HITACHI
Inspire the Next

TP1/COBOL adapter for Cosminexus　ユーザーズガイド

手引・操作書

3020-3-B08-H0

前書き

■ 対象製品

P-2636-F124 TP1/COBOL adapter for Cosminexus Version 2 02-09-/A（適用 OS：Windows NT 4.0，Windows 2000，Windows XP，Windows Server 2003(x86)）

P-2436-G124 TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 02-09（適用 OS：Windows NT 4.0，Windows 2000 Server，Windows Server 2003）

P-2636-F324 TP1/COBOL adapter for Cosminexus Version 2 02-10（適用 OS：Windows 2000 ※1，Windows XP，Windows Server 2003(x86)，Windows Server 2003 R2(x86)，Windows Vista，Windows Server 2008，Windows Server 2008 R2，Windows 7，Windows Server 2012，Windows Server 2012 R2，Windows 8.1，Windows 10，Windows Server 2016，Windows Server 2019）

P-2936-F324 TP1/COBOL adapter for Cosminexus Version 2(64) 02-10（適用 OS：Windows 8.1(x64)，Windows 10(x64)）

P-2436-G324 TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 02-10（適用 OS：Windows 2000 Server，Windows Server 2003，Windows Server 2003 R2，Windows Server 2008，Windows Server 2008 R2，Windows Server 2012，Windows Server 2012 R2，Windows Server 2016，Windows Server 2019）

P-1B36-G121 TP1/COBOL 拡張 Run Time System for Cosminexus Version 2 02-08（適用 OS：HP-UX 11.0，HP-UX 11i，HP-UX 11i V2(PA-RISC)）

P-1J36-G221 TP1/COBOL 拡張 Run Time System for Cosminexus Version 2(64) 02-09（適用 OS：HP-UX 11i V2 サーバ(IPF)，HP-UX 11i V3 サーバ(IPF)）

P-1M36-G121 TP1/COBOL 拡張 Run Time System for Cosminexus Version 2 02-09（適用 OS：AIX 5L V5.1, 5L V5.2, 5L V5.3）

P-1M36-G321 TP1/COBOL 拡張 Run Time System for Cosminexus Version 2 02-10（適用 OS：AIX 5L V5.2, 5L V5.3, V6.1, V7.1, V7.2）

P-9S36-G121 TP1/COBOL 拡張 Run Time System for Cosminexus Version 2 02-08（適用 OS：Red Hat Enterprise Linux AS 3.0，Red Hat Enterprise Linux ES 3.0）

P-9S36-G321 TP1/COBOL 拡張 Run Time System for Cosminexus Version 2 02-10（適用 OS：Red Hat Enterprise Linux AS 4, Red Hat Enterprise Linux ES 4, Red Hat Enterprise Linux 5 Advanced Platform, Red Hat Enterprise Linux 5, Red Hat Enterprise Linux Server 6）

P-9S36-G331 TP1/COBOL 拡張 Run Time System for Cosminexus Version 2 02-20（適用 OS：Red Hat Enterprise Linux Server 7）

P-9V36-G221 TP1/COBOL 拡張 Run Time System for Cosminexus Version 2 02-08（適用 OS：Red Hat Enterprise Linux AS 3.0(IPF)）

P-2636-F114 TP1/COBOL adapter for Cosminexus 01-10（適用 OS：Windows 98，Windows Me，Windows NT 4.0，Windows 2000，Windows XP）

P-2436-G114 TP1/COBOL 拡張 Server Run Time System for Cosminexus 01-10（適用 OS：Windows NT 4.0 Server，Windows 2000 Server）

P-1B36-G111 TP1/COBOL 拡張 Run Time System for Cosminexus 01-10（適用 OS：HP-UX 11.0，HP-UX 11i，HP-UX 11i V2(PA-RISC)）

P-1M36-G111 TP1/COBOL 拡張 Run Time System for Cosminexus 01-10（適用 OS：AIX 5L V5.1）

P-9336-G111 TP1/COBOL 拡張 Run Time System for Cosminexus 01-08（適用 OS：Solaris 7，Solaris 8）

注※1　Service Pack 4 以降です。

■ 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

その他記載の会社名，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

2019 年 9 月　3020-3-B08-H0

変更内容

変更内容（3020-3-B08-H0）TP1/COBOL adapter for Cosminexus Version 2(64) 02-10，TP1/COBOL adapter for Cosminexus Version 2 02-10，TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 02-10

追加・変更機能	変更箇所
• TP1/COBOL adapter for Cosminexus Version 2 の 64bit 版に対応した。	－

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，次のプログラムプロダクトの機能，およびプログラム作成方法について説明したものです。

- P-2636-F124　TP1/COBOL adapter for Cosminexus Version 2
- P-2436-G124　TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2
- P-1B36-G121　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
- P-1M36-G121　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
- P-9S36-G121　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
- P-2636-F114　TP1/COBOL adapter for Cosminexus
- P-2436-G114　TP1/COBOL 拡張 Server Run Time System for Cosminexus
- P-1B36-G111　TP1/COBOL 拡張 Run Time System for Cosminexus
- P-1M36-G111　TP1/COBOL 拡張 Run Time System for Cosminexus
- P-9336-G111　TP1/COBOL 拡張 Run Time System for Cosminexus
- P-2636-F324　TP1/COBOL adapter for Cosminexus Version 2
- P-2936-F324 TP1/COBOL adapter for Cosminexus Version 2(64)
- P-2436-G324　TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2
- P-9V36-G221　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
- P-1J36-G221　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2(64)
- P-9S36-G321　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
- P-9S36-G331　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
- P-1M36-G321　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2

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。

■ 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。

■ マニュアルの構成

このマニュアルは，次に示す章と付録から構成されています。

第 1 編　TP1/COBOL adapter for Cosminexus Version 2 による開発と実行

第 1 章　TP1/COBOL adapter for Cosminexus の概要

TP1/COBOL adapter for Cosminexus とは何かについて説明しています。

第 2 章　Web アプリケーションの開発

TP1 サーバ用 Web アプリケーションの開発に必要な操作，およびプログラミング時の注意事項について説明しています。

第 3 章　プログラムのコンパイル

作成した TP1 サーバ用の COBOL SPP/MHP 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ています。

第 4 章　アプリケーションサーバの環境整備

作成した COBOL SPP/MHP と Java プログラムを実行するための実行環境変数の設定内容について説明しています。

第 5 章　COBOL SPP/MHP と TP1/COBOL アクセスの環境整備

TP1/COBOL アクセス環境下において，COBOL SPP/MHP の実行環境を設定するための環境変数を指定する方法について説明しています。

第 6 章　プログラムの実行

作成した COBOL SPP/MHP と Java プログラムを実行するためのファイルの配置と実行について説明しています。

第 7 章　プログラムのデバッグ

作成した COBOL SPP/MHP と Java プログラムのデバッグに関する操作および環境変数の設定内容について説明しています。

第 8 章　COBOL SPP/MHP を呼び出すための API

COBOL SPP/MHP を呼び出すための API（Application Program Interface）について説明しています。

第 9 章　Windows/UNIX 混在システム構築上の注意事項

TP1/COBOL アクセスを使ってシステムを構築する際，構成する OS が異なる場合の注意事項について説明しています。

第 10 章　TP1/COBOL SOAP サービスクラス生成機能

TP1/COBOL アクセスを使って OpenTP1 サーバ環境下の COBOL SPP を Cosminexus SOAP アプリケーションとして呼び出す方法について説明しています。

第2編　TP1/COBOL adapter for Cosminexus による開発と実行

第 11 章　Web アプリケーションの開発

TP1/COBOL adapter for Cosminexus 01-xx を使った Web アプリケーションの開発（プログラム作成からコンパイルまで）について説明しています。

第 12 章　アプリケーションサーバの環境整備

作成した COBOL SPP と Java プログラムを実行するための実行環境変数の設定内容について説明しています。

第 13 章　COBOL SPP と TP1/COBOL アクセスの環境整備

TP1/COBOL アクセス環境下において，COBOL SPP の実行環境を設定するための環境変数を指定する方法について説明しています。

第 14 章　プログラムの実行とデバッグ

作成した COBOL SPP と Java プログラムの実行とデバッグに関する操作および環境変数の設定内容について説明しています。

付録 A　注意事項 / 制限事項

COBOL SPP/MHP を呼び出すための，TP1/COBOL アクセス用 Bean および Servlet 作成時の注意事項と制限事項について説明しています。

付録 B　TP1/COBOL adapter for Cosminexus で使用するファイル

TP1/COBOL adapter for Cosminexus で使用するファイルの説明をしています。

付録 C　Windows 版の組込み方法

Windows 版での TP1/COBOL adapter for Cosminexus および TP1/COBOL 拡張 Server Run Time System for Cosminexus のインストール方法とアンインストール方法について説明しています。

付録 D　HP-UX/Solaris/AIX/Linux 版の組込み方法

HP-UX/Solaris/AIX/Linux 版での TP1/COBOL 拡張 Run Time System for Cosminexus のインストール方法とアンインストール方法について説明しています。

付録 E　TP1/COBOL アクセス用 Bean の自動生成ソースイメージ

TP1/COBOL アクセス用 Bean の自動生成ソースイメージを例を用いて説明しています。

付録 F　TP1/COBOL アクセス用 Bean 生成時に出力するメッセージ

TP1/COBOL adapter for Cosminexus が出力するメッセージ内容について説明しています。

付録 G　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ています。

付録 H　プログラム例

TP1/COBOL アクセスを使用した場合の基本的なプログラムを例を用いて説明しています。

付録 I　このマニュアルの参考情報

このマニュアルを読むに当たっての参考情報を示します。

付録 J　用語解説

このマニュアルで使用している主な用語について説明しています。

目次

1 TP1/COBOL adapter for Cosminexus の概要

第 1 編では，TP1/COBOL adapter for Cosminexus Version 2 および TP1/COBOL 拡張 [Server] Run Time System for Cosminexus Version 2 の説明を行います。バージョン 01-10 以前の説明については，第 2 編をご覧ください。

また，第 1 編では COBOL に関する記述は COBOL2002 を前提に記述しておりますので，COBOL85 をご使用の場合は置き換えてお読みください。

この章では，COBOL で作成した OpenTP1 の SPP（Service Providing Program： OpenTP1 サービス提供プログラム）または MHP（Message Handling Program： OpenTP1 メッセージ処理専用プログラム）をアプリケーションサーバ Cosminexus から利用した場合の TP1/COBOL adapter for Cosminexus の特長と構成について説明します。

1.1 TP1/COBOL adapter for Cosminexus の特長

- Java アプリケーションプログラムと OpenTP1 の COBOL で作成したサービス提供プログラム（SPP）およびメッセージ処理専用プログラム（MHP）との通信を可能にしています。
- Cosminexus の環境から OpenTP1 を経由した他システムへの接続や，OpenTP1 上の業務システムへの接続のために使用します。
- TP1/COBOL adapter for Cosminexus が提供するライブラリ（以下 TP1/COBOL アクセス用ライブラリ）と開発ツールを用いて作成する TP1/COBOL アクセス用 Bean を使用することによって，既存の SPP/MHP に修正を加えることなく Servlet や JSP（Java Server Pages）等の Java UAP（User Application Program）から通信することができます。
- TP1/COBOL アクセスの機能を経由して，OpenTP1 サーバ環境下の COBOL SPP を Cosminexus SOAP アプリケーションとして呼び出すことも可能です。Cosminexus SOAP 連携機能については「10. TP1/COBOL SOAP サービスクラス生成機能」を参照してください。ただし，Windows 版 02-10 以降，Linux(IPF)版，HP-UX(IPF) 版，Linux(x86/x64)版 02-10 以降，および AIX 版 02-10 以降では本機能は使用できませんので，ご注意ください。

1.2 OpenTP1 および Cosminexus での位置づけ

TP1/COBOL アクセスは，アプリケーション・サーバである OpenTP1 のサーバ用構成製品です。

TP1/COBOL アクセスは，TP1/Client をラッピングすることで，Java UAP と OpenTP1 システムとの通信を実現します。TP1/COBOL アクセスの OpenTP1 での位置づけを図 1-1 に示します。

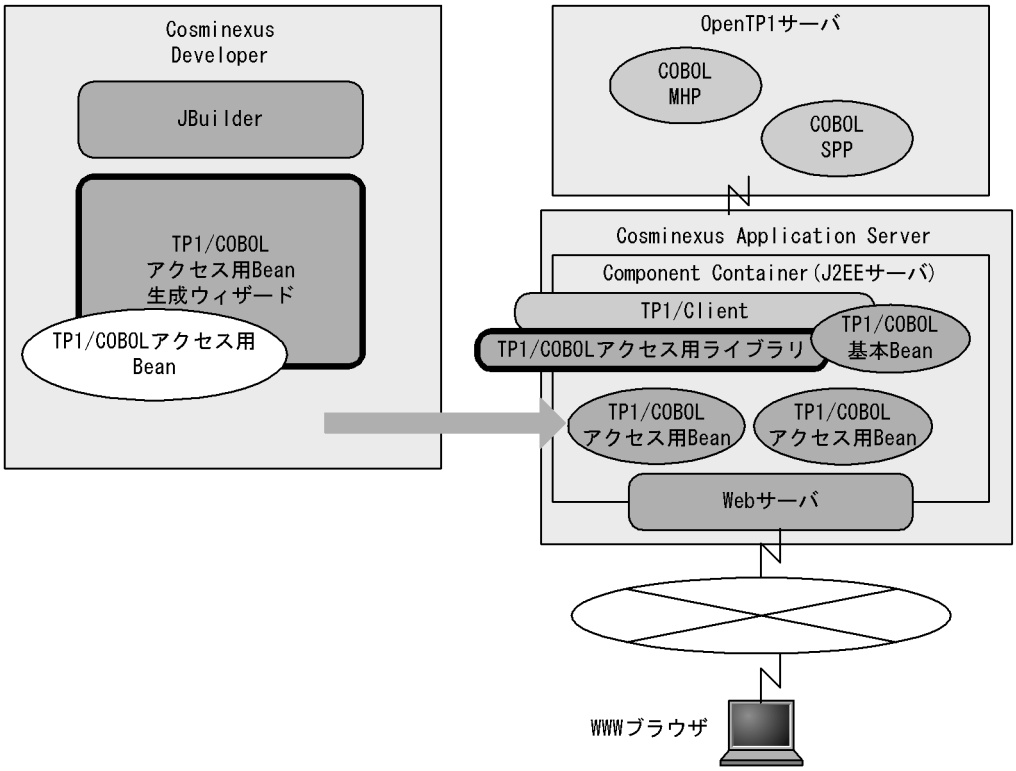
TP1/COBOL アクセス用 Bean 生成ツール，もしくは TP1/COBOL アクセス用 Bean 生成ウィザードは，開発環境で TP1/COBOL アクセス用 Bean を開発するときに，Java 環境から TP1/Client を経由して SPP にアクセスするための TP1/COBOL アクセス用 Bean を生成するウィザードを提供して，簡単に開発できるよう支援します。

TP1/COBOL アクセス用ライブラリは，TP1/COBOL にアクセスする Java UAP のリクエストを TP1/Client および Cosminexus TP1 Connector を経由して TP1 サーバへリモート通信させるための実行環境です。

Java UAP は TP1/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 SPP との通信を実現することができます。

また，MHP に対しては，TP1/COBOL 基本 Bean および TP1/COBOL アクセス用 Bean（TP1/Client/P，TP1/Client/W および TP1/Client/J 用）で提供するメソッドで通信を実現することができます。

図 1-1 TP1/COBOL アクセスの位置づけ



1.3　TP1/COBOL アクセスの概要

TP1/COBOL アクセスの概要について示します。

1.3.1　TP1/COBOL アクセスの処理概要

TP1/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。

TP1/COBOL アクセスは，TP1/Client/P または TP1/Client/W経由のアクセス，TP1/Client/J 経由のアクセスおよび Cosminexus TP1 Connector 経由のアクセス三つを用意しています。ただし，Windows(64bit)版，Linux(IPF)版，HP-UX(IPF) 版，Linux(x86/x64)版 02-10 以降，および AIX 版 02-10 以降では TP1/Client/P または TP1/Client/W経由のアクセスは機能提供していませんので，ご注意ください。Windows(64bit)版，Linux(IPF)版，HP-UX(IPF) 版，Linux(x86/x64)版 02-10 以降，および AIX 版 02-10 以降では TP1/Client/J 経由のアクセスまたは Cosminexus TP1 Connector 経由のアクセスをご使用ください。

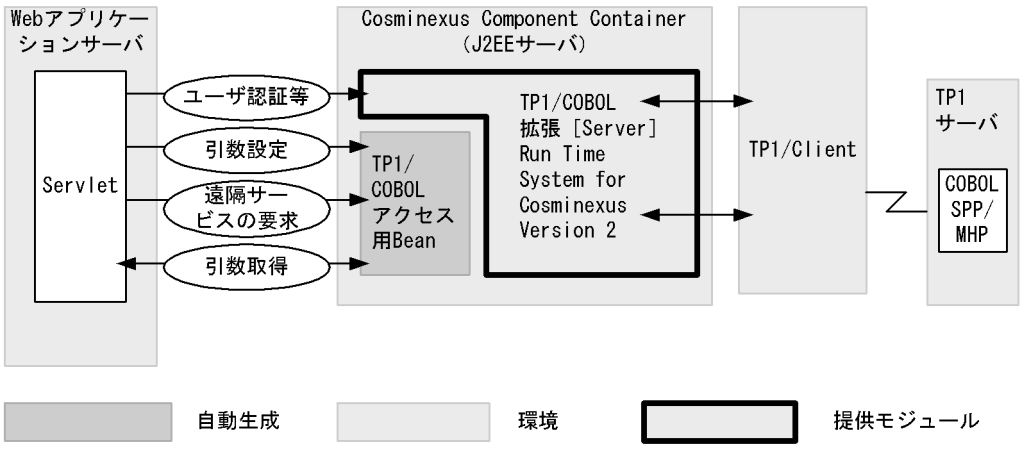
(1)　TP1/Client/P または TP1/Client/W 経由のアクセス

TP1/COBOL アクセスは，OpenTP1 の SPP/MHP との通信を可能にする API を Cosminexus 上の Java UAP のために JavaBeans として提供し，Java UAP からのリクエストを内部で TP1/Client のユーザ認証機能，RPC（Remote Procedure Call），ネームサービスを使用しない RPC，トランザクションを管理するプロセスを割り当てる RPC を実現します。

TP1/Client を動作させるためのクライアント環境定義値の設定は，ユーザが行う必要があります。クライアント環境定義の詳細は，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」をご覧ください。

図 1-2 に TP1/COBOL アクセスの処理概要を示します。

図 1-2　TP1/COBOL アクセスの処理概要



Servlet（Java UAP）は，TP1/COBOL 基本 Bean や TP1/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。

COBOL MHP に対する send/receive/receive2 メソッドは，COBOL SPP に対する RPC メソッドと同様，TP1/COBOL アクセス用 Bean で入力データを設定することができます。

1. TP1/COBOL 基本 Bean で提供するユーザ認証，UAP 開始，常時接続等の，OpenTP1 の各メソッド規定順呼び出し。
2. 入力データの設定は任意の基本項目に対して，生成した TP1/COBOL アクセス用 Bean の各 setter メソッドの呼び出し。
3. 生成した TP1/COBOL アクセス用 Bean の RPC メソッドの呼び出し，または，send/receive/receive2 メソッドの呼び出し。
4. 生成した TP1/COBOL アクセス用 Bean データの参照時の，各 getter メソッドの呼び出し。
5. TP1/COBOL 基本 Bean で提供する接続解除，UAP 終了，ユーザ認証解除などの OpenTP1 の各メソッド規定順呼び出し。

TP1/COBOL 基本 Bean で提供する send/receive/receive2 メソッドで呼び出す COBOL MHP に対するアクセスについて，以下に示します。

1. TP1/COBOL 基本 Bean で提供するユーザ認証，UAP 開始，常時接続等の，OpenTP1 の各メソッド規定順呼び出し。
2. 送信メッセージ（String 型）の設定，または受信メッセージ（String 型）領域の定義。
3. TCP/IP 通信メソッドの呼び出し。
4. 受信メッセージの参照。
5. TP1/COBOL 基本 Bean で提供する接続解除，UAP 終了，ユーザ認証解除などの OpenTP1 の各メソッド規定順呼び出し。

(2) TP1/Client/J 経由のアクセス

TP1/COBOL アクセスは，OpenTP1 の SPP/MHP との通信を可能にする API を Cosminexus 上の Java UAP(Servlet や EJB など)のために JavaBeans として提供し，Java UAP からのリクエストを内部で TP1/Client/J の RPC（Remote Procedure Call）を実現します。常時接続，接続解除等の TP1/Client/J が提供しているメソッド（RPC 以外）については，そのまま使用します。

TP1/Client/J を動作させるためのクライアント環境定義値の設定は，ユーザが行う必要があります。クライアント環境定義の詳細は，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

Java UAP は，TP1/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. TP1/Client/J が提供する常時接続等の OpenTP1 の各メソッド規定順呼び出し。

2. 入力データの設定は任意の基本項目に対して，生成した TP1/COBOL アクセス用 Bean の各 setter メソッドの呼び出し。
3. 生成した TP1/COBOL アクセス用 Bean の RPC メソッドの呼び出し，または，send/receive メソッド呼び出し。
4. 生成した TP1/COBOL アクセス用 Bean データの参照時の，各 getter メソッドの呼び出し。
5. TP1/Client/J が提供する接続解除等の OpenTP1 の各メソッド規定順呼び出し。

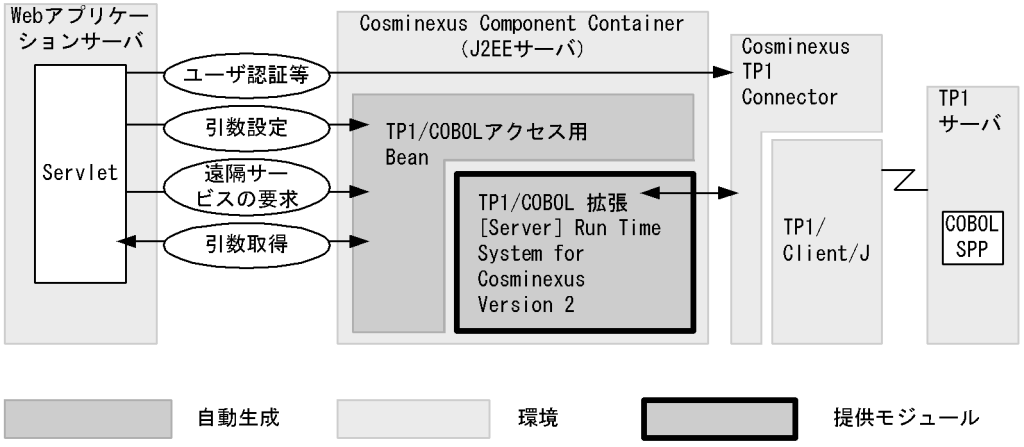
(3) Cosminexus TP1 Connector 経由のアクセス

TP1/COBOL アクセスは，OpenTP1 の SPP/MHP との通信を可能にする API を Cosminexus 上の Java UAP のために JavaBeans として提供します。Java UAP からのリクエストは，内部から Cosminexus TP1 Connector が提供する CCI（Common Client Interface）を使って実現します。Cosminexus TP1 Connector を経由することでコネクションプーリング機能を使用できます。

Java UAP は，TP1/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. Cosminexus TP1 Connector が提供する接続等の OpenTP1 の各メソッド規定順呼び出し。
2. 入力データの設定は任意の基本項目に対して，生成した TP1/COBOL アクセス用 Bean の各 setter メソッドの呼び出し。
3. 生成した TP1/COBOL アクセス用 Bean の CCI メソッドの呼び出し。
4. 生成した TP1/COBOL アクセス用 Bean データの参照時の，各 getter メソッドの呼び出し。
5. Cosminexus TP1 Connector が提供する接続解除等の OpenTP1 の各メソッド規定順呼び出し。

図 1-3 Cosminexus TP1 Connector 経由のアクセスの処理概要



1.3.2 TP1/COBOL アクセスが提供する機能

(1) TP1/Client/P または TP1/Client/W経由のアクセス

TP1/COBOL アクセスが提供するモジュールは，TP1/Client を呼び出す Java UAP と TP1/Client 間のインタフェースプログラムと，そのインタフェースプログラムと Java を結ぶ JNI（Java Native Interface）プログラムです。TP1/COBOL アクセス用 Bean は使用する SPP/MHP ごとに TP1/COBOL アクセス用 Bean 生成ツール，もしくは TP1/COBOL アクセス用 Bean 生成ウィザードを用いて利用者が作成します。

TP1/COBOL アクセスで使用できるのは，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」の「TP1/Client で使用できる関数（C 言語編）」の章で提供される関数（ただし，XATMI インタフェース機能を除く）です。

TP1/COBOL アクセスは，これらの関数をラッピングし，TP1/Client と同様のリクエストを Java UAP から発行するインタフェースを提供します。

(2) TP1/Client/J 経由または Cosminexus TP1 Connector 経由のアクセス

TP1/COBOL アクセスが提供するモジュールは，TP1/Client/J または Cosminexus TP1 Connector を呼び出す Java プログラムだけで構成されています。TP1/COBOL アクセス用 Bean は使用する SPP/MHP ごとに TP1/COBOL アクセス用 Bean 生成ツール，もしくは TP1/COBOL アクセス用 Bean 生成ウィザードを用いて利用者が作成します。ここで作成した TP1/COBOL アクセス用 Bean が提供するメソッドが SPP/MHP の API となります。

また，TP1/Client/J または Cosminexus TP1 Connector を使用することによって，EJB からの通信も可能となります。

1.3.3 TP1/COBOL 基本 Bean の役割と処理（TP1/Client/P または TP1/Client/W）

TP1/COBOL 基本 Bean は，TP1/Client の次のサービスを提供します。ユーザが実行した RPC 接続以外のサービスです。

- ユーザ認証機能
- 常設コネクション
- トランザクション制御
- TCP/IP 通信機能（COBOL MHP に対するアクセス時に使用します）
- 一方通知受信機能

上記の TP1/Client の各関数は，TP1/COBOL 基本 Bean のメソッドとして提供します。そして，TP1/Client の関数の引数は，基本 Bean の各メソッドの引数により参照および設定処理を実現します。

1.3.4　TP1/COBOL アクセス用 Bean の役割と処理

(1)　TP1/Client/P または TP1/Client/W経由のアクセス

TP1/COBOL アクセス用 Bean は，ユーザが実行した RPC リクエスト（サービス）およびメッセージ送受信リクエストを TP1/Client/P または TP1/Client/W の RPC 関数およびメッセージ送受信関数にラッピングし，TP1/COBOL アクセス用 Bean のメソッドとして提供します。

これらの関数の引数は，TP1/COBOL アクセス用 Bean の各メソッドにより参照および設定（getter と setter）処理を実現します。

(2)　TP1/Client/J 経由のアクセス

TP1/COBOL アクセス用 Bean は，TP1/Client/J が提供する RPC メソッドおよびメッセージ送受信メソッドをラッピングし，TP1/COBOL アクセス用 Bean のメソッドとして提供します。

TP1/Client/J の RPC メソッドの引数は，TP1/COBOL アクセス用 Bean の各メソッドにより参照および設定（getter と setter）処理を実現します。

(3)　Cosminexus TP1 Connector 経由のアクセス

TP1/COBOL アクセス用 Bean は，Cosminexus TP1 Connector 提供の RPC 機能を実行するメソッド，execute をラッピングし，TP1/COBOL アクセス用 Bean のメソッドとして提供します。

Cosminexus TP1 Connector の execute メソッドの引数は，TP1/COBOL アクセス用 Bean の各メソッドにより参照および設定（getter と setter）処理を実現します。execute メソッドの詳細は Cosminexus TP1 Connector に添付のマニュアル「uCosminexus TP1 Connector 利用ガイド」をご覧ください。

1.3.5　スレッド制御

TP1/COBOL アクセスで提供するライブラリはスレッドセーフです。Java UAP で複数のスレッドを実行して SPP/MHP へのリクエストを制御することができます。

1.3.6　注意事項

システムを構築する際に，Windows 環境と UNIX 環境が混在している場合は，文字コード等に注意してください。詳しい内容は「9.　Windows/UNIX 混在システム構築上の注意事項」をご覧ください。

2 Web アプリケーションの開発

この章では，TP1/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。

2.1　WEB アプリケーションの開発手順

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 SPP/MHP の作成
2. TP1/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

2.2 COBOL SPP/MHP の作成

TP1/COBOL アクセスを使用する場合に，COBOL SPP/MHP 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 SPP/MHP プログラムソースの構成

COBOL SPP/MHP を作成する場合，次の構成で作成することをお勧めします。また，COBOL 言語で作成された既存の SPP/MHP がある場合には，この節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 TP1/COBOL アクセスを経由して実行できる SPP/MHP として利用できます。

(1) COBOL SPP/MHP プログラムのソース構成

1. COBOL SPP/MHP プログラムソース
2. COBOL－Java 間で渡す引数の登録集原文
   （TP1/COBOL アクセス用 Bean 生成時に必要な登録集原文）

新規に COBOL 言語で SPP を作成する場合の具体的な方法は，マニュアル「分散トランザクション処理機能 OpenTP1 プログラム作成リファレンス COBOL 言語編」，MHP を作成する場合は，マニュアル「分散トランザクション処理機能 OpenTP1 プロトコル TP1/NET/TCP/IP 編」など OpenTP1 の各種マニュアルをご覧ください。

COBOL 言語で SPP/MHP をコーディングする際，コーディング上の注意および名称の付け方の注意等については，TP1/COBOL の作成方法に従ってください。

(2) COBOL SPP プログラムの例

COBOL SPP プログラムの例を次に示します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     SEARCHTP1.
DATA            DIVISION.
LINKAGE SECTION.
  COPY SEARCHTP1COPYIN.     ------------------ ※
*> 01 PERSONAL-DATA-IN.                      ┐
*>   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).                ┘
  77 IN-LEN       PIC S9(9) USAGE COMP.
  COPY SEARCHTP1COPYOUT.    ------------------ ※
*> 01 PERSONAL-DATA-OUT.                     ┐
*>   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(100).               ┘
  77 OUT-LEN      PIC S9(9) USAGE COMP.
 PROCEDURE DIVISION   USING PERDONAL-DATA-IN,IN-LEN
                            PERDONAL-DATA-OUT,OUT-LEN.
     MOVE P-NUMBER IN PERDONAL-DATA-IN
        TO P-NUMBER IN PERDONAL-DATA-OUT.
*>  検索処理
     IF P-NUMBER IN PERDONAL-DATA-IN = 100001  THEN
       MOVE  '日立 一郎'       TO  P-NAME IN PERDONAL-DATA-OUT
       MOVE  '日立市'          TO  P-ADDRESS IN PERDONAL-DATA-OUT
       MOVE  '/ICHIRO.GIF'     TO  P-GIF IN PERDONAL-DATA-OUT
     ELSE
:
     END-IF
     EXIT PROGRAM.
 END PROGRAM SEARCHTP1.
```

注※：LINKAGE SECTION の各データ名定義は，登録集にすることをお勧めします。

TP1/COBOL アクセス用 Bean 生成時にこの登録集を使用できます。

(3) COBOL MHP と通信する登録集原文の作成方法

作成した COBOL MHP のプログラムを TP1/COBOL アクセスで利用するために登録集原文作成する必要があります。以下に CBLDCMCF('RECEIVE ')を例に説明します。その他の関数についてはこれに倣って修正してください。なお，メッセージ送受信インタフェースについてはマニュアル「分散トランザクション処理機能 OpenTP1 プロトコル TP1/NET/TCP/IP 編」の３章をご覧ください。

以下の例では一意名３の集団項目がポイントになります。

- PROCEDURE DIVISION の指定

```
CALL 'CBLDCMCF' USING 一意名1 一意名2 一意名3
```

- DATA DIVISION の指定

```
01 一意名1.
   02 データ名A PIC X(8) VALUE 'RECEIVE '.
         :
01 一意名2.
   02 データ名O PIC X(4) VALUE SPACE.
         :
01 一意名3.
   02 データ名U PIC 9(x) USAGE COMP.
```

```
02 データ名V PIC X(x).
02 データ名Y.
   05 データ1    PIC X(n).
        :
   05 データX    PIC X(n).
```

- データ名 Y

  データ名 Y には受信したセグメントの内容が返されます。Java UAP から受け取るデータは，データ名 Y の項目に格納されます。MHP のデータから TP1/COBOL アクセス用 Bean を生成する場合，以下のように登録集原文を作成する必要があります。また，Bean 生成ウィザードおよび Bean 生成ツールで，COBOL に渡す登録集原文に指定します（TP1/Client/P，TP1/Client/W および TP1/Client/J の場合は Java UAP から send メソッドで送信されたデータを受信できます。）。

1. データ名 Y が集団項目の場合

   下の例では，レベル番号を 01 に変更したデータ名 Y と従属する下位項目を登録集原文として作成しています。

   MHP 定義

```
02 データ名Y.
   05 データ1    PIC X(n).
        :
   05 データX    PIC X(n).
```

   登録集原文

```
01 データ名Y.
   05 データ1    PIC X(n).
        :
   05 データX    PIC X(n).
```

2. データ名 Y が基本項目の場合

   下の例では，レベル番号を 01 に変更したデータ名 Y を登録集原文として作成しています。

   MHP 定義

```
02 データ名Y  PIC X(n).
```

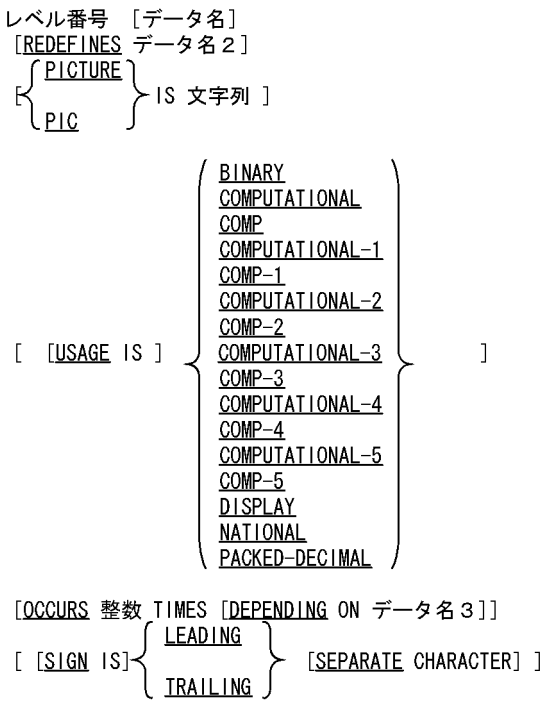
   登録集原文

```
01 データ名Y  PIC X(n).
```

2.2.2 TP1/COBOL アクセスを使用するための COBOL SPP/MHP の引数の規則

TP1/COBOL アクセスを使用するには COBOL SPP/MHP の引数は次の規則に従っている必要があります。

(1) 書き方



注

「(1)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。

(2) 構文規則

1. レベル番号，データ名，REDEFINES 句，PICTURE 句，USAGE 句，OCCURS 句，および 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，「(1)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，「(1)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。DEPENDING ON データ名 3 についての構文規則は，「2.2.3　DEPENDING ON データ名 3 を指定した可変長データだけの構文規則，一般規則」に記載します。
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。

(3) 一般規則

1. REDEFINES 句，USAGE 句，OCCURS 句，SIGN 句は，「(1)書き方」で記す表現形式の範囲において，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。DEPENDING ON データ名３についての構文規則は，「2.2.3　DEPENDING ON データ名３を指定した可変長データだけの構文規則，一般規則」に記載します。

(4) 使用上の注意事項

1. SIGN 句に SEPARATE CHARACTER 指定を書かない場合には，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　TP1/COBOL アクセス用 Bean 生成時の COBOL 定義注意事項/制限事項」をご覧ください。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合ったデータ項目の 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。

2.2.3 DEPENDING ON データ名３を指定した可変長データだけの構文規則，一般規則

(1) 構文規則

1. DEPENDING ON データ名３を指定すると，反復回数を可変とすることができます。

2. データ名３は，OCCURS 句が指定されたデータ項目を含む最上位項目の下位項目として定義しなければなりません。

```
正しい例（最上位項目WKGRPの下位項目として，データ名３WKCNTを定義している）
01   WKGRP.
 02  WKCNT   PIC 9(4).
 02  WKITEM  OCCURS 10 DEPENDING ON WKCNT  PIC X.
誤った例（最上位項目WKGRPの下位項目として，データ名３WKNOTFOUNDが定義されていない）
01   WKGRP.
 02  WKITEM  OCCURS 10 DEPENDING ON WKNOTFOUND  PIC X.
```

3. データ名３を OCCURS DEPENDING ON を指定したデータ項目のあとの文字位置を占める位置に定義してはなりません。

```
正しい例（可変反復データ項目WKITEMより前にWKCNTを定義している）
01   WKGRP.
 02  WKCNT   PIC 9(4).
 02  WKITEM  OCCURS 10 DEPENDING ON WKCNT  PIC X.
誤った例（可変反復データ項目WKITEMより後にWKCNTを定義している）
01   WKGRP.
```

```
 02  WKITEM  OCCURS 10 DEPENDING ON WKCNT  PIC X.
 02  WKCNT   PIC 9(4).
```

4. また，接頭辞および接尾辞を指定した COPY 文で展開するために，定義したデータ名と異なるデータ名３を指定している場合は，Bean 生成ウィザードで定義したデータ名の別名に，データ名３と同じ名称を指定してください。

(2) 一般規則

1. DEPENDING ON データ名３が指定された OCCURS 句の繰り返し回数は，以下に示す時点で変更されます。変更されるまでは，OCCURS 句に指定された最大繰り返し回数を繰り返し回数とします。

2. JavaUAP から COBOL SPP/MHP に渡す入力引数に定義されたデータ名３に setter メソッドで値を設定したときの値を繰り返し回数とします。(※1)(※2)

```
＜入力引数の定義例＞
01  WKGRP.
 02 WKCNT   PIC 9(4).
 02 WKITEM  OCCURS 10 DEPENDING ON WKCNT  PIC X.
＜JavaUAPの引数設定例＞
 :
bean.setWkcntI(new BigDecimal(4));     // 指定された4に，繰り返し回数を変更する
bean.setWkitemI("A",1);              // WKITEMの1番目の要素に"A"を設定する
 :
bean.setWkitemI("D",4);              // WKITEMの4番目の要素に"D"を設定する
 :
```

3. COBOL SPP/MHP から出力引数を返された際に，データ名３の領域に格納されている値を繰り返し回数とします(※2)。なお，出力引数中の可変繰り返しデータの繰り返し回数を JavaUAP で変更することはできません。

```
＜出力引数の定義例＞
01  WKGRP2.
 02 WKCNT2  PIC 9(4).
 02 WKITEM2 OCCURS 10 DEPENDING ON WKCNT2 PIC X.
＜JavaUAPの引数参照例＞
 :
bean.call(…);            // COBOL SPPから受け取った出力引数中のWKCNT2に5が
                         // 入っていると，WKITEM2の繰り返し回数も5になる
String item1 = (String)bean.getWkitem0(1); // WKITEM2の1番目の要素を取得する
 :
String item5 = (String)bean.getWkitem0(5); // WKITEM2の5番目の要素を取得する
 :
```

4. 添字を指定してデータの設定および参照を行う場合，１から繰り返し回数までの範囲の添字を指定できます。範囲外の添字を指定した場合，setter メソッドおよび getter メソッドで例外が発生します。

注※1

COBOL SPP/MHP に渡す入力引数に定義されたデータ名３を再定義したデータ項目に，setter メソッドで値を設定しても，繰り返し回数は変更されません。

```
＜入力引数の定義例＞
01  WKGRP.
 02 WKCNT   PIC 9(4).
 02 WKCNTR  REDEFINES WKCNT  PIC 9(4).
 02 WKITEM  OCCURS 10 DEPENDING ON WKCNT  PIC X.
＜JavaUAPの引数設定例＞
 :
bean.setWkcntrI(new BigDecimal(8));  // DEPENDING ONに指定されたデータ名ではないため，
                                     // WKITEMの繰り返し回数は変更されない
bean.setWkcntI(new BigDecimal(3));   // DEPENDING ONに指定されたデータ名であるため，
                                     // WKITEMの繰り返し回数を指定された3に変更する
 :
```

注※2

データ名3の値が，OCCURS 句の整数までの範囲外の場合，COBOL SPP/MHP に渡す入力引数に定義されたデータ名3を再定義したデータ項目に，setter メソッドで値を設定しても，繰り返し回数は変更されません。

```
＜入力引数の定義例＞
01  WKGRP.
 02 WKCNT   PIC 9(4).
 02 WKITEM  OCCURS 10 DEPENDING ON WKCNT  PIC X.
＜JavaUAPの引数設定例＞
 :
bean.setWkcntrI(new BigDecimal(13));  // DEPENDING ONに指定された繰り返し回数を超えて
                                      // いるため，WKITEMの繰り返し回数は変更されない
bean.setWkcntI(new BigDecimal(3));   // DEPENDING ONに指定された繰り返し回数の範囲内で
                                     // あるため，WKITEMの繰り返し回数を変更する
 :
```

2.2.4　SPP/MHP の引数を含む登録集原文の記述規則

1. COBOL SPP/MHP 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 SPP/MHP の引数の定義は一つ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」をご覧ください。）。
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・フリー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 CBLFREE 環境変数に指定した拡張子だけフリー形式，それ以外は固定形式として扱います。

   次に固定形式とフリー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）。

(フリー形式規則)

- 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）

(CBLFREE 環境変数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フリー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,エラーになった箇所以降の値は固定形式として扱います。

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。次の語以外の語は利用者語として扱います。

   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。

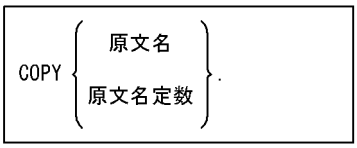
2.2.5 COPY 文

COBOL アクセスにおける COPY 文の機能について以下説明します。

(1)　言語規則

(a)　形式

COPY 文の指定形式は以下のようになります。



(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，空白の後に続け，分離符の終止符で止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 文のみ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，およびアンダーバーとし，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　TP1/COBOL アクセスを使用するための COBOL SPP/MHP 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なうことと同じです。
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. 固定形式の原文とフリー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」をご覧ください。

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完全ファイル名を引用符（”）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。完全ファイル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 で設定したフリー形式拡張子
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. ウィザードのステップ 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"としたとき，環境変数 CBLFREE が設定されていない場合，"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，CBLFREE と CBLLIB の環境変数を使用します。

(a)　CBLFREE

フリー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「2.2.4　SPP/MHP の引数を含む登録集原文の記述規則」をご覧ください。

(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　TP1/COBOL アクセス用 Bean の生成

TP1/COBOL アクセス用 Bean の生成方法について説明します。

Bean の生成には，「TP1/COBOL アクセス用 Bean 生成ウィザード」による生成方法と「TP1/COBOL アクセス用 Bean 生成ツール」による生成方法があります。uCosminexus Developer 07-00 以降をご使用になる場合，「TP1/COBOL アクセス用 Bean 生成ツール」で生成してください。「TP1/COBOL アクセス用 Bean 生成ツール」については，「2.3.4　TP1/COBOL アクセス用 Bean 生成ツール」をご覧ください。

「TP1/COBOL アクセス用 Bean 生成ウィザード」を使用する場合は，あらかじめ「TP1/COBOL adapter JBuilder への部品組み込み※」を使って，JBuilder に組み込んでおく必要があります。組み込み方法については「付録 C.1 (3)　JBuilder への TP1/COBOL アクセス用 Bean 生成ウィザードの組み込み」をご覧ください。

なお，生成された Bean は Javadoc に対応しています。

注※

「TP1/COBOL adapter JBuilder への部品組み込み」は，Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS では提供していません。

2.3.1　「TP1/COBOL アクセス用 Bean 生成ウィザード」による生成

JBuilder 上で TP1/COBOL アクセス用 Bean 生成ウィザードを使用することによって，COBOL SPP/MHP を呼び出すロジックを組み込んだ TP1/COBOL アクセス用 Bean のひな形を簡単に自動生成できます。これ以降「TP1/COBOL アクセス用 Bean 生成ウィザード」を「Bean 生成ウィザード」と省略します。

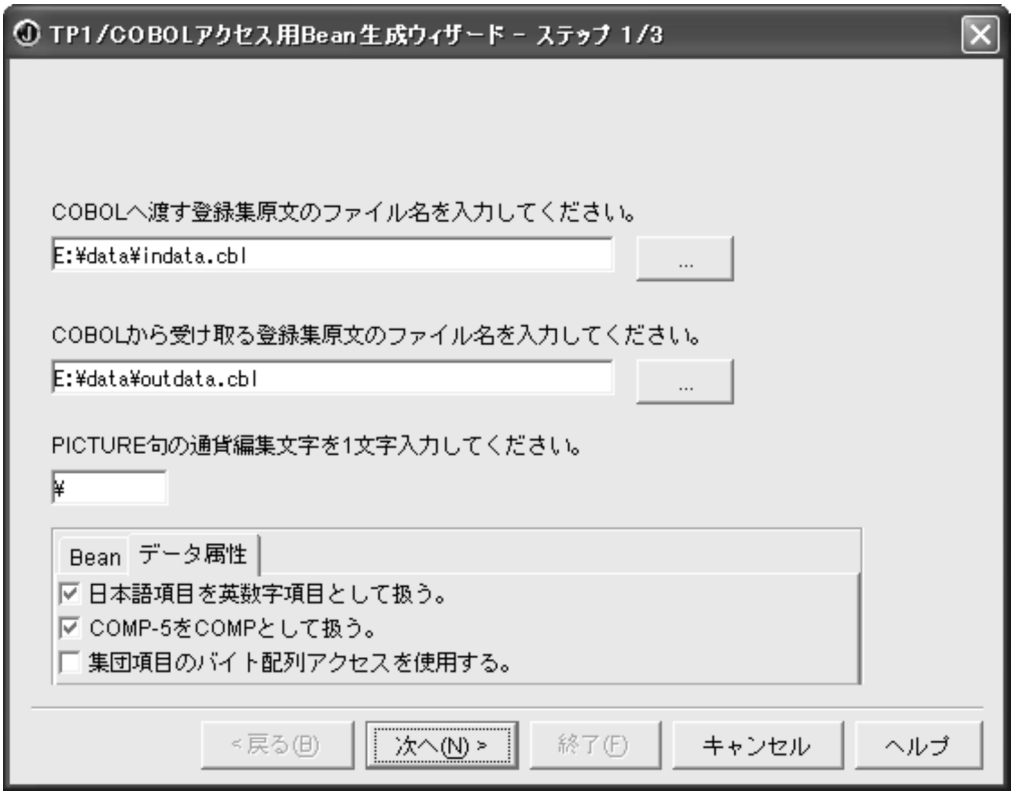
(1)　起動方法

JBuilder のオブジェクトギャラリ内の［一般］タブ内の" TP1/COBOL アクセス用 Bean"アイコンをダブルクリックします。または，JBuilder のメニューバーから［ウィザード］（(注）uCosminexus Developer から提供される JBuilder 2005 では［編集］メニューの［ウィザード］となります）の「TP1/COBOL アクセス用 Bean」をクリックします。

(2)　画面の説明

(a)　ステップ 1/3 画面

「Bean 生成ウィザード」が起動し，次のような「TP1/COBOL アクセス用 Bean 生成ウィザード-ステップ 1/3」画面が現れます。オプションパネルはタブで切り替えることができます。



ここでは，次の情報を入力します。

- SPP/MHP の入力パラメタ（COBOL へ渡す登録集原文）と出力パラメタ（COBOL から受け取る登録集原文）を持つデータ定義部の登録集原文のファイル名をフルパスで指定します（(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)。ここで，指定する登録集原文には条件があります。詳細は

「2.2.4　SPP/MHP の引数を含む登録集原文の記述規則」をご覧ください。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。

- 入力用原文および出力用原文の指定により，以下のメソッドを生成します。

	入力用原文指定時	出力用原文指定時
TP1/Client/P または TP1/Client/W	call,callTo,send および setter メソッド	receive,receive2 および getter メソッド
TP1/Client/J	call,send および setter メソッド	receive および getter メソッド
Cosminexus TP1 Connector	call および setter メソッド	getter メソッド

TP1/Client では，入力用原文と出力用原文の両方を省略することはできません。Cosminexus TP1 Connector では，入力用原文を省略することはできません。

- 必要に応じて，PICTURE 句の通貨編集用文字の変更を行います。デフォルトでは，’¥’ が指定されています。
- 集団項目に対するデータの設定および取得を行う場合は，「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行えません。
- 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，「別名を自動生成する。」チェックボックスをオンにします。デフォルトでは，別名は自動生成しないので，同じデータ名のデータ項目に対してデータの設定および取得を行うには「Bean 生成ウィザード」ステップ 2/3 で別名を指定する必要があります。
- 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 を指定した 2 進項目を COMP 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。デフォルトは，USAGE COMP-5 を指定した 2 進項目は，COMP として扱います。
- コンボボックスより生成したい Bean の形式を，以下の中から選択します。
  - TP1/Client/P または TP1/Client/W 経由のアクセスを使用する。※
  - TP1/Client/J 経由のアクセスを使用する。
  - Cosminexus TP1 Connector 経由のアクセスを使用する。

注※

Windows 64bit 版の場合，この選択肢は表示されません。

- 設定が完了したら，[次へ(N)>]ボタンを押すと入力用の「TP1/COBOL アクセス用 Bean 生成ウィザード：ステップ 2/3」画面に進みます。
- [キャンセル]ボタンを押すと，TP1/COBOL アクセス用 Bean の生成を取り消し，「Bean 生成ウィザード」画面を消去します。また，[ヘルプ]ボタンを押すと，この画面に対するヘルプを表示します。
- 指定された登録集原文を正常に解析できなかった場合，エラーメッセージを表示して処理が中止されます。

(b) ステップ 2/3 画面

1. 入力タブの画面

2. 出力タブの画面

「TP1/COBOL アクセス用 Bean 生成ウィザード-ステップ 2/3」画面には，「TP1/COBOL アクセス用 Bean 生成ウィザード-ステップ 1/3」画面で指定した入出力用の登録集原文を解析した情報から集団項目の情報が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）。

[入力]タブでは，SPP/MHP で実際に使用する入力用の引数を選択します。選択したデータ項目に対して TP1/COBOL アクセス用 Bean の中に当該プロパティが生成されます。

SPP/MHP で実際に使用するデータ項目だけを抽出することによって，TP1/COBOL アクセス用 Bean のサイズを SPP/MHP のサービスを受け取れる必要な大きさに最適化でき，資源を有効に活用できます。

[出力]タブでは，出力用のすべてのデータ項目を選択して TP1/COBOL アクセス用 Bean のプロパティを生成することによって，SPP/MHP から返される出力データの妥当性を確認するという使い方ができます。必要に応じて，選択するデータ項目を選んでください。

「パラメタテーブル」の編集の仕様については，「2.3.2　入出力用の「Bean 生成ウィザード」に表示されるパラメタテーブルの編集」をご覧ください。

設定が完了したら，[次へ(N)>]ボタンを押すと TP1/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われます。

(c)　ステップ 3/3 画面

「TP1/COBOL アクセス用 Bean 生成ウィザード-ステップ 3/3」画面は，TP1/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です。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。

  「クラス名.java」が生成されるファイル名になります。

以上のように画面に従って操作し，最後に，[終了]ボタンを押すと，JBuilder のプロジェクトに Java ソースファイルが生成され，当該 TP1/COBOL アクセス用 Bean が自動生成されます。

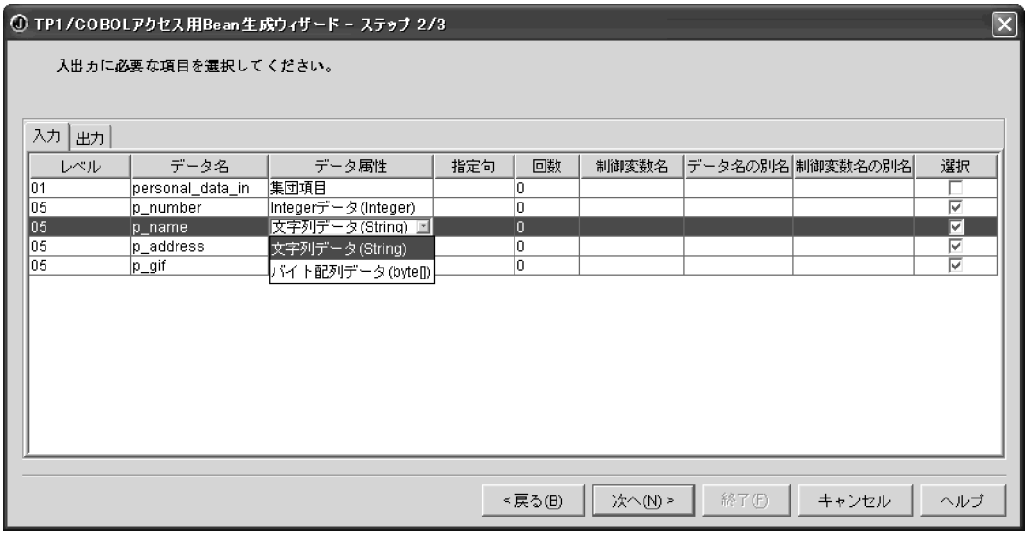
(TP1/COBOL アクセス用 Bean の自動生成ソースイメージは，「付録 E　TP1/COBOL アクセス用 Bean の自動生成ソースイメージ」をご覧ください)。

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 Temp 環境変数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。

2.3.2 入出力用の「Bean 生成ウィザード」に表示されるパラメタテーブルの編集

入出力用の「TP1/COBOL アクセス用 Bean 生成ウィザード-ステップ 2/3」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。



(1) 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。

(2) 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。この時，次の条件に該当するデータ項目名が変換されます。

- Java の言語仕様により，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，自動的にアンダーバー（_）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アンダーバー（_）で生成します。
- Bean 生成ウィザード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 の命名規則により，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。

(3) 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。

表 2-1 に各データ項目に対応する表示文字列を示します。

表 2-1 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 英数字項目 英数字編集項目 数字編集項目	文字列データ(String) バイト配列データ(byte[])※1
日本語項目 日本語編集項目	文字列データ(String) バイト配列データ(byte[])※1 日本語データ(String)※2
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)
集団項目	集団項目 集団項目(byte[])※3

注※1

バイト配列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。

注※2

Bean 生成ウィザードまたは Bean 生成ツール 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，バイト配列データへの変更はできません。

注※3

ウィザードまたはツールの 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。

(4) 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。

以下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）

この指定句フィールドを編集することはできません。

(注意事項)

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

(5) 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。

(6) 制御変数名フィールド

OCCURS DEPENDING ON 指定がある場合，制御変数名を表示します。

この制御変数名フィールドを編集することはできません。

(7) データの別名フィールド

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英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（-），アンダーバー（_）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 生成ウィザード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，ウィザードまたはツールの 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。

ウィザードまたはツールの 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. それぞれの引数で，同一データ名があるかをチェックする。
2. 同一データ名がある場合， 1 つ目のデータ名には別名は生成しない。
3. 2 つ目以降のデータ名に対して，データ名に$n(n:1 以上の整数)を付加した別名を生成する。
4. 制御変数となるデータ名に対して別名は生成しない。

(8) 制御変数名の別名フィールド

制御変数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。別名の編集用途は，次のとおりです。

- 制御変数名に日本語（データ名に英小文字，数字，ハイフン（-），アンダーバー(_)以外が含まれている場合。別名は，半角英数字でなければなりません。
- 同一引数内にある他のデータ項目を制御変数として指定する場合。同一引数内に，制御変数となるデータ項目が存在する必要があります。規則に反した場合，エラーとなります。

(9) 選択フィールド

使用する COBOL のデータ項目（すなわち，TP1/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，集団項目はウィザードまたはツール 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 以外の基本選択可能なデータ項目の選択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 TP1/COBOL アクセス用 Bean の中に setter/getter が生成されます。SPP/MHP で実際に使用するデータ項目を抽出することによって，TP1/COBOL アクセス用 Bean のサイズを SPP/MHP のサービスを受け取れる必要な大きさに最適化でき，資源を有効に活用できます。なお，制御変数として指定された項目に対して，本フィールドの指定にかかわらず常に setter/getter を生成します。

2.3.3 「TP1/COBOL アクセス用 Bean 生成ウィザード」でのバイト配列

英数字項目で「バイト配列データ(byte[])」は，「Bean 生成ウィザード」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。

「バイト配列データ(byte[])」は，以下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[])」が指定できるデータ項目

- 英字項目
- 英数字項目
- 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

日本語項目および日本語編集項目は，Bean 生成ウィザードまたは Bean 生成ツール 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。

2.3.4 TP1/COBOL アクセス用 Bean 生成ツール

「TP1/COBOL アクセス用 Bean 生成ツール」は JBuilder を使用しない開発環境で Bean の生成をできるようにしたものです。

ただし，ここでは[環境設定ダイアログ]の画面と「TP1/COBOL アクセス用 Bean 生成ツール-ステップ 1/3」の説明だけを行います。ほかの説明は「2.3.1 「TP1/COBOL アクセス用 Bean 生成ウィザード」による生成」をご覧ください。

これ以降「TP1/COBOL アクセス用 Bean 生成ツール」を「Bean 生成ツール」に省略します。

(1) 起動方法

Windows 画面で[スタート]ボタンをクリックし，[プログラム]－[COBOL2002]※1－[TP1/COBOL adapter for Cosminexus]の順でポイントし，[TP1/COBOL アクセス用 Bean 生成ツール環境設定] ※2 をクリックします。「TP1/COBOL アクセス用 Bean 生成ツール環境設定」の画面が開いたら，そこで環境設定を行います。

起動方法も同じように[スタート]－[プログラム]－[COBOL2002]※1－[TP1/COBOL adapter for Cosminexus]の順でポイントし，「TP1/COBOL アクセス用 Bean 生成ツール」※3 をクリックすることで起動できます。

注※ 1

COBOL85 をご使用の場合は，[ＣＯＢＯＬ８５]となります。

注※ 2

Windows(64bit)版の場合は，「TP1/COBOL アクセス用 Bean 生成ツール環境設定 64bit」となります。

注※ 3

Windows(64bit)版の場合は，「TP1/COBOL アクセス用 Bean 生成ツール 64bit」となります。

(2) 環境設定ダイアログ

VM オプションのチェックボックスをチェック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，JavaVM の初期値となります。

また，VM Option をチェックして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が出力されます。

なお，Windows(32bit)版の場合，ダイアログのタイトルは「TP1/COBOL アクセス用 Bean 生成ツール」となります。Windows(64bit)版の場合，ダイアログのタイトルは「TP1/COBOL アクセス用 Bean 生成ツール 64bit」となります。

(3) 「TP1/COBOL アクセス用 Bean 生成ツール」画面の説明

(a) 「ステップ 1/3」の画面

「TP1/COBOL アクセス用 Bean 生成ツール - ステップ 1/3」の画面の説明をします。2/3 および 3/3 の画面については「2.3.1 「TP1/COBOL アクセス用 Bean 生成ウィザード」による生成」をご覧ください。

なお，Windows(32bit)版の場合，ダイアログのタイトルは「TP1/COBOL アクセス用 Bean 生成ツール - ステップ n/3」(n はステップ数)となります。Windows(64bit)版の場合，ダイアログのタイトルは「TP1/COBOL アクセス用 Bean 生成ツール 64bit - ステップ n/3」(n はステップ数)となります。

「生成ファイルを出力するフォルダを入力してください。」および「serialVersionUID を生成する。」以外の項目は「2.3.1 「TP1/COBOL アクセス用 Bean 生成ウィザード」による生成」の説明をご覧ください。

「生成ファイルを出力するフォルダを入力してください。」には，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指定します。Java ソースファイルは，「(b) 生成ファイルのフォルダ規則」に従って生成されます。

「serialVersionUID を生成する。」は「(c) serialVersionUID の生成規則」をご覧ください。

(b) 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 画面で指定するパッケージ名を指します。生成する Java ソースファイル名もステップ 3/3 のクラス説明をご覧ください。

「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。

(c) serialVersionUID の生成規則

「serialVersionUID を生成する。」のチェックボックスをオンにすると，TP1/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。

(4) 制限事項

- 生成ツールを二つ以上起動できません。

2.4　Servlet の作成

TP1/COBOL アクセスにおいての Servlet の作成方法について説明します。

2.4.1　Servlet の作成方法

生成した TP1/COBOL アクセス用 Bean を呼び出す Servlet（Java UAP）を作成します。以下のことに注意して作成してください。

(1)　setter の使用

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 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).
```

