします。
- 行内注記（「*>」）はサポートしていません（ただし，1 文字目から書かれた場合コメント行扱い)。

(環境変数 CBLFREE に設定する拡張子の規則)

- 設定する拡張子は，先頭のピリオド（.）と 3 文字以内の英数字で設定します。
- 複数の拡張子を設定する場合は，半角空白で区切ります。ただし，.cbf は環境変数に指定しなくても自由形式として扱います。
- 拡張子の 3 文字の英字は，大文字小文字等価とします。
- 環境変数内の値が，規則に反している場合，エラーになった個所以降の値は固定形式として扱われます。

4. コンマ（,）とセミコロン（；）は分離符として扱いません。
5. このシステムでは次の語だけを予約語とみなします。これらの予約語以外は利用者定義用語として扱います。

   ADDRESS，BINARY，CHARACTER，COMP，COMP-1，COMP-2，COMP-3，COMP-4，COMP-5，COMPUTATIONAL，COMPUTATIONAL-1，COMPUTATIONAL-2，COMPUTATIONAL-3，COMPUTATIONAL-4，COMPUTATIONAL-5，COPY，DEPENDING，DISPLAY，IS，LEADING，NATIONAL，OCCURS，ON，PACKED-DECIMAL，PIC，PICTURE，POINTER，REDEFINES，SEPARATE，SIGN，TIMES，TRAILING，USAGE
6. タブ文字は 1 個の空白文字として扱います。
7. データ記述項の記述では全角文字と半角文字は非等価とします。

## 2.2.4 可変長データについて

Cosminexus 連携機能で使用できる可変長データは,「OCCURS DEPENDING ON」で指定できる可変長とは異なります。

データの形式は次のようになります。

| 長さ | バイト配列データ |
|---|---|

「長さ」：後続する「バイト配列データ」のデータ長（長さエリアは含まない）

COBOL UAP では，次のような形式です。

```
 01  VARDATAS.
  02  VARDATALEN   PIC  S9(9)  USAGE COMP.
  02  VARDATAITEM  PIC  X(nn).

nn：データ長を指定します
```

COBOL UAP の引数の記述は，次の 2 とおりあります。

## (1) 01 および 77 レベルのアドレスデータ項目を使用した可変長データの指定

＜指定例＞

```
01    VARDATA  USAGE ADDRESS.
```

COBOL UAP には，上記データの先頭アドレスが渡されるので，BY VALUE 指定で引数を受け取ります。

```
PROCEDURE DIVISION USING BY VALUE VARDATA.
```

受け取ったデータは「アドレス」なので，内容を参照するために「アドレス名にアドレスを設定して参照する」ことになります。

```
 :
WORKING-STORAGE SECTION.
 01  VARDATAS  ADDRESSED BY VARDATAS-ADDR.
  02  VARDATASLEN  PIC S9(9)  USAGE COMP.
  02  VARDATASITEM  PIC  X(10000).
 :
LINKAGE SECTION.
 01  VARDATA  USAGE ADDRESS.
PROCEDURE DIVISION  USING BY VALUE VARDATA.
    COMPUTE VARDATAS-ADDR=VARDATA.
    IF VARDATASLEN=ZERO THEN
       :
```

## (2) 01 および 77 レベルの英数字項目を使用した可変長データの指定

＜指定例＞

```
01    VARDATA  PIC X(nn).          nn：データ長を指定します
01    VARDATAS  REDEFINES VARDATA.
 02   VARDATASLEN  PIC S9(9)  USAGE COMP.
 02   VARDATASITEM PIC X(10000).
```

ただし，このままでは「英数字項目」として扱うため，COBOL アクセス用 Bean 生成ツールでデータ属性に「可変長データ(byte[])」を指定する必要があります。再定義項目は，COBOL UAP でだけ使用できます。引数は BY REFERENCE で受け取ります。

```
PROCEDURE DIVISION USING BY REFERENCE VARDATA.
```

受け取ったデータは再定義項目で参照してください。

```
 :
LINKAGE SECTION.
  01  VARDATA   PIC  X(10000).
  01  VARDATAS  REDEFINES VARDATA..
   02  VARDATASLEN  PIC S9(9)  USAGE COMP.
   02  VARDATASITEM  PIC  X(10000).
 PROCEDURE DIVISION  USING BY REFERENCE VARDATA.
     IF VARDATASLEN=ZERO THEN
       :
```

上記二つの指定方法の違いは，「可変長データの最大長をあらかじめ指定するかどうか」です。最大長が COBOL プログラムで記述できる最大領域長を超える可能性がある場合は「アドレスデータ項目」を使用してください。最大長が COBOL プログラムで記述できる最大領域長を超えない場合は「英数字項目」で指定することもできます。

可変長データに対する Java プログラムでの操作は次のようになります。

```
setter
    setVardata(byte[] data, int len)
getter
    getVardata()
```

setter では，データ自身をバイト配列データで指定します。このデータは，文字列データのような「文字コード変換」は行わない点に注意してください。また，データ長を len で指定します。

getter では，byte 配列が取得されます。byte 配列の長さは length プロパティで取得できます。

詳細の指定は，以降の章を参照してください。

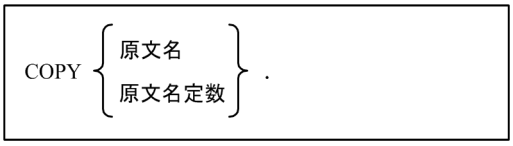
## 2.2.5 COPY 文

COBOL アクセスでの COPY 文の機能について説明します。

### (1) 言語規則

#### (a) 形式

COPY 文の指定形式は次のようになります。




### (b)　機能

COPY 文は，COBOL 登録集原文の中へ別の原文を複写します。

### (c)　構文規則

1. このシステムでは上記形式以外の構文はサポートしません。
2. COPY 文は，空白のあとに続け，分離符の終止符でやめなければなりません。
3. COPY 文を記述した行は，完結した COPY 文だけで構成されていなければなりません。
4. 原文名を構成する文字は，すべて半角で英文字（A～Z，a～z），数字，ハイフン，および下線とし，先頭は英文字でなければなりません。また，全角文字は使用できません。
5. 上記以外の構文規則は，「2.2.2　COBOL アクセスを使用するための COBOL UAP の引数の規則」に従ってください。

### (d)　一般規則

1. COPY 文を含む原文の解析は，論理的には，すべての COPY 文を処理してから，原文の解析処理を行うことと同じです。
2. COPY 文を処理すると，予約語 COPY に始まり終止符で終わる COPY 文全体が，原文名または原文名定数に対する原文で論理的に置き換わり，元の原文中に複写されます。
3. 登録集原文が文法規則に従っているかどうかは，原文だけでは決定できません。COPY 文を除いて，COBOL 登録集原文全体が文法規則に従っているかどうかは，すべての COPY 文が完全に処理されるまで決定できません。
4. COPY 文によって複写される原文の中に COPY 文がある場合，COPY 文が入れ子（nest）になっているといいます。
   - COPY 文の入れ子は 19 レベルまで許されます。
   - 再起的な複写は直接的にも間接的にも行ってはなりません。
5. 固定形式の原文と自由形式の原文を混在して使用してはなりません。
6. 原文名には，登録集原文が登録されているファイルの名称を，拡張子を付けないで指定します。登録集原文の検索順序については「(2)　登録集原文の検索順序」を参照してください。
7. 原文名定数は，登録集原文が登録されているファイルの絶対パス名を引用符（”）またはアポストロフィ（’）で囲んで指定します。このとき，ファイル名には拡張子も付けてください。絶対パス名に全角文字は使用できません。

## (2)　登録集原文の検索順序

原文名で指定したファイルは，拡張子，フォルダの二つの条件で検索されます。それぞれの検索順序は次の順序であり，両者のうちでは拡張子による検索順序が優先します。

### (a)　拡張子による検索順序

1. 環境変数 CBLFREE で設定した自由形式拡張子
2. .cbl
3. .cob
4. .cbf

### (b)　フォルダによる検索順序

1. 環境変数 CBLLIB で指定したフォルダ
2. 「COBOL アクセス用 Bean 生成ツールーステップ 1/3」画面で指定した原文が存在するフォルダ

例えば，原文名を"ARGUMENT_CPY"とした場合，環境変数 CBLFREE が設定されていないとき，"ARGUMENT_CPY.cbl"で 1.2.の順にフォルダを検索し，目的のファイルがなければ，次に"ARGUMENT_CPY.cob"で同様に検索します。

## (3)　環境変数

COPY 文では，環境変数 CBLFREE と CBLLIB を使用します。

### (a)　CBLFREE

自由形式正書法で書かれた原文として使用する登録集原文の拡張子を設定します。詳細は「2.2.3　COBOL UAP の引数を含む登録集原文の記述規則」を参照してください。

### (b)　CBLLIB

登録集原文を格納するフォルダを設定します。フォルダを複数指定する場合は，セミコロン（；）で区切って指定してください。また，フォルダに全角文字は使用できません。

（例）

```
CBLLIB=c:¥copylib;c:¥test
```

## 2.3　COBOL アクセス用 Bean の生成

ここでは，COBOL アクセス用 Bean および EJB の作成方法について説明します。

COBOL アクセス用 Bean および EJB は，COBOL アクセス用 Bean 生成ツールで生成します。COBOL アクセス用 Bean 生成ツールについては，「2.3.1　COBOL アクセス用 Bean 生成ツールによる生成」を参照してください。

生成された Bean は Javadoc に対応しています。

以降，COBOL アクセス用 Bean 生成ツールを「Bean 生成ツール」と表記します。

### 2.3.1　COBOL アクセス用 Bean 生成ツールによる生成

「Bean 生成ツール」を使用して，COBOL UAP を呼び出すロジックを組み込んだ COBOL アクセス用 Bean を生成します。

#### (1)　起動方法

Windows のプログラムを開始する操作で，［COBOL2002］－［Cosminexus 連携機能］をポイントし，［COBOL アクセス用 Bean 生成ツール環境設定］をクリックします。「COBOL アクセス用 Bean 生成ツール環境設定」の画面が開いたら，その画面で環境設定を行います。

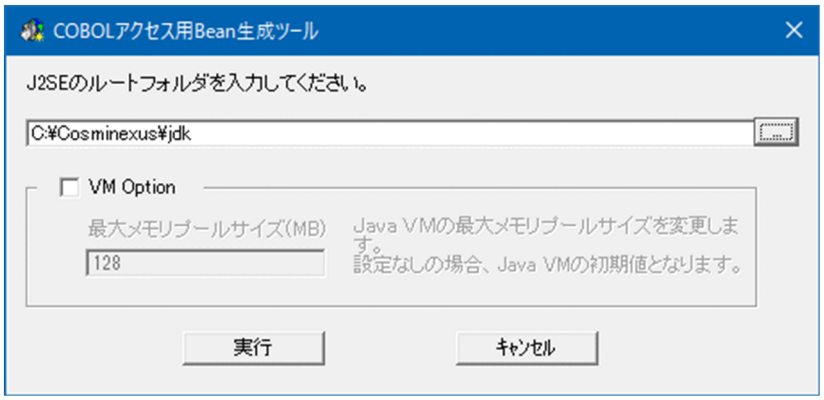
起動方法も同じように，Windows のプログラムを開始する操作で［COBOL2002］－［Cosminexus 連携機能］をポイントし，「COBOL アクセス用 Bean 生成ツール」をクリックすると起動できます。

Windows(x86) COBOL2002 と Windows(x64) COBOL2002 では，スタートメニューに表示されるメニュー名とウィンドウに表示されるタイトル名が異なります。ここでは，Windows(x86) COBOL2002 のメニュー名とタイトル名を使用します。Windows(x64) COBOL2002 を使用する場合は，次のように読み替えてください。

表 2-1　Windows(x86) COBOL2002 と Windows(x64) COBOL2002 のメニュー名とタイトル名

| Windows(x86) COBOL2002 | | Windows(x64) COBOL2002 | |
|---|---|---|---|
| スタートメニューに表示されるメニュー名 | ウィンドウに表示されるタイトル名 | スタートメニューに表示されるメニュー名 | ウィンドウに表示されるタイトル名 |
| COBOL アクセス用 Bean 生成ツール環境設定 | COBOL アクセス用 Bean 生成ツール | COBOL アクセス用 Bean 生成ツール環境設定 for COBOL2002 64bit | COBOL アクセス用 Bean 生成ツール for COBOL2002 64bit |
| COBOL アクセス用 Bean 生成ツール | | COBOL アクセス用 Bean 生成ツール for COBOL2002 64bit | |

## (2)　環境設定ダイアログ



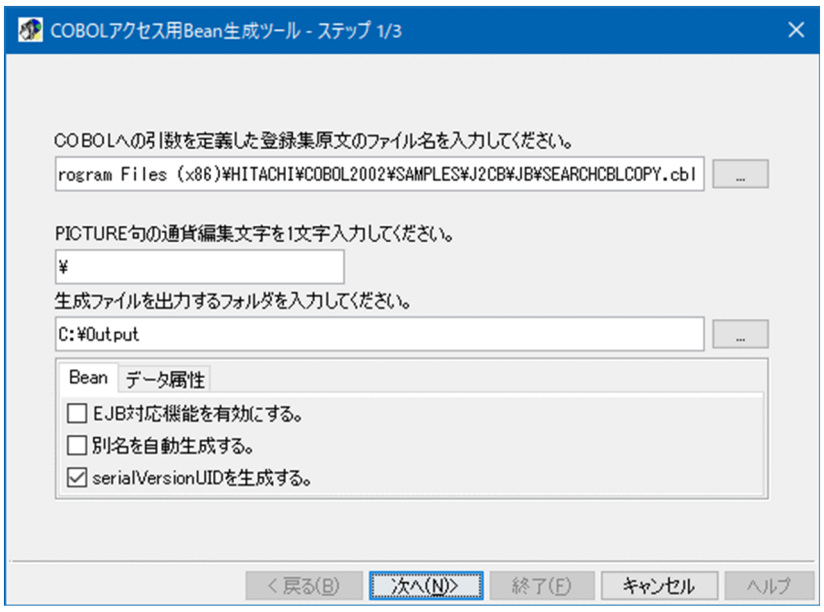

VM Option のチェックボックスをオンにすると最大メモリプールサイズを指定できるようになります。

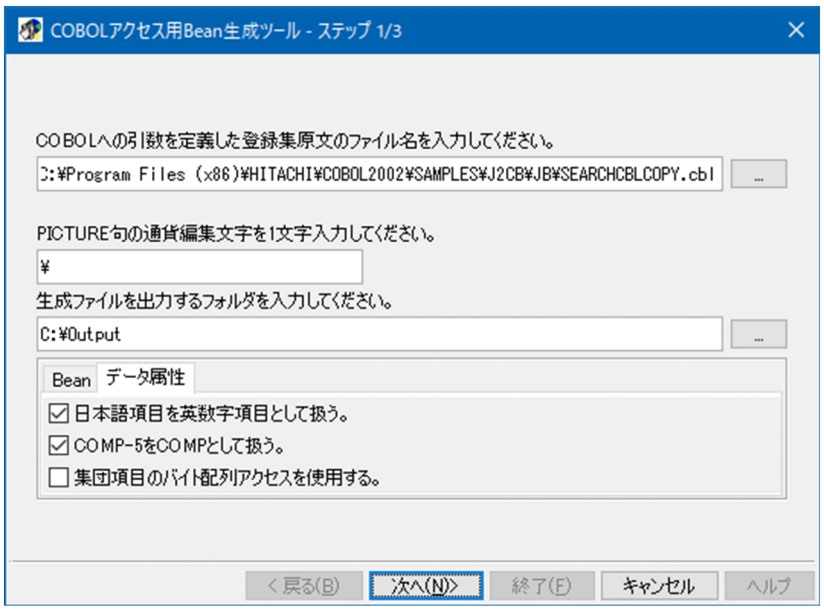
指定がない場合は，Java VM の初期値となります。

また，VM Option のチェックボックスがオンの状態で値を指定しない場合，または数値以外を指定した場合は設定エラーダイアログを出力します。

## (3)　COBOL アクセス用 Bean 生成ツールの画面

### (a)　ステップ 1/3 画面






この画面では，COBOL への引数を定義した登録集原文ファイル，PICTURE 句の通貨編集用文字，オプションおよび生成ファイルを出力するフォルダなどを指定します。オプションパネルはタブで切り替えることができます。指定が終了し，［次へ(N)>］ボタンを押すと「COBOL アクセス用 Bean 生成ツール － ステップ 2/3」画面に進みます。

- COBOL への引数を定義した登録集原文の指定

  COBOL UAP の引数を定義した登録集原文のファイル名※をフルパスで指定します。右側のボタンをクリックすると，参照ダイアログが表示されます。

  注※

  登録集原文のファイル名に全角文字は使用できません。

- PICTURE 句の通貨編集用文字の指定

  必要に応じて，PICTURE 句の通貨編集用文字を変更します。デフォルトでは，'¥'が指定されています。

- 生成ファイルを出力するフォルダの指定

  ツールが生成する Java ソースファイルを出力するフォルダ名をフルパスで指定します。Java ソースファイルは次のフォルダ規則に従って生成されます。生成される Java ソースファイル名は，ステップ 3/3 のクラス説明を参照してください。

  ＜生成ファイルのフォルダ規則＞

  - 指定フォルダ¥パッケージ名¥

    パッケージ名は，ステップ 3/3 で指定するパッケージ名を指します。

  - 指定フォルダ¥META-INF¥

    EJB 対応機能を有効にした場合は，Java ソースファイル群に加え，このフォルダ下に ejb_jar.xml も生成されます。

  「生成ファイルを出力するフォルダを入力してください。」の右側のボタンを押すと，参照ダイアログが表示されます。なお，指定したフォルダが存在しない場合は，そのフォルダを作成するかどうかのダイ

アログボックスが出力されます。「はい」を選んだ場合は，そのフォルダを作成し生成処理を続行します。「いいえ」を選んだ場合は，次の画面に移りません。再度，設定をやり直してください。

- EJB 対応機能

  EJB 対応 COBOL アクセス用 Bean とその他の EJB 関連ファイルを出力する場合は，チェックボックスをオンにします。 デフォルトでは，JavaBeans 対応 COBOL アクセス用 Bean だけを生成します。

- 別名の扱い

  同じデータ名のデータ項目に対して別名を自動生成する場合は，チェックボックスをオンにします。デフォルトでは，別名は自動生成しません。

- serialVersionUID の生成

  serialVersionUID を生成する場合は，チェックボックスをオンにします。デフォルトでは，serialVersionUID を生成します。

  チェックボックスをオンにすると，COBOL アクセス用 Bean のソース中に次の形式で serialVersionUID を生成します。

  (生成例)

  ```
  private static final long serialVersionUID = 1153786720640L;
  ```

  serialVersionUID の値は Bean 生成時点での 1970 年 1 月 1 日 00:00:00 GMT からの経過時間(ミリ秒)です。

- 日本語項目の扱い

  日本語項目および日本語編集項目を日本語項目として扱いたい場合はチェックボックスをオフにします。デフォルトでは，日本語項目および日本語編集項目を英数字項目として扱います。

- USAGE COMP-5 の扱い

  USAGE COMP-5 を指定した 2 進項目を COMP-5 として扱いたい場合はチェックボックスをオフにします。デフォルトでは，USAGE COMP-5 を指定した 2 進項目は，COMP として扱います。

- 集団項目の扱い

  集団項目に対するデータの設定および取得をする場合は，チェックボックスをオンにします。デフォルトでは，集団項目に対するデータの設定および取得はしません。

- [次へ(N)>] ボタン

  「COBOL アクセス用 Bean 生成ツール － ステップ 2/3」画面に進みます。

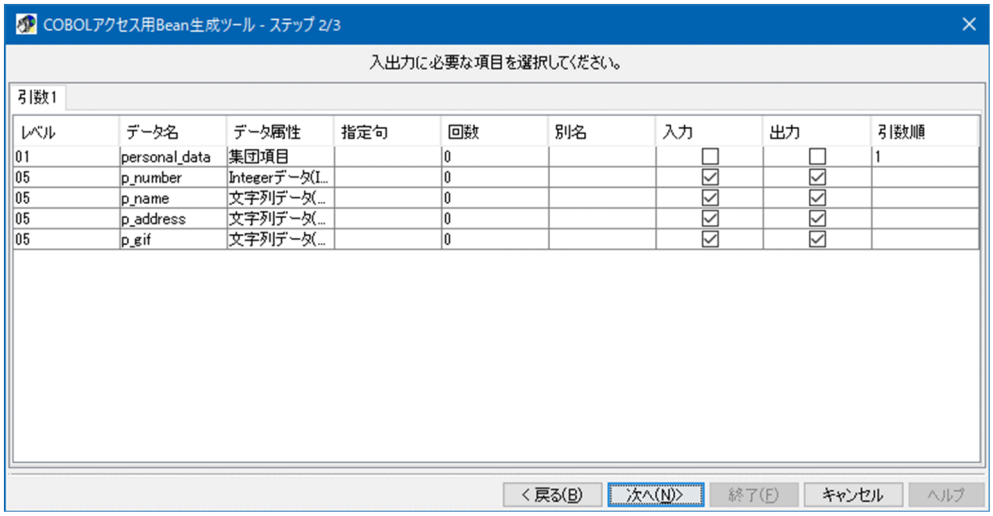
- [キャンセル] ボタン

  COBOL アクセス用 Bean の生成を取り消し，「Bean 生成ツール」画面を消去します。

次の「Bean 生成ツール」画面についても，[キャンセル] ボタンについての動作は同じです。

指定した登録集原文を正常に解析できなかった場合，エラーメッセージが表示されて処理が中止します。

## (b)　ステップ 2/3 画面




「COBOL アクセス用 Bean 生成ツールーステップ 2/3」画面には，「COBOL アクセス用 Bean 生成ツールーステップ 1/3」画面で指定した登録集原文を COBOL アクセスが解析した情報が引数ごとにテーブル表示されます（これ以降，このテーブルのことを「パラメタテーブル」とします)。

このパラメタテーブル中から，COBOL UAP で実際に使用する引数を選択します。選択したデータ項目に対して COBOL アクセス用 Bean の中に該当するデータ項目に対するアクセスメソッドが生成されます。

COBOL UAP で実際に使用するデータ項目だけを抽出することで，COBOL アクセス用 Bean のサイズを COBOL UAP のサービスを受け取れる必要なサイズに最適化でき，資源を有効に活用できます。

複数の引数がある場合，タブで表示する引数を切り替えます。

すべての引数の設定が完了したあと，［次へ(N)>］ボタンを押すと，「COBOL アクセス用 Bean 生成ツールーステップ 3/3」画面に進みます。

「COBOL アクセス用 Bean 生成ツールーステップ 2/3」画面に表示される表の各フィールドの編集方法について説明します。

- レベルフィールド

  指定された登録集原文に定義されたデータ名のレベル番号を表示します。

  このレベルフィールドを編集することはできません。

- データ名フィールド

  指定された登録集原文に定義されたデータ名を表示します。

  このとき，次の条件に該当するデータ項目名が変換されます。

  - Java の言語仕様によって，ハイフン（－）をデータ名として使えないため，COBOL のデータ項目名にハイフン（－）が含まれる場合，自動的に下線（_）に変換して表示し，対応するソース生成時のプロパティ名も下線（_）で生成します。

- 「Bean 生成ツール」ではプロパティを基本項目のデータ名から生成するため，基本項目のデータ名は修飾なしで一意でなければなりません。また，集団項目のバイト配列アクセスを使用する場合は，すべてのデータ名が修飾なしで一意でなければなりません。規則に反した場合，エラーとなります。
- JavaBeans の命名規則によって，COBOL のデータ項目名に英大文字が含まれる場合，自動的に英小文字に変換して表示し，対応するソース生成時のプロパティも英小文字で生成します。ただし，メソッドの場合，先頭の 1 文字だけを英大文字にして生成します。
- このデータ名フィールドを編集することはできません。
- FILLER は，$00001，$00002…のように先頭 1 文字が"$"で残り 5 文字が昇順の番号という名称でデータ名フィールドに表示します。FILLER 項目の最大数は 65,535 個で，65,535 個を超えた場合はエラーメッセージを出力して処理を中止します。

- データ属性フィールド

  指定された登録集原文に定義されたデータ属性を表示します。

  このデータ属性フィールドを編集することはできません。

  各データ項目に対応する表示文字列を次の表に示します。

### 表 2-2　各データ項目に対応する表示文字列

| データ項目 | 表示文字列（Java でのデータ属性を表示） |
|---|---|
| 英字項目<br>英数字項目<br>英数字編集項目<br>数字編集項目 | 文字列データ(String)<br>バイト配列データ(byte[])※1<br>可変長データ(byte[])※1，※2 |
| 日本語項目<br>日本語編集項目 | 文字列データ(String)<br>バイト配列データ(byte[])※1<br>可変長データ(byte[])※1，※2<br>日本語データ(String)※4 |
| アドレスデータ項目 | アドレスデータ(byte[])※1，※3 |
| 単精度内部浮動項目 | 単精度データ(Float) |
| 倍精度内部浮動項目 | 倍精度データ(Double) |
| 1～4 けたの小数を含まない 2 進項目 | Short データ(Short) |
| 1～4 けたの小数を含む 2 進項目 | 10 進データ(BigDecimal) |
| 5～9 けたの小数を含まない 2 進項目 | Integer データ(Integer) |
| 5～9 けたの小数を含む 2 進項目 | 10 進データ(BigDecimal) |
| 10～18 けたの小数を含まない 2 進項目 | Long データ(Long) |
| 10～18 けたの小数を含む 2 進項目 | 10 進データ(BigDecimal) |
| 外部 10 進 | 10 進データ(BigDecimal) |
| 内部 10 進 | 10 進データ(BigDecimal) |

| データ項目 | 表示文字列（Java でのデータ属性を表示） |
|---|---|
| 集団項目 | 集団項目<br>集団項目(byte[])※5 |

注※1

バイト配列データ，可変長データおよびアドレスデータを指定した場合，COBOL プログラムと受け渡すデータは，文字列データ(String)指定時と同様のコード変換は行いません（無変換）。コード変換が必要な場合は，Java プログラムまたは COBOL プログラムで行ってください。

また，バイト配列データ，可変長データおよびアドレスデータを指定した場合のデータ属性フィールドの指定，およびデータ形式については，「2.3.2　COBOL アクセス用 Bean 生成ツールでのバイト配列指定」を参照してください。

注※2

REDEFINES 句の指定のない 01 または 77 レベルで，長さが 5 バイト以上の英数字項目にだけ指定できます。また，可変長データを再定義することはできません。指定した場合，再定義項目は従属する下位項目も含めて無視されます。可変長データを指定したデータ項目は，データ長を表す 4 バイトの 2 進項目領域と，データを表す（全体の領域長−4）バイトのバイト配列領域と見なされます。この指定をすることで，COBOL に渡す領域長をデータ項目長よりも短くすることができます（使用例は「付録 F　プログラム例」を参照してください）。

注※3

REDEFINES 句の指定のない 01 または 77 レベルだけ指定できます。また，アドレスデータを再定義することはできません。指定した場合，再定義項目は従属する下位項目も含めて無視されます。アドレスデータを指定したデータ項目は，データ長を表す 4 バイトの 2 進項目領域とデータを表すバイト配列領域と見なされます。

注※4

「COBOL アクセス用 Bean 生成ツール−ステップ 1/3」画面で「日本語項目を英数字項目として扱う。」チェックボックスをオフにした場合，「日本語項目(String)」と表示します。この場合，バイト配列データおよび可変長データへの変更はできません。

注※5

「COBOL アクセス用 Bean 生成ツール−ステップ 1/3」画面で「集団項目のバイト配列アクセスを使用する。」チェックボックスをオンにした場合，「集団項目(byte[])」と表示します。

- 指定句フィールド

  各データ項目に指定された句の情報を表示します。

  次の内容が表示されます。

  「R」：REDEFINES 句の指定がある（再定義項目である）

  なお，PICTURE 句，USAGE 句，および OCCURS 句の情報は，ほかのフィールドで表示しているので表示しません。

  この指定句フィールドを編集することはできません。

- 回数フィールド

  回数は OCCURS 句がある場合，その回数を表示します。OCCURS 句がない場合，0 を表示します。

  この回数フィールドを編集することはできません。

- 別名フィールド

  日本語データ名，修飾なしで一意にならないデータ名に対する別名を入力する領域です。該当個所を選択すると，編集モードになります。 別名の編集を行った項目は，対応するソース生成時のプロパティ名が別名に置換されます。データ名と異なり，ハイフンのアンダーバーへの変換および英大文字の小文字への変換は行いません。別名の編集用途は，次のとおりです。

- データ名が日本語（データ名に英小文字，数字，ハイフン（－），下線（__）以外が含まれている場合。ただし FILLER の変換後の名称は除く）の場合は，必ず指定しなければなりません。これは，Java で日本語メソッド名および変数名を使用するのを避けるために設けた規則です。別名は，半角英数字でなければなりません。規則に反した場合，エラーとなります。
- 「Bean 生成ツール」ではプロパティ名を基本項目のデータ名から生成するため，基本項目のデータ名は修飾なしで一意でなければなりません。また，「COBOL アクセス用 Bean 生成ツールーステップ 1/3」画面で「集団項目のバイト配列アクセスを使用する。」のチェックボックスをオンにした場合は，すべてのデータ名が修飾なしで一意でなければなりません。規則に反した場合，エラーとなります。

なお，「COBOL アクセス用 Bean 生成ツールーステップ 1/3」画面で「別名を自動生成する。」チェックボックスをオンにした場合，すべてのデータ名に対して次の規則で別名を自動生成します。

1. 引数ごとに同名チェックを行う。
2. 同名のデータ名があった場合，1 つ目のデータ名には別名を生成しない。
3. 2 つ目以降のデータ名に対し，「元のデータ名$1」，「元のデータ名$2」...の順で別名を生成する。

- 入力／出力フィールド

  使用する COBOL のデータ項目（すなわち，COBOL アクセス用 Bean のプロパティ）を選択します。ただし，基本項目だけの指定であり，集団項目は「COBOL アクセス用 Bean 生成ツールーステップ 1/3」画面で「集団項目のバイト配列アクセスを使用する。」チェックボックスをオンにした場合だけ指定できます。デフォルトは，FILLER 以外の基本項目の入力／出力フィールドはすべて選択（チェックボックスオン）の状態になります。不要なデータはチェックボックスをオフにしてください。

  選択したデータ項目に対して COBOL アクセス用 Bean の中に setter／getter が生成されます。
  COBOL UAP で実際に使用するデータ項目だけを抽出することで，COBOL アクセス用 Bean のサイズを COBOL UAP のサービスを受け取れる必要なサイズに最適化でき，資源を有効に活用できます。

