Interstage

FUJITSU

Interstage Business Application Server V10.0.0

COBOLアプリケーション開発リファレンス

Windows/Solaris/Linux

B1X1-0092-01Z0(00)
2011年8月

まえがき

■本書の目的

本書は、“Interstage Business Application Server COBOLアプリケーション開発リファレンス”です。

本書は、NetCOBOL開発パッケージとInterstage Studio 互換ワークベンチを使って、COBOLアプリケーション、CORBAサーバアプリケーションおよびCORBAクライアントアプリケーションを開発する人のために必要な事項を説明しています。

本書は、Interstage Studio 互換ワークベンチにCOBOLプラグインを組み込んで使用するときに参照するマニュアルです。Interstage Studioに組み込んだCOBOLプラグインの使用方法とCOBOLまたはCORBAアプリケーションの開発方法について解説しています。Interstage Studio 互換ワークベンチのインストール方法および使用方法についてはInterstage Studioのマニュアルを参照してください。

■前提知識

本書を読む場合、以下の知識が必要です。

- 使用するOSに関する基本的な知識
- COBOLに関する基本的な知識
- Interstage Application Serverに関する基本的な知識
- Interstage Studioに関する基本的な知識
- リレーショナルデータベースに関する基本的な知識

■本書の構成

本書は、以下のように構成されています。

第1部 導入編

第1章 概要

COBOLプラグインをインストールすることで追加される機能について説明しています。

第2章 COBOLプラグインによって開発できるアプリケーション

COBOLプラグインをインストールすることで開発できるアプリケーションについて説明しています。

第2部 アプリケーション開発編

第3章 CORBAサーバアプリケーションを開発する

CORBAの機能およびCORBAサーバアプリケーションの開発方法について説明しています。

第4章 CORBAクライアントアプリケーションを開発する

CORBAクライアントアプリケーションの開発方法や留意事項について説明しています。

第5章 COBOL/CORBAリモート開発機能

サーバで動作するCOBOLアプリケーションまたはCORBAサーバアプリケーションをリモート開発する手順について説明しています。

第6章 プロジェクトをビルド・デバッグ・実行する

プロジェクトのビルドからデバッグまたは実行までの方法を説明しています。

第7章 アプリケーションを運用する

開発したアプリケーションを実際の運用環境で実行する方法について説明しています。

第3部 ワークベンチ操作編

第8章 ウィザード

プロジェクトやソースファイルを作成する場合に使用するウィザードについて説明しています。

第9章 パースペクティブ

COBOLパースペクティブについて説明しています。

第10章 エディタ

COBOLで使用するエディタについて説明しています。

第11章 ビュー

COBOLで使用するビューについて説明しています。

第4部 チュートリアル編

第12章 CORBAアプリケーションの開発について

CORBAアプリケーションの開発の概要を説明しています。

第13章 CORBAサーバアプリケーション開発

CORBAサーバアプリケーションの作成手順について説明しています。

第14章 CORBAクライアントアプリケーション開発

CORBAクライアントアプリケーションの作成手順について説明しています。

第5部 リファレンス編

第15章 COBOLサービスクラスの概要

COBOLサービスクラスの概要を説明しています。

第16章 CDCORBAクラス

CDCORBAクラスのメソッドについて説明しています。

付録A サンプルの使用方法

サンプルの使用法について説明しています。

付録B Tips

ワークベンチの便利な機能について説明しています。

付録C トラブルシューティング

ワークベンチで発生する問題を解決する方法を説明しています。

付録D 旧資産からの移行

旧資産の移行方法を説明しています。

付録E 名前に関する規則

名前の規則について説明しています。

■著作権

2011年8月 初版

目　次

第1部 導入編

第1章 概要

ここでは、Interstage Studio 互換ワークベンチにCOBOLプラグインをインストールすることによって提供される機能について説明します。Interstage Studioの概要については、"Interstage Studio 解説書"を参照してください。

1.1 COBOLプラグインとは

COBOLプラグインとは、Interstage Studio 互換ワークベンチ上にCOBOLアプリケーションおよびCORBAアプリケーションの開発環境の機能を提供するアドオンコンポーネントです。COBOLプラグインは、以下の機能を提供します。

- COBOLパースペクティブ
- COBOL/CORBA用ウィザード
- エディタ
  - COBOLエディタ
  - IDLエディタ
- ビュー
  - アウトラインビュー
  - テンプレートビュー
  - 依存ビュー
  - 構造ビュー
  - ウォッチビュー
- COBOLビルダ
- COBOLデバッガ
- CDCORBAクラス

COBOLプラグインは、Interstage Business Application Server 開発環境パッケージのCOBOL開発支援ツールをインストールする際にインストールされます。ただし、COBOLプラグインにはヘルププラグインが含まれていないため、Interstage StudioからCOBOL開発環境のヘルプを参照することはできません。

第2章 COBOLプラグインによって開発できるアプリケーション

ここでは、Interstage Studio 互換ワークベンチにCOBOLプラグインをインストールしたときに、Interstage Studioで開発できるアプリケーションについて説明します。

2.1 開発できるアプリケーション

COBOLプラグインをインストールしたInterstage Studioでは、以下のアプリケーションを開発できます。

- COBOLアプリケーション

  COBOL言語記述によるCORBAクライアントなどのアプリケーションを開発できます。COBOLアプリケーションを開発する場合は、「COBOLプロジェクト」を使用してください。

- CORBAサーバアプリケーション

  オブジェクト指向COBOL言語記述によるCORBAサーバアプリケーションを開発できます。CORBAサーバアプリケーションを開発する場合は、「CORBAサーバプロジェクト」を使用してください。

2.1.1 COBOLアプリケーション

COBOLアプリケーションとは、COBOL言語で記述されたCOBOLソースプログラムを、COBOLコンパイラおよびリンカによってビルドして作成するアプリケーションのことを指します。

ワークベンチを使ってCOBOLアプリケーションを開発するためのCOBOLプロジェクトを作成できます。COBOLプロジェクトにはCOBOLソースファイルや登録集ファイルなどを追加します。

COBOLアプリケーションは、ローカルPC上にインストールされたNetCOBOL開発パッケージを使って開発(ローカル開発)することと、サーバにインストールされたNetCOBOL開発パッケージを使ってサーバ上で動作するアプリケーションを開発(リモート開発)できます。

注意

COBOLアプリケーションをローカル開発する場合は、NetCOBOL開発パッケージ製品をローカルPC上に別途インストールする必要があります。COBOLアプリケーションをリモート開発する場合は、サーバ上にNetCOBOL開発パッケージ製品を別途インストールする必要があります。

2.1.2 CORBAアプリケーション

CORBAとは、OMG(Object Management Group:オブジェクト指向技術の標準化と普及を目的として1989年に設立された非営利団体)によって規定されたオブジェクト指向技術の仕様です。CORBA仕様として、以下の機能が提供されています。

- インタフェース定義言語IDL(Interface Definition Language)からサーバ/アプリケーション間の通信ライブラリである、スタブ、スケルトンを生成するしくみ。
- クライアント/サーバアプリケーションで連携するために必要なAPI。
- 異機種間上で動作するクライアント/サーバアプリケーションが連携するためのプロトコル(IIOP:Internet Inter-ORB Protocol)。

上記のCORBA仕様に準拠したアプリケーションを作成するための分散オブジェクト通信基盤は、Interstage CORBAサービスで提供されています。

注意

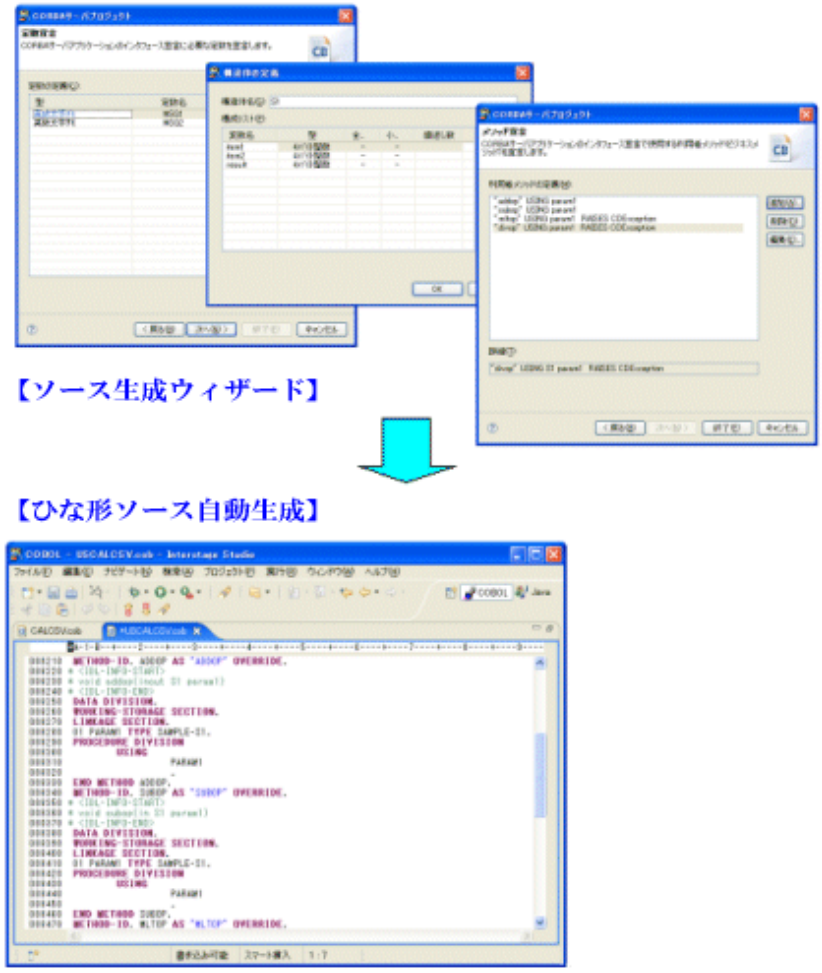
Java EEワークベンチではCORBAアプリケーションの開発をサポートしていません。

2.1.2.1 CORBAサーバアプリケーション開発

COBOLプラグインを組み込んだInterstage Studio 互換ワークベンチでは、COBOLによるCORBAサーバアプリケーションの開発が行えます。

ソース自動生成ウィザードで、サーバとクライアント間のインタフェースを指定し、CORBAサーバアプリケーション開発に必要な、IDLファイルおよびCOBOLプログラムソースのひな形を自動生成します。自動生成したソースに、ビジネスメソッドの処理を記述し、アプリケーションを開発します。

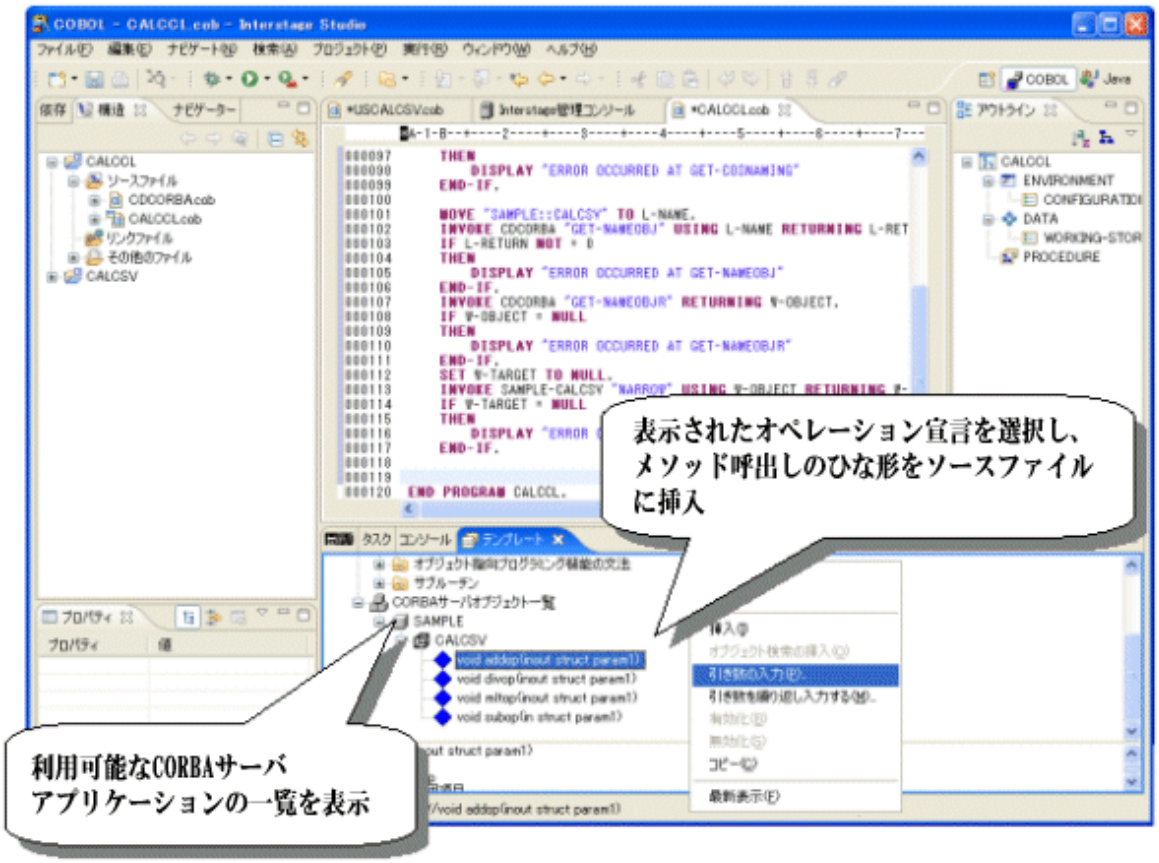
図2.1 CORBAサーバアプリケーション開発



2.1.2.2 CORBAクライアントアプリケーション開発

CORBAクライアントアプリケーションは、JavaまたはCOBOLで開発します。
入力テンプレートに表示されたCORBAサーバアプリケーションから任意のサーバアプリケーションを選択し、入力テンプレートから簡易にそのアプリケーションのメソッド呼出し処理を記述できます。COBOLプラグインを組み込んだInterstage Studio 互換ワークベンチでは、静的インタフェースの使用をサポートします。