（例）引数設定例

```
bean.setP_numberI (new Integer(number));
bean.setP_nameI ("");
bean.setP_addressI ("");
```

(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).
```

（例）JavaBeans の場合の引数設定例

```
int i = 0;
for (i = 0; i<10 ; i++) {
    bean.setB1I ("XXXXX", i);
}
```

(2)　COBOL SPP の場合の call メソッド呼び出し

Bean の setter を記述後に，TP1/COBOL アクセス用 Bean の call メソッドを呼び出します。

- TP1/Client/P および TP1/Client/W の場合

```
bean.call(クライアントID,”サービスグループ名”,”サービス名”,RPCの形態);
```

- TP1/Client/J の場合

```
bean.call(TP1Clientのインスタンス,”サービスグループ名”,”サービス名”,RPCの形態);
```

- Cosminexus TP1Connector の場合

```
bean.call(ConnectionFactoryインスタンス, Interactionインスタンス,InteractionSpecImplインスタンス);
```

(3) COBOL MHP の場合の send メソッド呼び出し

Bean の setter を記述後に，TP1/COBOL アクセス用 Bean の send メソッドを呼び出します。

[指定形式]

```
bean.send(クライアントID,”ホスト名”，“ポート番号”,コネクション解放の指定);
```

(4) COBOL MHP の場合の receive メソッド呼び出し

TP1/COBOL アクセス用 Bean の receive メソッドを呼び出します。受信したメッセージは Bean の getter メソッドで取得できます。

[指定形式]

```
bean.receive(クライアントID,”メッセージ受信時の最大待ち時間（秒）”， コネクション解放の指定);
```

(5) COBOL MHP の場合の receive2 メソッド呼び出し

TP1/COBOL アクセス用 Bean の receive2 メソッドを呼び出します。受信したメッセージは Bean の getter メソッドで取得できます。

[指定形式]

```
bean.receive2(クライアントID,”ホスト名”，“メッセージ受信時の最大待ち時間（秒）”,コネクション解放の指定);
```

(6) getter の指定

この値を 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).
```

（例）引数取得指定例

```
Integer wkint  = bean.getP_number0 ();
String  wkstr1 = bean.getP_name0 ();
String  wkstr2 = bean.getP_address0 ();
```

(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).
```

（例）引数取得例

```
String wkstr = bean.getB10 (5);
```

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 SPP の呼び出し

COBOL SPP は TP1/COBOL アクセス用 Bean を使用して呼び出します。TP1/COBOL アクセス用 Bean は，「TP1/COBOL アクセス用 Bean 生成ツール」，もしくは「TP1/COBOL アクセス用 Bean 生成ウィザード」を使用して生成します。

COBOL SPP は Servlet 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) TP1/Client/P および TP1/Client/W の場合

(a) UAP の開始手続き

- TP1/COBOL 基本 Bean のインスタンスを生成します。
- TP1/COBOL アクセス用 Bean のインスタンスを生成します。

- TP1/COBOL 基本 Bean の cltin メソッド（ユーザ認証要求機能）を発行します。
- TP1/COBOL 基本 Bean の open メソッド（UAP の開始）を発行します。

(b) データの設定

COBOL SPP の入力引数を TP1/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＋ I」となります。

例）COBOL の基本項目名が「ITEM01」の場合，「setItem01I」となります。

(c) COBOL SPP の呼び出し

TP1/COBOL アクセス用 Bean の call メソッドを発行して，COBOL SPP を呼び出します。

(d) データの取得

COBOL SPP の出力引数を TP1/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。（非応答型 RPC の場合，getter メソッドを発行すると例外が発生します。）メソッド名は，「get ＋ COBOL の基本項目名＋ O」となります。

例）COBOL の基本項目名が「ITEM01」の場合，「getItem01O」となります。

(e) UAP の終了手続き

- TP1/COBOL 基本 Bean の close メソッド（UAP の終了）を発行します。
- TP1/COBOL 基本 Bean の cltout メソッド（クライアントユーザの認証解除）を発行します。

(2) TP1/Client/J の場合

(a) UAP の開始手続き

- TP1/Client/J の TP1Client クラスのインスタンスを生成します。
- TP1/COBOL アクセス用 Bean のインスタンスを生成します。
- サーバとの接続の確立を行います。
  - Rap サーバの場合
    TP1Client クラスの openConnection メソッドを発行します。
  - scd サーバの場合
    TP1Client クラスの rpcOpen メソッドを発行します。

(b) データの設定

COBOL SPP の入力引数を TP1/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。上記，TP1/Client/P および TP1/Client/W の場合と同様です。

(c) COBOL SPP の呼び出し

TP1/COBOL アクセス用 Bean の call メソッドを発行して，COBOL SPP を呼び出します。

(d) データの取得

COBOL SPP の出力引数を TP1/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。上記，TP1/Client/P および TP1/Client/W の場合と同様です。

(e) UAP の終了手続き

サーバとの接続の解放を行います。

- Rap サーバの場合
  TP1Client クラスの closeConnection メソッドを発行します。
- scd サーバの場合
  TP1Client クラスの rpcClose メソッドを発行します。

(3) Cosminexus TP1 Connector の場合

(a) UAP の開始手続き

Cosminexus TP1 Connector で，以下を設定します。詳細は Cosminexus TP1 Connector に添付のマニュアル「uCosminexus TP1 Connector 利用ガイド」をご覧ください。

- ManagedConnectionFactory インスタンスの生成（Non-Managed 環境）
  JNDI ネーミングコンテキストの生成および JNDI 名前空間から ConnectionFactory インスタンスの取得（Managed 環境）
- ManagedConnectionFactory プロパティの設定
- ConnectionFactory インスタンスの生成
- コネクションの取得
- Interaction インスタンスの生成
- InteractionSpecImpl インスタンスの生成
- InteractionSpec プロパティの設定
  通信形態（setFlags）に同期応答型 RPC（DCNOFLAGS）を設定する。

(b) データの設定

COBOL SPP の入力引数を TP1/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。上記，TP1/Client/P および TP1/Client/W の場合と同様です。

(c) COBOL SPP の呼び出し

TP1/COBOL アクセス用 Bean の call メソッドを発行して，COBOL SPP を呼び出します。

(d) データの取得

COBOL SPP の出力引数を TP1/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。上記，TP1/Client/P および TP1/Client/W の場合と同様です。

(e) UAP の終了手続き

Cosminexus TP1 Connector の close メソッドを発行します。

2.4.3 COBOL MHP の呼び出し

COBOL MHP は TP1/COBOL アクセス用 Bean を使用して呼び出します。TP1/COBOL アクセス用 Bean は，「Bean 生成ウィザード」を使用して生成します。

COBOL MHP は Servlet 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) TP1/Client/P および TP1/Client/W で TP1/COBOL アクセス用 Bean の send/receive/receive2 メソッドを使用する場合

(a) UAP の開始手続き

- TP1/COBOL 基本 Bean のインスタンスを生成します。
- TP1/COBOL アクセス用 Bean のインスタンスを生成します。
- TP1/COBOL 基本 Bean の cltin メソッド（ユーザ認証要求機能）を発行します。
- TP1/COBOL 基本 Bean の open メソッド（UAP の開始）を発行します。

(b) データの設定

COBOL MHP の入力引数を TP1/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＋ I」となります。

例）COBOL の基本項目名が「ITEM01」の場合，「setItem01I」となります。

(c) COBOL MHP への送信

TP1/COBOL アクセス用 Bean の send メソッドを発行して，COBOL MHP にデータを送信します。

(d) COBOL MHP からの受信

TP1/COBOL アクセス用 Bean の receive メソッドを発行して，COBOL MHP から送信されたデータを受信します。

（障害発生時のメッセージ取得を行う receive2 メソッドを発行することもできます。）

(e) データの取得

COBOL MHP の出力引数を TP1/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。

（DCRPC_NOREPLY を指定して call メソッドを発行後，getter メソッドを発行すると例外が発生します。）

メソッド名は，「get ＋ COBOL の基本項目名＋ O」となります。

例）COBOL の基本項目名が「ITEM01」の場合，「getItem01O」となります。

(f) UAP の終了手続き

- TP1/COBOL 基本 Bean の close メソッド（UAP の終了）を発行します。
- TP1/COBOL 基本 Bean の cltout メソッド（クライアントユーザの認証解除）を発行します。

(2) TP1/Client/P および TP1/Client/W で TP1/COBOL 基本 Bean 提供の send/receive/receive2 メソッドを使用する場合

(a) UAP の開始手続き

- TP1/COBOL 基本 Bean のインスタンスを生成します。
- TP1/COBOL アクセス用 Bean のインスタンスを生成します。
- TP1/COBOL 基本 Bean の cltin メソッド（ユーザ認証要求機能）を発行します。
- TP1/COBOL 基本 Bean の open メソッド（UAP の開始）を発行します。

(b) データの設定

- COBOL MHP に送信したい String 型のデータを用意します。
- COBOL MHP から受信するために必要な，要素数 1 以上の String 型の配列を用意します。

(c) COBOL MHP への送信

TP1/COBOL 基本 Bean の send メソッドを発行して，COBOL MHP にデータを送信します。

(d) COBOL MHP からの受信

データ受信用に用意した String 配列を指定して，TP1/COBOL 基本 Bean の receive メソッドを発行して，COBOL MHP から送信されたデータを受信します。String 配列に受信データが格納されます。

（障害発生時のメッセージ取得を行う receive2 メソッドを発行することもできます。）

(e) UAP の終了手続き

- TP1/COBOL 基本 Bean の close メソッド（UAP の終了）を発行します。
- TP1/COBOL 基本 Bean の cltout メソッド（クライアントユーザの認証解除）を発行します。

(3) TP1/Client/J の場合

(a) UAP の開始手続き

- TP1/Client/J の TP1Client クラスのインスタンスを生成します。
- TP1/COBOL アクセス用 Bean のインスタンスを生成します。
- サーバとの接続の確立を行います。
  TP1Client クラスの rpcOpen メソッドを発行します。

(b) データの設定

COBOL MHP の入力引数を TP1/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。上記，TP1/Client/P および TP1/Client/W で TP1/COBOL アクセス用 Bean の send/receive/receive2 メソッドを使用する場合と同様です。

(c) COBOL MHP への送信

TP1/COBOL アクセス用 Bean の send メソッドを発行して，COBOL MHP にデータを送信します。

(d) COBOL MHP からの受信

TP1/COBOL アクセス用 Bean の receive メソッドを発行して，COBOL MHP から送信されたデータを受信します。

(e) データの取得

COBOL MHP の出力引数を，TP1/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。上記，TP1/Client/P および TP1/Client/W で TP1/COBOL アクセス用 Bean の send/receive/receive2 メソッドを使用する場合と同様です。

(f) UAP の終了手続き

サーバとの接続の解放を行います。TP1Client クラスの rpcClose メソッドを発行します。

(4) Cosminexus TP1 Connector の場合

(a) UAP の開始手続き

通信形態の設定値を除いて，「2.4.2(3) Cosminexus TP1 Connector の場合」と同様です。

通信形態（setFlags）に同期送受信（TCPIP_SENDRECV）を設定します。

(b)　データの設定

SPP を MHP と読み替える以外，「2.4.2(3) Cosminexus TP1 Connector の場合」と同様です。

(c)　COBOL MHP との送受信

SPP を MHP と読み替える以外，「2.4.2(3) Cosminexus TP1 Connector の場合」と同様です。

(d)　データの取得

SPP を MHP と読み替える以外，「2.4.2(3) Cosminexus TP1 Connector の場合」と同様です。

(e)　UAP の終了手続き

「2.4.2(3) Cosminexus TP1 Connector の場合」と同様です。

2.4.4　例外処理

TP1/COBOL アクセス用 Bean では，COBOL SPP/MHP でエラーが発生した場合，または，TP1/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。

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，「8.8　J2CBException ユーザインタフェース API」をご覧ください。

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 H　プログラム例」の HTML の作成例および JSP の作成例をご覧ください。「付録 H.1 TP1/Client/P および TP1/Client/W 版」,「付録 H.2　TP1/Client/J 版」および「付録 H.3 Cosminexus TP1 Connector 版」でそれぞれ作成例が記載されています。

2.6　ユーザプログラムの作成上の注意事項

COBOL SPP/MHP 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

このシステムでは，次の文字コードが使用できます。

- シフト JIS
- 日本語 EUC
- Unicode

また，CBLJ2TP1OPT 環境変数の 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，「5.3　TP1/COBOL アクセス環境変数の設定」をご覧ください。

(1)　Windows 版

- シフト JIS コードで決められた漢字
  デフォルトはシフト JIS コードですが，CBLJ2TP1OPT 環境変数の 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。

(2)　HP-UX/AIX/Linux 版

- 環境変数 LANG の設定値に従った文字コードの漢字
  デフォルトの文字コードは環境変数 LANG によって決まりますが，CBLJ2TP1OPT 環境変数の 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。

表 2–2　2 動作環境とデフォルトの文字コード

システム	環境変数 LANG の設定値	動作環境（ロケール）	文字コード
HP-UX	ja_JP.SJIS	シフト JIS	シフト JIS
	ja_JP.eucJP	日本語 EUC	日本語 EUC
AIX	Ja_JP	シフト JIS	シフト JIS
	ja_JP	日本語 EUC	日本語 EUC
Linux	ja_JP ja_JP.eucJP ja_JP.ujis	日本語 EUC	日本語 EUC
	ja_JP.UTF-8 ja_JP.utf8	UTF-8	UTF-8

2.6.2　COBOL コンパイラオプションについて

TP1/COBOL アクセスで 1 バイトの 2 進項目（-Bin1Byte コンパイラオプション）※は使えません。

注※

　COBOL85 をご使用の場合は，-B1 コンパイラオプションとなります。

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　コーディングについて

一つの Servlet または JSP 内では，cltin( )～cltout( )のメソッドを発行するようにコーディングしてください。

cltin メソッド，cltout メソッドおよび call メソッドを別々の Servlet または JSP で使用するようなコーディングをしないでください。

3 プログラムのコンパイル

この章では，作成した TP1 サーバ用の COBOL SPP/MHP 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 SPP/MHP のコンパイル

COBOL のコンパイル方法は，マニュアル「分散トランザクション処理機能 OpenTP1 プログラム作成リファレンス COBOL 言語編」をご覧ください。

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 版

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，以下のことに注意してください。

(1)　TP1/Client/P および TP1/Client/W の場合

クラスパスに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ>※¥LIB¥j2tp1run.jar」を指定する必要があります。

注※

開発環境で TP1/COBOL アクセスを使用するプログラムのコンパイルを行う場合は，「TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ」を「TP1/COBOL adapter for Cosminexus Version 2 インストールフォルダ」と読み替えてください。

(2)　TP1/Client/J の場合

クラスパスに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ>※¥LIB¥j2tp1run.jar」を指定する必要があります。また，TP1/Client/J から提供される TP1Client.jar も指定する必要があります。

注※

開発環境で TP1/COBOL アクセスを使用するプログラムのコンパイルを行う場合は，「TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ」を「TP1/COBOL adapter for Cosminexus Version 2 インストールフォルダ」と読み替えてください。

(3)　Cosminexus TP1 Connector の場合

クラスパスに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ>※3¥LIB¥j2tp1run.jar」を指定する必要があります。また，TP1/Client/J から提供される TP1Client.jar，Cosminexus TP1 Connector から提供される tp1connector.jar※1，および Cosminexus TP1 Connector から提供される tp1concommon.jar※2も指定する必要があります。

注※ 1

Cosminexus TP1 Connector の Non-Managed 環境を使用する場合に必要です。

注※ 2

Cosminexus TP1 Connector 02-00-/A 以降を使用する場合に必要です。

注※3

開発環境で TP1/COBOL アクセスを使用するプログラムのコンパイルを行う場合は，「TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ」を「TP1/COBOL adapter for Cosminexus Version 2 インストールフォルダ」と読み替えてください。

3.2.2 HP-UX/AIX/Linux 版

- 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 HP-UX/AIX/Linux 環境へ移して（バイナリコードで転送する）ご使用ください。

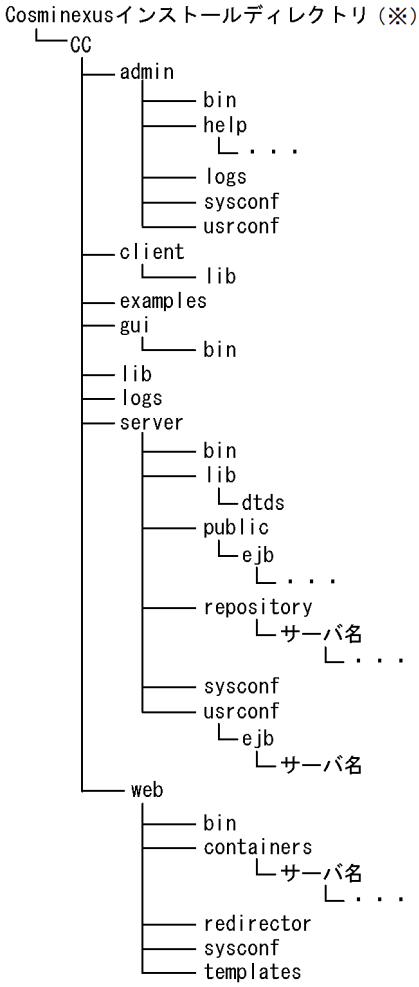
4 アプリケーションサーバの環境整備

作成した COBOL SPP/MHP 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 Web コンテナサーバを使用した場合，TP1/Client/J または Cosminexus TP1 Connector で J2EE サーバを使用した場合に分けて説明します。

4.1　環境を説明する前に

実行環境を説明する上で，次のことを前提とします。

- TP1/COBOL 拡張 [Server] Run Time System for Cosminexus Version 2，または TP1/COBOL adapter for Cosminexus Version 2 がインストールされていなければなりません。
- TP1 サーバ側の OpenTP1 の起動と停止方法については，マニュアル「分散トランザクション処理機能 OpenTP1 運用と操作」をご覧ください。
- Cosminexus が動作するために必要な定義および環境変数を設定してください。Cosminexus の環境定義の詳細については，マニュアル「Cosminexus　システム構築ガイド」，もしくはマニュアル「Cosminexus V9 アプリケーションサーバ　システム構築・運用ガイド」をご覧ください。
- EJB 経由で TP1/Client/J を使用する場合は，J2EE サーバの設定を行ってください。
- EJB 経由で Cosminexus TP1 Connector を使用する場合は，J2EE サーバの設定を行ってください。
- これ以降の説明で使用される Cosminexus Component Container のインストールディレクトリ階層を以下に示します。



注※

Cosminexus インストールディレクトリは，Windows 版ではデフォルトインストール先が「C:¥Program Files¥Hitachi¥Cosminexus」となっていますが，インストール時に変更可能です。それに対して HP-UX/AIX/Linux 版では/opt/Cosminexus と固定になっています。

そのため，CC までのディレクトリ構成(下の図)を<CC インストール先>と省略して以下説明します。

```
Cosminexusインストールディレクトリ
    └── CC
```

(例)

server ディレクトリを表す場合は，Windows 版では<CC インストール先>¥server，HP-UX/AIX/Linux 版では<CC インストール先>/server と表現します。

4.2　Web コンテナサーバを使用した場合

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する必要があります。

- Web サーバ
- Web コンテナサーバ

Web サーバの設定についてはご使用の Web サーバのマニュアルをご覧ください。

4.2.1　TP1/Client/P 使用時の環境変数の設定（Windows 版）

TP1/Client/P 使用時は TP1/COBOL アクセス Windows 版を使った設定を行います。Web コンテナサーバの CLASSPATH 環境変数の設定および Library Path 環境変数の設定を行ってください。設定内容について以下に説明します。

(1)　CLASSPATH 環境変数の設定

次の手順にて CLASSPATH 環境変数に「<TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ>※¥LIB¥j2tp1run.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¥j2tp1run.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。

注※

開発環境で TP1/COBOL アクセスを使用するプログラムのデバッグを行う場合は，「TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ」を「TP1/COBOL adapter for Cosminexus Version 2 インストールフォルダ」と読み替えてください。

(2)　LibraryPath の設定

次の手順で Library Path を設定します。

1. Windows の[スタートメニュー]から[設定]-[コントロールパネル]の順で[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ボタンをクリックします。

3. システム環境変数の PATH 環境変数に「<TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 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。）

注※

開発環境で TP1/COBOL アクセスを使用するプログラムのデバッグを行う場合は，「TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ」を「TP1/COBOL adapter for Cosminexus Version 2 インストールフォルダ」と読み替えてください。

4.2.2 TP1/Client/W 使用時の環境変数の設定（HP-UX/AIX/Linux 版）

TP1/Client/W 使用時は TP1/COBOL アクセス HP-UX，AIX または Linux 版を使った設定を行います。Web コンテナサーバでは CLASSPATH 環境変数の設定および Library Path 環境変数の設定を行ってください。設定内容について以下に説明します。

(1) CLASSPATH 環境変数の設定

次の手順にて CLASSPATH 環境変数に”/opt/HILNGcbl/lib/j2tp1run.jar”を設定します。

ただし，AIX COBOL2002 がインストールされている場合は,”/opt/HILNGcbl2k/lib/j2tp1run.jar”を設定する必要があります。

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/HILNGcbl/lib/j2tp1run.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) Library Path の設定

次の手順で Library Path を設定します。

(a) HP-UX 版

1. システム環境変数の SHLIB_PATH 環境変数に"/opt/HILNGcbl/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。

(b) AIX 版

1. システム環境変数の LIBPATH 環境変数に"/opt/HILNGcbl/lib"を追加します。ただし，AIX COBOL2002 がインストールされている場合は,”/opt/HILNGcbl2k/lib”を設定する必要があります。
2. 設定後，Web コンテナサーバを再起動します。

(c) Linux 版

1. システム環境変数の LD_LIBRARY_PATH 環境変数に"/opt/HILNGcbl/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。

4.2.3 TP1/Client/J 使用時の環境変数の設定

TP1/Client/J をお使いの場合は，TP1/Client/P または TP1/Client/W と同様の設定をした後，任意のディレクトリ下に TP1Client.jar を格納し，CLASSPATH 環境変数に TP1Client.jar を追加します。

(1) Windows 版（TP1/Client/P と同様）

以下に任意のフォルダを C:¥CLTJ として説明します。

1. C:¥CLTJ フォルダを作成し，そのフォルダ内に TP1Client.jar を格納します。
2. TP1/Client/P と同様に Web コンテナサーバのユーザ定義ファイル（usrconf.cfg）に C:¥CLTJ ¥TP1Client.jar を追加します。詳細はマニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

   ＜指定例＞

```
web.add.class.path=C:¥CLTJ¥TP1Client.jar
```

(2) HP-UX/AIX 版（TP1/Client/W と同様）

以下に任意のディレクトリを/usr/cltj として説明します。

1. /usr/cltj ディレクトリを作成し，そのディレクトリ内に TP1Client.jar を格納します。
2. TP1/Client/W と同様に Web コンテナサーバのユーザ定義ファイル（usrconf.cfg）に/usr/cltj/TP1Client.jar を追加します。詳細はマニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

   ＜指定例＞

```
web.add.class.path=/usr/cltj/TP1Client.jar
```

4.2.4　Cosminexus TP1 Connector 使用時の環境変数の設定

Cosminexus TP1 Connector をお使いの場合は，TP1/Client/P または TP1/Client/W と同様の設定をした後，任意のディレクトリ下に TP1Client.jar を格納し，CLASSPATH 環境変数に TP1Client.jar，tp1connector.jar（Non-Managed 環境）および tp1concommon.jar（Cosminexus TP1 Connector 02-00-/A 以降）を追加します。

(1)　Windows 版

TP1/Client/J と同様に Web コンテナサーバのユーザ定義ファイル（usrconf.cfg）に C:¥CLTJ¥TP1Client.jar，<Cosminexus TP1 Connector インストールフォルダ>¥lib¥tp1connector.jar（Non-Managed 環境）および<Cosminexus TP1 Connector インストールフォルダ>¥lib¥common¥tp1concommon.jar（Cosminexus TP1 Connector 02-00-/A 以降）を追加します。詳細は Cosminexus TP1 Connector に添付のマニュアル「uCosminexus TP1 Connector 利用ガイド」をご覧ください。

＜指定例＞

```
web.add.class.path=C:¥CLTJ¥TP1Client.jar
web.add.class.path=C:¥PROGRA~1¥Hitachi¥
TP1Connector¥lib¥tp1connector.jar
web.add.class.path=C:¥PROGRA~1¥Hitachi¥
TP1Connector¥lib¥common¥tp1concommon.jar
```

(2)　HP-UX/AIX/Linux 版

TP1/Client/J と同様に Web コンテナサーバのユーザ定義ファイル（usrconf.cfg）に/usr/cltj/TP1Client.jar，<Cosminexus TP1 Connector インストールディレクトリ>/lib/tp1connector.jar（Non-Managed 環境）および<Cosminexus TP1 Connector インストールディレクトリ>/lib/common/tp1concommon.jar（Cosminexus TP1 Connector 02-00-/A 以降）を追加します。詳細は Cosminexus TP1 Connector に添付のマニュアル「uCosminexus TP1 Connector 利用ガイド」をご覧ください。

＜指定例＞

```
web.add.class.path=/usr/cltj/TP1Client.jar
web.add.class.path=/opt/TP1Connector/lib/tp1connector.jar
web.add.class.path=/opt/TP1Connector/lib/common/tp1concommon.jar
```

4.3　EJB 経由で TP1/Client/J を使用した場合

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する必要があります。

- Web サーバ
- Web コンテナサーバ
- J2EE サーバ

Web サーバの設定についてはご使用の Web サーバのマニュアルをご覧ください。

4.3.1　J2EE サーバの環境変数の設定

J2EE サーバでは CLASSPATH 環境変数の設定および Library Path 環境変数の設定を行ってください。設定内容について以下に説明します。

(1)　J2EE サーバのユーザ定義

J2EE サーバのユーザ定義ファイル usrconf.cfg で次の設定を行ってください。

(a)　Windows 版

1. add.class.path キーに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ>※¥LIB¥j2tp1run.jar」を設定します。
2. add.class.path キーに「任意フォルダ¥TP1Client.jar」を設定します。
3. add.library.path キーに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ>※¥BIN」を設定します。

usrconf.cfg は<CC インストール先>¥server¥usrconf¥ejb¥<サーバ名称>の下に格納されています。

＜ 設定例 ＞

```
add.class.path=C:¥PROGRA~1¥Hitachi¥Cobol2~1¥LIB¥j2tp1run.jar
add.class.path=C:¥CLTJ¥TP1Client.jar
add.library.path=C:¥PROGRA~1¥Hitachi¥Cobol2~1¥BIN
```

設定例では TP1/Client/J のインストール先を任意のフォルダ「C:¥CLTJ」としています。

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。

注※

開発環境で TP1/COBOL アクセスを使用するプログラムのデバッグを行う場合は，「TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ」を「TP1/COBOL adapter for Cosminexus Version 2 インストールフォルダ」と読み替えてください。

(b)　HP-UX/AIX/Linux 版

1. add.class.path キーに「/opt/HILNGcbl/lib/j2tp1run.jar」を設定します。ただし，AIX COBOL2002 がインストールされている場合は，「/opt/HILNGcbl2k/lib/j2tp1run.jar」を設定する必要があります。
2. add.class.path キーに「任意ディレクトリ/TP1Client.jar」を設定します。
3. add.library.path キーに「/opt/HILNGcbl/lib」を設定します。ただし，AIX COBOL2002 がインストールされている場合は，「/opt/HILNGcbl2k/lib」を設定する必要があります。

usrconf.cfg は<CC インストール先>/server/usrconf/ejb/<サーバ名称>の下に格納されています。

＜ 設定例 ＞

```
add.class.path=/opt/HILNGcbl/lib/j2tp1run.jar
add.class.path=/usr/cltj/TP1Client.jar
add.library.path=/opt/HILNGcbl/lib
```

設定例では TP1/Client/J のインストール先を任意のディレクトリ「/usr/cltj」としています。

4.3.2　J2EE サーバの再起動

J2EE サーバがすでに起動されている場合は，環境設定を有効にするために再起動してください。詳細はマニュアル「Cosminexus システム運用ガイド」，もしくはマニュアル「Cosminexus V9 アプリケーションサーバ　システム構築・運用ガイド」をご覧ください。

4.4　EJB 経由で Cosminexus TP1 Connector を使用した場合

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する必要があります。

- Web サーバ
- Web コンテナサーバ
- J2EE サーバ

Web サーバの設定についてはご使用の Web サーバのマニュアルをご覧ください。

4.4.1　J2EE サーバの環境変数の設定

J2EE サーバでは CLASSPATH 環境変数の設定および Library Path 環境変数の設定を行ってください。設定内容について以下に説明します。

(1)　J2EE サーバのユーザ定義

J2EE サーバのユーザ定義ファイル usrconf.cfg で次の設定を行ってください。

(a)　Windows 版

1. add.class.path キーに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ>※¥LIB¥j2tp1run.jar」を設定します。
2. add.class.path キーに「任意フォルダ¥TP1Client.jar」を設定します。
3. add.class.path キーに「<Cosminexus TP1 Connector インストールフォルダ>¥lib¥tp1connector.jar」を設定します。
4. add.class.path キーに「<Cosminexus TP1 Connector インストールフォルダ>¥lib¥common¥tp1concommon.jar」を設定します。
5. add.library.path キーに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ>※¥BIN」を設定します。

usrconf.cfg は<CC インストール先>¥server¥usrconf¥ejb¥<サーバ名称>の下に格納されています。

< 設定例 >

```
add.class.path=C:¥PROGRA~1¥Hitachi¥Cobol2~1¥LIB¥j2tp1run.jar
add.class.path=C:¥CLTJ¥TP1Client.jar
add.class.path=C:¥PROGRA~1¥Hitachi¥
TP1Connector¥lib¥tp1connector.jar
add.class.path=C:¥PROGRA~1¥Hitachi¥
TP1Connector¥lib¥common¥tp1concommon.jar
add.library.path=C:¥PROGRA~1¥Hitachi¥Cobol2~1¥BIN
```

設定例では TP1/Client/J のインストール先を任意のフォルダ「C:¥CLTJ」としています。

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。

注※

開発環境で TP1/COBOL アクセスを使用するプログラムのデバッグを行う場合は，「TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 インストールフォルダ」を「TP1/COBOL adapter for Cosminexus Version 2 インストールフォルダ」と読み替えてください。

(b) HP-UX/AIX/Linux 版

1. add.class.path キーに「/opt/HILNGcbl/lib/j2tp1run.jar」を設定します。ただし，AIX COBOL2002 がインストールされている場合は，「/opt/HILNGcbl2k/lib/j2tp1run.jar」を設定する必要があります。
2. add.class.path キーに「任意ディレクトリ/TP1Client.jar」を設定します。
3. add.class.path キーに「/opt/TP1Connector/lib/tp1connector.jar」を設定します。
4. add.class.path キーに「/opt/TP1Connector/lib/common/tp1concommon.jar」を設定します。
5. add.library.path キーに「/opt/HILNGcbl/lib」を設定します。ただし，AIX COBOL2002 がインストールされている場合は，「/opt/HILNGcbl2k/lib」を設定する必要があります。

usrconf.cfg は<CC インストール先>/server/usrconf/ejb/<サーバ名称>の下に格納されています。

< 設定例 >

```
add.class.path=/opt/HILNGcbl/lib/j2tp1run.jar
add.class.path=/usr/cltj/TP1Client.jar
add.class.path=/opt/TP1Connector/lib/tp1connector.jar
add.class.path=/opt/TP1Connector/lib/common/tp1concommon.jar
add.library.path=/opt/HILNGcbl/lib
```

設定例では TP1/Client/J のインストール先を任意のディレクトリ「/usr/cltj」としています。

4.4.2 J2EE サーバの再起動

J2EE サーバがすでに起動されている場合は，環境設定を有効にするために再起動してください。詳細はマニュアル「Cosminexus　システム運用ガイド」，もしくはマニュアル「Cosminexus V9 アプリケーションサーバ　システム構築・運用ガイド」をご覧ください。

5 COBOL SPP/MHP と TP1/COBOL アクセスの環境整備

この章では，TP1/COBOL アクセス環境下において，COBOL SPP/MHP の実行環境の設定について説明します。設定は OpenTP1 側で行います。

5.1 COBOL SPP 環境変数の設定

TP1/COBOL アクセス環境下において，COBOL プログラムの実行環境を設定するための環境変数を指定する方法について説明します。設定は OpenTP1 側で行います。

5.1.1 Windows 版 TP1/LiNK の場合

1. [スタート]メニューから[TP1/LiNK]－＞[アプリケーション管理 SPP]の順で TP1/LiNK アプリケーション管理 SPP 画面を開きます。
2. TP1/LiNK アプリケーション管理 SPP 画面の[サーバ定義]ボタンを押すと，アプリケーション環境 SPP 画面が開きます。
3. アプリケーション環境 SPP 画面のユーザサーバ名を選択して[開く]ボタンを押すと，SPP 環境設定画面が開きます。
4. SPP 環境設定画面の[ユーザサーバの環境変数]の中に変数を設定します。

詳細な設定方法についてはマニュアル「分散アプリケーションサーバ TP1/LiNK 使用の手引」をご覧ください。

5.1.2 HP-UX 版 TP1/LiNK の場合

コマンドラインから dcsysset コマンドを入力します。

```
入力内容　[ dcsysset -u ユーザサーバ名 ]
```

詳細な設定方法についてはマニュアル「分散アプリケーションサーバ TP1/LiNK 使用の手引」をご覧ください。

5.1.3 Windows/HP-UX/AIX/Linux 版 TP1/Server Base の場合

コマンドラインから変更したいユーザサーバのユーザサービス定義ファイル（$DCCONFPATH/ユーザサーバ名）に変数を設定します。

```
入力内容　[ putenv 環境変数名 環境変数値 ]
```

詳細な設定方法についてはマニュアル「分散トランザクション処理機能 OpenTP1 システム定義」をご覧ください。

5.2　COBOL MHP 環境変数の設定

TP1/COBOL アクセス環境下において，COBOL プログラムの実行環境を設定するための環境変数を指定する方法について説明します。設定は OpenTP1 側で行います。

5.2.1　Windows 版 TP1/LiNK の場合

1. [スタート]メニューから[TP1/LiNK]－＞[アプリケーション管理 MHP]の順でアプリケーション管理画面を開きます。
2. TP1/Messaging アプリケーション管理 MHP 画面でユーザサーバ名を選択して[サーバ定義]ボタンを押すと，MHP 環境設定画面が開きます。
3. MHP 環境設定画面の[ユーザサーバの環境変数]の中に変数を設定します。

詳細な設定方法についてはマニュアル「TP1/Messaging 使用の手引」をご覧ください。

5.2.2　Windows/HP-UX/AIX/Linux 版 TP1/Server Base の場合

コマンドラインから変更したいユーザサーバのユーザサービス定義ファイル（$DCCONFPATH/ユーザサーバ名）に変数を設定します。

```
入力内容　　[　putenv 環境変数名　環境変数値　　]
```

詳細な設定方法についてはマニュアル「分散トランザクション処理機能 OpenTP1 システム定義」をご覧ください。

5.3　TP1/COBOL アクセス環境変数の設定

TP1/COBOL アクセスの環境変数の設定について説明します。

5.3.1　CBLJ2TP1OPT 環境変数

TP1/COBOL アクセス実行環境変数"CBLJ2TP1OPT"は，TP1/COBOL アクセスの実行時ライブラリに渡すオプションを設定することができます。設定方法は次のとおりです。なお，設定に反している場合は無効となります。また，同じオプション名を TP1/COBOL アクセス実行環境変数"CBLJ2TP1OPT"に指定した場合，最初に指定したオプション指定を有効にします。

(1)　オプションの設定方法（Windows 版）

オプションの設定方法を TP1/Client/P，TP1/Client/J および Cosminexus TP1 Connector に分けて説明します。各オプションの種類および説明は 5.3.2 以降をご覧ください。

注　Windows 64bit 版では TP1/Client/P は未サポート。

(a)　TP1/Client/P

環境変数は，シェル，コマンドプロンプトまたはコントロールパネルの中のシステムから次の形式で設定します。

[指定形式]

```
set CBLJ2TP1OPT=[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 codeconv オプション，codeconvflag オプション，comp5 オプション，dccm3 オプション，dccm3flag オプション，encode オプション，endian オプション，japanese オプションおよび unicode オプションです。

(b)　TP1/Client/J および Cosminexus TP1 Connector

CBLJ2TP1OPT 環境変数にオプションを設定するのではなく，Java のシステムプロパティにオプション tp1cobol.option を設定する点が，TP1/Client/P と異なるのでご注意ください。現在用意されているオプションは，codeconv オプション，codeconvflag オプション，comp5 オプション，encode オプション，endian オプション，japanese オプションおよび unicode オプションです。

- web コンテナサーバの場合

  usrconf.cfg ファイルの"add.jvm.arg"に-D オプションを付けて，tp1cobol.option を指定します。

  [指定形式]

  ```
  add.jvm.arg=-Dtp1cobol.option=OPTION [ : OPTION ]
  ```

[usrconf.cfg 格納フォルダ]

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

- J2EE サーバの場合

  usrconf.cfg ファイルの"add.jvm.arg"に-D オプションを付けて，tp1cobol.option を指定します。

  [指定形式]

  ```
  add.jvm.arg=-Dtp1cobol.option=OPTION [ : OPTION ]
  ```

  [usrconf.cfg 格納フォルダ]

  <CCインストール先>¥server¥usrconf¥ejb¥サーバ名

(2) オプションの設定方法（HP-UX/AIX/Linux 版）

オプションの設定方法を TP1/Client/W，TP1/Client/J および Cosminexus TP1 Connector に分けて説明します。各オプションの種類および説明は 5.3.2 以降をご覧ください。

(a) TP1/Client/W

HP-UX/AIX/Linux の環境変数に次の形式で設定します。（ただし，Linux(IPF)版，HP-UX(IPF) 版，Linux(x86/x64)版 02-10 以降，および AIX 版 02-10 以降では TP1/Client/W経由のアクセスは使用できません。）

[指定形式]

```
CBLJ2TP1OPT="[OPTION][: OPTION]"
export CBLJ2TP1OPT
```

[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 codeconv オプション，codeconvflag オプション，comp5 オプション，encode オプション，endian オプション，japanese オプションおよび unicode オプションです。

(b) TP1/Client/J および Cosminexus TP1 Connector

CBLJ2TP1OPT 環境変数にオプションを設定するのではなく，Java のシステムプロパティにオプション tp1cobol.option を設定する点が，TP1/Client/W と異なるのでご注意ください。オプションの設定値は同じです。

- Web コンテナサーバの場合

  usrconf.cfg ファイルの"add.jvm.arg"に-D オプションを付けて，tp1cobol.option を指定します。

  [指定形式]

  ```
  add.jvm.arg=-Dtp1cobol.option=OPTION [ : OPTION ]
  ```

  [usrconf.cfg 格納ディレクトリ]

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

- J2EE サーバの場合

  usrconf.cfg ファイルの"add.jvm.arg"に-D オプションを付けて，tp1cobol.option を指定します。

  [指定形式]

```
add.jvm.arg=-Dtp1cobol.option=OPTION [ : OPTION ]
```

  [usrconf.cfg 格納ディレクトリ]

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

5.3.2 codeconv オプション（Windows/HP-UX/AIX 版だけ）

(1) 機能

コード変換ライブラリ(Windows 版)および日立コード変換(HP-UX/AIX 版※)を使用した文字コード変換を行うことを指定します。

注※

本オプションは HP-UX(PA-RISC)版および AIX(32)環境だけ有効です。HP-UX(IPF)版および AIX(64)環境では有効となりません。

(2) 指定方法

(a) Windows 版

```
SET CBLJ2TP1OPT=codeconv( no | yes | UTF-16 )
```

codeconv オプション未指定時は，no として扱います。

(b) HP-UX/AIX 版

codeconv オプションは引用符(")で囲んで指定します。

```
CBLJ2TP1OPT="codeconv( no | yes | UTF-16 )"
export CBLJ2TP1OPT
```

codeconv オプション未指定時は，no として扱います。

yes および UTF-16 を指定すると，ホスト用文字コード変換を行います。yes の場合は SJIS(MS932)を，UTF-16 の場合は UTF-16BE を変換用文字コードとして使用します。

(3) 指定値の動作

codeconv オプションに yes または UTF-16 を指定した場合，以下の流れで文字コード変換を行います。

(a) setter,send

String(Unicode)－[変換]－＞ byte 配列(SJIS)※1－[codeconvflag に従い変換]※2－＞ byte 配列(EBCDIK/KEIS)

(b) getter,receive,recieve2

byte 配列(EBCDIK/KEIS)－[codeconvflag に従い変換]※2－＞ byte 配列(SJIS)※1－[変換]－＞ String(Unicode)

注※1

codeconv(UTF-16)指定時，byte 配列に格納する文字コードは UTF-16BE です。

注※2

変換時に CCO_UTF16 および CCO_BIGENDIAN_UTF16 オプションを使用します。

(4) 注意事項

- コード変換ライブラリおよび日立コード変換が組み込まれていない場合，このオプションを指定すると例外が発生します。
- このオプションに yes および UTF-16 を指定した場合，codeconvflag オプションに従って文字コード変換を行います。また，encode オプションおよび unicode オプションは無効となり，endian オプションは big が仮定されます。
- このオプションに no を指定した場合，codeconvflag オプションは無効となり，文字コード変換は行いません。
- 変換できない文字が指定された場合，setter/getter/receive/receive2/send メソッドで例外が発生することがあります。
- このオプションと dccm3 オプションを同時に指定した場合，dccm3 オプションの指定は無効となります。(Windows 版)
- Bean 生成ウィザードまたは Bean 生成ツールの 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生成した場合の日本語項目と，英数字項目で，日本語データが格納された場合に先頭の機能キャラクタの扱いが異なります。英数字項目の場合は，日本語データの開始を表す機能キャラクタが先頭に挿入されますが，日本語項目の場合は，日本語データの開始を表す機能キャラクタは先頭に挿入されません。日本語データを格納する場合は，英数字項目と日本語項目で格納されるデータが異なることに注意してください。

(5) この指定が有効となる項目

setter および getter 時に，このオプションで変換される項目は以下のとおりです。

- 文字列データ(String)：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
- 10 進データ(BigDecimal)：外部 10 進項目

2 進項目および内部浮動小数点項目はビッグエンディアン形式として扱われます。ただし，内部浮動小数点項目はホスト上で使用できる形式にはなりませんので，使用しないでください。

また，receive/receive2/send メソッドの引数に指定するメッセージもこの変換を行います。

5.3.3 codeconvflag オプション（Windows/HP-UX/AIX 版だけ）

(1) 機能

コード変換ライブラリ(Windows 版)および日立コード変換(HP-UX/AIX 版※)を使用した文字コード変換を行う際に使用する変換フラグを指定します。このオプションは，codeconv オプションに yes を指定した場合だけ有効です。

注※

本オプションは HP-UX(PA-RISC)版および AIX(32)環境だけ有効です。HP-UX(IPF)版および AIX(64)環境では有効となりません。

(2) 指定方法

(a) Windows 版

```
SET CBLJ2TP10PT=codeconvflag( フラグ[, フラグ]… )
```

codeconvflag オプション未指定時は，フラグ指定なしとして扱います。

(b) HP-UX/AIX 版

codeconvflag オプションは引用符(")で囲んで指定します。

```
CBLJ2TP10PT="codeconvflag( フラグ[, フラグ]… )"
export CBLJ2TP10PT
```

codeconvflag オプション未指定時は，フラグ指定なしとして扱います。

(3) 指定値の動作

フラグは，以下の文字列をコンマで区切って指定します。以下の文字列以外の指定は無効です。詳細は，マニュアル「コード変換ユーザーズガイド 解説・文法・操作書」(Windows 版)または「日立コード変換ユーザーズガイド 解説・文法・操作書」(HP-UX/AIX 版)をご覧ください。

表 5-1　codeconvflag オプションに指定できるフラグ

項番	フラグ	指定あり	指定なし
1	CCO_CNTLSHIFT	EBCDIC，EBCDIK または KEIS への変換で全角モード中に制御コードが出現した場合，シフトコード(0x0A,0x41)を付加します。	シフトコードを付加しません。
2	CCO_KEIS78	KEIS を’78 版で変換します。	KEIS を’83 版で変換します。
3	CCO_KEIS78BASIC	KEIS’78 版で，拡張文字セット 3 を使用しません。	KEIS’78 版で，拡張文字セット 3 を使用します。
4	CCO_TO_HALFSPACE	全角スペースを半角スペースに変換します。	全角スペースを全角スペースに変換します。
5	CCO_TO_FULLSPACE	半角スペース 2 バイトを全角スペースに変換します。	半角スペースを半角スペースに変換します。
6	CCO_UNDEFSWITCH	未定義コードを検出したとき，CCO_CONVBREAK の値は無視します。 データは変換しません。	未定義コードを検出したとき，CCO_CONVBREAK の指定に従います。
7	CCO_CONVBREAK	未定義コードを検出したとき，変換を中止します。	未定義コードを検出したとき，デフォルトコードに変換して処理を続行します。※
8	CCO_ADDSHIFT	2 バイトコードで終了した場合，KEIS へ変換したときは，シフトコード(0x0A,0x41)を付加します。	2 バイトコードで終了した場合，KEIS へ変換したときは，シフトコード(0x0A,0x41)を付加しません。
9	CCO_NOTUSEPUA	外字およびベンダ特殊文字を使用しません。	外字またはベンダ特殊文字を使用します。
10	CCO_FONTCHK	使用したコードにフォントがなければ，未定義コードとします。	指定したコードにフォントがなくても，対応するコードに変換します。
11	CCO_EBCDIC	EBCDIC で変換します。	EBCDIK で変換します。
12	CCO_STANDARD	EBCDIC または EBCDIK を標準コードで変換します。	EBCDIC または EBCDIK を準標準コードで変換します。

注※

SJIS->KEIS の場合：

1 バイトのデフォルトコードは 0x40

2 バイトのデフォルトコードは 0x4040

KEIS->SJIS の場合：

1 バイトのデフォルトコードは 0x20

2 バイトのデフォルトコードは 0x2020

フラグは，以下のように指定します。

codeconvflag(CCO_TO_HALFSPACE,CCO_ADDSHIFT)

未指定でよい場合は，codeconvflag オプションの指定は不要です。

(4) 注意事項

このオプションは，codeconv オプションに yes または UTF-16 指定時にだけ有効です。

5.3.4 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

(a) Windows 版

```
SET CBLJ2TP10PT=comp5( little | big)
```

comp5 オプション未指定時は，little として扱います。

(b) HP-UX/AIX 版

comp5 オプションは引用符（"）で囲んで指定します。

```
CBLJ2TP10PT="comp5( little | big )"
export CBLJ2TP10PT
```

comp5 オプション未指定時は，big として扱います。

(c) Linux 版

comp5 オプションは引用符（"）で囲んで指定します。

```
CBLJ2TP10PT="comp5( little | big )"
export CBLJ2TP10PT
```

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，以下の 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

5.3.5 dccm3 オプション

(1) 機能

DCCM3 環境下で稼動している COBOL プログラムを呼び出す際に必要な文字コード変換を行うことを指定します。このオプションは，Windows 版 TP1/Client/P でだけ有効です。その他の環境(HP-UX/AIX/Linux 版や TP1/Client/J など)では，このオプションは無効です。

(2) 指定方法

```
SET CBLJ2TP10PT=dccm3( no | yes )
```

dccm3 オプション未指定時は，no として扱います。

(3) 文字コード変換処理の流れ

dccm3 オプションに yes を指定した場合，以下の流れで文字コード変換を行います。

(a) setter,send

String(Unicode)－[変換]－＞ byte 配列(SJIS)－[dccm3flag に従い変換]－＞ byte 配列(EBCDIK/KEIS)

(b) getter,receive,recieve2

byte 配列(EBCDIK/KEIS)－[dccm3flag に従い変換]－＞ byte 配列(SJIS)－[変換]－＞ String(Unicode)

(4) 注意事項

- このオプションに yes を指定した場合，dccm3flag オプションに従って文字コード変換を行います。また，encode オプションおよび unicode オプションは無効となり，endian オプションは big が仮定されます。
- このオプションに no を指定した場合，dccm3flag オプションは無効となり，文字コード変換は行いません。
- 変換できない文字が指定された場合，setter/getter/receive/receive2/send メソッドで例外が発生することがあります。
- このオプションと codeconv オプションを同時に指定した場合，このオプションは無効となります。
- Bean 生成ウィザードまたは Bean 生成ツールの 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生成した場合の日本語項目と，英数字項目で，日本語データが格納された場合の先頭の機能キャラクタの扱いが異なります。英数字項目の場合は，日本語データの開始を表す機能キャラクタが先頭に挿入されますが，日本語項目の場合は，日本語データの開始を表す機能キャラクタは先頭に挿入されません。日本語データを格納する場合は，英数字項目と日本語項目で格納されるデータが異なることに注意してください。

(5) この指定が有効となる項目

setter および getter 時に，このオプションで変換される項目は以下のとおりです。

＜ TP1/Client/P 提供メソッドによって変換される項目＞

- 文字列データ(String)：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
- 10 進データ(BigDecimal)：外部 10 進項目

2 進項目および内部浮動小数点項目はビッグエンディアン形式として扱われます。ただし，内部浮動小数点項目はホスト上で使用できる形式にはなりませんので，使用しないでください。

また，receive/receive2/send メソッドの引数に指定するメッセージもこの変換を行います。

5.3.6 dccm3flag オプション

(1) 機能

DCCM3 環境下で稼動している COBOL プログラムを呼び出す際に必要な文字コード変換を行う際に使用する変換フラグを指定します。このオプションは，Windows 版 TP1/Client/P で dccm3 オプションに yes を指定した場合だけ有効です。

(2) 指定方法

```
SET CBLJ2TP1OPT=dccm3flag( フラグ )
```

dccm3flag オプション未指定時は，フラグ指定なしとして扱います。

(3) 指定値の動作

フラグは，以下の文字列を続けて指定します。詳細は，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」をご覧ください。