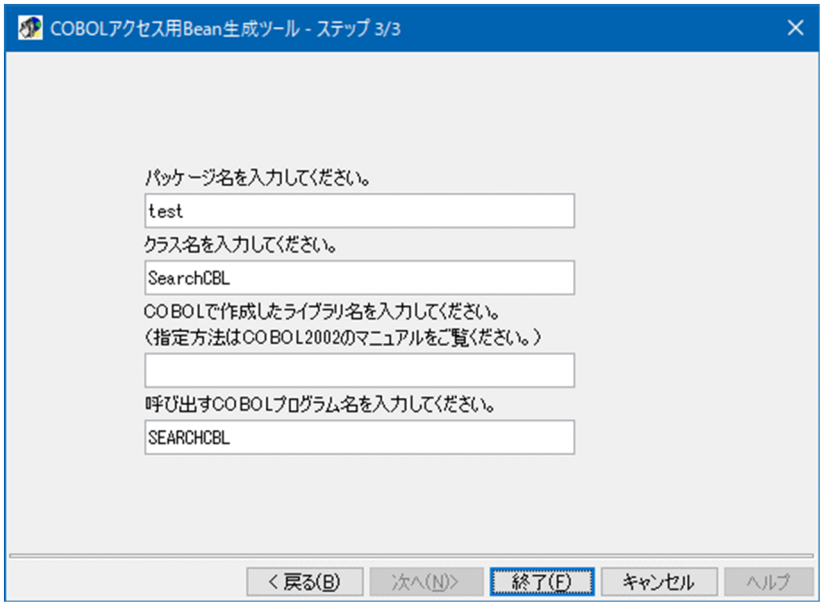
- 引数順フィールド

  PROCEDURE DIVISION USING の引数に指定した順に番号を設定します。

  01，77 レベルのデータ項目に対してだけ入力できます。該当個所を選択すると，編集モードになります。

  入力は数字だけであり，同じ番号を指定してはなりません。

### (c)　ステップ 3/3 画面




「COBOL アクセス用 Bean 生成ツールーステップ 3/3」画面には，COBOL アクセス用 Bean を生成するためのパッケージ名などを指定する画面が現れます。ここでは，次の情報を入力します。

- パッケージ名

  プロジェクトファイルから派生したパッケージ名が表示されます。ほかのパッケージ名を付けるには，このフィールドをクリックして新規の名前を入力します。

- クラス名

  クラス名を入力します。

  「COBOL アクセス用 Bean 生成ツールーステップ 1/3」画面で，「EJB 対応機能を有効にする。」をチェックしていない場合は，「クラス名.java」が生成されるファイル名になります。

  チェックした場合は，次の名称のファイルを生成します。

  - クラス名.java（COBOL アクセス用 Bean プログラム）
  - クラス名 EJBHome.java（ホームインタフェース）
  - クラス名 EJB.java（リモートインタフェース）
  - クラス名 EJBBean.java（Enterprise Bean）
  - ejb-jar.xml（デプロイ情報）

  なお，EJB 対応 COBOL アクセス用 Bean の場合，実行に際し jar ファイルの作成／登録を行う必要があります。詳細は，「2.3.3　EJB 用 jar ファイルの作成と登録」を参照してください。

- ライブラリ名

  (Windows の場合)

  　COBOL で作成した DLL 名を絶対パス名で指定します。

DLL 名だけを指定することもできますが，DLL 名だけを指定した場合には，実行時に dynamicpath オプションの指定が必要です。

セキュリティの理由から，絶対パス名で指定することを推奨します。

（指定例）

```
C:¥user_path¥SEARCH.dll  ····  拡張子.dllは必須
```

（UNIX の場合）

Bean を生成する場合には，共用ライブラリ名に拡張子を付けて絶対パス名で指定します。

（AIX の例）

```
/user_path/libSEARCH.a  ····  拡張子.aは必須
```

（Linux の例）

```
/user_path/libSEARCH.so  ····  拡張子.soは必須
```

dynamicpath オプションを使用する場合は，フォルダ名またはディレクトリ名，および拡張子を省略したライブラリ名を指定できます。

（Windows の例）

```
SEARCH  ····  拡張子.dllは指定しない
```

（AIX の例）

```
libSEARCH  ····  拡張子.aは指定しない
```

（Linux の例）

```
libSEARCH  ····  拡張子.soは指定しない
```

- 呼び出す COBOL プログラム名

  COBOL プログラム名は，英大文字と英小文字を区別します。

以上のように画面に従って操作し，最後に［終了(F)］ボタンを押すと，Java ソースファイルが生成され，該当する COBOL アクセス用 Bean が自動生成されます（COBOL アクセス用 Bean の自動生成ソースイメージは，「付録 C　COBOL アクセス用 Bean の自動生成ソースイメージ」を参照してください）。

自動生成したソースは編集しないでください。また，生成時は，作業用フォルダとして環境変数 TEMP に指定されているフォルダ下に j2cb フォルダを作成します。

## (4)　jar ファイルの作成／登録

EJB 対応 COBOL アクセス用 Bean の場合，実行に際し jar ファイルの作成／登録を行う必要があります。詳細については，「2.3.3　EJB 用 jar ファイルの作成と登録」を参照してください。

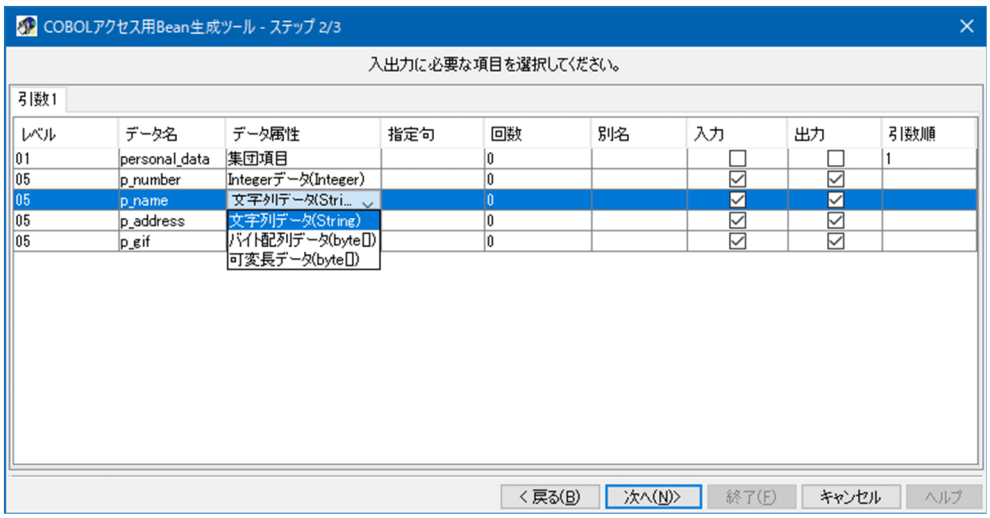
## (5)　制限事項

- 「Bean 生成ツール」を二つ以上起動できません。

## 2.3.2　COBOL アクセス用 Bean 生成ツールでのバイト配列指定

### (1)　「バイト配列データ(byte[])」または「可変長データ(byte[])」を指定する場合

英数字項目で「バイト配列データ(byte[])」または「可変長データ(byte[])」は，「Bean 生成ツール」に表示されるデータ属性の項目を変更して指定します。データ属性が「文字列データ(String)」でない場合，データ属性の項目はリスト表示されません。



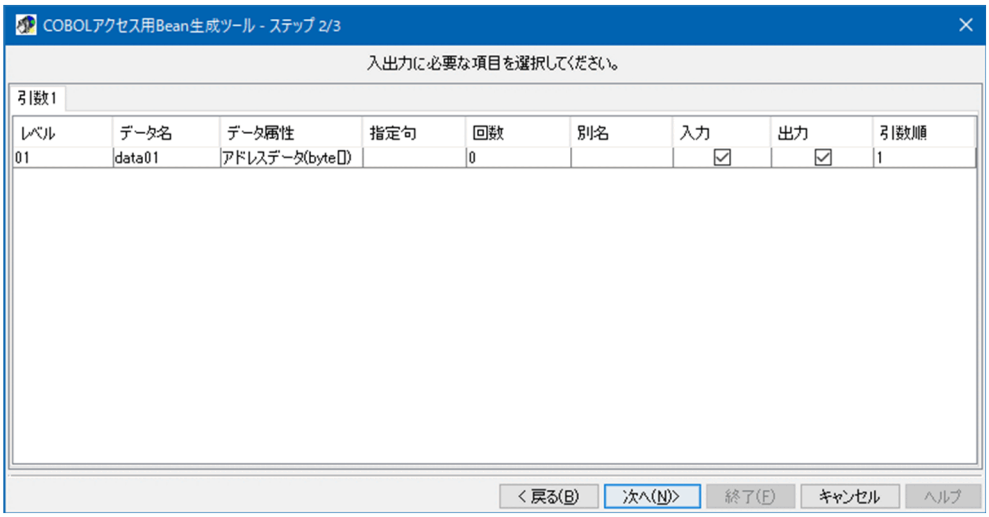

「可変長データ(byte[])」が指定できないデータの場合は，[次へ(N)]ボタンを押す際にエラーダイアログを表示します。「バイト配列データ(byte[])」，「可変長データ(byte[])」は，次の条件を満たす場合に指定できます。

1.「バイト配列データ(byte[])」，「可変長データ(byte[])」が指定できるデータ項目
   - 英数字項目
   - 英字項目
   - 英数字編集項目
   - 数字編集項目
   - 日本語項目
   - 日本語編集項目

   ただし，日本語項目および日本語編集項目は「COBOL アクセス用 Bean 生成ツール－ステップ 1/3」画面で「日本語項目を英数字項目として扱う。」チェックボックスがオンの場合だけ指定できます。
2.「可変長データ(byte[])」が指定できるデータ項目
   - レベルフィールドは 01 または 77 レベル
   - 指定句フィールドは REDEFINES 句未指定
   - COBOL 定義領域長は 5 バイト以上

## (2) 「アドレスデータ項目(byte[])」を指定する場合

アドレスデータ項目を指定します。データ属性に「アドレスデータ(byte[])」と表示されます。




「アドレスデータ(byte[])」が指定できないデータの場合は，[次へ(N)]ボタンを押す際にエラーダイアログを表示します。「アドレスデータ(byte[])」は，次の条件を満たす場合に指定できます。

「アドレスデータ(byte[])」が指定できるデータ項目

- レベルフィールドは 01 または 77 レベル
- 指定句フィールドは REDEFINES 句未指定

## (3) 「可変長データ(byte[])」および「アドレスデータ(byte[])」のデータ形式について

「可変長データ(byte[])」および「アドレスデータ(byte[])」を再定義することはできません。指定した場合，再定義項目は従属する下位項目も含めて無視されます。そのため，setter／getter も生成されません。

「可変長データ(byte[])」および「アドレスデータ(byte[])」は，ほかのデータ属性のように指定された領域長でデータアクセスをするのではなく，形式が異なります。形式は，下記のように分割されたデータ領域となります。

| ４バイトの２進項目 | （領域長－４）バイトのバイト配列領域 |
|---|---|
| 長さを表す | バイト配列データを表す |

COBOL プログラムでは，上記形式でデータを扱ってください。

例 1）

登録集原文に「01 V_DATA PIC X(100).」と記述されたデータ項目のデータ属性を，「可変長データ(byte[])」と指定した場合，長さを表す 4 バイトの 2 進項目と，96 バイトのバイト配列データに分割されます。

したがって，バイト配列データがとりうる（最大データ長＋ 4）バイトの領域長で，データ項目を定義する必要があります。「可変長データ(byte[])」でデータを設定(setter)および取得(getter)する方法については，「8.　COBOL アクセス用 Bean を呼び出すための API」を参照してください。

例 2)

登録集原文に「01 V_DATA USAGE ADDRESS.」と記述指定した場合，長さを表す 4 バイトの 2 進項目と，setter で指定された長さ分のバイト配列データの領域になります。

「アドレスデータ(byte[])」でデータを設定(setter)および取得(getter)する方法については，「8.　COBOL アクセス用 Bean を呼び出すための API」を参照してください。

## 2.3.3　EJB 用 jar ファイルの作成と登録

「Bean 生成ツール」で「EJB 対応機能を有効にする。」にチェックして Bean を生成した場合は，コンパイルした class ファイルなどを EJB サーバへデプロイするために jar ファイルにする必要があります。

生成されるファイルは次のとおりです。

| 項番 | 生成プログラム名 | 生成ファイル名規則 | 備考 |
|---|---|---|---|
| 1 | COBOL アクセス用 Bean | クラス名.java | なし。 |
| 2 | ホームインタフェース | クラス名 EJBHome.java | なし。 |
| 3 | リモートインタフェース | クラス名 EJB.java | なし。 |
| 4 | Enterprise Bean | クラス名 EJBBean.java | なし。 |
| 5 | デプロイ情報<br>(DD ファイル) | ejb-jar.xml | META-INF フォルダが自動生成され，そのフォルダに格納されます。 |

### (1)　jar ファイルの作成

「Bean 生成ツール」での jar ファイル作成手順を次に示します。

1. 「Bean 生成ツール」で生成されたファイルをコンパイルして class ファイルを生成します。
2. コマンドプロンプトを開き，「Bean 生成ツール」の「生成ファイルを出力するフォルダ」に移動し，次のような jar コマンドで jar ファイルを作成します。

```
jar  cvf  ¥temp¥xxxejb.jar META-INF
```

3. コマンドプロンプトを開き，class ファイル出力フォルダに移動し，次のような jar コマンドで jar ファイルを作成します。

   jar uvf ¥temp¥xxxejb.jar 「Bean 生成ツール」で指定したパッケージ文字列の先頭パッケージ

## (2)　jar ファイルの登録方法

生成された jar ファイルを Cosminexus に登録します。登録の方法は次のとおりです。

1. Cosminexus Component Container（J2EE サーバ）を起動します。
2. J2EE サーバ用のコマンドを使用して EJB の jar ファイルを登録します。
3. cjimportres コマンドで EJB-JAR ファイルをインポートします。
4. cjaddapp コマンドでインポート済みの EJB-JAR ファイルを J2EE アプリケーションに追加します。
5. cjstartapp コマンドで J2EE アプリケーションを開始して，クライアントからのリクエストを受け取ることができるようにします。

   Cosminexus Component Container の起動や J2EE サーバで使用するコマンドの詳細については，使用している Cosminexus のバージョンに対応した，マニュアル「Cosminexus　リファレンス　コマンド編」を参照してください。

## 2.4　Servlet の作成

COBOL アクセスでの Servlet の作成方法について説明します。

### 2.4.1　COBOL アクセスでの Servlet の作成方法

生成した COBOL アクセス用 Bean を呼び出す Servlet（Java UAP）を作成します。次のことに注意して作成してください。

#### (1)　setter の指定

一般の JavaBeans を利用した Servlet プログラムと同様に，自動生成された COBOL アクセス用 Bean を参考にして，COBOL 引数の設定を行います。設定時には setter（setXxx）を使用して引数領域をすべて設定しておかなければなりません。設定されていない領域の値は保証しません。

##### (a)　通常の例

（COBOL 引数の登録集原文例）

```
01  PERSONAL-DATA.
  05  P-NUMBER   PIC  9(9)  USAGE COMP.
  05  P-NAME     PIC  X(50).
  05  P-ADDRESS  PIC  X(100).
```

（JavaBeans の場合の引数設定例）

```
bean.setP_number(new Integer(number));
bean.setP_name("");
bean.setP_address("");
```

（EJB の場合の引数設定例）

```
remoteobj.setP_number(new Integer(number));
remoteobj.setP_name("");
remoteobj.setP_address("");
```

##### (b)　OCCURS 句を使った例

（COBOL 引数の登録集原文例）

```
01  G1.
  02  G2 OCCURS 10.
    05  B1   PIC  X(50).
```

（JavaBeans の場合の引数設定例）

```
int i=0;
for (i=0; i<10 ; i++) {
```

```
    bean.setB1(”XXXXX”, i);
}
```

(EJB の場合の引数設定例)

```
int i=0;
for (i=0; i<10 ; i++) {
    remoteobj.setB1(”XXXXX”, i);
}
```

### (c) 可変長データを指定した例

(COBOL 引数の登録集原文例)

```
01 XML-DATA    PIC X(10000004).
```

(JavaBeans の場合の引数設定例)

```
byte[] sdata=new byte[10000000];
bean.setXml_data(sdata, 50);
```

(EJB の場合の引数設定例)

```
byte[] sdata=new byte[10000000];
remoteobj.setXml_data(sdata, 50);
```

### (d) アドレスデータを指定した例

setter の引数には，データ(バイト配列)とデータ長を指定します。

(COBOL 引数の登録集原文例)

```
01 XML-DATA    USAGE ADDRESS.
```

(JavaBeans の場合の引数設定例)

```
byte[] sdata=new byte[20000000];
bean.setXml_data(sdata, 50);
```

(EJB の場合の引数設定例)

```
byte[] sdata=new byte[20000000];
remoteobj.setXml_data(sdata, 50);
```

## (2) callCOBOL メソッドの呼び出し

Bean の setter を記述後に COBOL アクセス用 Bean の callCOBOL メソッドを呼び出します。

(JavaBeans の場合)

```
bean.callCOBOL();
```

（EJB の場合）

```
remoteobj.callCOBOL();
```

## (3) getter の指定

Servlet で使用する場合は，Bean の getter（getXxx）を使用することで COBOL の引数を取得できます。

### (a) 通常の例

（COBOL 引数の登録集原文例）

```
01  PERSONAL-DATA.
  05  P-NUMBER   PIC  9(9)  USAGE COMP.
  05  P-NAME     PIC  X(50).
  05  P-ADDRESS  PIC  X(100).
```

（JavaBeans の場合の引数取得指定例）

```
Integer wkint =bean.getP_number();
String  wkstr1=bean.getP_name();
String  wkstr2=bean.getP_address();
```

（EJB の場合の引数取得指定例）

```
Integer wkint =remoteobj.getP_number();
String  wkstr1=remoteobj.getP_name();
String  wkstr2=remoteobj.getP_address();
```

### (b) OCCURS 句を使った例

（COBOL 引数の登録集原文例）

```
01  G1.
  02  G2 OCCURS 10.
  05  B1   PIC  X(50).
```

（JavaBeans の場合の引数取得指定例）

```
String wkstr=bean.getB1(5);
```

（EJB の場合の引数取得指定例）

```
String wkstr=bean.getB1(5);
```

### (c) 可変長データを指定した例

XML-DATA のデータ属性を「可変長データ(byte[])」にします。データ長は， データ(バイト配列)の length プロパティで取得します。

(COBOL 引数の登録集原文例)

```
01 XML-DATA    PIC X(10000004).
```

(JavaBeans の場合の引数取得指定例)

```
byte[] rdata=bean.getXml_data();
int rlen=rdata.length;    // 長さを取得する
```

(EJB の場合の引数取得指定例)

```
byte[] rdata=remoteobj.getXml_data();
int rlen=rdata.length;    // 長さを取得する
```

### (d) アドレスデータを指定した例

データ長は，データ(バイト配列)の length プロパティで取得します。

(COBOL 引数の登録集原文例)

```
01 XML-DATA    USAGE ADDRESS.
```

(JavaBeans の場合の引数取得指定例)

```
byte[] rdata=bean.getXml_data();
int rlen=rdata.length;   // 長さを取得する
```

(EJB の場合の引数取得指定例)

```
byte[] rdata=remoteobj.getXml_data();
int rlen=rdata.length;    // 長さを取得する
```

また，JSP で Bean を使用することもできます。

(例) Servlet から JSP 呼び出し指定例

```
javax.servlet.RequestDispatcher
rd=c.getRequestDispatcher("/Search.jsp");
rd.forward(req, res);
```

## 2.4.2 COBOL UAP の呼び出し

COBOL UAP は COBOL アクセス用 Bean を介して呼び出されます。COBOL アクセス用 Bean は，COBOL アクセス用 Bean 生成ツールを使用して生成します。

COBOL UAP は Java UAP から次の手順を実行することで呼び出すことができます。

1. UAP の開始手続き

   COBOL UAP に対応する COBOL アクセス用 Bean のインスタンスを生成します。

2. データの設定

COBOL UAP の引数を COBOL アクセス用 Bean の対応する setter メソッドを発行して設定します。メソッド名は，「set + COBOL の基本項目名」となります。

例）COBOL の基本項目名が「ITEM01」の場合，「setItem01」となります。

3. COBOL UAP の呼び出し

   COBOL アクセス用 Bean の callCOBOL メソッドを発行して，COBOL UAP を呼び出します。

4. データの取得

   COBOL UAP の引数を COBOL アクセス用 Bean の対応する getter メソッドを発行して取得します。メソッド名は，「get + COBOL の基本項目名」となります。

   例）COBOL の基本項目名が「ITEM01」の場合，「getItem01」となります。

### 2.4.3 例外処理

COBOL アクセス用 Bean では，COBOL UAP でエラーが発生した場合または COBOL アクセス用 Bean で異常が発生した場合，情報が取得可能な API を持つ例外を発生させます。

Servlet(Java UAP)で情報が取得可能な API を使用して例外処理を行う必要があります。

API のリファレンスについては，「8.3 J2CBException ユーザインタフェース API」または「8.4 EJB 用 Exception ユーザインタフェース API」を参照してください。

## 2.5　入力用 HTML と結果出力用 JSP の作成

作成した Servlet を起動するための入力用 HTML と結果出力用 JSP を作成します。

作成例は「付録 F　プログラム例」の HTML の作成例および JSP の作成例を参照してください。JavaBean 対応または EJB 対応でそれぞれ作成例が記載されています。

## 2.6　COBOL プログラム作成上の注意事項

COBOL UAP の作成上の注意事項について説明します。

### 2.6.1　文字コードについて

COBOL2002 で決められた文字コードが使用できます。Windows の場合は，シフト JIS コードが使用できます。UNIX の場合は，マニュアル「COBOL2002　使用の手引　手引編」を参照してください。また，環境変数 CBLJ2CBOPT の encode 指定によって，エンコードに従った文字コード体系に変換することもできます。文字コードの設定方法については「5.4　COBOL アクセスの環境変数の設定」，encode 指定については「5.4.4　encode オプション」を参照してください。

また，unicode 指定によって Unicode を使用することもできます。unicode 指定については「5.4.8 unicode オプション」を参照してください。

### 2.6.2　COBOL2002 コンパイラオプションについて

Java アプリケーションプログラムとインタフェースを持つ COBOL UAP は，DLL または共用ライブラリ（AIX の場合：*.a，Linux の場合：*.so）として作成してください。Java アプリケーションプログラムとインタフェースを持つ COBOL UAP では，設定が必要なオプション，設定してはいけないオプションは次のとおりです。

#### (1)　Windows の場合

##### (a)　設定が必要なオプション

- -MainNotCBL：副プログラムとしてコンパイルします。
- -Dll,Stdcall：DLL の形式を指定します（Dll の属性を Stdcall にします）。Windows(x86)だけで有効です。
- -Dll：DLL の形式を指定します（Dll の属性を fastcall にします）。Windows(x64)だけで有効です。
- -DllInit：呼び出し時に DLL を初期状態にします。
- -MultiThread：マルチスレッド機能を使用します。
  ただし，マルチスレッド機能を使用する場合，順ファイル，テキストファイルに対するファイル入出力は使用できないなど，幾つかの注意事項があります。詳細は，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。
- -Comp5：COMP-5 を指定できるようにします。

##### (b)　設定してはいけないオプション

- -Bin1Byte：1 バイトの 2 進項目を有効にします。

- -Dll,Cdecl： DLL の形式を指定します（Dll の属性を Cdecl にします）。Windows(x86)だけで有効です。

  そのほかのコンパイラオプションについては，必要に応じて設定してください。

### (2) UNIX の場合

#### (a) 設定が必要なオプション

- -PIC,{Std | Expand}：共用ライブラリに使う位置独立（PIC）コードを作成する場合に指定します。
- -MultiThread：マルチスレッド機能を使用します。ただし，マルチスレッド機能を使用する場合，順ファイル，テキストファイルに対するファイル入出力は使用できないなど，幾つかの注意事項があります。詳細はマニュアル「COBOL2002　使用の手引　手引編」を参照してください。
- -MainNotCBL：副プログラムとしてコンパイルします。
- -Comp5：COMP-5 を指定できるようにします。

#### (b) 設定してはいけないオプション

- -Bin1Byte：1 バイトの 2 進項目を有効にします。

  そのほかのコンパイラオプションについては，必要に応じて設定してください。

## 2.6.3 指定してはいけない文と指定

- PROCEDURE DIVISION に RETURNING は指定できません。
- STOP RUN 文は指定できません。

そのほかの注意事項および制限事項は「付録 A　注意事項／制限事項」を参照してください。

## 2.6.4 引数の個数

- Linux(x64)の場合，COBOL プログラムに指定できる引数の個数は，4,096 個までです。
- Windows(x86)の場合，COBOL プログラムに指定できる引数の個数に制限はありません。
- Windows(x64)，AIX(64)の場合，COBOL プログラムに指定できる引数の個数は，1,024 個までです。

そのほかの注意事項および制限事項は「付録 A　注意事項／制限事項」を参照してください。

# 3 プログラムのコンパイル

この章では，作成した COBOL UAP および Java プログラムのコンパイルに関する操作を説明します。

## 3.1　COBOL UAP のコンパイル

COBOL UAP のコンパイル方法は，マニュアル「COBOL2002　操作ガイド」またはマニュアル「COBOL2002　使用の手引　手引編」を参照してください。

### 3.1.1　Windows(x86)の場合

COBOL UAP の開発は開発マネージャを用いて作成およびコンパイルすることができます。また，次の例のようにコマンドを入力してコンパイルとリンクを行うこともできます。

（例）

```
ccbl2002 -MainNotCBL -Dll,Stdcall -DllInit -MultiThread
         -Comp5 sample.cbl
```

### 3.1.2　Windows(x64)の場合

COBOL UAP の開発は開発マネージャを用いて作成およびコンパイルすることができます。また，次の例のようにコマンドを入力してコンパイルとリンクを行うこともできます。

（例）

```
ccbl2002 -MainNotCBL -Dll -DllInit -MultiThread -Comp5 sample.cbl
```

### 3.1.3　AIX(64)の場合

COBOL UAP のコンパイルおよびリンケージ例を次に示します。

- コンパイルの例を次に示します。

```
ccbl2002 -MainNotCBL -PIC,Std -MultiThread -Comp5 sample.cbl
ld -o libsample.a sample.o -b64 -bpT:0x100000000 -bpD:0x110000000
   -bnoentry -bM:SRE -bexpall -L/opt/HILNGcbl2k64/lib -lcbl2k64
   -lcbl2kml64 -lcbl2kmp64 -lpthreads -lm
```

- リンケージ時は，「-L/opt/HILNGcbl2k64/lib -lcbl2k64 -lcbl2kml64 -lcbl2kmp64 -lpthreads -lm」が必要です。

### 3.1.4　Linux(x64)の場合

COBOL UAP のコンパイルおよびリンケージ例を次に示します。

- コンパイルの例を次に示します。

```
CBLSRCENCODING=SJIS
export CBLSRCENCODING
ccbl2002 -UniObjGen -MainNotCBL -PIC,Std -MultiThread -Comp5 sample.cbl
ld -shared -o libsample.so sample.o -L/opt/HILNGcbl2k64/lib -lcbl2k
   -lcbl2kmp -lm -Bstatic -lcbl2kml
```

- リンケージ時は，「-L/opt/HILNGcbl2k64/lib -lcbl2k -lcbl2kmp -lm -Bstatic -lcbl2kml」が必要です。
- COBOL ソースは，シフト JIS コードだけを対象としています。

## 3.2　Java プログラムのコンパイル

Java プログラムのコンパイル方法について説明します。

### 3.2.1　Windows の場合

作成した Java プログラム（Servlet や Bean）は，Cosminexus の開発環境を使用してコンパイルし class ファイルを生成します。ただし，次のことに注意してください。

#### (1)　JavaBean の場合

クラスパスに”COBOL2002 インストールフォルダ¥LIB¥j2cbrun.jar”を指定する必要があります。

#### (2)　EJB の場合

クラスパスに次の内容を指定する必要があります。

- Cosminexus 09-80 より前の場合
  - COBOL2002 インストールフォルダ¥LIB¥j2cbrun.jar
- Cosminexus 09-80 以降の場合
  - COBOL2002 インストールフォルダ¥LIB¥j2cbrun.jar
  - <Cosminexus インストールフォルダ>¥jdk¥lib¥hcompatlib¥hjdk.tpb.jar

  この場合，javac コマンドで「--add-modules=java.corba」オプションを指定しないでください。

### 3.2.2　UNIX の場合

作成した Java プログラム（Servlet や Bean）は，Windows 環境の Cosminexus の開発環境を使用してコンパイルし，class ファイルを UNIX 環境に移して（バイナリコードで転送する）使用してください。

# 4 アプリケーションサーバの環境整備

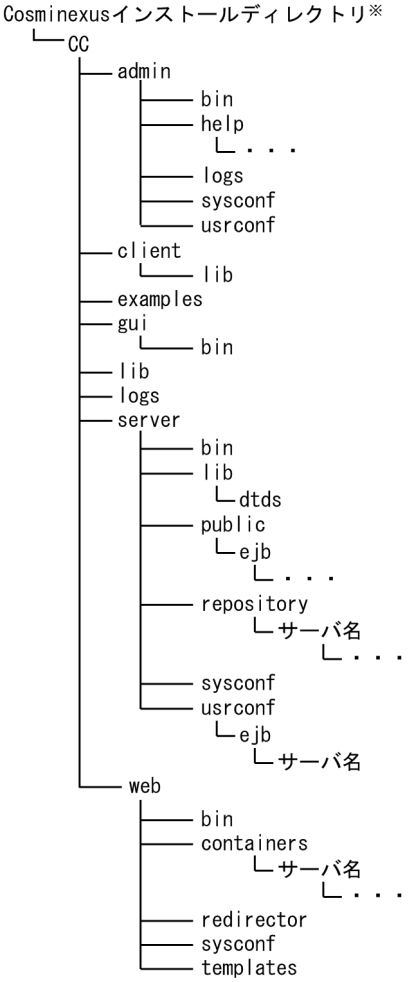
作成した COBOL UAP と Java プログラムを実行させるためには，まず実行環境を整えなくてはなりません。この章では，実行環境について Web コンテナサーバを使用した場合，および J2EE サーバを使用した場合について説明します。

Web コンテナサーバ（サーブレットエンジンモード）は，Cosminexus 製品が提供する互換用のモードです。プログラムの種類によっては Web コンテナサーバ上で動作できますが，J2EE サーバモードで動作させることをお勧めします。

## 4.1 Cosminexus の環境について

COBOL プログラムを実行する前に，Cosminexus の動作に必要な定義と環境変数を設定してください。Cosminexus の環境定義については，使用している Cosminexus のバージョンに対応した，マニュアル「Cosminexus　システム構築・運用ガイド」，またはマニュアル「Cosminexus　システム構築ガイド」を参照してください。

これ以降の説明で使用する Cosminexus Component Container のインストールディレクトリ階層を次に示します。




注※

Windows の場合：

これ以降 <Cosminexus インストールフォルダ>¥CC のことを省略して**<CC インストール先>**とします。例えば，server ディレクトリを表す場合，<CC インストール先>¥server と表します。

UNIX の場合：

これ以降 <Cosminexus インストールディレクトリ>/CC のことを省略して**<CC インストール先>**とします。例えば，server ディレクトリを表す場合，<CC インストール先>/server と表します。

## 4.2　サービスの起動確認

Cosminexus の実行環境を使用する場合に必要なサービスが起動されているか確認してください。

- Web サーバ
- Web コンテナサーバまたは J2EE サーバ

Web サーバの設定については使用する Web サーバのマニュアルを参照してください。

## 4.3　Web コンテナサーバの設定

Web コンテナサーバは JavaBeans および EJB で使用します。JavaBeans では CLASSPATH の設定および Library Path の設定，EJB では CLASSPATH の設定および EJB 呼び出し用の環境設定を行ってください。なお，Web コンテナサーバで EJB を呼び出す場合は，呼び出す EJB の実行環境用として「4.4　J2EE サーバの設定」も必要です。