図2.2 CORBAクライアントアプリケーション開発



Interstage StudioでのCORBAクライアントアプリケーションの開発は、Javaのプロジェクトまたは COBOLのプロジェクトに、IDLファイルから生成したスタブを追加して行います。

- COBOLのプロジェクトに対しては、CORBAスタブファイル生成ウィザードを使用してIDLファイルからスタブを生成し、プロジェクトに追加できます。
- Javaのプロジェクトに対しては、Interstage Application Serverが提供するIDLcコマンドを用いてスタブを生成し、プロジェクトに手動で追加します。IDLcコマンドの詳細および生成されるスタブについては"Interstage Application Server リファレンスマニュアル(コマンド編)"の"IDLc"の節を参照してください。

2.2 作成できるリソース

COBOLプラグインでは、以下のリソースが作成できます。

カテゴリ	リソース種別		説明
COBOL	COBOLプロジェクト		COBOLアプリケーションを開発するためのプロジェクトを作成します。
	CORBAサーバプロジェクト		CORBAサーバアプリケーションを開発するためのプロジェクトを作成します。
	ソース	COBOLソース	COBOLソースを作成します。
		CORBAサーバアプリケーション	CORBAサーバアプリケーションのひな型やIDLファイルを作成します。
		CORBAスタブファイル	IDLファイルからCORBAクライアントに必要なスタブファイルおよびその他のファイルを作成します。
		IDLファイル	IDLファイルを作成します。
		オブジェクト指向COBOLソース	オブジェクト指向COBOLソースを作成します。

カテゴリ	リソース種別		説明
		COBOL登録集	COBOL登録集ファイルを作成します。

第2部 アプリケーション開発編

第3章 CORBAサーバアプリケーションを開発する

ここでは、CORBAの機能およびCORBAサーバアプリケーションの開発方法について説明します。

3.1 概要

CORBA(Common Object Request Broker Architecture)とは、OMG(Object Management Group ： オブジェクト指向技術の標準化と普及を目的として1989年に設立された非営利団体)によって規定されたオブジェクト指向技術の仕様です。

3.1.1 CORBAの機能

CORBAの仕様として、以下の機能が提供されています。

- インタフェース定義言語IDL(Interface Definition Language)からサーバ/アプリケーション間の通信ライブラリである、スタブ、スケルトンを生成するしくみ
- クライアント/サーバアプリケーションで連携するために必要なAPI
- 異機種間上で動作するクライアント/サーバアプリケーションが連携するためのプロトコル(IIOP：Internet Inter-ORB Protocol)

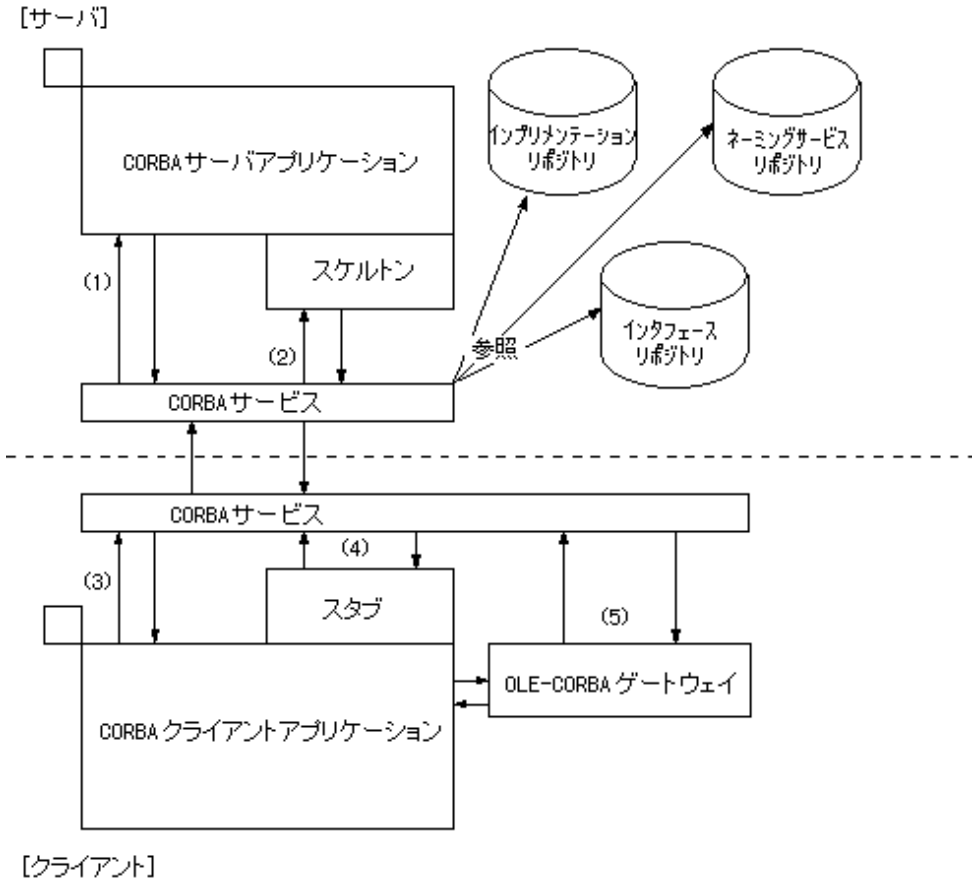
上記のCORBAの仕様に準拠したアプリケーションを作成するための分散オブジェクト通信基盤は、Interstage CORBAサービスで提供されています。以降、CORBAアプリケーションとは、CORBAサービスの提供するサービスおよび機能を使用して作成したアプリケーションのことを指します。
アプリケーションには、サーバで動作するサーバアプリケーションと、サーバアプリケーションを使用するクライアントアプリケーションがあります。ワークベンチでは、CORBAのサーバアプリケーション、クライアントアプリケーションの開発をサポートします。
なお、CORBAサービスの詳細は、以下のマニュアルを参照してください。

- Interstage Application Server チューニングガイド
- Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)
- Interstage Application Server リファレンスマニュアル(コマンド編)
- Interstage Application Server メッセージ集

3.1.2 CORBAアプリケーションの運用形態

CORBAアプリケーションの運用形態を、下図に示します。

図3.1 CORBAアプリケーションの運用形態



(1) 動的スケルトンインタフェース(CORBAサーバアプリケーション)

(2) 静的スケルトンインタフェース(CORBAサーバアプリケーション)

(3) 動的起動インタフェース(CORBAクライアントアプリケーション)

(4) 静的起動インタフェース(CORBAクライアントアプリケーション)

(5) OLE-CORBAゲートウェイ(CORBAクライアントアプリケーション)

COBOLプラグインでサポートしているのは、上記の(2)の静的スケルトンインタフェース(CORBAサーバアプリケーション)および(4)の静的起動インタフェース(CORBAクライアントアプリケーション)です。

上記の(1)～(5)について、以下のアプリケーションの場合に分けて説明します。

- CORBAサーバアプリケーション
- CORBAクライアントアプリケーション

CORBAサーバアプリケーション

CORBAサーバアプリケーションには、実装方式に基づいた以下の分類があります。

動的スケルトンインタフェース

スケルトンファイルをリンクする代わりに、CORBAサービスが提供するAPIを使用して、インタフェース情報やクライアントからの情報を取り出します。

静的スケルトンインタフェース

IDLファイルから生成されたスケルトンファイルをサーバアプリケーションにリンクする方法です。

一般的に、静的スケルトンインタフェースの方が動的スケルトンインタフェースに比べて、リポジトリ情報へのアクセス回数が少ない分、性能的に優れています。

CORBAクライアントアプリケーション

CORBAクライアントアプリケーションには、実装方式に基づいた以下の分類があります。

動的起動インタフェース

スタブファイルをリンクする代わりに、CORBAサービスが提供するAPIを使用してインタフェース情報を取り出し、サーバオブジェクトを呼び出すためのインタフェースを組み立てます。

静的起動インタフェース

IDLファイルから生成されたスタブファイルをクライアントアプリケーションにリンクする方法です。

OLE-CORBAゲートウェイ

OLEサーバ経由でCORBAサーバアプリケーションを使用します。

一般的に、動的起動インタフェースはサーバアプリケーションのインタフェースを動的に組み立てるため、簡単なインタフェースの変更に対して、自プログラムの変更をする必要がない場合があります。そういう点で保守性に優れています。反対に、静的起動インタフェースは、サーバプログラムと同様、サーバのリポジトリ情報にアクセスする回数が動的起動インタフェースに比べて少ないため、性能的に優れています。また、OLE-CORBAゲートウェイは、内部的に動的起動インタフェースと同様の処理を行っているため、性能的には動的起動インタフェースと同等ですが、記述量が非常に少なく、記述性に優れています。

3.1.3 ワークベンチのCORBAサポート範囲

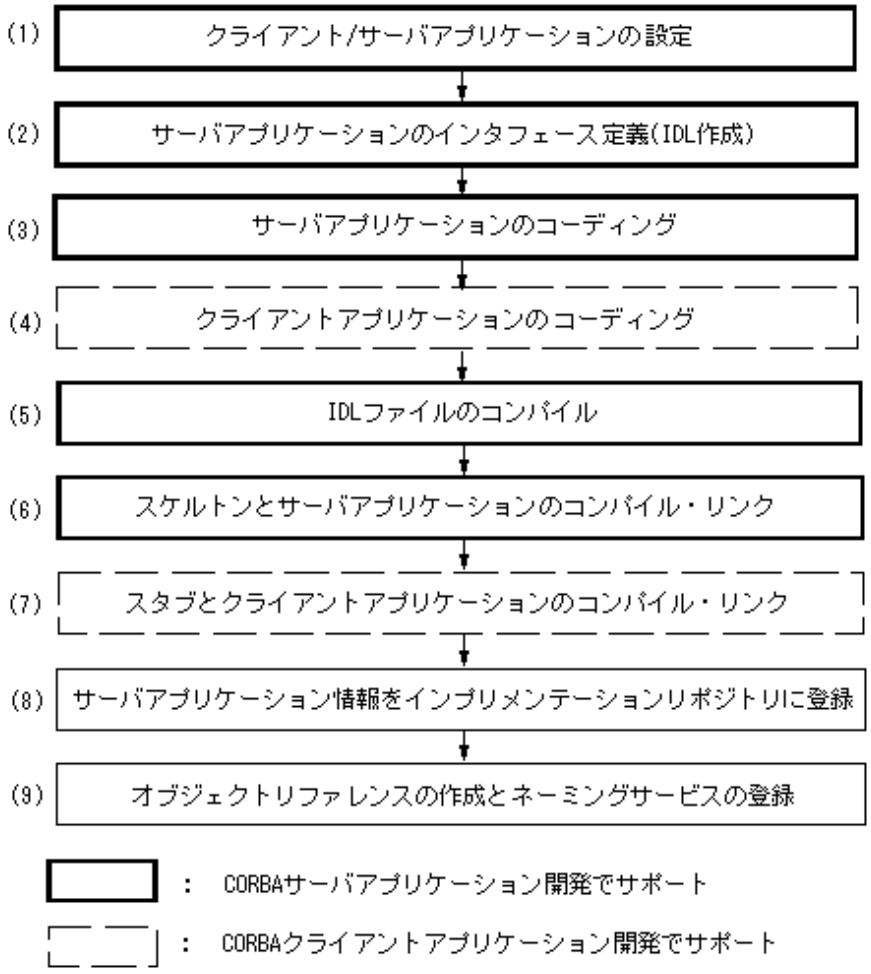
CORBAアプリケーションの開発作業およびワークベンチがサポートする範囲を、以下に示します。

- CORBAアプリケーション開発作業
- 実装形式とサポート言語

CORBAアプリケーション開発作業

CORBAアプリケーションの一般的な開発の流れは、下図のようになります。

図3.2 CORBAアプリケーションの一般的な開発の流れ



(1)～(3)および(5)～(6)をサーバアプリケーション開発で、(4)、(7)をクライアントアプリケーション開発でサポートします。

実装形式とサポート言語

CORBAアプリケーションの開発を以下の実装方式と言語種別でサポートします。

表3.1 実装方式とサポート言語

アプリケーション	実装方式	言語種別
サーバアプリケーション	静的スケルトンインタフェース	オブジェクト指向COBOL
クライアントアプリケーション	静的起動インタフェース	オブジェクト指向COBOL、COBOL