C：EBCDIC コードを使用します。未指定の場合，EBCDIK コードを使用します。

H：全角空白を半角空白 2 個に変換します。未指定の場合，全角空白のままにします。

7：KEIS78 コードを使用します。未指定の場合，KEIS83 コードを使用します。

I：無効なコードがあった場合，空白に変換します。未指定の場合，エラーにします。

T：タブコードを半角コードとして認識します。直前または直後のデータが全角コードの場合はシフトコードを付けます。未指定の場合，タブコードを半角コードとして認識しません。直前または直後のデータが全角コードの場合でもシフトコードは付けません。

L：制御コードを半角コードとして認識します。直前または直後のデータが全角コードの場合はシフトコードを付けます。未指定の場合，制御コードを半角コードとして認識しません。直前または直後のデータが全角コードの場合でもシフトコードは付けません。

フラグは，以下のように指定します。

```
dccm3flag(CH)
```

未指定でよい場合は，dccm3flag オプションの指定は不要です。

(4) 注意事項

このオプションは，dccm3 オプションに yes 指定時にだけ有効です。

5.3.7 encode オプション

(1) 機能

TP1/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) 指定方法

エンコード名は任意のエンコードで指定します。

(a) Windows 版

```
set CBLJ2TP10PT=encode(エンコード名)
```

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

(b) HP-UX/AIX/Linux 版

encode オプションは引用符（"）で囲んで指定します。

```
CBLJ2TP10PT="encode(エンコード名)"
export CBLJ2TP10PT
```

encode オプション未指定時は，以下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を，「表 5-2 デフォルトエンコード一覧」に記載しますのでご覧ください。

表 5-2 デフォルトエンコード一覧

OS	LANG 環境変数	コード系	デフォルトエンコード
HP-UX	ja_JP.SJIS	シフト JIS	SJIS
	ja_JP.eucJP	日本語 EUC	EUC_JP
AIX	Ja_JP	シフト JIS	Cp943C
	ja_JP	日本語 EUC	Cp33722C
Linux	ja_JP	日本語 EUC	EUC_JP_LINUX

OS	LANG 環境変数	コード系	デフォルトエンコード
	ja_JP.eucJP ja_JP.ujis		
	ja_JP.UTF-8 ja_JP.utf8	UTF-8	UTF-8※
Windows	−	シフト JIS	MS932

注※

Red Hat Enterprise Linux 4，Red Hat Enterprise Linux 5 および Red Hat Enterprise Linux Server 6 が対象 OS となる Linux(x86/x64)版 02-10 以降だけ。

(6) エンコード対象となる項目

1. COBOL SPP/MHP 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
2. cltin メソッドの引数 defpat
3. setConnectInf メソッドの引数 inf
4. receive メソッドの引数 buff
5. receive2 メソッドの引数 buff
6. send メソッドの引数 buff
7. acceptNotification メソッドの引数 defpath，および inf
8. cancelNotification メソッドの引数 defpath，および inf
9. setRaphost メソッドの引数 raphost
10. getRaphost メソッドの引数 raphost
11. openNotification メソッドの引数 defpath
12. chainedAcceptNotification メソッドの引数 inf

5.3.8　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/HP-UX/AIX/Linux で異なります。Windows/Linux ではリトルエンディアン形式で，HP-UX/AIX ではビッグエンディアン形式となります。Cosminexus と OpenTP1 間で異なる OS でバイナリ形式のデータを流通させたい場合に，このオプションを使って，流通させたいエンディアン形式に変換することができます。

詳細については，「9.3　Windows/UNIX の数字データ格納形式の相違」をご覧ください。

(2)　指定方法

(a)　Windows 版

```
SET CBLJ2TP10PT=endian( little | big )
```

endian オプション未指定時は，little として扱います。

(b)　HP-UX/AIX 版

endian オプションは引用符（"）で囲んで指定します。

```
CBLJ2TP10PT="endian( little | big )"
export CBLJ2TP10PT
```

endian オプション未指定時は，big として扱います。

(c)　Linux 版

endian オプションは引用符（"）で囲んで指定します。

```
CBLJ2TP10PT="endian( little | big )"
export CBLJ2TP10PT
```

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，以下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
- 以下の Bean で受け渡す，COMPUTATIONAL-5/COMP-5 指定の 2 進項目

1. TP1/COBOL adapter for Cosminexus Version 2 02-05 未満で生成した Bean
2. TP1/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

Windows 版 TP1/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，-BigEndian,Float コンパイラオプション※の両方とも指定するようにしてください。

注※

COBOL85 をご使用の場合は -Bb コンパイラオプション，-Fb コンパイラオプションを指定してください。

5.3.9 japanese オプション

(1) 機能

日本語項目の扱いについて指定します。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 TP1/COBOL アクセス用 Bean の日本語項目および日本語編集項目に対して有効です。

(2) 指定方法

(a) Windows 版

```
SET CBLJ2TP10PT=japanese( 1 | 2 )
```

japanese オプション未指定時は，2 として扱います。

(b) HP-UX/AIX/Linux 版

japanese オプションは引用符（"）で囲んで指定します。

```
CBLJ2TP10PT="japanese( 1 | 2 )"
export CBLJ2TP10PT
```

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，すべて半角空白で補います。

(4) 注意事項

このオプションは，以下の 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. TP1/COBOL adapter for Cosminexus Version 2 02-04 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

5.3.10 unicode オプション（Windows/HP-UX/AIX /Linux(x86/x64)版だけ)

(1) 機能

TP1/COBOL アクセスの，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) 指定方法

(a) Windows 版

```
SET CBLJ2TP10PT=unicode( no | big | little )
```

unicode オプション未指定時は，no として扱います。

(b) HP-UX/AIX 版

unicode オプションは引用符(")で囲んで指定します。

```
CBLJ2TP10PT="unicode( no | big | little )"
export CBLJ2TP10PT
```

unicode オプション未指定時は，no として扱います。

(c) Linux(x86/x64)版 02-10 以降

[usrconf.cfg 指定形式]

```
add.jvm.arg=-Dtp1cobol.option=unicode( no | big | little )
```

unicode 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

unicode(no)：文字列データを encode オプション指定にしたがってエンコードする。

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

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，TP1/COBOL アクセス用 Bean を生成してください。

(5) エンコード対象となる項目

1. COBOL SPP/MHP 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
2. 以下は英数字項目で受け渡すことから，UTF-8 でエンコードします。
   - receive メソッドの引数 buff
   - receive2 メソッドの引数 buff
   - send メソッドの引数 buff

(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

英数字項目：UTF-8 の半角空白(0x20)を補います。

日本語項目：japanese オプションの指定により，補う文字が異なります。big 指定時は以下の文字を補います。

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。

5.4　CBLJ2TP1_DDUMP 環境変数

5.4.1　機能

TP1/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表示する機能です。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た場合に，ダンプファイル（表示した引数情報を格納するファイル）を出力します。機能の詳細については，「7.2　引数情報表示機能」をご覧ください。

5.4.2　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指定します。

(1)　TP1/Client/P の場合

Windows の環境変数に設定します。

```
SET CBLJ2TP1_DDUMP=フォルダ名[,filesize(nnnn)]
```

(2)　TP1/Client/W の場合

CBLJ2TP1_DDUMP 環境変数は引用符（"）で囲んで指定します。

```
CBLJ2TP1_DDUMP="フォルダ名[,filesize(nnnn)]"
export CBLJ2TP1_DDUMP
```

(3)　TP1/Client/J および Cosminexus TP1 Connector の場合

Java のシステムプロパティにオプション tp1cobol.ddump を指定します。

- web コンテナサーバの場合

  usrconf.cfg ファイルの”add.jvm.arg”に-Dtp1cobol.ddump を指定します。

  ```
  add.jvm.arg=-Dtp1cobol.ddump=フォルダ名[,filesize(nnnn)]
  ```

  [usrconf.cfg 格納フォルダ]

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

- J2EE サーバの場合

  usrconf.cfg ファイルの”add.jvm.arg”に-Dtp1cobol.ddump を指定します。

  ```
  add.jvm.arg=-Dtp1cobol.ddump=フォルダ名[,filesize(nnnn)]
  ```

[usrconf.cfg 格納フォルダ]

<CC インストール先>¥server¥usrconf¥ejb¥サーバ名

filesize(nnnn)を指定する際は，カンマ(,)に続けて指定します。

フォルダ名：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。(Windows 版)

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。(HP-UX/AIX/Linux 版)

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 の指定がない場合は，10MB を仮定します。

filesize の指定可能な値は，1～2000 です。範囲外の値を指定した場合は，10 を仮定します。

5.4.3 指定例

TP1/Client/P を使用する場合の指定例です。

ファイルサイズの上限未指定の例

```
SET CBLJ2TP1_DDUMP=c:¥users
```

ファイルサイズの上限に 100MB を指定した例

```
SET CBLJ2TP1_DDUMP=c:¥users,filesize(100)
```

5.5　MHP に対してメッセージを送信するための設定

TP1/COBOL アクセス用 Bean の send メソッドを使用して MHP に対してメッセージを送信する場合，メッセージの先頭に 4 バイトのメッセージ長エリアを付加してメッセージを送信します。このメッセージを MHP で正しく受け取るには，OpenTP1 の受信メッセージの組み立て機能を使用してください。受信メッセージの組み立て機能の詳細は，マニュアル「分散トランザクション処理機能 OpenTP1 プロトコル TP1/NET/TCP/IP 編」をご覧ください。

6 プログラムの実行

プログラムを実行するためには，作成してきた TP1/COBOL と Java プログラムを適切な場所に配置する必要があります。

6.1　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。

6.1.1　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。

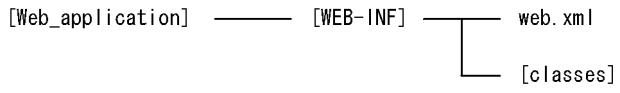
[Windows 版]

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

[HP-UX/AIX/Linux 版]

```
<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。

6.1.2　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（Eclipse をご使用の場合は，WAR プロジェクトで生成することもできます)。

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

構成の説明

[Web_application]は任意の名称

6.2　サーブレットの登録

この説明では，直接 web.xml ファイルを作成する方法を記述していますが，Eclipse をご使用の場合は，WAR プロジェクトで自動生成することができます。

6.2.1　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 版]

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

[HP-UX/AIX/Linux 版]

```
<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
```

6.2.2　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することができます）。

6.2.3　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。

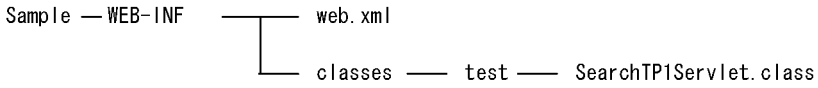
6.2.4　web.xml の作成例

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。

(1) Web アプリケーション構成

- アプリケーション名:Sample
- Servlet 名称:SearchTP1Servlet
- Package 名:test
- Mapping 名:/test.SearchTP1

(2) ディレクトリ構成



(3) web.xml ソースコード例

```
<web-app>
     <servlet>
          <servlet-name>SearchTP1Servlet</servlet-name>
          <servlet-class>
            test. SearchTP1Servlet
          </servlet-class>
     </servlet>
     <servlet-mapping>
          <servlet-name>
            SearchTP1Servlet
          </servlet-name>
          <url-pattern>
            /test. SearchTP1Servlet
          </url-pattern>
     </servlet-mapping>
</web-app>
```

6.3　プログラムの実行

当該 SPP/MHP を起動した後に，Cosminexus 環境で作成したアプリケーションを実行します。

6.3.1　TP1 サーバ側（SPP の起動方法）

(1)　開始

COBOL SPP の実行形式を OpenTP1 インストールディレクトリ下の aplib ディレクトリに格納してください。その後，次のいずれかの方法で起動してください。SPP はユーザサーバ単位で開始します（ただし，ここでは TP1/Server Base または TP1/LiNK がすでに起動されていることを前提に記述しています）。

- コマンドラインから dcsvstart コマンドを入力して開始します。
  （例）dcsvstart -u　ユーザサーバ名
- GUI 画面が用意されているシステムなら[アプリケーション環境 SPP]画面で自動起動などの設定を行ってください。

(2)　終了

SPP が終了するのは次の場合です。

- コマンドラインから dcsvstop コマンドを入力して終了させます。
  （例）dcsvstop　　ユーザサーバ名
- GUI 画面が用意されているシステムなら[TP1/LiNK アプリケーション管理 SPP]画面で停止ボタンを使って停止できます。

ここで紹介している方法は一部分ですので詳しい SPP の開始/終了方法については，マニュアル「分散トランザクション処理機能 OpenTP1 運用と操作」をご覧ください。

6.3.2　TP1 サーバ側（MHP の起動方法）

(1)　開始

COBOL MHP の実行形式を OpenTP1 インストールディレクトリ下の aplib ディレクトリに格納してください。その後，次のどれかの方法で起動してください。

MHP はユーザサーバ単位で開始します（ただし，ここでは TP1/LiNK および TP1/Messaging，または，TP1/Server Base および TP1/Message Control がすでに起動されていることを前提に記述しています）。

- コマンドラインから dcsvstart コマンドを入力して開始します。

（例）dcsvstart -u　ユーザサーバ名

- GUI 画面が用意されているシステムなら[アプリケーション管理 MHP]画面で自動起動などの設定を行ってください。

(2) 終了

MHP が終了するのは次の場合です。

- コマンドラインから dcsvstop コマンドを入力して終了させます。
  （例）dcsvstop　ユーザサーバ名
- GUI 画面が用意されているシステムなら[アプリケーション管理 MHP]画面で停止ボタンを使って停止できます。

ここで紹介している方法は一部分ですので詳しい MHP の開始/終了方法については，マニュアル「TP1/Messaging 使用の手引」および「分散トランザクション処理機能 OpenTP1 運用と操作」をご覧ください。

6.3.3 Cosminexus 側

作成したアプリケーションを実行します。

作成したおのおののプログラムをそれぞれが対応した場所に格納し，Web ブラウザを起動すれば実行できます。

Windows 環境では Web サーバに Internet Information Server を用いて，HP-UX/AIX/Linux 環境では Web サーバに Hitachi Web Server を用いて説明します。

なお，格納場所については，Cosminexus が対応した Web サーバおよび OpenTP1 がインストールされている場所を前提としています。

(1) Web コンテナサーバの場合

Web アプリケーションサーバに 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以下に示します。

(a) Windows 版

表 6-1　プログラムの格納場所例

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	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
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

項番	作成したファイル	格納場所
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
6	COBOL SPP/MHP ファイル	¥DCDIR¥aplib※

注※

¥DCDIR:TP1/Server Base または TP1/LiNK インストールフォルダ，¥Web コンテナサーバは<CC インストール先>¥web¥containers です。

(b) HP-UX/AIX/Linux 版

表 6-2 プログラムの格納場所例

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※
6	COBOL SPP/MHP ファイル	/DCDIR/aplib※

注※

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

/DCDIR は TP1/Server Base または TP1/LiNK インストールディレクトリです。

6.3.4 Cosminexus Version 6 06-50 および uCosminexus Application Server を使用する場合の注意事項

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」に追記してください。

【値】

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

「保護区リストファイル」の詳細は，マニュアル「Cosminexus リファレンス　定義編」，もしくはマニュアル「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）」を参照してください。

7 プログラムのデバッグ

この章では，プログラムのデバッグについて説明します。

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。

7.1.1　Servlet のデバッグ

通常の Servlet と同様な手順でデバッグすることができます。また，Cosminexus のログを参照できます。

詳細については，マニュアル「Cosminexus　アプリケーション開発ガイド」，もしくはマニュアル「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」を参照して必要な設定を行ってください。

7.1.2　COBOL SPP/MHP のデバッグ

COBOL プログラムのコンパイル時に，-Tdinf コンパイラオプション※を指定し，TP1/COBOL の実行時に環境変数（CBLTDEXEC=TD）を設定することにより，COBOL2002 テストデバッグを使用してデバッグすることができます。

コンパイラオプションと環境変数の詳細については，Windows 環境ではマニュアル「COBOL2002 ユーザーズガイド」および「COBOL2002 操作ガイド」，HP-UX/AIX/Linux 環境ではマニュアル「COBOL85 使用の手引」をご覧ください。

注※

COBOL85 をご使用の場合は，-T5 コンパイラオプションを指定してください。

7.1.3　UAP トレースの活用

UAP トレースは，TP1/Client が用意しているトレース機能です。TP1/COBOL アクセスでは，この UAP トレースを利用した障害調査ができます。

UAP トレースの機能と使い方については，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」または「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」および「分散トランザクション処理機能 OpenTP1 テスタ・UAP トレース使用の手引」をご覧ください。

7.2　引数情報表示機能

7.2.1　機能概要

TP1/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表示する機能です。CBLJ2TP1_DDUMP 環境変数※（TP1/Client/P）または Java のシステムプロパティのオプション tp1cobol.ddump※（TP1/Client/J または Cosminexus TP1 Connector）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，「5.4　CBLJ2TP1_DDUMP 環境変数」をご覧ください。

7.2.2　ダンプファイルの出力例

以下に，ダンプファイルの出力例を示します。詳細な出力例は「7.2.5　出力フォーマット」をご覧ください。

```
2007/09/28 11:32:11 (1) SearchTP1 TP1/COBOL 拡張 Server Run Time System for Cosminexus Versi
on 2  02-10  *** Data Area Dump ***
2007/09/28 11:32:11 (1) SearchTP1 < package  : test >
2007/09/28 11:32:11 (1) SearchTP1 < name  : setP_numberI >
2007/09/28 11:32:11 (1) SearchTP1 setter Before Data  J-Type:[String]  size: 12
2007/09/28 11:32:11 (1) SearchTP1 00000000: 41 42 43 44 45 46 47 48 - 49 4A 4B 4C
   *ABCDEGFHIJKL    *
2007/09/28 11:32:11 (1) SearchTP1 setter After Data   CBL-Type:[X(50)]  size: 50 location: 4
2007/09/28 11:32:11 (1) SearchTP1 00000000: 41 42 43 44 45 46 47 48 - 49 4A 4B 4C 20 20 20 2
0  *ABCDEFGHIJKL    *
2007/09/28 11:32:11 (1) SearchTP1 00000010: 20 20 20 20 20 20 20 20 - 20 20 20 20 20 20 20 2
0  *                *
```

7.2.3　ダンプファイル

ダンプファイルは，CBLJ2TP1_DDUMP 環境変数（TP1/Client/P）または Java のシステムプロパティのオプション tp1cobol.ddump（TP1/Client/J または Cosminexus TP1 Connector）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，TP1/COBOL アクセス用 Bean の種類によって，以下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	TP1/Client/P 用の TP1/COBOL アクセス用 Bean の場合	J2tp1_1.log , J2tp1_2.log
2	TP1/Client/J 用の TP1/COBOL アクセス用 Bean の場合	J2tp1j_1.log , J2tp1j_2.log

項番	対応するアクセス用 Bean	ファイル名
3	Cosminexus TP1 Connector 用の TP1/COBOL アクセス用 Bean の場合	J2tp1c_1.log , J2tp1c_2.log
4	TP1/COBOL SOAP クライアント用 Bean の場合	J2tp1sc_1.log, J2tp1sc_2.log
5	TP1/COBOL SOAP サーバ用 Bean の場合	J2tp1sv_1.log, J2tp1sv_2.log

この機能では，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た場合に行います。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。ファイルサイズの上限値に指定できる値は，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。

7.2.4　制限事項

この機能は，TP1/COBOL adapter for Cosminexus Version 2 以降で生成した TP1/COBOL アクセス用 Bean で使用することができます。TP1/COBOL adapter for Cosminexus 01-xx で生成した TP1/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。

7.2.5　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，以下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

   TP1/COBOL アクセス用 Bean を識別するための番号で，TP1/COBOL アクセス用 Bean を呼び出すごとに番号をカウントアップします。

   （番号は 1～2,147,483647 で，超えた場合は 1 となります）

4. Bean 名称：TP1/COBOL アクセス用 Bean のクラス名称

(3)　ヘッダ部

```
TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 VV-RR[-ZZ]  *** Data Area Dum
p ***
< package : *** 1 *** >
```

*** 1 *** ：TP1/COBOL アクセス用 Bean のパッケージ名称

(4)　データ領域ダンプ

setter の例

```
< name : setXxxx >
setter  Before Data  J-Type: [String]  size :50
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   *                *
```

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「7.2.6　setter/getter 引数情報表示時のデータ属性情報」をご覧ください）。

J-Type：

Java の定義を表します（詳細は「7.2.6　setter/getter 引数情報表示時のデータ属性情報」をご覧ください）。

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。

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）。

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。

ダンプの表示項目の説明

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。

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
```

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              *
```

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

以下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) TP1/COBOL adapter for Cosminexus 01-xxで生成したTP1/COBOLアクセス用Beanを実行した場合

```
TP1/COBOL Extended RTS for Cosminexus 02-10 *** Data Area Dump ***
  ### This program is outside an object ###
```

(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 ###
```

7.2.6 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，以下に記載します。

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) USAGE COMP※8	S9(4)※3 COMP	Short
		S9(4) USAGE COMP-5※12	S9(4) 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) USAGE COMP※8	9(4)※3 COMP	Short
		9(4) USAGE COMP-5※12	9(4) COMP-5	

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
4	5～9 けたの小数を含まない符号付き 2 進項目	S9(9) USAGE COMP※8	S9(9)※3 COMP	Integer
		S9(9) USAGE COMP-5※12	S9(9) 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) USAGE COMP※8	9(9)※3 COMP	Integer
		9(9) USAGE COMP-5※12	9(9) 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) USAGE COMP※8	S9(18)※3 COMP	Long
		S9(18) USAGE COMP-5※12	S9(18) 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) USAGE COMP※8	9(18)※3 COMP	Long
		9(18) USAGE COMP-5※12	9(18) COMP-5	
8	小数を含む符号付き 2 進項目	S9(17)V9(1) USAGE COMP※8	S9(17)V9(1)※4 COMP	BigDecimal
		S9(17)V9(1) USAGE COMP-5※12	S9(17)V9(1) COMP-5	
9	小数を含む符号なし 2 進項目	9(17)V9(1) USAGE COMP※8	9(17)V9(1)※4 COMP	BigDecimal
		9(17)V9(1) USAGE COMP-5※12	9(17)V9(1) COMP-5	
10	左独立符号付き外部 10 進項目	S9(18) SIGN LEADING SEPARATE	S9(18)※4 LEADING SEPARATE	BigDecimal
11	右独立符号付き外部 10 進項目	S9(18) SIGN TRAILING SEPARATE	S9(18)※4 TRAILING SEPARATE	BigDecimal
12	左符号付き外部 10 進項目	S9(18) SIGN LEADING	S9(18)※4 LEADING	BigDecimal
13	右符号付き外部 10 進項目	S9(18) SIGN TRAILING	S9(18)※4 TRAILING	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
13	符号なし外部 10 進項目	9(18)	9(18)※4	BigDecimal
14	単精度内部浮動小数点項目	USAGE COMP-1	COMP-1	Float
15	倍精度浮動小数点項目	USAGE COMP-2	COMP-2	Double
16	符号付き内部 10 進項目	S9(18) USAGE COMP-3※9	S9(18)※4 COMP-3	BigDecimal
17	符号なし内部 10 進項目	9(18) USAGE COMP-3※9	9(18)※4 COMP-3	BigDecimal
18	英数字項目（バイト配列）	X　英数字項目※6	X(n)※5 Byte	byte[]
		N　日本語項目※7※10	X(n)※5 Byte	byte[]
19	日本語項目（文字列）	N　日本語項目※7※11	N(n)※4	String
20	集団項目（バイト配列）	集団項目※13	G(n)※5 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

けた数は固定で表示します。(02-05 未満で生成した Bean からの呼び出し時)

けた数は正しく表示します。(02-05 未満で生成した Bean からの呼び出し時)

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 TP1/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

TP1/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

TP1/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

TP1/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

TP1/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 SPP/MHP を呼び出すための API

この章では，COBOL SPP/MHP を呼び出すための API（ユーザアプリケーションプログラムインタフェース）について説明します。

8.1　COBOL SPP/MHP を呼び出すために使用する API

8.1.1　TP1/Client/P または TP1/Client/W を経由して COBOL SPP/MHP を呼び出す場合

以下に示す API を使用して，COBOL SPP/MHP を呼び出すことができます。

表 8–1　TP1/Client/P,W を経由して COBOL SPP/MHP を呼び出す際に使用する API

項番	パッケージ名	クラス名	概要
1	任意※	任意※	生成した TP1/COBOL アクセス用 Bean で，受け渡す引数の設定，参照および COBOL SPP/MHP 呼び出しを行います。
2	jp.co.hitachi_sk.j2cb	TP1Access	ユーザ認証など，TP1/Client/P，W が提供するサービスを呼び出します。TP1/COBOL 基本 Bean として提供します。
3		J2CBException	COBOL SPP/MHP で発生したエラー情報は，このクラスで提供するメソッドで取得することができます。

注※

パッケージ名およびクラス名は，TP1/COBOL アクセス用 Bean 生成ウィザードまたは TP1/COBOL アクセス用 Bean 生成ツールの 3/3 画面で指定した名称となります。

TP1/COBOL アクセス用 Bean で使用できるメソッドについては，「8.2　TP1/COBOL アクセス用 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)」をご覧ください。TP1/COBOL 基本 Bean で使用できるメソッドについては，「8.3　TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)」をご覧ください。エラー情報取得方法については，「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.1.2　TP1/Client/J を経由して COBOL SPP/MHP を呼び出す場合

以下に示す API を使用して，COBOL SPP/MHP を呼び出すことができます。

表 8–2　TP1/Client/J を経由して COBOL SPP/MHP を呼び出す際に使用する API

項番	パッケージ名	クラス名	概要
1	任意※	任意※	生成した TP1/COBOL アクセス用 Bean で，受け渡す引数の設定，参照および COBOL SPP/MHP 呼び出しを行います。
2	jp.co.hitachi_sk.j2cb	J2CBException	COBOL SPP/MHP で発生したエラー情報は，このクラスで提供するメソッドで取得することができます。

注※

パッケージ名およびクラス名は，TP1/COBOL アクセス用 Bean 生成ウィザードまたは TP1/COBOL アクセス用 Bean 生成ツールの 3/3 画面で指定した名称となります。

TP1/COBOL アクセス用 Bean で使用できるメソッドについては，「8.4　TP1/COBOL アクセス用 Bean ユーザインタフェース API(TP1/Client/J)」をご覧ください。ユーザ認証などを行うクラス（TP1/Client/P または TP1/Client/W で提供されていた TP1/COBOL 基本 Bean）は，提供しません。TP1/Client/J が提供する API を直接使用してください。エラー情報取得方法については，「8.8 J2CBException ユーザインタフェース API」をご覧ください。

8.1.3　Cosminexus TP1 Connector を経由して COBOL SPP/MHP を呼び出す場合

以下に示す API を使用して，COBOL SPP/MHP を呼び出すことができます。

表 8-3　Cosminexus TP1 Connector を経由して COBOL SPP/MHP を呼び出す際に使用する API

項番	パッケージ名	クラス名	概要
1	任意※	任意※	生成した TP1/COBOL アクセス用 Bean で，受け渡す引数の設定，参照および COBOL SPP/MHP 呼び出しを行います。
2	jp.co.hitachi_sk.j2cb	J2CBException	COBOL SPP/MHP で発生したエラー情報は，このクラスで提供するメソッドで取得することができます。

注※

パッケージ名およびクラス名は，TP1/COBOL アクセス用 Bean 生成ウィザードまたは TP1/COBOL アクセス用 Bean 生成ツールの 3/3 画面で指定した名称となります。

TP1/COBOL アクセス用 Bean で使用できるメソッドについては，「8.5　TP1/COBOL アクセス用 Bean ユーザインタフェース API(Cosminexus TP1 Connector)」をご覧ください。ユーザ認証などを行うクラス（TP1/Client/P または TP1/Client/W で提供されていた TP1/COBOL 基本 Bean）は，提供しません。Cosminexus TP1 Connector が提供する API を直接使用してください。エラー情報取得方法については，「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.1.4　Cosminexus SOAP アプリケーションを経由して COBOL SPP/MHP を呼び出す場合（Windows/HP-UX/AIX 版だけ）

以下に示す API を使用して，COBOL SPP/MHP を呼び出すことができます。

表 8-4　Cosminexus SOAP アプリケーションを経由して COBOL SPP/MHP を呼び出す際に使用する API

項番	パッケージ名	クラス名	概要
1	任意※1	任意※	TP1/COBOL SOAP サービスクラス生成機能で生成した Bean で，受け渡す引数の設定および参照を行います。
2	jp.co.hitachi_sk.j2cb	TP1SOAPAccess	ユーザ認証など，TP1/Client/P が提供するサービスを呼び出します。TP1/COBOL SOAP 基本 Bean として提供します。
3		J2CBException	COBOL SPP/MHP で発生したエラー情報は，このクラスで提供するメソッドで取得することができます。

注※

パッケージ名およびクラス名は，TP1/COBOL SOAP サーバ用 Bean 生成ウィザードで指定した名称となります。

TP1/COBOL SOAP サービスクラス用 Bean で使用できるメソッドについては，「8.6　TP1/COBOLSOAP サービスクラス用 Bean ユーザインタフェース API」をご覧ください。

TP1/COBOL SOAP サービスクラス用基本 Bean で使用できるメソッドについては，「8.7　TP1/COBOL SOAP サービスクラス用基本 Bean ユーザインタフェース API」をご覧ください。

エラー情報取得方法については，「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.2 TP1/COBOL アクセス用 Bean ユーザインタフェース API（TP1/Client/P および TP1/Client/W）

TP1/COBOL アクセス用 Bean で使用できるメソッドの一覧を以下に記載します。

表 8-5　メソッド一覧

項番	メソッド名	機能
1	call	遠隔サービスの要求
2	callTo	通信先を指定した遠隔サービスの要求
3	getXxxO	出力引数データの取得
4	receive	メッセージの受信
5	receive2	メッセージの受信（障害時メッセージ受信）
6	send	メッセージの送信
7	setInLenFollowSetData	setter 発行時に入力引数長設定（このメソッドは，TP1/COBOL adapter for Cosminexus Version 2 で生成した TP1/COBOL アクセス用 Bean で有効です。TP1/COBOL adapter for Cosminexus で生成した TP1/COBOL アクセス用 Bean では無効です。）
8	setXxxI	入力引数データの設定

8.2.1 COBOL の各データ項目の設定方法と設定値について

Java のデータと受け渡しする COBOL の各データ項目の setter/getter 規則を以下にまとめます。

表 8-6　各データ項目に対応する表示文字列

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 英数字項目 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※6	1. String 文字列を encode オプション※1 に従って変換したバイト配列にする。 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。	COBOL のデータ値を encode オプション※1 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（String）
日本語項目 日本語編集項目※7	1. String 文字列を encode オプション※1 に従って変換したバイト配列にする。 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	1. COBOL のデータ値を encode オプション※1 に従って変換した String オブジェクトを作成する。 2. codeconv オプション※3 および dccm3 オプション※4 の指定があると，文字列前後	日本語データ（String）

COBOL データ項目	setter 変換規則	getter 変換規則	Java データ属性
	※5 の指定により，全角空白または半角空白が埋められる。 3. codeconv オプション※3 および dccm3 オプション※4 の指定があると，コード変換した文字列前後の機能キャラクタを削除して設定する。	に機能キャラクタを付加してコード変換した String オブジェクトを作成する。	
英字項目 英数字項目 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※6※8 集団項目※10	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。	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（byte[]）
単精度内部浮動項目	endian オプション※2 に従ってデータの並び順を変更する。	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（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	endian オプション※2 または comp5 オプション※9 に従ってデータの並び順を変更する。	endian オプション※2 または comp5 オプション※9 に従ってデータの並び順を変更する。	Short データ（Short）
5～9 けたの小数を含まない 2 進項目			Integer データ（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	• 小数点位置を合わせた 2 進データを作成する。 • endian オプション※2 に従ってデータの並び順を変更する。	• 小数点位置を合わせた数字データを作成する。 • endian オプション※2 に従ってデータの並び順を変更する。	10 進データ（BigDecimal）
外部１０進	• 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。 • 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 0x’7X’とする。 (*)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	• 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 0x’3X’とする。 • 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）

COBOL データ項目	setter 変換規則	getter 変換規則	Java データ属性
	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。		
内部１０進	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。 (*)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。 (*)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。 確保される領域長は，けた数から算出できる。 領域長=(桁数 / 2) + 1 (小数点以下は切り捨て) 偶数けた数の場合は，先頭の４ビットは 0 となる。 (*)符号は以下で表現する。 ・正符号：0xC （ビット値=1100） ・負符号：0xD （ビット値=1101） ・符号無：0xF （ビット値=1111） 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。	• 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。 • 符号以外の数値を数字データにし，符号を付加する。 (*)偶数けた数の場合は，先頭の４ビットは使用しない。 (*)数字部分の値が 0～9 以外の場合，値は保証できない。	10 進データ（BigDecimal）

注※1

encode オプションについては「5.3.7　encode オプション」をご覧ください。

注※2

endian オプションについては「5.3.8　endian オプション」をご覧ください。

注※3

codeconv オプションについては「5.3.2　codeconv オプション（Windows/HP-UX/AIX 版だけ）」をご覧ください。

注※4

dccm3 オプションについては「5.3.5　dccm3 オプション」をご覧ください。

注※5

japanese オプションについては「5.3.9　japanese オプション」をご覧ください。

注※6

Bean 生成ウィザードまたは Bean 生成ツール 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 TP1/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※7

Bean 生成ウィザードまたは Bean 生成ツール 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 TP1/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※8

Bean 生成ウィザードまたは Bean 生成ツール 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[])」に変更した場合，バイト配列データとして扱います。

注※9

TP1/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，「5.3.4 comp5 オプション」をご覧ください。

注※10

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。

8.2.2　TP1/COBOL アクセス用 Bean ユーザインタフェース API

TP1/COBOL アクセス用 Bean で使用できるメソッドについて説明します。

(1)　遠隔サービスの要求

(a)　遠隔サービスの要求

```
[ callメソッド ]

public void call(int cltid,
                 java.lang.String group,
                 java.lang.String service,
                 int flags)
          throws J2CBException
```

パラメタ：

cltid - クライアント ID

group - サービスグループ名

service - サービス名

flags - RPC の形態を指定

：TP1Const.DCNOFLAGS - 同期応答型 RPC

：TP1Const.DCRPC_NOREPLY - 非応答型 RPC

：TP1Const.DCRPC_CHAINED - 連鎖 RPC

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

出力引数の指定がない場合，flags に TP1Const.DCRPC_NOREPLY 以外を指定してこのメソッドを実行すると，例外が発生します。

(b)　通信先を指定した遠隔サービスの要求

```
[ callToメソッド ]

public void callTo(int cltid,
                   java.lang.String hostnm,
                   int scdport,
                   java.lang.String group,
                   java.lang.String service,
                   int flags)
            throws J2CBException
```

パラメタ：

cltid - クライアント ID

hostnm - サービス要求先のホスト名

scdport - サービス要求先のホストに存在するスケジュールサービスのポート番号

group - サービスグループ名

service - サービス名

flags - RPC の形態を指定

：TP1Const.DCNOFLAGS - 同期応答型 RPC

：TP1Const.DCRPC_NOREPLY - 非応答型 RPC

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

出力引数の指定がない場合，flags に TP1Const.DCRPC_NOREPLY 以外を指定してこのメソッドを実行すると，例外が発生します。

(2)　出力引数データの取得

```
[getXxxOメソッド]

public Object getXxxO( [ int dim1 { , int dim2 } … ] ) throws J2CBException
```

メソッド名 getXxxO 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。([setXxxI メソッド]の例をご覧ください)

パラメタ：

dim1,dim2… ：(添字が必要な場合の）各次元の添字

戻り値：

取得できたデータオブジェクト

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(3)　メッセージの送受信

(a)　メッセージの受信

```
[ receiveメソッド ]

public void receive(int cltid,
                    int timeout,
                    int flags)
             throws J2CBException
```

パラメタ：

cltid - クライアント ID

timeout - メッセージ受信時の最大待ち時間（秒）

flags - メッセージ受信後に，コネクションを解放するかどうかを指定

：TP1Const.DCNOFLAGS - 解放しない

：TP1Const.DCCLT_RCV_CLOSE - 解放する

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

出力引数を getter で取得できる点が，「8.3　TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)」で提供される receive メソッドと異なります。

(b)　メッセージの受信（障害時メッセージ受信）

```
[ receive2メソッド ]

public void receive2(int cltid,
                     int timeout,
                     int flags)
         throws J2CBException
```

パラメタ：

cltid - クライアント ID

timeout - メッセージ受信時の最大待ち時間（秒）

flags - メッセージ受信後に，コネクションを解放するかどうかを指定

：TP1Const.DCNOFLAGS - 解放しない

：TP1Const.DCCLT_RCV_CLOSE - 解放する

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

出力引数を getter で取得できる点が，「8.3　TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)」で提供される receive2 メソッドと異なります。

(c)　メッセージの送信

```
[ sendメソッド ]

public void send(int cltid,
                 java.lang.String hostname,
                 int portnum,
                 int flags)
         throws J2CBException
```

パラメタ：

cltid - クライアント ID

hostname - 接続する MHP が存在するノードのホスト名

portnum - MHP のポート番号

flags - メッセージ送信後に，コネクションを解放するかどうかを指定

：TP1Const.DCNOFLAGS - 解放しない

：TP1Const.DCCLT_SND_CLOSE - 解放する

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

入力引数を setter で設定できる点が，「8.3　TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)」で提供される send メソッドと異なります。

(4)　setter 発行時の入力引数長設定

```
[setInLenFollowSetDataメソッド]

public void setInLenFollowSetData()
```

TP1/Client に渡す入力引数の長さをこのメソッドを呼び出した直後の setXxx(obj)の末尾までの長さとします。また．設定するデータが英数字項目（String または byte[]）の場合は，obj を英数字項目に転記後の末尾までの長さとします。

このメソッドを呼び出す例

```
01  ARG.
 02 ARGCOMP PIC S9(9) USAGE COMP.    --> setter名はsetArgcompIとなる
 02 ARGALNUM PIC X(60).       --> setter名はsetArgalnumIとなる
```

上記の入力引数の設定前にこのメソッドを呼び出すと，入力引数の長さを変更できます。

```
（その1）ARGCOMPまでの入力引数とする場合
 setInLenFollowSetData();
 setArgcompI(new Integer(10));
 call(....);                  --> 入力引数の長さは4となる。
（その2）ARGALNUMの先頭８文字までを入力引数とする場合
 setArgcompI(new Integer(4));
 setInLenFollowSetData();
 setArgalnumI(”abcdefgh”);
 call(....);                  --> 入力引数の長さは12となる。
```

このメソッドを呼び出さない場合

入力引数の長さは，入力引数領域長です。

パラメタ：なし

戻り値：なし

例外：なし

(5)　入力引数データの設定

```
[setXxxIメソッド]

public void setXxxI(Object xxxI [ , int dim1 { , int dim2 } … ] ) throws J2CBException
```

メソッド名 setXxxI の Xxx および引数 xxxI 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入力引数にある場合，次に示す setter になります。

```
public void setWk_datanameI(Object wk_datanameI) throws J2CBException
```

- 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 xxxI を指定してください。データ項目の対応は，「表 8-6　各データ項目に対応する表示文字列」をご覧ください。
- [, 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，次に示す setter になります。

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

パラメタ：

xxxI - 設定するデータ

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「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.3 TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)

TP1Access クラスのメソッドについて機能ごとに説明します。

8.3.1 ユーザ認証機能

TP1Access クラスで提供するユーザ認証機能メソッドの一覧を以下に記載します。

表 8-7　TP1Access クラス（ユーザ認証機能）の提供メソッド

項番	メソッド名	機能
1	cltin	クライアントユーザの認証要求
2	cltout	クライアントユーザの認証解除

(1) クライアントユーザの認証要求機能

```
[ cltinメソッド ]

public void cltin(int[] cltid,
                  java.lang.String defpath,
                  java.lang.String targetHost,
                  java.lang.String logname,
                  java.lang.String password,
                  java.lang.String[] setHost,
                  int flags)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

ただし，要素数 1 以上の int 型配列を指定する。

配列[0]に値が設定される。

defpath - クライアント環境を定義した定義ファイルのパス名

targetHost - TP1/Server

logname - ログイン名

password - パスワード

setHost - 認証要求したホスト名

要素数 1 以上の String 配列を指定する。配列[0]に値が設定される。

flags - フラグ(ユーザ認証を抑止する場合は，

TP1Const.DCCLT_NO_AUTHENT を指定する。ユーザ認証を抑止しない場合は，TP1Const.DCNOFLAGS を指定する。)

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

TP1Const.DCCLT_NO_AUTHENT を指定した場合，setHost には値が設定されませんので，設定値の参照は行わないでください。TP1Const.DCCLT_NO_AUTHENT を指定した場合，setHost パラメタには，「null」を指定することをお奨めします。

(2)　クライアントユーザの認証解除

```
[ cltoutメソッド ]

public void cltout(int cltid,
                   int flags)
       throws J2CBException
```

パラメタ：cltid - クライアント ID

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.3.2　常設コネクション

TP1Access クラスで提供する常設コネクションメソッドの一覧を以下に記載します。

表 8-8　　TP1Access クラス（常設コネクション）の提供メソッド

項番	メソッド名	機能
1	connect	常設コネクション確立
2	disconnect	常設コネクション開放
3	getRaphost	常設コネクション確立要求先の取得
4	setConnectInf	端末識別情報の設定
5	setRaphost	常設コネクション確立要求先の指定

(1)　常設コネクション確立

```
[ connectメソッド ]

public void connect(int cltid,
                    int flags)
        throws J2CBException
```

パラメタ：

cltid - クライアント ID

flags - 常設コネクションを確立する通信相手（このシステムでは TP1Const.DCNOFLAGS を指定）

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(2)　常設コネクション開放

```
[ disconnectメソッド ]

public void disconnect(int cltid,
                       int flags)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(3)　常設コネクション確立要求先の取得

```
[ getRaphostメソッド ]

public void getRaphost(int cltid,
                       java.lang.String[] raphost,
                       int flags)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

raphost - 現在設定されている常設コネクション確立要求先のホスト名およびポート番号

要素数が 1 以上の String 配列を指定します。配列[0]に値が設定されます。

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(4)　端末識別情報の設定

```
[ setConnectInfメソッド ]

public synchronized void setConnectInf(int cltid,
                        java.lang.String inf,
                        short infLen,
                        int flags)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

inf - 端末識別情報

infLen - 端末識別情報長

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(5)　常設コネクション確立要求先の指定

```
[ setRaphostメソッド ]

public void setRaphost(int cltid,
                        java.lang.String raphost,
                        int flags)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

raphost - 常設コネクション確立要求先のホスト名およびポート番号

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.3.3　リモートプロシジャコール（RPC）

TP1Access クラスで提供するリモートプロシジャコールメソッドの一覧を以下に記載します。

表 8-9　　TP1Access クラス（リモートプロシジャコール(RPC)）の提供メソッド

項番	メソッド名	機能
1	close	UAP の終了
2	getWatchTime	サービスの応答待ち時間の参照
3	open	UAP の開始
4	setWatchTime	サービスの応答待ち時間の更新

(1)　UAP の終了

```
[ closeメソッド ]

public void close(int cltid,
                  int flags)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(2)　サービスの応答待ち時間の参照

```
[ getWatchTimeメソッド ]

public int getWatchTime(int cltid)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

戻り値：

TP1/Client/P または TP1/Client/W の dc_rpc_get_watch_time_s 関数のリターン値（詳細コードは，マニュアル「分散トランザクション処理機能　OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」をご覧ください）。

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(3)　UAP の開始

```
[ openメソッド ]

public void open(int cltid,
                 int flags)
               throws J2CBException
```

パラメタ：

cltid - クライアント ID

flags - 初期化する環境を指定(

TP1Const.DCNOFLAGS：SPP/MHP を呼び出すための環境

TP1Const.DCCLT_ONEWAY_SND：メッセージを一方送信するための環境

TP1Const.DCCLT_ONEWAY_RCV：メッセージを一方受信するための環境

TP1Const.DCCLT_SNDRCV：メッセージを送受信するための環境)

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(4)　サービスの応答待ち時間の更新

```
[ setWatchTimeメソッド ]

public void setWatchTime(int cltid,
                         int var)
                throws J2CBException
```

パラメタ：

cltid - クライアント ID

var - サービス応答待ち時間

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.3.4　トランザクション制御

TP1Access クラスで提供するトランザクション制御メソッドの一覧を以下に記載します。

表 8–10　　TP1Access（トランザクション制御）クラスの提供メソッド

項番	メソッド名	機能
1	begin	トランザクションの開始
2	chainedCommit	連鎖モードのコミット
3	chainedRollback	連鎖モードのロールバック
4	getTrnid	現在のトランザクションに関する識別子の取得
5	info	現在のトランザクションに関する情報の報告
6	unchainedCommit	非連鎖モードのコミット
7	unchainedRollback	非連鎖モードのロールバック

(1)　トランザクションの開始

```
[ beginメソッド ]

public void begin(int cltid)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(2)　連鎖モードのコミット

```
[ chainedCommitメソッド ]
```

```
public void chainedCommit(int cltid)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(3)　連鎖モードのロールバック

```
[ chainedRollbackメソッド ]

public void chainedRollback(int cltid)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(4)　現在のトランザクションに関する識別子の取得

```
[ getTrnidメソッド ]

public void getTrnid(int cltid,
                     java.lang.String[] trngid,
                     java.lang.String[] trnbid)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

trngid - トランザクショングローバル識別子，要素数 1 以上の String 配列を指定する。配列[0]に値が設定される。

trnbid - トランザクションブランチ識別子要素数 1 以上の String 配列を指定する。配列[0]に値が設定される。

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(5)　現在のトランザクションに関する情報の報告

```
[ infoメソッド ]

public int info(int cltid,
                java.lang.String flags)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

flags - null を指定

戻り値：

TP1/Client/P または TP1/Client/W の dc_trn_info_s 関数のリターン値（詳細コードは，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/W，TP1/Client/P 編」をご覧ください)。

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(6)　非連鎖モードのコミット

```
[ unchainedCommitメソッド ]

public void unchainedCommit(int cltid)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(7)　非連鎖モードのロールバック

```
[ unchainedRollbackメソッド ]

public void unchainedRollback(int cltid)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.3.5　TCP/IP 通信機能

TP1Access クラスで提供する TCP/IP 通信機能メソッドの一覧を以下に記載します。

表 8-11　　TP1Access クラス（TCP/IP 通信機能）の提供メソッド

項番	メソッド名	機能
1	receive	メッセージの受信
2	receive2	メッセージの受信（障害時メッセージ受信）（TCP/IP 通信機能）
3	send	メッセージの送信

(1)　メッセージの受信

```
[ receiveメソッド ]

public void receive(int cltid,
                    java.lang.String[] buff,
                    int recvleng,
                    int timeout,
                    int flags)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

buff - 受信したメッセージ，要素数 1 以上の String 配列を指定する。配列[0]に値が設定される。

recvleng - 受信するメッセージの長さ

timeout - メッセージ受信時の最大待ち時間（秒）

flags -メッセージ受信後，コネクションを解放するかどうか

- TP1Const.DCNOFLAGS：メッセージ受信後，コネクションを解放しません。
- TP1Const.DCCLT_RCV_CLOSE：メッセージ受信後，コネクションを解放します。

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

出力引数を String オブジェクトで受け取る点が，「8.2　TP1/COBOL アクセス用 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)」で提供される receive メソッドと異なります。

(2)　メッセージの受信（障害時メッセージ受信）

```
[ receive2メソッド ]

public void receive2(int cltid,
                     java.lang.String[] buff,
                     int[] recvleng,
                     int timeout,
                     int flags)
       throws J2CBException
```

パラメタ：

cltid - クライアント ID

buff - 受信したメッセージ，要素数 1 以上の String 配列を指定する。

配列[0]に値が設定される。

recvleng - 受信するメッセージの長さ，要素数 1 以上の int 配列を指定する。

Timeout - メッセージ受信時の最大待ち時間（秒）

flags - メッセージ受信後，コネクションを解放するかどうか

- TP1Const.DCNOFLAGS：メッセージ受信後，コネクションを解放しません。
- TP1Const.DCCLT_RCV_CLOSE：メッセージ受信後，コネクションを解放します。

戻り値：

なし。

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

出力引数を String オブジェクトで受け取る点が，「8.2　TP1/COBOL アクセス用 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)」で提供される receive2 メソッドと異なります。

(3)　メッセージの送信

```
[ sendメソッド ]

public void send(int cltid,
                 java.lang.String buff,
                 int sendleng,
                 java.lang.String hostname,
```

```
                int portnum,
                int flags)
        throws J2CBException
```

パラメタ：

cltid - クライアント ID

buff - 送信するメッセージ

sendleng - 送信するメッセージの長さ

hostname - 接続する MHP が存在するノードのホスト名

portnum - MHP のポート番号

flags - メッセージ送信後，コネクションを解放するかどうか

- TP1Const.DCNOFLAGS：メッセージを送信後，コネクションを解放しません。
- TP1Const.DCCLT_SND_CLOSE：メッセージを送信後，コネクションを解放します。

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

入力引数を String オブジェクトで渡す点が，「8.2　TP1/COBOL アクセス用 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)」で提供される send メソッドと異なります。

8.3.6　一方通知受信機能

TP1Access クラスで提供する一方通知受信機能メソッドの一覧を以下に記載します。

表 8-12　TP1Access クラス（TCP/IP 通信機能）の提供メソッド

項番	メソッド名	機能
1	acceptNotification	サーバからクライアントへの一方通知メッセージの受信
2	cancelNotification	一方通知待ち状態のキャンセル
3	chainedAcceptNotification	一方通知受信
4	closeNotification	一方通知連続受信の終了
5	openNotification	一方通知連続受信の開始

(1)　サーバからクライアントへの一方通知メッセージの受信

```
[ acceptNotificationメソッド ]
```

```
public void acceptNotification(java.lang.String defpath,
                               java.lang.String[] inf,
                               int[] infLen,
                               int port,
                               int timeout,
                               java.lang.String[] hostname,
                               java.lang.String[] nodeid,
                               int flags)
       throws J2CBException
```

パラメタ：

defpath - クライアント環境を定義した定義ファイルのパス名

inf - サーバからの通知メッセージ

要素数 1 以上の String 配列を指定する。配列[0]に値が設定される。

InfLen - サーバからの通知メッセージを格納する領域長

要素数 1 以上の int 配列を指定する。配列[0]に値が設定される。

port - クライアントのポート番号

timeout -タイムアウト値（秒）

hostname - 通知したサーバのホスト名

要素数 1 以上の String 配列を指定する。配列[0]に値が設定される。

nodeid - 通知したサーバのノード識別子

要素数 1 以上の String 配列を指定する。配列[0]に値が設定される。

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(2)　一方通知待ち状態のキャンセル

```
[ cancelNotificationメソッド ]

public void cancelNotification(java.lang.String defpath,
                               java.lang.String inf,
                               int infLen,
                               int port,
                               java.lang.String hostname,
                               int flags)
       throws J2CBException
```

パラメタ：

defpath - クライアント環境を定義した定義ファイルのパス名

inf - CUP（Client User Program）に通知するメッセージ

infLen - メッセージ長（inf の長さ）

port - 一方通知受信要求時に指定したポート番号

hostname - 一方通知受信待ち状態の CUP が存在するホスト名

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(3)　一方通知受信

```
[ chainedAcceptNotificationメソッド ]

public void chainedAcceptNotification(int ntfid,
                                      java.lang.String[] inf,
                                      int[] infLen,
                                      int timeout,
                                      java.lang.String[] hostname,
                                      java.lang.String[] nodeid,
                                      int flags)
       throws J2CBException
```

パラメタ：

ntfid - openNotification メソッドで受け取った一方通知受信 ID

inf - サーバからの通知メッセージ

要素数が 1 以上の String 配列を指定します。配列[0]に値が設定されます。

infLen - サーバからの通知メッセージ長

要素数が 1 以上の int 配列を指定します。配列[0]に値が設定されます。

timeout - タイムアウト値（秒）

hostname - 通知したサーバのホスト名

要素数が 1 以上の String 配列を指定します。配列[0]に値が設定されます。

nodeid - 通知したサーバのノード識別子

要素数が 1 以上の String 配列を指定します。配列[0]に値が設定されます。

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(4)　一方通知連続受信の終了

```
[ closeNotificationメソッド ]

public void closeNotification(int ntfid,
                              int flags)
       throws J2CBException
```

パラメタ：

ntfid - openNotification メソッドで受け取った一方通知受信 ID

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(5)　一方通知連続受信の開始

```
[ openNotificationメソッド ]

public void openNotification(int[] ntfid,
                             java.lang.String defpath,
                             int port,
                             int flags)
       throws J2CBException
```

パラメタ：

ntfid - 一方通知受信 ID

要素数が 1 以上の int 配列を指定します。配列[0]に値が設定されます。

defpath - クライアント環境を定義した定義ファイルのパス名

port - クライアントのポート番号

flags - TP1Const.DCNOFLAGS を指定

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.4 TP1/COBOL アクセス用 Bean ユーザインタフェース API（TP1/Client/J）

TP1/COBOL アクセス用 Bean で使用できるメソッドについて説明します。このシステムでは，通常の TP1 版の TP1Access クラスで提供していた基本 Bean ユーザインタフェース（ユーザ認証，コネクション確立など）を提供していませんので，TP1/Client/J が提供する API を直接使用してください。

表 8-13　メソッド一覧

項番	メソッド名	機能
1	call	遠隔サービスの要求
2	getXxxO	出力引数データの取得
3	receive	メッセージの受信
4	send	メッセージの送信
5	setInLenFollowSetData	setter 発行時に入力引数長設定（このメソッドは，TP1/COBOL adapter for Cosminexus Version 2 で生成した TP1/COBOL アクセス用 Bean で有効です。TP1/COBOL adapter for Cosminexus で生成した TP1/COBOL アクセス用 Bean では無効です。）
6	setXxxI	入力引数データの設定

8.4.1 遠隔サービスの要求

```
[ callメソッド ]

public void call(TP1Client cltj,
                 java.lang.String group,
                 java.lang.String service,
                 int flags)
          throws Exception
```

パラメタ：

cltj - TP1/Client/J のインスタンス

group - サービスグループ名（31 文字以内の ASCII 文字で指定します）。

service - サービス名（31 文字以内の ASCII 文字で指定します）。

flags - RPC の形態を指定

- DCNOFLAGS：同期応答型 RPC
- DCRPC_NOREPLY：非応答型 RPC
- DCRPC_CHAINED：連鎖 RPC

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

- TP1/Client/J 連携例外

  例外の詳細については，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

注意事項

出力引数の指定がない場合，flags に DCRPC_NOREPLY 以外を指定してこのメソッドを実行すると，例外が発生します。

8.4.2　メッセージの送受信

(1)　メッセージの受信

```
[ receiveメソッド ]

public void receive(TP1Client cltj,
                    int timeout,
                    int flags)
            throws Exception
```

パラメタ：

cltj - TP1/Client/J のインスタンス

timeout - メッセージ受信時の最大待ち時間（秒）

flags - メッセージ受信後に，コネクションを解放するかどうかを指定

- DCNOFLAGS：解放しない
- DCCLT_RCV_CLOSE：解放する

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

- TP1/Client/J 連携例外

  例外の詳細については，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

注意事項

このメソッドを呼び出す前に TP1/Client/J が提供する receive メソッドを呼び出さないでください。データを正しく受信できないことがあります。

(2)　メッセージの送信