設定内容について説明します。

### 4.3.1　CLASSPATH の設定

#### (1)　Windows の場合

次の手順で CLASSPATH に"COBOL2002 インストールフォルダ¥LIB¥j2cbrun.jar"を設定します。

1. Web コンテナサーバのユーザ定義ファイル（usrconf.cfg）をテキストエディタで開きます。

   **＜格納フォルダ＞**

   ＜ CC インストール先>¥web¥containers¥<サーバ名>¥usrconf

2. usrconf.cfg ファイルに"web.add.class.path=追加する jar ファイル"を指定します。

   **＜指定例＞**

   web.add.class.path=C:¥PROGRA~1¥Hitachi¥Cobol2~1¥LIB¥j2cbrun.jar

3. ファイルを保存し，Web コンテナサーバを再起動します。

#### (2)　AIX(64)または Linux(x64)の場合

次の手順で CLASSPATH に"/opt/HILNGcbl2k64/lib/j2cbrun.jar"を設定します。

1. Web コンテナサーバのユーザ定義ファイル（usrconf.cfg）をテキストエディタで開きます。

   **＜格納ディレクトリ＞**

   ＜ CC インストール先>/web/containers/<サーバ名>/usrconf

2. usrconf.cfg ファイルに"web.add.class.path=追加する jar ファイル"を指定します。

   **＜指定例＞**

   web.add.class.path=/opt/HILNGcbl2k64/lib/j2cbrun.jar

3. ファイルを保存し，Web コンテナサーバを再起動します。

### 4.3.2　Library Path の設定

次の手順で Library Path を設定します。

## (1)　Windows の場合

1. Windows の［コントロールパネル］を開き，［システム］をダブルクリックします。
2. システムのプロパティウィンドウの詳細タグ中の環境変数をクリックします。
3. システム環境変数 PATH に"COBOL2002 インストールフォルダ¥BIN"を追加します。
4. 設定の変更を適用するために，OK をクリックします。
5. 設定後 Web コンテナサーバを再起動します。
   （Web コンテナサーバを起動するコマンドプロンプトも必ず再起動してください）。

## (2)　AIX(64)の場合

1. システム環境変数 LIBPATH に"/opt/HILNGcbl2k64/lib"を追加します。
2. 設定後，Web コンテナサーバを再起動します。

## (3)　Linux(x64)の場合

1. システム環境変数 LD_LIBRARY_PATH に"/opt/HILNGcbl2k64/lib"を追加します。
2. 設定後，Web コンテナサーバを再起動します。

# 4.3.3　EJB 呼び出し用の環境設定(EJB を呼び出す場合だけ)

## (1)　RMI-IIOP のスタブの取得

cjgetstubsjar コマンドでアプリケーションの RMI-IIOP スタブおよびインタフェースの取得を行います。

＜例＞

- EJB Jar File-> 1.jar
- Stubs Jar File-> stubs.jar

## (2)　EJB クライアント Jar ファイルの取得

HiEJBClientStatic.jar ファイルを EJB クライアントアプリケーション側にコピーします。

＜格納フォルダ＞

- Windows の場合
  <CC インストール先>¥client¥lib¥HiEJBClientStatic.jar
- UNIX の場合
  <CC インストール先>/client/lib/HiEJBClientStatic.jar

## (3)　usrconf.cfg ファイルの変更

Web コンテナサーバのユーザ定義ファイル（usrconf.cfg）をテキストエディタで開きます。

＜格納フォルダ＞

- Windows の場合
  <CC インストール先>¥web¥containers¥<サーバ名>¥usrconf
- UNIX の場合
  <CC インストール先>/web/containers/<サーバ名>/usrconf

usrconf.cfg ファイルに"web.add.class.path=追加する jar ファイル"を指定します。

ここでは，(1)，(2)で取得したファイルを指定します。

- RMI-IIOP のスタブファイル（EJB Jar File，Stubs Jar File）
- EJB クライアント Jar ファイル

## 4.4　J2EE サーバの設定

J2EE サーバの CLASSPATH および Library Path の設定方法について説明します。

### 4.4.1　CLASSPATH および Library Path の設定

#### (1)　Windows の場合

J2EE サーバのユーザ定義 usrconf.cfg（<CC インストール先>¥server¥usrconf¥ejb¥<サーバ名称>¥usrconf.cfg）で次の設定を行う必要があります。

- add.class.path キーに”COBOL2002 インストールフォルダ¥LIB¥j2cbrun.jar”を設定します。
- add.library.path キーに”COBOL2002 インストールフォルダ¥BIN”を設定します。

＜設定例＞

```
add.class.path=E:¥PROGRA~1¥Hitachi¥Cobol2~1¥LIB¥j2cbrun.jar
add.library.path=E:¥PROGRA~1¥Hitachi¥Cobol2~1¥BIN
```

注意

パスに 8 文字を超えるファイル名または空白を含むファイル名を指定する場合，短いファイル名（PROGRA~1 など）を指定してください。短いファイル名は，コマンドプロンプトで"dir /X"コマンドを指定して確認してください。

#### (2)　AIX(64)または Linux(x64)の場合

J2EE サーバのユーザ定義 usrconf.cfg（<CC インストール先>/server/usrconf/ejb/<サーバ名称>/usrconf.cfg）で次の設定を行ってください。

- add.class.path キーに「/opt/HILNGcbl2k64/lib/j2cbrun.jar」を設定します。
- add.library.path キーに「/opt/HILNGcbl2k64/lib」を設定します。

＜設定例＞

```
add.class.path=/opt/HILNGcbl2k64/lib/j2cbrun.jar
add.library.path=/opt/HILNGcbl2k64/lib
```

# 5 COBOL アクセスの実行環境の整備

この章では，COBOL アクセス環境下で COBOL プログラムの実行環境を設定するための環境変数の指定方法について説明します。

## 5.1　COBOL アクセスでの COBOL 環境変数の設定

COBOL アクセス環境下の COBOL プログラムは，マルチスレッド環境で動作しているため，COBOL 環境変数が有効となる範囲は，指定方法によって次の 2 種類があります。

- COBOL アクセス環境のプロセス空間に有効となる環境変数
- COBOL アクセス環境で動作するスレッド単位に有効となる環境変数

なお，COBOL アクセス環境下では実行支援で生成する実行環境ファイルは使用できません。環境変数の有効範囲別に，COBOL 環境変数の指定方法を次に示します。

### 5.1.1　COBOL アクセスプロセス空間に有効な環境変数

COBOL アクセスプロセス空間に有効となる COBOL 環境変数は，次に示す二つの方法で設定できます。

- Cosminexus 起動コマンド発行前にシェルなどで，COBOL2002 が提供する実行時環境変数の設定で行います。※
- 実行時環境変数群が指定された環境設定ファイル名を「cobol.sysenvfile」に指定します。

環境設定ファイルに，環境変数 CBL_SYSOUT，CBL_SYSPUNCH，CBL_SYSERR，CBLABNLST，CBLDDUMP を指定することで，指定した物理ファイル名に対して自動的にスレッド識別子が付加されたスレッドごとの固有のファイル出力ができます。また，Cosminexus 起動コマンド発行前にシェルなどで環境変数 CBL_SYSOUT，CBL_SYSPUNCH，CBL_SYSERR，CBLABNLST，CBLDDUMP の設定も行っている場合は，環境設定ファイルで設定した COBOL 環境変数設定が有効となります。

環境変数 CBL_SYSOUT，CBL_SYSPUNCH，CBL_SYSERR，CBLABNLST，CBLDDUMP の詳細については，マニュアル「COBOL2002　ユーザーズガイド」またはマニュアル「COBOL2002　使用の手引　手引編」を参照してください。

注※

COBOL2002 が提供する実行時環境変数のうち，環境変数 CBLEXCEPT は有効となりません。

#### (1)　環境設定ファイルに指定する内容

環境設定ファイル（sysenv.txt）に実行環境設定変数を次のように指定します。

- Windows の場合

```
CBL_SYSOUT=C:¥TEMP¥SYSOUTFILE
CBL_SYSERR=C:¥TEMP¥SYSERRFILE
CBLABNLST=C:¥TEMP¥ABENDLST
CBLDDUMP=C:¥TEMP¥DUMPLST
```

- UNIX の場合

```
CBL_SYSOUT=/TEMP/SYSOUTFILE
CBL_SYSERR=/TEMP/SYSERRFILE
CBLABNLST=/TEMP/ABENDLST
CBLDDUMP=/TEMP/DUMPLST
```

## 5.1.2　COBOL アクセスでの COBOL 環境変数の設定

Web コンテナサーバで，COBOL 環境変数を設定します。

usrconf.cfg ファイルに"add.jvm.arg="に-D オプションを先頭に付けて，cobol.sysenvfile を指定します。

### (1)　設定例

環境設定ファイルを sysenv.txt とした場合の例を示します。

- Windows の場合

```
add.jvm.arg=-Dcobol.sysenvfile=E:¥work¥sysenv.txt
```

- UNIX の場合

```
add.jvm.arg=-Dcobol.sysenvfile=/work/sysenv.txt
```

### (2)　格納フォルダ

usrconf.cfg ファイルが格納されているフォルダの場所を次に示します。

- Windows の場合

  (JavaBeans 版)

```
＜CCインストール先＞¥web¥containers¥サーバ名¥usrconf¥usrconf.cfg
```

  (EJB 版)

```
＜CCインストール先＞¥server¥usrconf¥ejb¥サーバ名¥usrconf.cfg
```

- UNIX の場合

  (JavaBeans 版)

```
＜CCインストール先＞/web/containers/サーバ名/usrconf/usrconf.cfg
```

  (EJB 版)

```
＜CCインストール先＞/server/usrconf/ejb/サーバ名/usrconf.cfg
```

### (3)　環境設定ファイル作成

環境設定ファイル（sysenv.txt）に実行環境設定変数を指定します。

### (4) Web コンテナサーバの再起動

環境設定ファイルを保存し，Web コンテナサーバを再起動します。

## 5.1.3 注意事項

環境設定ファイルでの実行時環境設定変数の設定は，J2EE サーバと同じプロセス空間で有効です。また全角文字を含む環境設定ファイル名は使用できません。

環境設定ファイルはテキスト形式のファイルで，1 行に一つの実行時環境変数を 1,024 バイト以内で指定します。また，実行時環境変数を指定するそれぞれの行は，改行文字で終了していなければなりません。行始まりから"="の直前までを環境変数名，"="の直後から改行コードまたは EOF の直前までを環境変数の値とします。

環境変数の値に値を指定しない場合は，システム環境変数を無効にできます。指定した環境設定ファイルが存在しない場合，環境変数は設定されないで，そのまま処理を続行します。

## 5.2　スレッド単位の環境変数を設定する

COBOL プログラムの実行環境設定のため，さまざまな実行時オプションをプログラム別に設定する必要がある場合，ACCEPT，DISPLAY 文による環境変数へのアクセス機能を利用して設定します。

この機能の詳細については，マニュアル「COBOL2002　ユーザーズガイド」またはマニュアル「COBOL2002　使用の手引　手引編」を参照してください。

## 5.3　COBOL UAP の検索順序

### 5.3.1　Windows の場合

COBOL アクセス用 Bean 生成ツールで絶対パス名を指定した場合は，指定した絶対パス名を検索します。

dynamicpath オプションを指定した場合は，システム環境変数 PATH の登録順となります。

### 5.3.2　UNIX の場合

COBOL UAP の検索は，「COBOL アクセス用 Bean 生成ツールーステップ 3/3」画面でライブラリ名に指定した絶対パス名を検索します。「COBOL アクセス用 Bean 生成ツールーステップ 3/3」画面については，「2.3.1　COBOL アクセス用 Bean 生成ツールによる生成」の「(3)　COBOL アクセス用 Bean 生成ツールの画面」の「(c)　ステップ 3/3 画面」を参照してください。

また，dynamicpath オプションを指定すると，ディレクトリ名を省略した共用ライブラリを，次の環境変数に指定された順に検索できます。

- AIX の場合：システム環境変数 LIBPATH
- Linux の場合：システム環境変数 LD_LIBRARY_PATH

## 5.4　COBOL アクセスの環境変数の設定

COBOL アクセスの環境変数の設定について説明します。

### 5.4.1　環境変数 CBLJ2CBOPT

COBOL アクセス実行環境変数 CBLJ2CBOPT には，COBOL アクセス実行時ライブラリに渡すオプションを設定します。

設定方法は次のとおりです。設定に誤りがある場合は無効となります。

#### (1)　Windows の場合

環境変数は，コマンドプロンプトまたはコントロールパネルの［システム］で設定します。

指定形式

```
set CBLJ2CBOPT=[OPTION] [: OPTION]...
```

[OPTION]

実行時ライブラリに渡すオプションを指定します。現在用意されているオプションは comp5 オプション，dynamicpath オプション，encode オプション，endian オプション，japanese オプション，loadingrule オプション，および unicode オプションです。

#### (2)　AIX の場合

AIX の環境変数に次の形式で指定します。

指定形式

```
CBLJ2CBOPT="[OPTION][: OPTION]..."
export CBLJ2CBOPT
```

[OPTION]

実行時ライブラリに渡すオプションを指定します。指定するときは引用符(")で囲んでください。現在用意されているオプションは comp5 オプション，dynamicpath オプション，encode オプション，endian オプション，japanese オプション，loadingrule オプション，および unicode オプションです。

#### (3)　Linux(x64)の場合

Linux(x64)の環境変数に次の形式で指定します。

指定形式

```
CBLJ2CBOPT="[OPTION][: OPTION]..."
export CBLJ2CBOPT
```

[OPTION]

実行時ライブラリに渡すオプションを指定します。指定するときは引用符(")で囲んでください。現在用意されているオプションは comp5 オプション，dynamicpath オプション，encode オプション，endian オプション，japanese オプション，loadingrule オプション，および unicode オプションです。

注

日本語項目および日本語編集項目を使用する場合は，unicode オプションを必ず指定してください。詳細については，「付録 A.5　Linux で使用する場合の注意事項」を参照してください。

## 5.4.2　comp5 オプション

### (1)　機能

USAGE 句に COMPUTATIONAL-5 または COMP-5 を指定した 2 進項目のエンディアン形式を変更する場合に指定します。このオプションは，「COMP-5 を COMP として扱う。」チェックボックスをオフにして生成した Bean に対してだけ有効です。

### (2)　指定形式

- Windows の場合

```
SET CBLJ2CBOPT=comp5( little | big )
```

comp5 オプション未指定時は，little として扱います。

- AIX の場合

comp5 オプションは引用符（"）で囲んで指定します。

```
CBLJ2CBOPT="comp5( little | big )"
export CBLJ2CBOPT
```

comp5 オプション未指定時は，big として扱います。

- Linux の場合

comp5 オプションは引用符（"）で囲んで指定します。

```
CBLJ2CBOPT="comp5( little | big )"
export CBLJ2CBOPT
```

comp5 オプション未指定時は，little として扱います。

### (3)　指定値の動作

comp5(little)：COMPUTATIONAL-5/COMP-5 を指定した 2 進項目をリトルエンディアンで扱います。

comp5(big)：COMPUTATIONAL-5/COMP-5 を指定した 2 進項目をビッグエンディアンで扱います。

### (4) 注意事項

このオプションは，次の Bean で受け渡す，COMPUTATIONAL-5/COMP-5 指定の 2 進項目に対して有効です。

1. COBOL adapter for Cosminexus Version 2 02-05 以降で「COMP-5 を COMP として扱う。」チェックボックスをオフにして生成した Bean
2. Windows(x86) COBOL2002 Cosminexus 連携機能 01-02 以降の開発環境で，「COMP-5 を COMP として扱う。」チェックボックスをオフにして生成した Bean

## 5.4.3 dynamicpath オプション

### (1) 機能

COBOL アクセス用 Bean を生成する際に，呼び出す COBOL UAP ライブラリ名を絶対パス名指定でなくても COBOL UAP ライブラリを呼び出すことができます。この機能は，オプションの指定と次の環境変数の指定によって有効となります。

- Windows の場合：システム環境変数 PATH
- AIX の場合：システム環境変数 LIBPATH
- Linux の場合：システム環境変数 LD_LIBRARY_PATH

### (2) 指定形式

- Windows の場合

```
SET CBLJ2CBOPT=dynamicpath( yes | no )
```

- UNIX の場合

```
CBLJ2CBOPT="dynamicpath( yes | no )"
export CBLJ2CBOPT
```

dynamicpath オプション未指定時は，no として扱います（絶対パス名の指定が必要です）。

### (3) 指定値の動作

- Windows の場合

  yes 指定時は，指定されたライブラリ名でローディングします。システム環境変数 PATH に COBOL UAP ライブラリが格納されたフォルダ名を指定しておくと，COBOL UAP ライブラリをローディングできます。

  絶対パス名で指定された場合，指定された名称でローディングします。

dynamicpath オプション未指定時または no 指定時に，絶対パス名で指定されていない場合にはエラーとなります。

- UNIX の場合

  絶対パス名で指定されない（ / から始まらないファイル名）場合，指定されたライブラリ名でローディングします。ライブラリ名に拡張子がない場合は拡張子（AIX の場合：.a，Linux の場合：.so）を付けたファイル名をライブラリ名とします。システム環境変数 LIBPATH（AIX の場合）またはシステム環境変数 LD_LIBRARY_PATH（Linux の場合）に COBOL UAP ライブラリが格納されたディレクトリ名を指定しておくと，COBOL UAP ライブラリをローディングできます。

  絶対パス名で指定された場合，指定された名称でローディングします。

### (4) 注意事項

- UNIX の場合

  指定されたライブラリ名にピリオド(.)がある場合，dynamicpath オプションでは拡張子（AIX の場合：.a，Linux の場合：.so）は付加しません。

## 5.4.4 encode オプション

### (1) 機能

COBOL アクセスの文字列データ変換時のエンコードを変更します。

### (2) 指定形式

エンコード名は任意のエンコードで指定します。

- Windows の場合

```
set CBLJ2CBOPT=encode(エンコード名)
```

  encode オプション未指定時は，次に記載するデフォルトエンコードとして扱います。

- UNIX の場合

  encode オプションは引用符（"）で囲んで指定します。

```
CBLJ2CBOPT="encode(エンコード名)"
export CBLJ2CBOPT
```

  encode オプション未指定時は，次に記載するデフォルトエンコードとして扱います。

### (3) 指定値の動作

指定されたエンコードで文字列変換のエンコードを行います。呼び出す COBOL プログラムで使用する文字コードに対応したエンコードを指定してください。

### (4) 注意事項

指定された文字列の値はチェックしません。指定時には大文字，小文字にご注意ください。また，サポートされていないエンコードを指定した場合の動作は保証しません。

### (5) デフォルトエンコード

文字列変換のエンコードはシステムに依存します。

encode オプションを指定しない場合のデフォルトエンコードは「表 5-1　デフォルトエンコード一覧」を参照してください。

表 5-1　デフォルトエンコード一覧

| OS | システム環境変数 LANG | コード系 | デフォルトエンコード |
|---|---|---|---|
| Windows | − | シフト JIS | MS932 |
| AIX | Ja_JP | シフト JIS | Cp943C |
| | ja_JP | 日本語 EUC | Cp33722C |
| Linux | ja_JP.UTF-8 | UTF-8 | UTF-8 |

### (6) エンコード対象となる項目

COBOL プログラムに渡す引数で次に示す項目が対象となります。

- 英字項目
- 英数字項目
- 英数字編集項目
- 日本語項目
- 日本語編集項目

## 5.4.5 endian オプション

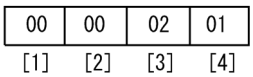
### (1) 機能

コンピュータで扱うバイナリデータ（2 進項目および内部浮動小数点項目）の形式が Windows，AIX，または Linux で異なります。Windows および Linux ではリトルエンディアン形式で，AIX ではビッグエンディアン形式となります。

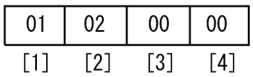
（例）

513（=X'00000201'）が 4 バイトの 2 進項目に配置された場合の違い

- ビッグエンディアン形式（AIX）




- リトルエンディアン形式（Windows，Linux）




Windows または Linux の COBOL では，通常リトルエンディアン形式でデータを扱いますが，COBOL プログラムのコンパイル時に-BigEndian オプションを指定すると，ビッグエンディアン形式でデータを扱うことができます。ビッグエンディアン形式でデータを扱うプログラムを Java から呼び出して使用したい場合には，このオプションに big を指定すると，ビッグエンディアン形式でデータを受け渡しすることができます。

## (2) 指定形式

- Windows の場合

```
SET CBLJ2CBOPT=endian( little | big )
```

  endian オプション未指定時は，little として扱います。

- AIX の場合

  endian オプションは引用符（"）で囲んで指定します。

```
CBLJ2CBOPT="endian( little | big )"
export CBLJ2CBOPT
```

  endian オプション未指定時は，big として扱います。

- Linux の場合

  endian オプションは引用符（"）で囲んで指定します。

```
CBLJ2CBOPT="endian( little | big )"
export CBLJ2CBOPT
```

  endian オプション未指定時は，little として扱います。

## (3) 指定値の動作

endian(little)： 2 進項目をリトルエンディアンで扱います。

endian(big)： 2 進項目をビッグエンディアンで扱います。

### (4) 注意事項

このオプションは，次の項目に対して有効です。

- 浮動小数点項目
- COMPUTATIONAL-5／COMP-5 を除いた 2 進項目
- 次に示す Bean で受け渡す，COMPUTATIONAL-5／COMP-5 指定の 2 進項目
  1. COBOL adapter for Cosminexus Version 2 02-05 未満で生成した Bean
  2. COBOL adapter for Cosminexus Version 2 02-05 以降で「COMP-5 を COMP として扱う。」チェックボックスをオンにして生成した Bean
  3. Windows(x86) COBOL2002 Cosminexus 連携機能 01-02 未満の開発環境で生成した Bean
  4. Windows(x86) COBOL2002 Cosminexus 連携機能 01-02 以降の開発環境で「COMP-5 を COMP として扱う。」チェックボックスをオンにして生成した Bean

Windows または Linux の場合 COBOL アクセスで，このオプションに big を指定する場合は，COBOL プログラムコンパイル時に-BigEndian,Bin コンパイラオプションと-BigEndian,Float コンパイラオプションを指定してください。

## 5.4.6 japanese オプション

### (1) 機能

日本語項目を扱う場合に指定します。このオプションは，「日本語項目を英数字項目として扱う。」チェックボックスをオフにして生成した Bean の日本語項目（日本語編集項目を含む）に対してだけ有効です。

### (2) 指定形式

- Windows の場合

```
SET CBLJ2CBOPT=japanese( 1 | 2 )
```

  japanese オプション未指定時は，2 として扱います。

- UNIX の場合

  japanese オプションは引用符（"）で囲んで指定します。

```
CBLJ2CBOPT="japanese( 1 | 2 )"
export CBLJ2CBOPT
```

  japanese オプション未指定時は，2 として扱います。

### (3) 指定値の動作

japanese(1)：設定するデータの長さが項目長よりも短い場合，半角の空白を補います。

japanese(2)：設定するデータの長さが項目長よりも短い場合，全角の空白を補います。空白を補う領域長が奇数バイトの場合は，すべて半角の空白で補います。

### (4) 注意事項

このオプションは，次の Bean で受け渡す日本語項目および日本語編集項目に対して有効です。

1. COBOL adapter for Cosminexus Version 2 02-05 以降で「日本語項目を英数字項目として扱う。」チェックボックスをオフにして生成した Bean
2. Windows(x86) COBOL2002 Cosminexus 連携機能 01-02 以降で「日本語項目を英数字項目として扱う。」チェックボックスをオフにして生成した Bean

## 5.4.7 loadingrule オプション

### (1) 機能

COBOL アクセス用 Bean から呼び出す COBOL UAP ライブラリのロード／アンロードを制御したい場合に指定します。

### (2) 指定形式

- Windows の場合

```
SET CBLJ2CBOPT=loadingrule( resident | noresident )
```

- UNIX の場合

  loadingrule オプションは引用符（"）で囲んで指定します。

```
CBLJ2CBOPT="loadingrule( resident | noresident )"
export CBLJ2CBOPT
```

loadingrule オプション未指定時は，resident として扱います。

### (3) 指定値の動作

loadingrule(resident)：COBOL アクセス用 Bean から呼び出されるすべての COBOL UAP ライブラリをアンロードしません。

loadingrule(noresident)：COBOL UAP ライブラリをアンロードします。ただし，環境変数 CBLJ2CBSTAYLIB に指定がある COBOL UAP ライブラリはアンロードしません。

### (4) 注意事項

- Windows の場合，COBOL アクセス用 Bean から呼び出す COBOL UAP ライブラリを COBOL ダイナミックリンク機能の COBOL UAP ライブラリとしても使用する場合，必ず環境変数 CBLJ2CBSTAYLIB に共用したい COBOL UAP ライブラリ名を指定する必要があります。UNIX の場合には，COBOL ダイナミックリンク機能の COBOL UAP ライブラリがアンロードされないため該当しません。
- COBOL アクセス用 Bean から呼び出す COBOL UAP ライブラリがすでにロードされている場合，ローディングしません。そのため，COBOL UAP からさらに呼び出した C UAP などで，COBOL アクセス用 Bean から呼び出す COBOL UAP ライブラリのアンロードはしないでください。

## 5.4.8 unicode オプション

### (1) 機能

COBOL UAP と受け渡す引数の，文字列データ変換時のエンコードを UTF-8 および UTF-16 で行うことを指定します。

### (2) 指定形式

- Windows の場合

```
SET CBLJ2CBOPT=unicode( no | big | little )
```

unicode オプション未指定時は，no として扱います。

- UNIX の場合

unicode オプションは引用符（”）で囲んで指定します。

```
CBLJ2CBOPT=”unicode( no | big | little )”
export CBLJ2CBOPT
```

unicode オプション未指定時は，no として扱います。

### (3) 指定値の動作

unicode(no)：文字列データを encode オプション指定に従ってエンコードします。encode オプション未指定時は，デフォルトエンコードで変換します。デフォルトエンコードについては，「5.4.4 encode オプション」の「(5) デフォルトエンコード」の「表 5-1 デフォルトエンコード一覧」を参照してください。

unicode(big/little)：文字列データを UTF-8 または UTF-16 でエンコードします。

英数字項目は UTF-8 でエンコードします。

日本語項目は，big 指定時は UTF-16BE で，little 指定時は UTF-16LE でエンコードします。

## (4) 注意事項

- このオプションに big または little を指定した場合，encode オプションの指定は無効となります。
- 日本語項目として扱う場合は，「日本語項目を英数字項目として扱う。」チェックボックスをオフにして，COBOL アクセス用 Bean を生成してください。

  また，Linux の場合は，unicode オプションを必ず指定してください。詳細については，「付録 A.5 Linux で使用する場合の注意事項」を参照してください。

## (5) エンコード対象となる項目

COBOL UAP に受け渡す引数で次に示す項目が対象となります。

- 英字項目
- 英数字項目
- 英数字編集項目
- 日本語項目
- 日本語編集項目

## (6) 設定するデータの長さが項目長よりも短い場合に補う文字

1. 英数字項目は UTF-8 の半角空白(0x20)を補います。
2. 日本語項目は japanese オプションの指定によって，補う文字が異なります。big 指定時は次に示す文字を補います。
   - japanese(1)：UTF-16 の半角空白(0x0020)を補います。
   - japanese(2)：UTF-16 の全角空白(0x3000)を補います。

## 5.5　環境変数 CBLJ2CB_DDUMP

### 5.5.1　機能

COBOL アクセス用 Bean の setter／getter 呼び出し時のデータ領域の情報を出力する機能です。環境変数または Java のシステムプロパティでフォルダ名が設定されない場合，ダンプファイル（引数情報を格納するファイル）は出力されません。機能の詳細については，「7.2　引数情報表示機能」を参照してください。

### 5.5.2　指定方法

ダンプファイル出力先フォルダ名とファイルサイズの上限を設定します。

- Windows の場合

  Windows の環境変数を設定します。

  ```
  SET CBLJ2CB_DDUMP=フォルダ名[,filesize(nnnn)]
  ```

  フォルダ名は，ダンプファイル出力先フォルダ名をドライブ名から指定します。

- UNIX の場合

  環境変数 CBLJ2CB_DDUMP は引用符（"）で囲んで指定します。

  ```
  CBLJ2CB_DDUMP="ディレクトリ名[,filesize(nnnn)]"
  export CBLJ2CB_DDUMP
  ```

  ディレクトリ名は，ダンプファイル出力先ディレクトリ名をルートディレクトリ（/）から指定します。

- 指定時の規則

  filesize(nnnn)を指定する際は，コンマ(,)を付けて指定します。

  nnnn：作成するダンプファイルのファイルサイズの上限を MB 単位で指定します。

  filesize の指定がない場合，10(MB)を仮定します。

  指定可能な範囲は，1～2000 です。範囲外の場合や指定がない場合は，10 を仮定します。

### 5.5.3　指定例

- Windows の場合

  ```
  ファイルサイズの上限を指定しない例
      SET CBLJ2CB_DDUMP=c:¥users

  ファイルサイズの上限に100MBを指定した例
      SET CBLJ2CB_DDUMP=c:¥users,filesize(100)
  ```

- UNIX の場合

```
ファイルサイズの上限を指定しない例
    CBLJ2CB_DDUMP=”/users”
    export CBLJ2CB_DDUMP

ファイルサイズの上限に100MBを指定した例
    CBLJ2CB_DDUMP=”/users,filesize(100)”
    export CBLJ2CB_DDUMP
```

## 5.6　環境変数 CBLJ2CBSTAYLIB

### 5.6.1　機能

COBOL アクセス用 Bean から呼び出す COBOL プログラムのライブラリが，呼び出し後にアンロードされる場合，環境変数 CBLJ2CBSTAYLIB を設定すると，アンロードしないで常駐化できます。

COBOL アクセス用 Bean を生成する際に指定したライブラリ名称を環境変数 CBLJ2CBSTAYLIB に指定しておくと，最初にライブラリを呼び出したときに常駐化されます。これ以降，常駐化されたライブラリを呼び出すようになります。

COBOL アクセス用 Bean を生成する際に指定したライブラリ名称が環境変数 CBLJ2CBSTAYLIB に指定されていない場合，AIX の Cosminexus 実行環境では，呼び出したライブラリはアンロードされます。Windows，または Linux の Cosminexus 実行環境では，呼び出したライブラリはアンロードされません。

### 5.6.2　指定形式

- Windows の場合

```
SET CBLJ2CBSTAYLIB=libname[:libname]・・・
```

指定例

```
SET CBLJ2CBSTAYLIB=sample1:sample2
```

- UNIX の場合

環境変数 CBLJ2CBSTAYLIB は引用符（"）で囲んで指定します。

```
CBLJ2CBSTAYLIB=”libname[:libname]・・・”
export CBLJ2CBSTAYLIB
```

指定例（AIX の場合）

```
CBLJ2CBSTAYLIB=”/home/lib/libsample1.a:/home/lib/libsample2.a”
export CBLJ2CBSTAYLIB
```

### 5.6.3　注意事項

- Windows の場合，dynamicpath オプションに yes を指定してください。 dynamicpath オプション未指定時または no 指定時には，環境変数 CBLJ2CBSTAYLIB で指定したライブラリは常駐化対象になりません。

- 環境変数 CBLJ2CBSTAYLIB に指定された COBOL プログラムのライブラリは，Cosminexus Application Server が終了するまで常駐します。使用頻度の低いライブラリを指定すると，使用可能なメモリ量が少なくなりますので，使用頻度の高いライブラリを常駐化するようにしてください。
- UNIX の場合，atexit システム関数で，終了時処理を行う関数を登録した COBOL プログラムのライブラリを呼び出して使用すると，Cosminexus Application Server の停止時にシグナル SIGILL（無効な命令）エラーが発生し，異常終了することがあります。これは，atexit システム関数で登録した関数が，COBOL プログラムのライブラリのアンロードと同時にアンロードされるために発生するものです。このような現象が発生する場合は，ライブラリを常駐化すると，この問題を発生しないようにすることができます。Windows の場合には，該当しません。
- UNIX の場合，dynamicpath オプションを指定して，環境変数 CBLJ2CBSTAYLIB で呼び出す COBOL 共用ライブラリを常駐化させたいときは，ディレクトリ名や拡張子を付けないで，ライブラリ名だけを指定してください。