3.2 開発の流れ

ワークベンチでは、以下の順序でCORBAサーバアプリケーションを開発します。

1. CORBAサーバアプリケーション生成ウィザードでひな型を作成する

   サーバアプリケーションのインタフェースを入力します。
   入力項目から、CORBAサーバアプリケーションの作成に必要なIDLファイル(インタフェース定義ファイル)とサーバアプリケーションのひな型を作成します。

2. ビジネスメソッドを実装する

   ビジネスメソッドとは、実際にアプリケーションで実行する処理を記述したメソッド(関数)を指します。CORBAサーバアプリケーション生成ウィザードにより、CORBAサーバアプリケーションが行うべき初期処理のひな型およびビジネスメソッドのひな型は作成されています。そのため、実際に実行したい処理だけコーディングします。

3. ビジネスメソッドを変更する

   CORBAサーバアプリケーション生成ウィザードで作成したひな型プログラムソースに、ビジネスメソッドの追加を行った場合には、同時にIDLファイルにも追加する必要があります。

4. ビルドする

   ビルドでは、以下の処理を一連の流れで行います。

   - IDLファイルのコンパイルにより、スタブ・スケルトンファイルの生成
   - スタブ・スケルトンファイルの翻訳
   - プログラムソースの翻訳
   - リンク

5. プログラムをデバッグする

   ワークベンチの提供するデバッガで、プログラムのデバッグを行います。
   デバッグは、ワークベンチと同一マシンにインストールされているCORBAサービスを使用し、persistentタイププログラムで行います。

CORBAサーバプロジェクトにリモート開発用の設定を追加することによって、スケルトンファイル／プログラムソースの翻訳およびリンク、デバッグをサーバ上で行うことができます。リモート開発への移行は以下の順序で行います。

1. サーバ連携情報を設定する

   リモート開発で使用するサーバのIPアドレス、OSの種別などを設定します。また、各プロジェクトに対して使用するサーバ情報を設定します。

2. メイクファイルを生成する

   翻訳オプションおよびリンクオプションの設定を行い、サーバ上での翻訳・リンク時に使用するメイクファイルを生成します。

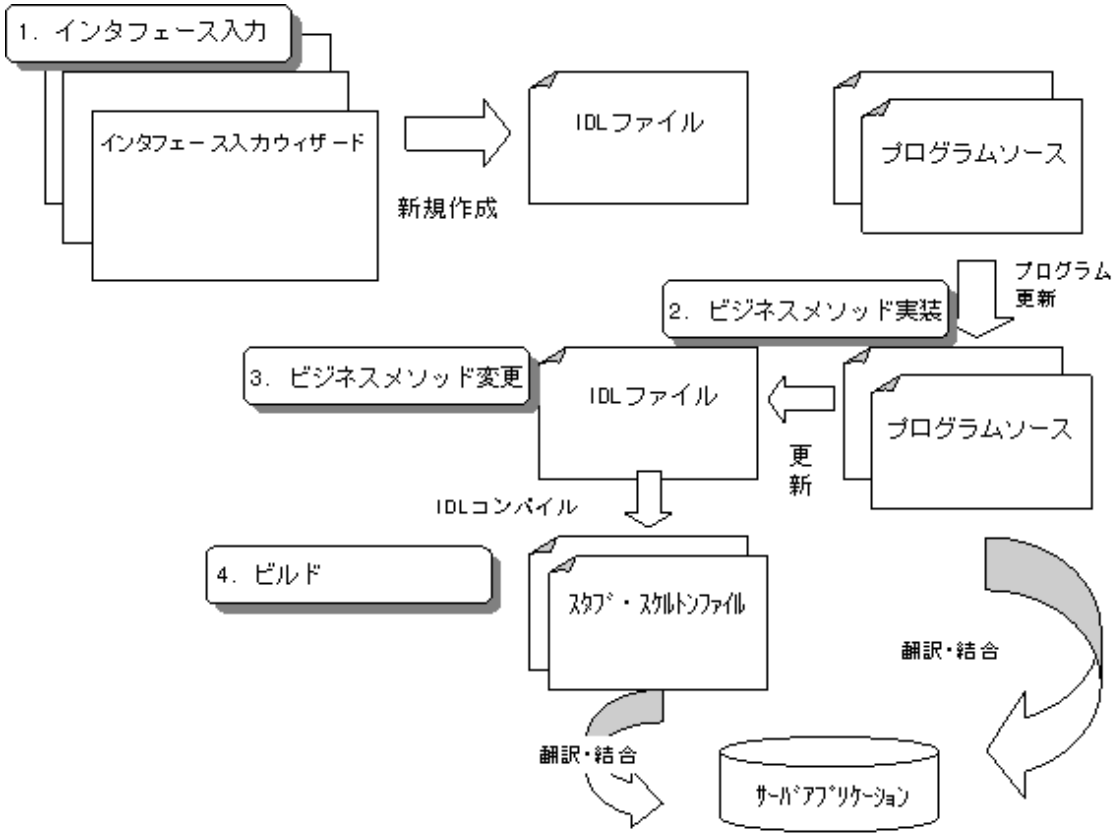
3. リモートビルドする

   生成したメイクファイルおよびプログラム資産をサーバに転送し、サーバ上で翻訳・リンクを行います。

4. リモートデバッグする

   ワークベンチの提供するデバッガで、サーバ上のプログラムをデバッグします。

図3.3 CORBAサーバアプリケーションの開発の流れ



3.2.1 CORBAサーバアプリケーション生成ウィザードでひな型を作成する

CORBAサーバアプリケーションの作成に必要なIDLファイル(インタフェース定義ファイル)とサーバアプリケーションのひな型を作成します。

プロジェクトの新規作成

CORBAサーバアプリケーションを開発するためのプロジェクトを選択します。
メニューから、[ファイル] > [新規] > [プロジェクト]を選択し、[新規プロジェクト]ダイアログボックスから[COBOL] > [CORBAサーバプロジェクト]を選択します。

注意

作成できるアプリケーション

CORBAサーバアプリケーションでは、ターゲット種別としてCORBAサーバだけが作成できます。
実際には、exeファイルとdllファイルが作成されます。

CORBAサーバアプリケーション生成ウィザードによるインタフェース入力

BOAインタフェースを利用したひな型を作成します。各インタフェースの詳細については、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。
ここでは、作成されるひな型の種類、IDLファイルの形式およびCORBAサーバアプリケーション生成ウィザードについて説明します。

- 作成されるひな型の種類
- 生成するIDLファイル形式
- CORBAサーバアプリケーション生成ウィザード

作成されるひな型の種類

以下に作成されるひな型ファイルの種類を説明します。

表3.2 作成されるひな型

生成ファイル	用途
IDLファイル	インタフェース定義言語ファイル
メインプログラムソース	CORBAサーバ初期/終了処理
ビジネスメソッドプログラムソース	ビジネスメソッド定義
サーバアプリケーション登録プログラムソース	サーバアプリケーションの登録

生成するIDLファイル形式

1プロジェクトで1モジュールの定義を行うことができます。モジュール内にビジネスメソッド用のクラスを定義することで、IDLファイルのInterface宣言がされます。IDLファイルの詳細については、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。ここでは、生成するIDLファイル形式およびIDLファイルで生成する宣言について説明します。

```
// モジュール宣言
module M1 {
    // インタフェース宣言
    interface intf1 {
        // 定数宣言
        const long a = 1 ;
        // 型宣言
        typedef sequence<long 10> b ;
        // 構造体宣言
        struct c {
            short item1 ;
            long item2 ;
            long long item3 ;
        } ;
        // 例外宣言
        exception CDException {
            string CDExceptionMsg ;
            long  CDExceptionCode ;
        } ;
        // オペレータ宣言
        long op1(in short param1, out long param2, inout long param3)
        raises ( CDException );
    };
};
```

表3.3 IDLファイルで生成する宣言

IDL定義	説明
モジュール宣言	サーバアプリケーションに対応する宣言です。1つのアプリケーションに対して1個のモジュール定義を行います。
インタフェース宣言	クラスに対応する宣言です。ビジネスメソッド定義用のクラスに対応します。
定数宣言	ビジネスメソッド内で使用する定数を宣言します。
型宣言	ビジネスメソッド内で使用する型の宣言を行います。繰り返し項目の定義に使用します。
構造体宣言	構造体の宣言を行います。オブジェクト指向COBOLでは、集団項目の定義に対応します。 CORBAサーバアプリケーション生成ウィザードでは、構造体の要素の型として構造体を定義することはできません。
例外宣言	例外が発生した場合に通知するレコードの内容を宣言します。実際の例外の通知処理などは、プログラム内でコーディングする必要があります。 以下の要素をオプションで自動生成します。内容は固定なので、変更したい場合は、CORBAサーバアプリケーション生成ウィザードでIDLファイルを生成後に修正してください。

IDL定義	説明

	exception CDException {
	    string CDExceptionMsg ;
	    long CDExceptionCode ;
	} ;

オペレータ宣言	ビジネスメソッドの宣言1個に対して、1個生成されます。また、オプションで例外宣言を選択した場合は、例外が発生した場合に例外宣言で宣言したレコードを返却する定義が生成されます。

CORBAサーバアプリケーション生成ウィザード

CORBAサーバアプリケーション生成ウィザードでは、アプリケーションのインタフェースを決定する項目を入力し、IDLファイルおよびプログラムソースのひな型を生成します。なお、CORBAサーバアプリケーション生成ウィザードで表示される各設定画面の詳細については、"8.1.2 CORBAサーバプロジェクトウィザード"を参照してください。

アプリケーションタイプによる生成ファイル名

生成されるファイルを以下に示します。

生成されるひな型	生成されるファイル名
メインプログラム	インタフェース名.cob
ビジネスメソッドクラス	USインタフェース名.cob
サーバアプリケーション登録プログラム	USINITインタフェース名.cob

3.2.2 ビジネスメソッドを実装する

CORBAサーバアプリケーション生成ウィザードで生成したビジネスメソッドクラスのひな型ソースに、メソッドの実装処理を記述してください。メインプログラムやサーバアプリケーション登録プログラムはそのまま使用可能です。
詳細は、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。
ここでは、「加算プログラム」をCORBAサーバアプリケーション生成ウィザードで定義した例を示します。

加算プログラムのIDLファイル

```
// モジュール宣言
module SAMPLE {
      // ユーザインタフェース宣言
      interface CALCULATE_ADD {
                long CALCULATE(in long param1,
                          in long param2);
      };
};
```

加算プログラムのメインプログラム

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  CALCULATE_ADD.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
      COPY CORBA--REP.
      COPY USCALCULATE_ADD--REP.
 .
 SPECIAL-NAMES.
      SYMBOLIC CONSTANT
      COPY CORBA--CONST.
  .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CORBA--COPY.
 COPY USCALCULATE_ADD--COPY.
 01  API-NAME        PIC   X(50).
```

```
 01  APL-NAME        PIC   X(64)   VALUE     "CALCULATE_ADD".
 01  ORB             USAGE OBJECT REFERENCE CORBA-ORB.
 01  BOA             USAGE OBJECT REFERENCE CORBA-BOA.
 01  IMPL-REP        USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
 01  IMPL            USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
 01  REP-ID          PIC   X(128) VALUE     "IDL:SAMPLE/CALCULATE_ADD:1.0".
 01  OBJ             USAGE OBJECT REFERENCE CORBA-OBJECT.
 01  EXCEPT-ID       USAGE OBJECT REFERENCE CORBA-STRING.
 01  EXCEPT-ID-VALUE PIC   X(50).
 LINKAGE SECTION.
 PROCEDURE DIVISION
                  .
*
*  ORBの初期化
*
      INVOKE CORBA "ORB_INIT" USING APL-NAME FJ-OM_ORBID RETURNING ORB.
*
*  BOAの初期化
*
      INVOKE ORB "BOA_INIT" USING APL-NAME CORBA-BOA_OAID RETURNING BOA.
*
*  インプリメンテーションリポジトリオブジェクトの取得
*
      INVOKE ORB "RESOLVE_INITIAL_REFERENCES" USING CORBA-OBJECTID_IMPLEMENTAT-001 RETURNING OBJ.
      INVOKE FJ-IMPLEMENTATIONREP "NARROW" USING OBJ RETURNING IMPL-REP.
*
*  インプリメンテーション情報の取得
*
      INVOKE IMPL-REP "LOOKUP_ID" USING REP-ID RETURNING OBJ.
      INVOKE CORBA-IMPLEMENTATIONDEF "NARROW" USING OBJ RETURNING IMPL.
*
      SET OBJ TO NULL.
*
*  サーバの活性化をODに通知する
*
      MOVE "CORBA::BOA::IMPL_IS_READY" TO API-NAME.
      INVOKE BOA "IMPL_IS_READY" USING IMPL.
*
      STOP RUN.
 END PROGRAM CALCULATE_ADD.
```

加算プログラムのサーバアプリケーション登録プログラム

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMPLE-CALCULATE_ADD--INIT.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 REPOSITORY.
      CLASS SAMPLE-CALCULATE_ADD
      CLASS SAMPLE-CALCULATE_ADD-IMPL
 .
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 LINKAGE SECTION.
 01 RET USAGE OBJECT REFERENCE SAMPLE-CALCULATE_ADD.
 PROCEDURE DIVISION
         RETURNING RET
                  .
      INVOKE SAMPLE-CALCULATE_ADD-impl "new" RETURNING RET.
      EXIT PROGRAM.
 END PROGRAM SAMPLE-CALCULATE_ADD--INIT.
```