```
[ sendメソッド ]

public void send(TP1Client cltj,
                 java.lang.String hostname,
                 int portnum,
                 int flags)
          throws Exception
```

パラメタ：

cltj - TP1/Client/J のインスタンス

hostname - 接続する MHP が存在するノードのホスト名

portnum - MHP のポート番号

flags - メッセージ送信後に，コネクションを解放するかどうかを指定

- DCNOFLAGS：解放しない
- DCCLT_SND_CLOSE：解放する

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

- TP1/Client/J 連携例外

  例外の詳細については，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

8.4.3　引数の設定と取得

引数の設定と取得については，TP1/Client/P および TP1/Client/W と同じです。「8.2.2　TP1/COBOL アクセス用 Bean ユーザインタフェース API」をご覧ください。

8.5 TP1/COBOL アクセス用 Bean ユーザインタフェース API(Cosminexus TP1 Connector)

TP1/COBOL アクセス用 Bean で使用できるメソッドについて説明します。このシステムでは，通常の TP1 版の TP1Access クラスで提供していた基本 Bean ユーザインタフェース（ユーザ認証，コネクション確立など）を提供していませんので，Cosminexus TP1 Connector が提供する API を直接使用してください。

表 8-14　メソッド一覧

項番	メソッド名	機能
1	call	遠隔サービスの要求，メッセージの送受信
2	getXxxO	出力引数データの取得
3	setInLenFollowSetData	setter 発行時に入力引数長設定（このメソッドは，TP1/COBOL adapter for Cosminexus Version 2 で生成した TP1/COBOL アクセス用 Bean で有効です。TP1/COBOL adapter for Cosminexus で生成した TP1/COBOL アクセス用 Bean では無効です。）
4	setXxxI	入力引数データの設定

8.5.1 遠隔サービスの要求

```
[ callメソッド ]

public void call(ConnectionFactory cxf,
                 Interaction ix,
                 InteractionSpecImpl ixSpec)
          throws Exception
```

機能：

遠隔サービスの要求

パラメタ：

cxf - Cosminexus TP1 Connector 提供の ConnectionFactory インスタンス

ix - Cosminexus TP1 Connector 提供の Interaction インスタンス

ixSpec - Cosminexus TP1 Connector 提供の InteractionSpecImpl インスタンス

SPP と送受信する場合は，通信形態(setFlags)に同期応答型 RPC(DCNOFLAGS)を設定する。

MHP と送受信する場合は，通信形態(setFlags)に同期送受信(TCPIP_SENDRECV)を設定する。

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

- Cosminexus TP1 Connector 連携例外

  例外の詳細については Cosminexus TP1 Connector に添付のマニュアル「uCosminexus TP1 Connector 利用ガイド」をご覧ください。

注意事項

出力引数の指定がない場合，非応答型以外でこのメソッドを実行すると，例外が発生します。

8.5.2　引数の設定と取得

引数の設定と取得については，TP1/Client/P および TP1/Client/W と同じです。「8.2.2　TP1/COBOL アクセス用 Bean ユーザインタフェース API」をご覧ください。

Cosminexus TP1 Connector 使用時は注意事項がありますのでご注意ください。詳細については Cosminexus TP1 Connector に添付のマニュアル「uCosminexus TP1 Connector 利用ガイド」をご覧ください。

8.6　TP1/COBOL SOAP サービスクラス用 Bean ユーザインタフェース API

TP1/COBOL SOAP サービスクラス用 Bean で使用できるメソッドの一覧を以下に記載します。

表 8-15　提供メソッド一覧

項番	メソッド名	機能
1	getBytesData	入力/出力引数の取得
2	getXxxI	入力引数データの取得
3	getXxxO	出力引数データの取得
4	setBytesData	入力/出力引数の設定
5	setGroupItemLenFollowSetData	setter 発行時に引数長設定（本メソッドは，TP1/COBOL adapter for Cosminexus Version 2 で生成した TP1/COBOL SOAP サーバ用 Bean および TP1/COBOL SOAP クライアント用 Bean で有効です。TP1/COBOL アクセス用 Bean では無効です。）
6	setXxxI	入力引数データの設定
7	setXxxO	出力引数データの設定

8.6.1　TP1/COBOL SOAP サービスクラス用 Bean ユーザインタフェース

TP1/COBOL アクセス用 Bean と異なり，TP1/COBOL SOAP サーバ用 Bean および TP1/COBOL SOAP クライアント用 Bean から，TP1/Client および Cosminexus TP1 Connector の呼び出すインタフェースはありません。そのため，バイト配列で入力/出力引数を設定および参照して，データの受け渡しを行います。

SOAP クライアントで設定および参照する入出力引数は，SOAP クライアントから SOAP サーバにバイト配列(byte[])で受け渡します。

SOAP サーバは，COBOL SPP を呼び出す処理を行うだけで，バイト配列で受け渡すデータを操作する必要はありませんが，SOAP サーバ用アクセス Bean を使用すると，SOAP サーバ上で COBOL SPP に受け渡す入出力引数を参照および変更することができます。SOAP サーバ上で COBOL SPP に受け渡す入出力引数を変更しない場合，呼び出す必要はありません。

TP1/COBOL アクセスでの Cosminexus SOAP アプリケーション連携の詳細は「10.　TP1/COBOL SOAP サービスクラス生成機能」をご覧ください。

(1)　Java UAP の処理の流れ

ここでは Java UAP の流れについて説明します。処理の流れの概要については，「10.2　TP1/COBOL SOAP クライアント/サーバ用 Bean の位置づけ」をご覧ください。ここではプログラムコードレベルの流れを紹介します。

(a) SOAP クライアント（サーブレット）の場合

```
   :                                    // :
TP1SPPCBean bean = new TP1SPPCBean();  // TP1/COBOL SOAPクライアント用Beanインスタンス生成
bean.setDataI(xxx);                    // 入力引数データの設定
   :                                    // :
TP1SPP_ServiceLocator uis = null;
TP1SPP_Port ui = null;
   :                                    // :
uis = new TP1SPP_ServiceLocator();     // WSDLからソースを生成で自動生成されたプログラム
   :                                    // :
ui = uis.getTP1server();               // WSDLからソースを生成で自動生成されたメソッド
   :                                    // :
byte[] inDat = bean.getBytesData(bean.inIndex);
                                       //TP1/COBOL SOAPクライアント用Beanから入力引数全体の
バイト配列取得
ByteArrayHolder outData = null;
ui.methodName(inDat, outData);         // SOAPサーバ上のスケルトン呼び出し
setBytesData(bean.outIndex, outData.value);
                                       // TP1/COBOL SOAPクライアント用Beanへ出力引数全体のバ
イト配列設定
String sdata = (Strinf)bean.getSdata0();       // 出力引数データの取得
```

(b) SOAP サーバの場合（入力/出力引数を変更しない場合）[Cosminexus TP1 Connector の例]

```
public byte[] uapMethod(byte[] inDat){// 入力引数を受け取り，出力引数を返す
   :
  javax.resource.cci.RecordFactory rf = cxf.getRecordFactory();
                                        // RecordFactoryを取得
  javax.resource.cci.IndexedRecord input = rf.createIndexedRecord("in_record");
                                       // 入力引数用のインデックスレコード作成
  javax.resource.cci.IndexedRecord output = rf.createIndexedRecord("out_record");
                                       // 出力引数用のインデックスレコード作成
  input.add(inDat);                    // 入力引数の設定
  output.add(new byte[skeltonClass.outMaxSize]);
                                       // 出力引数の設定
  boolean ret = ix.execute(ixSpec, input, output);
                                       // Cosminexus TP1 Connector経由でCOBOL SPPを呼び出す
  byte[] outDat = (byte[])output.get(0); // 出力引数のバイト配列を取得
   :
  return outDat;                       // 出力引数を返す
   :
```

(c) SOAP サーバの場合（入力/出力引数を変更する場合）[Cosminexus TP1 Connector の例]

```
public byte[] uapMethod(byte[] inDat){  // 入力引数を受け取り，出力引数を返す
   :
  TP1SPPSBean bean = new TP1SPPSBean();    // TP1/COBOL SOAPサーバ用Beanインスタンス生成
  bean.setBytesData(bean.inIndex, inDat);  // TP1/COBOL SOAPサーバ用Beanへ入力引数全体のバイ
ト配列設定
  String data1 = (String)bean.getData1I(); // 入力引数データの取得
   :
```

```
  bean.setData1I( “DATA OK” );                // 入力引数データの設定
   :
  byte[] editedInDat = (byte[])bean.getBytesData(bean.inIndex);
                                              //TP1/COBOL SOAPサーバ用Beanから入力引数全体のバ
イト配列取得
   :
  input.add(editedInDat);                     // 入力引数の設定
  output.add(new byte[skeltonClass.outMaxSize]);
                                              // 出力引数の設定
  boolean ret = ix.execute(ixSpec, input, output);
                                              // Cosminexus TP1 Connector経由でCOBOL SPPを呼び
出す
  byte[] outDat = (byte[])output.get(0);// 出力引数のバイト配列を取得
  bean.setBytesData(bean.outIndex, outDat); // TP1/COBOL SOAPサーバ用Beanへ出力引数全体のバ
イト配列設定
  String data2 = (String)bean.getData20(); // 出力引数データの取得
   :
  bean.setData20( “DATA NG” );                // 出力引数データの設定
   :
  byte[] editedOutDat = (byte[])bean.getBytesData(bean.outIndex);
                                             // TP1/COBOL SOAPサーバ用Beanから出力引数全体のバ
イト配列取得
  return editedOutDat;                        // 出力引数を返す
   :
```

(2)　引数の設定と取得

(a)　入力/出力引数の取得

```
[getBytesDataメソッド]

public byte[] getBytesData(int index) throws J2CBException
```

TP1/COBOL SOAP サーバ/クライアント用 Bean から，入力/出力引数を取得する。

パラメータ：

index - 引数種別

- bean 名.inIndex ：入力引数
- bean 名.outIndex：出力引数

戻り値：

入力/出力引数であるバイト配列データ

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(b)　入力引数データの取得

```
[getXxxIメソッド]

public Object getXxxI( [ int dim1 { , int dim2 } … ] ) throws J2CBException
```

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

その他詳細は，「8.2.2(2) 出力引数データの取得」をご覧ください。

パラメータ：

dim1,dim2…：（添字が必要な場合の）各次元の添字

戻り値：

取得できたデータオブジェクト

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(c)　出力引数データの取得

```
[getXxxOメソッド]

public Object getXxxO( [ int dim1 { , int dim2 } … ] ) throws J2CBException
```

詳細は，「8.2.2(2) 出力引数データの取得」をご覧ください。

パラメータ：

dim1,dim2…：（添字が必要な場合の）各次元の添字

戻り値：

取得できたデータオブジェクト

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(d)　setter 発行時に引数長設定

```
[setBytesDataメソッド]

public void setBytesData(int index, byte[] data, int setMode)
                         throws J2CBException
```

TP1/COBOL SOAP サーバ/クライアント用 Bean に入力/出力引数を設定する。

パラメータ：

index - 引数種別

- bean 名.inIndex ：入力引数

- bean 名.outIndex：出力引数

data - 設定する入力/出力引数のバイト配列データ

指定された引数のバイト配列は，TP1/COBOL SOAP サーバ/クライアント用 Bean で同じ長さで確保した作業領域にコピーされます。引数の定義長ではない点に，注意してください。

```
＜出力引数定義例＞
01    GRP1.
 02   ITEM1   PIC X(30).
 02   ITEM2   PIC X(20).
 02   ITEM3   PIC X(10).
＜Java UAP例＞
:         // 50バイトのバイト配列wkOutDataを設定する
bean.setBytesData(bean.inIndex, wkOutData, TP1Const.ASEXCEPT);
          // 定義長の60バイトではなく，引数に指定されたバイト配列長である
          // 50バイトのバイト配列を新たに確保し，指定された引数をコピーする
String wk3 = (String)bean.getItem30();    // 例外となる
```

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(e)　setter 発行時に引数長設定メソッド

```
[setGroupItemLenFollowSetDataメソッド]

public void setGroupItemLenFollowSetData(int index) throws J2CBException
```

引数種別を指定する点を除いて，「8.2.2(4) setter 発行時の入力引数長設定」と同様です。

パラメタ：

index - 引数種別

- inIndex ：入力引数
- outIndex：出力引数

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(f)　入力引数データの設定

```
[setXxxIメソッド]

public void setXxxI(Object xxxI [ , int dim1 { , int dim2 } … ] ) throws J2CBException
```

詳細は，「8.2.2(5)入力引数データの設定」と同様です。

パラメータ：

xxxI - 設定するデータ

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「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(g)　出力引数データの設定

[**setXxxO**メソッド]

```
public Object setXxxO(Object xxxO [ , int dim1 { , int dim2 } … ] ) throws J2CBException
```

出力引数に値を設定する点を除いて，「8.2.2(5) 入力引数データの設定」と同様です。

パラメータ：

xxxO - 設定するデータ

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「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.7 TP1/COBOL SOAP サービスクラス用基本 Bean ユーザインタフェース API(TP1/Client/P，TP1/Client/W)

SOAP クライアントから呼び出された SOAP サーバで，TP1/Client/P および TP1/Client/W を経由して COBOL SPP を呼び出す場合に，TP1/COBOL SOAP サービスクラス用基本 Bean で提供する API を使用します。提供する API は，TP1/COBOL 基本 Bean とほぼ同じです。

表 8-16　提供メソッド一覧

項番	メソッド名	機能
ユーザ認証機能(8.3 TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)をご覧ください)		
1	cltin	クライアントユーザの認証要求
2	cltout	クライアントユーザの認証解除
常設コネクション機能(8.3 TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)をご覧ください)		
3	connect	常設コネクション確立
4	disconnect	常設コネクション開放
5	getRaphost	常設コネクション確立要求先の取得
6	setConnectInf	端末識別情報の設定
7	setRaphost	常設コネクション確立要求先の指定
リモートプロシジャコール(RPC) (call および callTo メソッドを除いて，8.3 TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)をご覧ください)		
8	call	遠隔サービスの要求
9	callTo	通信先を指定した遠隔サービスの要求
10	close	UAP の終了
11	getWatchTime	サービスの応答待ち時間の参照
12	open	UAP の開始
13	setWatchTime	サービスの応答待ち時間の更新
トランザクション制御(8.3 TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)をご覧ください)		
14	begin	トランザクションの開始
15	chainedCommit	連鎖モードのコミット
16	chainedRollback	連鎖モードのロールバック
17	getTrnid	現在のトランザクションに関する識別子の取得
18	info	現在のトランザクションに関する情報の報告

項番	メソッド名	機能
19	unchainedCommit	非連鎖モードのコミット
20	unchainedRollback	非連鎖モードのロールバック
TCP/IP 通信機能		
21	receive	メッセージの受信
22	receive2	メッセージの受信（障害時メッセージ受信）
23	send	メッセージの送信
一方通知受信機能(8.3 TP1/COBOL 基本 Bean ユーザインタフェース API(TP1/Client/P および TP1/Client/W)をご覧ください)		
24	acceptNotification	サーバからクライアントへの一方通知メッセージの受信
25	chainedAcceptNotification	一方通知受信
26	cancelNotification	一方通知待ち状態のキャンセル
27	closeNotification	一方通知受信の終了
28	openNotification	一方通知受信の開始

太字で記述されたメソッド(call/callTo/receive/receive2/send)を除くメソッドは，8.3 と同様です。

8.7.1　遠隔サービスの要求

(1)　遠隔サービスの要求

```
[ callメソッド ]

public void call(int cltid,
                 java.lang.String group,
                 java.lang.String service,
                 byte[] inData,
                 int[] inLen,
                 byte[] outData,
                 int[] outLen,
                 int flags)
          throws J2CBException
```

パラメタ：

cltid - クライアント ID

group - サービスグループ名

service - サービス名

inData - 入力データ

inLen - 入力データ長。inLen[0]に入力データの長さを設定して呼び出す。

outData - 出力データ。あらかじめ確保しておいたバイト配列データを設定する。

outLen - 出力データ長。outLen[0]に出力データの長さを設定して呼び出す。リターン後は，outLen[0]に受け取った出力データ長が設定される。

flags - RPC の形態を指定

：TP1Const.DCNOFLAGS - 同期応答型 RPC

：TP1Const.DCRPC_NOREPLY - 非応答型 RPC

：TP1Const.DCRPC_CHAINED - 連鎖 RPC

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

出力引数の指定がない場合，非応答型 RPC(DCRPC_NOREPLY)以外を指定した call メソッドを実行すると例外が発生します。

(2)　通信先を指定した遠隔サービスの要求

```
[callToメソッド]

public void callTo(int cltid,
                   java.lang.String hostnm,
                   int scdport,
                   java.lang.String group,
                   java.lang.String service,
                   byte[] inData,
                   int[] inLen,
                   byte[] outData,
                   int[] outLen,
                   int flags)
            throws J2CBException
```

パラメタ：

cltid - クライアント ID

hostnm - サービス要求先のホスト名

scdport - サービス要求先のホストに存在するスケジュールサービスのポート番号

group - サービスグループ名

service - サービス名

inData - 入力データ

inLen - 入力データ長。inLen[0]に入力データの長さを設定して呼び出す。

outData - 出力データ。あらかじめ確保しておいたバイト配列データを設定する。

outLen - 出力データ長。outLen[0]に出力データの長さを設定して呼び出す。リターン後は，outLen[0]に受け取った出力データ長が設定される。

flags - RPC の形態を指定

：TP1Const.DCNOFLAGS - 同期応答型 RPC

：TP1Const.DCRPC_NOREPLY - 非応答型 RPC

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

注意事項：

出力引数の指定が無い場合，非応答型 RPC(DCRPC_NOREPLY)以外を指定した callTo メソッドを実行すると例外が発生します。

8.7.2　メッセージの送受信

(1)　メッセージの受信

```
[ receiveメソッド ]

public void receive(int cltid,
                    int timeout,
                    byte[] outData,
                    int[] outLen,
                    int flags)
            throws J2CBException
```

パラメタ：

cltid - クライアント ID

timeout - メッセージ受信時の最大待ち時間（秒）

outData - 出力データ。あらかじめ確保しておいたバイト配列データを設定する。

outLen - 出力データ長。outLen[0]に出力データの長さを設定して呼び出す。リターン後は，outLen[0]に受け取った出力データ長が設定される。

flags - メッセージ受信後に，コネクションを解放するかどうかを指定

：TP1Const.DCNOFLAGS - 解放しない

：TP1Const.DCCLT_RCV_CLOSE - 解放する

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(2)　メッセージの受信（障害時メッセージ受信）

```
[ receive2メソッド ]

public void receive2(int cltid,
                int timeout,
                byte[] outData,
                int[] outLen,
                int flags)
         throws J2CBException
```

パラメタ：

cltid - クライアント ID

timeout - メッセージ受信時の最大待ち時間（秒）

outData - 出力データ。あらかじめ確保しておいたバイト配列データを設定する。

outLen - 出力データ長。outLen[0]に出力データの長さを設定して呼び出す。リターン後は，outLen[0]に受け取った出力データ長が設定される。

flags - メッセージ受信後に，コネクションを解放するかどうかを指定

：TP1Const.DCNOFLAGS - 解放しない

：TP1Const.DCCLT_RCV_CLOSE - 解放する

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

(3)　メッセージの送信

```
[ sendメソッド ]

public void send(int cltid,
                java.lang.String hostname,
                int portnum,
                byte[] inData,
                int[] inLen,
                int flags)
         throws J2CBException
```

パラメタ：

cltid - クライアント ID

hostname - 接続する MHP が存在するノードのホスト名

portnum - MHP のポート番号

inData - 入力データ

inLen - 入力データ長。inLen[0]に入力データの長さを設定して呼び出す。

flags - メッセージ送信後に，コネクションを解放するかどうかを指定

：TP1Const.DCNOFLAGS - 解放しない

：TP1Const.DCCLT_SND_CLOSE - 解放する

戻り値：

なし

例外：J2CBException - 例外情報の取得

詳細は「8.8　J2CBException ユーザインタフェース API」をご覧ください。

8.8　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 について説明します。

8.8.1　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以下に記載します。

表 8-17　　J2CBException クラスの提供メソッド一覧

項番	メソッド名	機能
1	getCBLErrorCode	例外コードの取得
2	getErrorCode	例外情報コードの取得
3	getMessage	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
4	getName	発生した例外名を取得

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，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，例外情報を参照することができます。

(1)　例外コードの取得

```
[  getCblErrorCode メソッド  ]

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

戻り値：TP1/Client の各関数のエラーコード，例外コードまたはコード変換ライブラリ関数の戻り値。
異常終了してない場合は，null

(2)　TP1/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()
```

補足：

例外情報コードとは，エラー発生場所（メソッド）コード（3 けた）とエラー要因コード（4 けた）で構成されます。コードの詳細については，「付録 G　例外情報コード一覧」をご覧ください。また，これらのコードは 16 進コードです。

戻り値：例外情報コード

TP1/COBOL アクセスで発生した例外情報コードを取得します。

例外情報コードとは，先頭に"J2CB"が付加された１１桁のコード値です。

例外情報コードの詳細は，「付録 G 表 G-2」をご覧ください。

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）

(3)　TP1/COBOL アクセスのメッセージ文字列を取得

```
[ getMessage メソッド　]

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

戻り値：メッセージ文字列

TP1/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。

エラー要因と対策方法の詳細は，「付録 G　例外情報コード一覧」をご覧ください。

オーバーライド：

クラス java.lang.Throwable 内の getMessage

(4)　例外名を取得

```
[ getName メソッド　]

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

戻り値：例外名を返す。

詳細は表 8-18 をご覧ください。

8.8.2　提供クラスのメソッドで発生する例外

表 8-18　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	当該クラス，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

例外名	意味	エラー要因コードの候補
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	TP1/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 5000 番台
UNKNOWN	予期しないエラーが発生しました。	9999

9 Windows/UNIX 混在システム構築上の注意事項

この章では，TP1/COBOL アクセスを使って Windows/UNIX 混在の Web アプリケーションシステムを構築する際の注意事項や解決方法について説明します。

9.1　概要

TP1/COBOL アクセスを使って，Windows/UNIX 混在の Web アプリケーションをシステム構築する際，OS の仕様の違いによる考慮が必要です。

考慮すべき点の概要を以下に示します。

9.1.1　外字・特殊文字データの扱い

Windows/UNIX で文字コード体系が一部異なるため，外字・特殊文字のデータを異なる OS で構成される Cosminexus と OpenTP1 間で流通させる場合，エンコードの変換を考慮しなければなりません。

このエンコードの考慮は，Web ブラウザが Windows であれば，Cosminexus，OpenTP1 が共に UNIX であっても該当します。詳細は「9.2　OS 混在構築時の注意点（文字コード体系の違い）」をご覧ください。

9.1.2　２進項目および内部浮動小数点項目のデータの扱い

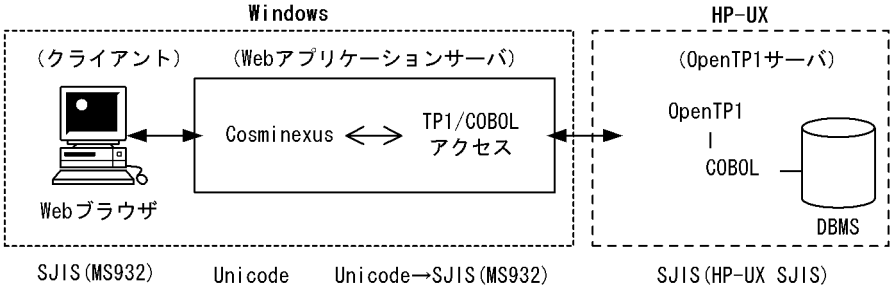
Windows と UNIX では，メモリ上に格納されるデータ形式が異なります。そのため，２進項目および内部浮動小数点項目のデータを Windows 版 Cosminexus と UNIX 版 OpenTP1 間で流通させる場合，データ格納形式変換の考慮が必要となります。詳細は「9.3　Windows/UNIX の数字データ格納形式の相違」をご覧ください。

9.2 OS 混在構築時の注意点（文字コード体系の違い）

9.2.1 構築例 1

以下に，フロントエンドに Windows 版 Cosminexus，バックエンドに HP-UX 版 OpenTP1 という形態で，Windows 環境と UNIX 環境が混在した Web システム構築例を示します。

図 9-1　異なる OS が混在したシステム構築例-1



各 OS での文字コード体系が一部異なるため，特殊文字や外字データを扱うときには注意が必要です。

図 9-1 では，Windows Cosminexus，HP-UX OpenTP1 という形態において，TP1/COBOL アクセスを使って通信を行っています。TP1/COBOL アクセスを使用することで，OpenTP1 での業務結果等を Web ブラウザに表示することが可能となります。この構築例の場合，TP1/COBOL アクセスで Unicode ⇔ SJIS(MS932)変換を行っている点に着目してください。

この場合の各製品の文字コード体系を表 9-1 に示します。

表 9-1　構築例 1 での文字コード体系の相違点

シ ス テ ム	文字コード体系
Web ブラウザ(Internet Explorer)	Windows 日本語版と呼ばれる SJIS(MS932)です。特殊文字として，NEC 特殊文字，NEC 選定 IBM 文字，IBM 拡張文字が割り当てられています。
Cosminexus サーバ	Cosminexus から提供される Java が SJIS(MS932)←→Unicode 変換を行います。Java プログラムではすべて，Unicode となります。
TP1/COBOL アクセス	Unicode←→SJIS(MS932)変換を行います。
OpenTP1 サーバ	SJIS(HP-UX SJIS)です。NEC 特殊文字，NEC 選定 IBM 文字，IBM 拡張文字が割り当てられていません（SJIS(MS932)変換を行うと NEC 特殊文字，NEC 選定 IBM 文字，IBM 拡張文字のコード値は HP-UX SJIS では外字コードエリアに属します）。

9.2.2　構築例 1 の注意点

構築例では，Windows 上の TP1/COBOL アクセスは SJIS(MS932)にデータを変換し，HP-UX 上の OpenTP1 では SJIS(HP-UX SJIS)でデータを操作しています。そのため，SJIS(HP-UX SJIS)の文字コード体系で割り当てられていない NEC 特殊文字などは，HP-UX 上の OpenTP1 環境下で動作する COBOL プログラムでは SJIS(MS932)の文字コード体系で扱う必要があります。

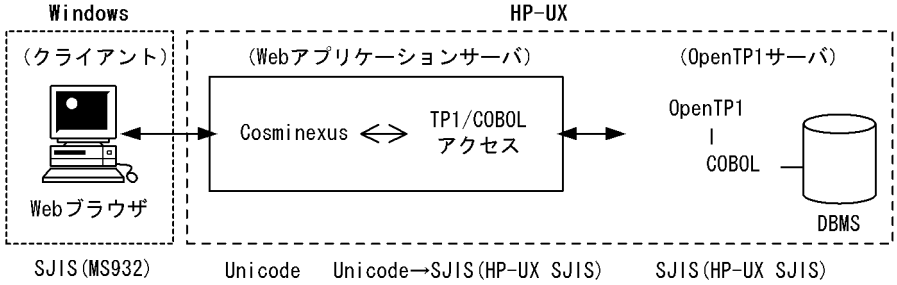
また，TP1/COBOL アクセスでは，変換する文字コード体系を指定する encode オプション(Windows 版では MS932 がデフォルトです)があり，変換エンコードとして「SJIS」を指定することもできます。ただし，「SJIS」を指定した場合，SJIS(MS932)の文字コード体系で使用できる特殊文字(NEC 特殊文字など)が，SJIS(HP-UX SJIS)の文字コード体系に対応する文字がないため変換できません。Web ブラウザで「SJIS(MS932)の文字コード体系で使用できる特殊文字(NEC 特殊文字など)を使用しない」のであれば，encode オプションに「SJIS」を指定することができます。

encode オプションの詳しい説明，および設定方法は，「5.3　TP1/COBOL アクセス環境変数の設定」をご覧ください。

9.2.3　構築例 2

以下に，フロントエンドに HP-UX 版 Cosminexus，バックエンドに HP-UX 版 OpenTP1 という形態で，Web ブラウザだけが Windows 環境の Web システム構築例を示します。

図 9-2　　異なる OS が混在したシステム構築例 2



この構築例の場合，TP1/COBOL アクセスで Unicode ⇔ SJIS(HP-UX SJIS)変換を行っている点に着目してください。

9.2.4　構築例 2 の注意点

図 9-2 の例では Web ブラウザが Windows であることから特殊文字および外字文字を含む場合，SJIS(MS932)でデータを流通させなければなりません。SJIS(MS932)でデータを流通させるためには，TP1/COBOL アクセスが用意している環境変数に指定する encode オプションに「MS932」を指定してください。Web ブラウザで「SJIS(MS932)の文字コード体系で使用できる特殊文字(NEC 特殊文字など)を使用しない」のであれば，encode オプションに「SJIS」を指定(HP-UX 版では SJIS がデフォルトです)することができます。

encode オプションの詳しい説明，および設定方法は，「5.3　TP1/COBOL アクセス環境変数の設定」をご覧ください。

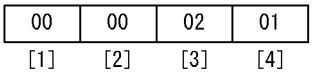
9.3 Windows/UNIX の数字データ格納形式の相違

UNIX 形式と Windows 形式では，バイナリ形式のデータ（2 進項目および内部浮動小数点項目）がメモリ上に配置されるときの順序が次の例のように逆になります。

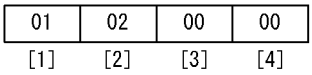
（例）

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(UNIX)



- リトルエンディアン形式(Windows)



このため，Cosminexus が Windows，OpenTP1 が UNIX の場合，バイナリ形式のデータは，リトルエンディアン形式のデータ値で COBOL SPP/MHP に渡ってしまいます。この場合，Windows 側で，endian(big)を指定することで，データの配置が反転され，ビッグエンディアン形式のデータ値で，COBOL SPP/MHP に渡すことができます。

また Cosminexus が UNIX，OpenTP1 が Windows の場合，Windows 側で，COBOL SPP/MHP コンパイル時に-BigEndian,Bin コンパイラオプション※1，-BigEndian,Float コンパイラオプション※2 を指定することで，COBOL SPP/MHP は，ビッグエンディアン形式でデータを受け取ることができます。

endian オプションの詳しい説明および設定方法は，「5.3　TP1/COBOL アクセス環境変数の設定」の設定をご覧ください。

-BigEndian,Bin コンパイラオプション※1，-BigEndian,Float コンパイラオプション※2 については，マニュアル「COBOL2002 ユーザーズガイド」をご覧ください。

注※ 1

COBOL85 をご使用の場合は，-Bb コンパイラオプションを指定してください。

注※ 2

COBOL85 をご使用の場合は，-Fb コンパイラオプションを指定してください。

10 TP1/COBOL SOAP サービスクラス生成機能

この章では，TP1/COBOL アクセスを使って OpenTP1 サーバ環境下の COBOL SPP を Cosminexus SOAP アプリケーションとして呼び出す方法について説明しています。

この機能は，Cosminexus SOAP アプリケーション開発方法を理解している方を前提とします。

なお，TP1/COBOL SOAP サービスクラス生成機能をご使用になる場合は，Cosminexus Version5 05-05(Windows 版だけ)，Cosminexus Version 6 06-00 以降(Windows/AIX 版だけ)または uCosminexus Developer 06-70/ uCosminexus Application Server 06-70(Windows 版だけ)をお使いください。

uCosminexus Developer 07-00 では TP1/COBOL SOAP サービスクラス生成機能を使って，新規に Cosminexus SOAP アプリケーションを開発することはできません。Cosminexus Version5 05-05，Cosminexus Version 6 06-00 以降または uCosminexus Developer 06-70 で開発した Cosminexus SOAP アプリケーションを uCosminexus Application Server07-00 でのみ実行することができます。

また，本製品の Windows 版 02-10 以降，Linux(IPF)版，および HP-UX(IPF)版では本機能は使用できませんので，ご注意ください。

10.1　概要

TP1/COBOL アクセスを経由して，Cosminexus SOAP アプリケーション※として OpenTP1 サーバ環境下の COBOL SPP を呼び出せるように，スケルトンクラス，TP1/COBOL SOAP クライアント用 Bean および TP1/COBOL SOAP サーバ用 Bean を自動生成します。

ウィザードにより自動生成された Bean を配置することで，SOAP アプリケーションの連携を可能にします。

注※

uCosminexus 06-70 の SOAP アプリケーション開発では，標準モードと互換モードの 2 つのモードが提供されますが，本製品では互換モードだけを対象とします。

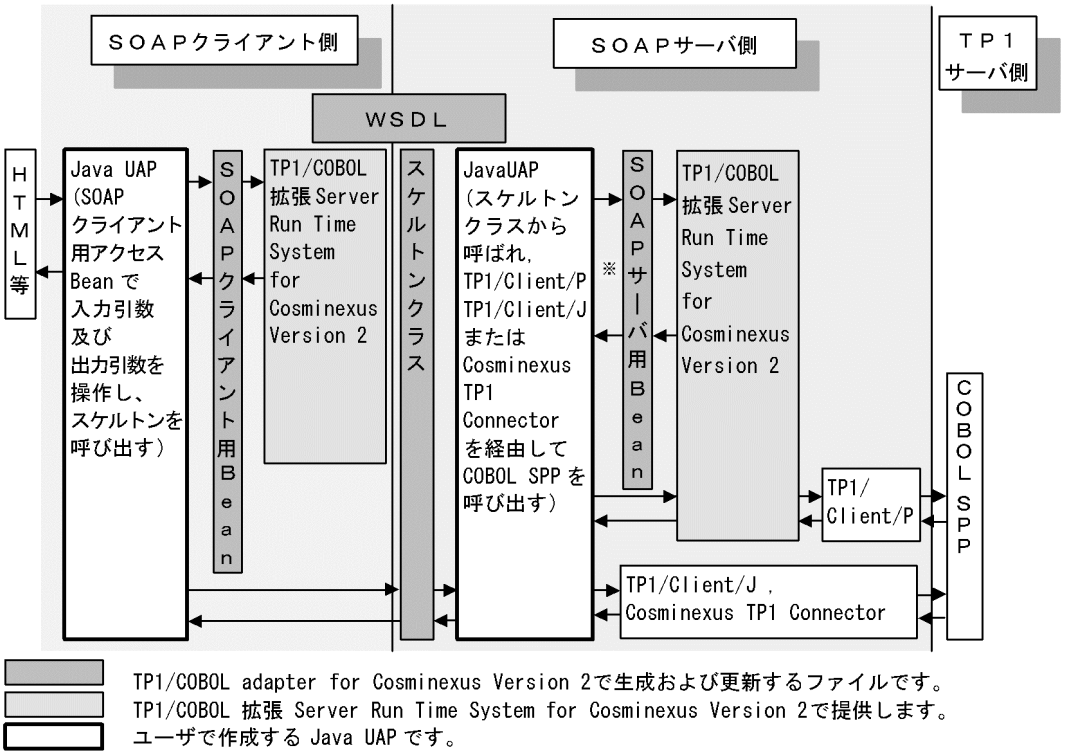
10.2　TP1/COBOL SOAP クライアント/サーバ用 Bean の位置づけ

SOAP クライアントの JavaUAP では，COBOL SPP の入力引数を構築し，SOAP サーバのスケルトンを経由して COBOL SPP を呼び出し，返された出力引数からデータの取得を行います。入力引数の構築および出力引数からデータを取得するために，TP1/COBOL SOAP クライアント用 Bean を呼び出します。

SOAP サーバの JavaUAP では，COBOL SPP を呼び出すために，TP1/Client および Cosminexus TP1 Connector との接続やユーザ認証などを行い，スケルトンから受け取った入力引数をそのまま COBOL SPP に渡します。COBOL SPP から返された出力引数も，そのままスケルトンに返します。入力引数の構築および出力引数のデータ取得は，SOAP クライアントの JavaUAP で行いますが，SOAP サーバで編集することも可能です。SOAP サーバで入力引数および出力引数を編集する場合は，TP1/COBOL SOAP サーバ用 Bean を呼び出します。

10.2.1　処理の流れ

図 10-1　SOAP サービス使用時の処理の流れ



注※

SOAP クライアントで設定および参照する入出力引数は，SOAP クライアントから SOAP サーバにバイト配列(byte[])で受け渡します。

SOAP サーバは，COBOL SPP を呼び出す処理を行うだけで，バイト配列で受け渡すデータを操作する必要はありませんが，SOAP サーバ用アクセス Bean を使用すると，SOAP サーバ上で COBOL SPP に受け渡す入出力引数を参照および変更することができます。SOAP サーバ上で COBOL SPP に受け渡す入出力引数を変更しない場合，呼び出す必要はありません。それぞれの引数のインターフェースに

ついては「8.6 TP1/COBOL SOAP サービスクラス用 Bean ユーザインタフェース API」をご覧ください。

10.3 事前準備

TP1/COBOL SOAP サービスクラス生成機能を使用する前に，次の準備が必要です。

1. Cosminexus をインストールする際に［SOAP アプリケーション開発支援機能］を組み込む。
2. ［SOAP アプリケーション開発支援機能］は使用できる状態にしておく。
3. JBuilder に TP1/COBOL アクセス開発環境を組み込む（組み込み方法は付録 C をご覧ください）。
4. JBuiler のデフォルトプロパティで JDK を Cosminexus の JDK に変更する。
5. JBuiler のデフォルトプロパティの必須ライブラリに次の項目を追加する。※
   - J2tp ライブラリ
   - hitj2ee.jar
   - TP1Client.jar
   - tp1connector.jar（Non-Managed 環境）
   - tp1concommon.jar（Cosminexus TP1 Connector 02-00-/A 以降）
   - Cosminexus SOAP ライブラリ

   注※

   Cosminexus Version 5 05-05 では，csmjaxp.jar（Cosminexus XML Processor）の追加登録が必要です。

なお，このマニュアルでは，Cosminexus SOAP アプリケーション開発方法を理解している方を前提としており，SOAP アプリケーションの開発方法については記載しません。

詳細はマニュアル「Cosminexus SOAP アプリケーション開発ガイド」をご覧ください。

10.4　開発手順

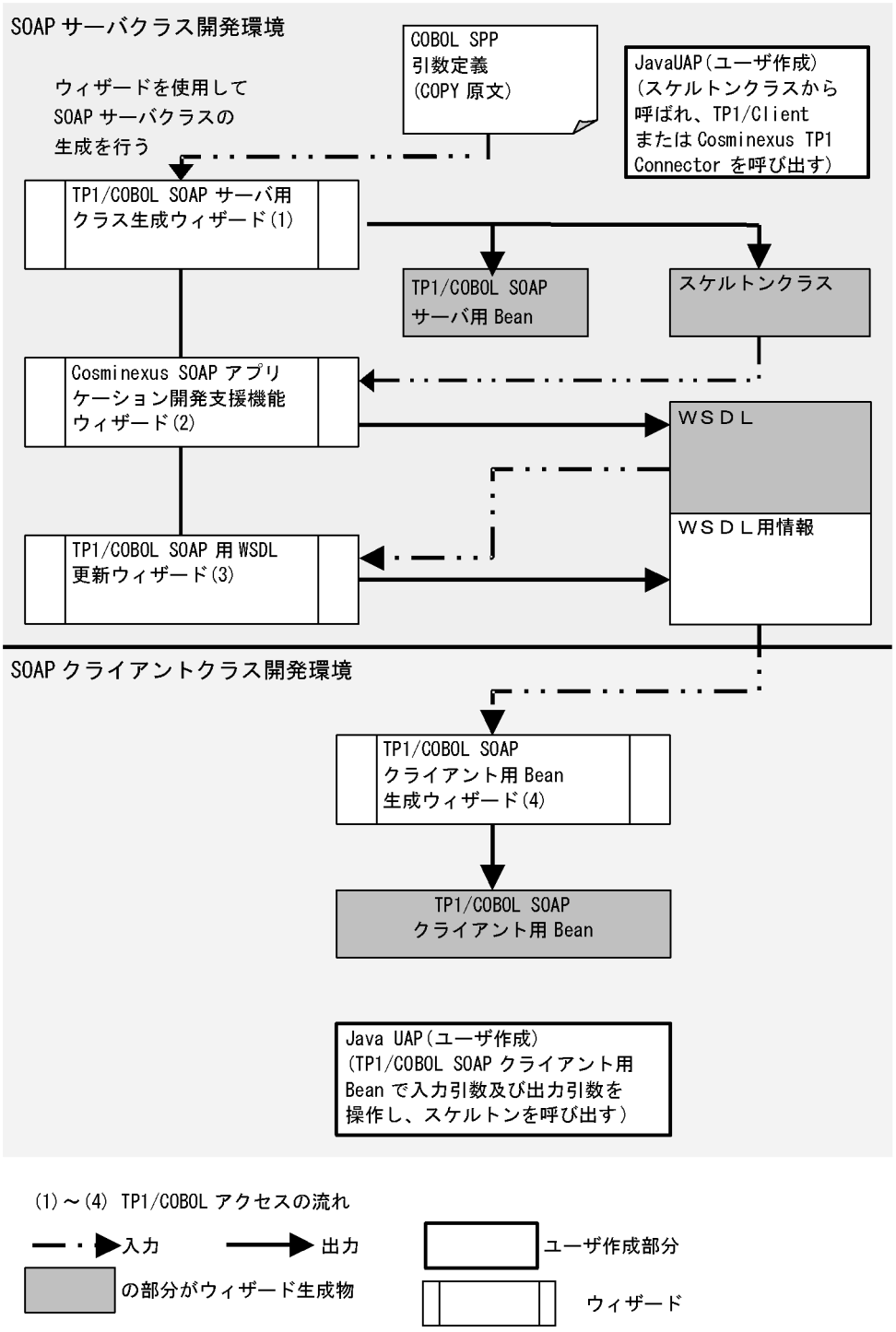
SOAP サーバアプリケーションおよび SOAP クライアントアプリケーションの開発手順は，マニュアル「Cosminexus SOAP アプリケーション開発ガイド」の「4. RPC を利用した SOAP アプリケーションの開発例」「4.2 既存の Java クラスを利用して SOAP アプリケーションを開発する場合」に従います。その手順の中でサーバ用サービスクラスの作成は，「TP1/COBOL SOAP サービスクラス生成機能」で提供する［TP1/COBOL SOAP 用サービスクラス生成ウィザード］を使用して自動生成することができます。

また，クライアントは，「TP1/COBOL SOAP サービスクラス生成機能」で提供する［TP1/COBOL SOAP クライアント用 Bean 生成ウィザード］を使用して自動生成することができます。

これ以降で説明される「TP1/COBOL SOAP サービスクラス生成機能」以外の詳細設定は，マニュアル「Cosminexus SOAP アプリケーション開発者ガイド」をご覧ください。

この機能を使用した開発の流れは次のようになります。

図 10-2　TP1/COBOL SOAP アプリケーション開発手順



10.4.1　SOAP サーバアプリケーション側

(a) JBuilder で新規プロジェクトを作成して，[Web アプリケーション]で必須項目を設定する。

(b) JavaUAP(スケルトンクラスから呼ばれ，TP1/Client または Cosminexus TP1 Connector を呼び出す)を作成する。

(c) 「TP1/COBOL SOAP サーバ用クラス生成ウィザード」に準備しておいた COBOL SPP の引数定義(COPY 原文)を指定して，スケルトンクラスおよび TP1/CCOBOL SOAP サーバ用 Bean を生成する。
(図 10-2 (1))

(d) プロジェクトをメイクして，クラスファイルを作成する。

(e) [SOAP サービスデプロイ定義の生成]ウィザードで，デプロイ定義を生成する。

(f) プロジェクトをメイクして，war ファイルを作成する。

(g) [クラスから WSDL の生成]ウィザードで，WSDL ファイルを生成する。
(図 10-2 (2))

(h) 「TP1/COBOL SOAP 用 WSDL 更新ウィザード」で，WSDL ファイルを更新する。
(図 10-2 (3))

10.4.2 SOAP クライアントアプリケーション側

(a) JBuilder で新規プロジェクトを作成する。

(b) [WSDL からソースの生成]に「10.4.1 (f)」で生成した WSDL を指定して，SOAP クライアントで必要なソースを生成する。

(c) 「TP1/COBOL SOAP クライアント用 Bean 生成ウィザード」に「10.4.1 (f)」で生成した WSDL を指定して，TP1/COBOL SOAP クライアント用 Bean を生成する。
(図 10-2 (4))

(d) JBuilder で[サーブレット]を生成し，実装部分を作成する。

(e) プロパティファイルや html ファイルを配置する。

(f) プロジェクトをメイクして，war ファイルを作成する。

[]で囲んだ部分は，マニュアル「Cosminexus SOAP アプリケーション開発ガイド」で説明された各ウィザードの名称です。

10.5　SOAP サーバアプリケーションの作成

SOAP サーバアプリケーションの作成方法について説明します。

SOAP サーバアプリケーションは，SOAP エンジンから呼び出されるスケルトンクラスとスケルトンクラスから呼び出されるユーザ作成の Java UAP から構成されます。これに加えて SOAP サーバ側で引数データの内容を参照・設定する必要がある場合は，TP1/COBOL SOAP サーバ用 Bean を使用します。スケルトンクラスおよび TP1/COBOL SOAP サーバ用 Bean は，「TP1/COBOL SOAP サーバ用クラス生成ウィザード」によって生成します。

10.5.1　「TP1/COBOL SOAP サーバ用クラス生成ウィザード」による生成

JBuilder 上で TP1/COBOL SOAP サーバ用クラス生成ウィザードを使用することによって，スケルトンクラスおよび TP1/COBOL SOAP サーバ用 Bean を自動生成できます。

なお，このウィザードを使用するには，JBuilder で［Web アプリケーション］を作成しておく必要があります。

(1)　起動方法

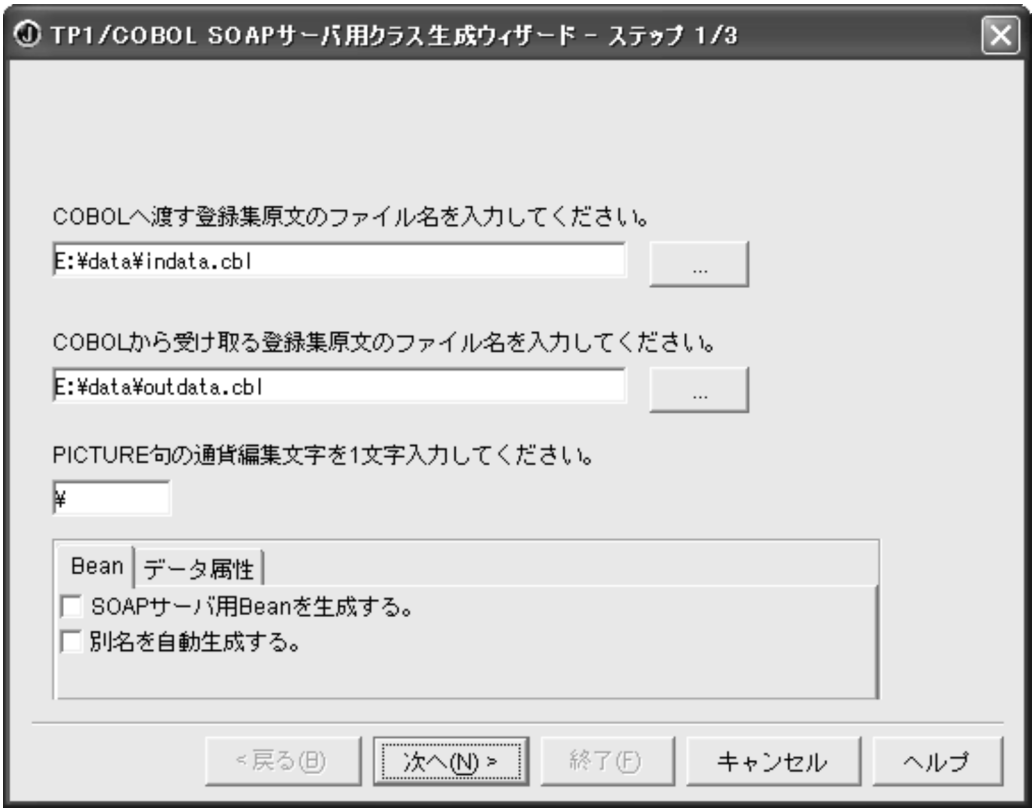
JBuilder のオブジェクトギャラリ内の[一般]タブ内にある「TP1/COBOL SOAP サーバ用クラス」アイコンをダブルクリックします。

または，メニューバーの[ウィザード]（(注）uCosminexus Developer から提供される JBuilder 2005 では［編集］メニューの［ウィザード］となります）から「TP1/COBOL SOAP サーバ用クラス」を選択します。

(2)　画面の説明

(a)　ステップ 1/3 画面

「TP1/COBOL SOAP サーバ用クラス生成ウィザード」を起動すると，次のような「TP1/COBOL SOAP サーバ用クラス生成ウィザード - ステップ 1/3」画面が現れます。



ここでは，次の情報を入力します。

- [SOAP サーバ用 Bean を生成する。]のチェックボックスをオンした場合，SOAP サーバ側で引数データの内容を参照・設定するための Bean を生成します。SOAP サーバ側で引数データの内容を操作する必要がない場合，チェックボックスをオフにしてください。

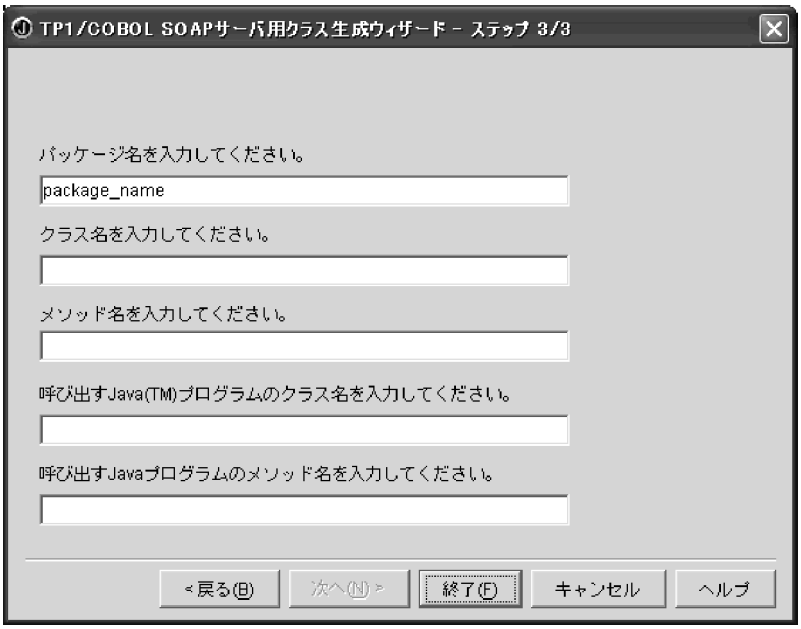
[SOAP サーバ用 Bean を生成する。]のチェックボックス以外は「TP1/COBOL アクセス用 Bean 生成ウィザード - ステップ 1/3」と同じです。詳細は「2.3.1 「TP1/COBOL アクセス用 Bean 生成ウィザード」による生成」をご覧ください。設定が完了したら，[次へ>]ボタンを押して「TP1/COBOL SOAP サーバ用クラス生成ウィザード-ステップ 2/3」画面に進みます。

(b)　ステップ 2/3 画面

ここで入力する情報は，「TP1/COBOL アクセス用 Bean 生成ウィザード - ステップ 2/3」と同じです。詳細は「2.3.2　入出力用の「Bean 生成ウィザード」に表示されるパラメタテーブルの編集」をご覧ください。TP1/COBOL SOAP サーバ用 Bean で設定および参照したいデータ項目の「選択」チェックボックスをオンにします。

設定が完了したら，[次へ>]ボタンを押すと「TP1/COBOL SOAP サーバ用クラス生成ウィザード-ステップ 3/3」画面に進みます。

(c)　ステップ 3/3 画面



ここでは，次の情報を入力します。指定は省略できません。すべての項目に対して入力が必要です。

- パッケージ名

JBuilder の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（ホスト名など）を入力します。（例：localhost）

- クラス名
  スケルトンクラスの名称を入力します。名称は任意です。（例：CBLTP1）
- メソッド名
  スケルトンクラスのメソッド名を指定します。名称は任意です。（例：TP1_method）
- 呼び出す Java プログラムのクラス名
  スケルトンクラスから呼ばれるユーザ作成の Java UAP のクラス名を指定します。（例：CBLTP1UAP）
- 呼び出す Java プログラムのメソッド名
  スケルトンクラスから呼ばれるユーザ作成の Java UAP のメソッド名を指定します。（例：CBLTP1UAPmethod）

最後に[終了(F)]ボタンを押すと，JBuilder のプロジェクトにスケルトンクラスと TP1/COBOL SOAP サーバ用 Bean となる Java ソースが自動生成されます。自動生成される Java ソースファイル名は以下のようになります。

- スケルトンクラス：クラス名.java（例：CBLTP1.java）
- TP1/COBOL SOAP サーバ用 Bean：クラス名 SBean.java（例：CBLTP1SBean.java）

生成例は付録 E.4 に記載しています。

(d)　呼び出す Java プログラムについて

自動生成を行った後，呼び出す Java プログラム（JavaUAP）を作成してください。OpenTP1 と接続するために使用できる API については「8.7　TP1/COBOL SOAP サービスクラス用基本 Bean ユーザインタフェース API(TP1/Client/P，TP1/Client/W)」，マニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引き TP1/Client/J 編」または Cosminexus TP1 Connector に添付のマニュアル「uCosminexus TP1 Connector 利用ガイド」をご覧ください。この他，作成例を付録 H.6 に記載しています。

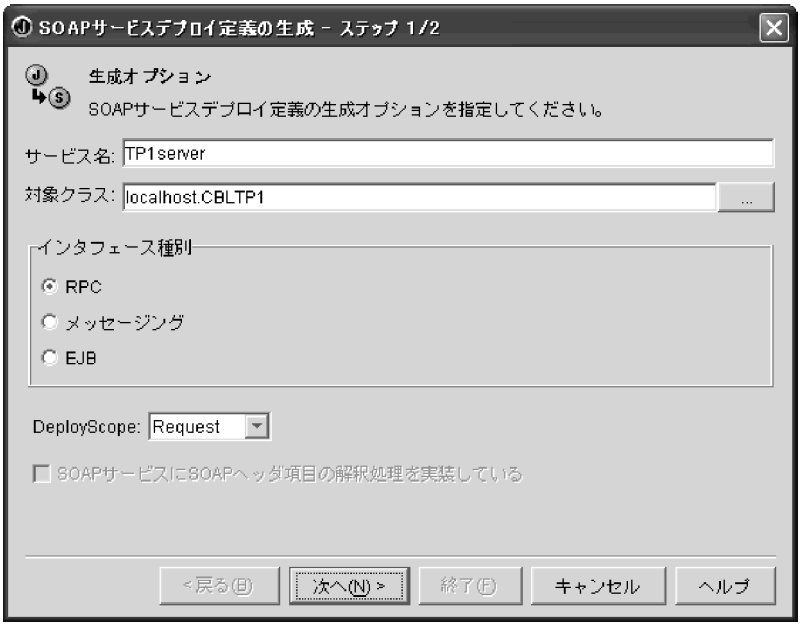
10.5.2　Cosminexus SOAP アプリケーション開発支援での設定

「10.3　事前準備」ができていることが前提です。ここでは JBuilder に組み込まれた［SOAP サービスデプロイ定義の生成］と［クラスから WSDL の生成］を使用します。

(1)　［SOAP サービスデプロイ定義の生成］

JBuilder のオブジェクトギャラリで Cosminexus 製品群の入っているタグを選択し，［SOAP サービスデプロイ定義の生成］を起動します。ステップ 1/2 画面で次の情報を設定してください。

(a) ステップ 1/2 画面



ここでは次の情報を設定してください。

- サービス名

  サービス名は任意です。ここで付けた名称は（2）［クラスから WSDL の生成］のロケーションで使用します。(例：TP1server)

- 対象クラス

  ここは必ず，スケルトンクラス(パッケージ名.クラス名)を指定します。

(例：localhost.CBLTP1)

(b) ステップ 2/2 画面

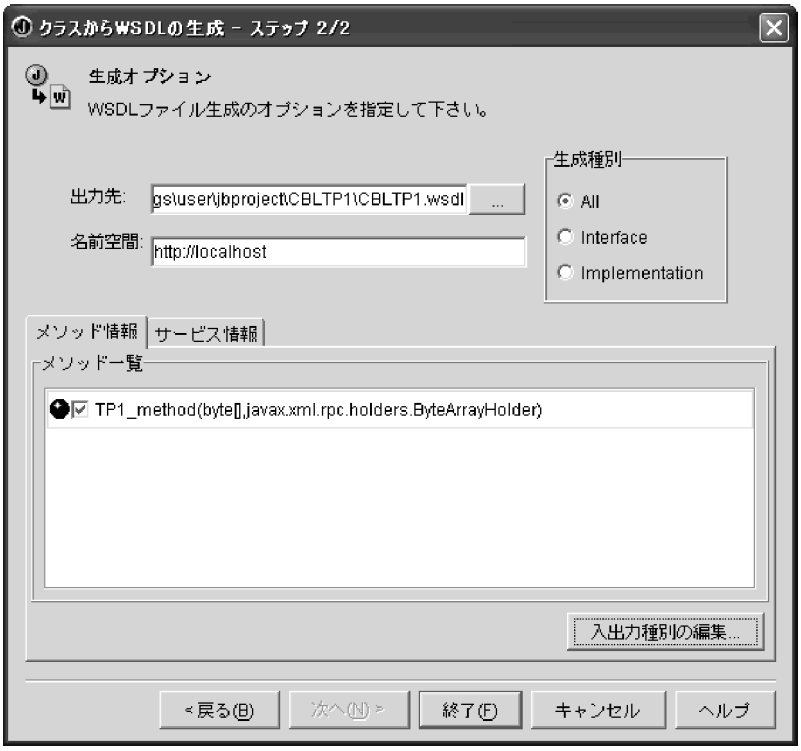
- 「入出力種別の編集...」

  公開メソッド一覧からメソッドを選択して「入出力種別の編集...」を開いてください。引数情報の入出力種別で INOUT になっているところを OUT に変更してください。

あとはウィザードの指示に従い SOAP サービスデプロイ定義の生成を行います。

(2) ［クラスから WSDL の生成］

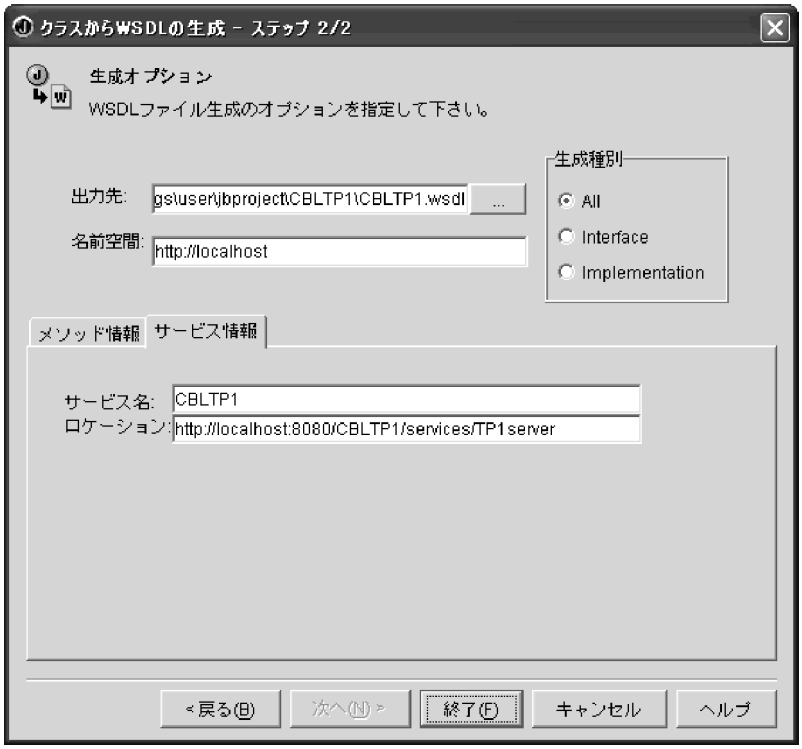
JBuilder のオブジェクトギャラリで Cosminexus 製品群の入っているタグを選択し，［クラスから WSDL の生成］を起動します。ステップ 2/2 画面で次の情報を設定してください。



ここでは次の情報を設定します。

- 「入出力種別の編集...」

  メソッド情報タブでメソッドを選択して「入出力種別の編集...」を開いてください。引数情報の入出力種別で INOUT になっているところを OUT に変更してください。



- サービス名

  デフォルトで JBuilder のプロジェクト名が入っていますが，変更することもできます。

- ロケーション
  最後尾の名前には（1）のサービス名でつけた名前を指定してください。

ここはクライアントからサーバを検索するときの大事なアドレスです。WSDL で正しい情報が作成されていないと，クライアントとサーバはつながりませんので，ご注意ください。

あとはウィザードの指示に従い WSDL ファイルの生成を行います。

ここで生成された WSDL ファイルは「TP1/COBOL SOAP 用 WSDL 更新ウィザード」で使用します。

(3) war ファイルの作成

JBuilder のメイクで war を作成します。

war ファイルの作成は次で説明する WSDL 更新を行ったあとで作成しても問題ありません。

10.5.3 WSDL ファイルの更新

生成された WSDL ファイルを「TP1/COBOL SOAP 用 WSDL 更新ウィザード」を使用して更新します。

(1) 起動方法

JBuilder のオブジェクトギャラリ内の[一般]タブ内にある「TP1/COBOL SOAP 用 WSDL 更新」アイコンをダブルクリックします。

または，メニューバーの[ウィザード]（(注）uCosminexus Developer から提供される JBuilder 2005 では［編集］メニューの［ウィザード］となります）から「TP1/COBOL SOAP 用 WSDL 更新」を選択します。

(2) 画面の説明

「TP1/COBOL SOAP 用 WSDL 更新ウィザード」を起動すると，次のような画面が表示されます。

ここで，10.5.2(2)で生成した WSDL ファイルを指定して「OK」ボタンをクリックすると，WSDL ファイルが更新されます。

10.6　SOAP クライアントアプリケーションの作成

ここでは 10.5.3 で更新した WSDL ファイルを使用して，クライアント側で必要なファイルを生成します。

10.6.1　クライアント側の実装に必要なファイルの生成

クライアント側の実装に必要なファイルをサーバ側で生成した更新済み WSDL から生成します。手順は次のとおりです。

1. クライアント側は JBuilder で新たにプロジェクトを作成する。
2. JBuilder のオブジェクトギャラリで Cosminexus 製品群の入っているタグを選択し，［WSDL からソースの生成］を使用する。
3. WSDL からソースの生成－ステップ 1/2 画面に 10.5.3 で更新した WSDL ファイルを指定する。

詳細手順は，マニュアル「Cosminexus SOAP アプリケーション開発ガイド」をご覧ください。

10.6.2　「TP1/COBOL SOAP クライアント用 Bean 生成ウィザード」による生成

10.6.1 で作成したプロジェクト上で，更新済み WSDL を使用してクライアント用のアクセス Bean を生成します。

JBuilder に組み込まれた TP1/COBOL SOAP クライアント用 Bean 生成ウィザードを使用することによって，TP1/COBOL SOAP クライアント用 Bean を自動生成できます。

(1)　起動方法

JBuilder のオブジェクトギャラリ内の[一般]タブ内にある「TP1/COBOL SOAP クライアント用 Bean」アイコンをダブルクリックします。

または，メニューバーの[ウィザード]（(注）uCosminexus Developer から提供される JBuilder 2005 では［編集］メニューの［ウィザード］となります）から「TP1/COBOL SOAP クライアント用 Bean」を選択します。

(2)　画面の説明

(a)　ステップ 1/3 画面

「TP1/COBOL SOAP クライアント用 Bean 生成ウィザード」を起動すると，次のような「TP1/COBOL SOAP クライアント用 Bean 生成ウィザード - ステップ 1/3」画面が現れます。

10.5.3 で更新した WSDL ファイルを指定します。

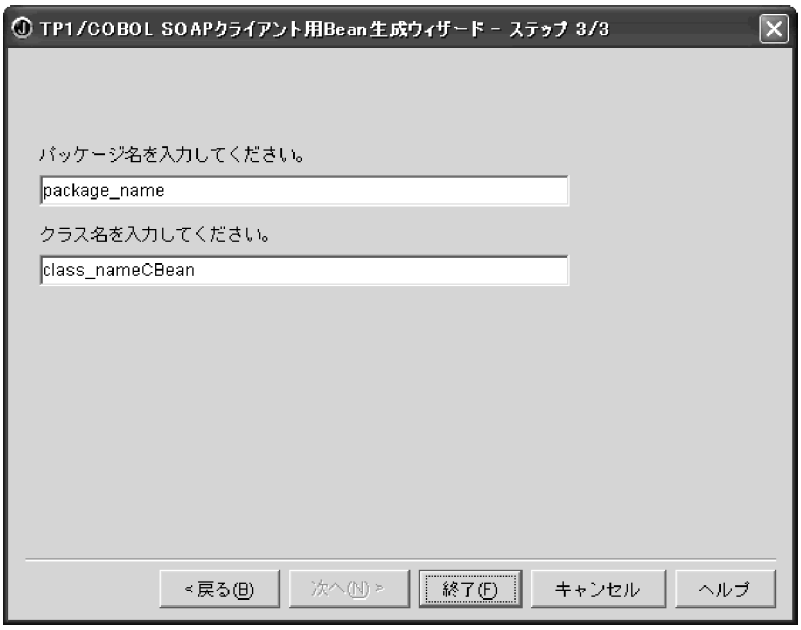
WSDL ファイルを指定し，[次へ>]ボタンを押すと「TP1/COBOL SOAP クライアント用 Bean 生成ウィザード-ステップ 2/3」画面に進みます。

(b)　ステップ 2/3 画面

引数の情報が表示されます。ここで表示される内容はデータ名の別名フィールドおよび制御変数名の別名フィールドがないことを除いて「TP1/COBOL アクセス用 Bean 生成ウィザード - ステップ 2/3」と同じです。詳細は「2.3.2　入出力用の「Bean 生成ウィザード」に表示されるパラメタテーブルの編集」をご覧ください。ただし，選択フィールド以外のフィールドは変更できません。TP1/COBOL SOAP クライアント用 Bean で設定および参照したいデータ項目の「選択」チェックボックスをオンにします。設定

が完了したら，[次へ>]ボタンを押して「TP1/COBOL SOAP クライアント用 Bean 生成ウィザード-ステップ 3/3」画面に進みます。

(c)　ステップ 3/3 画面



次の情報を入力します。指定は省略できません。すべての項目に対して入力が必要です。

- 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つけるにはこのフィールドをクリックして新規の名称（ホスト名など）を入力します。（例：localhost）

- クラス名

  クラスの名称を指定します。初期値は「TP1/COBOL SOAP サーバ用クラス生成ウィザード」で指定したスケルトンクラス名の後に「CBean」を付加した名称が表示されます。ほかのクラス名をつけるにはこのフィールドをクリックして新規の名称を入力します。

  最後に[終了(F)]ボタンを押すと，JBuilder のプロジェクトに TP1/COBOL SOAP クライアント用 Bean となる Java ソースが自動生成されます。自動生成される Java ソースファイル名は，次のようになります。

- TP1/COBOL SOAP クライアント用 Bean：クラス名.java

  （初期値は「スケルトンクラス名称 CBean.java」）

(3)　クライアント側の処理を実装する

自動生成を行った後，SOAP サービスクラスを利用するクライアント側の JavaUAP（サーブレットなど）を作成してください。使用する API については「8.6　TP1/COBOL SOAP サービスクラス用 Bean ユーザインタフェース API」をご覧ください。詳しい作成手順は，マニュアル「Cosminexus SOAP アプリケーション開発ガイド」をご覧ください。

この他，作成例は付録 H.6 に記載しています。

(4) 必要なファイルの配置

SOAP アプリケーションに必要なプロパティファイルや呼び出すときに使用する html ファイルなどを作成して，プロジェクト内に配置してください。詳細は，マニュアル「Cosminexus SOAP アプリケーション開発ガイド」をご覧ください。

(5) war ファイルの作成

「Cosminexus SOAP アプリケーション開発者ガイド」に従って必要なファイル（サーブレットや html ファイルなど）をプロジェクト内に収めたら，JBuilder のメイクを実行して war を作成してください。

10.7　実行環境設定

10.7.1　SOAP アプリケーションの実行環境設定

Cosminexus Version 5 05-05(Windows 版だけ)または Cosminexus Version 6 06-00 以降(Windows/AIX 版だけ)または uCosminexus Application Server 06-70/07-00(Windows 版だけ)で実行することができます。環境設定については，マニュアル「Cosminexus SOAP アプリケーション開発ガイド」をご覧ください。TP1/COBOL 拡張 [Server] Run Time System for Cosminexus Version 2 の実行環境設定は，「4.アプリケーションサーバの環境整備」の Web コンテナサーバの環境設定をご覧ください。

10.7.2　TP1/COBOL SOAP サービスクラス生成機能のシステムプロパティ設定

TP1/COBOL SOAP サービスクラス生成機能を使用する場合のシステムプロパティ設定については，「5.3 TP1/COBOL アクセス環境変数の設定」をご覧ください。このシステムでサポートされているオプションは次のとおりです。

(1)　TP1/COBOL SOAP クライアント用 Bean および TP1/COBOL SOAP サーバ用 Bean に対する Java のシステムプロパティ設定

(a)　tp1cobol.option

- codeconv
- codeconvflag
- comp5
- encode
- endian
- japanese

(b)　tp1cobol.ddump

- 有効：指定したフォルダに引数情報を出力する。

(2)　TP1/COBOL 基本 Bean の cltin，call メソッドなどに対する Java のシステムプロパティ設定(TP1/Client/P または TP1/Client/W)

このシステムでサポートされているオプションは次のとおりです。

(a) tp1cobol.option

- codeconv
- codeconvflag
- comp5
- dccm3
- dccm3flag
- encode
- endian
- japanese

(b) tp1cobol.ddump

- 無効：引数情報の設定および参照は，TP1/COBOL 基本 Bean では行わない。

注意：

TP1/COBOL SOAP サービスクラス生成機能を経由した COBOL SPP 呼び出しでは，環境変数指定は全て無効となります。そのため，TP1/COBOL SOAP サービスクラス生成機能を経由した COBOL SPP 呼び出しと，TP1/COBOL アクセス用 Bean を経由した COBOL SPP 呼び出しを同一の環境で混在して使用する場合，それぞれ-Dtp1cobol.option 指定と CBLJ2TP1OPT 環境変数指定を参照します。したがって，それぞれ異なるオプション指定をすると，同じ COBOL SPP を呼び出しても異なる結果になることがあります。

10.8　プログラムの実行

10.8.1　Web アプリケーションの配置

マニュアル「Cosminexus SOAP アプリケーション開発ガイド」に従って作成した SOAP サーバアプリケーションおよび SOAP クライアントアプリケーションは，war ファイルで作成されています。これらをマニュアル「Cosminexus SOAP アプリケーション開発ガイド」に従って，Cosminexus Application Server に配置してください。

10.8.2　プログラムの実行

作成した war ファイルを配置して，Cosminexus を起動すると，SOAP アプリケーションが展開され使用できるようになります。COBOL SPP と連携するには次のような手順で行ってください。

1. サーバ用およびクライアント用の war ファイルを配置
2. Cosminexus（Hitachi Web Server 含む）を起動
3. COBOL SPP が登録された OpenTP1 サーバを起動
4. COBOL SPP のサービスを開始
5. Web ブラウザを起動する。
6. Web ブラウザから SOAP クライアントアプリケーションを実行し，SOAP サーバアプリケーションを経由して COBOL SPP を実行

11 Web アプリケーションの開発

第 2 編では，Version 2 未満の製品についての説明を行います。Version 2 についての説明は第 1 編をご覧ください。また，Version 2 未満の製品は COBOL2002 に対応しておりませんので，ご注意ください。

なお，Solaris 版 TP1/COBOL アクセスを使用する場合は，Cosminexus Version 4 をご使用ください。

この章では，TP1/COBOL adapter for Cosminexus を使った Web アプリケーションの開発（プログラム作成からコンパイルまで）について説明します。

11.1　Web アプリケーションの開発手順

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 SPP の作成
2. TP1/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

11.2　COBOL SPP の作成

TP1/COBOL アクセスを使用する場合に，COBOL SPP 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。

11.2.1　COBOL　SPP プログラムソースの構成

COBOL　SPP を作成する場合，次の構成で作成することをお勧めします。また，COBOL 言語で作成された既存の SPP がある場合には，この節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 TP1/COBOL アクセスを経由して実行できる SPP として利用することができます。

(1)　COBOL SPP プログラムのソース構成

1. COBOL SPP プログラムソース
2. COBOL－Java 間で渡す引数の登録集原文
   （TP1/COBOL アクセス用 Bean 生成時に必要な登録集原文）

新規に COBOL 言語で SPP を作成する場合の具体的な方法は，マニュアル「分散トランザクション処理機能 OpenTP1 プログラム作成リファレンス COBOL 言語編」など OpenTP1 の各種マニュアルをご覧ください。

COBOL 言語で SPP をコーディングする際，コーディング上の注意および名称の付け方の注意等については，TP1/COBOL の作成方法に従ってください。

(2)　COBOL プログラムの例

COBOL プログラムの例を次に示します。