  指定例

```
CBLJ2CBSTAYLIB="libsample1:libsample2"
export CBLJ2CBSTAYLIB
```

# 6 プログラムの実行

プログラムを実行するには，これまで作成してきたそれぞれのファイルを適切な場所に配置する必要があります。この章では，Web アプリケーションサーバに Cosminexus の Web コンテナサーバを用いてプログラムを実行する方法を，ファイルの種類に分けて説明します。

なお，Web コンテナサーバ（サーブレットエンジンモード）は，Cosminexus 製品が提供する互換用のモードです。プログラムの種類によっては Web コンテナサーバ上で動作できますが，J2EE サーバモードで動作させることをお勧めします。J2EE サーバ上での実行方法については，使用している Cosminexus のバージョンに対応した，マニュアル「Cosminexus　アプリケーション開発ガイド」を参照してください。

## 6.1　プログラムの実行方法

Web コンテナサーバ上での実行方法について説明します。実施する手順は次のとおりです。

1. Web アプリケーションの配置
2. サーブレットの登録
3. プログラムの実行

J2EE サーバ上での実行方法については，使用している Cosminexus のバージョンに対応した，マニュアル「Cosminexus　アプリケーション開発ガイド」を参照してください。

### 6.1.1　Web アプリケーションの配置

アプリケーションを実行する前に，次に示すどちらかの手順を実行してください。

#### (1)　XML ファイルで配置する場合

次のディレクトリの下に Web アプリケーションを配置します。

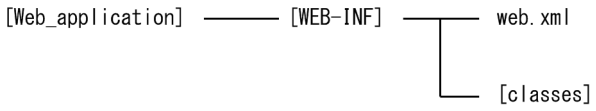
- Windows の場合

```
<CCインストール先>¥web¥containers¥<サーバ名>¥webapps
```

- UNIX の場合

```
<CCインストール先>/web/containers/<サーバ名>/webapps
```

Web アプリケーションに必要なディレクトリおよび xml ファイルの構成を次に示します。




構成の説明

[Web_application]：任意の名称

[WEB-INF]ディレクトリ：固定名称のディレクトリ

web.xml ファイル：web アプリケーションで使用する

[classes]ディレクトリ：クラスファイル格納ディレクトリ（固定名称）

パッケージの場合，classes の下にパッケージ名のディレクトリを作成し，このディレクトリにクラスファイル（servlet，bean）を配置します。

#### (2)　war ファイルで配置する場合

次に示すコマンドで war ファイルを作成します。

```
jar  cvf  [Web_application].war  [Web_application]
```

構成の説明

[Web_application]は任意の名称

## 6.1.2　サーブレットの登録

### (1)　サーブレットの登録方法

サーブレットを登録する場合，次の XML ファイルに設定します。

- Windows の場合

```
<CCインストール先>¥web¥containers¥
<サーバ名>¥webapps¥[Web_application]¥WEB-INF¥web.xml
```

- UNIX の場合

```
<CCインストール先>/web/containers/
<サーバ名>/webapps/[Web_application]/WEB-INF/web.xml
```

web.xml の記述は<web-app></web-app>がルートタグになります。

サーバ名：Cosminexus がインストールされているマシン名

[Web_application]：任意の名称

### (2)　サーブレットの追加方法

サーブレットの追加方法は，<servlet></servlet>タグ内の<servlet-name></servlet-name>タグにサーブレットの名称を，<servlet-class></servlet-class>タグにサーブレットクラスをパッケージ.クラス名で指定します。この場合，.class の指定は必要ありません（web.xml 内には複数のサーブレットを登録できます)。

### (3)　別名でマッピングする場合

作成したサーブレットを別名でマッピングする場合は，<servlet></servlet>タグ内の<servlet-mapping></servlet-mapping>タグにサーブレット名称を指定し，<url-pattern></url-pattern>にマッピング名称を指定します。

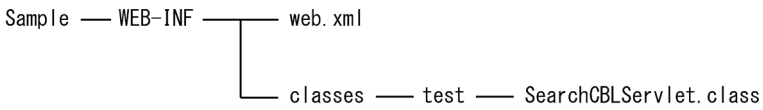
### (4)　web.xml の作成例

次に示す Web アプリケーションを例に web.xml の作成例を示します。

### (a) Web アプリケーション構成

- アプリケーション名:Sample
- Servlet 名称:SearchCBLServlet
- Package 名:test
- Mapping 名: /test.SearchCBLServlet

### (b) ディレクトリ構成




### (c) web.xml ソースコード例

```
<web-app>
     <servlet>
          <servlet-name>SearchCBLServlet</servlet-name>
          <servlet-class>
            test. SearchCBLServlet
          </servlet-class>
     </servlet>
     <servlet-mapping>
          <servlet-name>SearchCBLServlet</servlet-name>
          <url-pattern>
            /test.SearchCBLServlet
          </url-pattern>
     </servlet-mapping>
</web-app>
```

## 6.1.3 プログラムの実行

作成したそれぞれのプログラムをそれぞれが対応した場所に格納し，Web アプリケーションサーバを起動すれば，実行できます。なお，Web アプリケーションサーバ起動中に再作成した COBOL DLL ファイル（Windows の場合），または COBOL 共用ライブラリ（UNIX の場合）を入れ替えることはできません。COBOL DLL ファイル（Windows の場合），または COBOL 共用ライブラリ（UNIX の場合）を再作成した場合は，Web アプリケーションサーバを停止後，再作成した COBOL DLL ファイル（Windows の場合），または COBOL 共用ライブラリ（UNIX の場合）を再配置し，Web アプリケーションサーバを再起動してください。

## (1) Web コンテナサーバで JavaBean を実行する場合

### (a) Windows の場合

Web アプリケーションサーバに Cosminexus の Web コンテナサーバを使用した場合の各プログラムの格納場所を次に示します。

表 6-1 JavaBean を実行する場合のプログラムの格納場所例（Windows の場合）

| 項番 | 作成したファイル | 格納場所 |
|---|---|---|
| 1 | Html ファイル | ¥Web コンテナサーバ¥サーバ名¥webapps¥アプリケーション名※1 |
| 2 | JSP ファイル | ¥Web コンテナサーバ¥サーバ名¥webapps¥アプリケーション名※1 |
| 3 | Servlet クラスファイル | ¥Web コンテナサーバ¥サーバ名¥webapps¥アプリケーション名¥WEB-INF¥classes¥パッケージ名※1 |
| 4 | Bean クラスファイル | ¥Web コンテナサーバ¥サーバ名¥webapps¥アプリケーション名¥WEB-INF¥classes¥パッケージ名※1 |
| 5 | web.xml ファイル | ¥Web コンテナサーバ¥サーバ名¥webapps¥アプリケーション名¥WEB-INF※1 |
| 6 | COBOL DLL ファイル | COBOL アクセス用 Bean 生成ツールで指定した COBOL ライブラリ格納フォルダ※2 |

注※1

¥Web コンテナサーバは<CC インストール先>¥web¥containers です。

注※2

Windows の場合，dynamicpath オプション指定時は，システム環境変数 PATH に COBOL DLL 格納フォルダを指定する必要があります。

### (b) UNIX の場合

Web アプリケーションサーバに Cosminexus の Web コンテナサーバを使用した場合の各プログラムの格納場所を次に示します。

表 6-2 JavaBean を実行する場合のプログラムの格納場所例（UNIX の場合）

| 項番 | 作成したファイル | 格納場所 |
|---|---|---|
| 1 | Html ファイル | /opt/Hitachi/httpsd/htdocs |
| 2 | JSP ファイル | /opt/Hitachi/httpsd/htdocs |
| 3 | Servlet クラスファイル | /Web コンテナサーバ/サーバ名/webapps/アプリケーション名/WEB-INF/classes/パッケージ名※ |
| 4 | Bean クラスファイル | /Web コンテナサーバ/サーバ名/webapps/アプリケーション名/WEB-INF/classes/パッケージ名※ |
| 5 | web.xml ファイル | /Web コンテナサーバ/サーバ名/webapps/アプリケーション名/WEB-INF※ |

| 項番 | 作成したファイル | 格納場所 |
|---|---|---|
| 6 | COBOL 共用ライブラリ | COBOL アクセス用 Bean 生成ツールで指定した COBOL 共用ライブラリ格納ディレクトリ |

注※

/Web コンテナサーバは<CC インストール先>/web/containers です。

## (2) Web コンテナサーバで EJB だけを使用する（呼び出す）場合

### (a) Windows の場合

Web アプリケーションサーバに Cosminexus の Web コンテナサーバを使用した場合の各プログラムの格納場所を次に示します。

表 6-3 EJB だけを使用する（呼び出す）場合のプログラムの格納場所例（Windows の場合）

| 項番 | 作成したファイル | 格納場所 |
|---|---|---|
| 1 | Html ファイル | ¥Web コンテナサーバ¥サーバ名¥webapps¥アプリケーション名※ |
| 2 | Servlet クラスファイル | ¥Web コンテナサーバ¥サーバ名¥webapps¥アプリケーション名¥WEB-INF¥classes¥パッケージ名※ |

注※

¥Web コンテナサーバは<CC インストール先>¥web¥containers です。

### (b) UNIX の場合

Web アプリケーションサーバに Cosminexus の Web コンテナサーバを使用した場合の各プログラムの格納場所を次に示します。

表 6-4 EJB だけを使用する（呼び出す）場合のプログラムの格納場所例（UNIX の場合）

| 項番 | 作成したファイル | 格納場所 |
|---|---|---|
| 1 | Html ファイル | /opt/Hitachi/httpsd/htdocs |
| 2 | Servlet クラスファイル | /Web コンテナサーバ/サーバ名/webapps/アプリケーション名/WEB-INF/classes/パッケージ名※ |

注※

/Web コンテナサーバは<CC インストール先>/web/containers です。

## (3) Cosminexus Component Container 使用時の注意事項

Cosminexus Component Container の「業務アプリケーションの実行監視機能」や「業務アプリケーションの強制停止機能」,「業務アプリケーションのリデプロイ機能」を利用する場合には，次の値を Cosminexus Component Container の提供する「保護区リストファイル」に追加してください。

【値】

```
jp.co.hitachi_sk.j2cb.*
```

「保護区リストファイル」の詳細は，使用しているCosminexusのバージョンに対応した，マニュアル「Cosminexus　リファレンス　定義編（サーバ定義）」を参照してください。

## 6.2　EJB 対応 COBOL アクセス使用時の注意事項

COBOL プログラム中の DB アクセスは JTS（Java Transaction Service）に連携できません。

# 7 プログラムのデバッグ

この章では，作成した COBOL UAP と Java プログラムのデバッグに関する操作を説明します。

## 7.1　プログラムのデバッグ

プログラムのデバッグについて説明します。COBOL UAP のデバッグ機能は，Linux では使用できません。

### 7.1.1　Servlet のデバッグ

通常の Servlet と同様の手順でデバッグできます。

Servlet のデバッグについては，使用している Cosminexus のバージョンに対応した，マニュアル「Cosminexus アプリケーション開発ガイド」を参照してください。

### 7.1.2　COBOL UAP のデバッグ（Windows，AIX の場合）

COBOL アクセス環境下では，テストデバッガを使用してデバッグできます。デバッグは，Web コンテナサーバを使用した開発環境で行うことができます。

COBOL2002 ではコンパイル時に，デバッグオプション -TDInf を指定し，Cosminexus 連携の実行時に COBOL2002 が提供する環境変数（CBLTDEXEC など）を設定することで，COBOL2002 テストデバッガを使用して COBOL プログラムのデバッグを行うことができます。

テストデバッガの使用方法については，マニュアル「COBOL2002　操作ガイド」またはマニュアル「COBOL2002　使用の手引　操作編」を参照してください。

#### (1)　Windows の場合

コントロールパネルなどの環境変数設定画面で次の環境変数を設定すると，COBOL プログラムを連動して実行できます。

- CBLTDEXEC=TD
- CBLPIDIR=CBPファイルのパス名



### (a)　Web コンテナサーバまたは Java EE サーバを使用する場合

ユーザ定義ファイルに「cobol.sysenvfile=環境設定ファイルの物理ファイル名」を設定します。詳細は,「5.1　COBOL アクセスでの COBOL 環境変数の設定」を参照してください。

また，Java UAP で COBOL UAP の実行時エラーメッセージの内容を取得できます。取得のためのメソッドは「8.3　J2CBException ユーザインタフェース API」および「8.4　EJB 用 Exception ユーザインタフェース API」を参照してください。

COBOL UAP の実行時エラーメッセージの内容を取得するためには，環境変数 CBL_SYSERR で実行時エラーメッセージの出力先ファイルを指定する必要があります。環境変数 CBL_SYSERR の詳細については，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

## (2)　AIX の場合

次の環境変数を設定すると，ラインモードによる COBOL プログラムを連動して実行できます。

```
・CBLTDEXEC=TL
  ＜指定例＞
    CBLTDEXEC="TL -Library COBOL共用ライブラリ名
                  -Execute /opt/Cosminexus/CC/web/bin/cjstartweb"
・CBLPIDIR=CBPファイルのパス名
・CBLTDDISPLAY
  ＜指定例＞
    CBLTDDISPLAY=10.210.41.18:0.0
・CBLLPATH=COBOL共用ライブラリが格納されているパス名
  ＜指定例＞
    CBLLPATH=/javacobol/s93tpl/TD2002/tp/cobol
・CBLLSLIB=COBOL共用ライブラリ名
  環境変数CBLTDEXECでCOBOL共用ライブラリ名を指定しなかった場合，
  この環境変数でCOBOL共用ライブラリを指定する。
```

## 7.2　引数情報表示機能

COBOL アクセス用 Bean の setter／getter 呼び出し時のデータ領域の情報を出力する機能です。環境変数 CBLJ2CB_DDUMP※に出力先フォルダ名を指定することで，引数情報の表示内容をダンプファイル（表示した引数情報を格納するファイル）に出力します。

注※

指定方法については，「5.5　環境変数 CBLJ2CB_DDUMP」を参照してください。

### 7.2.1　ダンプファイルの出力例

次に，ダンプファイルの出力例を示します。詳細な出力例は「7.2.4　出力フォーマット」を参照してください。

```
2013/09/30 11:32:11 (1) Search COBOL 拡張 Server Run Time System for Cosminexus Version 2  02-
13  *** Data Area Dump ***
2013/09/30 11:32:11 (1) Search < package  : test >
2013/09/30 11:32:11 (1) Search < name  : setP_number >
2013/09/30 11:32:11 (1) Search setter Before Data  J-Type:[String]  size: 12
2013/09/30 11:32:11 (1) Search 00000000: 41 42 43 44 45 46 47 48 - 49 4A 4B 4C
*ABCDEGFHIJKL    *
2013/09/30 11:32:11 (1) Search setter After Data   CBL-Type:[X(50)]  size: 50 location: 4
2013/09/30 11:32:11 (1) Search 00000000: 41 42 43 44 45 46 47 48 - 49 4A 4B 4C 20 20 20 20
*ABCDEFGHIJKL    *
2013/09/30 11:32:11 (1) Search 00000010: 20 20 20 20 20 20 20 20 - 20 20 20 20 20 20 20 20
*                *
```

### 7.2.2　ダンプファイル

ダンプファイルは，環境変数 CBLJ2CB_DDUMP で指定したフォルダに出力されます。出力するファイル名は，COBOL アクセス用 Bean の種類によって，次のファイル名になります。

| 項番 | 対応するアクセス用 Bean | ファイル名 |
|---|---|---|
| 1 | JavaBean 用の COBOL アクセス用 Bean の場合 | J2cb_1.log , J2cb_2.log |
| 2 | EJB 用の COBOL アクセス用 Bean の場合 | J2cbe_1.log, J2cbe_2.log |

引数情報表示機能では，二つのダンプファイルを切り替えながら出力します。切り替えは，ダンプファイルのファイルサイズが，指定されたファイルサイズの上限を超える場合に行います。ファイルサイズの上限値を指定しない場合のデフォルトサイズは，10MB です。ファイルサイズに指定できる値は，1～2000 で，単位は MB です。

指定したフォルダに上記ファイルが存在しない場合は，新規にファイルを作成します。すでにファイルが存在する場合は，ファイルの最後に追加書きします。ダンプファイル出力先フォルダの指定がない場合は，この機能は無効となり，ダンプファイルは出力しません。

すでに存在するファイルが読み取り専用の場合は，例外が発生します。

複数スレッドで実行する際も，上記ファイルに追加書きします。

## 7.2.3　制限事項

この機能は，Windows(x86) COBOL2002 Cosminexus 連携機能または COBOL adapter for Cosminexus Version 2 以降で生成した COBOL アクセス用 Bean で使用できます。COBOL adapter for Cosminexus （01-xx）で生成した COBOL アクセス用 Bean を実行した場合，引数情報は表示しません。

## 7.2.4　出力フォーマット

### (1)　出力ファイルの概要

出力ファイルは，次のように出力されます。

```
識別情報        ヘッダ部識別情報
識別情報        データ領域ダンプリスト
```

### (2)　識別情報

```
 yyyy/MM/dd    hh:mm:ss    (X)  Bean名称
    1.            2.        3.     4.
```

1. yyyy：年（西暦） MM：月　dd：日
2. hh：時（1～24） mm：分　ss：秒
3. X：ローカル識別番号

   Bean を呼び出すごとにローカルな番号が設定されます（番号は 1～2,147,483,647 で，超えた場合は 1 となります)。
4. Bean 名称：Bean のクラス名称

### (3)　ヘッダ部

Windows の場合

```
COBOL 拡張 Server Run Time System for Cosminexus Version 2※  VV-RR(-ZZ)      *** Data Area
Dump ***
```

注※　Windows(x64)では，COBOL 拡張 Server Run Time System for Cosminexus Version 2(64) になります。

```
< package : *** 1 *** >
```

*** 1 *** ：Bean のパッケージ名称。パッケージがない場合は，”no-package”。

それぞれのヘッダ部には，識別情報が出力されます。

UNIX の場合

```
COBOL 拡張 Run Time System for Cosminexus Version 2  VV-RR(-ZZ)       *** Data Area Dump ***
```

```
< package : *** 1 *** >
```

*** 1 *** ：Bean のパッケージ名称。パッケージがない場合は，”no-package”。

それぞれのヘッダ部には，識別情報が出力されます。

## (4)　データ領域ダンプ

### (a)　基本型 setter の例

```
< name : setXxxx >
setter  Before Data  J-Type: [String]  size :50  [specified length:10]
00000000: 41 42 43 44 45 46 47 48 - 49 4A 4B 4C              *ABCDEFGHIJKL    *
  1.      2.                                              3.
setter After Data    CBL-Type : [X(50)]  size :50  location : 14
00000000: 41 42 43 44 45 46 47 48 - 49 4A 4B 4C 20 20 20 20  *ABCDEFGHIJKL    *
00000010: 20 20 20 20 20 20 20 20 - 20 20 20 20 20 20 20 20  *                *
                                    :
```

注

各行に識別情報が出力されます。

Before は変換前の情報を，After は変換後の情報をそれぞれ表します。setter の場合は，setter の引数に指定された値が Before に，引数領域に設定される値が After に表示されます。getter の場合は，引数領域の値が Before に，リターンする値が After に表示されます。

name：setter／getter のメソッド名を表します。

CBL-Type：

COBOL の定義を表します（詳細は「7.2.5　setter／getter 引数情報表示時のデータ属性情報」を参照してください）。

J-Type：

Java の定義を表します（詳細は「7.2.5　setter／getter 引数情報表示時のデータ属性情報」を参照してください）。

size：該当する引数データの領域長を表します（バイト数）。

setter の場合

Before は，setter で指定したデータのデータ長を表します（バイト数）。

After は，COBOL データ定義の領域長を表します（バイト数）。

getter の場合

Before は，COBOL データ定義の領域長を表します（バイト数）。

After は，getter で取得するデータのデータ長を表します（バイト数）。

specified length：

可変長データおよびアドレスデータを指定した場合，Setter の Before にユーザが指定したデータ長を表示します。

データ長に 0 以下を指定した場合は，0 を仮定し，表示します。

データ長が（全体長-4）を超える場合は，[全体長-4] を算出し，表示します。

そのほかのデータ項目には表示されません。

location：

引数領域先頭からの相対ロケーションを表します。COBOL データ定義の場合に，表示します。

1.：表示するデータ先頭からの相対ロケーションを表します（16 進表示）。

2.：データを 16 進表示します（16 バイト分表示します）。

3.：データをそのまま表示します。表示できない文字は，”?”を表示します。

可変長データおよびアドレスデータ項目のダンプ表示で setter の After と getter の Before の先頭 4 バイトはデータ長を表します。

同一内容のダンプ表示が続く場合は，先頭行に続く行に，次を表示します。

同一内容の行が 2 行ある場合

```
         LINE  00000010  SAME AS ABOVE
```

同一内容の行が 3 行以上ある場合

```
         LINES  00000010 - 00000020  SAME AS ABOVE
```

## (b)　基本型 getter の例

```
< name : getXxxx >
getter  Before Data  CBL-Type : [X(50)]  size :50  location : 14
00000000: 46 46 46 46 46 46 46 46 - 46 46 46 46 46 46 46 46  *FFFFFFFFFFFFFFFF*
            LINES 00000010 - 00000020 SAME AS ABOVE
00000030: 46 46                                              *FF              *
getter After Data    J-Type: [String]  size :50
00000000: 46 46 46 46 46 46 46 46 - 46 46 46 46 46 46 46 46  *FFFFFFFFFFFFFFFF*
            LINES 00000010 - 00000020 SAME AS ABOVE
00000030: 46 46                                              *FF              *
```

注

各行に識別情報が出力されます。

OCCURS 句が指定された場合は，指定された添字を setter／getter のメソッド名に表示します。

OCCURS 句が指定されたデータ項目に対する setter の例

```
< name : setXxxx[1,1,1,1,1,1,1] >
```

## (5) 引数情報表示時のエラーケース

次に記載するエラーケースは，情報表示時のエラー表示です。例外は発生しません。

### (a) setter の引数に指定した Object の型が誤っている場合

```
### Invalid Data Format ### : オブジェクト名
```

オブジェクト名：指定されたオブジェクト名を表示します。

### (b) 変換時にエラーが発生した場合

```
### The exception occurred ###
```

### (c) COBOL adapter for Cosminexus 01-xx で作成した COBOL アクセス用 Bean の場合

Windows の場合

```
COBOL 拡張 Server Run Time System for Cosminexus VV-RR(-ZZ)※  *** Data Area Dump ***
### This program is outside an object ###
```

注※　VV-RR(-ZZ)は流用元製品バージョン番号（例：02-13）を表します。

UNIX の場合

```
COBOL 拡張 Run Time System for Cosminexus VV-RR(-ZZ)※  *** Data Area Dump ***
### This program is outside an object ###
```

注※　VV-RR(-ZZ)は流用元製品バージョン番号（例：02-12）を表します。

### (d) getter で取得するデータが出力情報サイズに含まれない場合

```
### Data area is outside the range ###
```

### (e) getter で取得するデータの一部が出力情報サイズに含まれない場合

```
### Some argument data is outside an effective data area ###
```

## 7.2.5 setter／getter 引数情報表示時のデータ属性情報

ダンプファイルに出力されるデータ型を，表 7-1 に示します。

## 表 7-1　ダンプファイルに出力されるデータ型

| 項番 | COBOL のデータ属性 | COBOL のデータ定義※1 | ダンプファイル出力時のデータ属性表示 | |
|---|---|---|---|---|
| | | | CBL-Type※2 | J-Type |
| 1 | 英数字項目（文字列） | X　英数字項目※6 | X(n)※5 | String |
| | | N　日本語項目※7※10 | X(n)※5 | String |
| 2 | 1～4 けたの小数を含まない符号付き 2 進項目 | S9(4)<br>USAGE COMP※8 | S9(4)※3<br>COMP | Short |
| | | S9(4)<br>USAGE COMP-5※12 | S9(4)<br>COMP-5 | |
| 3 | 1～4 けたの小数を含まない符号なし 2 進項目 | 9(4)<br>USAGE COMP※8 | 9(4)※3<br>COMP | Short |
| | | 9(4)<br>USAGE COMP-5※12 | 9(4)<br>COMP-5 | |
| 4 | 5～9 けたの小数を含まない符号付き 2 進項目 | S9(9)<br>USAGE COMP※8 | S9(9)※3<br>COMP | Integer |
| | | S9(9)<br>USAGE COMP-5※12 | S9(9)<br>COMP-5 | |
| 5 | 5～9 けたの小数を含まない符号なし 2 進項目 | 9(9)<br>USAGE COMP※8 | 9(9)※3<br>COMP | Integer |
| | | 9(9)<br>USAGE COMP-5※12 | 9(9)<br>COMP-5 | |
| 6 | 10～18 けたの小数を含まない符号付き 2 進項目 | S9(18)<br>USAGE COMP※8 | S9(18)※3<br>COMP | Long |
| | | S9(18)<br>USAGE COMP-5※12 | S9(18)<br>COMP-5 | |
| 7 | 10～18 けたの小数を含まない符号なし 2 進項目 | 9(18)<br>USAGE COMP※8 | 9(18)※3<br>COMP | Long |
| | | 9(18)<br>USAGE COMP-5※12 | 9(18)<br>COMP-5 | |
| 8 | 小数を含む符号付き 2 進項目 | S9(17)V9(1)<br>USAGE COMP※8 | S9(17)V9(1)※4<br>COMP | BigDecimal |
| | | S9(17)V9(1)USAGE<br>COMP-5※12 | S9(17)V9(1)<br>COMP-5 | |
| 9 | 小数を含む符号なし 2 進項目 | 9(17)V9(1)<br>USAGE COMP※8 | 9(17)V9(1)※4<br>COMP | BigDecimal |

| 項番 | COBOL のデータ属性 | COBOL のデータ定義※1 | ダンプファイル出力時のデータ属性表示 | |
|---|---|---|---|---|
| | | | CBL-Type※2 | J-Type |
| | | 9(17)V9(1)<br>USAGE COMP-5※12 | 9(17)V9(1)<br>COMP-5 | |
| 10 | 左独立符号付き外部 10 進項目 | S9(18)<br>SIGN LEADING<br>SEPARATE | S9(18)※4<br>LEADING<br>SEPARATE | BigDecimal |
| 11 | 右独立符号付き外部 10 進項目 | S9(18)<br>SIGN TRAILING<br>SEPARATE | S9(18)※4<br>TRAILING<br>SEPARATE | BigDecimal |
| 12 | 符号なし外部 10 進項目 | 9(18) | 9(18)※4 | BigDecimal |
| 13 | 左符号付き外部 10 進項目 | S9(18)<br>SIGN LEADING | S9(18)※4<br>LEADING | BigDecimal |
| 14 | 右符号付き外部 10 進項目 | S9(18)<br>SIGN TRAILING | S9(18)※4<br>TRAILING | BigDecimal |
| 15 | 単精度内部浮動小数点項目 | USAGE COMP-1 | COMP-1 | Float |
| 16 | 倍精度浮動小数点項目 | USAGE COMP-2 | COMP-2 | Double |
| 17 | 符号付き内部 10 進項目 | S9(18)<br>USAGE COMP-3※9 | S9(18)※4<br>COMP-3 | BigDecimal |
| 18 | 符号なし内部 10 進項目 | 9(18)<br>USAGE COMP-3※9 | 9(18)※4<br>COMP-3 | BigDecimal |
| 19 | 英数字項目（バイト配列） | X　英数字項目※6 | X(n)※5<br>Byte | byte[] |
| | | N　日本語項目※7※10 | X(n)※5<br>Byte | byte[] |
| 20 | 日本語項目（文字列） | N　日本語項目※7※11 | N(n)※4 | String |
| 21 | 先頭 4 バイトの 2 進形式 | X(10)　(英数字項目) | X(10)<br>VARIABLE | byte[]V |
| 22 | アドレス項目 | ADDRESS | ADDRESS | byte[]P |
| 23 | 集団項目（バイト配列） | 集団項目※13 | G(n)※5<br>Byte | byte[] |

注※1

指定例です。

注※2

表示例です。表示されるけた数は，COBOL のデータ定義によります。

注※3

COBOL のデータ定義で指定されたけた数にかかわらず，このけた数で表示されます（02-05 未満で生成した Bean の場合)。

COBOL のデータ定義で指定されたけた数で表示されます(02-05 以降)。

注※4

COBOL のデータ定義で指定されたけた数で表示されます。

注※5

バイト数で表示されます。

注※6

このほかに，英字項目，英数字編集項目，数字編集項目があります。

注※7

このほかに，日本語編集項目があります。

注※8

このほかに，COMP-4，BINARY があります。

また，02-05 未満で生成した Bean および COBOL アクセス用 Bean 生成時に COMP-5 を COMP として扱うことを指定した COMP-5 も含みます。

注※9

このほかに，PACKED-DECIMAL があります。

注※10

COBOL アクセス用 Bean 生成時に，英数字項目として扱うことを指定した場合に，このような表示となります。CBL-Type で表示される長さは，けた数を 2 倍した値となります。

注※11

COBOL アクセス用 Bean 生成時に，日本語項目として扱うことを指定した場合に，このような表示となります。

注※12

COBOL アクセス用 Bean 生成時に，COMP-5 として扱うことを指定した 2 進項目の場合に，このような表示となります。

注※13

COBOL アクセス用 Bean 生成時に，集団項目のバイト配列アクセスをすることを指定した場合に，このような表示となります。

# 8 COBOL アクセス用 Bean を呼び出すための API

この章では，COBOL アクセス用 Bean を呼び出すための API（Application Programming Interface）について説明します。

## 8.1　提供クラス

COBOL アクセス用 Bean で提供するクラスの一覧を表 8-1 に示します。

### 表 8-1　提供クラス一覧

| 項番 | パッケージ名 | クラス名 | 機能 |
|---|---|---|---|
| 1 | jp.co.hitachi_sk.j2cb | CBLAccess | COBOL 呼び出しクラス |
| 2 | | J2CBException | COBOL プログラムで発生したエラー情報取得クラス（JavaBean 用） |
| 3 | | J2CBErrorUtil | COBOL プログラムで発生したエラー情報取得クラス（EJB 用） |

それぞれのクラスについては，「8.2　COBOL アクセス用 Bean ユーザインタフェース API」，「8.3 J2CBException ユーザインタフェース API」，および「8.4　EJB 用 Exception ユーザインタフェース API」で説明します。

## 8.2　COBOL アクセス用 Bean ユーザインタフェース API

CBLAccess クラスで提供するメソッドの一覧を表 8-2 に示します。

### 表 8–2　CBLAccess クラスの提供メソッド一覧

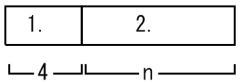
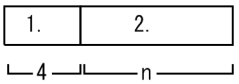
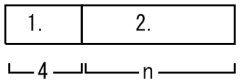
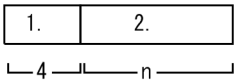
| 項番 | メソッド名 | 機能 |
|---|---|---|
| 1 | callCOBOL | COBOL 呼び出し |
| 2 | setXxx | 引数データの設定 |
| 3 | getXxx | 引数データの取得 |