加算プログラムのビジネスメソッドクラス

```
CLASS-ID. SAMPLE-CALCULATE_ADD-IMPL AS "SAMPLE-CALCULATE_ADD-IMPL" INHERITS
    SAMPLE-CALCULATE_ADD
  .
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.
     COPY CORBA--REP.
     COPY USCALCULATE_ADD--REP.
.
SPECIAL-NAMES.
     SYMBOLIC CONSTANT
     COPY CORBA--CONST.
 .
OBJECT.
DATA DIVISION.
WORKING-STORAGE SECTION.
COPY CORBA--COPY.
COPY USCALCULATE_ADD--COPY.
PROCEDURE DIVISION
                  .
METHOD-ID. CALCULATE AS "CALCULATE" OVERRIDE.
* <IDL-INFO-START>
* long CALCULATE(in long param1,in long param2)
* <IDL-INFO-END>
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
01 復帰値 PIC S9(9) COMP-5.
01 PARAM1 PIC S9(9) COMP-5.
01 PARAM2 PIC S9(9) COMP-5.
PROCEDURE DIVISION
        USING
                  PARAM1
                  PARAM2
        RETURNING 復帰値
                  .
     COMPUTE 復帰値 = PARAM1 + PARAM2.
END METHOD CALCULATE.
END OBJECT.
END CLASS SAMPLE-CALCULATE_ADD-IMPL.
```

3.2.3 ビジネスメソッドを変更する

メソッド宣言の変更、追加および削除をした場合は、IDLファイルをエディタで開き、更新する必要があります。なお、IDLファイルの文法については、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。

3.2.4 ビルドする

CORBAサーバプロジェクト作成時に、以下のオプションについては自動的にビルドオプションに設定されます。そのほかのオプションの設定が必要な場合は、プロジェクトの[プロパティ]より、[ビルド]を選択して、オプションを設定してください。
また、ビルド時にはIDLファイルの翻訳(IDLコンパイラ)が行われ、生成されたスケルトンなどサーバアプリケーションに必要なファイルが翻訳対象として追加されます。

翻訳オプション

- Unicodeを使用しない場合

```
LIB(Interstage Application Serverインストールフォルダ¥odwin¥include¥oocob)
REPIN(Interstage Application Serverインストールフォルダ¥odwin¥rep)
THREAD(MULTI)
```

- Unicodeを使用する場合

```
LIB(Interstage Application Serverインストールフォルダ¥odwin¥include¥oocob)
REPIN(Interstage Application Serverインストールフォルダ¥odwin¥rep¥Unicode)
THREAD(MULTI)
RCS(UCS2)
```

リンカオプション

以下のライブラリを結合します。

- Unicodeを使用しない場合

```
Interstage Application Serverインストールフォルダ¥odwin¥lib¥odoocobsv.lib
Interstage Application Serverインストールフォルダ¥odwin¥lib¥odcnsoocob.lib
```

- Unicodeを使用する場合

```
Interstage Application Serverインストールフォルダ¥odwin¥lib¥odoocobsvuc.lib
Interstage Application Serverインストールフォルダ¥odwin¥lib¥odcnsoocobuc.lib
```

注意

ワークベンチでビルドした場合には、Windows上で実行可能な形式でファイルを作成します。Solarisなどで運用する場合には、翻訳やリンクを運用するOS上で行う必要があります。

3.2.5 プログラムをデバッグする

プログラムのデバッグは、クライアントアプリケーションを実際に作成して行います。同一端末にインストールされているInterstage Application Server上で、CORBAワークユニット起動構成を使用してサーバアプリケーションをデバッグします。

デバッグオプションの設定

プログラムをデバッグする場合は、プロジェクトの[プロパティ]で[ターゲット]を選択し、[ビルドモード]で[デバッグ]を指定します。ビルドモードを変更した場合は、プロジェクトの再ビルドをする必要があります。

クライアントプログラムの作成

デバッグするCORBAサーバアプリケーションを呼び出す、CORBAクライアントアプリケーションを作成します。CORBAクライアントアプリケーションの作成方法については、"第4章 CORBAクライアントアプリケーションを開発する"を参照してください。

CORBAワークユニット起動構成でのデバッグ

サーバアプリケーションのデバッグは以下の手順で行います。
詳細については、"6.2 プロジェクトをデバッグする"を参照してください。

1. CORBAワークユニット起動構成でデバッグを開始します。CORBAワークユニット起動構成を使用すると、CORBAワークユニットにアプリケーションが自動配備され、そのあとCORBAワークユニットがデバッグ起動されます。
2. 正常にCORBAワークユニットが起動後、クライアントアプリケーションを実行し、実際にサーバアプリケーションのデバッグを行います。

3.2.6 CORBAサーバアプリケーションをリモート開発する

ローカルPC上で開発していたCORBAサーバプロジェクトにリモート開発用の設定を追加し、サーバ上でCORBAサーバアプリケーションを開発する手順を説明します。

リモート開発機能は、サーバ側でリモート開発機能が必要とするサービスが動作していることが前提となっています。サーバ側の設定については、"5.2 リモート開発のための環境設定"を参照してください。

サーバ連携情報を設定する

リモート開発で使用するサーバの情報を設定します。サーバ連携情報は、ワークスペース内で共有されます。各設定項目の詳細については、"5.2.4 ローカルPC側の環境設定"を参照してください。

サーバ連携情報の設定

以下の手順でサーバ連携情報の設定を行います。

1. メニューから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
2. [設定]ダイアログボックスの左のペインで[COBOL] > [リモート開発]を選択します。[リモート開発]ページが表示されます。
3. [リモート開発]ページで[新規]を選択し、表示された[新規]ダイアログボックスにてサーバ連携情報を設定します。

プロジェクトのサーバ情報設定

以下の手順でリモート開発の情報をプロジェクトに設定します。

1. 依存ビューまたは構造ビューでリモート開発のサーバ情報を設定するプロジェクトを選択します。
2. コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. [プロパティ]ダイアログボックスの左のペインで[リモート開発]を選択して表示される [リモート開発]ページでリモート開発の情報を設定します。

メイクファイルを生成する

リモート開発では、メイクファイルを使用してサーバ上でプログラムをビルドします。リモートビルドを実行する前に翻訳・リンクオプションの設定およびメイクファイル生成を行います。

メイクファイルの生成

以下の手順でメイクファイルを生成します。

1. 依存ビューまたは構造ビューでメイクファイルを生成するプロジェクトを選択します。
2. メニューから[プロジェクト] > [リモート開発] > [メイクファイル生成]を選択するか、コンテキストメニューから[リモート開発] > [メイクファイル生成]を選択します。[メイクファイル生成]ダイアログボックスが表示されます。
3. [メイクファイル生成]ダイアログボックスの[生成条件]にメイクファイル生成時の条件が表示されます。表示された[生成条件]でメイクファイルを生成するのであれば、[OK]をクリックすることによりメイクファイルが生成され、[生成条件]に表示された転送対象のファイルと共にサーバ側に転送されます。表示された[生成条件]を変更したい場合は、[オプション設定]をクリックして[生成条件]の内容を変更してください。

メイクファイル生成条件の変更

[メイクファイル生成]ダイアログボックスに表示されている、以下のメイクファイル生成時の条件を変更できます。変更は、[メイクファイル生成]ダイアログボックスで[オプション設定]をクリックし、[オプション設定]ダイアログボックスの各タブから行います。

- ターゲット名
- プリコンパイラ
- 翻訳オプション
- リンクオプション

[登録集名]タブでは、プロジェクトのプロパティの[ビルド]ページで設定した値の参照だけができます。[登録集名]タブで指定した値はメイクファイルには反映されません。登録集名の指定を有効にするには、サーバ側の環境変数にIN/OFで指定した登録集名を環境変数名とし、その値として登録集ファイルの格納されているディレクトリを設定してください。

注意

CORBAサーバプロジェクトで作成するアプリケーションは静的プログラム構造でなければならないため、DLOAD翻訳オプションは指定しないでください。

リモートビルドする

リモートビルドは以下の手順で実行します。

1. 依存ビューまたは構造ビューでリモートビルドするプロジェクトを選択します。
2. メニューバーから[プロジェクト] > [リモート開発] > [ビルド]を選択するか、コンテキストメニューから[リモート開発] > [ビルド]を選択します。

リモートビルドは前回のビルド以降に変更されたリソースがビルドされます。前回のビルド以降に変更されていないものも含め、すべてのリソースをビルドするには、以下の手順で再ビルドを実行します。

1. 依存ビューまたは構造ビューで再ビルドするプロジェクトを選択します。
2. メニューバーから[プロジェクト] > [リモート開発] > [再ビルド]を選択するか、コンテキストメニューから[リモート開発] > [再ビルド]を選択します。

リモートデバッグする

サーバアプリケーションのリモートデバッグは以下の手順で行います。詳細については、"5.5 リモートデバッグ"を参照してください。

デバッグを開始するにはあらかじめサーバ上でサーバアプリケーションの登録を行っておく必要があります。サーバアプリケーションの登録については、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。

1. サーバ側でアタッチデバッグ用の環境変数を設定し、サーバアプリケーションを起動します。
2. ローカルPC側でリモートCOBOLアプリケーション起動構成を使用したアタッチデバッグを実行します。
3. クライアントアプリケーションを実行し、実際にサーバアプリケーションのデバッグを行います。

3.3 運用

開発したCORBAサーバアプリケーションを実際に運用するためには、以下の作業が必要です。
ここでは、運用環境の設定およびサーバアプリケーションの登録について説明します。

3.3.1 CORBAサーバアプリケーションの各種登録

CORBAサーバアプリケーションを実行するためには、各種リポジトリへの登録処理が必要となります。各種リポジトリ、登録方法の詳細は"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。

インタフェースリポジトリ

インタフェースリポジトリは、以下の場合に必要となります。

- クライアントアプリケーションがOLE-GATEWAY方式の場合
- ワークベンチのCORBAクライアントアプリケーション開発でCORBAサーバオブジェクト一覧機能を使用する場合

インタフェースリポジトリの登録は、登録したいIDLファイルを依存ビューなどで選択し、ポップアップメニューから[インタフェースリポジトリの登録]を選択することで行うことができます。

ビルドオプションとして、IDLcの[インタフェースリポジトリの登録]を指定している場合には、ビルド時に自動的に登録されます。

注意

インタフェースリポジトリの登録は、対象となるインタフェースがすでに存在する場合は、上書きされます。

インプリメンテーションリポジトリ

インプリメンテーションリポジトリの登録は必須です。サーバアプリケーションの実行に必要な情報が格納されます。登録は、Interstage管理コンソールまたはサーバ上のCORBAサービスのコマンドで行います。

ネーミングサービス名

ネーミングサービス名は、オブジェクト指向COBOLで作成したCORBAサーバアプリケーションの場合は必ず登録する必要があります。Interstage管理コンソールなどでワークユニット起動時に登録するように設定する方法と、CORBAサービスコマンドで登録する方法があります。

3.4 留意事項

ここでは、CORBAサーバアプリケーションを開発する際の留意事項を示します。

3.4.1 使用可能なデータ型

以下に、CORBAデータ型と各言語のデータ型との対応を示します。

定義可能なデータ型については、"Interstage Studioユーザーズガイド"を参照してください。

表3.4 COBOL(オブジェクト指向COBOL)マッピング

CORBA型	COBOL(オブジェクト指向COBOL)マッピング型	COBOL Native型
long	CORBA-long	PIC S9(9) COMP-5
unsigned long	CORBA-unsigned-long	PIC 9(9) COMP-5
short	CORBA-short	PIC S9(4) COMP-5
unsigned short	CORBA-unsigned-short	PIC 9(4) COMP-5
long long	CORBA-long-long	PIC S9(18) COMP-5
unsigned long long	CORBA-unsigned-long-long	PIC 9(18) COMP-5
float	CORBA-float	COMP-1
double	CORBA-double	COMP-2
char	CORBA-char	PIC X
wchar	CORBA-wchar	PIC N
octet	CORBA-octet	PIC X
boolean	CORBA-boolean	PIC 1(1)
fixed <m+n,n>	使用できません	PIC S9(m+n,n) PACKED-DECIMAL
string(固定長)	PIC X(n)	PIC X(n)
string(可変長)	使用できません	使用できません
wstring(固定長)	PIC N(n)	PIC N(n)
wstring(可変長)	使用できません	使用できません
enum	CORBA-enum	PIC 9(10) COMP-5
any	使用できません	使用できません
構造体(固定長)	集団項目	集団項目
構造体(可変長)	クラス	クラス

CORBA型	COBOL(オブジェクト指向COBOL)マッピング型	COBOL Native型
共用体	クラス	クラス
sequence型(固定長)	クラス	クラス
sequence型(可変長)	クラス	クラス
array型	使用できません	使用できません

CORBAサーバアプリケーション生成ウィザードでは、以下のデータ型をサポートしていません。これらのデータ型を使用する場合、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照して、ひな型生成後にIDLファイルおよびプログラムソースを修正してください。

- octet
- string(可変長)
- wstring(可変長)
- enum
- 構造体(可変長)
- 共用体
- sequence型(可変長)