```
IDENTIFICATION DIVISION.
PROGRAM-ID.     SEARCHTP1.
DATA            DIVISION.
LINKAGE SECTION.
  COPY SEARCHTP1COPYIN.     ------------------- ※
*> 01 PERSONAL-DATA-IN.                  ┐
*>    05 P-NUMBER    PIC 9(9) USAGE COMP.|
*>    05 P-NAME      PIC X(50).          |- 登録原文集の展開
*>    05 P-ADDRESS   PIC X(100).         |
*>    05 P-GIF       PIC X(50).          ┘
  77 IN-LEN        PIC S9(9) USAGE COMP.
  COPY SEARCHTP1COPYOUT.    ------------------- ※
*> 01 PERSONAL-DATA-OUT.                 ┐
*>    05 P-NUMBER    PIC 9(9) USAGE COMP.|
*>    05 P-NAME      PIC X(50).          |- 登録原文集の展開
*>    05 P-ADDRESS   PIC X(100).         |
*>    05 P-GIF       PIC X(50).          ┘
  77 OUT-LEN       PIC S9(9) USAGE COMP.
 PROCEDURE DIVISION   USING PERDONAL-DATA-IN,IN-LEN
                            PERDONAL-DATA-OUT,OUT-LEN.
     MOVE P-NUMBER IN PERDONAL-DATA-IN
        TO P-NUMBER IN PERDONAL-DATA-OUT.
*>  検索処理
     IF P-NUMBER IN PERDONAL-DATA-IN = 100001  THEN
       MOVE  '日立 一郎'       TO  P-NAME IN PERDONAL-DATA-OUT
       MOVE  '日立市'          TO  P-ADDRESS IN PERDONAL-DATA-OUT
       MOVE  '/ICHIRO.GIF'     TO  P-GIF IN PERDONAL-DATA-OUT
     ELSE
:
     END-IF

     EXIT PROGRAM.

 END PROGRAM SEARCHTP1.
```

注※

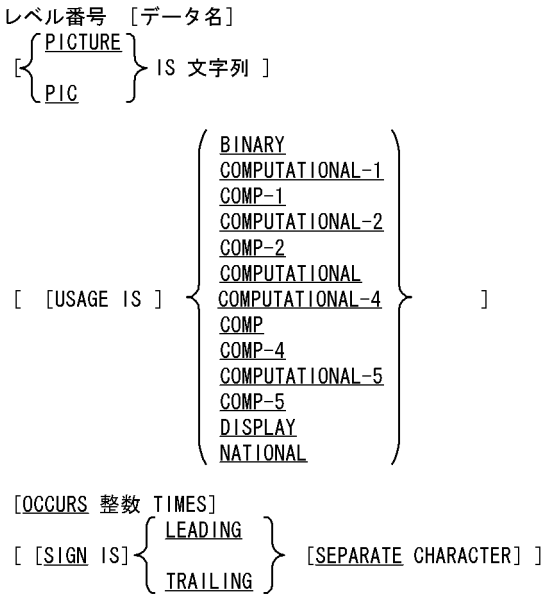
LINKAGE SECTION の各データ名定義は，登録集にすることをお勧めします。

TP1/COBOL アクセス用 Bean 生成時にこの登録集を使用できます。

11.2.2　TP1/COBOL アクセスを使用するための COBOL SPP の引数の規則

TP1/COBOL アクセスを使用するには COBOL SPP の引数は次の規則に従っている必要があります。

【書き方】



注

【書き方】の規則（括弧や下線などの意味）は，マニュアル「COBOL85 言語」の「2. COBOL85 マニュアルの記法」に従っています。

【構文規則】

1. レベル番号，データ名，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 までの範囲内でなければなりません。なお，01 レベルの基本項目は指定できません。
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，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「COBOL85 言語」の構文規則に従ってください。

【一般規則】

1. USAGE 句，OCCURS 句，SIGN 句は，【書き方】で記す表現形式の範囲において，マニュアル「COBOL85 言語」の一般規則に準拠します。
2. レベル番号 01 において，基本項目は指定できません。

【使用上の注意事項】

1. SIGN 句に SEPARATE CHARACTER 指定を書かない場合には，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「COBOL85 言語」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　TP1/COBOL アクセス用 Bean 生成時の COBOL 定義注意事項/制限事項」をご覧ください。付録 A.1 では回避方法も記載しています。

11.2.3　SPP の引数を含む登録集原文の記述規則

1. COBOL SPP 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 SPP の引数の定義は一つ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 文があってはなりません）
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・フリー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 CBLFREE 環境変数に指定した拡張子だけフリー形式，それ以外は固定形式として扱います。

   次に固定形式とフリー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）。

   (フリー形式規則)

   - 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）

   (CBLFREE 環境変数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フリー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,エラーになった箇所以降の値は固定形式として扱います。

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。次の語以外の語は利用者語として扱います。

   BINARY，CHARACTER，COMP，COMP-1，COMP-2，COMP-4，COMP-5，COMPUTATIONAL，COMPUTATIONAL-1，COMPUTATIONAL-2，COMPUTATIONAL-4，COMPUTATIONAL-5，DISPLAY，IS，LEADING，NATIONAL，OCCURS，PIC，PICTURE，POINTER，SEPARATE，SIGN，TIMES，TRAILING，USAGE

   タブ文字は 1 個の空白文字として扱います。

   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。

11.2.4 Version 2 との相違点

TP1/COBOL adapter for Cosminexus Version 2 では次の項目が追加されていますが TP1/COBOL adapter for Cosminexus では未サポートです。

- REDEFINES 句
- COMP-3，COMPUTATIONAL-3 および PACKED-DECIMAL
- バイト配列データ
- COPY 文
- 日本語項目

11.3　TP1/COBOL アクセス用 Bean の生成

ここでは，TP1/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。作成方法には「TP1/COBOL アクセス用 Bean 生成ウィザード」による生成方法と「TP1/COBOL アクセス用 Bean 生成ツール」による生成方法があります。

「TP1/COBOL アクセス用 Bean 生成ウィザード」を使用する場合は，あらかじめ JBuilder に組み込んでおく必要があります。組み込み方法については「付録 C.1 (3)　JBuilder への TP1/COBOL アクセス用 Bean 生成ウィザードの組み込み」をご覧ください。

11.3.1　「TP1/COBOL アクセス用 Bean 生成ウィザード」による生成（推奨）

JBuilder 上で TP1/COBOL アクセス用 Bean 生成ウィザードを使用することによって，COBOL SPP を呼び出すロジックを組み込んだ TP1/COBOL アクセス用 Bean のひな形を簡単に自動生成できます。これ以降「TP1/COBOL アクセス用 Bean 生成ウィザード」を「Bean 生成ウィザード」に省略します。

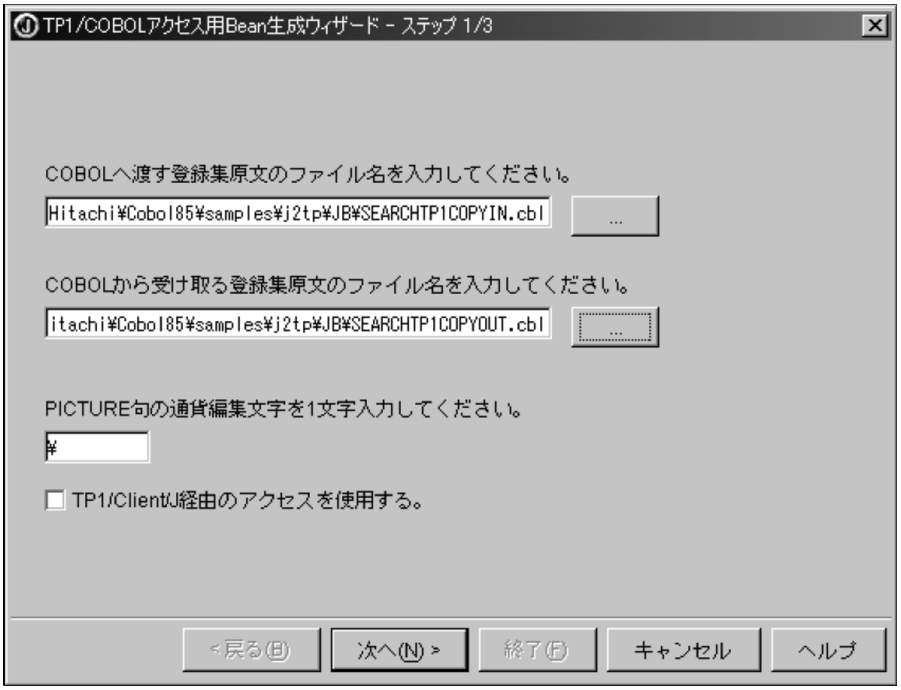
(1)　起動方法

JBuilder のオブジェクトギャラリ内の［新規］タブ内の" TP1/COBOL アクセス用 Bean"アイコンをダブルクリックします。または，JBuilder のメニューバーから［ウィザード］の「TP1/COBOL アクセス用 Bean」をクリックします。

(2)　画面の説明

(a)　ステップ 1/3 画面

「Bean 生成ウィザード」が起動し，次のような「TP1/COBOL アクセス用 Bean 生成ウィザード：ステップ 1/3」画面が現れます。



ここでは，次の情報を入力します。

- SPP の入力パラメタ（COBOL へ渡す登録集原文）と出力パラメタ（COBOL から受け取る登録集原文）を持つデータ定義部の登録集原文のファイル名をフルパスで指定します（(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)。ここで，指定する登録集原文には条件があります。詳細は「11.2.3 SPP の引数を含む登録集原文の記述規則」をご覧ください。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。
- 必要に応じて，PICTURE 句の通貨編集用文字の変更を行います。デフォルトでは，’¥’ が指定されています。
- 「TP1/Client/J 経由のアクセスを使用する。」にチェックした場合は，TP1/Client/J 経由用の Bean を生成します。チェックしない（初期状態）場合は，TP1/Client/P，または TP1/Client/W 経由の Bean を生成します。

設定が完了したら，[次へ(N)>]ボタンを押すと入力用の「TP1/COBOL アクセス用 Bean 生成ウィザード：ステップ 2/3」画面に進みます。

[キャンセル]ボタンを押すと，TP1/COBOL アクセス用 Bean の生成を取り消し，「Bean 生成ウィザード」画面を消去します。また，[ヘルプ]ボタンを押すと，この画面に対するヘルプを表示します。

- 指定された登録集原文を正常に解析できなかった場合，エラーメッセージを表示して処理が中止されます。

(b)　ステップ 2/3 画面

「TP1/COBOL アクセス用 Bean 生成ウィザード：ステップ 2/3」画面には，「TP1/COBOL アクセス用 Bean 生成ウィザード：ステップ 1/3」画面で指定した入出力用の登録集原文を解析した情報から集団項目の情報が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）。

[入力]タブでは，SPP で実際に使用する入力用の引数を選択します。選択したデータ項目に対して TP1/COBOL アクセス用 Bean の中に当該プロパティが生成されます。

SPP で実際に使用するデータ項目だけを抽出することによって，TP1/COBOL アクセス用 Bean のサイズを SPP のサービスを受け取れる必要な大きさに最適化でき，資源を有効に活用できます。

[出力]タブでは，出力用のすべてのデータ項目を選択して TP1/COBOL アクセス用 Bean のプロパティを生成することによって，SPP から返される出力データの妥当性を確認するという使い方ができます。必要に応じて，選択するデータ項目を選んでください。

「パラメタテーブル」の編集の仕様については，「11.3.2　入出力用の「Bean 生成ウィザード」に表示されるパラメタテーブルの編集」をご覧ください。

設定が完了したら，[次へ(N)>]ボタンを押すと，「TP1/COBOL アクセス用 Bean 生成ウィザードーステップ 3/3」画面に進みます。

(c)　ステップ 3/3 画面

「TP1/COBOL アクセス用 Bean 生成ウィザード：ステップ 3/3」画面は，TP1/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です。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。

  「クラス名.java」が生成されるファイル名になります。

以上のように画面に従って操作し，最後に，[終了]ボタンを押すと，JBuilder のプロジェクトに Java ソースファイルが生成され，当該 TP1/COBOL アクセス用 Bean が自動生成されます。

（TP1/COBOL アクセス用 Bean の自動生成ソースイメージは，「付録 E　TP1/COBOL アクセス用 Bean の自動生成ソースイメージ」を参考にしてください。）

11.3.2　入出力用の「Bean 生成ウィザード」に表示されるパラメタテーブルの編集

入出力用の「Bean 生成ウィザード」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。

1. レベル番号フィールド

   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。

   このレベル番号フィールドを編集することはできません。

2. 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。この時，次の条件に該当するデータ項目名が変換されます。

- Java の言語仕様により，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，自動的にアンダーバー（_）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アンダーバー（_）で生成します。
- Bean 生成ウィザード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。規則に反した場合，エラーとなります。
- JavaBeans の命名規則により，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。

3. 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。

表 11-1 に各データ項目に対応する表示文字列を示します。

表 11-1　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 英数字項目 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	文字列データ(String)
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)
集団項目	集団項目

4. 繰り返し回数フィールド

繰り返し回数は 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。

この繰り返し回数フィールドを編集することはできません。

5. 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。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（－），アンダーバー（__）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 生成ウィザード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。

  規則に反した場合，エラーとなります。

6. 選択フィールド

使用する COBOL のデータ項目（すなわち，TP1/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，集団項目は指定できません。デフォルトは，FILLER 以外の選択可能なデータ項目の選択フィールドはすべて選択（チェック印付き）の状態になります。不要なデータは選択（チェック印）を外してください。

選択したデータ項目に対して TP1/COBOL アクセス用 Bean の中に setter/getter が生成されます。SPP で実際に使用するデータ項目を抽出することによって，TP1/COBOL アクセス用 Bean のサイズを SPP のサービスを受け取れる必要な大きさに最適化でき，資源を有効に活用できます。

(1) Version 2 との相違点

「Bean 生成ウィザード-ステップ 2/3」および「Bean 生成ツール-ステップ 2/3」に表示される表の各フィールドの構成および名称が異なります。データ属性フィールドのデータ項目の属性も Version 2 とは異なりますので，ご注意ください。

11.3.3 TP1/COBOL アクセス用 Bean 生成ツール

「TP1/COBOL アクセス用 Bean 生成ツール」は JBuilder を使用しない開発環境で Bean の生成をできるようにしたものです。これ以降「TP1/COBOL アクセス用 Bean 生成ツール」を「Bean 生成ツール」に省略します。

(1) 起動方法

Windows 画面で[スタート]ボタンをクリックし，[プログラム]－[ＣＯＢＯＬ８５]－[TP1/COBOL adapter for Cosminexus]の順でポイントし，[TP1/COBOL アクセス用 Bean 生成ツール環境設定]をク

リックします。「TP1/COBOL アクセス用 Bean 生成ツール環境設定」の画面が開いたら，そこで環境設定を行います。

起動方法も同じように[スタート]－[プログラム]－[ＣＯＢＯＬ８５]－[TP1/COBOL adapter for Cosminexus]の順でポイントし，「TP1/COBOL アクセス用 Bean 生成ツール」をクリックすることで起動できます。

(2) 環境設定ダイアログ

VM オプションのチェックボックスをチェック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。

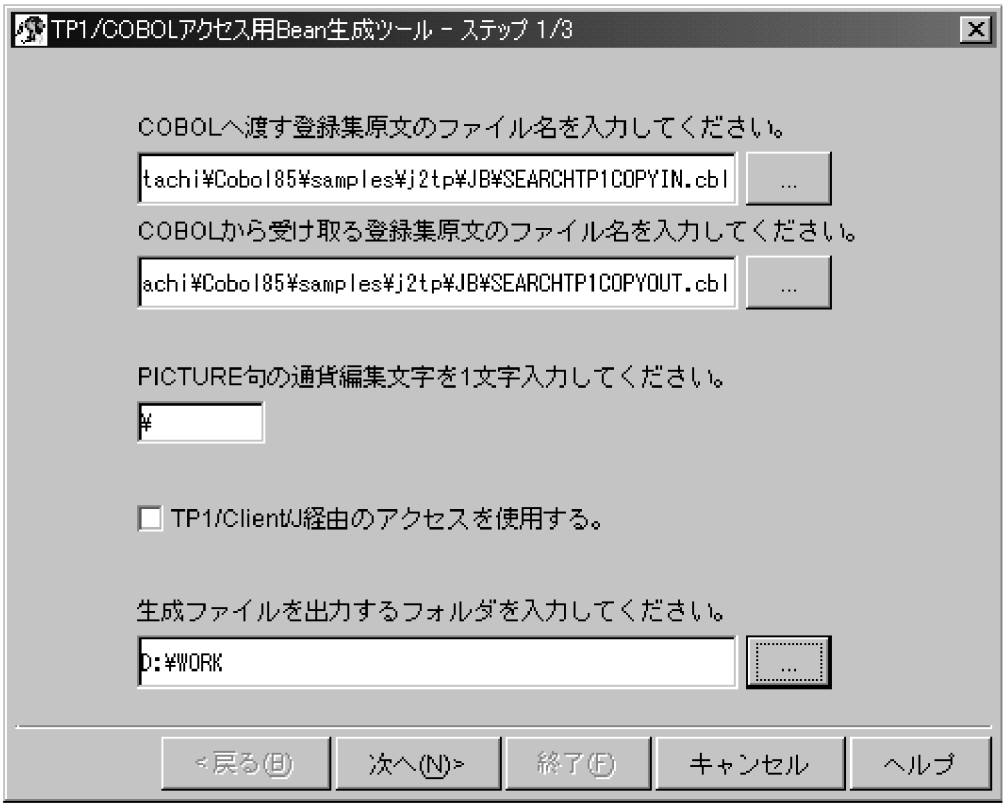
指定がない場合は，JavaVM の初期値となります。

また，VM Option をチェックして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が出力されます。

(3) 「TP1/COBOL アクセス用 Bean 生成ツール」の画面説明

(a) ステップ 1/3 の画面

「TP1/COBOL アクセス用 Bean 生成ツール - ステップ 1/3」の画面の説明をします。2/3 および 3/3 の画面については「11.3.1 「TP1/COBOL アクセス用 Bean 生成ウィザード」による生成（推奨)」をご覧ください。



「生成ファイルを出力するフォルダを入力してください。」以外の項目については「11.3.1 「TP1/COBOLアクセス用 Bean 生成ウィザード」による生成（推奨)」の説明をご覧ください。

「生成ファイルを出力するフォルダを入力してください。」には，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指定します。Java ソースファイルは，次の規則に従って生成されます。

(b) 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¥

パッケージ名とは，ステップ 3/3 画面で指定するパッケージ名を指します。生成する Java ソースファイル名もステップ 3/3 のクラス説明をご覧ください。

「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。

(4) 制限事項

- 生成ツールを二つ以上起動できません。
- ターミナルサービスでは動作しません。

11.4　Servlet の作成

生成した TP1/COBOL アクセス用 Bean を呼び出す Servlet（Java UAP）の作成方法については第 1 編「2.4　Servlet の作成」をご覧ください。

ただし，可変長に関する内容は該当しません。

また，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，第 1 編「8.8　J2CBException ユーザインタフェース API」をご覧ください。

11.5　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 H　プログラム例」の HTML の作成例および JSP の作成例をご覧ください。

11.6　ユーザプログラムの作成上の注意事項

COBOL SPP の作成上の注意事項について説明します。

11.6.1　日本語文字の扱い

このシステムでの日本語文字の扱いは次のとおりです。また，文字コードの設定方法は「5.3　TP1/COBOL アクセス環境変数の設定」をご覧ください。

(1)　Windows/HP-UX/AIX 版

- シフト JIS コードで決められた漢字

  デフォルトはシフト JIS コードですが，CBLJ2TP1OPT 環境変数の 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。

(2)　Solaris 版

- 日本語 EUC コードで決められた漢字
- PCK（シフト JIS）コードで決められた漢字

11.6.2　コンパイラオプションについて

TP1/COBOL アクセスで 1 バイトの 2 進項目（-B1 コンパイラオプション）は使えません。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。

11.6.3　コーディングについて

一つの Servlet または JSP 内では，cltin()～cltout()のメソッドを発行するようにコーディングしてください。cltin メソッド，cltout メソッドおよび call メソッドを別々の Servlet または JSP で行うようなコーディングはしないでください。

11.7 COBOL SPP のコンパイル

COBOL SPP のコンパイル方法は，マニュアル「分散トランザクション処理機能 OpenTP1 プログラム作成リファレンス COBOL 言語編」をご覧ください。

11.8　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。

11.8.1　Windows 版

作成した Java プログラム（Servlet や Bean）は，JBuilder を使用してコンパイルして class ファイルを生成します。

コンパイル時には，JBuilder の[プロジェクトプロパティ]の[必須ライブラリ]に以下の項目を追加してください。

(1)　TP1/Client/P および TP1/Client/W の場合

- [J2tp]ライブラリ

(2)　TP1/Client/J の場合

- [J2tp]ライブラリ
- TP1/Client/J の TP1Client.jar

JBuilder の使用方法は，JBuilder ヘルプをご覧ください。また，JBuilder を使用すると，配布用ウィザードが jar ファイルなどの作成や配布を支援します。

11.8.2　HP-UX/Solaris/AIX 版

- 作成した Java プログラム（Servlet や Bean）は，Windows 環境にて JBuilder を使用してコンパイルし，class ファイルを HP-UX/Solaris/AIX 環境へ移して（バイナリコードで転送する）ご使用ください。
- または，Windows 環境で作成した Java ソースを HP-UX/Solaris/AIX 環境でコンパイルし，class ファイルを生成してご使用ください。

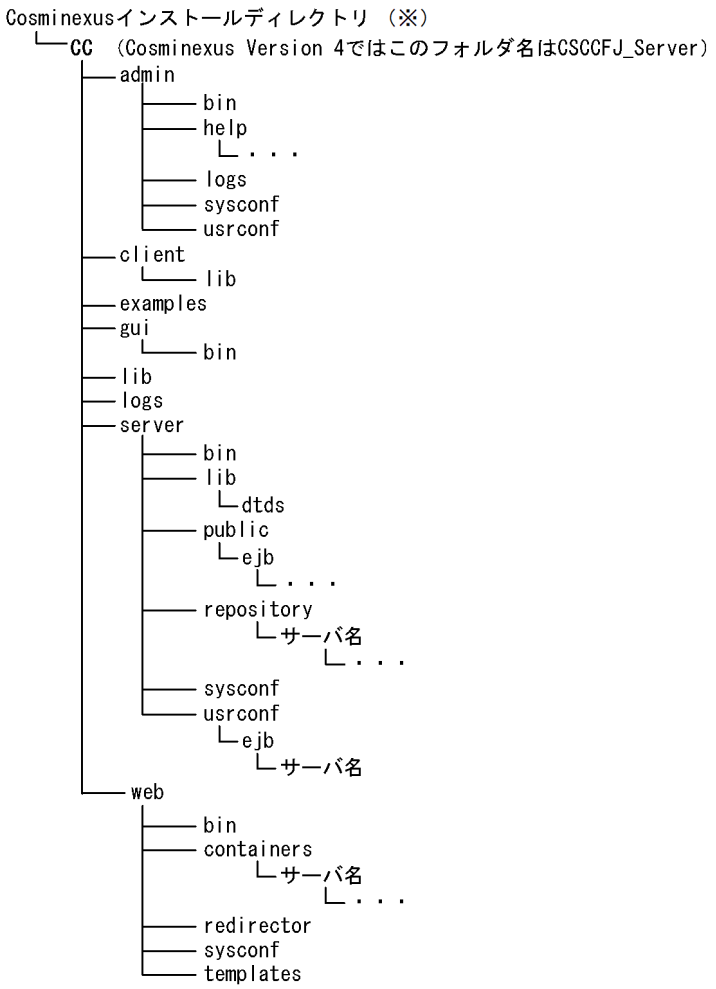
12 アプリケーションサーバの環境整備

作成した COBOL SPP と Java プログラムの実行を実行させるためには，まず実行環境を整えなくてななりません。この章では Web コンテナサーバを使用した場合，TP1/Client/J で J2EE サーバを使用した場合および JRun を使用した場合に分けて説明します。

12.1　環境を説明する前に

実行環境を説明する上で，以下のことを前提とします。

- TP1/COBOL 拡張 [Server] Run Time System for Cosminexus がインストールされていなければなりません。
- TP1 サーバ側の OpenTP1 の起動と停止方法については，マニュアル「分散トランザクション処理機能 OpenTP1 運用と操作」をご覧ください。
- Cosminexus が動作するために必要な定義および環境変数を設定してください。Cosminexus の環境定義の詳細については，マニュアル「Cosminexus 解説」をご覧ください。
- EJB 経由で TP1/Client/J を使用する場合は，J2EE サーバの設定を行ってください(Solaris 版は該当しません)。
- これ以降の説明で使用される Cosminexus Component Container のインストールディレクトリ階層を以下に示します。ただし，Solaris 版の Cosminexus 製品バージョンは Version 4 です。



注※

Cosminexus インストールディレクトリは，Windows 版ではデフォルトインストール先が「C:¥Program Files¥Hitachi¥Cosminexus」となっていますが，インストール時に変更可能です。それに対して HP-UX/Solaris/AIX 版では/opt/cosminexus と固定になっています。

そのため，CC までのディレクトリ構成(下の図)を<CC インストール先>と省略して以下説明します。

```
Cosminexusインストールディレクトリ
    └── CC
```

(例)

server ディレクトリを表す場合は，Windows 版では<CC インストール先>¥server，HP-UX/Solaris/AIX 版では<CC インストール先>/server と表現します。

12.2　Web コンテナサーバを使用した場合

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する必要があります。

- Web サーバ
- Web コンテナサーバ

Web サーバの設定についてはご使用の Web サーバのマニュアルをご覧ください。

12.2.1　TP1/Client/P 使用時の環境変数の設定（Windows 版）

TP1/Client/P 使用時は TP1/COBOL アクセス Windows 版を使った設定を行います。Web コンテナサーバの CLASSPATH 環境変数の設定および Library Path 環境変数の設定を行ってください。設定内容について以下に説明します。

(1)　CLASSPATH 環境変数の設定

次の手順にて CLASSPATH 環境変数に「<TP1/COBOL 拡張 Server Run Time System for Cosminexus インストールフォルダ>¥LIB¥j2tp1run.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¥Cobol85¥LIB¥j2tp1run.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)　Library Path の設定

次の手順で Library Path を設定します。

1. Windows の[スタート]メニューから[設定]-[コントロールパネル]の順で[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. システム環境変数の PATH 環境変数に「<TP1/COBOL 拡張 Server Run Time System for Cosminexus 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。

6.（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。）

12.2.2　TP1/Client/W 使用時の環境変数の設定（HP-UX/Solaris/AIX 版）

(1)　CLASSPATH 環境変数の設定

TP1/Client/W 使用時は TP1/COBOL アクセス HP-UX/Solaris/AIX 版を使った設定を行います。Web コンテナサーバの CLASSPATH 環境変数の設定および Library Path 環境変数の設定を行ってください。設定内容について以下に説明します。

次の手順にて CLASSPATH 環境変数に”/opt/HILNGcbl/lib/j2tp1run.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/HILNGcbl/lib/j2tp1run.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)　Library Path の設定

次の手順で Library Path を設定します。

(a)　HP-UX 版

1. システム環境変数の SHLIB_PATH 環境変数に”/opt/HILNGcbl/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。

(b)　Solaris 版

1. システム環境変数の LD_LIBRARY_PATH 環境変数に”/opt/HILNGcbl/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。

(c)　AIX 版

1. システム環境変数の LIBPATH 環境変数に”/opt/HILNGcbl/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。

12.2.3　TP1/Client/J 使用時の環境変数の設定(Windows/HP-UX/AIX 版)

TP1/Client/J をお使いの場合は，TP1/Client/P または TP1/Client/W と同様の設定をした後，任意のディレクトリ下に TP1Client.jar を格納し，CLASSPATH 環境変数に TP1Client.jar を追加します。

(1)　Windows 環境の場合（TP1/Client/P と同様）

以下に任意のフォルダを C:¥CLTJ として説明します。

1. C:¥CLTJ フォルダを作成し，そのフォルダ内に TP1Client.jar を格納します。
2. TP1/Client/P と同様に Web コンテナサーバのユーザ定義ファイル（usrconf.cfg）に C:¥CLTJ¥TP1Client.jar を追加します。詳細はマニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

＜指定例＞

```
web.add.class.path=C:¥CLTJ¥TP1Client.jar
```

(2)　HP-UX/AIX 環境の場合（TP1/Client/W と同様）

以下に任意のディレクトリを/usr/cltj として説明します。

1. /usr/cltj ディレクトリを作成し，そのディレクトリ内に TP1Client.jar を格納します。
2. TP1/Client/W と同様に Web コンテナサーバのユーザ定義ファイル（usrconf.cfg）に/usr/cltj/TP1Client.jar を追加します。詳細はマニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

＜指定例＞

```
web.add.class.path=/usr/cltj/TP1Client.jar
```

12.3　EJB 経由で TP1/Client/J を使用した場合(Windows/HP-UX/AIX 版)

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する必要があります。

- Web サーバ
- Web コンテナサーバまたは JRun
- J2EE サーバ

Web サーバの設定についてはご使用の Web サーバのマニュアルをご覧ください。

12.3.1　J2EE サーバの環境変数の設定

J2EE サーバでは CLASSPATH 環境変数の設定および Library Path 環境変数の設定を行ってください。設定内容について以下に説明します。

(1)　J2EE サーバのユーザ定義

J2EE サーバのユーザ定義ファイル usrconf.cfg で次の設定を行ってください。

(a)　Windows 版

1. add.class.path キーに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus インストールフォルダ>¥LIB¥j2tp1run.jar」を設定します。
2. add.class.path キーに「任意フォルダ¥TP1Client.jar」を設定します。
3. add.library.path キーに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus インストールフォルダ>¥BIN」を設定します。

usrconf.cfg は<CC インストール先>¥server¥usrconf¥ejb¥<サーバ名称>の下に格納されています。

＜ 設定例 ＞

```
add.class.path=C:¥PROGRA~1¥Hitachi¥Cobol85¥LIB¥j2tp1run.jar
add.class.path=C:¥CLTJ¥TP1Client.jar
add.library.path=C:¥PROGRA~1¥Hitachi¥Cobol85¥BIN
```

設定例では TP1/Client/J のインストール先を任意のフォルダ「C:¥CLTJ」としています。

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。

(b) HP-UX/AIX 版

1. add.class.path キーに「/opt/HILNGcbl/lib/j2tp1run.jar」を設定します。
2. add.class.path キーに「任意ディレクトリ/TP1Client.jar」を設定します。
3. add.library.path キーに「/opt/HILNGcbl/lib」を設定します。

usrconf.cfg は<CC インストール先>/server/usrconf/ejb/<サーバ名称>の下に格納されています。

＜ 設定例 ＞

```
add.class.path=/opt/HILNGcbl/lib/j2tp1run.jar
add.class.path=/usr/cltj/TP1Client.jar
add.library.path=/opt/HILNGcbl/lib
```

設定例では TP1/Client/J のインストール先を任意のディレクトリ「/usr/cltj」としています。

12.3.2 J2EE サーバの再起動

J2EE サーバがすでに起動されている場合は，環境設定を有効にするために再起動してください。詳細はマニュアル「Cosminexus Component Container 使用の手引」をご覧ください。

12.4　JRun を使用した場合

アプリケーションサーバとして JRun を使用した場合の実行環境の設定について説明します。

JRun 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
- JRun

Web サーバの設定についてはご使用の Web サーバのマニュアルをご覧ください。

AIX 版の JRun はありません。

12.4.1　TP1/Client/P 使用時の環境変数の設定（Windows 版）

TP1/Client/P 使用時は TP1/COBOL アクセス Windows 版を使った設定を行います。JRun のバージョンによって設定個所の名称や画面名称が異なることがあるので，画面を使っての説明は省きます。お使いの JRun で ClassPath や LibraryPath を以下のように修正してください。

(1)　CLASSPATH 環境変数の設定

次の手順にて CLASSPATH 環境変数に「<TP1/COBOL 拡張 Server Run Time System for Cosminexus インストールフォルダ>¥LIB¥j2tp1run.jar」を設定します。

1. JRun の画面中の Java Setting で ClassPath フィールドに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus インストールフォルダ>¥LIB¥j2tp1run.jar」を追加します。
2. 設定内容を更新します。
3. JRun を再起動します。

(2)　LibraryPath 環境変数の設定

次の手順にて JRun の LibraryPath 環境変数に「<TP1/COBOL 拡張 Server Run Time System for Cosminexus インストールフォルダ>¥BIN」を設定します。

1. JRun の画面中の Java Setting で LibraryPath フィールドに「<TP1/COBOL 拡張 Server Run Time System for Cosminexus インストールフォルダ>¥BIN」を追加します。
2. 設定内容を更新します。
3. JRun を再起動します。

12.4.2　TP1/Client/W 使用時の環境変数の設定（HP-UX/Solaris 版）

JRun のバージョンによって設定個所の名称や画面名称が異なることがあるので，画面を使っての説明は省きます。お使いの JRun で ClassPath や LibraryPath を以下のように修正してください。

(1)　CLASSPATH 環境変数の設定

次の手順にて CLASSPATH 環境変数に"/opt/HILNGcbl/lib/j2tp1run.jar"を設定します。

1. JRun の画面中の Java Setting で ClassPath フィールドに"/opt/HILNGcbl/lib/j2tp1run.jar"を追加します。
2. 設定内容を更新します。
3. JRun を再起動します。

(2)　LibraryPath 環境変数の設定

次の手順にて LibraryPath 環境変数に"/opt/HILNGcbl/lib"を設定します。

1. JRun の画面中の Java Setting で LibraryPath フィールドに"/opt/HILNGcbl/lib"を追加します。
2. 設定内容を更新します。
3. JRun を再起動します。

12.4.3　TP1/Client/J 使用時の環境変数の設定(Windows/HP-UX 版)

TP1/Client/J をお使いの場合は，TP1/Client/P または TP1/Client/W と同様の設定をした後，任意のディレクトリ下に TP1Client.jar を格納し，CLASSPATH 環境変数に TP1Client.jar を追加します。

(1)　Windows 版（TP1/Client/P と同様）

以下に任意のフォルダを C:¥CLTJ として説明します。

1. C:¥CLTJ フォルダを作成し，そのフォルダ内に TP1Client.jar を格納します。
2. TP1/Client/P と同様に JRun の Java Setting で ClassPath フィールドに C:¥CLTJ¥TP1Client.jar を追加します。詳細はマニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

(2)　HP-UX 版（TP1/Client/W と同様）

以下に任意のディレクトリを/usr/cltj として説明します。

1. /usr/cltj ディレクトリを作成し，そのディレクトリ内に TP1Client.jar を格納します。

2. TP1/Client/W と同様に JRun の Java Setting で ClassPath フィールドに/usr/cltj/TP1Client.jar を追加します。詳細はマニュアル「分散トランザクション処理機能 OpenTP1 クライアント使用の手引 TP1/Client/J 編」をご覧ください。

13 COBOL SPP と TP1/COBOL アクセスの環境整備

この章では，TP1/COBOL アクセス環境下において，COBOL SPP の実行環境を設定するための環境変数を指定する方法について説明します。設定は OpenTP1 側で行います。

13.1　COBOL SPP 環境変数の設定

TP1/COBOL アクセス環境下において，COBOL SPP の実行環境を設定するための環境変数を指定する方法について説明します。設定は OpenTP1 側で行います。

13.1.1　Windows 版（TP1/LiNK の場合）

1. [スタート]メニューから[TP1/LiNK]－＞[アプリケーション管理 SPP]の順で TP1/LiNK アプリケーション管理 SPP 画面を開きます。
2. TP1/LiNK アプリケーション管理 SPP 画面の[サーバ定義]ボタンを押すと，アプリケーション環境 SPP 画面が開きます。
3. アプリケーション環境 SPP 画面のユーザサーバ名を選択して[開く]ボタンを押すと，SPP 環境設定画面が開きます。
4. SPP 環境設定画面の[ユーザサーバの環境変数]の中に諸変数を設定します。

詳細な設定方法についてはマニュアル「分散アプリケーションサーバ TP1/LiNK 使用の手引」をご覧ください。

13.1.2　HP-UX 版（TP1/LiNK の場合）

コマンドラインから dcsysset コマンドを入力します。

```
入力内容　　[　dcsysset　-u　ユーザサーバ名　　]
```

詳細な設定方法についてはマニュアル「分散アプリケーションサーバ TP1/LiNK 使用の手引」をご覧ください。

13.1.3　HP-UX/Solaris/AIX 版（TP1/Server Base の場合）

コマンドラインから変更したいユーザサーバのユーザサービス定義ファイル（$DCCONFPATH/ユーザサーバ名）に変数を設定します。