### 8.2.1　COBOL の各データ項目の設定方法と設定値

Java のデータと受け渡しする COBOL の各データ項目の setter／getter 規則を表 8-3 に示します。

### 表 8–3　各データ項目の setter／getter 規則

| COBOL データ項目 | setter 変換規則 | getter 変換規則 | Java データ属性 |
|---|---|---|---|
| 英字項目<br>英数字項目<br>英数字編集項目<br>数字編集項目<br>日本語項目<br>日本語編集項目※4 | 1. String 文字列を encode オプション※1 または unicode オプション※9 に従って変換したバイト配列にする。<br>2. 1.の値の長さが COBOL データ項目長よりも長い場合は切り捨てられ，短い場合は半角空白が埋められる。 | 1. COBOL のデータ値を encode オプション※1 または unicode オプション※9 に従って変換した String オブジェクトを作成する。 | 文字列データ<br>(String) |
| 日本語項目<br>日本語編集項目※5 | 1. String 文字列を encode オプション※1 または unicode オプション※9 に従って変換したバイト配列にする。<br>2. 1.の値の長さが COBOL データ項目長よりも長い場合は切り捨てられ，短い場合は japanese オプション※3 に従って全角空白または半角空白が埋められる。 | 1. COBOL のデータ値を encode オプション※1 または unicode オプション※9 に従って変換した String オブジェクトを作成する。 | 日本語データ<br>(String) |
| 英字項目<br>英数字項目<br>英数字編集項目<br>数字編集項目<br>日本語項目<br>日本語編集項目※4※6<br>集団項目※8 | 1. COBOL のデータ領域に byte 配列の値を無変換で設定する。Byte 配列の長さが長い場合は切り捨てられ，短い場合は null(0x00)が埋められる。 | 1. COBOL のデータ領域を無変換で byte 配列にする。 | バイト配列データ<br>(byte[]) |

| COBOL データ項目 | setter 変換規則 | getter 変換規則 | Java データ属性 |
|---|---|---|---|
| 英字項目<br>英数字項目<br>英数字編集項目<br>数字編集項目<br>日本語項目<br>日本語編集項目※4※6 | 次のデータに変換する。<br>  <br><br>全体長は可変で,<br>1. 長さを表す int(setter で指定された長さを設定)<br>2. データ自身を表す byte 配列<br>(配列長は，setter で指定された長さ)<br>(setter で指定されたデータを無変換で設定)<br>である。 | 1. 次に示すデータから byte 配列に変換する。<br>  <br><br>2. 先頭 4 バイトの長さ(1.)分の byte 配列に，データ自身(2.)を無変換でコピーする。<br>(*) 長さは，byte 配列の length プロパティで取得する。 | 可変長データ<br>(byte[]) |
| アドレスデータ項目 | 次のデータに変換する。<br>  <br><br>全体長は可変で,<br>1. 長さを表す int(setter で指定された長さを設定)<br>2. データ自身を表す byte 配列<br>(配列長は，setter で指定された長さ)<br>(setter で指定されたデータを無変換で設定)<br>である。 | 1. 次に示すデータから byte 配列に変換する。<br>  <br><br>2. 先頭 4 バイトの長さ(1.)分の byte 配列に，データ自身(2.)を無変換でコピーする。<br>(*) 長さは，byte 配列の length プロパティで取得する。 | アドレスデータ<br>(byte[]) |
| 単精度内部浮動項目 | 1. endian オプション※2 に従ってデータの並び順を変更する。 | 1. endian オプション※2 に従ってデータの並び順を変更する。 | 単精度データ<br>(Float) |
| 倍精度内部浮動項目 | | | 倍精度データ<br>(Double) |
| 1～4 けたの小数を含まない 2 進項目 | 1. endian オプション※2 または comp5 オプション※7 に従ってデータの並び順を変更する。 | 1. endian オプション※2 または comp5 オプション※7 に従ってデータの並び順を変更する。 | Short データ<br>(Short) |
| 5～9 けたの小数を含まない 2 進項目 | | | Integer データ<br>(Integer) |
| 10～18 けたの小数を含まない 2 進項目 | | | Long データ(Long) |
| 小数を含む 2 進項目 | 1. 小数点位置を合わせた 2 進データを作成する。<br>2. endian オプション※2 に従ってデータの並び順を変更する。 | 1. endian オプション※2 に従ってデータの並び順を変更する。<br>2. 小数点位置を合わせた数字データを作成する。 | 10 進データ<br>(BigDecimal) |

| COBOL データ項目 | setter 変換規則 | getter 変換規則 | Java データ属性 |
|---|---|---|---|
| 外部 10 進 | 1. 符号を除いて，小数点位置を合わせた，数字データを作成する。<br>2. 独立符号であれば，空白／＋／－のどれかを文字データに付加する。そうでなければ，符号付きで負の値の場合は，符号を含むバイト位置の先頭 4 ビットを 0x7X とする。<br>(*) 数値のけた数が項目のけた数よりも大きい場合，けた落ちが発生する。整数けたでは左端の数値から，小数けたでは右端の数値からけた落ちが発生する。 | 1. 独立符号であれば，空白／＋／－のどれかから符号を決定する。そうでなければ，符号を含むバイト位置の先頭 4 ビットが 0x3X であれば正符号と，0x7X であれば負符号とし，4 ビットを 0x3X とする。<br>2. 符号も含めて，小数点位置を合わせた，数字データを作成する。 | 10 進データ (BigDecimal) |
| 内部 10 進 | 1. 符号を除いて，小数点位置を合わせた，内部 10 進項目の形式のデータを作成する。<br>(*) 数値のけた数が項目のけた数よりも大きい場合，けた落ちが発生する。整数けたでは左端の数値から，小数けたでは右端の数値からけた落ちが発生する。<br>(*) 内部 10 進項目は，1 バイトで数字 2 けたを表現します。各けたは 0～9 の数字で，最右端の 4 ビットで符号を表現する。<br>確保される領域長は，けた数から算出できる。<br>領域長=( けた数 / 2 ) + 1<br>(小数点以下は切り捨て)<br>偶数けた数の場合は，先頭の 4 ビットは 0 となる。<br>(*) 符号は次のように表現する。<br>• 正符号：0xC (ビット値=1100)<br>• 負符号：0xD (ビット値=1101)<br>• 符号無：0xF (ビット値=1111)<br>符号なし内部 10 進項目に負の値を設定しようとすると，エラーになる。 | 1. 符号部分を取り出し，符号を決定する。符号付き内部 10 進項目の場合は，符号部分が‘+’／‘−’以外の場合はすべて‘+’とし，符号なし内部 10 進項目の場合は，常に‘+’とする。<br>2. 符号以外の数値を数字データにし，符号を付加する。<br>(*) 偶数けた数の場合は，先頭の 4 ビットは使用しない。<br>(*) 数字部分の値が 0～9 以外の場合，値は保証しない。 | 10 進データ (BigDecimal) |

注※1

encode オプションについては「5.4.4　encode オプション」を参照してください。

注※2

endian オプションについては「5.4.5　endian オプション」を参照してください。

注※3

japanese オプションについては「5.4.6　japanese オプション」を参照してください。

注※4

「COBOL アクセス用 Bean 生成ツールーステップ 1/3」画面で「日本語項目を英数字項目として扱う」チェックボックスをオンにして COBOL アクセス用 Bean を生成した場合，日本語項目および日本語編集項目は英数字項目と同じ扱いとなります。

注※5

「COBOL アクセス用 Bean 生成ツールーステップ 1/3」画面で「日本語項目を英数字項目として扱う。」チェックボックスをオフにして COBOL アクセス用 Bean を生成した場合，日本語項目および日本語編集項目は日本語項目として扱います。

注※6

「COBOL アクセス用 Bean 生成ツールーステップ 2/3」画面でデータ属性フィールドを「バイト配列データ(byte[])」または「可変長データ(byte[])」に変更した場合，バイト配列データまたは可変長データとして扱います。

注※7

COBOL アクセス用 Bean 生成時に COMP-5 として扱うことを指定した 2 進項目の場合，comp5 オプションに従います。comp5 オプションについては，「5.4.2　comp5 オプション」を参照してください。

注※8

COBOL アクセス用 Bean 生成時に「集団項目のバイト配列アクセスを使用する。」チェックボックスをオンにした場合，バイト配列データとして扱います。

注※9

unicode オプションについては「5.4.8　unicode オプション」を参照してください。

## 8.2.2　CBLAccess クラス

CBLAccess クラスのメソッドについて説明します。

## (1)　COBOL 呼び出し

### (a)　JavaBean 対応 callCOBOL メソッド

[callCOBOL メソッド]

```
public int callCOBOL()
              throws J2CBException
```

**パラメタ**：なし

**戻り値**：COBOL エラーレベル

0：正常

-1：COBOL エラー以外

1：COBOL 実行時エラー　I レベルエラーメッセージ出力

2：COBOL 実行時エラー　W レベルエラーメッセージ出力

**例外**：J2CBException - 例外情報の取得

詳細は「8.3　J2CBException ユーザインタフェース API」を参照してください。

### (b)　EJB 対応 callCOBOL メソッド

[callCOBOL メソッド]

```
public int callCOBOL()
            throws RemoteException,Exception
```

**パラメタ**：なし

**戻り値**：COBOL エラーレベル

0：正常

-1：COBOL エラー以外

1：COBOL 実行時エラー　I レベルエラーメッセージ出力

2：COBOL 実行時エラー　W レベルエラーメッセージ出力

**例外**：RemoteException，Exception - 例外情報の取得

詳細は，「8.4　EJB 用 Exception ユーザインタフェース API」を参照してください。

## (2)　引数データの設定

### (a)　JavaBean 対応 setXxx メソッド

[setXxx メソッド]

```
public void setXxx(Object val [ , int dim1 { , int dim2 } ... ] )
            throws J2CBException
```

- メソッド名 setXxx の Xxx は，設定するデータ項目のデータ名を表します。

例）02 WK-DATANAME PIC X(10). と定義されたデータが引数にある場合，

```
public void setWk_dataname(Object val) throws J2CBException
```

という setter になります。

setter を呼び出してデータ項目に値を設定する場合，データ項目に対応するデータ属性のパラメタ val を指定してください。データ項目の対応は，「表 2-2　各データ項目に対応する表示文字列」を参照してください。

- [ , int dim1 { , int dim2 } ... ]は，OCCURS 句による繰り返しがあることを表します。
  Dim は次元の個数分作成されます。

例）2 次元の文字データが引数にある場合，

```
public void setWk_dataname(Object val, int dim1, int dim2)
                throws J2CBException
```

という setter になります。

**パラメタ：**

val：設定するデータオブジェクト

dim1,dim2...：(添字が必要な場合の)各次元の添字

**戻り値：**なし

**例外：**J2CBException - 例外情報の取得

詳細は「8.3　J2CBException ユーザインタフェース API」を参照してください。

## (b)　EJB 対応 setXxx メソッド

[setXxx メソッド]

```
public void setXxx(String※ val [ , int dim1 { , int dim2 } ... ] )
            throws RemoteException,Exception
```

注※　マッピングする Java でのデータ属性で生成される。

- メソッド名 setXxx の Xxx は，設定するデータ項目のデータ名を表します。また，[ , int dim1 { , int dim2 } ... ]は，OCCURS 句による繰り返しがあることを表します（[setXxx メソッド]の例を参照してください)。

**パラメタ：**

val：設定するデータオブジェクト

dim1,dim2...：(添字が必要な場合の)各次元の添字

**戻り値：**なし

**例外：**RemoteException，Exception - 例外情報の取得

詳細は，「8.4　EJB 用 Exception ユーザインタフェース API」を参照してください。

## (c)　JavaBean 対応 setXxx メソッド(可変長データおよびアドレスデータ用)

[setXxx メソッド]

```
public void setXxx(Object val, int len)
            throws J2CBException
```

- メソッド名 setXxx の Xxx は，設定するデータ項目のデータ名を表します（[setXxx メソッド]の例を参照してください)。

**パラメタ：**

val：設定するデータオブジェクト

len：データ長

（補足）

- データ長 len に指定できる値は 0 以上です。0 未満を指定した場合は，0 を仮定します。さらに可変長データでデータ長 len が(全体長-4)を超える場合は，(全体長-4)を仮定します。
- 設定するデータ val の長さがデータ長 len より短い場合，データ項目に val をすべて設定します（val の長さ以降の領域の値は不定です）。
- 設定するデータ val の長さがデータ長 len より長い場合，データ項目に val を len の長さ分だけ設定します。

**戻り値：**なし

**例外：**J2CBException - 例外情報の取得

詳細は「8.3　J2CBException ユーザインタフェース API」を参照してください。

### (d)　EJB 対応 setXxx メソッド(可変長データおよびアドレスデータ)

[setXxx メソッド]

```
public void setXxx(byte[] val, int len)
            throws RemoteException,Exception
```

- メソッド名 setXxx の Xxx は，設定するデータ項目のデータ名を表します（[setXxx メソッド]の例を参照してください）。

**パラメタ：**

val：設定するデータオブジェクト

len：データ長

（補足）

- データ長 len に指定できる値は 0 以上です。0 未満を指定した場合は，0 を仮定します。さらに可変長データでデータ長 len が(全体長-4)を超える場合は，(全体長-4)を仮定します。
- 設定するデータ val の長さがデータ長 len より短い場合，データ項目に val をすべて設定します（val の長さ以降の領域の値は不定です）。
- 設定するデータ val の長さがデータ長 len より長い場合，データ項目に val を len の長さ分だけ設定します。

**戻り値：**なし

**例外：**RemoteException，Exception - 例外情報の取得

詳細は，「8.4　EJB 用 Exception ユーザインタフェース API」を参照してください。

## (3)　引数データの取得

### (a)　JavaBean 対応 getXxx メソッド

[getXxx メソッド]

```
public Object getXxx( [ int dim1 { , int dim2 } ... ] )
            throws J2CBException
```

- メソッド名 getXxx の Xxx は，取得するデータ項目のデータ名を表します。また，[ int dim1 { , int dim2 } ... ]は，OCCURS 句による繰り返しがあることを表します（[setXxx メソッド]の例を参照してください)。

**パラメタ：**

dim1,dim2...：(添字が必要な場合の)各次元の添字

**戻り値：**

取得したデータオブジェクト

**例外：**J2CBException - 例外情報の取得

詳細は「8.3　J2CBException ユーザインタフェース API」を参照してください。

### (b)　EJB 対応 getXxx メソッド

[getXxx メソッド]

```
public String※ getXxx( [ int dim1 { , int dim2 } ... ] )
            throws RemoteException,Exception
```

注※　マッピングする Java でのデータ属性で生成される。

- メソッド名 getXxx の Xxx は，取得するデータ項目のデータ名を表します。また，[ int dim1 { , int dim2 } ... ]は，OCCURS 句による繰り返しがあることを表します（[setXxx メソッド]の例を参照してください)。

**パラメタ：**

dim1,dim2...：(添字が必要な場合の)各次元の添字

**戻り値：**

取得したデータオブジェクト

**例外：**RemoteException，Exception - 例外情報の取得

詳細は，「8.4　EJB 用 Exception ユーザインタフェース API」を参照してください。

## 8.3　J2CBException ユーザインタフェース API

J2CBException ユーザインタフェースの API につい説明します。

### 8.3.1　J2CBException クラスのメソッド

J2CBException クラスで提供するメソッドの一覧を表 8-4 に示します。

表 8-4　J2CBException クラスの提供メソッド一覧

| 項番 | メソッド名 | 機能 |
|---|---|---|
| 1 | getCblErrorCode | COBOL UAP の呼び出しで発生した例外コードを取得する。 |
| 2 | getCblMessageID | 該当する COBOL 実行時エラーメッセージ ID を取得する。 |
| 3 | getDetailMessage | COBOL 実行時エラーメッセージ ID に対応するメッセージを取得する。 |
| 4 | getErrorCode | COBOL アクセスで発生した例外情報コードを取得する。 |
| 5 | getMessage | COBOL アクセスで発生した例外情報コードに対するメッセージ文字列を取得する。 |
| 6 | getName | COBOL アクセスで発生した例外名を取得する。 |

J2CBException クラスのメソッドについて説明します。

これらの例外用メソッドを使用して，COBOL 側で発生したエラー情報などを取得します。プログラム中からこれらのメソッドを使用し，例外情報を参照できます。

#### (1)　例外コードの取得

[getCblErrorCode メソッド]

```
public java.lang.String getCblErrorCode()
```

**戻り値**：null／x00002135（8501）／そのほかのコード（例外種別コード）

null：COBOL アクセスで異常終了した。

x00002135（8501）：実行時デバッグ機能オプション※1 またはテストデバッグ機能オプション※2 を指定した COBOL UAP で異常終了した場合，または COBOL 実行時エラーが発生して異常終了した場合，当コードを返します。getCblMessageID メソッドなどからエラー情報を取得して原因を調査してください。

**そのほかのコード**：実行時デバッグ機能オプション※1 またはテストデバッグ機能オプション※2 の指定がない COBOL UAP で例外が発生した場合，システムの例外種別コード値を返します。

注※1　実行時デバッグ機能オプション

- -DebugInf
- -DebugInf,Trace

- -DebugCompati
- -DebugData
- -DebugRange

注※2　テストデバッグ機能オプション

- -TDInf
- -CVInf

## (2)　実行時エラーメッセージ ID 取得

[getCblMessageID メソッド]

```
public java.lang.String getCblMessageID()
```

**戻り値：**

KCCC のプリフィクスなしの番号の文字列

COBOL 実行時エラーメッセージ ID が存在しない場合は，null

## (3)　該当エラーメッセージの取得

[getDetailMessage メソッド]

```
public java.lang.String getDetailMessage()
```

**戻り値：**該当する COBOL 実行時エラーメッセージ

環境変数 CBL_SYSERR で指定したファイルに出力された実行時エラーメッセージの中から，該当する例外オブジェクトに格納された COBOL 実行時エラーメッセージ ID に対応するメッセージを取得します（COBOL 実行時エラーメッセージ ID が存在しない場合は，null)。

## (4)　COBOL アクセスの例外情報コードの取得

[getErrorCode メソッド]

```
public java.lang.String getErrorCode()
```

**戻り値：**例外情報コード

COBOL アクセスで発生した例外情報コードを取得します。

例外情報コードは，「付録 E　例外情報コード一覧」の「表 E-1　例外名一覧（JavaBean/EJB 共通)」と「表 E-2　JavaBean 用エラー発生場所（メソッド）コード」を参考にしてください。また，これらのコードは 16 進コードです。

なお，例外情報コードは，先頭に"J2CB"が付加されて次のようになります。

（例）J2CB6052002 の場合

J2CB　：プリフィクス

605　：エラー発生場所（メソッド）コード（3 けた）

2002　：エラー要因コード（4 けた）

## (5)　COBOL アクセスのメッセージ取得

[getMessage メソッド]

```
public java.lang.String getMessage()
```

**戻り値：**メッセージ文字列

COBOL アクセスで発生したエラー発生場所（メソッド）とエラー要因をメッセージ文字列で返します。

null：情報なし

**オーバーライド：**クラス java.lang.Throwable 内の getMessage

## (6)　例外名の取得

[getName メソッド]

```
public java.lang.String getName()
```

**戻り値：**発生した例外名を返します。

詳細は，「8.3.2　提供クラスのメソッドで発生する例外」の「表 8-5　ユーザがキャッチできる提供クラスのメソッドで発生する例外」に示すユーザがキャッチできる提供クラスのメソッドで発生する例外を参照してください。

# 8.3.2　提供クラスのメソッドで発生する例外

ユーザがキャッチできる提供クラスのメソッドで発生する例外を表 8-5 に示します。

表 8-5　ユーザがキャッチできる提供クラスのメソッドで発生する例外

| 例外名 | 意味 | エラー要因コードの候補 |
|---|---|---|
| SYS_ERR | システムエラーが発生しました。 | 0000 番台 |
| J_ENV_ERR | 該当クラス，または，コンストラクタ，メソッドが見つかりません。 | 1000 番台 |
| INVALID_TYPE | 型情報が不正です。型不正が発生したメソッド名を参照して型を確認してください。 | 1100 番台 |
| INVALID_ARG | 引数が不正です。引数不正が発生したメソッド名を参照して引数を確認してください。 | 2000 番台 |
| UNRECOVERABLE | COBOL アクセス内で回復不能なエラーが発生しました。 | 4000 番台<br>5000 番台 |

| 例外名 | 意味 | エラー要因コードの候補 |
| --- | --- | --- |
| UNKNOWN | 予期しないエラーが発生しました。 | 9999 |

## 8.4　EJB 用 Exception ユーザインタフェース API

EJB 用 Exception ユーザインタフェース API について説明します。

### 8.4.1　ユーザインタフェース API

Exception クラスについて説明します。

COBOL アクセスを Enterprise Bean として使用する場合のエラーは，Exception クラスで取得してください。エラー情報を取得する場合，Exception クラスのメソッド getMessage()に例外情報の文字列を返します。

COBOL アクセスでは，エラー情報を解析するためのクラス(J2CBErrorUtil)を用意しています。
J2CBErrorUtil クラスのメソッドを使用して，COBOL で発生したエラー情報などを取得します。

プログラム中からこれらのメソッドを使用し，取得したい例外情報を参照できます。

[ J2CBErrorUtil クラス ]

```
public class J2CBErrorUtil implements java.io.Serializable
```

**機能：**エラー情報の解析

**コンストラクタの詳細：**public J2CBErrorUtil (String s)

**パラメタ：**s－例外情報の文字列

J2CBErrorUtil クラスを記述した Servlet の例を次に示します。

```
try {
  remoteobj=homeobj.create();
} catch (Exception e) {
  J2CBErrorUtil wkobj=new J2CBErrorUtil(e.getMessage());
  out.println("Create時エラー発生 :");
  out.println("<p>getEJBCblErrorCode=" + wkobj.getEJBCblErrorCode() + "</p>");
  out.println("<p>getEJBCblMessageID=" + wkobj.getEJBCblMessageID() + "</p>");
  out.println("<p>getEJBDetailMessage=" + wkobj.getEJBDetailMessage() + "</p>");
  out.println("<p>getEJBErrorCode=" + wkobj.getEJBErrorCode() + "</p>");
  out.println("<p>getEJBMessage=" + wkobj.getEJBMessage() + "</p>");
  out.println("<p>getEJBName=" + wkobj.getEJBName() + "</p>");
  e.printStackTrace();
  return;
}
```

J2CBErrorUtil クラスで提供するメソッドの一覧を表 8-6 に示します。

表 8-6　J2CBErrorUtil クラスの提供メソッド一覧

| 項番 | メソッド名 | 機能 |
|---|---|---|
| 1 | getEJBCblErrorCode | COBOL UAP の呼び出しで発生した例外コードを取得する。 |
| 2 | getEJBCblMessageID | 該当する COBOL 実行時エラーメッセージ ID を取得する。 |
| 3 | getEJBDetailMessage | COBOL 実行時エラーメッセージ ID に対応するメッセージを取得する。 |
| 4 | getEJBErrorCode | COBOL アクセスで発生した例外情報コードを取得する。 |
| 5 | getEJBMessage | COBOL アクセスで発生した例外情報コードに対するメッセージ文字列を取得する。 |
| 6 | getEJBName | COBOL アクセスで発生した例外名を取得する。 |

J2CBErrorUtil クラスのメソッドについて説明します。

## (1)　例外コードの取得

[ getEJBCblErrorCode メソッド ]

```
public java.lang.String getEJBCblErrorCode()
```

**戻り値：**null／x00002135（8501）／そのほかのコード（例外種別コード）

**null：**COBOL アクセスで異常終了した。

**x00002135（8501）：**実行時デバッグ機能オプション※1 またはテストデバッグ機能オプション※2 を指定した COBOL UAP で異常終了した場合，または COBOL 実行時エラーが発生して異常終了した場合，当コードを返します。getCblMessageID メソッドなどからエラー情報を取得して原因を調査してください。

**そのほかのコード：**実行時デバッグ機能オプション※1 またはテストデバッグ機能オプション※2 の指定がない COBOL UAP で例外が発生した場合，システムの例外種別コード値を返します。

注※1　実行時デバッグ機能オプション

- -DebugInf
- -DebugInf,Trace
- -DebugCompati
- -DebugData
- -DebugRange

注※2　テストデバッグ機能オプション

- -TDInf
- -CVInf

## (2)　実行時エラーメッセージ ID 取得

[ getEJBCblMessageID メソッド ]

```
public java.lang.String getEJBCblMessageID()
```

**戻り値：**KCCC のプリフィクスなしの番号の文字列

null：情報なし

## (3)　該当エラーメッセージの取得

[ getEJBDetailMessage メソッド ]

```
public java.lang.String getEJBDetailMessage()
```

**戻り値：**該当する COBOL 実行時エラーメッセージ

環境変数 CBL_SYSERR で指定したファイルに出力された実行時エラーメッセージの中から，該当する例外オブジェクトに格納された COBOL 実行時エラーメッセージ ID に対応するメッセージを取得します。

null：情報なし

## (4)　COBOL アクセスの例外情報コードの取得

[ getEJBErrorCode メソッド ]

```
public java.lang.String getEJBErrorCode()
```

**戻り値：**例外情報コード

COBOL アクセスで発生した例外情報コードを取得します。

例外情報コードは，先頭に”J2CB”が付加された 11 けたのコード値です。

例外情報コードは，「付録 E　例外情報コード一覧」の「表 E-2　JavaBean 用エラー発生場所（メソッド）コード」と「表 E-3　EJB 用エラー発生場所（メソッド）コード」を参考にしてください。

また，COBOL アクセス以外で発生した例外情報コードは，先頭に"ETC:"が付加されます。

null：情報なし

## (5)　COBOL アクセスのメッセージ取得

[ getEJBMessage メソッド ]

```
public java.lang.String getEJBMessage()
```

**戻り値：**メッセージ文字列

COBOL アクセスで発生したエラー発生場所（メソッド）とエラー要因をメッセージ文字列で返します。

null：情報なし

オーバーライド：クラス java.lang.Throwable 内の getMessage

### (6)　例外名の取得

[ getEJBName メソッド ]

```
public java.lang.String getEJBName()
```

戻り値：発生した例外名を返す。

null：情報なし

詳細は「表 8-7　ユーザがキャッチできる例外」を参照してください。

## 8.4.2　ユーザがキャッチできる例外

ユーザがキャッチできる例外を表 8-7 に示します。

表 8-7　ユーザがキャッチできる例外

| 例外名 | 意味 | エラー要因コードの候補 |
|---|---|---|
| SYS_ERR | システムエラーが発生しました。 | 0000 番台 |
| J_ENV_ERR | 該当クラス，または，コンストラクタ，メソッドが見つかりません。 | 1000 番台 |
| INVALID_TYPE | 型情報が不正です。型不正が発生したメソッド名を参照して型を確認してください。 | 1100 番台 |
| INVALID_ARG | 引数が不正です。引数不正が発生したメソッド名を参照して引数を確認してください。 | 2000 番台 |
| UNRECOVERABLE | COBOL アクセス内で回復不能なエラーが発生しました。 | 4000 番台<br>5000 番台 |
| UNKNOWN | 予期しないエラーが発生しました。 | 9999 |

# 付録

# 付録 A　注意事項／制限事項

Cosminexus 連携機能を使用する際の注意事項，および制限事項について説明します。

## 付録 A.1　COBOL アクセス用 Bean 生成時の COBOL 定義注意事項／制限事項

COBOL アクセス用 Bean 生成時は，COBOL UAP 引数定義を別ファイルにして，COBOL データ定義として指定する必要があります。この COBOL データ定義で，変更が必要な項目を表 A-1 に示します。

なお，表 A-1 には制限事項も含めて記述しています。制限事項に該当する場合は，COBOL UAP の引数を変更してください。

### 表 A-1　COBOL アクセス用 Bean 生成時の COBOL 定義注意事項／制限事項一覧

| 項番 | COBOL 定義 | 回避方法詳細 |
|---|---|---|
| 1 | EXTERNAL 句 | (1)を参照 |
| 2 | GLOBAL 句 | (2)を参照 |
| 3 | BLANK WHEN ZERO 句 | (3)を参照 |
| 4 | SYNCHRONIZED 句 | (4)を参照 |
| 5 | JUSTIFIED 句 | (5)を参照 |
| 6 | VALUE 句 | (6)を参照 |
| 7 | RENAMES 句 | (7)を参照 |
| 8 | 条件名 | (8)を参照 |
| 9 | 外部浮動小数点項目 | (9)を参照 |
| 10 | 外部ブール項目 | (10)を参照 |
| 11 | 内部ブール項目 | (11)を参照 |
| 12 | OCCURS 句（OCCURS 整数 TIMES 以外） | (12)を参照<br>ただし，反復回数が可変の場合，COBOL データ定義によっては回避方法がありませんので COBOL UAP 引数定義を変更してください。 |
| 13 | 位取り | (13)を参照 |
| 14 | 行内注記（*>） | (14)を参照 |
| 15 | 修飾 | (15)を参照 |
| 16 | 登録集原文中の COPY 文(COPY { 原文名 \| 原文名定数 } .以外) | (16)を参照 |
| 17 | SAME AS 句 | (17)を参照 |

| 項番 | COBOL 定義 | 回避方法詳細 |
|---|---|---|
| 18 | TYPE 句および TYPEDEF 句 | (18)を参照 |
| 19 | 翻訳指令 | (19)を参照 |
| 20 | PICTURE 句（数字項目のけた拡張機能） | (20)を参照 |
| 21 | 動的長基本項目 | (21)を参照 |
| 22 | 日本語集団項目 | (22)を参照 |

## (1) EXTERNAL 句

LINKAGE SECTION で指定できない EXTERNAL 句は削除してください。

Java マッピングデータ属性は，EXTERNAL 句の指定有無にかかわらず，EXTERNAL 句がない場合と同じです。

## (2) GLOBAL 句

GLOBAL 句は削除してください。

Java マッピングデータ属性は，GLOBAL 句の指定有無にかかわらず，GLOBAL 句がない場合と同じです。

## (3) BLANK WHEN ZERO 句

数字項目の編集を指定する BLANK WHEN ZERO 句は削除してください。

また，数字項目に指定された BLANK WHEN ZERO 句である場合は，同じ領域長の英数字項目に変更してください。

操作は数字編集項目と同様に，編集した値を設定し，受け取ったデータを Java プログラムで数値に変換して使用します。

**[ 例 ]**

＜変更前＞

```
05  HTC-OUTDATA                 PIC 9(6)
      USAGE DISPLAY  BLANK WHEN ZERO.
```

＜変更後＞

```
05  HTC-OUTDATA                 PIC X(6).
      USAGE DISPLAY  BLANK WHEN ZERO.
```

## (4) SYNCHRONIZED 句

計算機記憶の固有の境界に従った，基本項目の配置を指定する SYNCHRONIZED 句は削除してください。また，SYNCHRONIZED 句によって確保されていた暗黙の FILLER を明示的に定義してください。暗黙の FILLER については，マニュアル「COBOL2002　言語　標準仕様編」の SYNCHRONIZED 句の記述個所を参照してください。

[ 例 ]

＜変更前＞

```
01  HTC-DATA.
  03  HTC-INPUT-HEADER  SYNC.
    05  HTC-INPUT-COMMON1          PIC X(3).
    05  HTC-INPUT-COMMON2          PIC S9(4)  USAGE COMP.
```

＜変更後＞

```
01  HTC-DATA.
  03  HTC-INPUT-HEADER  ~~SYNC.~~
    05  HTC-INPUT-COMMON1            PIC X(3).
    05  FILLER                       PIC X.
    05  HTC-INPUT-COMMON2            PIC S9(4)  USAGE COMP.
```

## (5) JUSTIFIED 句

けた寄せを表す JUSTIFIED 句は削除してください。

Java マッピングデータ属性は，JUSTIFIED 句の指定有無にかかわらず，JUSTIFIED 句がない場合と同じです。

## (6) VALUE 句

初期値や条件名に対応する値を指定する VALUE 句は削除してください。

Java マッピングデータ属性は，VALUE 句の指定有無にかかわらず，VALUE 句がない場合と同じです。

## (7) RENAMES 句

基本項目を組み合わせて新たな集団を作る（再命名項目）RENAMES 句は指定できないので，66 レベル項目から削除してください。

[ 例 ]