3.4.2 繰返し項目の操作

繰返し項目を、以下のようにsequenceデータ型にマッピングします。

表3.5 繰返し項目のマッピング

次元数	オブジェクト指向COBOL
1次元	sequence型(固定長)
2次元～5次元	使用できません

繰り返し定義のIDLファイルへの展開

ここでは、繰り返し定義のIDLファイルへの展開について説明します。

1次元の場合

1次元の場合は、sequence型にマッピングします。

オブジェクト指向COBOLの場合

データ型に4バイト整数、型名にa、繰り返し数に10を指定した場合の展開について示します。

```
typedef sequence<long, 10> a ;
```

繰返し項目の操作

ここでは、繰返し項目の操作について説明します。

1次元の場合(オブジェクト指向COBOL)

繰返し項目の操作を以下に示します。以下のIDLファイルをコンパイルした場合、sequence型は、シーケンスクラスにマッピングされます。

IDLファイル

```
module ODsample {
      typedef sequence<long,10> sampleseq;
      interface seqtest {
               sampleseq op1(in sampleseq param1,
                        out sampleseq param2,
```

```
                        inout sampleseq param3);
    };
};
```

シーケンスクラスのファクトリメソッド、オブジェクトメソッド、プロパティを以下に示します。

表3.6 シーケンスクラス

カテゴリ	メソッド名/プロパティ名	機能
ファクトリメソッド	NEW-WITH-LENGTH	指定の長さでシーケンスクラスを作成します。
オブジェクトメソッド	GET-VALUE	指定番号の要素の値を取得します。
	SET-VALUE	指定番号の要素に値を設定します。
	CLONE	シーケンスクラスを複製します。
プロパティ	SEQ-MAXIMUM	シーケンス最大長の値を取得します。
	SEQ-LENGTH	シーケンス長の値の取得／設定を行います。

繰返し項目の使用例

```
 METHOD-ID. OP1 AS "OP1" OVERRIDE.
* <IDL-INFO-START>
* sampleseq op1(in sampleseq param1,out sampleseq param2,inout sampleseq param3)
* <IDL-INFO-END>
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01 SEQ-VALUE TYPE  CORBA-LONG.
 01 I         TYPE  CORBA-UNSIGNED-LONG.

 LINKAGE SECTION.
 01 復帰値  TYPE ODSAMPLE-SAMPLESEQ.
 01 PARAM1  TYPE ODSAMPLE-SAMPLESEQ.
 01 PARAM2  TYPE ODSAMPLE-SAMPLESEQ.
 01 PARAM3  TYPE ODSAMPLE-SAMPLESEQ.
 PROCEDURE DIVISION
         USING
                  PARAM1
                  PARAM2
                  PARAM3
         RETURNING 復帰値
                  .

*  IN PARAMETER
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
       INVOKE PARAM1 "GET-VALUE" USING I RETURNING SEQ-VALUE
     END-PERFORM.

*  OUT PARAMETER
     INVOKE SEQUENCE-LONG-10 "NEW" RETURNING PARAM2.
     MOVE 10 TO SEQ-LENGTH OF PARAM2.
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
       COMPUTE SEQ-VALUE = I * 100
       INVOKE PARAM2 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.

*  INOUT PARAMETER
     PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
       INVOKE PARAM3 "GET-VALUE" USING I RETURNING SEQ-VALUE
       COMPUTE SEQ-VALUE = SEQ-VALUE * 100
       INVOKE PARAM3 "SET-VALUE" USING I SEQ-VALUE
     END-PERFORM.

*  RESULT
```

```
    INVOKE SEQUENCE-LONG-10 "NEW" RETURNING 復帰値.
    MOVE 10 TO SEQ-LENGTH OF 復帰値.
    PERFORM VARYING I FROM 1 BY 1 UNTIL I > 10
      COMPUTE SEQ-VALUE = I * 10000
      INVOKE 復帰値 "SET-VALUE" USING I SEQ-VALUE
    END-PERFORM.

END METHOD OP1.
```

ポイント

inモードまたはinoutモードのパラメタの場合は、実際のデータ領域はすでに存在するため新たに確保する必要はありません。
outモードのパラメタまたは復帰値の場合には実際の領域をサーバアプリケーションで確保する必要があります。

3.4.3 構造体の操作

構造体を、以下のように集団項目にマッピングします。

表3.7 構造体のマッピング

種別	オブジェクト指向COBOL
構造体(固定長)	集団項目
構造体(可変長)	使用できません

以下のIDLファイルをコンパイルした場合の構造体の使用例を示します。

IDLファイル

```
module ODsample {
      struct samplestruct {
               long item1;
               long item2;
      };
      interface structtest {
               samplestruct op1(in samplestruct param1,
                         out samplestruct param2,
                         inout samplestruct param3);
      };
};
```

構造体項目の例

```
 METHOD-ID. OP1 AS "OP1" OVERRIDE.
* <IDL-INFO-START>
* samplestruct op1(in samplestruct param1,out samplestruct param2,inout samplestruct param3)
* <IDL-INFO-END>
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 LINKAGE SECTION.
 01 復帰値 TYPE ODSAMPLE-SAMPLESTRUCT.
 01 PARAM1 TYPE ODSAMPLE-SAMPLESTRUCT.
 01 PARAM2 TYPE ODSAMPLE-SAMPLESTRUCT.
 01 PARAM3 TYPE ODSAMPLE-SAMPLESTRUCT.
 PROCEDURE DIVISION
         USING
                  PARAM1
                  PARAM2
                  PARAM3
         RETURNING 復帰値
                  .
```

```
*  IN PARAMETER
*  OUT PARAMETER
*  INOUT PARAMETER
     MOVE ITEM1 OF PARAM1 TO ITEM1 OF PARAM2.
     MOVE ITEM2 OF PARAM3 TO ITEM2 OF PARAM2.

     MOVE 2 TO ITEM1 OF PARAM3.
     MOVE 3 TO ITEM2 OF PARAM3.

*  RESULT
     MOVE 4 TO ITEM1 OF 復帰値.
     MOVE 5 TO ITEM2 OF 復帰値.

 END METHOD OP1.
```

3.4.4 継承について

IDL生成機能では、継承の記述および#include文を生成することはできません。
モジュール、インタフェースを継承する場合は、IDLファイルをエディタで編集して継承部分の記述を追加してください。

3.4.5 マルチインスタンスシステムについて

マルチインスタンスシステムで運用するプログラムの場合は、ウィザードで生成したプログラムのORBの初期化部分を修正する必要があります。
マルチインスタンスシステムおよびORB初期化の方法については、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"および"Interstage Application Server リファレンスマニュアル(コマンド編)"を参照してください。

第4章 CORBAクライアントアプリケーションを開発する

ここでは、CORBAクライアントアプリケーションの開発方法や留意事項について説明します。

4.1 概要

CORBA(Common Object Request Broker Architecture)とは、OMG(Object Management Group：オブジェクト指向技術の標準化と普及を目的として1989年に設立された非営利団体)によって規定されたオブジェクト指向技術の仕様です。

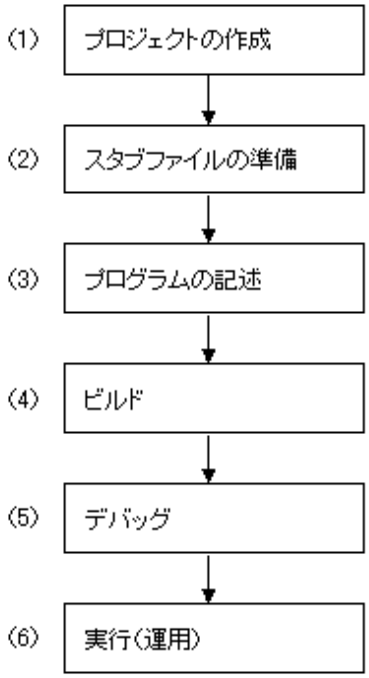
ここでは、CORBAクライアントアプリケーションの開発方法について説明します。なお、CORBAサーバの機能については、"3.1.1 CORBAの機能"を参照してください。

4.2 開発の流れ

以下の順序でCORBAクライアントアプリケーションを開発します。

1. COBOLプロジェクト生成ウィザードでプロジェクトを作成する
2. スタブファイルを準備する
3. CORBAクライアントプログラムを記述する
4. CORBAクライアントアプリケーションをビルドする
5. CORBAクライアントアプリケーションをデバッグする

図4.1 CORBAクライアントアプリケーションの開発の流れ



以降では開発の流れの説明に加え、例として加算プログラムを説明します。
なお、サーバアプリケーションのプログラムは、CORBAサービスのサーバ製品のサンプルプログラムに同等のものがあります。

- サーバアプリケーション

  加算プログラムです。
  2個の入力パラメタを受け取り、その加算結果を復帰値として返却します。

- クライアントアプリケーション

  2個のパラメタをコンソールより入力し、そのパラメタをサーバアプリケーションに渡し、加算結果をコンソールに表示します。

4.2.1 COBOLプロジェクト生成ウィザードでプロジェクトを作成する

CORBAクライアントアプリケーションとなるCOBOLプロジェクトを作成します。

プロジェクトの新規作成

メニューから、[ファイル] > [新規] > [プロジェクト]を選択し、[新規プロジェクト]ダイアログボックスから[COBOL] > [COBOLプロジェクト]を選択します。COBOLプロジェクトウィザードが起動されますので、ウィザードの指示に従い、プロジェクト名とターゲットの定義を行います。

COBOLプロジェクトをCORBAクライアントアプリケーションとして使用するには、次に表示されるウィザードの[ビルド環境]画面で[CORBAクライアントのビルド環境を設定]をチェックします。さらに記述言語（オブジェクト指向COBOLか標準のCOBOLか）、オブジェクトの形式（マルチスレッド上で実行するかシングルスレッド上で実行するか）、およびCDCORBAクラスを使用するかどうかを選択します。

CDCORBAクラスはCORBAクライアントの作成を支援するために本製品が提供しているものです。[CDCORBAクラスを使用する]をチェックすると、新規作成されたプロジェクトにCDCORBAクラスのソースが組み込まれます。CDCORBAクラスの詳細については、"第16章 CDCORBAクラス"を参照してください。

4.2.2 スタブファイルを準備する

静的起動インタフェースプログラムに結合するスタブファイルを準備する必要があります。スタブファイルの準備にはCORBAスタブファイル生成ウィザードを使用します。以下の手順でプロジェクトにスタブファイルを追加してください。

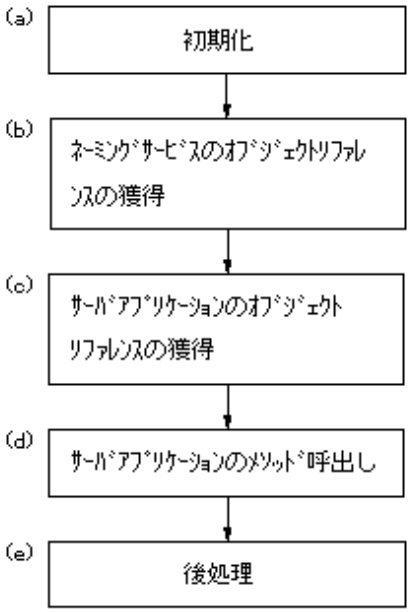
スタブファイルのプロジェクトへの追加方法

1. メニューから、[ファイル] > [新規] > [その他]を選択し、[新規]ダイアログボックスから[COBOL] > [ソース] > [CORBAスタブファイル]を選択します。CORBAスタブファイル生成ウィザードが起動されます。ウィザードの指示に従い、IDLファイルと登録先プロジェクト、生成言語の設定を行います。
2. CORBAスタブファイル生成ウィザードでは、[IDLファイル]で定義したIDLファイルをIDLコンパイラで翻訳し、CORBAクライアントに必要なスタブファイルおよびその他のファイルを生成します。生成されたファイルは[登録先プロジェクト]で指定したプロジェクトに登録されます。

4.2.3 CORBAクライアントプログラムを記述する

一般的なクライアントプログラムの記述内容を、以下に示します。

図4.2 クライアントプログラムの処理順



ここでは、コンソールから入力したパラメタ2個をサーバプログラムに渡し、その結果をコンソールに表示するプログラムの例を示しながら、説明します。

- 対象となるサーバアプリケーション
- オブジェクト指向COBOLプログラム(静的起動)でのプログラム記述例
- COBOLプログラム(静的起動)でのプログラム記述例

対象となるサーバアプリケーション

サーバ側のアプリケーションを以下に示します。

IDLファイル

パラメタを2個受け取り、その加算結果を復帰値として返却するサーバアプリケーションです。ネーミングサービス名は、「Sample::POAintf」で登録されているものとします。

```
// モジュール宣言
module SAMPLE {
   // ユーザインタフェース宣言
   interface CALCULATE_ADD {
           long CALCULATE(in long param1,in long param2);
   };
};
```

オブジェクト指向COBOLプログラム(静的起動)でのプログラム記述例

静的起動インタフェースを使用した場合のオブジェクト指向COBOLプログラム作成上のポイントを、プログラム記述例を使用して説明します。

プログラム作成上のポイント

以下のようにCONFIGURATION SECTIONを記述します。

－ REPOSITORY
COPY CORBA--REP(CORBAサービス提供)
COPY CosNaming--REP(CORBAサービス提供)
COPY IDLファイル名--REP(IDLファイルから生成したもの)
CLASS CDCORBA(本製品提供)

－ SYMBOLIC CONSTANT
COPY CORBA--CONST(CORBAサービス提供)
COPY COSNAMING--CONST.
COPY プロジェクト名--CONST(IDLファイルから生成したもの)

ORBの初期化で使用する変数をWORKING-STORAGE SECTIONで定義します。

以下のような変数を宣言します。

－ W-OBJECT(サーバアプリケーションのオブジェクト取得結果格納用のCORBA-OBJECT型のオブジェクト変数)。

－ W-TARGET(IDLファイルから生成したサーバオブジェクト型のオブジェクト変数)「モジュール名-インタフェース名」で生成されます。

以下の処理を記述します。これらの処理は、テンプレートより入力できます。

－ ORB初期化

－ ネーミングサービス初期化

－ サーバオブジェクト取得

－ ターゲットオブジェクト取得

－ サーバアプリケーションのメソッド呼出し

プログラム記述例

メインプログラム:CALCULATE_ADD_CLIENT.cob

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. MAIN.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.