```
入力内容　　[　putenv 環境変数名　環境変数値　　]
```

詳細な設定方法についてはマニュアル「分散トランザクション処理機能 OpenTP1 システム定義」をご覧ください。

13.2　TP1/COBOL アクセスの環境変数の設定

TP1/COBOL アクセスの環境変数の設定について説明します。

13.2.1　CBLJ2TP1OPT 環境変数

TP1/COBOL アクセス実行環境変数"CBLJ2TP1OPT"は，TP1/COBOL アクセス実行時ライブラリに渡すオプションを設定することができます。設定方法は次のとおりです。なお，設定に反している場合は無効となります。

(1)　オプションの設定方法（Windows 版）

オプションの設定方法を TP1/Client/P および TP1/Client/J に分けて説明します。各オプションの種類および説明は 13.2.2 以降をご覧ください。

(a)　TP1/Client/P

環境変数は，シェル，コマンドプロンプトまたはコントロールパネルの中のシステムから次の形式で設定します。

[指定形式]

```
set CBLJ2TP1OPT=[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 encode オプションと endian オプションです。

(b)　TP1/Client/J

CBLJ2TP1OPT 環境変数にオプションを設定するのではなく，Java のシステムプロパティにオプション tp1cobol.option を設定する点が，TP1/Client/P と異なるのでご注意ください。オプションの設定値は同じです。

- web コンテナサーバの場合

  usrconf.cfg ファイルの"add.jvm.arg"に-D オプションを付けて，tp1cobol.option を指定します。

  [指定形式]

  ```
  add.jvm.arg=-Dtp1cobol.option=OPTION [ : OPTION ]
  ```

  [usrconf.cfg 格納フォルダ]

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

- J2EE サーバの場合

  usrconf.cfg ファイルの"add.jvm.arg"に-D オプションを付けて，tp1cobol.option を指定します。

[指定形式]

```
add.jvm.arg=-Dtp1cobol.option=OPTION [ : OPTION ]
```

[usrconf.cfg 格納フォルダ]

<CCインストール先>¥server¥usrconf¥ejb¥サーバ名

- JRun の場合

  JRun 管理コンソール起動後，「Java Setting」の「Java Arguments」フィールドに-D オプションを付けて，tp1cobol.option を指定します。

  [指定形式]

  ```
  -Dtp1cobol.option=OPTION [ : OPTION ]
  ```

(2) オプションの設定方法(HP-UX/Solaris/AIX 版)

オプションの設定方法を TP1/Client/W および TP1/Client/J に分けて説明します。各オプションの種類および説明は 13.2.2 以降をご覧ください。

(a) TP1/Client/W

HP-UX/Solaris/AIX の環境変数に次の形式で設定します。

[指定形式]

```
CBLJ2TP1OPT="[OPTION][: OPTION]"
export CBLJ2TP1OPT
```

[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 encode オプションと endian オプションです。

(b) TP1/Client/J(HP-UX/AIX 版)

CBLJ2TP1OPT 環境変数にオプションを設定するのではなく，Java のシステムプロパティにオプション tp1cobol.option を設定する点が，TP1/Client/W と異なるのでご注意ください。オプションの設定値は同じです。

- Web コンテナサーバの場合

  usrconf.cfg ファイルの"add.jvm.arg"に，-D オプションを付けて，tp1cobol.option を指定します。

  [指定形式]

  add.jvm.arg=-Dtp1cobol.option=OPTION [: OPTION]

  [usrconf.cfg 格納ディレクトリ]

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

- J2EE サーバの場合

  usrconf.cfg ファイルの"add.jvm.arg"に，-D オプションを付けて，tp1cobol.option を指定します。

[指定形式]

```
add.jvm.arg=-Dtp1cobol.option=OPTION [ : OPTION ]
```

[usrconf.cfg 格納ディレクトリ]

<CCインストール先>/server/usrconf/ejb/<サーバ名>

- JRun の場合(HP-UX 版)

  JRun 管理コンソール起動後，「Java Setting」の「Java Arguments」フィールドに，-D オプションを付けて，tp1cobol.option を指定します。

  [指定形式]

```
-Dtp1cobol.option=OPTION [ : OPTION ]
```

13.2.2 encode オプション

(1) 機能

TP1/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

encode は小文字，エンコード名は任意のエンコードで指定します。

(a) Windows 版

```
set CBLJ2TP10PT=encode(エンコード名)
```

(b) HP-UX/Solaris/AIX 版

encode オプションは引用符（"）で囲んで指定します。

```
CBLJ2TP10PT＝”encode(エンコード名)”
export CBLJ2TP10PT
```

(3) 指定例

(a) Windows 版

```
set CBLJ2TP10PT=encode(MS932)
```

(b) HP-UX/Solaris/AIX 版

```
CBLJ2TP10PT = ”encode(MS932)”
export CBLJ2TP10PT
```

(4)　encode オプション指定無しの時

文字列変換のエンコードはシステムに依存します。

(a)　Windows 版

デフォルトエンコードは「MS932」となります。

(b)　HP-UX 版

環境変数 LANG が"ja_JP.SJIS"のときは，デフォルトエンコード「SJIS」となります。

(c)　Solaris 版

環境変数 LANG が"ja"のときは，デフォルトエンコード「EUC_JP」となります。

(d)　AIX 版

環境変数 LANG が"Ja_JP"のとき（SJIS）は，デフォルトエンコード「Cp943C」となります。

(5)　encode オプション指定の時

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。ただし，指定された文字列の値のチェックはしませんので指定時には大文字，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。対象としているエンコードは「MS932」，「SJIS」，「EUC_JP」，「Cp943C」です。

(6)　エンコード対象となる項目

1. COBOL SPP 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
2. cltin メソッドの引数 defpath
3. setConnectInf メソッドの引数 inf
4. receive メソッドの引数 buff
5. receive2 メソッドの引数 buff
6. send メソッドの引数 buff
7. acceptNotification メソッドの引数 defpath，および inf
8. cancelNotification メソッドの引数 defpath，および inf

13.2.3　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 と UNIX とで異なります。Windows ではリトルエンディアン形式，UNIX ではビッグエンディアン形式となります。Cosminexus と OpenTP1 間で異なる OS でバイナリ形式のデータを流通させたい場合に，このオプションを使って，流通させたいエンディアン形式に変換することができます。

詳細については，「9.3　Windows/UNIX の数字データ格納形式の相違」をご覧ください。

なお，このオプションは Solaris 版 TP1/COBOL アクセスでは指定できません。

(2)　指定方法

endian(little | big)：

　endian は小文字，endian の種類には little または big のいずれかを指定します。

(a)　Windows 版

```
SET CBLJ2TP10PT = endian( little | big )
```

(b)　HP-UX/AIX 版

```
CBLJ2TP10PT＝"endian( little | big )"
export CBLJ2TP10PT
```

(3)　endian オプション指定無しの時

(a)　Windows 版

endian の値は既定値で little です。

(b)　HP-UX/AIX 版

endian の値は既定値で big です。

(4)　注意事項

- このオプションは 2 進項目および浮動小数点項目に有効です。
- Windows 版 TP1/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-Bb コンパイラオプションと-Fb コンパイラオプションを指定してください。
- Windows 版 TP1/COBOL アクセスで，COMPUTATIONAL-5/COMP-5 のデータ項目がある場合は，このオプションに big は指定しないでください。

14 プログラムの実行とデバッグ

プログラムを実行するためにはこれまで作成してきたおのおのファイルを適切な場所に配置することが必要となります。この章では，プログラムの実行方法をファイルの種類に分けて説明します。

14.1　Web アプリケーションの配置

Web アプリケーションの配置等に関しては，第 1 編「6.1　Web アプリケーションの配置」をご覧ください。

Solaris 版に関しては，HP-UX/AIX 版の記述と同じです。

14.2　サーブレットの登録

サーブレットの登録に関しては，第 1 編「6.2 サーブレットの登録」をご覧ください。

Solaris 版に関しては，HP-UX/AIX 版の記述と同じです。

14.3　プログラムの実行

プログラムの実行方法に関しては，第 1 編「6.3 プログラムの実行」をご覧ください。

Solaris 版に関しては，HP-UX 版の記述と同じです。

14.4　プログラムのデバッグ

「プログラムのデバッグ」に関しては，第 1 編「7.　プログラムのデバッグ」をご覧ください。

付録

付録 A　注意事項 / 制限事項

TP1/COBOL アクセス用 Bean 生成時の COBOL 定義注意事項と制限事項を付録 A.1 で，TP1/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付録 A.2 で説明します。

付録 A.1　TP1/COBOL アクセス用 Bean 生成時の COBOL 定義注意事項/制限事項

TP1/COBOL アクセス用 Bean 生成時は，COBOL SPP/MHP 引数定義を別ファイルにして，入力用／出力用 COBOL データ定義として指定する必要があります。この COBOL データ定義において，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 SPP/MHP の引数を変更してください。

表 A-1　TP1/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	該当バージョン	回避方法詳細
1	EXTERNAL 句	全バージョン	(1)に記載
2	GLOBAL 句	全バージョン	(2)に記載
3	BLANK WHEN ZERO 句	全バージョン	(3)に記載
4	SYNCHRONIZED 句	全バージョン	(4)に記載
5	JUSTIFIED 句	全バージョン	(5)に記載
6	VALUE 句	全バージョン	(6)に記載
7	RENAMES 句	全バージョン	(7)に記載
8	条件名	全バージョン	(8)に記載
9	外部浮動小数点項目	全バージョン	(9)に記載
10	外部ブール項目	全バージョン	(10)に記載
11	OCCURS 句（OCCURS 整数 TIMES［DEPENDING ON データ名］以外）	全バージョン	(11)に記載
12	位取り	全バージョン	(12)に記載
13	フリー形式の行内注記（*>）	全バージョン	(13)に記載
14	修飾	全バージョン	(14)に記載
15	登録集原文中の COPY 文 (COPY{原文名|原文名定数}.以外)	全バージョン	(15)に記載
16	SAME AS 句※	全バージョン	(16)に記載

項番	COBOL 定義	該当バージョン	回避方法詳細
17	TYPE 句および TYPEDEF 句※	全バージョン	(17)に記載
18	翻訳指令※	全バージョン	(18)に記載
19	入力情報エリアサイズ値の変更	02-02 未満	(19)に記載
20	出力情報エリアサイズ値の変更	02-00 未満	(20)に記載
21	内部 10 進項目	02-00 未満	(21)に記載
22	内部ブール項目	02-00 未満	(22)に記載
23	REDEFINES 句	02-00 未満	(23)に記載
24	01 および 77 レベルの基本項目	02-00 未満	(24)に記載
25	登録集原文中の COPY 文 (COPY{原文名|原文名定数}.)	02-02 未満	(25)に記載
26	OCCURS 句(OCCURS 整数 TIMES [DEPENDING ON データ名])	02-05 未満	(26)に記載

注※　COBOL2002 で新しく追加された機能です。

(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

LINKAGE SECTION で指定できない 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 SHNCHRONIZE 句の記述個所をご覧ください。

[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。再命名したデータ項目を使用する場合は，代わりに被再命名項目を使用するか，REDEFINES 句と同様に登録集原文を別に作成して使用してください。

[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。英数字項目の桁数は，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) OCCURS 句（OCCURS 整数 TIMES［DEPENDING ON データ名］以外）

「OCCURS 整数 TIMES」および「DEPENDING ON データ名」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。

[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).
```

(12) 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。

(13) フリー形式の行内注記（*>）

使用できません。削除してください。

(14) 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 生成ウィザードで別名を指定して，一意の名称にしてください。

(15) 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。

(16) SAME AS 句

使用できません。削除してください。

(17) TYPE 句および TYPEDEF 句

使用できません。削除してください。

(18)　翻訳指令

使用できません。削除してください。

(19)　入力情報エリアサイズ値の変更

(02-02 以降)

COBOL プログラム(SPP)に渡す入力情報エリアサイズを setInLenFollowSetData メソッドとそれに続く setter メソッドにより変更することができます。詳細は「8.2.2 TP1/COBOL アクセス用 Bean ユーザインタフェース API」をご覧ください。

(02-02 未満)

入力情報エリアサイズを変更することはできません。

(20)　出力情報エリアサイズ値の変更

(02-01 以降，ただし TP1 Connector が対応しているバージョンは 02-02 以降)

COBOL プログラム(SPP)で出力情報サイズを設定している場合に，取得しようとするデータが取得できない（出力情報サイズで指定された有効データに含まれない）ことがあります。データが取得できる場合と取得できない場合について，次にまとめます。

表 A–2　　情報エリアサイズ変更時にデータ取得動作

データ項目	有効データに含まれない	有効データに一部分含まれる	有効データに全部含まれる
文字データ (String)	例外発生 J2CByyy2051	データを取得する※	データを取得する
文字データ (バイト配列 byte[])	例外発生 J2CByyy2051	データを取得する	データを取得する
文字データ 以外	例外発生 J2CByyy2051	例外発生 J2CByyy2052	データを取得する

注※

文字データ（String）を一部分取得する場合，設定されているデータによって，コード変換ができない場合があります。変換できなかった場合，文字数が 0 文字の String オブジェクトとなる場合があります。また，'?'に変換されることもあります。その場合は，「(1)COBOL プログラム(SPP)で設定している出力情報サイズを，設定したデータ値が十分に含まれるサイズで設定する」か，「(2)Unicode にない文字を使っている場合は，文字を削除するか Unicode に変換できる別の文字に変更する」ようにしてください。

補足：TP1/COBOL-Java 連携では，COBOL プログラム(SPP)を呼び出す際，第 4 引数は，第 3 引数(出力情報定義エリア）サイズをもとに算出した値を設定しています。第 3 引数は，COBOL プログラム(SPP)を呼び出す際にデータ領域を確保しますが，確保した領域長と異なる長さを COBOL プログラム(SPP)で第 4 引数に設定されると，上記のような動作となります。

また，第４引数（出力情報定義エリアサイズ）の取得はできません。

(02-01 未満)

COBOL プログラム(SPP)で出力情報サイズを設定している場合は，登録集原文で指定する引数の長さを設定値と一致するように変更し，Bean の生成を行ってください。

補足：TP1/COBOL-Java 連携では，COBOL プログラム(SPP)を呼び出す際，第４引数は，第３引数（出力情報定義エリア）サイズをもとに算出した値を設定しています。第３引数は，COBOL プログラム(SPP)を呼び出す際にデータ領域を確保しますが，確保した領域長と異なる長さを COBOL プログラム(SPP)で第４引数に設定されると，正常に動作しない場合があります。

(21) 内部 10 進項目（USAGE COMP-3／COMPUTATIONAL-3／PACKED-DECIMAL）

(02-00 以降)

この機能は Version 2 でサポートされました。

(02-00 未満)

使用できません。回避方法がありませんので，削除するか，COBOL SPP/MHP 引数定義を変更してください。

(22) 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。英数字項目の桁数は，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).
```

(23) REDEFINES 句

(02-00 以降)

この機能は Version 2 でサポートされました。

(02-00 未満)

異なるデータ属性で，同一の領域を示す REDEFINES 句は使用できません。以下に示す方法で，COBOL データ定義を変更してください。

1. 再定義された（REDEFINES データ名で指定された）データ項目（被再定義項目）を使用したい場合は，「再定義項目の REDEFINES 句だけを削除するのではなく，再定義項目と従属するデータ項目も削除する」ようにしてください。
2. 再定義した（REDEFINES 句指定のある）データ項目（再定義項目）を使用したい場合は，「被再定義項目及び使用しない再定義項目とそれらに従属するデータ項目を削除し，使用する再定義項目の REDEFINES 句だけを削除する」ようにしてください。
3. 異なるデータ属性で同一の領域を使用したい場合は，項番 1，項番 2 で示した手順にそって必要な登録集原文を作成してください。この場合，複数個の登録集原文から対応した Bean を作成し，一つの Java プログラムから用途に応じて呼び分けてください。

[例]

＜変更前＞

```
01  HTC-INPUT-DATA.
   03  HTC-INPUT-HEADER.
     05  HTC-INPUT-COMMON1            PIC  X(05).
     05  HTC-INPUT-COMMON2            PIC  X(05).
   03  HTC-INPUT-DETAIL1              PIC  X(500).
   03  HTC-INPUT-DETAIL2     REDEFINES   HTC-INPUT-DETAIL1.
     05  HTC-INPUT-DETAIL2-CODE       PIC  X(20).
*>
   03  HTC-INPUT-DETAIL3     REDEFINES   HTC-INPUT-DETAIL1.
     05  HTC-INPUT-DETAIL3-CODE       PIC  X(20).
     05  HTC-INPUT-DETAIL3-COUNT      PIC S9(10)
           DISPLAY  SIGN  LEADING  SEPARATE.
     05  HTC-INPUT-DETAIL3-DATE       PIC  X(10).
     05  HTC-INPUT-DETAIL3-DATA-NO    PIC  X(20).
     05  HTC-INPUT-DETAIL3-ID-NO      PIC  X(05).
```

＜変更後＞

1. 被再定義項目を使用する場合

```
01  HTC-INPUT-DATA.
   03  HTC-INPUT-HEADER.
     05  HTC-INPUT-COMMON1            PIC  X(05).
     05  HTC-INPUT-COMMON2            PIC  X(05).
   03  HTC-INPUT-DETAIL1              PIC  X(500).
   03  HTC-INPUT-DETAIL2     REDEFINES   HTC-INPUT-DETAIL1.
     05  HTC-INPUT-DETAIL2-CODE       PIC  X(20).
*>
   03  HTC-INPUT-DETAIL3     REDEFINES   HTC-INPUT-DETAIL1.
     05  HTC-INPUT-DETAIL3-CODE       PIC  X(20).
     05  HTC-INPUT-DETAIL3-COUNT      PIC S9(10)
           DISPLAY  SIGN  LEADING  SEPARATE.
     05  HTC-INPUT-DETAIL3-DATE       PIC  X(10).
     05  HTC-INPUT-DETAIL3-DATA-NO    PIC  X(20).
     05  HTC-INPUT-DETAIL3-ID-NO      PIC  X(05).
```

2. 再定義項目を使用する場合

- 再定義項目 1 を使用する場合 -

```
01  HTC-INPUT-DATA.
   03  HTC-INPUT-HEADER.
     05  HTC-INPUT-COMMON1            PIC  X(05).
     05  HTC-INPUT-COMMON2            PIC  X(05).
   03  HTC-INPUT-DETAIL1              PIC  X(500).
   03  HTC-INPUT-DETAIL2     REDEFINES   HTC-INPUT-DETAIL1.
     05  HTC-INPUT-DETAIL2-CODE       PIC  X(20).
*>
   03  HTC-INPUT-DETAIL3     REDEFINES   HTC-INPUT-DETAIL1.
     05  HTC-INPUT-DETAIL3-CODE       PIC  X(20).
     05  HTC-INPUT-DETAIL3-COUNT      PIC S9(10)
           DISPLAY  SIGN  LEADING  SEPARATE.
     05  HTC-INPUT-DETAIL3-DATE       PIC  X(10).
     05  HTC-INPUT-DETAIL3-DATA-NO    PIC  X(20).
     05  HTC-INPUT-DETAIL3-ID-NO      PIC  X(05).
```

- 再定義項目 2 を使用する場合 -

```
01  HTC-INPUT-DATA.
   03  HTC-INPUT-HEADER.
     05  HTC-INPUT-COMMON1            PIC  X(05).
     05  HTC-INPUT-COMMON2            PIC  X(05).
   03  HTC-INPUT-DETAIL1              PIC  X(500).
   03  HTC-INPUT-DETAIL2     REDEFINES   HTC-INPUT-DETAIL1.
     05  HTC-INPUT-DETAIL2-CODE       PIC  X(20).
*>
   03  HTC-INPUT-DETAIL3     REDEFINES   HTC-INPUT-DETAIL1.
     05  HTC-INPUT-DETAIL3-CODE       PIC  X(20).
     05  HTC-INPUT-DETAIL3-COUNT      PIC S9(10)
           DISPLAY  SIGN  LEADING  SEPARATE.
     05  HTC-INPUT-DETAIL3-DATE       PIC  X(10).
     05  HTC-INPUT-DETAIL3-DATA-NO    PIC  X(20).
     05  HTC-INPUT-DETAIL3-ID-NO      PIC  X(05).
```

3. 異なるデータ属性で同一の領域を使用する場合

例については項番 1，項番 2 をご覧ください。

(24)　01 および 77 レベルの基本項目

(02-00 以降)

記述できます。

(02-00 未満)

指定したい基本項目のレベル番号を下位のレベル番号(02～49)に変更し，上位に 01 レベルの集団項目を作成してください。

[例]

＜変更前＞

```
01  HTC-OUTDATA                      PIC X(100).
```

＜変更後＞

```
01  HTC-DUMMY01.
 03  HTC-OUTDATA                    PIC X(100).
```

ただし，COBOL プログラムで引数として 01 レベル基本項目を受け取り，作業場所節の集団項目に転記後，集団項目内の基本項目で参照するような場合，作業場所節の集団項目を登録集原文として使用した方が良い場合もあります。

＜変更前＞

```
01  HTC-OUTDATA                    PIC X(100).
```

＜変更後＞

```
01  WK-OUTDATA.
  03  WK-OUTDATA1                   PIC X(8).
  03  WK-OUTDATA2                   PIC S9(5)
        SIGN LEADING SEPARATE.
   :
   :
   :
```

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

(02-02 以降)

記述できます。

(02-02 未満)

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。

(26)　OCCURS 句（OCCURS 整数 TIMES[DEPENDING ON データ名])

(02-05 以降)

OCCURS 整数 TIMES［DEPENDING ON データ名］が記述できます。DEPENDING ON 指定は繰り返し回数が可変であることを表します。

(02-05 未満)

OCCURS 整数 TIMES 以外は指定できません。

「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。

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).
```

付録 A.2　TP1/COBOL アクセス用 Bean および Servlet 作成時の注意事項/制限事項

TP1/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-3 に示します。

表 A-3　TP1/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（-），アンダーバー（_）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 SPP 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 SPP/MHP のデータ項目名にハイフン（-）が含まれる場合，自動的にアンダーバー（_）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アンダーバー（_）で生成される。
4	COBOL SPP/MHP 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。

項番	内容
6	一つの servlet，または一つの JSP にて，open～close を発行する必要がある。複数の servlet，または JSP でクライアント ID を引継ぎ，rpc コールするような使い方はできない。

付録 A.3　COBOL ユーザプログラム作成時の注事事項（ライブラリファイル名称）

COBOL ユーザ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異なる名称のファイル名称にしてください。

表 A-4　　ライブラリファイル名称一覧

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 版	j2cbpars.dll(02-00 未満) j2tppars2.dll(02-xx) j2cbwJNI.dll(02-00 未満) j2tpwJNI2.dll(02-xx)	j2tp1clt.dll j2tp1cnv.dll
HP-UX 版	該当せず	libj2tp1clt.sl libj2tp1cnv.sl
Solaris/Linux 版	該当せず	libj2tp1clt.so
AIX 版	該当せず	libj2tp1clt.a libj2tp1cnv.a

付録 A.4　Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS で使用する場合の注意事項

- バージョン 02-10 以降の開発環境を Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS で使用する場合，標準権限で実行してください。
- Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS では，JIS X0213 の第 3 水準漢字，および第 4 水準漢字を含む Unicode の文字をフォルダ名，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ます。ただし，このシステムでは，Unicode の文字列は使用できません。使用できる文字はシフト JIS の範囲だけです。
- Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS では，Windows リソース保護（WRP）によって，Windows リソース（OS ファイル，フォルダなど）が保護されるので，TP1/COBOL アクセス用 Bean 生成ツールで生成ファイルを出力するフォルダに Windows リソース保護（WRP）下を指定すると，意図しないフォルダにリダイレクトされます。Windows リソース保護（WRP）下のフォルダを指定しないでください。

付録 B　TP1/COBOL adapter for Cosminexus で使用するファイル

TP1/COBOL adapter for Cosminexus で使用するファイルの一覧を次に示します。

表 B-1　TP1/COBOL adapter for Cosminexus で使用するファイル

拡張子	ファイル種別	内容
.cbl 等※	COBOL SPP/MHP 引数定義ファイル	呼び出す COBOL SPP/MHP の引数を定義したファイル。
.class	Java 実行ファイル	TP1/COBOL adapter を呼び出す，Java 実行ファイル。
.java	TP1/COBOL アクセス用 Bean（Java ソースファイル）	生成する TP1/COBOL アクセス用 Bean を格納するファイル。
	TP1/COBOL SOAP 用サービスクラス（Java ソースファイル）	生成するスケルトンクラスを格納するファイル。
	TP1/COBOL SOAP サーバ用 Bean（Java ソースファイル）	生成する SOAP サーバ用 Bean を格納するファイル。
	TP1/COBOL SOAP クライアント用 Bean（Java ソースファイル）	生成する SOAP クライアント用 Bean を格納するファイル。
.wsdl	Web サービス記述言語ファイル	Web サービスが提供する機能を定義したファイル。

注※　.cbl/.cob/.cbf または任意の拡張子

付録 C　Windows 版の組込み方法

Windows 版の開発環境および実行環境のインストールとアンインストール方法について説明します。

付録 C.1　開発環境のインストール

開発環境には TP1/COBOL adapter for Cosminexus (Version 2)をインストールします。以下の指示に従ってください。

(1)　インストール時の注意事項

1. 前提ソフトウェアがインストールされているか，確認してください。

   また，前提ソフトウェアの Cosminexus から提供される次のソフトウェアがインストールされていることを確認してください。

   - Cosminexus Developer

2. 現在起動しているほかのアプリケーション，サービスを終了させてインストールを開始してください。
3. Windows NT4.0/Windows 2000/Windows XP/Windows Server 2003/Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS にインストールする場合は，必ずシステム管理者（administrator）の権限を持つユーザ ID を使用してください。Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS の場合は，管理者権限で起動してください。

(2)　インストール手順

TP1/COBOL adapter for Cosminexus (Version 2)のインストール手順は次のとおりです。

1. 日立総合インストーラを起動します。
2. 画面の指示に従い，インストールを進めてください。
3. 最後に「完了」のダイアログが表示されれば，インストールは完了です。

TP1/COBOL アクセス用 Bean 生成ウィザードは，TP1/COBOL adapter for Cosminexus (Version 2)インストール後に，別途組み込んでください。ただし，uCosminexus Developer 07-00 以降では，JBuilder は標準提供されていません。JBuilder をご使用になる場合のみ，COBOL アクセス用 Bean 生成ウィザードの組み込みを実施してください。

(3)　JBuilder への TP1/COBOL アクセス用 Bean 生成ウィザードの組み込み

JBuilder 上で TP1/COBOL adapter for Cosminexus (Version 2)が提供するウィザードを使用するために，JBuilder へ部品として組み込みます。

(a)　事前準備

1. JBuilder が使用する設定ファイルを作成するため，このツールを起動する前に必ず JBuilder を起動してください。
2. 組み込み時は JBuilder を終了させておいてください。

(b)　組み込み方法

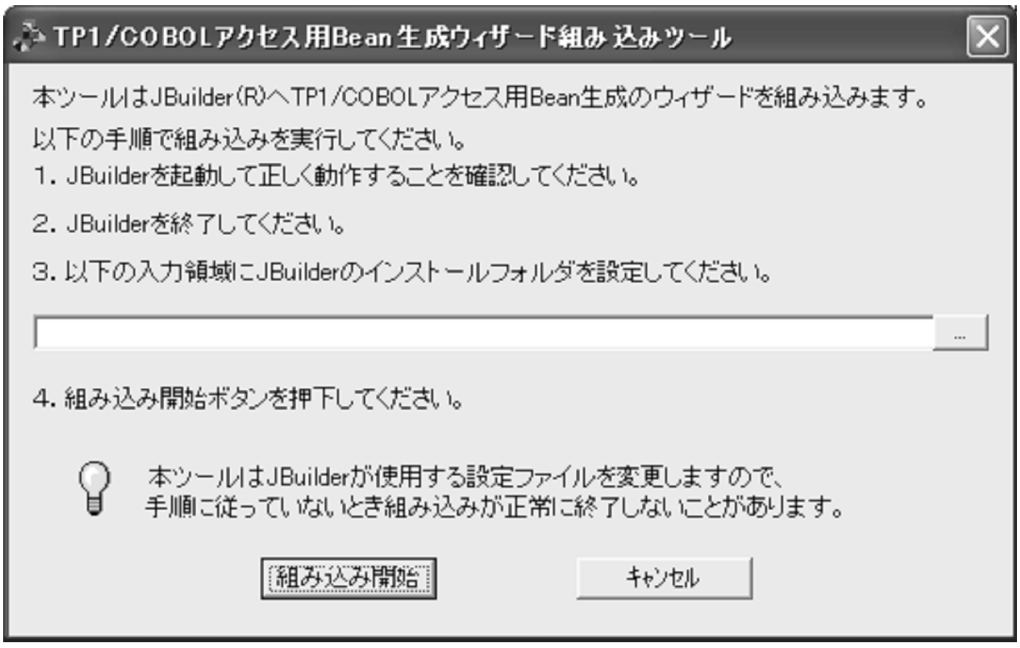
インストール先フォルダ¥BIN の J2tpWins.exe は，ウィザードを JBuilder に組み込むためにインストール後に起動するツールです。組み込み方法は次のとおりです。

1. [スタート]メニューの[プログラム]から，[COBOL2002]※の[TP1/COBOL adapter for Cosminexus]の[TP1/COBOL adapter JBuilder への部品組み込み]の順でポイントし，選択します。

   注※

   COBOL85 用にインストールした場合は，[ＣＯＢＯＬ８５]となります。

2. ダイアログが表示されますので，JBuilder のインストールフォルダの完全パスを設定してください（パスを選択する場合，右横のブラウジングボタンをご利用ください)。
3. 組み込み開始ボタンを押すと JBuilder にウィザードの組み込みを開始します。



4. JBuilder の終了を要求する警告文が表示されます。

5. 正常に JBuilder への組み込みが完了すると，次のメッセージが出力され，[OK]を押して組み込み終了です。

ただし，すでに TP1/COBOL アクセス用 Bean 生成ウィザードが組み込まれていれば，下記のメッセージが表示され組み込み処理が中止されます。

（注意事項）

TP1/COBOL アクセス用 Bean 生成ウィザードを正しく組み込むためには JBuilder インストール先フォルダ下に J2SE（Java 2 Standard Edition）のフォルダ（例：C:¥Borland ¥JBuilder2005¥JDK1.4）が存在していなければなりません。

Windows XP および Windows Server 2003 の場合，上記組み込み方法の 1.にある[プログラム]メニューは，[すべてのプログラム]メニューとなります。

付録 C.2　実行環境のインストール

実行環境には TP1/COBOL 拡張 Server Run Time System for Cosminexus (Version 2)をインストールします。以下の指示に従ってください。

(1)　インストール時の注意事項

1. 前提ソフトウェアがインストールされているか，確認してください。

   また，前提ソフトウェアの Cosminexus から提供される次のソフトウェアがインストールされていることを確認してください。

   - Cosminexus Application Server

2. 現在起動しているほかのアプリケーション，サービスを終了させてインストールを開始してください。
3. Windows NT4.0 Server/Windows 2000 Server/Windows Server 2003/Windows Server 2008 以降のサーバ OS にインストールする場合は，必ずシステム管理者（administrator）の権限を持つユー

ザ ID を使用してください。Windows Server 2008 以降のサーバ OS の場合は，管理者権限で起動してください。

(2) インストール手順

TP1/COBOL 拡張 Server Run Time System for Cosminexus のインストール手順は次のとおりです。

1. 日立総合インストーラを起動します。
2. 画面の指示に従い，インストールを進めてください。
3. 最後に「完了」のダイアログが表示されれば，インストールは完了です。

付録 C.3 開発環境のアンインストール

TP1/COBOL adapter for Cosminexus (Version 2)をアンインストールします。

(1) TP1/COBOL adapter for Cosminexus (Version 2)の削除

TP1/COBOL adapter for Cosminexus (Version 2)を削除する場合，Windows のプログラムをアンインストールする操作で次のプログラムを選択してアンインストールしてください。

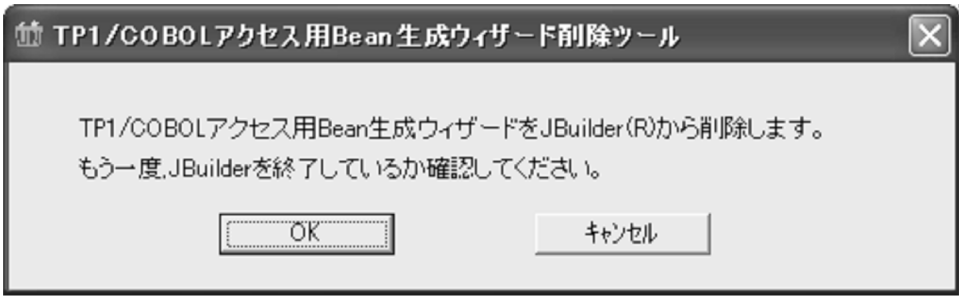
32bit 版の場合：TP1/COBOL adapter for Cosminexus (Version 2)

64bit 版の場合：TP1/COBOL adapter for Cosminexus Version 2(64)

(2) TP1/COBOL アクセス用 Bean 生成ウィザードの削除

TP1/COBOL アクセス用 Bean 生成ウィザードを JBuilder からアンインストールする手順は，次のとおりです。

1. TP1/COBOL adapter for Cosminexus (Version 2)のインストール先フォルダ¥BIN の J2tpWinsu.exe を起動します。
2. ダイアログが表示されたら，OK ボタンをクリックしてください。



(3)　アンインストール時の注意事項

アンインストーラを使用しないで，インストールしたファイルやインストールフォルダを削除したり，インストールフォルダ中のファイルを削除したりしないでください。

付録 C.4　実行環境のアンインストール

COBOL 拡張 Server Run Time System for Cosminexus (Version 2)をアンインストールします。

(1)　TP1/COBOL 拡張 Server Run Time System for Cosminexus (Version 2)の削除

TP1/COBOL 拡張 Server Run Time System for Cosminexus (Version 2)を削除する場合，Windows のプログラムをアンインストールする操作で［TP1/COBOL 拡張 Server Run Time System for Cosminexus (Version 2)］を選択してアンインストールしてください。

(2)　アンインストール時の注意事項

アンインストーラを使用しないで，インストールしたファイルやインストールフォルダを削除したり，インストールフォルダ中のファイルを削除したりしないでください。

付録 D　HP-UX/Solaris/AIX/Linux 版の組込み方法

HP-UX/Solaris/AIX/Linux 版では実行環境だけを提供していますので，ここでは TP1/COBOL 拡張 Run Time System for Cosminexus のインストールとアンインストール方法について説明します。

付録 D.1　インストール

TP1/COBOL 拡張 Run Time System for Cosminexus をインストールします。以下の指示に従ってください。

(1)　インストール時の注意事項

インストールの際には次の項目に注意してください。

1. 前提ソフトウェアがインストールされているか，確認してください。また，前提ソフトウェアの Cosminexus から提供されている次のソフトウェアがインストールされていることを確認してください。
   - Cosminexus Application Server
2. 現在起動しているほかのアプリケーションを終了させてインストールを開始してください。
3. インストールする場合は，必ずスーパユーザ（root）権限で行ってください。

(2)　インストール手順

組み込みおよび削除は，「Hitachi PP Installer」を使用します。

(a)　Hitachi PP Installer の取り出し方（CD-ROM からの起動）

次のコマンドを実行することにより，CD-ROM セットアッププログラムが「Hitachi PP　Installer」をハードディスク上にインストールし，「Hitachi PP Installer」を起動します。

HP-UX 版の場合：/cdrom/HPUX/setup /cdrom

Solaris 版の場合：/cdrom/SOLARIS/setup /cdrom

AIX 版の場合：/cdrom/AIX/setup /cdrom

Linux 版の場合：/mnt/cdrom/linux/setup /mnt/cdrom

注意

/cdrom にはご使用になる CD-ROM をマウントしたディレクトリを指定してください。なお，すでに CD-ROM セットアッププログラムを実行し，「Hitachi PP Installer」がハードディスク上にある場合は，次のように-i オプションを指定して，直接「Hitachi PP Installer」を起動してください。

```
/etc/hitachi_setup  -i /cdrom
```

(b) PP のインストール

必ず TP1/COBOL 拡張 Run Time System for Cosminexus を使用していない状態で，「Hitachi PP Installer」または CD-ROM セットアッププログラムを起動してください。

メインメニューで［I］を選択すると，次のような PP インストール画面が表示されます。

```
        PP-No.          VR          PP-Name
< > 001 P-1B36-G111     xx-xx       TP1/COBOL Extended RTS for Cosminexus

F) Forward B) Back J) Down K) Up Space) select/unselect I) Install Q) Quit
```

xx-xx:インストール対象 PP のバージョン番号

以降，指示に従いインストールを進めてください。

付録 D.2 アンインストール

TP1/COBOL 拡張 Run Time System for Cosminexus をアンインストールします。

(1) TP1/COBOL 拡張 Run Time System for Cosminexus の削除

削除するためには，必ず TP1/COBOL 拡張 Run Time System for Cosminexus を使用していない状態で，次のコマンドを入力してください。

```
/etc/hitachi_setup
```

メインメニューで［D］を選択して，以降 指示に従い削除を行ってください。

(2) アンインストール時の注意事項

アンインストーラを使用しないで，インストールしたファイルやインストールディレクトリを削除したり，インストールディレクトリ中のファイルを削除したりしないでください。

付録 E　TP1/COBOL アクセス用 Bean の自動生成ソースイメージ

TP1/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 E.1　ソースイメージの生成例（TP1/Client/P および TP1/Client/W）

「Bean 生成ツール」で「TP1/Client/P または W 経由のアクセスを使用する。」を選択した場合の生成例です。（他のチェックボックスはデフォルト指定です。）

入力用原文指定時は call/callTo/send メソッドと setter およびコンストラクタ中の setInXXX を，出力用原文指定時は receive/receive2 メソッドと getter およびコンストラクタ中の setOutXXX を生成します。

(1)　OCCURS 句がない場合

TP1/COBOL の引数に OCCURS 句がない場合の例を次に示します。

＜入力用＞

```
01 PERSONAL-DATA-IN.
 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).
```

＜出力用＞

```
01 PERSONAL-DATA-OUT.
 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(100).
```

橙色の部分は条件によって生成文字列が異なります。

ウィザード処理で別名を指定した場合，次のファイル中のデータ名は，すべて別名に置き換わります。

```
package パッケージ名;

import jp.co.hitachi_sk.j2cb.*;

public class クラス名 extends TP1RPC {
/*
 *Generated by TP1/COBOL adapter for Cosminexus Version 2 02-10
 *DO NOT EDIT THIS FILE
 *2007/09/28
 */
  private static final long serialVersionUID = 1153786728653L;
  private static boolean noload = true;
```

```
  //入力項目
  private static final int inIndex = 0;
  protected static GroupAccess myInGroup = null;
  //出力項目
  private static final int outIndex = 1;
  protected static GroupAccess myOutGroup = null;
/**
 *コンストラクタ
 */
  public クラス名() {
    super();
    //データアクセスのための情報を設定
    setInSize(204);
    setInLvl("01,05,05,05,05");
    setInName("personal_data_in,p_number,p_name,p_address,p_gif");
    setInType("G,UI0900,C(50),C(100),C(50)");
    setInAddress("0,0,4,54,154");
    setOutSize(254);
    setOutLvl("01,05,05,05,05");
    setOutName("personal_data_out,p_number,p_name,p_address,p_gif");
    setOutType("G,UI0900,C(50),C(100),C(100)");
    setOutAddress("0,0,4,54,154");
  }
/**
 *初期化処理
 */
  private void init( ) throws J2CBException {
    //初期ロードのみ実行する
    if (noload) {
      init2();
    }
    else {
      setInformation(myInGroup, myOutGroup);
    }
  }
/**
 *初期化処理2
 */
  private synchronized void init2( ) throws J2CBException {
    if (noload) {
      makeGroupAccess();
      myInGroup = inGroup;
      myOutGroup = outGroup;
      noload = false;
    }
    else {
      setInformation(myInGroup, myOutGroup);
    }
  }
/**
 *RPC
 */
  public void call( int cltid, String group, String service, int flags) throws J2CBException
 {
    init();
    super.call(cltid, group, service, flags);
  }
  public void callTo( int cltid, String hostnm, int scdport, String group, String service, i
```

```
nt flags) throws J2CBException {
    init();
    super.callTo(cltid, hostnm, scdport, group, service, flags);
  }
  public void send( int cltid, String hostname, int portnum, int flags) throws J2CBException
 {
    init();
    super.send(cltid, hostname, portnum, flags);
  }
  public void receive( int cltid, int timeout, int flags) throws J2CBException {
    init();
    super.receive(cltid, timeout, flags);
  }
  public void receive2( int cltid, int timeout, int flags) throws J2CBException {
    init();
    super.receive2(cltid, timeout, flags);
  }

  //入力項目用メソッド
/**
 *p_numberI設定メソッド
 *@param p_numberI Integerオブジェクト
 */
  public void setP_numberI( Object p_numberI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_number", p_numberI);
  }
/**
 *p_nameI設定メソッド
 *@param p_nameI Stringオブジェクト
 */
  public void setP_nameI( Object p_nameI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_name", p_nameI);
  }
/**
 *p_addressI設定メソッド
 *@param p_addressI Stringオブジェクト
 */
  public void setP_addressI( Object p_addressI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_address", p_addressI);
  }
/**
 *p_gifI設定メソッド
 *@param p_gifI Stringオブジェクト
 */
  public void setP_gifI( Object p_gifI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_gif", p_gifI);
  }
  //出力項目用メソッド
/**
 *p_number0取得メソッド
 *@return Integerオブジェクト
 */
  public Object getP_number0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_number");
  }
/**
 *p_name0取得メソッド
 *@return Stringオブジェクト
```

```
 */
  public Object getP_name0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_name");
  }
/**
 *p_address0取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_address0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_address");
  }
/**
 *p_gif0取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_gif0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_gif");
  }
}
```

(2) OCCURS 句がある場合

TP1/COBOL 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).


  //入力項目用メソッド
/**
 *b1I設定メソッド
 *@param data Stringオブジェクト
 *@param dim1 int
 */
  public void setB1I( Object data,int dim1) throws J2CBException {
    StringBuffer wkbuf;
    String acsstr1 = "g1.g2[";
    String acsstr2 = "].b1";
    wkbuf = new StringBuffer(acsstr1);
    wkbuf.append(dim1);
    wkbuf.append(acsstr2);
    setInData(inIndex, wkbuf.toString(), data);
  }
  //出力項目用メソッド
/**
 *b10取得メソッド
 *@param dim1 int
 *@return Stringオブジェクト
 */
  public Object getB10( int dim1) throws J2CBException {
    StringBuffer wkbuf;
    String acsstr1 = "g1.g2[";
    String acsstr2 = "].b1";
    wkbuf = new StringBuffer(acsstr1);
    wkbuf.append(dim1);
```

```
    wkbuf.append(acsstr2);
    return getOutData(outIndex, wkbuf.toString());
  }
```

(3) OCCURS DEPENDING ON 指定がある場合

TP1/COBOL の引数に OCCURS DEPENDING ON 指定がある場合の例を次に示します。

```
01 G1.
  02 I PIC S9(9) USAGE COMP.
  02 G2 OCCURS 10 DEPENDING ON I.
    03 B1 PIC X.


package パッケージ名;

import jp.co.hitachi_sk.j2cb.*;

public class クラス名 extends TP1RPC {
/*
 *Generated by TP1/COBOL adapter for Cosminexus Version 2 02-10
 *DO NOT EDIT THIS FILE
 *2007/09/28
 */
  private static final long serialVersionUID = 1153786735781L;
  private static boolean noload = true;
  //入力項目
  private static final int inIndex = 0;
  protected static GroupAccess myInGroup = null;
  //出力項目
  private static final int outIndex = 1;
  protected static GroupAccess myOutGroup = null;
/**
 *コンストラクタ
 */
  public クラス名() throws J2CBException {
    super();
    //データアクセスのための情報を設定
    setInSize(14);
    setInLvl("01,02,02,03");
    setInName("g1,i,g2[10],b1");
    setInType("G,SI0900,G,C(1)");
    setInAddress("0,0,4,4");
    setOutSize(14);
    setOutLvl("01,02,02,03");
    setOutName("g1,i,g2[10],b1");
    setOutType("G,SI0900,G,C(1)");
    setOutAddress("0,0,4,4");
    init();
  }
/**
 *初期化処理
 */
  private void init( ) throws J2CBException {
    //初期ロードのみ実行する
    if (noload) {
```

```
      init2();
    }
    else {
      setInformation(myInGroup, myOutGroup);
    }
  }
/**
 *初期化処理2
 */
  private synchronized void init2( ) throws J2CBException {
    if (noload) {
      makeGroupAccess();
      myInGroup = inGroup;
      myOutGroup = outGroup;
      noload = false;
    }
    else {
      setInformation(myInGroup, myOutGroup);
    }
  }
/**
 *RPC
 */
  public void call( int cltid, String group, String service, int flags) throws J2CBException
 {
    super.call(cltid, group, service, flags);
    calcTimes();
  }
  public void callTo( int cltid, String hostnm, int scdport, String group, String service, i
nt flags) throws J2CBException {
    super.callTo(cltid, hostnm, scdport, group, service, flags);
    calcTimes();
  }
  public void send( int cltid, String hostname, int portnum, int flags) throws J2CBException
 {
    super.send(cltid, hostname, portnum, flags);
  }
  public void receive( int cltid, int timeout, int flags) throws J2CBException {
    super.receive(cltid, timeout, flags);
    calcTimes();
  }
  public void receive2( int cltid, int timeout, int flags) throws J2CBException {
    super.receive2(cltid, timeout, flags);
    calcTimes();
  }
  //入力項目用メソッド
/**
 *iI設定メソッド
 *@param iI Integerオブジェクト
 */
  public void setII( Object iI) throws J2CBException {
    setInData(inIndex, "g1.i", iI);
    int[] acsInt = {2};
    InitInOcr(inIndex, acsInt, iI);
  }
/**
 *b1I設定メソッド
 *@param data Stringオブジェクト
```

```
 *@param dim1 int
 */
  public void setB1I( Object data,int dim1) throws J2CBException {
    StringBuffer wkbuf;
    String acsstr1 = "g1.g2[";
    String acsstr2 = "].b1";
    wkbuf = new StringBuffer(acsstr1);
    wkbuf.append(dim1);
    wkbuf.append(acsstr2);
    setInData(inIndex, wkbuf.toString(), data);
  }
  //出力項目用メソッド
/**
 *i0取得メソッド
 *@return Integerオブジェクト
 */
  public Object getI0( ) throws J2CBException {
    return getOutData(outIndex, "g1.i");
  }
/**
 *b10取得メソッド
 *@param dim1 int
 *@return Stringオブジェクト
 */
  public Object getB10( int dim1) throws J2CBException {
    StringBuffer wkbuf;
    String acsstr1 = "g1.g2[";
    String acsstr2 = "].b1";
    wkbuf = new StringBuffer(acsstr1);
    wkbuf.append(dim1);
    wkbuf.append(acsstr2);
    return getOutData(outIndex, wkbuf.toString());
  }
/**
 *制御変数処理メソッド
 */
  private void calcTimes( ) throws J2CBException {
    InitStart();
    int[] acsInt1 = {2};
    InitOutOcr(outIndex, acsInt1, getI0());
    InitEnd();
  }
}
```

付録 E.2　ソースイメージの生成例（TP1/Client/J）

「Bean 生成ツール」で「TP1/Client/J 経由のアクセスを使用する。」を選択した場合の生成例です。（他のチェックボックスはデフォルト指定です。）

入力用原文指定時は call/send メソッドと setter およびコンストラクタ中の setInXXX を，出力用原文指定時は receive メソッドと getter およびコンストラクタ中の setOutXXX を生成します。

(1) OCCURS 句がない場合

TP1/COBOL の引数に OCCURS 句がない場合の例を次に示します。

＜入力用＞

```
01 PERSONAL-DATA-IN.
 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).
```

＜出力用＞

```
01 PERSONAL-DATA-OUT.
 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(100).
```

橙色の部分は条件によって生成文字列が異なります。

ウィザード処理で別名を指定した場合，次のファイル中のデータ名は，すべて別名に置き換わります。

```
package パッケージ名;

import jp.co.hitachi_sk.j2cb.*;
import JP.co.Hitachi.soft.OpenTP1.*;

public class クラス名 extends TP1CLTJ {
/*
 *Generated by TP1/COBOL adapter for Cosminexus Version 2 02-10
 *DO NOT EDIT THIS FILE
 *2007/09/28
 */
  private static final long serialVersionUID = 1153786736457L;
  private static boolean noload = true;
  //入力項目
  private static final int inIndex = 0;
  protected static GroupAccess myInGroup = null;
  //出力項目
  private static final int outIndex = 1;
  protected static GroupAccess myOutGroup = null;
/**
 *コンストラクタ
 */
  public クラス名() {
    super();
    //データアクセスのための情報を設定
    setInSize(204);
    setInLvl("01,05,05,05,05");
    setInName("personal_data_in,p_number,p_name,p_address,p_gif");
    setInType("G,UI0900,C(50),C(100),C(50)");
    setInAddress("0,0,4,54,154");
    setOutSize(204);
    setOutLvl("01,05,05,05,05");
```

```
    setOutName("personal_data_in,p_number,p_name,p_address,p_gif");
    setOutType("G,UI0900,C(50),C(100),C(50)");
    setOutAddress("0,0,4,54,154");
  }
/**
 *初期化処理
 */
              :
              :
/**
 *初期化処理2
 */
              :
              :
/**
 *RPC
 */
  public void call( TP1Client cltj, String group, String service, int flags) throws Exceptio
n {
    init();
    super.call(cltj, group, service, flags);
  }
  public void send( TP1Client cltj, String hostname, int portnum, int flags) throws Exceptio
n {
    init();
    super.send(cltj, hostname, portnum, flags);
  }
  public void receive( TP1Client cltj, int timeout, int flags) throws Exception {
    init();
    super.receive(cltj, timeout, flags);
  }
              :
              :
}
```

：は「TP1/Client/P または W 経由のアクセスを使用する。」を選択した場合と同じです。詳細は，「付録 E.1　ソースイメージの生成例（TP1/Client/P および TP1/Client/W）」の「(1)　OCCURS 句がない場合」をご覧ください。

(2)　OCCURS 句がある場合

「TP1/Client/P または W 経由のアクセスを使用する。」を選択した場合と同じです。詳細は，「付録 E.1 ソースイメージの生成例（TP1/Client/P および TP1/Client/W）」の「(2)　OCCURS 句がある場合」をご覧ください。

(3)　OCCURS DEPENDING ON 指定がある場合

「TP1/Client/P または W 経由のアクセスを使用する。」を選択した場合と同じです。詳細は，「付録 E.1 ソースイメージの生成例（TP1/Client/P および TP1/Client/W）」の「(3)　OCCURS DEPENDING ON 指定がある場合」をご覧ください。

付録 E.3　ソースイメージの生成例（Cosminexus TP1 Connector）

「Bean 生成ツール」で「Cosminexus TP1 Connector 経由のアクセスを使用する。」を選択した場合の生成例です。（他のチェックボックスはデフォルト指定です。）

入力用原文指定時は call メソッドと setter およびコンストラクタ中の setInXXX を，出力用原文指定時は getter およびコンストラクタ中の setOutXXX を生成します。

(1)　OCCURS 句がない場合

TP1/COBOL の引数に OCCURS 句がない場合の例を次に示します。

＜入力用＞

```
01 PERSONAL-DATA-IN.
 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).
```

＜出力用＞

```
01 PERSONAL-DATA-OUT.
 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(100).
```

橙色の部分は条件によって生成文字列が異なります。

ウィザード処理で別名を指定した場合，次のファイル中のデータ名は，すべて別名に置き換わります。

```
package パッケージ名;

import javax.resource.cci.*;
import jp.co.hitachi_sk.j2cb.*;
import jp.co.hitachi_system.tp1connector.*;

public class クラス名 extends TP1CNTR {
/*
 *Generated by TP1/COBOL adapter for Cosminexus Version 2 02-10
 *DO NOT EDIT THIS FILE
 *2007/09/28
 */
  private static final long serialVersionUID = 1153786795146L;
  private static boolean noload = true;
  //入力項目
  private static final int inIndex = 0;
  protected static GroupAccess myInGroup = null;
  //出力項目
  private static final int outIndex = 1;
  protected static GroupAccess myOutGroup = null;
/**
 *コンストラクタ
```

```
 */
  public クラス名() {
    super();
    //データアクセスのための情報を設定
    setInSize(204);
    setInLvl("01,05,05,05,05");
    setInName("personal_data_in,p_number,p_name,p_address,p_gif");
    setInType("G,UI0900,C(50),C(100),C(50)");
    setInAddress("0,0,4,54,154");
    setOutSize(204);
    setOutLvl("01,05,05,05,05");
    setOutName("personal_data_out,p_number,p_name,p_address,p_gif");
    setOutType("G,UI0900,C(50),C(100),C(100)");
    setOutAddress("0,0,4,54,154");
  }
/**
 *初期化処理
 */
              :
              :
/**
 *初期化処理2
 */
              :
              :
/**
 *RPC
 */
  public void call( ConnectionFactory cxf, Interaction ix, InteractionSpecImpl ixSpec) throw
s Exception {
    init();
    super.call(cxf, ix, ixSpec);
  }
              :
              :
}
```

:は「TP1/Client/P または W 経由のアクセスを使用する。」を選択した場合と同じです。詳細は,「付録 E.1　ソースイメージの生成例（TP1/Client/P および TP1/Client/W）」の「(1)　OCCURS 句がない場合」をご覧ください。

(2)　OCCURS 句がある場合

「TP1/Client/P または W 経由のアクセスを使用する。」を選択した場合と同じです。詳細は,「付録 E.1 ソースイメージの生成例（TP1/Client/P および TP1/Client/W）」の「(2)　OCCURS 句がある場合」をご覧ください。

(3)　OCCURS DEPENDING ON 指定がある場合

「TP1/Client/P または W 経由のアクセスを使用する。」を選択した場合と同じです。詳細は,「付録 E.1 ソースイメージの生成例（TP1/Client/P および TP1/Client/W）」の「(3)　OCCURS DEPENDING ON 指定がある場合」をご覧ください。

付録 E.4　TP1/COBOL SOAP サービスクラス生成機能ソースイメージの生成例

(1)　SOAP サーバ上の Java UAP を呼び出すスケルトンクラスの生成例

「TP1/COBOL SOAP サーバ用クラス生成ウィザード」を使用して自動生成したスケルトンクラスの例を次に示します。

```
package パッケージ名;

import javax.xml.rpc.holders.*;

public class クラス名 {
/*
 *Generated by TP1/COBOL adapter for Cosminexus Version 2 02-08
 *DO NOT EDIT THIS FILE
 *2005/07/06
 */
  // 入力引数最大長
  public static final int inMaxSize  = 204;
  // 出力引数最大長
  public static final int outMaxSize = 254;

  public void メソッド名(byte[] inData,
    ByteArrayHolder outData) throws Exception {
    try{
      呼び出すJavaプログラムのクラス名 uap = new 呼び出すJavaプログラムのクラス名();
      outData.value = uap.呼び出すJavaプログラムのメソッド名(inData);
    }catch(Exception e) {
      e.printStackTrace();
      throw e;
    }
  }
}
```