＜変更前＞

```
01  HTC-DATA.
 03  HTC-INPUT-HEADER.
    05  HTC-INPUT-COMMON1            PIC X(3).
    05  HTC-INPUT-COMMON2            PIC X(2).
  66  HTC-RENAME  RENAMES HTC-INPUT-COMMON2.
```

＜変更後＞

```
01  HTC-DATA.
  03  HTC-INPUT-HEADER.
    05  HTC-INPUT-COMMON1            PIC X(3).
    05  HTC-INPUT-COMMON2            PIC X(2).
  ~~66  HTC-RENAME  RENAMES HTC-INPUT-COMMON2.~~
```

## (8)　条件名

条件名を指定する 88 レベルは指定できないので，88 レベル項目を削除してください。Java マッピングデータ属性は，条件名の指定有無にかかわらず，条件名がない場合と同じです。

## (9)　外部浮動小数点項目

外部浮動小数点項目は，英数字項目に変更してください。英数字項目のけた数は，外部浮動小数点項目の形式から算出してください。Java プログラムでは，String データを浮動小数点データに変換して使用してください。

[ 例 ]

＜変更前＞

```
  05  HTC-OUTDATA1          PIC +9V999E+99  USAGE DISPLAY.
  05  HTC-OUTDATA2          PIC +9.999E+99  USAGE DISPLAY.
```

＜変更後＞

```
  05  HTC-OUTDATA1          PIC X(9).
  05  HTC-OUTDATA2          PIC X(10).
```

## (10)　外部ブール項目

外部ブール項目は，英数字項目に変更してください。

外部ブール項目の場合，英数字項目のけた数は，ブール項目のけた数と同じです。

また，Java プログラムでは，文字'0'と'1'を使用します。

[ 例 ]

＜変更前＞

```
  05  HTC-OUTDATA1          PIC 1(8)  USAGE DISPLAY.
```

＜変更後＞

```
  05  HTC-OUTDATA1          PIC X(8).
```

## (11) 内部ブール項目

内部ブール項目は，バイト配列にマッピングした英数字項目に変更してください。英数字項目のけた数は，内部ブール項目が占有するバイト数と同じです。また，Java プログラムでは，バイト配列をビット操作して使用してください。なお，内部ブール項目で遊びビットが取られる場合は，Java プログラムで遊びビットを考慮してください。

[ 例 ]

＜変更前＞

```
  05  HTC-OUTDATA2          PIC 1(8)  USAGE BIT.
```

＜変更後＞

```
  05  HTC-OUTDATA2          PIC X(1).
```

## (12) OCCURS 句（OCCURS 整数 TIMES 以外）

「OCCURS 整数 TIMES」以外の指定は，削除してください。

Java マッピングデータ属性は，INDEXED 指定，KEY 指定の有無にかかわらず，INDEXED 指定，KEY 指定がない場合と同じです。

DEPENDING ON 指定は繰り返し回数が可変であることを表します。

後続のデータ項目の開始位置が，繰り返し回数によって変わる場合は使用できません。項目の開始位置が，繰り返し回数に依存しない場合は，DEPENDING ON 指定を削除して使用してください。

[ 例 ]

＜変更前＞

```
01  HTC-DATA.
  03  HTC-INPUT-HEADER
        OCCURS 10 TIMES  INDEXED BY HTC-SOEJI.
    05  HTC-INPUT-COMMON1          PIC X(3).
    05  HTC-INPUT-COMMON2          PIC X(2).
```

＜変更後＞

```
01  HTC-DATA.
  03  HTC-INPUT-HEADER
        OCCURS 10 TIMES  INDEXED BY HTC-SOEJI.
    05  HTC-INPUT-COMMON1          PIC X(3).
    05  HTC-INPUT-COMMON2          PIC X(2).
```

## (13) 位取り

PICTURE 文字列に指定した文字'P'を削除してください。Java プログラムでは，想定した位取りを意識した演算を行う必要があります。

[ 例 ]

＜変更前＞

```
05  HTC-OUTDATA              PIC 9(4)PP.
```

＜変更後＞

```
05  HTC-OUTDATA              PIC 9(4)PP=
```

上記の例で，COBOL プログラムで HTC-OUTDATA に 12300 という値が入っている場合，Java プログラムで見る HTC-OUTDATA は 123 という値になります。

したがって，Java プログラムでこの値を使用する場合は，100 倍した値を使用する必要があります。

## (14) 行内注記（*>）

使用できません。削除してください。

## (15) 修飾

基本項目のデータ名は，Bean の setter／getter メソッド名称として使用されるので，修飾しなくても一意になっている必要があります。

一意となっていない場合は，「Bean 生成ツール」で別名を指定して，一意の名称にしてください。

## (16) 登録集原文中の COPY 文(COPY { 原文名 | 原文名定数 } .以外)

COPY 文で指定された COPY 原文を登録集原文に展開し，COPY 文を削除してください。

## (17) SAME AS 句

使用できません。削除してください。

## (18) TYPE 句および TYPEDEF 句

使用できません。削除してください。

## (19) 翻訳指令

使用できません。削除してください。

### (20) PICTURE 句（数字項目のけた拡張機能）

数字項目のけた拡張機能は使用できません。PICTURE 句に指定するけた数は 18 けた以下にしてください。

### (21) 動的長基本項目

DYNAMIC LENGTH 句による動的長基本項目は使用できません。削除してください。

### (22) 日本語集団項目

GROUP-USAGE 句による日本語集団項目は使用できません。削除してください。

## 付録 A.2 COBOL アクセス用 Bean および Servlet 作成時の留意する必要のある項目

COBOL アクセス用 Bean および Servlet を作成するときの注意事項と制限事項を表 A-2 に示します。

表 A–2 COBOL アクセス用 Bean および Servlet 作成時の注意事項／制限事項

| 項番 | 内容 |
|---|---|
| 1 | COBOL のデータ名が日本語（データ名に英小文字，数字，ハイフン（−），下線（__）以外が含まれている場合。ただし FILLER の変換後の名称は除く）の場合，半角英数字の別名を必ず指定しなければならない。Java プログラム中には日本語のデータ名ではなく，指定した別名を記述する。 |
| 2 | COBOL プログラム名称は大文字，小文字を非等価として扱われる。同じプログラム名やデータ名は大文字，小文字が正しく一致していなければ実行時にエラーとなる。 |
| 3 | COBOL のデータ項目名にハイフン（−）が含まれる場合，自動的に下線（_）に変換して表示され，対応するソース生成時のプロパティ名も下線（_）で生成される。 |
| 4 | COBOL のデータ項目名に英大文字が含まれる場合，自動的に英小文字に変換して表示され，対応するソース生成時のプロパティも英小文字で生成される。ただし，メソッドの場合，先頭の 1 文字だけを英大文字にして生成される。 |
| 5 | Servlet プログラムには，setter を使用してすべての引数の値を必ず設定しておかなければならない。 |

## 付録 A.3 COBOL プログラム作成時の注意事項（ライブラリファイル名）

COBOL プログラムをコンパイルしてライブラリファイルを作成する際には，提供しているライブラリとは異なるファイル名にしてください。

表 A–3 ライブラリファイル名の一覧

| OS | 開発環境で使用しているファイル | 実行環境で使用しているファイル |
|---|---|---|
| Windows | j2cb2kpars.dll<br>j2cb2kwJNI.dll | j2cb2krt.dll |

| OS | 開発環境で使用しているファイル | 実行環境で使用しているファイル |
|---|---|---|
| Linux | − | libj2cb2krt.so |
| AIX(64) | − | libj2cb2krt64.a |

(凡例)

−：該当しない

## 付録 A.4　Windows OS 固有の注意事項

- Cosminexus 連携機能の開発環境は標準権限で実行してください。

  なお，プログラムを管理者権限で実行する場合の一般的な注意事項の詳細については，マニュアル「COBOL2002　ユーザーズガイド」を参照してください。

- 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。

- COBOL アクセス用 Bean 生成ツールで生成ファイルの出力先として，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。

- COBOL アクセス用 Bean 生成ツールでは，設定された Java ルートフォルダ情報および Java VM メモリサイズ情報を，COBOL アクセス用 Bean 生成ツール環境設定ファイル（j2cb2kw.ini）に格納しています。

  COBOL アクセス用 Bean 生成ツール環境設定ファイル（j2cb2kw.ini）の保存先は次のとおりです。

  Windows(x86)の場合

  ```
  環境変数ALLUSERSPROFILEが示すフォルダ¥Hitachi¥COBOL2002¥bin
  ```

  Windows(x64)の場合

  ```
  環境変数ALLUSERSPROFILEが示すフォルダ¥Hitachi¥COBOL2002 64bit¥bin
  ```

  環境変数 ALLUSERSPROFILE で指定されたフォルダに作成するファイルは，デフォルトではファイルを作成したユーザだけが更新できます。複数のユーザで更新したい場合は，該当ファイルごとにアクセス権限を設定する，管理者権限で更新するなど，対処してください。

## 付録 A.5　Linux で使用する場合の注意事項

- Linux では，UTF-8 環境で使用してください。

- 日本語項目および日本語編集項目を使用する場合は，unicode オプションを必ず指定してください。

  unicode オプション未指定時には，日本語項目データはデフォルトロケールである UTF-8 に変換されるため，Java と COBOL 間で日本語項目データを正しく変換できません。

日本語項目データを受け渡したい場合は，unicode オプションに big※1 または little※2 を指定してください。unicode オプションに big または little を指定する場合は，COBOL2002 の実行時環境変数である環境変数 CBLUNIENDIAN の指定と合わせてください。環境変数 CBLUNIENDIAN の指定と合っていない場合には，Java と COBOL 間で日本語項目データを正しく変換できません。

注※1

UTF-16BE 変換することを意味します。

注※2

UTF-16LE 変換することを意味します。

- 日本語項目および日本語編集項目を使用する場合は，COBOL アクセス用 Bean を生成する際に，「日本語項目を英数字項目として扱う。」チェックボックスをオフにしてください。

  チェックボックスをオンにした場合には，日本語項目データは英数字項目として扱われ UTF-8 変換されるため，COBOL2002 での格納データ形式 UTF-16 と異なり，Java と COBOL 間で日本語項目データを正しく変換できません。

# 付録 B　Cosminexus 連携機能，Cosminexus 連携機能の実行ライブラリで使用するファイル

Cosminexus 連携機能および Cosminexus 連携機能の実行ライブラリで使用するファイルの一覧を表 B-1 に示します。

表 B-1　Cosminexus 連携機能，Cosminexus 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 |
|---|---|---|---|---|
| UAP 引数定義ファイル | .cbl ほか※1 | Cosminexus 連携機能から呼び出したい COBOL UAP 引数を定義したファイル。 | － | Cosminexus 連携機能 |
| COBOL ソースファイル | .cbl ほか※2 | COBOL 原始プログラムを格納するファイル。 | － | コンパイラ |
| Cosminexus 上 Java 実行ファイル | .class | Cosminexus 連携機能を呼び出す Java 実行ファイル。 | － | Cosminexus 連携機能 |
| DLL ファイル | .dll | 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 | － |
| 共用ライブラリ | .a<br>または<br>.so※3 | 共用ライブラリを格納するファイル。 | | |
| COBOL アクセス用 Bean（Java ソースファイル） | .java | Cosminexus 連携機能が出力する COBOL アクセス用 Bean を格納するファイル。 | Cosminexus 連携機能 | － |
| EJB 関連 Java ソースファイル | .java | Cosminexus 連携機能が出力する EJB 用ホームインタフェース，リモートインタフェース，Enterprise Bean を格納するファイル。 | Cosminexus 連携機能 | － |
| ライブラリファイル（Windows の場合） | .lib | ライブラリを格納するファイル。ライブラリには，オブジェクトプログラムのライブラリである標準ライブラリと，DLL の関数情報を保持するインポートライブラリがある。 | リンカ，LIB コマンド※4 | － |
| オブジェクトファイル | .obj<br>または<br>.o※5 | 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 |
| デプロイ情報（DD ファイル） | .xml | Cosminexus 連携機能が出力する EJB 用デプロイ情報を格納するファイル。 | Cosminexus 連携機能 | － |

（凡例）

－：該当しない

注※1

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 COBOL 引数定義ファイルの場合
  .cbf，または環境変数 CBLFREE で指定した拡張子を除く，すべての拡張子
- 自由形式正書法で書かれた COBOL 引数定義ファイルの場合
  .cbf，または環境変数 CBLFREE で指定した拡張子

注※2

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，または環境変数 CBLFREE で指定した拡張子

注※3

OS に応じて次の拡張子を使用します。

- AIX の場合：.a
- Linux の場合：.so

注※4

LIB コマンドとは，ライブラリ管理ツール LIB の意味です。

注※5

OS に応じて次の拡張子を使用します。

- Windows の場合：.obj
- UNIX の場合：.o

# 付録 C　COBOL アクセス用 Bean の自動生成ソースイメージ

COBOL アクセス用 Bean 自動生成ソースイメージを登録集原文と対応させて説明します。

また，「Bean 生成ツール」で生成された Bean は Javadoc に対応しています。

## 付録 C.1　ソースイメージの生成例

## (1)　COBOL アクセス用 Bean の生成例

「Bean 生成ツール」で，「EJB 対応機能を有効にする。」にチェックしない場合の生成例です（そのほかのチェックボックスはデフォルト指定です）。

なお，Windows(x64)では，コメント行個所の「COBOL adapter for Cosminexus Version 2」は，「COBOL adapter for Cosminexus Version 2(64)」となります。

### (a)　OCCURS 句がない場合

COBOL UAP の引数に OCCURS 句がない場合の例を次に示します。

```
01 DATAA  PIC X(10).
01 DATAB.
 05 DATAB1 PIC S9(9).
 05 DATAB2.
  10 DATAB21 USAGE COMP-1.
  10 DATAB22 PIC S9(9)  USAGE COMP.
77 DATAC  USAGE COMP-2.
```

太字（色付き）は条件によって生成文字列が異なります。

また，太字（色付き）の変数名は「Bean 生成ツール」で別名を指定した場合，別名になります。

```
package パッケージ名;

import jp.co.hitachi_sk.j2cb.*;

public class クラス名 extends CBLAccess {
/*
 *Generated by COBOL adapter for Cosminexus Version 2 02-13
 *DO NOT EDIT THIS FILE
 *2013/09/30
 */
  //インデックスを定義
  private static final long serialVersionUID = 1347585438734L;
  private static final int dataaIndex=0;
  private static final int databIndex=1;
  private static final int datacIndex=2;
  private static boolean noload=true;
  private static int[] mySize=null;
```

```
  private static GroupAccess[] myGroupAccess=null;
  private static String[] myTypeInfo=null;
  private static int myLength=0;
/**
 *コンストラクタ
 */
  public クラス名() throws J2CBException {
    super();
    init();
    initialData();
    //データアクセスのための情報を設定
    setDllName("呼び出すライブラリ名");
    setFunctionName("呼び出すプログラム名");
  }
/**
 *初期化処理
 */
  private void init( ) throws J2CBException {
    //初期ロードだけ実行する
    if (noload) {
      init2();
    }
    else {
      setInformation(mySize, myGroupAccess, myTypeInfo, myLength);
    }
  }
/**
 *初期化処理2
 */
  private synchronized void init2( ) throws J2CBException {
    if (noload) {
      String[] typeTmp={
       "C(10)",
       "G,CT0900,G,F,SI0900",
       "D" };
      String[] sizeTmp={
       "10",
       "17",
       "8" };
      setType(typeTmp, sizeTmp);
      makeGroupAccess(dataaIndex,
       "01",
       "dataa",
       "0");
      makeGroupAccess(databIndex,
       "01,05,05,10,10",
       "datab,datab1,datab2,datab21,datab22",
       "0,0,9,9,13");
      makeGroupAccess(datacIndex,
       "77",
       "datac",
       "0");
      mySize=size;
      myGroupAccess=groupAccess;
      myTypeInfo=typeInfo;
      myLength=length;
      noload=false;
    }
```

```
    else {
      setInformation(mySize, myGroupAccess, myTypeInfo, myLength);
    }
  }
/**
 *dataa設定メソッド
 *@param data Stringオブジェクト
 */
  public void setDataa( Object data) throws J2CBException {
    setData(dataaIndex, "dataa", data);
  }
/**
 *datab1設定メソッド
 *@param data BigDecimalオブジェクト
 */
  public void setDatab1( Object data) throws J2CBException {
    setData(databIndex, "datab.datab1", data);
  }
/**
 *datab21設定メソッド
 *@param data Floatオブジェクト
 */
  public void setDatab21( Object data) throws J2CBException {
    setData(databIndex, "datab.datab2.datab21", data);
  }
/**
 *datab22設定メソッド
 *@param data Integerオブジェクト
 */
  public void setDatab22( Object data) throws J2CBException {
    setData(databIndex, "datab.datab2.datab22", data);
  }
/**
 *datac設定メソッド
 *@param data Doubleオブジェクト
 */
  public void setDatac( Object data) throws J2CBException {
    setData(datacIndex, "datac", data);
  }
/**
 *dataa取得メソッド
 *@return Stringオブジェクト
 */
  public Object getDataa( ) throws J2CBException {
    return getData(dataaIndex, "dataa");
  }
/**
 *datab1取得メソッド
 *@return BigDecimalオブジェクト
 */
  public Object getDatab1( ) throws J2CBException {
    return getData(databIndex, "datab.datab1");
  }
/**
 *datab21取得メソッド
 *@return Floatオブジェクト
 */
  public Object getDatab21( ) throws J2CBException {
```

```
    return getData(databIndex, "datab.datab2.datab21");
  }
/**
 *datab22取得メソッド
 *@return Integerオブジェクト
 */
  public Object getDatab22( ) throws J2CBException {
    return getData(databIndex, "datab.datab2.datab22");
  }
/**
 *datac取得メソッド
 *@return Doubleオブジェクト
 */
  public Object getDatac( ) throws J2CBException {
    return getData(datacIndex, "datac");
  }
}
```

## (b)　OCCURS 句がある場合

COBOL UAP の引数に OCCURS 句がある場合の例を次に示します。

```
01 G1.
 02 G2 OCCURS 10.
  03 B1 PIC X(50).
```

OCCURS ありの基本項目参照メソッド

```
/**
 *b1設定メソッド
 *@param data Stringオブジェクト
 *@param dim1 int
 */
  public void setB1( Object data, int dim1) throws J2CBException {
    String acsstr1="g1.g2[";
    String acsstr2="].b1";
    StringBuffer wkbuf;
    wkbuf=new StringBuffer( acsstr1 );
    wkbuf.append( dim1 );
    wkbuf.append( acsstr2 );
    setData( g1Index, wkbuf.toString(), data );
  }
/**
 *b1取得メソッド
 *@param dim1 int
 *@return Stringオブジェクト
 */
  public Object getB1( int dim1) throws J2CBException {
    String acsstr1="g1.g2[";
    String acsstr2="].b1";
    StringBuffer wkbuf;
    wkbuf=new StringBuffer( acsstr1 );
    wkbuf.append( dim1 );
    wkbuf.append( acsstr2 );
```

```
    return getData(g1Index, wkbuf.toString());
  }
```

## (2)　EJB 対応 COBOL アクセス用 Bean の生成例

「Bean 生成ツール」で，「EJB 対応機能を有効にする。」にチェックした場合の例を次に示します（そのほかのチェックボックスはデフォルト指定です）。

太字（色付き）は条件によって生成文字列が異なります。

なお，Windows(x64)では，コメント行個所の「COBOL adapter for Cosminexus Version 2」は，「COBOL adapter for Cosminexus Version 2(64)」となります。

### (a)　基本部分の生成例

```
package パッケージ名;

import jp.co.hitachi_sk.j2cb.*;

public class クラス名 extends CBLEJBAccess {
/*
 *Generated by COBOL adapter for Cosminexus Version 2 02-13
 *DO NOT EDIT THIS FILE
 *2013/09/30
 */
  //インデックスを定義
  private static final long serialVersionUID = 1347585448203L;
  private static final int personal_dataIndex=0;
  private boolean noload=true;
  private int[] mySize=null;
  private GroupAccess[] myGroupAccess=null;
  private String[] myTypeInfo=null;
  private int myLength=0;
/**
 *コンストラクタ
 */
  public クラス名() throws J2CBException {
    super();
    init();
    initialData();
    //データアクセスのための情報を設定
    setDllName("呼び出すライブラリ名");
    setFunctionName("呼び出すプログラム名");
  }
/**
 *初期化処理
 */
  private void init( ) throws J2CBException {
    //初期ロードだけ実行する
    if (noload) {
      String[] typeTmp={
       "G,UI0900,C(50),C(100),C(50)" };
      String[] sizeTmp={
       "204" };
```

```
      setType(typeTmp, sizeTmp);
      makeGroupAccess(personal_dataIndex,
       "01,05,05,05,05",
       "personal_data,p_number,p_name,p_address,p_gif",
       "0,0,4,54,154");
      mySize=size;
      myGroupAccess=groupAccess;
      myTypeInfo=typeInfo;
      myLength=length;
      noload=false;
    }
    else {
      setInformation(mySize, myGroupAccess, myTypeInfo, myLength);
    }
  }
/**
 *p_number設定メソッド
 *@param data Integerオブジェクト
 */
  public void setP_number( Object data) throws J2CBException {
    setData(personal_dataIndex, "personal_data.p_number", data);
  }
/**
 *p_name設定メソッド
 *@param data Stringオブジェクト
 */
  public void setP_name( Object data) throws J2CBException {
    setData(personal_dataIndex, "personal_data.p_name", data);
  }
/**
 *p_address設定メソッド
 *@param data Stringオブジェクト
 */
  public void setP_address( Object data) throws J2CBException {
    setData(personal_dataIndex, "personal_data.p_address", data);
  }
/**
 *p_gif設定メソッド
 *@param data Stringオブジェクト
 */
  public void setP_gif( Object data) throws J2CBException {
    setData(personal_dataIndex, "personal_data.p_gif", data);
  }
/**
 *p_number取得メソッド
 *@return Integerオブジェクト
 */
  public Object getP_number( ) throws J2CBException {
    return getData(personal_dataIndex, "personal_data.p_number");
  }
/**
 *p_name取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_name( ) throws J2CBException {
    return getData(personal_dataIndex, "personal_data.p_name");
  }
/**
```

```
 *p_address取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_address( ) throws J2CBException {
    return getData(personal_dataIndex, "personal_data.p_address");
  }
/**
 *p_gif取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_gif( ) throws J2CBException {
    return getData(personal_dataIndex, "personal_data.p_gif");
  }
}
```

### (b)　ホームインタフェースの生成例

```
package パッケージ名;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface クラス名EJBHome extends EJBHome {
/*
 *Generated by COBOL adapter for Cosminexus Version 2 02-13
 *DO NOT EDIT THIS FILE
 *2013/09/30
 */
/*
 *EJBのインスタンスの作成，除去のためのホームインタフェースを定義
 */
/**
 *createメソッド<BR>
 *ホームインタフェースは，createメソッドを一つ以上サポートし，
 *EJBean内で「ejbCreate」と呼ばれるメソッドに対応していなければ
 *なりません。<BR>
 *この二つのメソッドのパラメタ設定は同じです。
 */
  クラス名EJB create( ) throws CreateException ,RemoteException ,Exception ;
}
```

### (c)　リモートインタフェースの生成例

```
package パッケージ名;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface クラス名EJB extends EJBObject {
/*
 *Generated by COBOL adapter for Cosminexus Version 2 02-13
 *DO NOT EDIT THIS FILE
 *2013/09/30
 */
/*
```

```
 *EJBのビジネスメソッドのためのリモートインタフェースを定義
 */
  public int callCOBOL( ) throws RemoteException,Exception;
  public void setP_number( Integer val) throws
         RemoteException,Exception;
  public void setP_name( String val) throws
         RemoteException,Exception;
  public void setP_address( String val) throws
         RemoteException,Exception;
  public void setP_gif( String val) throws
         RemoteException,Exception;
  public Integer getP_number( ) throws RemoteException,Exception;
  public String getP_name( ) throws RemoteException,Exception;
  public String getP_address( ) throws RemoteException,Exception;
  public String getP_gif( ) throws RemoteException,Exception;
}
```

## (d)　Enterprise Bean の生成例

```
package パッケージ名;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import jp.co.hitachi_sk.j2cb.*;

public class クラス名EJBBean implements SessionBean {
/*
 *Generated by COBOL adapter for Cosminexus Version 2 02-13
 *DO NOT EDIT THIS FILE
 *2013/09/30
 */
/*
 *EJBのビジネスメソッドを実装する
 */
  //クラスの変数
  private static final long serialVersionUID = 1153786720297L;
  private クラス名 gEntity=null;
/**
 *COBOL呼び出し
 */
  public int callCOBOL( ) throws RemoteException,Exception{
    int rtn=0;
    try {
      rtn=gEntity.callCOBOL();
    } catch(J2CBException wkexp) {
      throw new Exception(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new RemoteException(wkmsg);
    }
    return rtn;
  }
/**
```

```
 *p_number設定メソッド
 */
  public void setP_number( Integer val) throws RemoteException,Exception{
    try {
      gEntity.setP_number(val);
    } catch(J2CBException wkexp) {
      throw new Exception(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new RemoteException(wkmsg);
    }
  }
/**
 *p_name設定メソッド
 */
  public void setP_name( String val) throws RemoteException,Exception{
    try {
      gEntity.setP_name(val);
    } catch(J2CBException wkexp) {
      throw new Exception(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new RemoteException(wkmsg);
    }
  }
/**
 *p_address設定メソッド
 */
  public void setP_address( String val) throws RemoteException,Exception{
    try {
      gEntity.setP_address(val);
    } catch(J2CBException wkexp) {
      throw new Exception(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new RemoteException(wkmsg);
    }
  }
/**
 *p_gif設定メソッド
 */
  public void setP_gif( String val) throws RemoteException,Exception{
    try {
      gEntity.setP_gif(val);
    } catch(J2CBException wkexp) {
      throw new Exception(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new RemoteException(wkmsg);
    }
```

```
  }
/**
 *p_number取得メソッド
 */
  public Integer getP_number( ) throws RemoteException,Exception{
    Integer rtn=null;
    try {
      rtn=(Integer)gEntity.getP_number();
    } catch(J2CBException wkexp) {
      throw new Exception(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new RemoteException(wkmsg);
    }
    return rtn;
  }
/**
 *p_name取得メソッド
 */
  public String getP_name( ) throws RemoteException,Exception{
    String rtn=null;
    try {
      rtn=(String)gEntity.getP_name();
    } catch(J2CBException wkexp) {
      throw new Exception(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new RemoteException(wkmsg);
    }
    return rtn;
  }
/**
 *p_address取得メソッド
 */
  public String getP_address( ) throws RemoteException,Exception{
    String rtn=null;
    try {
      rtn=(String)gEntity.getP_address();
    } catch(J2CBException wkexp) {
      throw new Exception(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new RemoteException(wkmsg);
    }
    return rtn;
  }
/**
 *p_gif取得メソッド
 */
  public String getP_gif( ) throws RemoteException,Exception{
    String rtn=null;
    try {
```

```
      rtn=(String)gEntity.getP_gif();
    } catch(J2CBException wkexp) {
      throw new Exception(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new RemoteException(wkmsg);
    }
    return rtn;
  }
/**
 *ejbCreate()の呼び出し
 */
  public void ejbCreate( ) throws CreateException,RemoteException,Exception{
    try {
      gEntity=new クラス名();
    } catch(J2CBException wkexp) {
      throw new CreateException(wkexp.getRemoteEJBMessage());
    } catch(Exception ee) {
      String wkmsg=null;
      wkmsg=ee.getMessage();
      ee.printStackTrace();
      throw new CreateException(wkmsg);
    }
  }
/**
 *関連するセッションコンテキストの設定
 */
  public void setSessionContext( SessionContext ctx) {
  }
/**
 *終了前処理
 */
  public void ejbRemove( ) {
    gEntity=null;
  }
/**
 *活性化メソッド
 */
  public void ejbActivate( ) {
  }
/**
 *非活性化メソッド
 */
  public void ejbPassivate( ) {
  }
}
```

## (e) デプロイ情報（DD ファイル）の生成例

```
<?xml version="1.0" encoding="MS932"?>
<!--
Generated by COBOL adapter for Cosminexus Version 2 02-13
DO NOT EDIT THIS FILE
2013/09/30
-->
```

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN" "http
://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
  <display-name>クラス名EJBBean</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <home>パッケージ名.クラス名EJBHome</home>
      <remote>パッケージ名.クラス名EJB</remote>
      <ejb-class>パッケージ名.クラス名EJBBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <method-intf>Home</method-intf>
      <method-name>create</method-name>
    </method>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>callCOBOL</method-name>
    </method>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>setP_number</method-name>
    </method>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>setP_name</method-name>
    </method>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>setP_address</method-name>
    </method>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>setP_gif</method-name>
    </method>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getP_number</method-name>
    </method>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getP_name</method-name>
    </method>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
```

```
      <method-intf>Remote</method-intf>
      <method-name>getP_address</method-name>
    </method>
    <method>
      <ejb-name>クラス名EJB.クラス名EJBHome</ejb-name>
      <method-intf>Remote</method-intf>
      <method-name>getP_gif</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
  </assembly-descriptor>
</ejb-jar>
```

# 付録 D　COBOL アクセス用 Bean 生成時に出力するメッセージ

Cosminexus 連携機能が出力するメッセージ内容について説明します。

## 付録 D.1　メッセージの形式

Cosminexus 連携機能が出力するメッセージは，次の形式です。

```
KCCCnnnnJ-x
メッセージテキスト
追加メッセージ
```

KCCC：プリフィクス

nnnn：メッセージ番号

J：Cosminexus 連携機能のエラーであることを示す。

x：メッセージレベル

E 重大エラー

W:警告エラー

I：エラーではないメッセージ

追加メッセージ：入力した登録集原文にエラーがある場合，次のメッセージを表示します。