   REPOSITORY.
* ObjectDirectorの標準登録集（リポジトリ宣言用）
     COPY CORBA--REP.
* ネーミングサービスの標準登録集（リポジトリ宣言用）
     COPY CosNaming--REP.
* IDLコンパイラが出力した登録集（リポジトリ宣言用）
     COPY USCALCULATE_ADD--REP.
* COBOLプラグインが提供しているCORBAクライアント開発用クラス
     CLASS CDCORBA.

  SPECIAL-NAMES.
     ARGUMENT-NUMBER IS 引数番号
     ARGUMENT-VALUE  IS 引数内容
     SYMBOLIC CONSTANT
* ObjectDirectorの標準登録集（定数宣言用）
     COPY CORBA--CONST.
* ネーミングサービスの標準登録集（定数宣言用）
     COPY COSNAMING--CONST.
* IDLコンパイラが出力した登録集（定数宣言用）
     COPY USCALCULATE_ADD--CONST.
     .

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  L-APL-NAME PIC X(50) VALUE "CALCULATE_ADD_CLIENT".
 01 W-OBJECT OBJECT REFERENCE CORBA-OBJECT.
 01 W-TARGET  OBJECT REFERENCE SAMPLE-CALCULATE_ADD.

 01 W-PARAM1 PIC S9(9) COMP-5.
 01 W-PARAM2 PIC S9(9) COMP-5.
 01 W-RETURN PIC S9(9) COMP-5.
 01  L-RETURN PIC S9(9) COMP-5.
```

```
 01  L-NAME PIC X(128) VALUE "SAMPLE::CALCULATE_ADD".

 PROCEDURE DIVISION.
*    ORB初期化
     INVOKE CDCORBA "GET-ORB" USING L-APL-NAME RETURNING L-RETURN.
     IF L-RETURN NOT = 0
     THEN
        DISPLAY "ERROR OCCURRED AT GET-ORB"
     END-IF
*    ネーミングサービス初期化
     INVOKE CDCORBA "GET-COSNAMING" RETURNING L-RETURN.
     IF L-RETURN NOT = 0
     THEN
        DISPLAY "ERROR OCCURRED AT GET-COSNAMING"
     END-IF

     INVOKE CDCORBA "GET-NAMEOBJ" USING L-NAME RETURNING L-RETURN.
     IF L-RETURN NOT = 0
     THEN
        DISPLAY "ERROR OCCURRED AT GET-NAMEOBJ"
     END-IF

*    サーバオブジェクトの取得
     SET W-OBJECT TO NULL.
     INVOKE CDCORBA "GET-NAMEOBJR" RETURNING W-OBJECT.
     IF W-OBJECT = NULL
     THEN
        DISPLAY "ERROR OCCURRED AT GET-NAMEOBJR"
     END-IF

*    ターゲットオブジェクト取得
     SET W-TARGET TO NULL.
     INVOKE SAMPLE-CALCULATE_ADD "NARROW" USING W-OBJECT RETURNING W-TARGET.
     IF W-TARGET = NULL
     THEN
        DISPLAY "ERROR OCCURRED AT NARROW"
     END-IF

*    サーバアプリケーションのメソッドの呼出し
     DISPLAY "第一引数を入力してください：" WITH NO ADVANCING.
     ACCEPT  W-PARAM1.
     DISPLAY "第二引数を入力してください：" WITH NO ADVANCING.
     ACCEPT  W-PARAM2.

     INVOKE W-TARGET "CALCULATE" USING W-PARAM1 W-PARAM2 RETURNING W-RETURN.

     DISPLAY "加算結果：" W-RETURN.

*    実行を終了します。
     EXIT PROGRAM.
 END PROGRAM MAIN.
```

COBOLプログラム(静的起動)でのプログラム記述例

静的起動インタフェースを使用した場合のCOBOLプログラムの記述例を示します。個々の処理記述は異なりますが、処理の流れなどはOOCOBOLの場合と同じです。環境部(ENVIRONMENT DIVISION)、データ部(DATA DIVISION)、個々の処理の詳細については、"Interstage Application Serverアプリケーション作成ガイド(CORBAサービス編)"を参照してください。

プログラム記述例

メインプログラム:CALCULATE_ADD_COBCLIENT.cob

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. MAIN.
```

```
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
   SPECIAL-NAMES.
     ARGUMENT-NUMBER IS ARG-C
     ARGUMENT-VALUE  IS ARG-V
     SYMBOLIC CONSTANT
     COPY SYMBOL-CONST IN CORBA.
     .

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 COPY CONST IN CORBA.
* ORB SETTING PARAMETER
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY CURRENT-ARG-C.
 01 CURRENT-ARG-V.
    02 FILLER OCCURS 6.
       03 CURRENT-ARG-V-VALUE USAGE POINTER.
 01 APLI-NAME PIC X(30) VALUE "CALCULATE_ADD_CLIENT".
 01 TMP-STRING-BUF PIC X(20).
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY ARG-COUNT.

* 文字列長
 01 COPY ULONG IN CORBA REPLACING CORBA-UNSIGNED-LONG BY STRING-LENGTH.

* 作業ポインタ
 01 TEMP-BUF POINTER.

* CORBA-ENVIRONMENT
 01 COPY ENVIRONMENT IN CORBA REPLACING CORBA-ENVIRONMENT BY ENV.

* CORBA-ORB
 01 COPY ORB IN CORBA REPLACING CORBA-ORB BY ORB.

* CORBA-BOA
 01 COPY BOA IN CORBA REPLACING CORBA-BOA BY BOA.

* CORBA-OBJECT(各作業用)
 01 COPY OBJECT IN CORBA REPLACING CORBA-OBJECT BY OBJ.

* 例外発生時の例外ID
 01 MESS PIC X(30).

* Naming Serviceリポジトリ
 01 COPY COSNAMING-NAMINGCONTEXT IN CORBA REPLACING
         COSNAMING-NAMINGCONTEXT BY COS-NAMING.

* ネーミングサービス名
 01 STR-BUF PIC X(30).

* そのほかネーミングコンテキスト操作用
 01 COPY COSNAMING-NAME IN CORBA REPLACING COSNAMING-NAME BY NAME.
 01 NAME-A USAGE POINTER.
 01 COPY COSNAMING-NAMECOMPONENT IN CORBA REPLACING
         COSNAMING-NAMECOMPONENT BY NAME-COMPONENT.
 01 NAME-COMPONENT-A USAGE POINTER.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY NUM.

* メソッドの復帰値
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY RET.
* メソッドのパラメタ
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM1.
 01 COPY LONG IN CORBA REPLACING CORBA-LONG BY PARAM2.
```

```
 PROCEDURE DIVISION.
* サーバアプリ起動パラメタの設定
* 起動パラメタの最初には、クライアントアプリケーション名を設定
     ACCEPT CURRENT-ARG-C FROM ARG-C.
     COMPUTE CURRENT-ARG-C = CURRENT-ARG-C + 1.
     PERFORM VARYING ARG-COUNT FROM 1 BY 1 UNTIL ARG-COUNT > CURRENT-ARG-C
        IF ARG-COUNT = 1
           MOVE APLI-NAME TO TMP-STRING-BUF
        ELSE
           ACCEPT TMP-STRING-BUF FROM ARG-V
        END-IF
        MOVE FUNCTION LENG (TMP-STRING-BUF) TO STRING-LENGTH
        CALL "CORBA-STRING-SET" USING
              CURRENT-ARG-V-VALUE (ARG-COUNT)
              STRING-LENGTH
              TMP-STRING-BUF
     END-PERFORM.
     SET CURRENT-ARG-V-VALUE (ARG-COUNT) TO NULL.

*
*    ORBの初期化
*    (CORBA-ORB-INIT,CORBA-ORB-BOA-INIT)
*
     MOVE       12 TO STRING-LENGTH.
     CALL       "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           FJ-OM-ORB-ID.
     CALL "CORBA-ORB-INIT" USING
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           ORB.
     CALL "CORBA-FREE" USING TEMP-BUF.
     PERFORM ENV-CHECK.

     MOVE 15 TO STRING-LENGTH.
     CALL       "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-BOA-OA-ID.
     CALL "CORBA-ORB-BOA-INIT" USING
           ORB
           CURRENT-ARG-C
           CURRENT-ARG-V
           TEMP-BUF
           ENV
           BOA.
     CALL "CORBA-FREE" USING TEMP-BUF.
     PERFORM ENV-CHECK.

*
*    Naming Serviceのリポジトリの獲得
*
     MOVE FUNCTION LENG ( CORBA-ORB-OBJECTID-NAMESERVICE ) TO STRING-LENGTH.
     CALL "CORBA-STRING-SET" USING
           TEMP-BUF
           STRING-LENGTH
           CORBA-ORB-OBJECTID-NAMESERVICE.
     CALL "CORBA-ORB-RESOLVE-INITIAL-REFERENCES" USING
           ORB
           TEMP-BUF
```

```
           ENV
           COS-NAMING.
      CALL "CORBA-FREE" USING TEMP-BUF.
      PERFORM ENV-CHECK.

*
*     サーバアプリケーションを検索
*
      MOVE FUNCTION LENG (STR-BUF) TO STRING-LENGTH.
      MOVE "SAMPLE::CALCULATE_ADD" TO STR-BUF.
      CALL "CORBA-STRING-SET" USING
           IDL-ID OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
      MOVE " " TO STR-BUF.
      CALL "CORBA-STRING-SET" USING
           KIND OF NAME-COMPONENT
           STRING-LENGTH
           STR-BUF.
      MOVE 1 TO SEQ-LENGTH OF NAME.
      MOVE 1 TO SEQ-MAXIMUM OF NAME.
      MOVE 1 TO NUM.
      CALL "CORBA-SEQUENCE-COSNAMING-NAMECOMPONENT-ALLOCBUF" USING
           SEQ-MAXIMUM OF NAME
           SEQ-BUFFER OF NAME.
      MOVE FUNCTION ADDR ( NAME ) TO NAME-A.
      MOVE FUNCTION ADDR ( NAME-COMPONENT ) TO NAME-COMPONENT-A.
      CALL "CORBA-SEQUENCE-ELEMENT-SET" USING
           NAME-A
           NUM
           NAME-COMPONENT-A.
      CALL "COSNAMING-NAMINGCONTEXT-RESOLVE" USING
              COS-NAMING
              NAME
              ENV
              OBJ.
      MOVE "COSNAMING-NAMINGCONTEXT-RESOLVE" TO MESS.
      PERFORM ENV-CHECK.

*
*     サーバオブジェクトの呼出し
*     サンプル名-インタフェース名-オペレータ名
*

*     加算
      DISPLAY "第一引数を入力してください：" WITH NO ADVANCING.
      ACCEPT  PARAM1.
      DISPLAY "第二引数を入力してください：" WITH NO ADVANCING.
      ACCEPT  PARAM2.

      CALL "SAMPLE-CALCULATE-ADD-CALCULATE" USING
           OBJ
           PARAM1
           PARAM2
           ENV
           RET.
      PERFORM ENV-CHECK.

      DISPLAY "加算結果：" RET.
*
*     後処理
*
```

```
     CALL "CORBA-OBJECT-RELEASE" USING OBJ ENV.
     CALL "CORBA-OBJECT-RELEASE" USING COS-NAMING ENV.
     PERFORM ENV-CHECK.
     CALL "CORBA-FREE" USING SEQ-BUFFER OF NAME.
*
* CORBA-ENVIRONMENTの情報を参照して例外が発生しているかどうか
* チェック
*
 ENV-CHECK SECTION.
     EVALUATE TRUE
         WHEN CORBA-NO-EXCEPTION OF MAJOR OF ENV
              CONTINUE
         WHEN CORBA-USER-EXCEPTION OF MAJOR OF ENV
              DISPLAY "USER-EXCEPTION   : "
              MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
              CALL "CORBA-STRING-GET" USING
                                  IDL-ID OF ENV
                                  STRING-LENGTH
                                  MESS
              DISPLAY "ID : " MESS
              EXIT PROGRAM
         WHEN CORBA-SYSTEM-EXCEPTION OF MAJOR OF ENV
              DISPLAY "SYSTEM-EXCEPTION : "
              MOVE FUNCTION LENG (MESS) TO STRING-LENGTH
              CALL "CORBA-STRING-GET" USING
                                  IDL-ID OF ENV
                                  STRING-LENGTH
                                  MESS
              DISPLAY "ID : " MESS
              EXIT PROGRAM
     END-EVALUATE.
 ENV-CHECK-END.
     EXIT.

 END PROGRAM MAIN.
```