注：橙色の部分はウィザードで指定した名前になります。

スケルトンクラスの引数は，inData および outData で固定の名称となります。

(2)　TP1/COBOL SOAP サーバ用 Bean の生成例

「TP1/COBOL SOAP サーバ用クラス生成ウィザード」を使用して自動生成した Bean の例を次に示します。

```
package パッケージ名;

import jp.co.hitachi_sk.j2cb.*;

public class クラス名SBean extends TP1SOAPSV {
/*
 *Generated by TP1/COBOL adapter for Cosminexus Version 2 02-08
 *DO NOT EDIT THIS FILE
```

```
 *2005/07/06
 */
  private static boolean noload = true;
  //入力項目
  protected static final int inIndex = 0;
  protected static GroupAccess myInGroup = null;
  //出力項目
  protected static final int outIndex = 1;
  protected static GroupAccess myOutGroup = null;
/**
 *コンストラクタ
 */
  public クラス名SBean() throws J2CBException {
    super();
    //データアクセスのための情報を設定
    setInSize(204);
    setInLvl("01,05,05,05,05");
    setInName("personal_data_in,p_number,p_name,p_address,p_gif");
    setInType("G,UI0900,C(50),C(100),C(50)");
    setInAddress("0,0,4,54,154");
    setOutSize(254);
    setOutLvl("01,05,05,05,05");
    setOutName("personal_data_out,p_number,p_name,p_address,p_gif");
    setOutType("G,UI0900,C(50),C(100),C(100)");
    setOutAddress("0,0,4,54,154");
    init();
  }
/**
 *初期化処理
 */
  private void init( ) throws J2CBException {
    //初期ロードのみ実行する
    if (noload) {
      init2();
    }
    else {
      setInformation(myInGroup, myOutGroup);
    }
  }
/**
 *初期化処理2
 */
  private synchronized void init2( ) throws J2CBException {
    if (noload) {
      makeGroupAccess();
      myInGroup = inGroup;
      myOutGroup = outGroup;
      noload = false;
    }
    else {
      setInformation(myInGroup, myOutGroup);
    }
  }
  //入力項目用メソッド
/**
 *p_numberI設定メソッド
 *@param p_numberI Integerオブジェクト
 *@return void
```

```
 */
  public void setP_numberI( Object p_numberI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_number", p_numberI);
  }
/**
 *p_nameI設定メソッド
 *@param p_nameI Stringオブジェクト
 *@return void
 */
  public void setP_nameI( Object p_nameI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_name", p_nameI);
  }
/**
 *p_addressI設定メソッド
 *@param p_addressI Stringオブジェクト
 *@return void
 */
  public void setP_addressI( Object p_addressI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_address", p_addressI);
  }
/**
 *p_gifI設定メソッド
 *@param p_gifI Stringオブジェクト
 *@return void
 */
  public void setP_gifI( Object p_gifI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_gif", p_gifI);
  }
/**
 *p_numberI取得メソッド
 *@return Integerオブジェクト
 */
  public Object getP_numberI( ) throws J2CBException {
    return getOutData(inIndex, "personal_data_in.p_number");
  }
/**
 *p_nameI取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_nameI( ) throws J2CBException {
    return getOutData(inIndex, "personal_data_in.p_name");
  }
/**
 *p_addressI取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_addressI( ) throws J2CBException {
    return getOutData(inIndex, "personal_data_in.p_address");
  }
/**
 *p_gifI取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_gifI( ) throws J2CBException {
    return getOutData(inIndex, "personal_data_in.p_gif");
  }
  //出力項目用メソッド
/**
```

```
 *p_number0設定メソッド
 *@param p_number0 Integerオブジェクト
 *@return void
 */
  public void setP_number0( Object p_number0) throws J2CBException {
    setInData(outIndex, "personal_data_out.p_number", p_number0);
  }
/**
 *p_name0設定メソッド
 *@param p_name0 Stringオブジェクト
 *@return void
 */
  public void setP_name0( Object p_name0) throws J2CBException {
    setInData(outIndex, "personal_data_out.p_name", p_name0);
  }
/**
 *p_address0設定メソッド
 *@param p_address0 Stringオブジェクト
 *@return void
 */
  public void setP_address0( Object p_address0) throws J2CBException {
    setInData(outIndex, "personal_data_out.p_address", p_address0);
  }
/**
 *p_gif0設定メソッド
 *@param p_gif0 Stringオブジェクト
 *@return void
 */
  public void setP_gif0( Object p_gif0) throws J2CBException {
    setInData(outIndex, "personal_data_out.p_gif", p_gif0);
  }
/**
 *p_number0取得メソッド
 *@return Integerオブジェクト
 */
  public Object getP_number0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_number");
  }
/**
 *p_name0取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_name0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_name");
  }
/**
 *p_address0取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_address0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_address");
  }
/**
 *p_gif0取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_gif0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_gif");
```

```
  }
}
```

注：橙色の部分はウィザードで指定した名前になります。

(3) TP1/COBOL SOAP クライアント用 Bean の生成例

「TP1/COBOL SOAP クライアント用 Bean 生成ウィザード」を使用して自動生成した Bean の例を次に示します。

```
package パッケージ名;

import jp.co.hitachi_sk.j2cb.*;

public class クラス名CBean extends TP1SOAPCLT {
/*
 *Generated by TP1/COBOL adapter for Cosminexus Version 2 02-08
 *DO NOT EDIT THIS FILE
 *2005/07/06
 */
  private static boolean noload = true;
  //入力項目
  protected static final int inIndex = 0;
  protected static GroupAccess myInGroup = null;
  //出力項目
  protected static final int outIndex = 1;
  protected static GroupAccess myOutGroup = null;
/**
 *コンストラクタ
 */
  public クラス名CBean() throws J2CBException {
    super();
    //データアクセスのための情報を設定
    setInSize(204);
    setInLvl("01,05,05,05,05");
    setInName("personal_data_in,p_number,p_name,p_address,p_gif");
    setInType("G,UI0900,C(50),C(100),C(50)");
    setInAddress("0,0,4,54,154");
    setOutSize(254);
    setOutLvl("01,05,05,05,05");
    setOutName("personal_data_out,p_number,p_name,p_address,p_gif");
    setOutType("G,UI0900,C(50),C(100),C(100)");
    setOutAddress("0,0,4,54,154");
    init();
  }
/**
 *初期化処理
 */
  private void init( ) throws J2CBException {
    //初期ロードのみ実行する
    if (noload) {
      init2();
    }
    else {
      setInformation(myInGroup, myOutGroup);
```

```
    }
  }
/**
 *初期化処理2
 */
  private synchronized void init2( ) throws J2CBException {
    if (noload) {
      makeGroupAccess();
      myInGroup = inGroup;
      myOutGroup = outGroup;
      noload = false;
    }
    else {
      setInformation(myInGroup, myOutGroup);
    }
  }
  //入力項目用メソッド
/**
 *p_numberI設定メソッド
 *@param p_numberI Integerオブジェクト
 *@return void
 */
  public void setP_numberI( Object p_numberI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_number", p_numberI);
  }
/**
 *p_nameI設定メソッド
 *@param p_nameI Stringオブジェクト
 *@return void
 */
  public void setP_nameI( Object p_nameI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_name", p_nameI);
  }
/**
 *p_addressI設定メソッド
 *@param p_addressI Stringオブジェクト
 *@return void
 */
  public void setP_addressI( Object p_addressI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_address", p_addressI);
  }
/**
 *p_gifI設定メソッド
 *@param p_gifI Stringオブジェクト
 *@return void
 */
  public void setP_gifI( Object p_gifI) throws J2CBException {
    setInData(inIndex, "personal_data_in.p_gif", p_gifI);
  }
  //出力項目用メソッド
/**
 *p_number0取得メソッド
 *@return Integerオブジェクト
 */
  public Object getP_number0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_number");
  }
/**
```

```
 *p_name0取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_name0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_name");
  }
/**
 *p_address0取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_address0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_address");
  }
/**
 *p_gif0取得メソッド
 *@return Stringオブジェクト
 */
  public Object getP_gif0( ) throws J2CBException {
    return getOutData(outIndex, "personal_data_out.p_gif");
  }
}
```

付録 F　TP1/COBOL アクセス用 Bean 生成時に出力するメッセージ

この節では，TP1/COBOL adapter for Cosminexus が出力するメッセージ内容について記載します。

付録 F.1　メッセージの形式

```
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：COBOL adapter for Cosminexus または TP1/COBOL adapter for Cosminexus 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，以下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 F.2　メッセージ一覧

(1)　TP1/COBOL アクセス用 Bean 生成ウィザード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
KCCC2001J-E	*** 1 ***の構文解析処理でエラーが発生しました。 予期しないエラーが発生しました。
KCCC2002J-E	*** 1 ***の構文解析処理でエラーが発生しました。 予期しない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
KCCC2003J-E	*** 1 ***の構文解析処理でエラーが発生しました。 メモリ不足が発生しました。
KCCC2004J-E	*** 1 ***の構文解析処理でエラーが発生しました。 I/O エラーが発生しました。
KCCC2005J-E	*** 1 ***の構文解析処理でエラーが発生しました。 メモリ不足が発生しました。
KCCC2006J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文の解析処理で予期しないエラーが発生しました。
KCCC2007J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC2008J-E	*** 1 ***の構文解析処理でエラーが発生しました。 指定された通貨記号が不当です。
KCCC2009J-E	*** 1 ***の構文解析処理でエラーが発生しました。 I/O エラーが発生しました。
KCCC2011J-E	*** 1 ***の構文解析処理でエラーが発生しました。 本製品で許していない語が現れました。 または書き方に誤りがあります。
KCCC2012J-E	*** 1 ***の構文解析処理でエラーが発生しました。 1 行の文字数が 255 文字を超えています。
KCCC2013J-E	*** 1 ***の構文解析処理でエラーが発生しました。 入力行数が 32,760 行を超えました。
KCCC2014J-E	*** 1 ***の構文解析処理でエラーが発生しました。 FILLER 項目の数が 65,535 個を超えました。
KCCC2015J-E	*** 1 ***の構文解析処理でエラーが発生しました。 語，または PICTURE 文字列の長さが 30 字を超えています。
KCCC2016J-E	*** 1 ***の構文解析処理でエラーが発生しました。 数字の桁数が 18 桁を超えています。
KCCC2017J-E	*** 1 ***の構文解析処理でエラーが発生しました。 終止符の直後には空白が必要です。
KCCC2018J-E	*** 1 ***の構文解析処理でエラーが発生しました。 最初のデータ定義のレベル番号が 01/77 以外です。
KCCC2019J-E	*** 1 ***の構文解析処理でエラーが発生しました。 レベル番号は先に出た基本項目の属する集団のうちの どれかのレベル番号と同じでなければなりません。
KCCC2020J-E	*** 1 ***の構文解析処理でエラーが発生しました。 レベル番号の指定に誤りがあります。

メッセージ ID	メッセージテキスト
KCCC2021J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句が 2 重に定義されています。
KCCC2022J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句の書き方に誤りがあります。
KCCC2023J-E	*** 1 ***の構文解析処理でエラーが発生しました。 USAGE 句が 2 重に定義されています。
KCCC2024J-E	*** 1 ***の構文解析処理でエラーが発生しました。 OCCURS 句が 2 重に定義されています。
KCCC2025J-E	*** 1 ***の構文解析処理でエラーが発生しました。 OCCURS 句に指定した反復回数の値が正しくありません。
KCCC2026J-E	*** 1 ***の構文解析処理でエラーが発生しました。 SIGN 句が 2 重に定義されています。
KCCC2027J-E	*** 1 ***の構文解析処理でエラーが発生しました。 REDEFINES 句が 2 重に定義されています。
KCCC2028J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC2029J-E	*** 1 ***の構文解析処理でエラーが発生しました。 集団項目に PICTURE 句を指定することはできません。
KCCC2030J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句と USAGE 句の指定が矛盾しています。
KCCC2031J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句あるいは USAGE 句と SIGN 句の指定が矛盾しています。
KCCC2032J-E	*** 1 ***の構文解析処理でエラーが発生しました。 基本項目に USAGE 句も PICTURE 句も指定されていません。
KCCC2033J-E	*** 1 ***の構文解析処理でエラーが発生しました。 01/77 レベル以外に USAGE ADDRESS/POINTER は指定できません。
KCCC2034J-E	*** 1 ***の構文解析処理でエラーが発生しました。 USAGE POINTER/ADDRESS は指定できません。
KCCC2035J-E	*** 1 ***の構文解析処理でエラーが発生しました。 日本語文字が 72 カラムと 73 カラムにまたがっています。
KCCC2036J-E	*** 1 ***の構文解析処理でエラーが発生しました。 7 カラム目に不当な文字が指定されています。
KCCC2041J-E	*** 1 ***の構文解析処理でエラーが発生しました。 プログラム論理エラーが発生しました。
KCCC2042J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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
KCCC2043J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC2044J-E	*** 1 ***の構文解析処理でエラーが発生しました。 パーサエンジンがエラー終了しました。
KCCC2051J-E	*** 1 ***の構文解析処理でエラーが発生しました。 登録集原文ファイルが開けません。
KCCC2052J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文の解析処理で I/O エラーが発生しました。
KCCC2053J-E	*** 1 ***の構文解析処理でエラーが発生しました。 1 行の文字数が 255 文字を超えています。
KCCC2054J-E	*** 1 ***の構文解析処理でエラーが発生しました。 入力行数が 32,760 行を超えました。
KCCC2055J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文のネストレベルが 19 を超えました。
KCCC2056J-E	*** 1 ***の構文解析処理でエラーが発生しました。 登録集原文ファイルが開けません。
KCCC2057J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文の解析処理で I/O エラーが発生しました。
KCCC2058J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC2059J-E	*** 1 ***の構文解析処理でエラーが発生しました。 原文名に誤りがあります。
KCCC2060J-E	*** 1 ***の構文解析処理でエラーが発生しました。 登録集原文ファイルが見つかりません。
KCCC2061J-E	*** 1 ***の構文解析処理でエラーが発生しました。 フリー形式の原文と固定形式の原文は混在できません。
KCCC2071J-E	REDEFINES 句がある項目，またはそれに従属する項目には， OCCURS DEPENDING ON 句を指定できません。
KCCC2072J-E	REDEFINES 句の作用対象，またはそれに従属する項目には， OCCURS DEPENDING ON 句を指定できません。
KCCC2099J-E	*** 1 ***の構文解析処理でエラーが発生しました。 解析処理中に例外が発生しました。
KCCC2101J-W	*** 1 ***の構文解析処理で警告エラーが発生しました。 REDEFINES 句がデータ名の後ろにありません。 直前にあるものと仮定します。

メッセージ ID	メッセージテキスト
KCCC2102J-W	*** 1 ***の構文解析処理で警告エラーが発生しました。 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。 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。 REDEFINES 句を無視します。
KCCC2103J-W	*** 1 ***の構文解析処理で警告エラーが発生しました。 再定義項目と被再定義項目の大きさを同じにするか，または再定義項目を小さくする必要があります。 再定義項目の大きさを採用します。
KCCC2301J-E	TEMP フォルダを作成できません。
KCCC2302J-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。
KCCC2311J-E	COBOL へ渡す登録集原文のファイルの指定がありません。
KCCC2312J-E	COBOL へ渡す登録集原文のファイルが見つかりません。
KCCC2313J-E	COBOL へ渡す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。
KCCC2322J-E	COBOL から受け取る登録集原文のファイルが 見つかりません。
KCCC2323J-E	COBOL から受け取る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。
KCCC2308J-E	j2tppars2.dll をローディングできません。
KCCC2331J-E	COBOL へ渡す登録集原文のファイル中に複数の集団項目の指定はできません。
KCCC2341J-E	COBOL から受け取る登録集原文のファイル中に複数の集団項目の指定はできません。
KCCC2351J-E	登録集原文のファイルの指定がありません。
KCCC2401J-E	入力の別名に Java(TM)の変数名規則として不当な文字が含まれています。 文字: "*** 3 ***"
KCCC2403J-E	入力の別名(*** 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。別名を再設定してください。 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？
KCCC2405J-E	入力のデータ名(*** 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。別名を設定してください。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？
KCCC2406J-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。 別名を設定してください。
KCCC2407J-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。 別名を再設定してください。
KCCC2408J-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。 別名を設定してください。

メッセージ ID	メッセージテキスト
KCCC2409J-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。 別名を再設定してください。
KCCC2421J-E	出力の別名に Java(TM)の変数名規則として不当な文字が含まれています。 文字: "*** 3 ***"
KCCC2423J-E	出力の別名(*** 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。別名を再設定してください。 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？
KCCC2425J-E	出力のデータ名(*** 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。別名を設定してください。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？
KCCC2426J-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。 別名を設定してください。
KCCC2427J-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。 別名を再設定してください。
KCCC2428J-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。 別名を設定してください。
KCCC2429J-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。 別名を再設定してください。
KCCC2431J-E	制御変数(*** 7 ***)は，OCCURS 句が指定されたデータ項目を含む引数内に定義しなければなりません。
KCCC2432J-E	制御変数(*** 7 ***)は，その制御変数を指定した可変反復データ項目のあとに定義してはなりません。
KCCC2433J-E	OCCURS 句がある項目，またはそれに従属する項目(*** 7 ***)は，制御変数に指定できません。
KCCC2434J-E	制御変数(*** 7 ***)は整数項目でなければなりません。
KCCC2435J-E	制御変数(*** 7 ***)は，可変反復データ項目のあとに定義してはなりません。
KCCC2471J-E	パッケージ名の指定がありません。
KCCC2472J-E	パッケージ名として正しくありません。
KCCC2473J-E	クラス名の指定がありません。
KCCC2474J-E	クラス名として正しくありません。
KCCC2501J-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？ファイル名：*** 4 ***
KCCC2502J-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。
KCCC2503J-E	プロジェクトへのソース追加初期処理で エラーが発生しました。
KCCC2504J-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。
KCCC2505J-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。

メッセージ ID	メッセージテキスト
KCCC2506J-E	クラスを生成できませんでした。
KCCC2601J-E	解析処理中に例外が発生しました。
KCCC2602J-E	TP1/COBOL アクセス用 Bean 生成処理が異常終了しました。
KCCC2651J-E	テーブルの初期化処理中に例外が発生しました。
KCCC2661J-E	メモリ不足が発生しました。

(2) TP1/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
KCCC3001J-E	*** 1 ***の構文解析処理でエラーが発生しました。 予期しないエラーが発生しました。
KCCC3002J-E	*** 1 ***の構文解析処理でエラーが発生しました。 予期しないエラーが発生しました。
KCCC3003J-E	*** 1 ***の構文解析処理でエラーが発生しました。 メモリ不足が発生しました。
KCCC3004J-E	*** 1 ***の構文解析処理でエラーが発生しました。 I/O エラーが発生しました。
KCCC3005J-E	*** 1 ***の構文解析処理でエラーが発生しました。 メモリ不足が発生しました。
KCCC3006J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文の解析処理で予期しないエラーが発生しました。
KCCC3007J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC3008J-E	*** 1 ***の構文解析処理でエラーが発生しました。 指定された通貨記号が不当です。
KCCC3009J-E	*** 1 ***の構文解析処理でエラーが発生しました。 I/O エラーが発生しました。
KCCC3011J-E	*** 1 ***の構文解析処理でエラーが発生しました。 本製品で許していない語が現れました。 または書き方に誤りがあります。
KCCC3012J-E	*** 1 ***の構文解析処理でエラーが発生しました。 1 行の文字数が 255 文字を超えています。
KCCC3013J-E	*** 1 ***の構文解析処理でエラーが発生しました。 入力行数が 32,760 行を超えました。
KCCC3014J-E	*** 1 ***の構文解析処理でエラーが発生しました。 FILLER 項目の数が 65,535 個を超えました。

メッセージ ID	メッセージテキスト
KCCC3015J-E	*** 1 ***の構文解析処理でエラーが発生しました。 語，または PICTURE 文字列の長さが 30 字を超えています。
KCCC3016J-E	*** 1 ***の構文解析処理でエラーが発生しました。 数字の桁数が 18 桁を超えています。
KCCC3017J-E	*** 1 ***の構文解析処理でエラーが発生しました。 終止符の直後には空白が必要です。
KCCC3018J-E	*** 1 ***の構文解析処理でエラーが発生しました。 最初のデータ定義のレベル番号が 01/77 以外です。
KCCC3019J-E	*** 1 ***の構文解析処理でエラーが発生しました。 レベル番号は先に出た基本項目の属する集団のうちの どれかのレベル番号と同じでなければなりません。
KCCC3020J-E	*** 1 ***の構文解析処理でエラーが発生しました。 レベル番号の指定に誤りがあります。
KCCC3021J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句が 2 重に定義されています。
KCCC3022J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句の書き方に誤りがあります。
KCCC3023J-E	*** 1 ***の構文解析処理でエラーが発生しました。 USAGE 句が 2 重に定義されています。
KCCC3024J-E	*** 1 ***の構文解析処理でエラーが発生しました。 OCCURS 句が 2 重に定義されています。
KCCC3025J-E	*** 1 ***の構文解析処理でエラーが発生しました。 OCCURS 句に指定した反復回数の値が正しくありません。
KCCC3026J-E	*** 1 ***の構文解析処理でエラーが発生しました。 SIGN 句が 2 重に定義されています。
KCCC3027J-E	*** 1 ***の構文解析処理でエラーが発生しました。 REDEFINES 句が 2 重に定義されています。
KCCC3028J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC3029J-E	*** 1 ***の構文解析処理でエラーが発生しました。 集団項目に PICTURE 句を指定することはできません。
KCCC3030J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句と USAGE 句の指定が矛盾しています。
KCCC3031J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句あるいは USAGE 句と SIGN 句の指定が矛盾しています。

メッセージ ID	メッセージテキスト
KCCC3032J-E	*** 1 ***の構文解析処理でエラーが発生しました。 基本項目に USAGE 句も PICTURE 句も指定されていません。
KCCC3033J-E	*** 1 ***の構文解析処理でエラーが発生しました。 01/77 レベル以外に USAGE ADDRESS/POINTER は指定できません。
KCCC3034J-E	*** 1 ***の構文解析処理でエラーが発生しました。 USAGE POINTER/ADDRESS は指定できません。
KCCC3035J-E	*** 1 ***の構文解析処理でエラーが発生しました。 日本語文字が 72 カラムと 73 カラムにまたがっています。
KCCC3036J-E	*** 1 ***の構文解析処理でエラーが発生しました。 7 カラム目に不当な文字が指定されています。
KCCC3041J-E	*** 1 ***の構文解析処理でエラーが発生しました。 プログラム論理エラーが発生しました。
KCCC3042J-E	*** 1 ***の構文解析処理でエラーが発生しました。 パーサエンジンがスタックオーバーフローしました。
KCCC3043J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC3044J-E	*** 1 ***の構文解析処理でエラーが発生しました。 パーサエンジンがエラー終了しました。
KCCC3051J-E	*** 1 ***の構文解析処理でエラーが発生しました。 登録集原文ファイルが開けません。
KCCC3052J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文の解析処理で I/O エラーが発生しました。
KCCC3053J-E	*** 1 ***の構文解析処理でエラーが発生しました。 1 行の文字数が 255 文字を超えています。
KCCC3054J-E	*** 1 ***の構文解析処理でエラーが発生しました。 入力行数が 32,760 行を超えました。
KCCC3055J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文のネストレベルが 19 を超えました。
KCCC3056J-E	*** 1 ***の構文解析処理でエラーが発生しました。 登録集原文ファイルが開けません。
KCCC3057J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文の解析処理で I/O エラーが発生しました。
KCCC3058J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。

メッセージ ID	メッセージテキスト
KCCC3059J-E	*** 1 ***の構文解析処理でエラーが発生しました。 原文名に誤りがあります。
KCCC3060J-E	*** 1 ***の構文解析処理でエラーが発生しました。 登録集原文ファイルが見つかりません。
KCCC3061J-E	*** 1 ***の構文解析処理でエラーが発生しました。 フリー形式の原文と固定形式の原文は混在できません。
KCCC3071J-E	REDEFINES 句がある項目，またはそれに従属する項目には， OCCURS DEPENDING ON 句を指定できません。
KCCC3072J-E	REDEFINES 句の作用対象，またはそれに従属する項目には， OCCURS DEPENDING ON 句を指定できません。
KCCC3099J-E	*** 1 ***の構文解析処理でエラーが発生しました。 解析処理中に例外が発生しました。
KCCC3101J-W	*** 1 ***の構文解析処理で警告エラーが発生しました。 REDEFINES 句がデータ名の後ろにありません。 直前にあるものと仮定します。
KCCC3102J-W	*** 1 ***の構文解析処理で警告エラーが発生しました。 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。 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。 REDEFINES 句を無視します。
KCCC3103J-W	*** 1 ***の構文解析処理で警告エラーが発生しました。 再定義項目と被再定義項目の大きさを同じにするか， または再定義項目を小さくする必要があります。 再定義項目の大きさを採用します。
KCCC3301J-E	TEMP フォルダを作成できません。
KCCC3302J-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。
KCCC3303J-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。
KCCC3305J-W	フォルダ(*** 5 ***)が見つかりません。作成しますか？
KCCC3306J-E	フォルダを作成できませんでした。
KCCC3307J-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。
KCCC3308J-E	j2tppars2.dll をローディングできません。
KCCC3311J-E	COBOL へ渡す登録集原文のファイルの指定がありません。
KCCC3312J-E	COBOL へ渡す登録集原文のファイルが見つかりません。
KCCC3313J-E	COBOL へ渡す登録集原文に指定された値は

メッセージ ID	メッセージテキスト
	ファイル名ではありません。
KCCC3322J-E	COBOL から受け取る登録集原文の ファイルが見つかりません。
KCCC3323J-E	COBOL から受け取る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。
KCCC3331J-E	COBOL へ渡す登録集原文のファイル中に 複数の集団項目の指定はできません。
KCCC3341J-E	COBOL から受け取る登録集原文のファイル中に 複数の集団項目の指定はできません。
KCCC3351J-E	登録集原文のファイルの指定がありません。
KCCC3391J-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？
KCCC3401J-E	入力の別名に Java(TM)の変数名規則として 不当な文字が含まれています。文字: "*** 3 ***"
KCCC3403J-E	入力の別名(*** 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。別名を再設定してください。 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？
KCCC3405J-E	入力のデータ名(*** 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。別名を設定してください。 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？
KCCC3406J-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。 別名を設定してください。
KCCC3407J-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。 別名を再設定してください。
KCCC3408J-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。 別名を設定してください。
KCCC3409J-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。 別名を再設定してください。
KCCC3421J-E	出力の別名に Java(TM)の変数名規則として不当な文字が含まれています。 文字: "*** 3 ***"
KCCC3423J-E	出力の別名(*** 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。別名を再設定してください。 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？
KCCC3425J-E	出力のデータ名(*** 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。別名を設定してください。 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？

メッセージ ID	メッセージテキスト
KCCC3426J-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。 別名を設定してください。
KCCC3427J-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。 別名を再設定してください。
KCCC3428J-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。 別名を設定してください。
KCCC3429J-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。 別名を再設定してください。
KCCC3431J-E	制御変数(*** 7 ***)は，OCCURS 句が指定されたデータ項目を含む引数内に定義しなければなりません。
KCCC3432J-E	制御変数(*** 7 ***)は，その制御変数を指定した可変反復データ項目のあとに定義してはなりません。
KCCC3433J-E	OCCURS 句がある項目，またはそれに従属する項目(*** 7 ***)は，制御変数に指定できません。
KCCC3434J-E	制御変数(*** 7 ***)は整数項目でなければなりません。
KCCC3435J-E	制御変数(*** 7 ***)は，可変反復データ項目のあとに定義してはなりません。
KCCC3471J-E	パッケージ名の指定がありません。
KCCC3472J-E	パッケージ名として正しくありません。
KCCC3473J-E	クラス名の指定がありません。
KCCC3474J-E	クラス名として正しくありません。
KCCC3501J-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？ファイル名：*** 4 ***
KCCC3502J-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。
KCCC3504J-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。
KCCC3505J-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。
KCCC3506J-E	クラスを生成できませんでした。
KCCC3601J-E	解析処理中に例外が発生しました。
KCCC3602J-E	TP1/COBOL アクセス用 Bean 生成処理が異常終了しました。
KCCC3651J-E	テーブルの初期化処理中に例外が発生しました。
KCCC3661J-E	メモリ不足が発生しました。
KCCC3801J-E	2 重起動しました。終了します。
KCCC3802J-E	内部エラー：*** 6 ***
KCCC3811J-E	内部エラー：*** 6 ***
KCCC3812J-E	フォルダとして有効でありません。

メッセージ ID	メッセージテキスト
KCCC3813J-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。
KCCC3814J-E	設定したフォルダは不当です。再設定してください。
KCCC3815J-E	ドライブ指定が必要です。再設定してください。
KCCC3816J-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。
KCCC3817J-E	J2SE(TM)のルートフォルダのパスを正しく設定してください。
KCCC3818J-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。
KCCC3819J-E	J2SE(TM)のルートフォルダを設定してください。
KCCC3820J-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。
KCCC3821J-E	最大メモリプールサイズの値が指定されていません。
KCCC3822J-E	設定ファイルの出力処理でエラーが発生しました。
KCCC3831J-E	初期化処理中にエラーが発生しました。
KCCC3832J-E	J2SE(TM)のルートフォルダが見つかりません。 ルートフォルダを再設定してください。
KCCC3833J-E	生成ツールを起動できませんでした。 ファイルが見つかりません。 J2SE(TM)のルートフォルダを再設定してください。
KCCC3834J-E	生成ツールを起動できませんでした。 ファイルにアクセスできません。 J2SE(TM)のルートフォルダ下のアクセス権を見直してください。
KCCC3835J-E	生成ツールを起動できませんでした。メモリ不足です。
KCCC3836J-E	生成ツールを起動できませんでした。*** 6 ***
KCCC3837J-E	生成ツール実行時にエラーが発生しました。 実行環境を見直してください。
KCCC3881J-E	初期処理時にエラーが発生しました。
KCCC3841J-E	2 重起動しました。終了します。
KCCC3842J-E	内部エラー：*** 6 ***
KCCC3843J-E	初期処理時にエラーが発生しました。
KCCC3851J-E	ヘルプを起動できません。*** 6 ***
KCCC3852J-E	ヘルプファイルが見つかりません。
KCCC3853J-E	ヘルプ起動時にエラーが発生しました。
KCCC3854J-E	ヘルプ起動時に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
KCCC3855J-E	内部エラー：*** 6 ***
KCCC3861J-E	終了処理時にエラーが発生しました。
KCCC3862J-E	終了処理時にエラーが発生しました。
KCCC3871J-E	j2tpwJNI2.dll をローディングできません。
KCCC3872J-E	ヘルプ起動時にエラーが発生しました。
KCCC3873J-E	終了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？

(3) TP1/COBOL アクセス用 Bean 生成ウィザード組み込み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
KCCC2901J-I	JBuilder(R)への組み込み処理が終了しました。
KCCC2902J-I	JBuilder(R)からの削除処理が終了しました。
KCCC2903J-E	TP1/COBOL アクセス用 Bean 生成ウィザードは 既に組み込まれています。
KCCC2904J-E	TP1/COBOL アクセス用 Bean 生成ウィザードは 組み込まれていません。
KCCC2905J-E	組み込み処理中に例外が発生しました。
KCCC2906J-E	削除処理中に例外が発生しました。
KCCC2907J-E	組み込み情報ファイルが見つかりませんでした。
KCCC2908J-E	組み込み情報ファイルの入力処理でエラーが発生しました。
KCCC2909J-E	ファイルのコピーに失敗しました。*** 6 ***
KCCC2910J-E	ライブラリファイルの出力処理でエラーが発生しました。
KCCC2921J-E	2 重起動しました。終了します。
KCCC2922J-E	内部エラー：*** 6 ***
KCCC2931J-E	フォルダとして有効でありません。
KCCC2932J-I	もう一度,JBuilder(R)を終了しているか確認してください。
KCCC2933J-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。
KCCC2934J-E	設定したインストールフォルダは不当です。 再設定してください。
KCCC2935J-E	ドライブ指定が必要です。再設定してください。

メッセージ ID	メッセージテキスト
KCCC2936J-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。
KCCC2937J-E	設定した値は組み込み処理上使用できません。 再設定してください。
KCCC2938J-E	インストールフォルダを設定してください。
KCCC2939J-E	組み込み情報ファイルが見つかりません。*** 4 ***
KCCC2940J-E	組み込み情報ファイル作成に失敗しました。
KCCC2941J-E	J2SE(TM)のインストールフォルダが見つかりません。
KCCC2942J-E	組み込みプロセスを起動できませんでした。
KCCC2943J-E	組み込み処理中にエラーが発生しました。
KCCC2944J-E	組み込み処理中にエラーが発生しました。
KCCC2945J-E	アンインストール情報ファイル： *** 4 *** の入出力処理でエラーが発生しました。
KCCC2946J-E	組み込み処理の入出力処理でエラーが発生しました。
KCCC2951J-E	2 重起動しました。終了します。
KCCC2952J-E	内部エラー：*** 6 ***
KCCC2961J-E	組み込み情報ファイルが見つかりません。*** 4 ***
KCCC2962J-E	*** 4 *** の入出力処理でエラーが発生しました。
KCCC2963J-E	JBuilder(R)インストールフォルダの取得に失敗しました。
KCCC2964J-E	組み込み情報ファイル作成に失敗しました。
KCCC2965J-E	J2SE(TM)のインストールフォルダが見つかりません。
KCCC2966J-E	組み込みプロセスを起動できませんでした。
KCCC2967J-E	JBuilder(R)からの削除処理中にエラーが発生しました。
KCCC2968J-E	JBuilder(R)からの削除処理中にエラーが発生しました。
KCCC2969J-E	削除処理の入出力処理でエラーが発生しました。
KCCC4001J-E	内部エラー：*** 6 ***

(4) TP1/COBOL SOAP サーバ用クラス生成ウィザード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
KCCC9001J-E	*** 1 ***の構文解析処理でエラーが発生しました。 予期しない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
KCCC9002J-E	*** 1 ***の構文解析処理でエラーが発生しました。 予期しないエラーが発生しました。
KCCC9003J-E	*** 1 ***の構文解析処理でエラーが発生しました。 メモリ不足が発生しました。
KCCC9004J-E	*** 1 ***の構文解析処理でエラーが発生しました。 I/O エラーが発生しました。
KCCC9005J-E	*** 1 ***の構文解析処理でエラーが発生しました。 メモリ不足が発生しました。
KCCC9006J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文の解析処理で予期しないエラーが発生しました。
KCCC9007J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC9008J-E	*** 1 ***の構文解析処理でエラーが発生しました。 指定された通貨記号が不当です。
KCCC9009J-E	*** 1 ***の構文解析処理でエラーが発生しました。 I/O エラーが発生しました。
KCCC9011J-E	*** 1 ***の構文解析処理でエラーが発生しました。 本製品で許していない語が現れました。 または書き方に誤りがあります。
KCCC9012J-E	*** 1 ***の構文解析処理でエラーが発生しました。 1 行の文字数が 255 文字を超えています。
KCCC9013J-E	*** 1 ***の構文解析処理でエラーが発生しました。 入力行数が 32,760 行を超えました。
KCCC9014J-E	*** 1 ***の構文解析処理でエラーが発生しました。 FILLER 項目の数が 65,535 個を超えました。
KCCC9015J-E	*** 1 ***の構文解析処理でエラーが発生しました。 語，または PICTURE 文字列の長さが 30 字を超えています。
KCCC9016J-E	*** 1 ***の構文解析処理でエラーが発生しました。 数字の桁数が 18 桁を超えています。
KCCC9017J-E	*** 1 ***の構文解析処理でエラーが発生しました。 終止符の直後には空白が必要です。
KCCC9018J-E	*** 1 ***の構文解析処理でエラーが発生しました。 最初のデータ定義のレベル番号が 01/77 以外です。
KCCC9019J-E	*** 1 ***の構文解析処理でエラーが発生しました。 レベル番号は先に出た基本項目の属する集団のうちのどれかのレベル番号と同じでなければなりません。

メッセージ ID	メッセージテキスト
KCCC9020J-E	*** 1 ***の構文解析処理でエラーが発生しました。 レベル番号の指定に誤りがあります。
KCCC9021J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句が 2 重に定義されています。
KCCC9022J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句の書き方に誤りがあります。
KCCC9023J-E	*** 1 ***の構文解析処理でエラーが発生しました。 USAGE 句が 2 重に定義されています。
KCCC9024J-E	*** 1 ***の構文解析処理でエラーが発生しました。 OCCURS 句が 2 重に定義されています。
KCCC9025J-E	*** 1 ***の構文解析処理でエラーが発生しました。 OCCURS 句に指定した反復回数の値が正しくありません。
KCCC9026J-E	*** 1 ***の構文解析処理でエラーが発生しました。 SIGN 句が 2 重に定義されています。
KCCC9027J-E	*** 1 ***の構文解析処理でエラーが発生しました。 REDEFINES 句が 2 重に定義されています。
KCCC9028J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC9029J-E	*** 1 ***の構文解析処理でエラーが発生しました。 集団項目に PICTURE 句を指定することはできません。
KCCC9030J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句と USAGE 句の指定が矛盾しています。
KCCC9031J-E	*** 1 ***の構文解析処理でエラーが発生しました。 PICTURE 句あるいは USAGE 句と SIGN 句の指定が矛盾しています。
KCCC9032J-E	*** 1 ***の構文解析処理でエラーが発生しました。 基本項目に USAGE 句も PICTURE 句も指定されていません。
KCCC9033J-E	*** 1 ***の構文解析処理でエラーが発生しました。 01/77 レベル以外に USAGE ADDRESS/POINTER は指定できません。
KCCC9034J-E	*** 1 ***の構文解析処理でエラーが発生しました。 USAGE POINTER/ADDRESS は指定できません。
KCCC9035J-E	*** 1 ***の構文解析処理でエラーが発生しました。 日本語文字が 72 カラムと 73 カラムにまたがっています。
KCCC9036J-E	*** 1 ***の構文解析処理でエラーが発生しました。 7 カラム目に不当な文字が指定されています。
KCCC9041J-E	*** 1 ***の構文解析処理でエラーが発生しました。 プログラム論理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
KCCC9042J-E	*** 1 ***の構文解析処理でエラーが発生しました。 パーサエンジンがスタックオーバーフローしました。
KCCC9043J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC9044J-E	*** 1 ***の構文解析処理でエラーが発生しました。 パーサエンジンがエラー終了しました。
KCCC9051J-E	*** 1 ***の構文解析処理でエラーが発生しました。 登録集原文ファイルが開けません。
KCCC9052J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文の解析処理で I/O エラーが発生しました。
KCCC9053J-E	*** 1 ***の構文解析処理でエラーが発生しました。 1 行の文字数が 255 文字を超えています。
KCCC9054J-E	*** 1 ***の構文解析処理でエラーが発生しました。 入力行数が 32,760 行を超えました。
KCCC9055J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文のネストレベルが 19 を超えました。
KCCC9056J-E	*** 1 ***の構文解析処理でエラーが発生しました。 登録集原文ファイルが開けません。
KCCC9057J-E	*** 1 ***の構文解析処理でエラーが発生しました。 COPY 文の解析処理で I/O エラーが発生しました。
KCCC9058J-E	*** 1 ***の構文解析処理でエラーが発生しました。 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。
KCCC9059J-E	*** 1 ***の構文解析処理でエラーが発生しました。 原文名に誤りがあります。
KCCC9060J-E	*** 1 ***の構文解析処理でエラーが発生しました。 登録集原文ファイルが見つかりません。
KCCC9061J-E	*** 1 ***の構文解析処理でエラーが発生しました。 フリー形式の原文と固定形式の原文は混在できません。
KCCC9071J-E	REDEFINES 句がある項目，またはそれに従属する項目には，OCCURS DEPENDING ON 句を指定できません。
KCCC9072J-E	REDEFINES 句の作用対象，またはそれに従属する項目には，OCCURS DEPENDING ON 句を指定できません。
KCCC9099J-E	*** 1 ***の構文解析処理でエラーが発生しました。 解析処理中に例外が発生しました。
KCCC9101J-W	*** 1 ***の構文解析処理で警告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
	REDEFINES 句がデータ名の後ろにありません。 直前にあるものと仮定します。
KCCC9102J-W	*** 1 ***の構文解析処理で警告エラーが発生しました。 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。 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。 REDEFINES 句を無視します。
KCCC9103J-W	*** 1 ***の構文解析処理で警告エラーが発生しました。 再定義項目と被再定義項目の大きさを同じにするか，または再定義項目を小さくする必要があります。 再定義項目の大きさを採用します。
KCCC9301J-E	TEMP フォルダを作成できません。
KCCC9302J-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。
KCCC9312J-E	COBOL へ渡す登録集原文のファイルが見つかりません。
KCCC9313J-E	COBOL へ渡す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。
KCCC9322J-E	COBOL から受け取る登録集原文のファイルが見つかりません。
KCCC9323J-E	COBOL から受け取る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。
KCCC9308J-E	j2tppars2.dll をローディングできません。
KCCC9331J-E	COBOL へ渡す登録集原文のファイル中に複数の集団項目の指定はできません。
KCCC9341J-E	COBOL から受け取る登録集原文のファイル中に複数の集団項目の指定はできません。
KCCC9351J-E	登録集原文のファイルの指定がありません。
KCCC9401J-E	入力の別名に Java(TM)の変数名規則として不当な文字が含まれています。 文字: "*** 3 ***"
KCCC9403J-E	入力の別名(*** 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。別名を再設定してください。 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？
KCCC9405J-E	入力のデータ名(*** 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。別名を設定してください。 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？
KCCC9406J-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。 別名を設定してください。
KCCC9407J-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。 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
KCCC9408J-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。 別名を設定してください。
KCCC9409J-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。 別名を再設定してください。
KCCC9421J-E	出力の別名に Java(TM)の変数名規則として不当な文字が含まれています。 文字: "*** 3 ***"
KCCC9423J-E	出力の別名(*** 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。別名を再設定してください。 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？
KCCC9425J-E	出力のデータ名(*** 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。別名を設定してください。 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？
KCCC9426J-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。 別名を設定してください。
KCCC9427J-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。 別名を再設定してください。
KCCC9428J-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。 別名を設定してください。
KCCC9429J-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。 別名を再設定してください。
KCCC9431J-E	制御変数(*** 7 ***)は，OCCURS 句が指定されたデータ項目を含む引数内に定義しなければなりません。
KCCC9432J-E	制御変数(*** 7 ***)は，その制御変数を指定した可変反復データ項目のあとに定義してはなりません。
KCCC9433J-E	OCCURS 句がある項目，またはそれに従属する項目(*** 7 ***)は，制御変数に指定できません。
KCCC9434J-E	制御変数(*** 7 ***)は整数項目でなければなりません。
KCCC9435J-E	制御変数(*** 7 ***)は，可変反復データ項目のあとに定義してはなりません。
KCCC9471J-E	パッケージ名の指定がありません。
KCCC9472J-E	パッケージ名として正しくありません。
KCCC9473J-E	クラス名の指定がありません。
KCCC9474J-E	クラス名として正しくありません。
KCCC9475J-E	メソッド名の指定がありません。
KCCC9476J-E	メソッド名として正しくありません。

メッセージ ID	メッセージテキスト
KCCC9477J-E	呼び出す Java(TM)プログラムのクラス名の指定がありません。
KCCC9478J-E	呼び出す Java(TM)プログラムのクラス名が正しくありません。
KCCC9479J-E	クラス名と呼び出す Java(TM)プログラムのクラス名に同じ名称は指定できません。
KCCC9480J-E	呼び出す Java(TM)プログラムのメソッド名の指定がありません。
KCCC9481J-E	呼び出す Java(TM)プログラムのメソッド名が正しくありません。
KCCC9501J-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？ファイル名：*** 4 ***
KCCC9502J-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。
KCCC9503J-E	プロジェクトへのソース追加初期処理でエラーが発生しました。
KCCC9504J-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。
KCCC9505J-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。
KCCC9506J-E	クラスを生成できませんでした。
KCCC9510J-E	I/O エラーが発生しました。
KCCC9601J-E	解析処理中に例外が発生しました。
KCCC9603J-E	TP1/COBOL SOAP サーバ用クラス生成処理が異常終了しました。
KCCC9651J-E	テーブルの初期化処理中に例外が発生しました。

(5)　TP1/COBOL SOAP 用 WSDL 編集ウィザード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
KCCC9901J-E	TP1/COBOL SOAP サーバ用クラスを生成してください。
KCCC9902J-E	WSDL ファイル名の指定がありません。
KCCC9903J-E	WSDL ファイルが見つかりません。
KCCC9904J-E	WSDL ファイルに指定された値はファイル名ではありません。
KCCC9905J-I	WSDL ファイルの更新が完了しました。
KCCC9909J-E	TP1/COBOL SOAP 用 WSDL 更新処理が異常終了しました。
KCCC9910J-E	WSDL ファイルはすでに更新されています。
KCCC9911J-E	WSDL ファイルの更新処理中に例外が発生しました。
KCCC9651J-E	テーブルの初期化処理中に例外が発生しました。

(6) TP1/COBOL SOAP クライアント用 Bean 生成ウィザード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
KCCC9951J-E	WSDL ファイル名の指定がありません。
KCCC9952J-E	WSDL ファイルが見つかりません。
KCCC9953J-E	WSDL ファイルに指定された値はファイル名ではありません。
KCCC9954J-E	WSDL ファイルの解析処理中に例外が発生しました。
KCCC9955J-E	パッケージ名の指定がありません。
KCCC9956J-E	パッケージ名として正しくありません。
KCCC9957J-E	クラス名の指定がありません。
KCCC9958J-E	クラス名として正しくありません。
KCCC9959J-E	TP1/COBOL SOAP クライアント用 Bean 生成処理が異常終了しました。
KCCC9961J-E	メモリ不足が発生しました。
KCCC9971J-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？ファイル名：*** 4 ***
KCCC9972J-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。
KCCC9973J-E	プロジェクトへのソース追加初期処理でエラーが発生しました。
KCCC9974J-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。
KCCC9975J-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。
KCCC9976J-E	クラスを生成できませんでした。
KCCC9981J-E	WSDL ファイルを更新してください。
KCCC9982J-E	WSDL 解析処理で I/O エラーが発生しました。
KCCC9983J-E	WSDL 解析処理中にエラーが発生しました。 WSDL ファイルが壊れています。再作成してください。

※表中の *** n *** は，以下を表します。

*** 1 ***："COBOL へ渡す登録集原文"または"COBOL から受け取る登録集原文"

*** 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

*** 7 ***：制御変数名または制御変数名の別名

付録 G　例外情報コード一覧

この節では，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記載します。

付録 G.1　例外情報コードの形式

getErrorCode で取得する例外情報コードは，以下の形式です。

```
J2CByyynnnn
```

J2CB　プリフィクス

yyy：エラー発生場所（メソッド）コード：表 G-1 を参照

nnnn：エラー要因コード：表 G-2 を参照

表 G-1　エラー発生場所（メソッド）コード一覧

コード（16 進）	場所
001	getData
010	setData
050	init0
201	cltin
202	cltout
203	connect
204	disconnect
205	setConnectInf
206	call
207	close
208	getWatchTime
209	open
20A	setWatchTime
20B	begin
20C	chainedCommit
20D	chainedRollback
20E	getTrnid

コード（16 進）	場所
20F	info
210	unchainedCommit
211	unchainedRollback
212	receive(基本 Bean)
213	receive2(基本 Bean)
214	send(基本 Bean)
215	acceptNotification
216	cancelNotification
217	makeInit
218	codeConvertToKEIS
219	codeConvertToSJIS
220	callTo
221	receive(アクセス用 Bean)
222	receive2(アクセス用 Bean)
223	send(アクセス用 Bean)
224	getRaphost
225	setRaphost
226	openNotification
227	closeNotification
228	chainedAcceptNotification
230	codeConvLibToKEIS
231	codeConvLibToSJIS
232	codeConvLibOpen
233	codeConvLibClose
500	GroupAccess
701	TP1RPC.getOutData
702	TP1RPC.setInData
703	TP1RPC.InitInOcr
704	TP1RPC.InitOutOcr
710	TP1CLTJ.getOutData
711	TP1CLTJ.setInData

コード（16 進）	場所
712	TP1CLTJ.InitInOcr
713	TP1CLTJ.InitOutOcr
720	TP1CNTR.getOutData
721	TP1CNTR.setInData
722	TP1CNTR.InitInOcr
723	TP1CNTR.InitOutOcr
732	TP1SOAPSV.InitInOcr
733	TP1SOAPSV.InitOutOcr
742	TP1SOAPCLT.InitInOcr
743	TP1SOAPCLT.InitOutOcr
800	getEnv
801	getOption
999	不明

表 G-2　例外名一覧

エラー要因コードの候補	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

付録 G.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 G.1 を参照

YYY：エラー発生場所（メソッド）名称（表 G-1 参照）

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 G.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 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 コンテナを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 G.4　メッセージ一覧

J2CByyy0001

（Windows/HP-UX/AIX/Linux 版）
TP1/Client でエラーが発生しました。
TP1/Client のリターン値を元に，原因調査してください。
プログラム名称=*** 1 ***
（Solaris 版）
外部モジュールでエラーが発生しました。*** 1 ***

要因

TP1/Client でエラーが発生した。
*** 1 ***：TP1/Client のリターン値

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

出力され TP1/Client のリターン値を元に，エラー原因を修正して再実行する。

J2CByyy0002

（Windows/HP-UX/AIX/Linux 版）
予期しないエラーが発生しました。[コード値=*** 9 ***]
（Solaris 版）
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。

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

（Windows/HP-UX/AIX/Linux 版）
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

COBOL アクセス実行環境変数

TP1/COBOL ユーザインタフェース API

文字コード変換

(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。

J2CByyy0100

コード変換ライブラリがローディングできませんでした。

要因

コード変換ライブラリ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)

コード変換ライブラリをインストールし，環境を見直して，再実行する。

J2CByyy0101

コード変換処理中にエラーが発生しました。Code=*** 1 ***

要因

コード変換処理中にエラーが発生しました。

*** 1 *** : コード変換ライブラリ関数(CodeConvString)の戻り値

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

*** 1 ***を元に，エラー原因を修正して再実行する。

J2CByyy0102

コード変換ライブラリでエラーが発生しました。コード変換ライブラリのリターン値をもとに，原因調査してください。Code=*** 1 ***[,System=*** 2 ***] (*** 3 ***)

要因

コード変換ライブラリ呼び出し中にエラーが発生した。

*** 1 *** : コード変換ライブラリ関数の戻り値

*** 2 *** : システムの値

*** 3 *** : コード変換ライブラリの関数名

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

エラー情報を元に，エラー原因を修正して再実行する。

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

（Windows/HP-UX/AIX/Linux 版）
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。
（Solaris 版）
符号なしの項目ですが，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

TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートしていないデータ項目属性を，TP1/COBOL アクセス用 Bean で指定した。考えられる要因は以下のどちらかである。

1. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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 TP1/COBOL アクセス用 Bean を実行した。
2. 生成した TP1/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. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートされていないデータ項目を取り除いて TP1/COBOL アクセス用 Bean を作成し，再実行する。または，TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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. TP1/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

TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートしていないデータ項目属性を，TP1/COBOL アクセス用 Bean で指定した。考えられる要因は以下のどちらかである。

1. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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 TP1/COBOL アクセス用 Bean を実行した。
2. 生成した TP1/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. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートされていないデータ項目を取り除いて TP1/COBOL アクセス用 Bean を作成し，再実行する。または，TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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. TP1/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

TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートしていないデータ項目属性を，TP1/COBOL アクセス用 Bean で指定した。考えられる要因は以下のどちらかである。

1. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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 TP1/COBOL アクセス用 Bean を実行した。
2. 生成した TP1/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. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートされていないデータ項目を取り除いて TP1/COBOL アクセス用 Bean を作成し，再実行する。または，TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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. TP1/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

TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートしていないデータ項目属性を，TP1/COBOL アクセス用 Bean で指定した。考えられる要因は以下のどちらかである。

1. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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 TP1/COBOL アクセス用 Bean を実行した。
2. 生成した TP1/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. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートされていないデータ項目を取り除いて TP1/COBOL アクセス用 Bean を作成し，再実行する。または，TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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. TP1/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

TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートしていないデータ項目属性を，TP1/COBOL アクセス用 Bean で指定した。考えられる要因は以下のどちらかである。

1. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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 TP1/COBOL アクセス用 Bean を実行した。

2. 生成した TP1/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. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートされていないデータ項目を取り除いて TP1/COBOL アクセス用 Bean を作成し，再実行する。または，TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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. TP1/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

TP1/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。考えられる要因は以下のどちらかである。

1. 指定した添字の値が範囲外である。
2. 生成した TP1/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. TP1/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，TP1/COBOL adapter for Cosminexus [Version 2] で生成した TP1/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

TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートしていないデータ項目属性を，TP1/COBOL アクセス用 Bean で指定した。考えられる要因は以下のどちらかである。

1. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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 TP1/COBOL アクセス用 Bean を実行した。
2. 生成した TP1/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. TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートされていないデータ項目を取り除いて TP1/COBOL アクセス用 Bean を作成し，再実行する。または，TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] 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. TP1/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

TP1/COBOL 拡張 [Server] Run Time System for Cosminexus [Version 2] でサポートしていないデータ項目属性を，TP1/COBOL アクセス用 Bean で指定した。生成した TP1/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)

TP1/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，TP1/COBOL adapter for Cosminexus [Version 2] で生成した TP1/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)

TP1/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。TP1/COBOL adapter for Cosminexus [Version 2] で生成した TP1/COBOL アクセス用 Bean でもこのエラーとなる場合は，TP1/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

TP1/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 TP1/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)

TP1/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，TP1/COBOL adapter for Cosminexus [Version 2] で生成した TP1/COBOL アクセス用 Bean で Java プログラムを作成し，再実行する。TP1/COBOL adapter for Cosminexus [Version 2] で生成した TP1/COBOL アクセス用 Bean でもこのエラーとなる場合は，TP1/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(TM)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(TM)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」で設定された数値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」で数値データを設定する個所を調査・修正して，再実行する。

J2CByyy2014

（Windows/HP-UX/AIX/Linux 版）
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。

J2CByyy2015

（Windows/HP-UX/AIX/Linux 版）
OCCURS DEPENDING ON 制御変数に指定した値が不当です。最大繰り返し回数を仮定します。
指定値=*** 1 ***

要因

OCCURS DEPENDING ON 制御変数に不当な値を指定した。
指定できる値は，0 <= 指定値 <= 最大繰り返し回数である。
*** 1 ***：指定した値

(S)

最大繰り返し回数を仮定し，例外を発生させる。

(P)

制御変数に指定する値を見直して Java プログラムを作成し，再実行する。

J2CByyy2016

（Windows/HP-UX/AIX/Linux 版）
引数番号に指定した値が不当です。引数番号=*** 1 ***

要因

setGroupItemLenFollowSetData，setByteData および GetByteData メソッドの引数に指定した，引数番号が不当である。
*** 1 ***：指定した値

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

引数番号に指定する値を見直して Java プログラムを作成し，再実行する。

J2CByyy2017

（Windows/HP-UX/AIX/Linux 版）
設定または取得するデータ領域が存在しません。要因情報=*** 1 ***

要因

setByteData で設定した領域を超えて，設定または取得しようとした。

*** 1 ***：要因情報

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

setByteData メソッドで設定する領域を見直して Java プログラムを作成し，再実行する。

J2CByyy2018

（Windows/HP-UX/AIX/Linux 版）
OCCURS DEPENDING ON 制御変数の値が取得できません。最大繰り返し回数を仮定します。

要因

制御変数の値を取得しようとしたが，設定された領域を超えているため，取得できなかった。

(S)

最大繰り返し回数を仮定し，例外を発生させる。

(P)

出力引数の設定値を見直すか，setByteData メソッドで設定する領域を見直して Java プログラムを作成し，再実行する。

J2CByyy2019

（Windows/HP-UX/AIX/Linux 版）
バイト配列のデータ領域が未設定です。

要因

setByteData メソッドでバイト配列データを設定せずに，データを設定および参照しようとした。

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

setByteData メソッドでバイト配列データを設定して Java プログラムを作成し，再実行する。

J2CByyy2051

有効データ領域外のため，データが取得できませんでした。

要因

「COBOL プログラム(SPP/MHP)で出力情報サイズを設定している」場合で，指定された出力情報サイズにまったく含まれないデータを取得しようとした。

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

データが取得できなくてもよい（動作として正しい）のであれば，そのまま処理を続行する。データが取得できなければならないのであれば，取得しようとするデータが出力情報に含まれる（COBOL プログラム(SPP/MHP)で設定した出力情報サイズを大きくする）ように修正して，再実行する。

J2CByyy2052

文字データ以外でデータの一部が有効データ領域外のため，データが取得できませんでした。

要因

「COBOL プログラム(SPP/MHP)で出力情報サイズを設定している」場合で，指定された出力情報サイズに一部分含まれないデータを取得しようとした。(文字データを除く)

(データの開始位置が出力情報サイズに指定された有効データ領域内である点が，J2CByyy2051 と異なる)

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

データが取得できなくてもよい（動作として正しい）のであれば，そのまま処理を続行する。データが取得できなければならないのであれば，取得しようとするデータが出力情報に含まれる（COBOL プログラム(SPP/MHP)で設定した出力情報サイズを大きくする）ように修正して，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（数字を設定すべき個所が数字でない）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，以下の形式である。

(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(TM)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

（Windows/HP-UX/AIX/Linux 版）
Java が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。要因として，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

（Windows/HP-UX/AIX/Linux 版）
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

（Windows/HP-UX/AIX/Linux 版）
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

（Windows/HP-UX/AIX/Linux 版）
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

（Windows/HP-UX/AIX/Linux 版）
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 ***

要因

引数情報出力ファイル作成時にエラーが発生した。

*** 1 ***：例外情報

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

例外情報を元にファイルの書き込み権限等を見直して，再実行する。

J2CByyy5003

（Windows/HP-UX/AIX/Linux 版）
引数情報出力処理中に I/O エラーが発生しました。*** 1 ***

要因

引数情報出力処理中に例外が発生した。

*** 1 ***：例外情報

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

例外情報を元にエラー原因を修正して，再実行する。

J2CByyy5004

（Windows/HP-UX/AIX/Linux 版）
引数情報出力処理中にエラーが発生しました。*** 1 ***

要因

引数情報出力処理中に例外が発生した。

*** 1 ***：例外情報

(S)

プログラムの実行を中止し，例外を発生させる。

(P)

例外情報を元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。「COBOL 拡張 [Server] Run Time System for Cosminexus Version 2」と「TP1/COBOL 拡張 [Server] Run Time System for Cosminexus Version 2」が一つの PC 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 H　プログラム例

TP1/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 E　TP1/COBOL アクセス用 Bean の自動生成ソースイメージ」をご覧ください）。

付録 H.1　TP1/Client/P および TP1/Client/W 版

(1)　COBOL 引数の登録集原文と COBOL プログラム例

```
入力用COBOL引数の登録集原文例
       01 PERSONAL-DATA-IN.
         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).
出力用COBOL引数の登録集原文例
       01 PERSONAL-DATA-OUT.
         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(100).
COBOLプログラム例
      *>************************************************
      *>    COBOL SPP(Search)                           *
      *>************************************************
       IDENTIFICATION  DIVISION.
       PROGRAM-ID.  SEARCHTP1.
       DATA            DIVISION.
       LINKAGE SECTION.
         COPY SEARCHTP1COPYIN.
      *> 01 PERSONAL-DATA-IN.
      *>   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).
       77 IN-LEN PIC S9(9) USAGE COMP.
         COPY SEARCHTP1COPYOUT.
      *> 01 PERSONAL-DATA-OUT.
      *>   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(100).
       77 OUT-LEN PIC S9(9) USAGE COMP.