```
n[,n]…行目付近を見直してください。
```

n はエラーが発生した行番号（相対行番号）を示します。

ただし，厳密な行番号が出力できない場合があるため，該当行またはその前後の行番号となります。

特定の行に関連しないエラーの場合，追加メッセージは表示されません。

## 付録 D.2　メッセージ一覧

### (1)　COBOL アクセス用 Bean 生成ツールが出力するメッセージ

| メッセージ ID | メッセージテキスト |
|---|---|
| KCCC1001J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>予期しないエラーが発生しました。 |
| KCCC1002J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>予期しないエラーが発生しました。 |
| KCCC1003J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。 |

| メッセージ ID | メッセージテキスト |
|---|---|
| | メモリ不足が発生しました。 |
| KCCC1004J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>I/O エラーが発生しました。 |
| KCCC1005J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>メモリ不足が発生しました。 |
| KCCC1006J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>COPY 文の解析処理で予期しないエラーが発生しました。 |
| KCCC1007J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>解析情報を格納する一時ファイルを作成できませんでした。 |
| KCCC1008J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>指定された通貨記号が不当です。 |
| KCCC1009J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>I/O エラーが発生しました。 |
| KCCC1011J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>本製品で許していない語が現れました。<br>または書き方に誤りがあります。 |
| KCCC1012J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>1 行の文字数が 255 文字を超えています。 |
| KCCC1013J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>入力行数が 32,760 行を超えました。 |
| KCCC1014J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>FILLER 項目の数が 65,535 個を超えました。 |
| KCCC1015J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>語，または PICTURE 文字列の長さが 30 字を超えています。 |
| KCCC1016J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>数字の桁数が 18 桁を超えています。 |
| KCCC1017J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>終止符の直後には空白が必要です。 |
| KCCC1018J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>最初のデータ定義のレベル番号が 01/77 以外です。 |
| KCCC1019J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>レベル番号は先に出た基本項目の属する集団のうちの<br>どれかのレベル番号と同じでなければなりません。 |
| KCCC1020J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>レベル番号の指定に誤りがあります。 |
| KCCC1021J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。 |

| メッセージ ID | メッセージテキスト |
| --- | --- |
| | PICTURE 句が 2 重に定義されています。 |
| KCCC1022J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>PICTURE 句の書き方に誤りがあります。 |
| KCCC1023J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>USAGE 句が 2 重に定義されています。 |
| KCCC1024J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>OCCURS 句が 2 重に定義されています。 |
| KCCC1025J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>OCCURS 句に指定した反復回数の値が正しくありません。 |
| KCCC1026J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>SIGN 句が 2 重に定義されています。 |
| KCCC1027J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>REDEFINES 句が 2 重に定義されています。 |
| KCCC1028J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>01/77 レベルのデータ項目に OCCURS 句を指定することはできません。 |
| KCCC1029J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>集団項目に PICTURE 句を指定することはできません。 |
| KCCC1030J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>PICTURE 句と USAGE 句の指定が矛盾しています。 |
| KCCC1031J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>PICTURE 句あるいは USAGE 句と SIGN 句の指定が矛盾しています。 |
| KCCC1032J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>基本項目に USAGE 句も PICTURE 句も指定されていません。 |
| KCCC1033J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>01/77 レベル以外に USAGE ADDRESS/POINTER は指定できません。 |
| KCCC1034J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>USAGE POINTER/ADDRESS は指定できません。 |
| KCCC1035J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>日本語文字が 72 カラムと 73 カラムにまたがっています。 |
| KCCC1036J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>7 カラム目に不当な文字が指定されています。 |
| KCCC1041J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>プログラム論理エラーが発生しました。 |
| KCCC1042J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>パーサエンジンがスタックオーバーフローしました。 |

| メッセージ ID | メッセージテキスト |
|---|---|
| KCCC1043J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>データ記述項の書き方に誤りがあるか，サポートされていない構文です。 |
| KCCC1044J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>パーサエンジンがエラー終了しました。 |
| KCCC1051J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>登録集原文ファイルが開けません。 |
| KCCC1052J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>COPY 文の解析処理で I/O エラーが発生しました。 |
| KCCC1053J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>1 行の文字数が 255 文字を超えています。 |
| KCCC1054J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>入力行数が 32,760 行を超えました。 |
| KCCC1055J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>COPY 文のネストレベルが 19 を超えました。 |
| KCCC1056J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>登録集原文ファイルが開けません。 |
| KCCC1057J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>COPY 文の解析処理で I/O エラーが発生しました。 |
| KCCC1058J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>COPY 文の書き方に誤りがあります。またはサポートされていない構文です。 |
| KCCC1059J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>原文名に誤りがあります。 |
| KCCC1060J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>登録集原文ファイルが見つかりません。 |
| KCCC1061J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>フリー形式の原文と固定形式の原文は混在できません。 |
| KCCC1071J-E | OCCURS 句の書き方に誤りがあります。 |
| KCCC1072J-E | OCCURS 句の書き方に誤りがあります。 |
| KCCC1099J-E | COBOL への引数を定義した登録集原文の構文解析処理でエラーが発生しました。<br>解析処理中に例外が発生しました。 |
| KCCC1101J-W | COBOL への引数を定義した登録集原文の構文解析処理で警告エラーが発生しました。<br>REDEFINES 句がデータ名の後ろにありません。<br>直前にあるものと仮定します。 |
| KCCC1102J-W | COBOL への引数を定義した登録集原文の構文解析処理で警告エラーが発生しました。<br>REDEFINES 句のあるデータ項目のレベル番号と等しいレベル番号が見つかりません。<br>または REDEFINES 句で指定されたデータ名が定義されていません。 |

| メッセージ ID | メッセージテキスト |
| --- | --- |
| | REDEFINES 句を無視します。 |
| KCCC1103J-W | COBOL への引数を定義した登録集原文の構文解析処理で警告エラーが発生しました。<br>再定義項目と被再定義項目の大きさを同じにするか,<br>または再定義項目を小さくする必要があります。<br>再定義項目の大きさを採用します。 |
| KCCC1301J-E | TEMP フォルダを作成できません。 |
| KCCC1302J-E | 通貨編集文字の指定がありません。または指定に誤りがあります。 |
| KCCC1303J-E | 生成ファイルを出力するフォルダの指定がありません。 |
| KCCC1305J-W | フォルダ(*** 5 ***)が見つかりません。作成しますか？ |
| KCCC1306J-E | フォルダを作成できませんでした。 |
| KCCC1307J-E | 生成ファイルを出力するフォルダに指定された値はフォルダ名ではありません。 |
| KCCC1308J-E | j2cb2kpars.dll をローディングできません。 |
| KCCC1311J-E | COBOL への引数を定義した登録集原文のファイルの指定がありません。 |
| KCCC1312J-E | COBOL への引数を定義した登録集原文のファイルが見つかりません。 |
| KCCC1313J-E | 登録集原文に指定された値はファイル名ではありません。 |
| KCCC1361J-E | OCCURS 句の DEPENDING ON 指定はサポートされていません。 |
| KCCC1391J-I | 生成ファイルの出力が完了しました。生成ツールを終了しますか？ |
| KCCC1401J-E | 引数*** 1 ***の別名に Java の変数名規則として不当な文字が含まれています。<br>文字: "*** 3 ***" |
| KCCC1403J-E | 引数*** 1 ***の別名(*** 2 ***)に英文字,数字,アンダーバー,ドル記号以外が含まれています。<br>別名を再設定してください。<br>すべての別名に対してチェック処理をスキップしますか？ |
| KCCC1405J-E | 引数*** 1 ***のデータ名(*** 2 ***)に<br>英文字,数字,アンダーバー,ドル記号以外が含まれています。<br>別名を設定してください。<br>すべてのデータ名に対してチェック処理をスキップしますか？ |
| KCCC1406J-E | データ名(*** 2 ***)を使用して生成するソースでは変数名が一意になりません。<br>別名を設定してください。 |
| KCCC1407J-E | 別名(*** 2 ***)を使用して生成するソースでは変数名が一意になりません。<br>別名を再設定してください。 |
| KCCC1408J-E | データ名(*** 2 ***)を使用して生成するソースではメソッド名が一意になりません。<br>別名を設定してください。 |
| KCCC1409J-E | 別名(*** 2 ***)を使用して生成するソースではメソッド名が一意になりません。<br>別名を再設定してください。 |

| メッセージ ID | メッセージテキスト |
|---|---|
| KCCC1441J-E | 再定義項目は可変長データとして指定できません。 |
| KCCC1442J-E | 長さが 5 バイト未満の項目は可変長データとして指定できません。 |
| KCCC1443J-W | 可変長データを再定義しています。<br>再定義項目および従属する項目に対する setter/getter は生成されません。 |
| KCCC1444J-E | 01/77 レベル以外の項目は可変長データとして指定できません。 |
| KCCC1445J-E | 再定義項目はアドレスデータとして指定できません。 |
| KCCC1446J-W | アドレスデータを再定義しています。<br>再定義項目および従属する項目に対する setter/getter は生成されません。 |
| KCCC1447J-E | 引数*** 1 ***の引数順の指定がありません。 |
| KCCC1448J-E | 引数*** 1 ***の引数順の値が不正です。 |
| KCCC1449J-E | 引数*** 1 ***の引数順に 0 以下の値が指定されています。 |
| KCCC1450J-E | 引数*** 1 ***の引数順の値が項目数を超えています。 |
| KCCC1451J-E | 同じ引数順の値(*** 1 ***)が重複して指定されています。 |
| KCCC1471J-E | パッケージ名の指定がありません。 |
| KCCC1472J-E | パッケージ名として正しくありません。 |
| KCCC1473J-E | クラス名の指定がありません。 |
| KCCC1474J-E | クラス名として正しくありません。 |
| KCCC1475J-E | ライブラリ名の指定がありません。 |
| KCCC1476J-E | プログラム名の指定がありません。 |
| KCCC1501J-W | ファイルはパッケージに既に存在します。<br>上書きしますか？ファイル名：*** 4 *** |
| KCCC1502J-E | 生成ソースを格納するファイルが作成できませんでした。 |
| KCCC1504J-E | 生成ソースを格納するフォルダが作成できませんでした。 |
| KCCC1505J-E | ファイルを作成できませんでした。またはファイルにアクセスできませんでした。 |
| KCCC1506J-E | クラスを生成できませんでした。 |
| KCCC1511J-W | ファイルはパッケージに既に存在します。上書きしますか？ |
| KCCC1512J-E | 生成ソースを格納するファイルが作成できませんでした。 |
| KCCC1514J-E | 生成ソースを格納するフォルダが作成できませんでした。 |
| KCCC1515J-E | ファイルを作成できませんでした。またはファイルにアクセスできませんでした。 |
| KCCC1516J-E | クラスを生成できませんでした。 |
| KCCC1517J-E | クラスを生成できませんでした。 |

| メッセージ ID | メッセージテキスト |
|---|---|
| KCCC1518J-E | クラスを生成できませんでした。 |
| KCCC1519J-E | クラスを生成できませんでした。 |
| KCCC1520J-E | 生成ソースを格納するフォルダが作成できませんでした。 |
| KCCC1601J-E | 解析処理中に例外が発生しました。 |
| KCCC1602J-E | COBOL アクセス用 Bean 生成処理が異常終了しました。 |
| KCCC1651J-E | テーブルの初期化処理中に例外が発生しました。 |
| KCCC1661J-E | メモリ不足が発生しました。 |
| KCCC1801J-E | 2 重起動しました。終了します。 |
| KCCC1802J-E | 内部エラー：*** 6 *** |
| KCCC1811J-E | 内部エラー：*** 6 *** |
| KCCC1812J-E | フォルダとして有効でありません。 |
| KCCC1813J-E | 最後に '¥' を付加した状態で 260 バイト以内にしてください。 |
| KCCC1814J-E | 設定したフォルダは不当です。再設定してください。 |
| KCCC1815J-E | ドライブ指定が必要です。再設定してください。 |
| KCCC1816J-E | 固定ドライブでなければなりません。再設定してください。 |
| KCCC1817J-E | J2SE のルートフォルダを正しく設定してください。 |
| KCCC1818J-E | 設定した値は処理上使用できません。再設定してください。 |
| KCCC1819J-E | J2SE のルートフォルダを設定してください。 |
| KCCC1820J-E | 最大メモリプールサイズに整数以外が指定されています。 |
| KCCC1821J-E | 最大メモリプールサイズの値が指定されていません。 |
| KCCC1822J-E | 設定ファイルの出力処理でエラーが発生しました。 |
| KCCC1831J-E | 初期化処理中にエラーが発生しました。 |
| KCCC1832J-E | J2SE のルートフォルダが見つかりません。<br>ルートフォルダを再設定してください。 |
| KCCC1833J-E | 生成ツールを起動できませんでした。<br>ファイルが見つかりません。<br>J2SE のルートフォルダを再設定してください。 |
| KCCC1834J-E | 生成ツールを起動できませんでした。<br>ファイルにアクセスできません。<br>J2SE のルートフォルダ下のアクセス権を見直してください。 |
| KCCC1835J-E | 生成ツールを起動できませんでした。メモリ不足です。 |
| KCCC1836J-E | 生成ツールを起動できませんでした。*** 6 *** |

| メッセージ ID | メッセージテキスト |
|---|---|
| KCCC1837J-E | 生成ツール実行時にエラーが発生しました。<br>実行環境を見直してください。 |
| KCCC1841J-E | 2 重起動しました。終了します。 |
| KCCC1842J-E | 内部エラー：*** 6 *** |
| KCCC1843J-E | 初期処理時にエラーが発生しました。 |
| KCCC1851J-E | ヘルプを起動できません。*** 6 *** |
| KCCC1852J-E | ヘルプファイルが見つかりません。 |
| KCCC1853J-E | ヘルプ起動時にエラーが発生しました。 |
| KCCC1854J-E | ヘルプ起動時にエラーが発生しました。 |
| KCCC1855J-E | 内部エラー：*** 6 *** |
| KCCC1861J-E | 終了処理時にエラーが発生しました。 |
| KCCC1862J-E | 終了処理時にエラーが発生しました。 |
| KCCC1871J-E | j2cb2kwJNI.dll をローディングできません。 |
| KCCC1872J-E | ヘルプ起動時にエラーが発生しました。 |
| KCCC1873J-E | 終了処理時にエラーが発生しました。 |
| KCCC1881J-E | 初期処理時にエラーが発生しました。 |
| KCCC5001J-I | 本当に終了してよろしいですか？ |

注　表中の *** n *** は，次のとおりです。

*** 1 ***：引数の順番を示す数字

*** 2 ***：データ名または別名

*** 3 ***：Java の変数名規則として不当な文字

*** 4 ***：ファイル名

*** 5 ***：フォルダ名

*** 6 ***：詳細メッセージ

# 付録 E　例外情報コード一覧

例外情報コードとメッセージ内容，エラー発生要因，システムの処理およびプログラマの対策について説明します。

## 付録 E.1　例外情報コードの形式

getErrorCode および getEJBErrorCode で取得する例外情報コードは，次の形式です。

```
J2CByyynnnn
```

J2CB：プリフィクス

　COBOL アクセス以外で発生した例外情報コードの場合は，”ETC:”となることもあります。

yyy：

　エラー発生場所（メソッド）コード：表 F-2 および表 F-3 を参照してください。

nnnn：

　エラー要因コード：表 F-1 のように分類されます。

**表 E-1　例外名一覧（JavaBean/EJB 共通）**

| エラー要因コード | 例外名 | 意　味 |
|---|---|---|
| 0000～0FFF | SYS_ERR | システムエラー |
| 1000～10FF | J_ENV_ERR | 実行環境エラー |
| 1100～1FFF | INVALID_TYPE | データ項目属性エラー |
| 2000～2FFF | INVALID_ARG | 引数指定エラー |
| 4000～5FFF | UNRECOVERABLE | 回復不能エラー |
| 9999 | UNKNOWN | 予期しないエラー |

## (1)　JavaBean 用エラー発生場所（メソッド）コード

**表 E-2　JavaBean 用エラー発生場所（メソッド）コード**

| コード(16 進) | 場所 |
|---|---|
| 001 | getData |
| 010 | setData |
| 050 | init() |
| 101 | callCOBOL |
| 500 | GroupAccess |

| コード(16 進) | 場所 |
|---|---|
| 601 | CBLAccess.getData(int) |
| 602 | CBLAccess.setData(int，Object) |
| 603 | CBLAccess.getData(int，String) |
| 604 | CBLAccess.setData(int，String，Object) |
| 605 | CBLAccess.callCOBOL |
| 606 | CBLAccess.makeGroupAccess |
| 607 | CBLAccess.initialData |
| 608 | CBLAccess.makeVarData |
| 800 | getEnv |
| 801 | getOption |
| 999 | 不明 |

## (2)　EJB 用エラー発生場所（メソッド）コード

表 E-3　EJB 用エラー発生場所（メソッド）コード

| コード(16 進) | 場所 |
|---|---|
| 001 | getData |
| 010 | setData |
| 050 | init() |
| 101 | CallCOBOL |
| 651 | CBLEJBAccess.getData(int) |
| 652 | CBLEJBAccess.setData(int，Object) |
| 653 | CBLEJBAccess.getData(int，String) |
| 654 | CBLEJBAccess.setData(int，String，Object) |
| 655 | CBLEJBAccess.callCOBOL |
| 656 | CBLEJBAccess.makeGroupAccess |
| 657 | CBLEJBAccess.initialData |
| 658 | CBLEJBAccess.makeVarData |
| 800 | getEnv |
| 801 | getOption |
| 999 | 不明 |

## 付録 E.2　メッセージ文字列の形式

getMessage で取得するメッセージ文字列は，次の形式です。

```
例外情報コード：YYY 実行中にエラーが発生しました。 MSG
```

例外情報コード：付録 F.1 を参照してください。

YYY：エラー発生場所（メソッド）名称（表 F-2 または表 F-3 参照を参照してください)。

MSG：エラー要因コード別のメッセージ内容

また，メッセージ一覧は，メッセージ文字列を次の形式で記載しています。

### 例外情報コード

MSG

要因

　メッセージの説明を示す。

（S）

　システムの処置を示す。

（P）

　プログラム作成者の処置を示す。

## 付録 E.3　メッセージ文字列の取得方法

メッセージ文字列は，J2CBException クラスの getMessage または getEJBMessage メソッドで取得できます。

例：

```
 :
try {
  : (引数設定，プログラム呼び出しなど)
} catch ( J2CBException e ) {
 String msg=e.getMessage();
 System.out.println("Message=[" + msg + "]");
}
  :
```

また，Java プログラムのどこでエラーとなったかがわからない場合は，printStackTrace で表示することもできます。

例：

```
 :
try {
  : (引数設定，プログラム呼び出しなど)
} catch ( J2CBException e ) {
 e.printStackTrace();
}
 :
```

上記の例で標準出力に出力した場合の出力先は次のようになります。

### (1) Web コンテナ使用した場合の出力先

通常は，「Web コンテナを起動したコンソール画面」となります。

## 付録 E.4 メッセージ一覧

### J2CByyy0001

COBOL で作成したプログラムでエラーが発生しました。
プログラム名称=*** 1 ***[，シグナル種別=*** 2 ***]

要因

COBOL で作成したプログラムでエラーが発生した。考えられる要因は次のどちらかである。

1. COBOL プログラム実行中に異常終了した。
2. COBOL プログラム実行中に STOP RUN が実行された。

*** 1 ***：プログラム名称

(Windows(x86)の場合)

プログラム名称は， _プログラム名@n の形式です（stdcall 呼び出し規約に準拠した名称です)。

n ：次の計算式で求められる値を指定します。

n=引数の数 × 4

例：プログラム名が SUB1 で，2 個の引数がある場合のプログラム名称は「_SUB1@8」となります。

(Windows(x64)，UNIX の場合)

プログラム名称は，プログラム名そのままです。

*** 2 ***：シグナル種別

4 ： SIGILL

6 ： SIGIOT（AIX の場合だけ）

7 ： SIGEMT（AIX の場合)，SIGBUS（Linux の場合）

8 ： SIGFPE

10： SIGBUS（AIX の場合※）

11： SIGSEGV

注※　Linux の場合，シグナル種別番号 10 はありません。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

1. 異常終了時要約情報リストが出力されていれば，このリストを基に，COBOL で作成したプログラムのエラー原因を修正して再実行する。異常終了時要約情報リスト※1 が未出力であれば，「実行時デバッグ機能オプション※2 またはテストデバッグ機能オプション※3 指定でコンパイルして作成したプログラム」で，環境変数 CBLABNLST を指定して実行する。出力された異常終了時要約情報リストを基に，調査・修正する。
2. STOP RUN 文を探し，EXIT PROGRAM に変更する。

注※1　異常終了時要約情報リストは，COBOL で作成したプログラムが異常終了した際に出力されるリストで，「実行時デバッグ機能オプション※2 またはテストデバッグ機能オプション※3 指定でコンパイルして作成したプログラム」で「環境変数 CBLABNLST が設定されている」場合に出力されます。

注※2　実行時デバッグ機能オプション

- -DebugInf
- -DebugInf,Trace
- -DebugCompati
- -DebugData
- -DebugRange

注※3　テストデバッグ機能オプション

- -TDInf
- -CVInf

## J2CByyy0002

予期しないエラーが発生しました。[コード値=*** 9 ***]

要因

予期しないエラーが発生した。

*** 9 ***：「例外の種類を識別する」値

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

エラー発生場所（メソッド）コードと「例外の種類を識別する」値を控えて，保守員に連絡する。

## J2CByyy0003

指定されたライブラリがロードできませんでした。ライブラリ名=*** 2 ***,エラー番号=*** 3 ***

要因

「COBOL で作成したライブラリ」がロードできない。考えられる要因は次のどれかである。

1.「COBOL で作成したライブラリ」のロード時に下記[2]以外のエラーが発生した。

2. COBOL アクセス用 Bean 生成時に指定した「ライブラリ名」が誤っている。

*** 2 ***：ロードしようとした「COBOL で作成したライブラリ」名称。

*** 3 ***：エラー番号（詳細は下記参照）

（Windows の場合）

Microsoft C++(JNI 部分)の関数(LoadLibrary)で発生したエラーの詳細情報を表します。エラー番号の形式は，次のようになります。

```
エラー番号=Windows APIが返すエラーコード
```

主なエラーコードとエラーの内容を次に示します。

**表 E-4　実行時の主なエラーコードとエラーの内容**

| エラーコード | エラーの内容 |
|---|---|
| 126 | 指定されたライブラリが見つからない。 |
| 193 | 有効な Windows アプリケーションではない。 |

（UNIX の場合）

C++(JNI 部分)の関数で発生したエラーの詳細情報を表します。エラー番号の形式は，次のようになります。

```
error = errno
```

errno については，man 機能の errno(2)で参照できます。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

1. 表示された「COBOL で作成したライブラリ」が存在し，実行可能であることを確認後，再実行する。

2.「ライブラリ名」を正しく設定して COBOL アクセス用 Bean を再作成し，再実行する。

## J2CByyy0004

指定されたライブラリ内のプログラム名称が見つかりませんでした。プログラム名称=*** 1 ***

要因

「COBOL で作成したライブラリ」中に，COBOL アクセス用 Bean 生成時に指定した「呼び出す COBOL プログラム名」が見つからない。「ライブラリ名」と「呼び出す COBOL プログラム名」のどちらかが誤っていることが考えられる。

*** 1 ***：プログラム名称（上記 J2CByyy0001 を参照してください）

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

「ライブラリ名」と「呼び出す COBOL プログラム名」を正しく設定して COBOL アクセス用 Bean を再作成し，再実行する。

## J2CByyy0005

COBOL の実行時ライブラリまたは関数が見つかりませんでした。

要因

COBOL2002 がインストールされていないか，環境設定が正しく行われていない。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

COBOL2002 をインストールし，環境設定を見直して，再実行する。

## J2CByyy0008

引数のデータ領域が確保できませんでした。

要因

「COBOL で作成したプログラム」と受け渡す引数データ領域を作成しようとしたが，作成できなかった。要因として，メモリ不足が考えられる。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy0009

データ変換時に必要な作業領域が確保できませんでした。

要因

setter/getter で行うデータ変換中にメモリ不足が発生した。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy0010

領域が確保できませんでした。領域種別=*** 1 ***

要因

領域種別*** 1 ***に示す領域を作成しようとしたが，作成できなかった。要因として，メモリ不足が考えられる。

*** 1 ***：領域種別

シグナル連鎖機能

COBOL 環境変数

COBOL サービスルーチン

COBOL プログラム名称

COBOL ライブラリ名称

COBOL アクセス実行環境変数

スレッド初期設定処理

環境設定ファイル

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy0011

ライブラリの指定が絶対パスでありません。

要因

COBOL アクセス用 Bean 生成時に指定したライブラリ名が絶対パス指定になっていない。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

COBOL アクセス用 Bean 生成時に指定するライブラリ名を絶対パス指定に変更して再生成後，再実行する。または，dynamicpath オプションで yes を指定し，再実行する。

## J2CByyy0013

既に COBOL プログラムから Java プログラムを呼び出しているため，その Java プログラムから COBOL プログラムを呼び出せません。

要因

Java プログラム呼び出し機能が実行中である。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

COBOL プログラムを別プロセスで呼び出すなどで対処する。

## J2CByyy1001

java/lang/String クラス，コンストラクタまたはメソッドが見つかりません。

要因

利用可能なメモリが不足している。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy1011

java/math/BigDecimal クラス，コンストラクタまたはメソッドが見つかりません。

要因

利用可能なメモリが不足している。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy1021

java/lang/Integer クラス，コンストラクタまたはメソッドが見つかりません。

要因

利用可能なメモリが不足している。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy1031

java/lang/Short クラス，コンストラクタまたはメソッドが見つかりません。

要因

利用可能なメモリが不足している。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy1041

java/lang/Long クラス，コンストラクタまたはメソッドが見つかりません。

要因

利用可能なメモリが不足している。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy1051

java/lang/Float クラス，コンストラクタまたはメソッドが見つかりません。

要因

利用可能なメモリが不足している。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy1061

java/lang/Double クラス，コンストラクタまたはメソッドが見つかりません。

要因

利用可能なメモリが不足している。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy10A1

jp/co/hitachi_sk/j2cb/J2CBException クラス，コンストラクタまたはメソッドが見つかりません。

要因

利用可能なメモリが不足している。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy1101

符号なし数字項目に格納するデータに，符号が設定されています。

要因

符号なし数字項目に対して，符号付きの数値を設定しようとした。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

符号なし数字項目に設定しているデータに符号が付いたデータが設定されていないかを見直して，再実行する。

## J2CByyy1102

予期しない型情報です。*** 4 ***

要因

Cosminexus 連携機能でサポートしていないデータ項目属性を，COBOL アクセス用 Bean で指定した。考えられる要因は次のどちらかである。

1. Cosminexus 連携機能でサポートしていないデータ項目を記述した COBOL プログラムから生成した COBOL アクセス用 Bean を実行した。
2. 生成した COBOL アクセス用 Bean を変更した。

*** 4 ***：エラーとなった型情報文字列

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

1. Cosminexus 連携機能でサポートされていないデータ項目を取り除いて COBOL アクセス用 Bean を作成し，再実行する。または，Cosminexus 連携機能を，データ項目がサポートされたバージョンにバージョンアップして，再実行する。
2. COBOL アクセス用 Bean に加えた変更を戻して Java プログラムを作成し，再実行する。

## J2CByyy1103

型情報からデータ長を特定できませんでした。*** 4 ***

要因

Cosminexus 連携機能でサポートしていないデータ項目属性を，COBOL アクセス用 Bean で指定した。考えられる要因は次のどちらかである。

1. Cosminexus 連携機能でサポートしていないデータ項目を記述した COBOL プログラムから生成した COBOL アクセス用 Bean を実行した。
2. 生成した COBOL アクセス用 Bean を変更した。

*** 4 ***：エラーとなった型情報文字列

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

1. Cosminexus 連携機能でサポートされていないデータ項目を取り除いて COBOL アクセス用 Bean を作成し，再実行する。または，Cosminexus 連携機能を，データ項目がサポートされたバージョンにバージョンアップして，再実行する。
2. COBOL アクセス用 Bean に加えた変更を戻して Java プログラムを作成し，再実行する。

## J2CByyy1104

型情報から全桁数を特定できませんでした。*** 4 ***

要因

Cosminexus 連携機能でサポートしていないデータ項目属性を，COBOL アクセス用 Bean で指定した。考えられる要因は次のどちらかである。

1. Cosminexus 連携機能でサポートしていないデータ項目を記述した COBOL プログラムから生成した COBOL アクセス用 Bean を実行した。
2. 生成した COBOL アクセス用 Bean を変更した。

*** 4 ***：エラーとなった型情報文字列

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

1. Cosminexus 連携機能でサポートされていないデータ項目を取り除いて COBOL アクセス用 Bean を作成し，再実行する。または，Cosminexus 連携機能を，データ項目がサポートされたバージョンにバージョンアップして，再実行する。
2. COBOL アクセス用 Bean に加えた変更を戻して Java プログラムを作成し，再実行する。

## J2CByyy1105

型情報から小数桁数を特定できませんでした。*** 4 ***

要因

Cosminexus 連携機能でサポートしていないデータ項目属性を，COBOL アクセス用 Bean で指定した。考えられる要因は次のどちらかである。

1. Cosminexus 連携機能でサポートしていないデータ項目を記述した COBOL プログラムから生成した COBOL アクセス用 Bean を実行した。
2. 生成した COBOL アクセス用 Bean を変更した。

*** 4 ***：エラーとなった型情報文字列

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

1. Cosminexus 連携機能でサポートされていないデータ項目を取り除いて COBOL アクセス用 Bean を作成し，再実行する。または，Cosminexus 連携機能を，データ項目がサポートされたバージョンにバージョンアップして，再実行する。
2. COBOL アクセス用 Bean に加えた変更を戻して Java プログラムを作成し，再実行する。

## J2CByyy1106

型情報の小数桁数が全桁数を超えています。*** 4 ***

要因

Cosminexus 連携機能でサポートしていないデータ項目属性を，COBOL アクセス用 Bean で指定した。考えられる要因は次のどちらかである。

1. Cosminexus 連携機能でサポートしていないデータ項目を記述した COBOL プログラムから生成した COBOL アクセス用 Bean を実行した。
2. 生成した COBOL アクセス用 Bean を変更した。

*** 4 ***：エラーとなった型情報文字列

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

1. Cosminexus 連携機能でサポートされていないデータ項目を取り除いて COBOL アクセス用 Bean を作成し，再実行する。または，Cosminexus 連携機能を，データ項目がサポートされたバージョンにバージョンアップして，再実行する。
2. COBOL アクセス用 Bean に加えた変更を戻して Java プログラムを作成し，再実行する。

## J2CByyy2001

アクセス文字列から項目を特定できませんでした。
アクセス文字列:*** 5 ***

要因

COBOL アクセス用 Bean の初期処理で指定するデータ名称と異なるデータ名称を，setter/getter で指定した。

考えられる要因は次のどちらかである。

1. 指定した添字の値が範囲外である。
2. 生成した COBOL アクセス用 Bean を変更した。

*** 5 ***：アクセス文字列

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

1. 添字の値を正しい値に変更し，再実行する。
2. COBOL アクセス用 Bean に加えた変更を戻して Java プログラムを作成し，再実行する。または，Cosminexus 連携機能で生成した COBOL アクセス用 Bean で Java プログラムを作成し，再実行する。

## J2CByyy2002

レベル番号，型または識別文字列が不正です。
level:*** 6 *** name:*** 7 *** type:*** 4 ***

要因

Cosminexus 連携機能でサポートしていないデータ項目属性を，COBOL アクセス用 Bean で指定した。考えられる要因は次のどちらかである。

1. Cosminexus 連携機能でサポートしていないデータ項目を記述した COBOL プログラムから生成した COBOL アクセス用 Bean を実行した。
2. 生成した COBOL アクセス用 Bean を変更した。

*** 4 ***：型情報文字列

*** 6 ***：レベル番号文字列

*** 7 ***：識別文字列

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

1. Cosminexus 連携機能でサポートされていないデータ項目を取り除いて COBOL アクセス用 Bean を作成し，再実行する。または，Cosminexus 連携機能を，データ項目がサポートされたバージョンにバージョンアップして，再実行する。
2. COBOL アクセス用 Bean に加えた変更を戻して Java プログラムを作成し，再実行する。

## J2CByyy2003

識別子の配列情報が不正です。識別子:*** 7 ***

要因

Cosminexus 連携機能でサポートしていないデータ項目属性を，COBOL アクセス用 Bean で指定した。生成した COBOL アクセス用 Bean を変更したことが考えられる。

*** 7 ***：識別文字列

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

COBOL アクセス用 Bean に加えた変更を戻して Java プログラムを作成し，再実行する。または，Cosminexus 連携機能で生成した COBOL アクセス用 Bean で Java プログラムを作成し，再実行する。

## J2CByyy2004

入力の引数が設定されていません。

要因

予期しないエラーが発生した。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

エラー発生場所（メソッド）コードを控え，保守員に連絡する。

## J2CByyy2005

入力のアドレスが設定されていません。

要因

予期しないエラーが発生した。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

エラー発生場所（メソッド）コードを控え，保守員に連絡する。

## J2CByyy2006

入力の型情報が設定されていません。

要因

予期しないエラーが発生した。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

エラー発生場所（メソッド）コードを控え，保守員に連絡する。

## J2CByyy2007

型情報に NULL が設定されています。

要因

内部情報作成時に使用するデータ型情報が不当である。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

COBOL アクセス用 Bean を書き換えている場合は元に戻して Java アプリケーションを作成し，再実行する。Cosminexus 連携機能で生成した COBOL アクセス用 Bean でもこのエラーとなる場合は，COBOL アクセス用 Bean を控えて，保守員に連絡する。

## J2CByyy2008

不正なインデックス情報を得ました。index=*** 8 ***

要因

内部情報の作成，参照時に使用するインデックス値が不当である。生成した COBOL アクセス用 Bean を変更したことが考えられる。

*** 8 ***：インデックス値

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

COBOL アクセス用 Bean を書き換えている場合は元に戻して Java アプリケーションを作成し，再実行する。Cosminexus 連携機能で生成した COBOL アクセス用 Bean でもこのエラーとなる場合は，COBOL アクセス用 Bean を控えて，保守員に連絡する。

## J2CByyy2009

アクセス文字列に NULL が設定されています。

要因

COBOL アクセス用 Bean の初期処理で指定するデータ名称と異なるデータ名称を，getter で指定した。生成した COBOL アクセス用 Bean を変更したことが考えられる。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

COBOL アクセス用 Bean に加えた変更を戻して Java プログラムを作成し，再実行する。または，Cosminexus 連携機能で生成した COBOL アクセス用 Bean で Java プログラムを作成し，再実行する。Cosminexus 連携機能で生成した COBOL アクセス用 Bean でもこのエラーとなる場合は，COBOL アクセス用 Bean を控えて，保守員に連絡する。

## J2CByyy2011

サーバから Java オブジェクトに変換できない数値データを得ました。

*** 9 ***

要因

「COBOL で作成したプログラム」で設定された数値データ値が不当である。

*** 9 ***：エラーとなった数値データ

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

「COBOL で作成したプログラム」で数値データを設定する個所を調査・修正して，再実行する。

## J2CByyy2012

サーバから Java オブジェクトに変換できない文字データを得ました。

要因

「COBOL で作成したプログラム」で設定された文字列データ値が不当である。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

「COBOL で作成したプログラム」で文字列データを設定する個所を調査・修正して，再実行する。

## J2CByyy2013

引数の数が 4096 個を超えています。
引数の数が 1024 個を超えています。

Windows(x64)，AIX(64)の場合は「引数の数が 1024 個を超えています。」，Linux の場合は「引数の数が 4096 個を超えています。」が出力されます。

要因

引数の個数が，指定可能な個数（4,096 個または 1,024 個）を超えている。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

引数の個数を，指定可能な個数（4,096 個または 1,024 個）以下にする。

## J2CByyy2014

setter で設定したデータの属性が，データ項目に対応するデータ属性と異なっています。

要因

setXxx の引数に指定したオブジェクトが，設定しようとするデータ項目に対応するデータ属性ではない。
例：
小数桁のない 2 進データ項目の setter の引数に String を指定した

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

setter の引数を，データ項目に対応するデータ属性に合わせた Java プログラムを作成し，再実行する。

## J2CByyy4001

文字列をネイティブ文字列に変換できません。*** 9 ***

要因

「COBOL で作成したプログラム」に渡す引数情報作成時に，不当な文字データ（数字を設定する必要がある個所が数字でない）があるか，ライブラリ名称，プログラム名称に変換できない不当な文字データがある。もしくは，「COBOL で作成したプログラム」で設定された引数に，変換できない文字データがある。

*** 9 ***：補足情報で，次の形式である。

(name)　name=ライブラリ名称またはプログラム名称

(type)　type=変換時のデータ項目属性

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

Java プログラムで設定している数字データおよびライブラリ名称，プログラム名称を見直して，再実行する。または，「COBOL で作成したプログラム」で設定しているデータを見直して，再実行する。

## J2CByyy4003

Java 配列が生成できませんでした。

要因

「COBOL で作成したプログラム」で設定された引数データを参照するために，Java 配列を作成しようとしたが，作成できなかった。要因として，メモリ不足が考えられる。

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

十分な空きメモリ領域を確保して，再実行する。

## J2CByyy4010

Java 提供する関数で例外が発生しました。メモリ不足が考えられます。*** 1 ***

要因

Java が提供する関数で例外が発生した。メモリ不足が考えられる。

*** 1 ***：発生場所を示す内部情報

(S)

プログラムの実行を中止し，例外を発生させる。

(P)
　十分な空きメモリ領域を確保して，再実行する。

## J2CByyy4100

不正なエンコードが指定されています。

要因
　encode で不当な指定をした。

(S)
　プログラムの実行を中止し，例外を発生させる。

(P)
　正しい encode 指定をして，再実行する。

## J2CByyy4101

IO エラーが発生しました。

要因
　ワークストリームアクセス時にエラーとなった。

(S)
　プログラムの実行を中止し，例外を発生させる。

(P)
　エラー発生場所（メソッド）コードを控えて，保守員に連絡する。

## J2CByyy5001

引数情報出力処理中にメモリ不足が発生しました。

要因
　使用可能なメモリが不足している。

(S)
　プログラムの実行を中止し，例外を発生させる。

(P)
　十分な空きメモリ領域を確保して，再実行する。

## J2CByyy5002

引数情報出力ファイル作成時にエラーが発生しました。*** 1 ***
*** 1 ***:例外情報

要因

引数情報出力ファイル作成時にエラーが発生した。読み込み権限がないことが考えられる。

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

例外情報を基に，ファイルの書き込み権限などを見直して再実行する。

## J2CByyy5003

引数情報出力処理中に I/O エラーが発生しました。*** 1 ***
*** 1 ***:例外情報

要因

引数情報出力処理中に例外が発生した。

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

例外情報を基にエラー原因を修正して再実行する。

## J2CByyy5004

引数情報出力処理中にエラーが発生しました。*** 1 ***
*** 1 ***:例外情報

要因

引数情報出力処理中に例外が発生した。

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

例外情報を基にエラー原因を修正して再実行する。

## J2CByyy9999

予期しないエラーが発生しました。*** 9 ***

要因

予期しないエラーが発生した。

*** 9 ***：「例外の種類を識別する」値

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

エラー発生場所（メソッド）コードと「例外の種類を識別する」値を控えて，保守員に連絡する。

### J2CByyynnnn（上記一覧にない番号）

メッセージ取得中にエラーが発生しました。

要因

メッセージ取得中にエラーが発生した。「Cosminexus 連携機能の実行時ライブラリ」と「TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2」が一つのコンピュータにインストールされているが，バージョンが異なっている。

（S）

プログラムの実行を中止し，例外を発生させる。

（P）

上記 2 製品で提供している jar ファイル（j2cbrun.jar と j2tp1run.jar）の指定を，バージョンが新しいものから指定する。または，上記 2 製品のバージョンを合わせる。

# 付録 F　プログラム例

COBOL アクセスを使用した場合の基本的なプログラム例を示します（COBOL 引数の登録集原文から生成した Bean の生成例は，「付録 C　COBOL アクセス用 Bean の自動生成ソースイメージ」を参照してください）。

## 付録 F.1　JavaBean 版

### (1)　COBOL 引数の登録集原文と COBOL プログラム例