4.2.4 CORBAクライアントアプリケーションをビルドする

静的起動インタフェースの場合は、CORBAクライアントアプリケーションをビルドする前に、スタブファイルを準備し、プロジェクトに登録する必要があります。詳細は、"4.2.2 スタブファイルを準備する"を参照してください。

オブジェクト指向COBOL静的起動インタフェースのビルドに必要な設定

オブジェクト指向COBOL静的起動インタフェースの場合、以下のビルドオプションを設定する必要があります。

なお、COBOLプロジェクト作成時に[CORBAクライアントのビルド環境を設定]をチェックした場合には、これらのビルドオプションはプロジェクトに自動的に設定されます。

翻訳オプション

```
LIB(Interstage Application Serverインストールフォルダ¥odwin¥include¥oocob)
REPIN(Interstage Application Serverインストールフォルダ¥odwin¥rep)
THREAD(MULTI)またはTHREAD(SINGLE)
```

リンカオプション

以下のライブラリを結合します。

```
Interstage Application Serverインストールフォルダ¥odwin¥lib¥odoocob.lib
Interstage Application Serverインストールフォルダ¥odwin¥lib¥odcnoocob.lib
```

COBOL静的起動インタフェースのビルドに必要な設定

COBOL静的起動インタフェースの場合、以下のビルドオプションを設定する必要があります。

なお、COBOLプロジェクト作成時に[CORBAクライアントのビルド環境を設定]をチェックした場合には、これらのビルドオプションはプロジェクトに自動的に設定されます。

翻訳オプション

```
THREAD(MULTI)またはTHREAD(SINGLE)
```

COBOL登録集名

```
CORBA=Interstage Application Serverインストールフォルダ¥odwin¥include¥cobol
```

リンカオプション

以下のライブラリを結合します。

```
Interstage Application Serverインストールフォルダ¥odwin¥lib¥odcobcbl.lib  …  シングルスレッドの場合
Interstage Application Serverインストールフォルダ¥odwin¥lib¥odcobcblmt.lib  …  マルチスレッドの場合
```

4.2.5 CORBAクライアントアプリケーションをデバッグする

CORBAクライアントオブジェクトのデバッグはCOBOLアプリケーション起動構成を使用して行います。あらかじめ、プロジェクトの[プロパティ]で[ターゲット]を選択し、[ビルドモード]で[デバッグ]を指定してプロジェクトのビルドを行ってください。

注意

CORBAサーバオブジェクトのデバッグ

CORBAサーバオブジェクトのデバッグは、リモートCOBOLアプリケーション起動構成を使用します。

COBOLアプリケーション起動構成でのデバッグ方法については、"6.2 プロジェクトをデバッグする"を参照してください。

4.3 運用

CORBAクライアントアプリケーションを実行する前に、以下の環境設定を確認してください。

4.3.1 共通事項

実装方式に依存しない共通事項を以下に示します。詳細は、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。アプリケーション起動時の留意事項については、ソフトウェア説明書を参照してください。

- サーバアプリケーションタイプがPERSISTENTの場合など、事前に起動されていなければならない場合は、サーバ管理者などにサーバアプリケーションの起動を依頼してください。
- クライアントアプリケーションには直接関係ありませんが、サーバアプリケーションが動作するためには、インプリメンテーションリポジトリ情報がサーバで登録されていなければなりません。
- CORBAクライアントアプリケーションからサーバオブジェクトを参照する場合は、ネーミングサービスリポジトリに、サーバオブジェクトの情報が登録されていなければなりません。
- Interstage Application Serverのクライアント機能をインストールしている場合、使用するネーミングサーバをCORBAサービスの提供するファイル"inithost"で指定します。

  ー ネーミングサーバの指定例1：

  Interstage Application Serverが、"c:¥interstage¥APS"にインストールされており、かつ、サーバ"SERV01"のネーミングサービスを使用する場合は、"c:¥interstage¥APS¥odwin¥etc¥inithost"に以下の定義が必要になります。

```
SERV01    8002
  .       .
  .       .
```

－ ネーミングサーバの指定例2：
マルチインスタンスシステムのサーバに接続する場合は、接続するシステムはポート番号によって決定されます。

```
SERV01    8002    サーバ SERV01のインスタンス1へアクセス
SERV01    8003    サーバ SERV01のインスタンス2へアクセス
```

4.3.2 オブジェクト指向COBOL静的起動インタフェースの環境設定

オブジェクト指向COBOL静的起動インタフェースの場合に必要な環境設定について、以下に示します。

・ 本製品のランタイムシステム、オブジェクト指向COBOLランタイムシステムがインストールされているフォルダが、環境変数"PATH"に設定されている必要があります。

4.3.3 COBOL静的起動インタフェースの環境設定

COBOL静的起動インタフェースの場合に必要な環境設定について、以下に示します。

・ COBOLランタイムシステムがインストールされているフォルダが、環境変数"PATH"に設定されている必要があります。

4.4 留意事項

CORBAクライアントアプリケーションを開発する際の留意事項を示します。

4.4.1 CORBAのデータ型

CORBAのデータ型については、"3.4.1 使用可能なデータ型"を参照してください。

第5章 COBOL/CORBAリモート開発機能

ここでは、サーバで動作するCOBOLアプリケーションまたはCORBAサーバアプリケーションをリモート開発する手順について説明します。また、特に断りのない限り、COBOLアプリケーションおよびCORBAサーバアプリケーション共通の手順としてCOBOLアプリケーションの場合を例に説明します。

リモート開発を行うためには、サーバ側のオペレーティングシステム向けのNetCOBOLがインストールされている必要があります。サーバにインストールするNetCOBOLのバージョンおよびEditionは、サーバ製品のインストールガイドを参照してください。

オペレーティングシステム名	CPUアーキテクチャ
Windows Server 2003 R2	Itanium
	x64
Windows Server 2008	Itanium
	x64
Solaris	SPARC
Red Hat Enterprise Linux AS/ES	x86
Red Hat Enterprise Linux ES	Itanium
	x64

NetCOBOLの各プラットフォーム向けの製品は、基本的に同じCOBOLの言語仕様を提供しています。
このため、多くの場合は、Windowsシステム上で開発したアプリケーション資産を使用して、同じ動作をする各サーバのアプリケーションを作成できます。
NetCOBOLでは、この言語の基本的な機能だけではなく、リモート開発を支援するために次のような機能を提供しています。

- メイクファイル生成機能

  サーバでCOBOLアプリケーションをビルドするために必要となるメイクファイルを生成する機能です。

- ビルド機能

  サーバでCOBOLアプリケーションを翻訳・リンクするための機能です。

- リモートデバッグ機能

  サーバ上で動作するCOBOLアプリケーションをソースレベルでデバッグするための対話型デバッガです。

5.1 リモート開発の流れ

新規にサーバで動作するアプリケーションを開発する、またはすでにWindowsシステムで稼働しているCOBOLアプリケーションをサーバ側シテスムに移植するために、リモート開発を適用する場合の開発手順について説明します。

プログラミング(ソース、登録集、定義体、オーバレイ)

COBOLソースを始めとする各種アプリケーション資産をローカルPC上で作成・更新します。

- COBOLソースプログラム
- COBOL登録集原文(COPY句)
- 画面帳票定義体
- オーバレイパターン

この際、作成・更新するアプリケーション資産をワークベンチのCOBOLプロジェクトまたはCORBAサーバプロジェクトに登録します。この登録情報を元にサーバ側での翻訳・リンク用のメイクファイルを生成できます。

翻訳・リンク(構文チェック)

ワークベンチを使用して、作成・更新したアプリケーション資産を翻訳・リンクします。この作業は、次のような目的で行います。

ー 作成したアプリケーション資産に誤りや矛盾がないことをまず確認する。

ー ワークベンチに登録したアプリケーション資産の依存関係をチェックする。

ー 単体テスト用の実行形式プログラムを作成する。

単体テスト

ローカルPCで翻訳・リンクしたアプリケーションを使用して、そのアプリケーションに閉じた範囲の機能をテストします。デバッグ機能用の翻訳オプション (CHECK、COUNT、TRACE)とワークベンチのCOBOLデバッガを使用して、ローカルPC上でアプリケーションの誤りを発見できます。

サーバ側での翻訳・リンク

ローカルPC上で翻訳・リンクしたアプリケーションはサーバ側では動作しません。このため、ローカルPC上で作成・更新したアプリケーション資産をサーバへ転送して、サーバ側のNetCOBOLを用いて、改めて翻訳・リンクします。

サーバ側での翻訳・リンクに必要となる以下の操作は、ワークベンチで提供されます。

ー プリコンパイラ連携

ー メイクファイル生成

ー リモートビルド

ただし、これら機能を使用するに先立って、以下の設定が必要となります。

ー リモート開発の環境設定

ローカルPC上で作成・更新したアプリケーション資産のサーバへの転送は、メイクファイル生成、リモートビルドの実行時に自動的に行われます。

サーバ側のアプリケーションのデバッグ

サーバ側で翻訳・リンクしたアプリケーションのデバッグは、ワークベンチから呼び出される対話型デバッガによりリモートデバッグします。

5.2 リモート開発のための環境設定

ここでは、リモート開発環境構築に必要な次の設定の詳細を説明します。

サーバへのサービスの導入と起動

リモート開発機能は、サーバシステム側でリモート開発機能が必要とするサービスが動作していることが前提となっています。したがって、これらのサービスの導入と起動が必要です。

サーバ側に導入が必要なサービスは、サーバのOSおよびNetCOBOLの製品バージョンで決まります。

- サーバのOSがSolaris、Linux(x86)またはLinux(Itanium)で、かつ、サーバのNetCOBOLの製品バージョンがV10以前の場合
  - ー ftpd
  - ー rexec
- 上記以外
  - ー NetCOBOLリモート開発サービス

また、サーバのOSがSolarisまたはLinuxの場合、リモートデバッグ機能を使用するには、次のサービスも必要となります。

- telnetd

サーバ側のユーザ環境の設定

リモート開発の各機能でサーバ側のユーザ環境に定義された環境変数の値を参照するため、その値を設定する必要があります。

ローカルPC側の環境設定

各開発者が使用するInterstage Studioでサーバと連携するための情報を設定する必要があります。

注意

"サーバへのサービスの導入と起動"および"サーバ側のユーザ環境の設定"の操作を行う場合、サーバの管理者権限が必要となります。

5.2.1 サーバへのNetCOBOLリモート開発サービスの導入と起動

ここではサーバへのNetCOBOLリモート開発サービスの導入と起動方法について説明します。

注意

サーバのOSがSolaris、Linux(x86)またはLinux(Itanium)で、かつ、サーバのNetCOBOLの製品バージョンがV10以前の場合はftpd/rexecサービスを使用してください。これ以外の場合はNetCOBOLリモート開発サービスを使用してください。

5.2.1.1 Solarisサーバの場合

Solarisサーバの場合、次のサービスの導入と起動が必要です。

- NetCOBOLリモート開発サービス

NetCOBOLリモート開発サービスは、サーバ上にNetCOBOL 開発・運用パッケージをインストールすると自動的にインストールされます。ただし、セキュリティ上の配慮から、デフォルトではサービスが自動的に起動しないように設定されています。

NetCOBOLリモート開発サービスの起動方法は、サーバの"NetCOBOL 使用手引書"を参照してください。

5.2.1.2 Linuxサーバの場合

Linuxサーバの場合、次のサービスの導入と起動が必要です。

- NetCOBOLリモート開発サービス

NetCOBOLリモート開発サービスは、サーバ上にNetCOBOL 開発・運用パッケージをインストールすると自動的にインストールされます。ただし、セキュリティ上の配慮から、デフォルトではサービスが自動的に起動しないように設定されています。

NetCOBOLリモート開発サービスの起動方法は、サーバの"NetCOBOL 使用手引書"を参照してください。

5.2.1.3 Windowsサーバの場合

Windowsサーバの場合、次のサービスの導入と起動が必要です。

- NetCOBOLリモート開発サービス

NetCOBOLリモート開発サービスは、サーバ上にNetCOBOL 開発・運用パッケージをインストールすると自動的にインストールされます。ただし、セキュリティ上の配慮から、デフォルトではサービスが自動的に起動しないように設定されています。

NetCOBOLリモート開発サービスを起動または停止させる

NetCOBOLリモート開発サービスを起動または停止するには、以下の手順を行ってください。

1. NetCOBOLリモート開発サービスを実行しているWindowsに管理者アカウントでログオンし、[スタート] > [管理ツール] > [サービス]を選択します。
2. サービスの一覧から[NetCOBOL Remote Development Services]を選択します。
3. サービスを開始する場合は、[操作] > [すべてのタスク] > [開始]を選択します。サービスを停止する場合は、[操作] > [すべてのタスク] > [停止]を選択します。