       PROCEDURE DIVISION USING PERSONAL-DATA-IN IN-LEN
                                PERSONAL-DATA-OUT OUT-LEN.
           MOVE P-NUMBER IN PERSONAL-DATA-IN TO P-NUMBER IN
                                        PERSONAL-DATA-OUT.
      *>  検索処理
            IF P-NUMBER IN PERSONAL-DATA-IN = 100001 THEN
              MOVE '日立 一郎' TO P-NAME IN PERSONAL-DATA-OUT
```

```
            MOVE '日立市' TO P-ADDRESS IN PERSONAL-DATA-OUT
            MOVE '/ICHIRO.GIF' TO P-GIF IN PERSONAL-DATA-OUT
          ELSE
            IF P-NUMBER IN PERSONAL-DATA-IN = 100002 THEN
              MOVE '日立 二郎' TO P-NAME IN PERSONAL-DATA-OUT
              MOVE '久留米市' TO P-ADDRESS IN PERSONAL-DATA-OUT
              MOVE '/JIRO.GIF' TO P-GIF IN PERSONAL-DATA-OUT
            ELSE
              MOVE '登録されていません' TO P-NAME IN
                                          PERSONAL-DATA-OUT
              MOVE SPACE TO P-ADDRESS IN PERSONAL-DATA-OUT
              MOVE '/NOREGIST.GIF' TO P-GIF IN PERSONAL-DATA-OUT
            END-IF
          END-IF.
          EXIT  PROGRAM.

       END PROGRAM  SEARCHTP1.
```

(2)　Java UAP(Servlet)例

```
package test;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.beans.Beans;
import jp.co.hitachi_sk.j2cb.*;
//数字項目(外部10進項目，内部10進項目および小数けたを含む2進項目)を
//使用する場合は，以下のようにjava.math.BigDecimalをimportしてください。
//import java.math.BigDecimal;

public class SearchTP1Servlet 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");

    SearchTP1 bean = null;
    TP1Access baseBean = null;
      try {
```

```
        //Beanのインスタンス生成
        bean = (SearchTP1) Beans.instantiate(this.getClass().
                        getClassLoader(),"test.SearchTP1");
        baseBean = (TP1Access) Beans.instantiate(
                        this.getClass().getClassLoader(),
                       "jp.co.hitachi_sk.j2cb.TP1Access");
      } catch (ClassNotFoundException excp) {
        excp.printStackTrace();
        return;
      }

      try {
        int[] idArr = new int[1];
        String[] host = new String[8];
        baseBean.cltin(idArr,null, "Hostname", "guest",null,
                                                host, 0);
        baseBean.open(idArr[0], 0);

        /* Beanにパラメタを設定 */
        bean.setP_numberI(new Integer(number));
        bean.setP_nameI("");
        bean.setP_addressI("");
        bean.setP_gifI("");
        try {
          bean.call(idArr[0], "SAMPLE", "SEARCHTP1", 0);
        } catch (Exception e) {
          baseBean.close(idArr[0], 0);
          baseBean.cltout(idArr[0], 0);
          e.printStackTrace();
          return;
        }

        baseBean.close(idArr[0], 0);
        baseBean.cltout(idArr[0], 0);

        //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("/SearchTP1.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>
SearchTP1
</TITLE>
</HEAD>
<BODY>
<h3><center>TP1/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.SearchTP1Servlet method="POST">
Number : <input type="text" name="Number" value="100001">
<BR><BR> Submit を押すと servlet SearchTP1Servlet を実行
<BR><BR><input type=submit value="Submit"></form>
</center>
</BODY>
</HTML>
```

(4)　JSP の作成例

```
<!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.SearchTP1" />
<html>
  <center>Result of Search.
    <table border="1">
      <tr><td colspan="2" align="center">
        <jsp:getProperty name="bean" property="p_number0" />
      </td><tr><td>Name</td><td>
        <jsp:getProperty name="bean" property="p_name0" />
      </td><tr><td>Address</td><td>
        <jsp:getProperty name="bean" property="p_address0" />
      <tr><td colspan="2" align="center">
       <img src="../servlet<jsp:getProperty name="bean" property="p_gif0" />">
      </td></table>
  </center>
</html>
```

付録 H.2　TP1/Client/J 版

(1)　COBOL 引数の登録集原文と COBOL プログラム例

付録 H.1 の TP1/Client/P および TP1/Client/W 版と同じです。「付録 H.1　TP1/Client/P および TP1/Client/W 版」をご覧ください。

(2)　Java UAP(Servlet)例

(a)　Rap Server 使用時の概要

```
                :
import  jp.co.hitachi_sk.j2cb.*;   // パッケージのインポート
import  JP.co.Hitachi.soft.OpenTP1.*;   //
                :
TP1Client tp1 = new TP1Client();        // インスタンス生成
生成クラス名  cls = new 生成クラス名(); //
                :
tp1.openConnection("server",12000);     // RAPサーバに接続
                :
cls.setData1I("");                      // データの設定
                :
cls.call(tp1,"group","service",tp1.DCNOFLAGS);
                                        // RPC実行
                :
String str = cls.getData1O();           // データの取得
                :
tp1.closeConnection();                 // サーバとの接続解消
                :
```

(b)　scd サーバ使用時の概要（スケジューラダイレクト機能）

```
                :
import  jp.co.hitachi_sk.j2cb.*;   // パッケージのインポート
import  JP.co.Hitachi.soft.OpenTP1.*;   //
                :
TP1Client tp1 = new TP1Client();        // インスタンス生成
生成クラス名  cls = new 生成クラス名(); //
                :
tp1.rpcOpen();                          // RPC環境の初期化
                :
cls.setData1I("");                      // データの設定
                :
cls.call(tp1,"group","service",tp1.DCNOFLAGS);
                                        // RPC実行
                :
String str = cls.getData1O();           // データの取得
                :
tp1.rpcClose();                         // RPC環境の開放
                :
```

(c)　servlet 作成例

```
package test;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.beans.Beans;
import jp.co.hitachi_sk.j2cb.*;
import JP.co.Hitachi.soft.OpenTP1.*;
```

```
//数字項目(外部10進項目，内部10進項目および小数けたを含む2進項目)を
//使用する場合は，以下のようにjava.math.BigDecimalをimportしてください。
//import java.math.BigDecimal;

public class SearchTP1Servlet 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  = "";
    String server  = "";
    String[] host  = new String[1];
    String group   = "";
    String service = "";
    String wkport  = "";
    int    port    = 10020;

    try {
      number  = req.getParameter("Number");
      server  = req.getParameter("SERVER");
      host[0] = req.getParameter("HOST");
      group   = req.getParameter("GROUP");
      service = req.getParameter("SERVICE");
      wkport  = req.getParameter("PORT");


    } catch (Exception e) {
      e.printStackTrace();
    }

    port = Integer.parseInt(wkport);

    res.setContentType("text/html; charset=shift_jis");

    //Beanのインスタンス生成
    SearchTP1 bean  = new SearchTP1();
    TP1Client tp1 = new TP1Client();


    try {
      int[] idArr = new int[1];

      /* Beanにパラメタを設定 */

      bean.setP_numberI(new Integer(number));
      bean.setP_nameI("");
      bean.setP_addressI("");
      bean.setP_gifI("");

      try {
```

```
      if (server.equals("rap") ) {
        //rapサーバ　接続確立
        tp1.openConnection(host[0],port);
      } else {
        //scdサーバ　 接続確立

        //任意の名称の定義ファイルをフルパスで指定した場合
        //tp1.rpcOpen("C:¥¥OpenTP1CltJ¥¥cltjenv.ini");
        /* ファイルの定義内容
        dcscddirect=Y
        dchost=OpenTP1のホスト名
        dcscdport=10010(rapリスナーのポート番号等)
        */

        //システムプロパティで指定している場合
        tp1.rpcOpen();
      }

    } catch(TP1ClientException ex) {
      System.out.println("open failed : " + ex.toString());
      ex.printStackTrace();
      return;
    }
    try {

      //SPPを呼び出します
      bean.call(tp1,group,service,TP1Client.DCNOFLAGS);

      if (server.equals("rap") ) {
        //rapサーバ　 接続開放
        tp1.closeConnection();
      } else {
        //scdサーバ　 接続開放
        tp1.rpcClose();
      }

    } catch(Exception e2) {
      e2.printStackTrace();
      System.out.println("error" + e2.toString());
      return;
    }

    //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("/SearchTP1.jsp");
    rd.forward(req, res);

  } catch (Exception e) {
    System.out.println("test failed  ");
    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>
SearchTP1
</TITLE>
</HEAD>
<BODY>
<h3>
<center>TP1/COBOL adapter for Cosminexus のサンプル</center>
<center>(TP1/Client/J対応版)</center>
</h3>
<hr>
<center>
<FORM  action=/servlet/test.SearchTP1Servlet method="POST">
接続サーバ選択
<SELECT name="SERVER">
<OPTION  value="rap" SELECTED > rapサーバ </OPTION>
<OPTION  value="scd"          > scdサーバ </OPTION>
</SELECT><br>
<hr>
<table border="1">
<tr><td colspan="2" align="center">
<center>設定</center>
</td><tr><td>OpenTP1ホスト名 (*1)</td><td>
<input type="text" name="HOST"    value="servername">
</td><tr><td>ポート番号 (*1)</td><td>
<input type="text" name="PORT"   value="10020">
</td><tr><td>サービスグループ名</td><td>
<input type="text" name="GROUP"   value="SAMPLE">
</td><tr><td>サービス名</td><td>
<input type="text" name="SERVICE" value="SEARCHTP1">
</td></table>
<br>
(*1):rapサーバ接続時に必須
<hr>
<br>
100001 と 100002が登録されています。<br>
Number : <input type="text" name="Number" value="100001">
<BR><BR> Submit を押すと servlet SearchTP1Servlet を実行
<BR><BR><input type=submit value="Submit"></form>
</center>
</BODY>
</HTML>
```

(4)　JSP の作成例

```
<!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.SearchTP1" />
<html>
<center>Result of Search.
<table border="1">
<tr><td colspan="2" align="center">
  <jsp:getProperty name="bean" property="p_number0" />
</td><tr><td>Name</td><td>
  <jsp:getProperty name="bean" property="p_name0" />
</td><tr><td>Address</td><td>
  <jsp:getProperty name="bean" property="p_address0" />
<tr><td colspan="2" align="center">
  <img src="../servlet<jsp:getProperty name="bean" property="p_gif0" />">
</td></table>
</center>
</html>
```

付録 H.3　Cosminexus TP1 Connector 版

(1)　COBOL 引数の登録集原文と COBOL プログラム例

付録 H.1 の TP1/Client/P および TP1/Client/W 版と同じです。「付録 H.1　TP1/Client/P および TP1/Client/W 版」をご覧ください。

(2)　Java UAP(Servlet)例

(a)　Cosminexus TP1 Connector 使用時の概要

```
                :
import  jp.co.hitachi_sk.j2cb.*;   // パッケージのインポート
import  JP.co.Hitachi.soft.OpenTP1.*;   //
import  javax.resource.cci.*;   //
import  jp.co.hitachi_system.tp1connector.*;   //
                :
生成クラス名  cls = new 生成クラス名(); //
                :
javax.naming.Context ctx = new javax.naming.InitialContext();
              //ManagedConnectionFactoryインスタンス生成
                :
javax.resource.cci.Connection cx = cxf.getConnection();
              //ConnectionFactoryインスタンスの生成
javax.resource.cci.Interaction ix = cx.createInteraction();
              //コネクション取得
InteractionSpecImpl ixSpec = new InteractionSpecImpl();
              //InteractionSpecImplインスタンスの生成
                :
```

```
cls.setData1I("");                    // データの設定
              :
cls.call(cxf, ix, ixSpec);
                                      // RPC実行
              :
String str = cls.getData10();         // データの取得
              :
cx.close();                           // サーバとの切断
              :
```

(b) servlet 作成例

```
package testc;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.beans.Beans;
import jp.co.hitachi_sk.j2cb.*;
import JP.co.Hitachi.soft.OpenTP1.*;
import javax.resource.cci.*;
import jp.co.hitachi_system.tp1connector.*;
//数字項目(外部10進項目, 内部10進項目および小数けたを含む2進項目)を
//使用する場合は, 以下のようにjava.math.BigDecimalをimportしてください。
//import java.math.BigDecimal;

public class SearchTP1Servlet 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  = "";
    String group   = "";
    String service = "";

    try {
      number  = req.getParameter("Number");
      group   = req.getParameter("GROUP");
      service = req.getParameter("SERVICE");


    } catch (Exception e) {
      e.printStackTrace();
    }

    res.setContentType("text/html; charset=shift_jis");

    //Beanのインスタンス生成
    SearchTP1 bean  = new SearchTP1();
```

```
    try {
          //ManagedConnectionFactoryインスタンス生成
           ManagedConnectionFactoryImpl mcf =
                   new ManagedConnectionFactoryImpl();

           //ManagedConnectionFactoryプロパティ設定
           // 引数に指定されたTP1/Client/J定義ファイル
           mcf.setConfFileName("C:¥¥OpenTP1CltJ¥¥cltjenv.ini");
/* ファイルの定義内容
   dcscddirect=Y
   dchost=OpenTP1のホスト名
   dcscdport=10010(スケジュールサービスのポート番号)
*/

           //ConnectionFactoryインスタンスの生成
           javax.resource.cci.ConnectionFactory cxf =
                   (javax.resource.cci.ConnectionFactory)
                           mcf.createConnectionFactory();

           //コネクション取得
           javax.resource.cci.Connection cx = cxf.getConnection();

           //Interactionインスタンスの生成
           javax.resource.cci.Interaction ix = cx.createInteraction();

           //InteractionSpecImplインスタンスの生成
           InteractionSpecImpl ixSpec = new InteractionSpecImpl();

           //InteractionSpecプロパティ設定
           ixSpec.setServiceGroupName(group);        //サービスグループ名称
           ixSpec.setServiceName(service);           //サービス名称
           ixSpec.setFlags(ixSpec.DCNOFLAGS);          //RPC種別
           ixSpec.setWatchTime(10);                    //応答待ち時間


      /* Beanにパラメタを設定 */

      bean.setP_numberI(new Integer(number));
      bean.setP_nameI("");
      bean.setP_addressI("");
      bean.setP_gifI("");

      try {

                   //SPPを呼び出します
                   bean.call(cxf,ix,ixSpec);

            } catch(Exception e2) {
                 e2.printStackTrace();
                 System.out.println("error" + e2.toString());
                 cx.close();
                 return;
            }

           //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("/SearchTP1C.jsp");
            rd.forward(req, res);
                  cx.close();
     } catch (Exception e) {
      System.out.println("test failed  ");
               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>
SearchTP1
</TITLE>
</HEAD>
<BODY>
<h3>
<center>TP1/COBOL adapter for Cosminexus のサンプル</center>
<center>(Cosminexus TP1 Connector対応版)</center>
</h3>
<hr>
<center>
<FORM  action=/servlet/testc.SearchTP1Servlet method="POST">
<hr>
<table border="1">
<tr><td colspan="2" align="center">
<center>設定</center>
</td><tr><td>サービスグループ名</td><td>
<input type="text" name="GROUP"   value="SAMPLE">
</td><tr><td>サービス名</td><td>
<input type="text" name="SERVICE" value="SEARCHTP1">
</td></table>
<br>
<hr>
<br>
100001 と 100002が登録されています。<br>
Number : <input type="text" name="Number" value="100001">
<BR><BR> Submit を押すと servlet SearchTP1Servlet を実行
<BR><BR><input type=submit value="Submit"></form>
</center>
```

```
</BODY>
</HTML>
```

(4) JSP の作成例

```
<!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="testc.SearchTP1" />
<html>
  <center>Result of Search.
    <table border="1">
      <tr><td colspan="2" align="center">
        <jsp:getProperty name="bean" property="p_number0" />
      </td><tr><td>Name</td><td>
        <jsp:getProperty name="bean" property="p_name0" />
      </td><tr><td>Address</td><td>
        <jsp:getProperty name="bean" property="p_address0" />
      <tr><td colspan="2" align="center">
        <img src="../servlet<jsp:getProperty name="bean" property="p_gif0" />">
    </td></table>
  </center>
</html>
```

付録 H.4 Cosminexus TP1 Connector 版（Managed 環境）

(1) COBOL 引数の登録集原文と COBOL プログラム例

付録 H.1 の TP1/Client/P および TP1/Client/W 版と同じです。「付録 H.1 TP1/Client/P および TP1/Client/W 版」をご覧ください。

(2) Java UAP(Servlet)例

(a) Cosminexus TP1 Connector 使用時の概要

```
                :
import  jp.co.hitachi_sk.j2cb.*;   // パッケージのインポート
import  JP.co.Hitachi.soft.OpenTP1.*;   //
import  javax.resource.cci.*;   //
import  jp.co.hitachi_system.tp1connector.*;   //
                :
生成クラス名  cls = new 生成クラス名(); //
                :
javax.naming.Context ctx = new javax.naming.InitialContext();
             //ManagedConnectionFactoryインスタンス生成
javax.resource.cciConnectionFactory cvf =
        (javax.resource.cci.ConnectionFactory)ctx.lookup(RA_NAME);
javax.resource.cci.Connection cx = cxf.getConnection();
             //ConnectionFactoryインスタンスの生成
```

```
javax.resource.cci.Interaction ix = cx.createInteraction();
            //コネクション取得
InteractionSpecImpl ixSpec = new InteractionSpecImpl();
            //InteractionSpecImplインスタンスの生成
              :
cls.setData1I("");                      // データの設定
              :
cls.call(cxf, ix, ixSpec);
                                        // RPC実行
              :
String str = cls.getData10();           // データの取得
              :
cx.close();                            // サーバとの切断
              :
```

(b) servlet 作成例

```
package testc;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.beans.Beans;
import jp.co.hitachi_sk.j2cb.*;
import JP.co.Hitachi.soft.OpenTP1.*;
import javax.resource.cci.*;
import jp.co.hitachi_system.tp1connector.*;
//数字項目(外部10進項目, 内部10進項目および小数けたを含む2進項目)を
//使用する場合は, 以下のようにjava.math.BigDecimalをimportしてください。
//import java.math.BigDecimal;

public class SearchTP1Servlet extends HttpServlet {
  private static final long serialVersionUID = 0L;
  ServletContext c;
  private static final String RA_NAME = "java:comp/env/OpenTP1";

  //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 {
    try {

      String number  = "";
      String group   = "";
      String service = "";

      Res.setContextType("text/html; charset=shift_jis");
      PrintWriter pw = res.getWriter();

      // 初期画面表示
      /* ******************* */
```

```
/* **  入力用 HTML  ** */
/* ****************** */

pw.println("<HTML>");
pw.println("<HEAD>");
pw.println("<META HTTP-EQUIV=¥"Content-Type¥" CONTENT=¥"text/html;
        charset=shift_jis¥">");
pw.println("<TITLE>");
pw.println("SearchTP1");
pw.println("</TITLE>");
pw.println("</HEAD>");
pw.println("<BODY>");
pw.println("<h3>");
pw.println( "<center>TP1/COBOL adapter for Cosminexusのサンプル
          </center>");
pw.println( "<center>(Cosminexus TP1 Connector対応版) </center>" );
pw.println( "<center> Managed 環境</center>" );
pw.println("</h3>");
pw.println("<hr>");
pw.println("<FORM  action=/servlet/testc.SearchTP1Servlet
           method=¥"POST¥">");
pw.println("<hr>");
pw.println("<table border=¥"1¥">");
pw.println("<tr><td colspan=¥"2¥" align=¥"center¥">");
pw.println( "<center>設定</center>" );
pw.println( "</td><tr><td>サービスグループ名</td><td>" );
pw.println("<input type=¥"text¥" name=¥"GROUP¥"
           value=¥"SAMPLE¥">");
pw.println( "</td><tr><td>サービス名</td><td>" );
pw.println("<input type=¥"text¥" name=¥"SERVICE¥"
           value=¥"SEARCHTP1¥">");
pw.println("</td></table>");
pw.println("<br>");
pw.println("<hr>");
pw.println("<br>");
pw.println( "100001 と 100002が登録されています。
pw.println( "<BR><BR> Submit を押すと Servlet SearchTP1Servlet
           を実行" );
pw.println("<BR><BR><input type=submit value=¥"Submit¥"></form>");
pw.println("</center>");
pw.println("</BODY>");
pw.println("</HTML>");

try {
  number  = req.getParameter("Number");
  group   = req.getParameter("GROUP");
  service = req.getParameter("SERVICE");
} catch (Exception e) {
  e.printStackTrace();
}

//Beanのインスタンス生成
SearchTP1 bean  = new SearchTP1();

// ****************
// * managed 環境 *
// ****************
javax.naming.Context ctx = new javax.naming.InitialContext();
```

```
javax.resource.cci.ConnectionFactory cxf =
       (javax.resource.cci.ConnectionFactory)
                          ctx.lookup(RA_NAME);


// ****************************************
// *  managed , non-managed 環境共通処理
// * ****************************************
// コネクション取得
javax.resource.cci.Connection cx = cxf.getConnection();

//Interactionインスタンスの生成
javax.resource.cci.Interaction ix = cx.createInteraction();

//InteractionSpecImplインスタンスの生成
InteractionSpecImpl ixSpec = new InteractionSpecImpl();

//InteractionSpecプロパティ設定
ixSpec.setServiceGroupName(group);        //サービスグループ名称
ixSpec.setServiceName(service);           //サービス名称
ixSpec.setFlags(ixSpec.DCNOFLAGS);        //RPC種別
ixSpec.setWatchTime(10);                  //応答待ち時間

// Beanにパラメタを設定

bean.setP_numberI(new Integer(number));
bean.setP_nameI("");
bean.setP_addressI("");
bean.setP_gifI("");

try {

  //SPPを呼び出します
  bean.call(cxf,ix,ixSpec);

} catch(Exception e) {
  e.printStackTrace();
  cx.close();
  return;
}

/* ****************** */
/* **  結果用 HTML  ** */
/* ****************** */
pw.println("<html>");
pw.println("<center>Result of Search.");
pw.println("<table border=¥"1¥">");
pw.println("<tr><td colspan=¥"2¥" align=¥"center¥">");
pw.println(bean.getP_number0());
pw.println("</td><tr><td>Name</td><td>");
pw.println(bean.getP_name0());
pw.println("</td><tr><td>Address</td><td>");
pw.println(bean.getP_address0());
pw.println("<tr><td colspan=¥"2¥" align=¥"center¥">");
pw.println("<img src=" + bean.getP_gif0() + ">");
pw.println("</td></table>");
pw.println("</center>");
pw.println("</html>");
```

```
      cx.close();

   } catch (Exception e) {
      e.printStackTrace();
      return;
   }

 }

  // HTTP Post リクエストの処理
  public void doPost(HttpServletRequest request, HttpServletResponse response) throws Servle
tException, IOException {
    doGet(request, response);
  }
}
```

付録 H.5　OCCURS 句を使用した例

TP1/Client/P または TP1/Client/W 版，TP1/Client/J 版および Cosminexus TP1 Connector 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

```
入力COBOL引数の登録集原文例
       01 G1.
        02 G2  OCCURS 10.
         03 B1         PIC X(50).
出力COBOL引数の登録集原文例
       01 F1.
        02 F2  OCCURS 10.
         03 E1         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 E1(10).
              :
```

(2)　Java UAP(Servlet)例

```
 :
int i = 0;
for ( i = 0; i < 10; i++ ) {
      bean.setB1I("XXXXX", i);      …  B1(0)～B1(9)へ"XXXXX"を設定
    }
```

```
    :
    String wkstr = bean.getE10(5);    … B1(5)を取得
    :
```

付録 H.6　SOAP 用 JavaUAP の作成例

SOAP サーバアプリケーション用の JavaUAP の作成例です。ここで記載されている javaUAP の内容は TP1/COBOL SOAP サービスクラス生成機能で生成された「スケルトン名+SBean.java」中のメソッドを使用して作成しています。SOAP サーバ側で引数データの内容を操作する必要がない場合，「スケルトン名+SBean.java」を使用する必要ありません。

また，この例題は TP1/COBOL アクセスに付随の Cosminexus TP1 Connector のサンプルプログラムを元に作成されています。そのため，登録集原文の内容はここでは記載しておりませんので，サンプルをご覧ください。

なお，例題で使用されている「CBLTP1」はウィザードで指定したスケルトン名称，「TP1_method」はウィザードで指定したメソッド名称，「CBLTP1UAP」は呼び出される JavaUAP 名称，「CBLTP1UAPmethod」は呼び出される JavaUAP のメソッド名称となっています。

(1)　SOAP サーバ用の JavaUAP の作成例

橙色の部分は，TP1/COBOL SOAP サーバ用 Bean を使用して，入出力引数の参照・設定を行う際の作成例です。

SOAP サーバ上で入出力引数の参照・設定を行わない場合は不要です。

```
package localhost;

import jp.co.hitachi_sk.j2cb.*;
import JP.co.Hitachi.soft.OpenTP1.*;
import java.math.BigDecimal;
import javax.resource.cci.*;
import jp.co.hitachi_system.tp1connector.*;

public class CBLTP1UAP {
  public byte[] CBLTP1UAPmethod(byte[] inData) {

    byte[] outData = null;
    CBLTP1SBean bean = null;
    String group = "SAMPLE";
    String service = "SEARCHTP1";
    Integer p_number = null;
    String p_name = null;
    String p_address = null;
    String p_gif = null;

    try {
      bean = new CBLTP1SBean();
    } catch ( Exception e ) {
```

```
      e.printStackTrace();
      throw e;
    }

    try {
      bean.setBytesData(CBLTP1SBean.inIndex, inData);
      p_number = (Integer)bean.getP_numberI();
      p_name = (String)bean.getP_addressI();
      p_address = (String)bean.getP_addressI();
      p_gif = (String)bean.getP_gifI();
      outData = new byte[CBLTP1.outMaxSize];
    } catch ( Exception e ) {
      e.printStackTrace();
      throw e;
    }

    try {
      //ManagedConnectionFactoryインスタンス生成
      ManagedConnectionFactoryImpl mcf =
                  new ManagedConnectionFactoryImpl();

      //ManagedConnectionFactoryプロパティ設定
      // 引数に指定されたTP1/Client/J定義ファイル
      // 実行環境に合わせて必ず変更してください。
      mcf.setConfFileName("C:¥¥OpenTP1ClientJ¥¥cltjenv.ini");

      //ConnectionFactoryインスタンスの生成
      javax.resource.cci.ConnectionFactory cxf =
                  (javax.resource.cci.ConnectionFactory)
                          mcf.createConnectionFactory();

      //コネクション取得
      javax.resource.cci.Connection cx = cxf.getConnection();

      //Interactionインスタンスの生成
      javax.resource.cci.Interaction ix = cx.createInteraction();

      //InteractionSpecImplインスタンスの生成
      InteractionSpecImpl ixSpec = new InteractionSpecImpl();

      //InteractionSpecプロパティ設定
      ixSpec.setServiceGroupName(group);        //サービスグループ名称
      ixSpec.setServiceName(service);           //サービス名称
      ixSpec.setFlags(ixSpec.DCNOFLAGS);        //RPC種別
      ixSpec.setWatchTime(10);                  //応答待ち時間

      // OpenTP1呼び出し実行
      RecordFactory rf = cxf.getRecordFactory();
      IndexedRecord input = rf.createIndexedRecord("in_record");
      IndexedRecord output = rf.createIndexedRecord("out_record");
      input.add(inData);
      outData = new byte[CBLTP1.outMaxSize];
      output.add(outData);
      boolean ret = ix.execute(ixSpec, input, output);
      cx.close();
    } catch (Exception e) {
      e.printStackTrace();
      throw e;
```

```
    }

    try {
      bean.setBytesData(CBLTP1SBean.outIndex, outData);
      p_number = (Integer)bean.getP_number0();
      p_name = (String)bean.getP_address0();
      p_address = (String)bean.getP_address0();
      p_gif = (String)bean.getP_gif0();
    } catch ( Exception e ) {
      e.printStackTrace();
      throw e;
    }
    return outData;
  }
}
```

(2)　SOAP クライアント用の JavaUAP の作成例

プログラム中で使用されている「CBLTP1_ServiceLocator」,「getTP1server」および「CBLTP1_Port」は WSDL からソース生成を行った際に自動生成されたソース名称およびメソッド名称です。
「CBLTP1CBean」は「SOAP 用クライアント Bean 生成ウィザード」で生成されたプログラムです。クライアント側 UAP ではこの「CBLTP1CBean」中のメソッドを使用します。

```
package localhost;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import javax.xml.rpc.holders.*;

import com.cosminexus.c4web.management.Management;
import com.cosminexus.c4web.management.ClientID;

public class CBLTP1Servlet extends HttpServlet {
    static final private String CONTENT_TYPE = "text/html; charset=Shift_JIS";

    // クライアント識別子
    private ClientID cltID = null;

    public void init() throws ServletException
    {
        // SOAPクライアントの開始
        cltID = Management.initializeClient();
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws Serv
letException, IOException{
      // クライアント識別子と実行スレッドを関連付ける
      Management.connectClientIDtoCurrentThread(cltID);

      String var0 = request.getParameter("Number");
      if (var0 == null) {
        var0 = "100001";
      }
```

```
Integer pnumber = new Integer(var0);
String pname = null;
String paddress = null;
String pgif = null;
CBLTP1_ServiceLocator uis = null;
CBLTP1_Port skeltonClass = null;
byte[] inData = null;
byte[] outDataValue = new byte[CBLTPICBeam.outMaxSi e;];
javax.xml.rpc.holders.ByteArrayHolder outData =
            new javax.xml.rpc.holders.ByteArrayHolder(outDataValue);
CBLTP1CBean bean = null;

try {
  // サービスのインタフェースクラスを生成
  uis = new CBLTP1_ServiceLocator();
} catch ( RuntimeException e ) {
  response.setContentType(CONTENT_TYPE);
  PrintWriter out = response.getWriter();
  out.println("<html>");
  out.println("<head><title>CBLTP1CServlet Error.</title></head>");
  out.println("<body>");
  out.println("サービスのインタフェースクラス生成に失敗しました。");
  out.println("</body></html>");
  return;
}

try {
  // クライアントのインタフェースクラスを取得
  skeltonClass = uis.getTP1server();
} catch ( Exception e ) {
  response.setContentType(CONTENT_TYPE);
  PrintWriter out = response.getWriter();
  out.println("<html>");
  out.println("<head><title>CBLTP1CServlet Error.</title></head>");
  out.println("<body>");
  out.println("クライアントのインタフェースクラス取得に失敗しました。");
  out.println("</body></html>");
  return;
}

try {
  // クライアントアクセス用のBean生成
  bean = new CBLTP1CBean();
} catch ( Exception e ) {
  response.setContentType(CONTENT_TYPE);
  PrintWriter out = response.getWriter();
  out.println("<html>");
  out.println("<head><title>CBLTP1CServlet Error.</title></head>");
  out.println("<body>");
  out.println("クライアントアクセス用Bean生成に失敗しました。");
  out.println("</body></html>");
  return;
}

try {
  // 入力引数項目の設定
  bean.setP_numberI(pnumber);
} catch ( Exception e ) {
```

```
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>CBLTP1CServlet Error.</title></head>");
        out.println("<body>");
        out.println("入力引数項目設定に失敗しました。");
        out.println("</body></html>");
        return;
      }

      try {
        // 入力引数の取得
        inData = bean.getBytesData(CBLTP1CBean.inIndex);
      } catch ( Exception e ) {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>CBLTP1CServlet Error.</title></head>");
        out.println("<body>");
        out.println("入力引数取得に失敗しました。");
        out.println("</body></html>");
        return;
      }

      try {
        // サービスメソッドの呼び出し
        skeltonClass.TP1_Method(inData, outData);
      } catch ( Exception e ) {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>CBLTP1CServlet Error.</title></head>");
        out.println("<body>");
        out.println("サービスメソッド呼び出しに失敗しました。<br>");
        out.println("e.getMessage()=" + e.getMessage() + "<br>");
        out.println("e.getLocalizedMessage()=" + e.getLocalizedMessage() + "<br>");
        out.println("e.toString()=" + e.toString() + "<br>");
        out.println("</body></html>");
        e.printStackTrace();
        return;
      }
      if ( outData.value == null ) {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>CBLTP1CServlet Error.</title></head>");
        out.println("<body>");
        out.println("出力引数がnullです。終了します。");
        out.println("</body></html>");
        return;
      }

      try {
        // 出力引数の設定
        bean.setBytesData(CBLTP1CBean.outIndex, outData.value);
      } catch ( Exception e ) {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
```

```
        out.println("<html>");
        out.println("<head><title>CBLTP1CServlet Error.</title></head>");
        out.println("<body>");
        out.println("出力引数設定に失敗しました。");
        out.println("</body></html>");
        return;
      }

      try {
        // 出力引数項目の取得
        pname = (String)bean.getP_name0();
        paddress = (String)bean.getP_address0();
        pgif = (String)bean.getP_gif0();
      } catch ( Exception e ) {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>CBLTP1CServlet Error.</title></head>");
        out.println("<body>");
        out.println("出力引数項目取得に失敗しました。");
        out.println("</body></html>");
        return;
      }

      response.setContentType(CONTENT_TYPE);
      PrintWriter out = response.getWriter();
      out.println("<html>");
      out.println("<head><title>CBLTP1CServlet</title></head>");
      out.println("<body>");
      out.println("<center>Result of Search.");
      out.println("<table border=¥"1¥">");
      out.println("<tr><td colspan=¥"2¥" align=¥"center¥">" + pnumber + "</td></tr>");
      out.println("<tr><td>Name</td><td>" + pname + "</td></tr>");
      out.println("<tr><td>Address</td><td>" + paddress + "</td></tr>");
      out.println("<tr><td colspan=¥"2¥" align=¥"center¥"><img src=¥"." + pgif + "¥"/></td><
/tr>");
      out.println("</table>");
      out.println("</center>");
      out.println("</body></html>");
    }
    //リスースの後処理
    public void destroy()
    {
        // SOAPクライアントの終了
        Management.finalizeClient(cltID);
    }
}
```

付録 I　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 I.1　関連マニュアル

このマニュアルの関連マニュアルを次に示します。必要に応じてお読みください。

- COBOL2002　ユーザーズガイド　（3021-3-600）
- COBOL2002　操作ガイド　（3021-3-601）
- COBOL2002　使用の手引　手引編（3021-3-602）
- COBOL2002　使用の手引　操作編（3021-3-603）
- COBOL2002　言語　標準仕様編　（3021-3-604）
- COBOL2002　言語　拡張仕様編　（3021-3-605）

注※　COBOL85 をお使いの方は次のマニュアルをご覧ください。

なお，このマニュアルでは第 1 編は COBOL2002 を前提に記述しております。

- COBOL85　ユーザーズガイド　（3020-3-852）
- COBOL85　操作ガイド　（3020-3-873）
- COBOL85　使用の手引　（3000-3-354）
- COBOL85　言語　（3020-3-782）
- OpenTP1 Version 6　分散トランザクション処理機能　OpenTP1　システム定義　（3000-3-943）
- OpenTP1 Version 6　分散トランザクション処理機能　OpenTP1　運用と操作　（3000-3-944）
- OpenTP1 Version 6　分散トランザクション処理機能　OpenTP1　プログラム作成リファレンス　COBOL 言語編　（3000-3-946）
- OpenTP1 Version 6　分散トランザクション処理機能　OpenTP1　テスタ・UAP トレース使用の手引　（3000-3-948）
- OpenTP1 Version 6　分散トランザクション処理機能　OpenTP1　クライアント使用の手引　TP1/Client/W，TP1/Client/P 編　（3000-3-949）
- OpenTP1 Version 6　分散トランザクション処理機能　OpenTP1　クライアント使用の手引　TP1/Client/J 編　（3000-3-950）
- OpenTP1 Version 7 分散トランザクション処理機能　OpenTP1　システム定義　（3000-3-D52）
- OpenTP1 Version 7 分散トランザクション処理機能　OpenTP1　運用と操作　（3000-3-D53）
- OpenTP1 Version 7 分散トランザクション処理機能　OpenTP1　プログラム作成リファレンス　COBOL 言語編　（3000-3-D55）

- OpenTP1 Version 7 分散トランザクション処理機能　OpenTP1　テスタ・UAP トレース使用の手引　(3000-3-D57)
- OpenTP1 Version 7 分散トランザクション処理機能　OpenTP1　クライアント使用の手引　TP1/Client/W，TP1/Client/P 編　(3000-3-D58)
- OpenTP1 Version 7 分散トランザクション処理機能　OpenTP1　クライアント使用の手引　TP1/Client/J 編　(3000-3-D59)
- コード変換　ユーザーズガイド　解説・文法・操作書　(3020-7-350)
- 日立コード変換ユーザーズガイド　(3020-7-351)
- 日立コード変換ユーザーズガイド　(3020-7-355)
- 日立コード変換ユーザーズガイド　(3000-7-415)
- Cosminexus　システム構築ガイド　(3020-3-M06)
- Cosminexus　システム運用ガイド　(3020-3-M07)
- Cosminexus　アプリケーション開発ガイド　(3020-3-M41)
- Cosminexus　リファレンス　定義編　(3020-3-M11)
- Cosminexus SOAP アプリケーション開発ガイド　(3020-3-M47)
- Cosminexus　解説　(3000-3-931)
- Cosminexus Version 6　システム構築ガイド　(Windows(R)用)　(3020-3-E53)
- Cosminexus Version 6　システム運用ガイド　(Windows(R)用)　(3020-3-E54)
- Cosminexus Version 6　アプリケーション開発ガイド　(3020-3-E55)
- Cosminexus Version 6　リファレンス　(Windows(R)用)　(3020-3-E56)
- Cosminexus Version 6　システム構築ガイド　(UNIX(R)用)　(3000-3-985)
- Cosminexus Version 6　システム運用ガイド　(UNIX(R)用)　(3000-3-986)
- Cosminexus Version 6　リファレンス　(UNIX(R)用)　(3000-3-987)
- Cosminexus SOAP アプリケーション開発ガイド　(3020-3-E32)
- 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)

付録 I.2　このマニュアルでの表記

このマニュアルでは，製品種別によって，記載書き分けを行っています。本文中において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 版	Windows(32bit)版	P-2636-F124　TP1/COBOL adapter for Cosminexus Version 2 P-2436-G124　TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2 P-2636-F324　TP1/COBOL adapter for Cosminexus Version 2 P-2436-G324　TP1/COBOL 拡張 Server Run Time System for Cosminexus Version 2
	Windows(64bit)版	P-2936-F324　TP1/COBOL adapter for Cosminexus Version 2(64)
HP-UX 版	HP-UX(PA-RISC)版	P-1B36-G121　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
	HP-UX(IPF※1)版※2	P-1J36-G221　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2(64)
AIX 版		P-1M36-G121　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2 P-1M36-G321　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
Linux 版	Linux(x86)版	P-9S36-G121　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
	Linux(IPF※1)版※2	P-9V36-G221　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2
	Linux(x86/x64)版	P-9S36-G321　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2 P-9S36-G331※3　TP1/COBOL 拡張 Run Time System for Cosminexus Version 2

注※1

IPF：Itanium(R)　Processor　Family

注※2

HP-UX(IPF)：64bit 版 HP-UX(IPF)

Linux(IPF)：64bit 版 Linux(IPF)

注※3

64bit ネイティブモードだけで動作します。

また，このマニュアルでは，各製品を次のように表記しています。

- Microsoft(R) Internet Explorer を Internet Explorer と表記しています。
- Microsoft(R) Windows(R) 98 Operating System を Windows 98 と表記しています。
- Microsoft(R) Windows NT(R) Server Network Operating System Version 4.0 および Microsoft(R) Windows NT(R) Server Network Operating System Version 4.0 Enterprise Edition を Windows NT 4.0 Server と表記しています。

- Microsoft(R) Windows NT(R) Workstation Operating System 4.0 を Windows NT 4.0 Workstation と表記しています。

なお，Windows NT 4.0 Server と Windows NT 4.0 Workstation とで機能差異がない場合，Windows NT 4.0 と表記しています。

- Cosminexus Web Contents Generator を JRun と表記しています。
- TP1/COBOL adapter for Cosminexus Version 2 および TP1/COBOL adapter for Cosminexus を TP1/COBOL アクセスと表記しています。
- Eclipse 3.1.1 を Eclipse と表記しています。
- Microsoft(R) Windows(R) 2000 Server Operating System，Microsoft(R) Windows(R) 2000 Advanced Server Operating System および Microsoft(R) Windows(R) 2000 Datacenter Server Operating System を Windows 2000 Server と表記しています。
- Microsoft(R) Windows(R) Millennium Edition Operating System を Windows Me と表記しています。
- Microsoft(R) Windows(R) 2000 Professional Operating System，Microsoft(R) Windows(R) 2000 Server Operating System，Microsoft(R) Windows(R) 2000 Advanced Server Operating System および Microsoft(R) Windows(R) 2000 Datacenter Server Operating System を Windows 2000 と表記しています。
- Microsoft(R) Windows(R) XP Professional を Windows XP と表記しています。
- Microsoft(R) Windows Server(R) 2003，Standard Edition Operating System，Microsoft(R) Windows Server(R) 2003，Enterprise Edition Operating System，Microsoft(R) Windows Server(R) 2003 R2，Standard Edition Operating System および Microsoft(R) Windows Server(R) 2003 R2，Enterprise Edition Operating System を Windows Server 2003(x86)と表記しています。
- Microsoft(R) Windows Server(R) 2003，Standard x64 Edition Operating System，Microsoft(R) Windows Server(R) 2003，Enterprise x64 Edition Operating System，Microsoft(R) Windows Server(R) 2003 R2，Standard x64 Edition Operating System および Microsoft(R) Windows Server(R) 2003 R2，Enterprise x64 Edition Operating System を Windows Server 2003 x64 Editions と表記しています。
- Windows Server 2003(x86)と Windows Server 2003 x64 Editions に共通な場合は，Windows Server 2003 と表記しています。
- Microsoft(R) Windows Vista(R) Business，Microsoft(R) Windows Vista(R) Enterprise および Microsoft(R) Windows Vista(R) Ultimate を Windows Vista と表記しています。
- Microsoft(R) Windows Server(R) 2008 Standard 日本語版,
  Microsoft(R) Windows Server(R) 2008 Enterprise 日本語版,
  Microsoft(R) Windows Server(R) 2008 Standard 32-bit 日本語版,
  Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit 日本語版,

Microsoft(R) Windows Server(R) 2008 R2 Standard 日本語版，
Microsoft(R) Windows Server(R) 2008 R2 Enterprise 日本語版および
Microsoft(R) Windows Server(R) 2008 R2 Datacenter 日本語版を
Windows Server 2008 と表記しています。

- Microsoft(R) Windows(R) 7 Professional 日本語版(32 ビット版)，
  Microsoft(R) Windows(R) 7 Enterprise 日本語版(32 ビット版)および
  Microsoft(R) Windows(R) 7 Ultimate 日本語版(32 ビット版)を
  Windows 7 x86 と表記しています。
  Microsoft(R) Windows(R) 7 Professional 日本語版(64 ビット版)，
  Microsoft(R) Windows(R) 7 Enterprise 日本語版(64 ビット版)および
  Microsoft(R) Windows(R) 7 Ultimate 日本語版(64 ビット版)を
  Windows 7 x64 と表記しています。
  Windows 7 x86 と Windows 7 x64 に共通な場合は，Windows 7 と表記しています。
- Microsoft(R) Windows Server(R) 2012 Standard 日本語版，
  Microsoft(R) Windows Server(R) 2012 Datacenter 日本語版，
  Microsoft(R) Windows Server(R) 2012 R2 Standard 日本語版および
  Microsoft(R) Windows Server(R) 2012 R2 Datacenter 日本語版を
  Windows Server 2012 と表記しています。
- Windows(R) 8 Pro 日本語版(32 ビット版)，
  Windows(R) 8 Enterprise 日本語版(32 ビット版)，
  Windows(R) 8.1 Pro 日本語版(32 ビット版)および
  Windows(R) 8.1 Enterprise 日本語版(32 ビット版)を
  Windows 8 x86 と表記しています。
- Windows(R) 8 Pro 日本語版(64 ビット版)，
  Windows(R) 8 Enterprise 日本語版(64 ビット版)，
  Windows(R) 8.1 Pro 日本語版(64 ビット版)および
  Windows(R) 8.1 Enterprise 日本語版(64 ビット版)を
  Windows 8 x64 と表記しています。
  Windows 8 x86 と Windows 8 x64 に共通な場合は，Windows 8 と表記しています。
- Windows(R) 10 Pro 日本語版(32 ビット版)および
  Windows(R) 10 Enterprise 日本語版(32 ビット版)を
  Windows 10 x86 と表記しています。
  Windows(R) 10 Pro 日本語版(64 ビット版)および
  Windows(R) 10 Enterprise 日本語版(64 ビット版)を
  Windows 10 x64 と表記しています。
  Windows 10 x86 と Windows 10 x64 に共通な場合は，Windows 10 と表記しています。

- Microsoft(R) Windows Server(R) 2016 Standard 日本語版および Microsoft(R) Windows Server(R) 2016 Datacenter 日本語版を Windows Server 2016 と表記しています。
- Microsoft(R) Windows Server(R) 2019 Standard 日本語版および Microsoft(R) Windows Server(R) 2019 Datacenter 日本語版を Windows Server 2019 と表記しています。
- Microsoft$_{(R)}$ Internet Information Server を Internet Information Server と表記しています。
- uCosminexus Developer，uCosminexus Service Architect，および Cosminexus Developer を総称して，Cosminexus Developer と表記しています。
- uCosminexus Application Server，uCosminexus Service Platform，および Cosminexus Application Server を総称して，Cosminexus Application Server と表記しています。
- uCosminexus TP1 Connector，および Cosminexus TP1 Connector を総称して，Cosminexus TP1 Connector と表記しています。
- uCosminexus TP1/Client/J，および TP1/Client/J を総称して，TP1/Client/J と表記しています。
- Cosminexus Component Container（Windows/HP-UX/AIX 版）または Cosminexus Server Component Container for Java（Solaris 版）を Component Container と表記しています。
- Cosminexus Component Container（Windows/HP-UX/AIX 版），または Cosminexus Server Component Container for Java（Solaris 版）において，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（Windows/HP-UX/AIX 版）または Cosminexus Server Component Container for Java（Solaris 版）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。
- TP1/Client/P，TP1/Client/W，TP1/Client/J および TP1 Connector を総称して TP1/Client と表記します。
- AIX COBOL2002 Net Server Suite および AIX COBOL2002 Net Server Runtime を総称して AIX COBOL2002 と表記します。

また，特に，Windows 版，HP-UX 版，Solaris 版，AIX 版，または Linux 版と記述がないかぎり，全システムでの説明となっております。さらに，HP-UX/Solaris/AIX/Linux を総称して，Unix と表記します。

付録 I.3　英略語

このマニュアルで使用する英略語の一覧とその内容の説明を示します。

英略語	英字での表記	内容
API	Application Program Interface	アプリケーションプログラムインタフェース

英略語	英字での表記	内容
CUP	Client User Program	WS，または PC 側のサービスを要求するプログラム
CCI	Common Client Interface	アプリケーションコンポーネント用の標準クライアントインタフェース
J2SE	Java 2 Standard Edition	Sun Microsystems,Inc より公開されている規格仕様。
JNI	Java Native Interface	C 言語などで作成されたプログラムと Java を連携させるためのインタフェース
JSP	Java Server Pages	Java を使って動的コンテンツを作成する手法。JSP ページとは，HTML の中に JSP 仕様で決められた独自のタグを記述したコンテンツ。JSP タグの解析には解析用の JSP プロセッサが必要である。
MHP	Message Handling Program	OpenTP1 の UAP でメッセージ処理専用のプログラム
RPC	Remote Procedure Call	リモートプロシジャコール
SPP	Service Providing Program	OpenTP1 の UAP でサーバの役割をするプログラム
UAP	User Application Program	ユーザアプリケーションプログラム

付録 I.4　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 J　用語解説

(英字)

Cosminexus TP1 Connector

TP1/Client/J と連携して OpenTP1 に対するリソースアダプタとして機能する製品です。

Eclipse

Eclipse プロジェクト(eclipse.org)が提供するオープンソースの統合開発環境です。ソースコードの編集支援機能やデバッグ機能など，アプリケーションの開発効率を向上させる各種機能を備えています。

getter

プロパティから値を取得するメソッドです。

JavaBeans

Java プログラムから利用できるソフトウエア部品の仕様です。個々の部品を Bean と呼びます。

JBuilder

Cosminexus の構成製品になっている JBuilder 2005 Enterprise，JBuilder X Enterprise，JBuilder 9 Enterprise，および JBuilder 7 Enterprise を JBuilder と記述します。JBuilder は，Java2 プラットフォームをサポートし，Java アプリケーション，アプレット，Servlet，JavaBeans などのビジュアル開発を実現します。

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。

setter

プロパティに値を設定するメソッドです。

TP1/Client/J

クライアント用マシンの Java 環境から，OpenTP1 のサーバへサービスを要求できるようにする製品です。

TP1/Client/P

PC をクライアント用マシンとして，OpenTP1 のサーバへサービスを要求できるようにする製品です。

TP1/Client/W

WS をクライアント用マシンとして，OpenTP1 のサーバへサービスを要求できるようにする製品です。

TP1/LiNK

分散システム環境で，UAP のスケジュールなどの基本制御をする製品です。TP1/Server Base に比べ，小規模な部門に適用できます。

TP1/Messaging

OpenTP1 のメッセージ制御をする製品です。TP1/LiNK があらかじめ組み込まれている環境で動作し，TCP/IP 通信機能を使用したオンラインシステムを構築できます。

TP1/Message Control

OpenTP1 のメッセージ制御機能の管理を行う製品です。

TP1/Server

TP1/Server Base および TP1/LiNK の総称です。

TP1/Server Base

分散トランザクション処理の基本制御をする製品です。トランザクションを制御したり，UAP をスケジュールしたりします。

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。主な符号化文字集合として UCS-2，UCS-4 がある。主なエンコーディングスキーマとして UTF-8，UTF-16 がある。

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。1 文字を 2 バイトまたは 4 バイトで表現する。UTF-16 では 2 バイト文字の 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）がある。

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。ASCII 文字を 1 バイト，日本語文字および半角かたかなは 3 バイトで表現する。

WRP(Windows Resource Protection)

Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS から導入された，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。特定の OS ファイル，フォルダ，およびレジストリキーといった，Windows の読み取り専用リソースを保護する。

（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。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付けられ，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た権限です。Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS では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換えまでの権限は持ちません。

管理者ユーザ

Windows Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS 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。

（タ行）

登録集原文

TP1/COBOL アクセス用 Bean を生成するための入力情報です。TP1/COBOL アクセス用 Bean の COBOL データ定義規則に従って作成した SPP の in/out パラメタの情報です。

（ハ行）

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 Vista 以降のクライアント OS または Windows Server 2008 以降のサーバ OS の Users グループの属するアカウントです。

索引

U

W

X

あ

い

う

お

か

き

く

こ

さ

よ

り

れ