```
COBOL引数の登録集原文例
       01 PERSONAL-DATA.
         05 P-NUMBER   PIC 9(9) USAGE COMP.
         05 P-NAME     PIC X(50).
         05 P-ADDRESS  PIC X(100).
         05 P-GIF      PIC X(50).

COBOLプログラム例
      *>*****************************************************
      *>    COBOLサンプルプログラム(Search)                *
      *>*****************************************************
       IDENTIFICATION  DIVISION.
       PROGRAM-ID.     SEARCHCBL.
       DATA            DIVISION.
       LINKAGE SECTION.
         COPY SEARCHCBLCOPY.
      *>  01 PERSONAL-DATA.
      *>    05 P-NUMBER   PIC 9(9) USAGE COMP.
      *>    05 P-NAME     PIC X(50).
      *>    05 P-ADDRESS  PIC X(100).
      *>    05 P-GIF      PIC X(50).

       PROCEDURE DIVISION USING PERSONAL-DATA.

      *>   検索処理
           IF P-NUMBER=100001 THEN
             MOVE '日立 一郎' TO P-NAME
             MOVE '日立市' TO P-ADDRESS
             MOVE '/ICHIRO.GIF' TO P-GIF
           ELSE
             IF P-NUMBER=100002 THEN
               MOVE '日立 二郎' TO P-NAME
               MOVE '久留米市' TO P-ADDRESS
               MOVE '/JIRO.GIF' TO P-GIF
             ELSE
               MOVE '登録されていません' TO P-NAME
               MOVE SPACE TO P-ADDRESS
               MOVE '/NOREGIST.GIF' TO P-GIF
             END-IF
           END-IF.
           EXIT  PROGRAM.
```

```
       END PROGRAM  SEARCHCBL.
```

## (2)　Servlet(Java UAP)例

```
package test;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.beans.Beans;
import jp.co.hitachi_sk.j2cb.*;

public class SearchCBLServlet extends HttpServlet {
  private static final long serialVersionUID = 0L;
  ServletContext c;

  //グローバル変数の初期化
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
    c=config.getServletContext();
  }

  //HTTP Get リクエストの処理
  public void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {
    String number="";
    try {
      number=req.getParameter("Number");
    } catch (Exception e) {
      e.printStackTrace();
    }

    res.setContentType("text/html; charset=shift_jis");

    SearchCBL bean=null;
    try {
      //Beanのインスタンス生成
      bean=(SearchCBL) Beans.instantiate(
                           this.getClass().getClassLoader(),
                           "test.SearchCBL");
    } catch (ClassNotFoundException excp) {
      excp.printStackTrace();
      return;
    }

    try {

      /* Beanにパラメタを設定 */
      try {
        bean.setP_number(new Integer(number));
      } catch (NumberFormatException e) {
        bean.setP_number(new Integer(0));
      }
      bean.setP_name("");
      bean.setP_address("");
```

```
      bean.setP_gif("");

      bean.callCOBOL();

      //JSP中で"bean"という名称でプロパティを参照できるようにする。
      req.setAttribute("bean",bean);
      //ここでJSPを呼びます
      javax.servlet.RequestDispatcher rd =
                c.getRequestDispatcher("/SearchCBL.jsp");
      rd.forward(req, res);

    } catch (J2CBException e) {
      e.printStackTrace();
      return;

    }
}


  //HTTP Post リクエストの処理
  public void doPost(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
    doGet(request, response);
  }

}
```

## (3) HTML の作成例

```
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=shift_jis">
    <TITLE>
      Search
    </TITLE>
  </HEAD>
  <BODY>
    <h3><center>COBOL adapter for Cosminexusのサンプル</center></h3>
    <center>100001 と 100002が登録されています。
      <FORM  action=/servlet/test.SearchCBLServlet method="POST">
        Number : <input type="text" name="Number" value="100001">
        <BR><BR> Submit を押すと servlet SearchCBLServlet を実行
        <BR><BR><input type=submit value="Submit"></form>
    </center>
  </BODY>
</HTML>
```

## (4) JSP の作成例

```
<%-- This is a JSP of the SEARCH CBL sample. --%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<%@ page contentType="text/html; charset=shift_jis"
                    import="jp.co.hitachi_sk.j2cb.J2CBException" %>
<jsp:useBean id="bean" scope="request" class="test.SearchCBL" />
```

```
<html>
<center>Result of Search.
<table border="1">
<tr><td colspan="2" align="center">
     <jsp:getProperty name="bean" property="p_number" />
</td><tr><td>Name</td><td>
     <jsp:getProperty name="bean" property="p_name" />
</td><tr><td>Address</td><td>
     <jsp:getProperty name="bean" property="p_address" />
<tr><td colspan="2" align="center">
<img src="../servlet<jsp:getProperty name="bean" property="p_gif" />">
</td></table>
</center>
</html>
```

# 付録 F.2　EJB 対応版

## (1)　COBOL 引数の登録集原文と COBOL プログラム例

前述の JavaBean 版と同じです。「付録 F.1　JavaBean 版」を参照してください。

## (2)　Servlet(Java UAP)例

```
package testejb;

// import定義
import javax.servlet.*;
import javax.servlet.http.*;
import javax.naming.*;
import javax.rmi.*;
import java.io.*;
import java.util.*;
import javax.transaction.UserTransaction;
import javax.ejb.CreateException;
import java.rmi.*;
import java.lang.RuntimeException.*;
import jp.co.hitachi_sk.j2cb.*;

//* EJB パッケージ名 *
import testejb.*;

public class SearchEJBServlet extends HttpServlet {

  // ********************************
  // *  ホームインタフェース定義
  // ********************************
  private static final long serialVersionUID = 0L;
  SearchEJBHome   homeobj;

  String wkstr;
  boolean flgget=true;
  ServletContext  gservCont=null;
```

```
  UserTransaction tx;

  public void init(ServletConfig config) throws ServletException {
    Context context;
    try {
      gservCont=config.getServletContext();
      java.util.Properties prop=new java.util.Properties();
      prop.setProperty(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
                       "com.hitachi.software.ejb.jndi.InsContextFactory");
      prop.setProperty(javax.naming.Context.OBJECT_FACTORIES,
                       "com.hitachi.software.ejb.jndi.InsNamingFactory");
      prop.setProperty(javax.naming.Context.STATE_FACTORIES,
                       "com.hitachi.software.ejb.jndi.InsNamingFactory");
      prop.setProperty(javax.naming.Context.PROVIDER_URL, "iiopname://localhost:900");
      javax.naming.Context ctx=new javax.naming.InitialContext(prop);


  //  ************************************************
  //  * ctx.lookup("HITACHI_EJB//SERVERS//**1**//EJB//**2**//**3**.**4**");
  //  * **1** : サーバ名(servername)
  //  * **2** : アプリケーション名
  //  * **3** : リモート名
  //  * **4** : ホーム名
  //  *
  //  * homeobj=(**5**)PortableRemoteObject.narrow(objref,**5**.class);
  //  * **5** : ホーム名
  //  *
  //  ************************************************
  //  * 注意 : lookupの**1**(servername)は環境に合わせて変更してください
  //  ************************************************

      Object objref =
       ctx.lookup("HITACHI_EJB/SERVERS/servername/EJB/SearchEJBBean/SearchEJB.SearchEJBHome"
);
      homeobj=(SearchEJBHome)PortableRemoteObject.narrow(objref,SearchEJBHome.class);

    } catch(Exception e) {
      e.printStackTrace();
      return;
    }
}

  // Post の処理
  public void doPost(HttpServletRequest req,
                  HttpServletResponse res)
                  throws ServletException, IOException {
    flgget=false;
    doGetPost(req,res);
  }

  // Get の処理
  public void doGet(HttpServletRequest req,
                  HttpServletResponse res)
                  throws ServletException, IOException {
    flgget=true;
    doGetPost(req,res);
  }
```

```
  // GetPost の処理
  public void doGetPost(HttpServletRequest req,
                HttpServletResponse res)
                throws ServletException, IOException {

  // ********************************
  // *  リモートインタフェース定義
  // ********************************



    SearchEJB       remoteobj;

    // Htmからのデータ取得
    String number="";
    try {
      number=req.getParameter("Number");
    } catch (Exception e) {
      System.out.println("htm入力データエラー発生 :");
      e.printStackTrace();
      return;
    }


    // ****************************************************
    // * 文字エンコーディングの設定とそのライタの取得
    // ****************************************************

    res.setContentType("text/html; charset=shift_jis");
    PrintWriter out=res.getWriter();
    // HTMLを書き出す
    out.println("<html><head>");
   out.println("<meta http-equiv=¥"Content-Type¥" content=¥"text/html; charset=ISO-2022-JP¥"
>");
    out.println("<title>Test- (EJB呼び出し)-</title></head>");
    out.println("<body><center><h1>EJB動作確認</h1><br>");

    // EJBをCreate
    try {
      remoteobj=homeobj.create();
    } catch (Exception e) {
      J2CBErrorUtil wkobj=new J2CBErrorUtil(e.getMessage());
      out.println("Create時エラー発生 :");
      out.println("<p>getEJBCblErrorCode=" + wkobj.getEJBCblErrorCode() + "</p>");
      out.println("<p>getEJBCblMessageID=" + wkobj.getEJBCblMessageID() + "</p>");
      out.println("<p>getEJBDetailMessage=" + wkobj.getEJBDetailMessage() + "</p>");
      out.println("<p>getEJBErrorCode=" + wkobj.getEJBErrorCode() + "</p>");
      out.println("<p>getEJBMessage=" + wkobj.getEJBMessage() + "</p>");
      out.println("<p>getEJBName=" + wkobj.getEJBName() + "</p>");
      e.printStackTrace();
      return;
    }

    try {
      // Beanにパラメタを設定
      try {
        remoteobj.setP_number(new Integer(number));
      } catch (NumberFormatException e) {
```

```
      out.println("入力データエラー発生 :" + e.getMessage());
      e.printStackTrace();
      return;
    }
    remoteobj.setP_name("");
    remoteobj.setP_address("");
    remoteobj.setP_gif("");
  } catch(Exception e) {
    out.println("setBean エラー発生");
    out.println("Error MSG : " + e.getMessage());
    return;
  }

  // COBOLプログラムの呼び出し
  try {
    remoteobj.callCOBOL();
  } catch (Exception e) {
   out.println("COBOLプログラム呼び出し時 エラー発生 :");
    J2CBErrorUtil wkobj=new J2CBErrorUtil(e.getMessage());
    out.println("<p>getEJBCblErrorCode=" + wkobj.getEJBCblErrorCode() + "</p>");
    out.println("<p>getEJBCblMessageID=" + wkobj.getEJBCblMessageID() + "</p>");
    out.println("<p>getEJBDetailMessage=" + wkobj.getEJBDetailMessage() + "</p>");
   out.println("<p>getEJBErrorCode=" + wkobj.getEJBErrorCode() + "</p>");
    out.println("<p>getEJBMessage=" + wkobj.getEJBMessage() + "</p>");
    out.println("<p>getEJBName=" + wkobj.getEJBName() + "</p>");
    e.printStackTrace();
    return;
  }
  // データの出力
  try {
    out.println("<center>Result of Search.");
   out.println("<table border='1'>");
    out.println("<tr><td colspan='2' align='center'>");
    out.println( remoteobj.getP_number() );
    out.println("</td><tr><td>Name</td><td>");
    out.println( remoteobj.getP_name() );
   out.println("</td><tr><td>Address</td><td>");
    out.println( remoteobj.getP_address() );
    out.println("<tr><td colspan='2' align='center'><img src='../servlet");
   out.println(remoteobj.getP_gif());
    out.println("'>");
    out.println("<tr>");
    out.println("</table>");
    out.println("</center>");
  } catch(Exception e) {
    out.println("getBean時 エラー発生");
   J2CBErrorUtil wkobj=new J2CBErrorUtil(e.getMessage());
    out.println("<p>getEJBCblErrorCode=" + wkobj.getEJBCblErrorCode() + "</p>");
    out.println("<p>getEJBCblMessageID=" + wkobj.getEJBCblMessageID() + "</p>");
    out.println("<p>getEJBDetailMessage=" + wkobj.getEJBDetailMessage() + "</p>");
    out.println("<p>getEJBErrorCode=" + wkobj.getEJBErrorCode() + "</p>");
   out.println("<p>getEJBMessage=" + wkobj.getEJBMessage() + "</p>");
    out.println("<p>getEJBName=" + wkobj.getEJBName() + "</p>");
    e.printStackTrace();
    return;
  }

  out.println("</body></html>");
```

```
    out.close();
  }

}
```

## (3) HTML の作成例

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=shift_jis">
<TITLE>
SearchEJB
</TITLE>
</HEAD>
<BODY>
<h3><center>COBOL adapter for Cosminexusのサンプル<BR>（EJB版）</center></h3>
<hr><br>
<center>100001 と 100002が登録されています。
<FORM  action=/servlet/testejb.SearchEJBServlet method="POST">
Number：<input type="text" name="Number" value="100001">
<BR><BR> Submit を押すと servlet SearchEJBServlet を実行
<BR><BR><input type=submit value="Submit"></form>
</center>
</BODY>
</HTML>
```

# 付録 F.3　OCCURS 句を使用した例

JavaBean 版と EJB 対応版で共通な OCCURS 句指定時の例を該当部分だけ抜き出して記載します。

## (1) COBOL 引数の登録集原文と COBOL プログラム例

```
COBOL引数の登録集原文例
       01 G1.
        02 G2  OCCURS 10.
         03 B1         PIC X(50).
COBOLプログラム例
         :
       PROCEDURE DIVISION USING ...

      *> 検索処理
           IF B1(1)=SPACE THEN
              :
           END-IF
              :
           MOVE 'TEST OK' TO B1(10).
              :
```

## (2)　Servlet(Java UAP)例

```
    :
    int i=0;
    for ( i=0; i < 10; i++ ) {          [B1(0)～B1(9)へ”XXXXX”を設定]
      bean.setB1(”XXXXX”, i);             [JavaBean版]
      remoteobj.setB1(”XXXXX”, i);        [EJB対応版]
    }
    :                                     [B1(5)を取得]
    String wkstr=bean.getB1(5);         [JavaBean版]
    String wkstr=remoteobj.getB1(5);    [EJB対応版]
    :
```

# 付録 F.4　可変長データを使用した例

JavaBean 版と EJB 対応版で共通な OCCURS 句指定時の例を該当部分だけ抜き出して記載します。

## (1)　COBOL 引数の登録集原文と COBOL プログラム例

```
COBOL引数の登録集原文例
       01 XML-DATA     PIC X(10000004).
       01 XML-DATA-R   REDEFINES XML-DATA.
         05 XML-LEN    PIC S9(9) USAGE COMP.     [可変長データの長さ部分]
         05 XML-ITEM   PIC X(10000000).     [可変長データ自身]

COBOLプログラム例
         :
       PROCEDURE DIVISION USING ...

      *> 検索処理
           IF XML-LEN > ZERO THEN
             MOVE XML-ITEM(1:XML-LEN) TO WK
              :
           END-IF
```

## (2)　Servlet(Java UAP)例

```
 :
byte[] sdata=new byte[10000000];
 :  (sdataにバイト配列データを設定)       [50バイトのデータを設定]
bean.setXml_data(sdata, 50);              [JavaBean版]
remoteobj.setXml_data(sdata, 50);         [EJB対応版]
}
 :                                          [データを取得する]
byte[] rdata=bean.getXml_data();         [JavaBean版]
byte[] rdata=remoteobj.getXml_data();    [EJB対応版]
 :
```

## 付録 F.5　アドレスデータを使用した例

JavaBean 版と EJB 対応版で共通なアドレスデータ指定時の例を，該当部分だけ抜き出して記載します。

### (1)　COBOL 引数の登録集原文と COBOL プログラム例

```
COBOL引数の登録集原文例
       01 XML-POINTER     USAGE ADDRESS.

COBOLプログラム例
         :
       WORKING-STORAGE SECTION.
       01 XML-DATA   ADDRESSED BY XML-DATA-ADDR.
         05 XML-LEN    PIC S9(9)  USAGE COMP.     [可変長データの長さ部分]
         05 XML-ITEM   PIC X(10000000).     [可変長データ自身]
          :
       PROCEDURE DIVISION USING BY VALUE XML-POINTER.
          :
       COMPUTE XML-DATA-ADDR=XML-POINTER.   [受け取ったアドレスを設定]
           IF XML-LEN > ZERO THEN
             MOVE XML-ITEM(1:XML-LEN) TO WK
              :
           END-IF
```

### (2)　Java UAP(Servlet)例

```
 :
byte[] sdata=new byte[20000000];
 :  (sdataにバイト配列データを設定)       [50バイトのデータを設定]
bean.setXml_pointer(sdata, 50);                [JavaBean版]
    remoteobj.setXml_pointer(sdata, 50);       [EJB対応版]
    }
     :                                        [データを取得する]
    byte[] rdata=bean.getXml_pointer();      [JavaBean版]
    byte[] rdata=remoteobj.getXml_pointer(); [EJB対応版]
     :
```

# 付録 G　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 G.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　（4010-1J-800）
- COBOL2002　操作ガイド　（4010-1J-801）
- COBOL2002　使用の手引　手引編　（4010-1J-802）
- COBOL2002　使用の手引　操作編　（4010-1J-803）
- COBOL2002　言語　標準仕様編　（4010-1J-804）
- COBOL2002　言語　拡張仕様編　（4010-1J-805）
- COBOL2002　Java プログラム呼び出し機能ガイド　（4010-1J-807）
- COBOL2002　メッセージ　（4010-1J-809）
- Cosminexus V9 アプリケーションサーバ　システム構築・運用ガイド　（3020-3-Y02）
- Cosminexus V9 アプリケーションサーバ　リファレンス　コマンド編　（3020-3-Y15）
- Cosminexus V9 アプリケーションサーバ　リファレンス　定義編（サーバ定義）（3020-3-Y16）
- Cosminexus V9 アプリケーションサーバ　アプリケーション開発ガイド　（3020-3-Y20）
- Cosminexus V11 アプリケーションサーバ　システム構築・運用ガイド　（3021-3-J02）
- Cosminexus V11 アプリケーションサーバ　リファレンス　コマンド編　（3021-3-J15）
- Cosminexus V11 アプリケーションサーバ　リファレンス　定義編（サーバ定義）（3021-3-J16）
- Cosminexus V11 アプリケーションサーバ　アプリケーション開発ガイド　（3021-3-J20）

なお，このマニュアルでは，次の Cosminexus 製品を前提に記述しています。

- Windows(x86)の場合
  - Cosminexus Version 9
  - Cosminexus Version 11
- Windows(x64)，Linux(x64)の場合
  - Cosminexus Version 11

### このマニュアルでのマニュアル名の表記

このマニュアルでは，Cosminexus のマニュアルを次のように表記しています。

| 表記 | マニュアル名 |
|---|---|
| Cosminexus　システム構築・運用ガイド | • Cosminexus V9 アプリケーションサーバ　システム構築・運用ガイド<br>• Cosminexus V11 アプリケーションサーバ　システム構築・運用ガイド |
| Cosminexus　リファレンス　コマンド編 | • Cosminexus V9 アプリケーションサーバ　リファレンス　コマンド編<br>• Cosminexus V11 アプリケーションサーバ　リファレンス　コマンド編 |
| Cosminexus　リファレンス　定義編（サーバ定義） | • Cosminexus V9 アプリケーションサーバ　リファレンス　定義編（サーバ定義）<br>• Cosminexus V11 アプリケーションサーバ　リファレンス　定義編（サーバ定義） |
| Cosminexus　アプリケーション開発ガイド | • Cosminexus V9 アプリケーションサーバ　アプリケーション開発ガイド<br>• Cosminexus V11 アプリケーションサーバ　アプリケーション開発ガイド |

## 付録 G.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 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 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 |

このマニュアル中では，「COBOL2002 Cosminexus 連携機能」を「このシステム」または「COBOL アクセス」と表記しています。また，「Windows」と表現している場合は「Windows COBOL2002 Cosminexus 連携機能」を，「UNIX」と表現している場合は「AIX COBOL2002 Cosminexus 連携機能」，「Linux COBOL2002 Cosminexus 連携機能」を示しています。

また，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-2636-2354<br>P-2436-5354<br>P-2436-6354<br>P-2636-4354<br>P-2636-7354 |
| | Windows(x64) | | P-2936-2354<br>P-2936-5354<br>P-2936-6354<br>P-2936-7354 |
| UNIX | AIX | AIX(64) | －※ |
| | Linux | Linux(x64) | P-9W36-1251<br>P-9W36-2251 |

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。

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 |
|---|---|---|---|
| Cosminexus | Cosminexus Version 9 または Cosminexus Version 11 | Cosminexus Application Server | uCosminexus Application Server |
| | | | uCosminexus Service Platform |
| | | Cosminexus Developer | uCosminexus Developer |
| | | | uCosminexus Service Architect |
| UNIX | AIX | | AIX V7.1 |
| | | | AIX V7.2 |
| | | | AIX 7.3 |
| | 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) |

このマニュアルで使用している表記と，対応する Java 関連用語を次に示します。

| マニュアルでの表記 | Java 関連用語 |
|---|---|
| Enterprise JavaBeans または EJB | Enterprise JavaBeans |
| Java EE | Java Platform, Enterprise Edition |

| マニュアルでの表記 | Java 関連用語 |
|---|---|
| J2EE | Java 2 Platform, Enterprise Edition |
| J2SE | Java 2 Platform, Standard Edition |
| JAR | Java Archive |
| Java VM | Java Virtual Machine |
| JNDI | Java Naming and Directory Interface |
| JNI | Java Native Interface |
| JSP | JavaServer Pages |
| Servlet またはサーブレット | Java Servlet |
| WAR | Web ARchive |

- Cosminexus Component Container では，J2EE コンテナを生成・実行する環境を J2EE サーバと表記しています。
- Cosminexus Component Container の Web コンテナの動作モードがサーブレットエンジンモードの場合を，Web コンテナサーバと表記しています。
- 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 G.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 H　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。

### DD（Deployment Descriptor）

Enterprise Bean 作成に関する属性およびアプリケーション構築に関する属性を XML 形式で記述したものです。

### Entity Bean

永続的オブジェクトのことです。

### getter

プロパティから値を取得するメソッドです。

### 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。

### RMI-IIOP（Remote Method Invocation over Internet Inter-ORB Protocol）

JavaRMI と JavaIDL を統合した API です。EJB コンポーネントとそれぞれのコンテナは，Remote Method Invocation（RMI）テクノロジを使って，分散オブジェクト間のメソッド呼び出しを実装します。

### Servlet

サーバ・サイド・プログラムとして稼働する Java プログラムです。

### Session Bean

ユーザとのセッションごとに生成，消滅するオブジェクトです。

### setter

プロパティに値を設定するメソッドです。

### 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です。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。

### 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，半角かたかなは 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。

### WRP（Windows Resource Protection）

Windows システムの安定性，および信頼性を向上させるための機能です。特定の OS ファイル，フォルダ，レジストリ キーなど，Windows の読み取り専用リソースを保護します。

## （ア行）

### アプリケーションサーバ

Web での業務開発のためのアプリケーション基盤機能を提供する製品です。日立アプリケーションサーバ Cosminexus は，ブラウザなどのフロントエンド層と DB や既存システムなどのバックエンド層の間に位置づけられ，業務の開発から運用までに一貫した環境を提供します。

### ウィザード

パッケージ・ソフト製品に組み込まれるヘルプ機能の一種です。設定作業を支援する対話型のナビゲータ機能を一般にウィザードと呼びます。表や文書の書式設定やクロス集計表の作成など，細かなノウハウが求められる複雑なパラメタ設定作業を対話形式で誘導（ナビゲーション）します。

## （カ行）

### 管理者権限

管理者ユーザに与えられる権限です。製品のインストールなどに関する権限までは持ちますが，システムファイルの書き替えまでの権限は持ちません。

### 管理者ユーザ

Windows の Administrators グループに属するアカウントです。ビルトイン管理者（Administrator アカウント）とは区別されます。

## （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。

## （タ行）

### 登録集原文

COBOL アクセス用 Bean を生成するための入力情報です。COBOL アクセスの COBOL データ定義規則に従って作成した COBOL UAP のパラメタの情報です。

## （ハ行）

### バイトオーダ

2 バイト以上のデータの記録を行なう順序のことです。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いいます。2 バイトの UTF-16 は，バイトオーダを意識します。

### 標準権限

標準ユーザに与えられる権限です。故意に，または誤ってシステムに変更を加える許可を持ちません。

### 標準ユーザ

Windows の Users グループに属するアカウントです。

## （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