注意

Windowsファイアウォールが有効になっている場合、NetCOBOLリモート開発サービスを起動しても、ローカルPCからの通信がブロックされ、NetCOBOLリモート開発サービスに届きません。この場合は、[コントロールパネル]の[Windowsファイアウォール]アプレットを使用して、NetCOBOLリモート開発サービスの通信を通すように設定してください。NetCOBOLリモート開発サービスはデフォルトではTCP

の61999番ポートを使用します。また、Windowsファイアウォールの設定を変更した場合は、開発期間が終了したあとに設定を元に戻してください。

NetCOBOLリモート開発サービスを自動起動するように設定する

サーバ上のWindowsの起動時に、NetCOBOLリモート開発サービスが自動的に起動するように設定することもできます。そのように設定する場合は、以下の手順を行ってください。

1. NetCOBOLリモート開発サービスを実行しているWindowsに管理者アカウントでログオンし、[スタート] > [管理ツール] > [サービス]を選択します。
2. サービスの一覧から[NetCOBOL Remote Development Services]を選択し、[操作] > [プロパティ]を選択します。
3. [NetCOBOL Remote Development Servicesのプロパティ]ダイアログボックスの[全般]タブの[スタートアップの種類]を[自動]に変更します。

注意

一般的に、サーバ上で不要なサービスを実行するのはセキュリティ上好ましくありません。NetCOBOLリモート開発サービスを自動起動するように設定した場合は、開発期間が終了したあとに、サービスが自動起動しないように[スタートアップの種類]を[手動]に戻してください。

NetCOBOLリモート開発サービスの使用するポートを変更する

NetCOBOLリモート開発サービスはデフォルトでTCPの61999番ポートを使用します。ポート番号を変更する場合は、以下の手順を行ってください。

1. NetCOBOLリモート開発サービスを実行しているWindowsに管理者アカウントでログオンし、[スタート] > [管理ツール] > [サービス]を選択します。
2. サービスの一覧から[NetCOBOL Remote Development Services]を選択し、[操作] > [プロパティ]を選択します。
3. [NetCOBOL Remote Development Servicesのプロパティ]ダイアログボックスの[全般]タブの[開始パラメータ]に「/port:<ポート番号>」と記入します。例えば、1234番ポートを指定する場合は、「/port:1234」と記述します。[開始パラメータ]はサービスが実行中である場合は変更できません。サービスが実行中である場合は一度サービスを停止させてください。

ポートを変更する場合は他のネットワークプログラムが使用していないポート番号を選択してください。

注意

Windowsサーバのリモート開発の場合、WindowsサーバにWindowsアプリケーションを開発するためのソフトウェア開発キット(SDK)が必要になります。必要となるソフトウェア開発キットについては、サーバ側NetCOBOLの"ソフトウェア説明書"を参照してください。

5.2.2 サーバへのftpd/rexecサービスの導入と起動

ここではサーバへのftpdおよびrexecサービスの導入と起動方法について説明します。

注意

サーバのOSがSolaris、Linux(x86)またはLinux(Itanium)で、かつ、サーバのNetCOBOLの製品バージョンがV10以前の場合はftpd/rexecサービスを使用します。これ以外の場合はNetCOBOLリモート開発サービスを使用してください。

5.2.2.1 Solarisサーバの場合

Solarisサーバの場合、ftpdおよびrexecサービスは、デフォルトではオペレーティングシステムの導入時にインストールされ、常に起動するようになっています。

注意

Solaris10の場合は、rexecサービスはオペレーティングシステムの導入時に起動するようになっていません。次の操作が必要になります。

- ステータスの確認

  次のコマンドを実行してステータスを確認してください。

```
# svcs -a | grep rexec
disabled   18:28:10 svc:/network/rexec:default
```

  実行結果の出力の先頭が"disabled"である場合は、rexecサービスを起動してください。

- rexecサービスの起動

  次のコマンドを実行してrexecサービスを起動してください。

```
# svcadm enable svc:/network/rexec:default
```

Solarisサーバの場合、システム設定の変更は設定ファイルの内容を直接確認し、必要であればそれを修正することが一般的ですので、その方法について説明します。

1. サービスの状態の確認

   Solarisサーバではftpdおよびrexecは、inetd(インターネットデーモン)から呼び出されるサービスです。したがって、次のファイルの内容を確認します。

   ー /etc/services

   ー /etc/inetd.conf

   "/etc/services"および"/etc/inetd.conf"内のftpdおよびrexecに関する記述が存在し、それが有効であれば、以降の作業は必要ありません。次にその例を示します。

   ー /etc/servicesの例

```
#
# Network services, Internet style
#
…
ftp 21/tcp
…
## UNIX specific services
## these are NOT officially assigned
#
```

   ー /etc/inetd.confの例

```

…
# FTPD - FTP server daemon
ftp stream tcp6 nowait root /usr/sbin/in.ftpd in.ftpd -a
…
# REXECD - rexec daemon (BSD protocols)
exec stream tcp nowait root /usr/sbin/in.rexecd in.rexecd
exec stream tcp6 nowait root /usr/sbin/in.rexecd in.rexecd
…
```

   ftpdまたはrexecの設定行がコメント化(行頭に"#")されている場合は、以降の作業を行ってください。

2. サービスの設定の変更

   "/etc/services"および"/etc/inetd.conf"を修正します。

3. サービスの起動

   ftpdおよびrexecは、inetd配下で起動されるサービスであるため、inetdを再起動します。

次のコマンドを実行してください。

```
# kill -HUP `cat /var/run/inetd.pid`

または

# ps -ea | grep inetdinetdのプロセスIDが表示される。
# kill -HUP inetdのプロセスID
```

5.2.2.2 Linuxサーバの場合

Linuxサーバの場合、ftpdおよびrexecサービスは、オペレーティングシステムの導入時にインストールされていない場合もあるため、これらのパッケージが導入済みかどうかを確認します。
なお、Linuxでは、この種のシステム設定のためにGUIを持つツールが用意されている場合がありますが、GUIツールはバージョンおよび個々のシステムの設定による違いが大きいため、コマンドによる操作方法について説明します。

1. パッケージの確認

   パッケージがインストール済みか確認するには次の形式でrpmコマンドを実行します。

   ```
   # rpm -query パッケージ名
   ```

   ftpdおよびrexecに必要なパッケージ名は次のとおりです。

   - ftpd : vsftpd
   - rexec : rshおよびrsh-server

   次に示すようにインストールされているパッケージの情報が表示されている場合、パッケージはインストール済みです。サービスの状態の確認に進んでください。

   ```
   # rpm -query vsftpd
   vsftpd-2.0.1-5
   # rpm -query rsh
   rsh-0.17-17
   # rpm -query rsh-server
   rsh-server0.17-17
   ```

   パッケージの情報が表示されない場合、パッケージのインストールを行う必要があります。

   注意

   Linuxシステムで使用されるftpdのパッケージには、バージョンやディストリビューションの違いによりいくつか種類があります。次のようなものが使われている場合もあります。

   - wu-ftpd
   - proftpd

2. パッケージの導入

   次のrpmコマンドを使用してパッケージを導入します。

   ```
   rpm -Uvh パッケージ名
   ```

3. サービスの状態の確認

   /sbin/chkconfigコマンドを次の形式で使用して、システム起動時のサービス開始の設定を確認します。

   ```
   /sbin/chkconfig --list サービス名
   ```

   例えば、次のような結果が得られる場合、ftpd(vsftpd)とrexecはシステム起動時に開始されない設定になっています。

   ```
   # /sbin/chkconfig --list vsftpd
   vsftpd 0:off 1:off 2:off 3:off 4:off 5:off 6:off
   ```

```
# /sbin/chkconfig -list rexec
rexec off
```

4. サービスの設定の変更

   /sbin/chkconfigコマンドを次の形式で使用して、システム起動時にサービスが開始されるように設定を確認します。

```
/sbin/chkconfig [--level レベル] サービス名 on
```

   以下、システム開始時にサービスが開始されるように設定を変更し、その変更を確認する例について示します。

```
# /sbin/chkconfig --level 5 vsftpd on
# /sbin/chkconfig --list vsftpd
vsftpd 0:off 1:off 2:off 3:off 4:off 5:on 6:off
# /sbin/chkconfig rexec on
# /sbin/chkconfig --list rexec
rexec on
```

5. サービスの起動

   /sbin/serviceコマンドを次の形式で使用して、サービスを開始します。

   なお、rexecの場合は、xinetd配下で起動するサービスであるため、xinetdを再起動する必要があります。

```
# sbin/service vsftpd start
vsftpd 用の vsftpd を起動中: [ OK ]
# /sbin/service xinetd stop
xinetd を停止中: [ OK ]
# /sbin/service xinetd start
xinetd を起動中: [ OK ]
```

5.2.3 サーバ側のユーザ環境の設定

5.2.3.1 UNIXサーバの場合

ここではSolarisサーバとLinuxサーバの設定情報が共通であるため、SolarisとLinuxを合わせてUNIXと記述しています。

UNIXアプリケーションのリモート開発時に、UNIXサーバ側の設定が必要な環境変数の詳細と、その設定方法について説明します。

コード変換

アプリケーション資産の送受信に必要なコード変換をUNIXサーバ側のInterstage Charset Managerを使用する場合には、次に示す環境変数を設定する必要があります。なお、以下の説明では、{CHARSET_BASED}がInterstage Charset Managerのインストール先を示すものとして説明します。

- LD_LIBRARY_PATH

  Interstage Charset Managerの共用ライブラリを格納したパスを指定するために、以下の指定を環境変数LD_LIBRARY_PATHに追加します。

```
${CHARSET_BASED}/lib
```

ビルド

リモート開発のビルドは、UNIXサーバ側のcobol翻訳コマンドを使用して、COBOLアプリケーションの翻訳・リンクが行われます。このため、次に示す4つの環境変数の指定は必須です。なお、以下の説明では、{COB_BASED}がNetCOBOLのインストール先を示すものとして説明します。

- PATH

  cobol翻訳コマンドの格納パスを指定するため、以下の指定を環境変数PATHに追加します。

```
${COB_BASED}/bin
```

・ LD_LIBRARY_PATH

COBOLランタイムの共用ライブラリを格納したパスを指定するために、以下の指定を環境変数LD_LIBRARY_PATHに追加します。

```
${COB_BASED}/lib
```

・ NLS_PATH

COBOL翻訳時およびCOBOLアプリケーション実行時に出力されるメッセージの格納先を指定するため、以下の指定を環境変数NLS_PATHに追加します。

```
${COB_BASED}/lib/nls/%L/%N.cat:${COB_BASED}/lib/nls/C/%N.cat
```

・ LANG

COBOLアプリケーションで使用する文字コード系を指定します。翻訳時はこの指定がCOBOLソース中の日本語文字の有無とそのコード系の判定に使用されます。

COBOLアプリケーションで使用する文字コード系環境変数LANGの指定値を以下に示します。

システム	日本語の使用の有無と文字コード			
	無し	有り		
		EUC	Shift_JIS	Unicode(UTF8)
Solaris	C	ja	ja_JP.PCK	ja_JP.UTF-8
Linux(x86)	C	ja_JP.eucJP	－	ja_JP.UTF-8
Linux(Itanium)	C	ja_JP.eucJP	－	ja_JP.UTF-8
Linux(x64)	C	－	ja_JP.WINDOWS-31J	ja_JP.UTF-8

LANGを除く環境変数の設定は、そのためのシェルスクリプトが各UNIX系システムのNetCOBOL製品に用意されており、通常はそれを使用します。

翻訳・リンク時に必須の環境変数を設定するためのシェルスクリプトを以下に示します。

システム	格納場所	ファイル名	備考
Solaris	/opt/FJSVcbl/config	cobol.csh	csh用
Linux(x86)	/opt/FJSVcbl/config	cobol.sh	sh/bash用
Linux(Itanium)		cobol.csh	csh/tcsh用
Linux(x64)	/opt/FJSVcbl64/config	cobol.sh	sh/bash用
		cobol.csh	csh/tcsh用

その他、必要に応じて次のような環境変数を指定します。詳細はサーバの"NetCOBOL 使用手引書"を参照してください。

・ COBOLOPTS

開発対象の個々のアプリケーションに依存せず共通に指定する必要のある翻訳オプションがある場合、この環境変数を使用します。次のようなオプションを指定するのに有効です。

－ COBOLのデバッグ機能に関するオプション

－ 翻訳リストに関するオプション

・ COBCOPY/FORMLIB/FILELIB

複数の開発者が共用する必要があるCOBOL登録集、画面帳票定義体、ファイル定義体などがある場合、その格納ディレクトリを指定します。

・ 登録集名

IN/OFで指定した登録集名を環境変数名とした環境変数に、登録集ファイルの格納されているディレクトリを設定します。

環境変数設定用のシェルスクリプト例

ここでは、設定する必要のある環境変数が次のようであると仮定して、その環境変数を設定するためのスクリプトの例を示します。

- 資産の転送時に関係する環境変数

  サーバ側でInterstage Charset Managerを使用するための設定を環境変数LD_LIBRARY_PATHに追加する。

- ビルドに関係する環境変数

  ― COBOLアプリケーションの翻訳・リンクに必須の環境変数は、NetCOBOLで提供されているシェルスクリプトで設定する。

  ― 開発対象のアプリケーションが使用する文字コードはEUCとし、それを環境変数LANGに指定する。

  ― COBOLソースの翻訳リストは、共通のフォルダに保存する。

  ― 開発者が共通して参照する登録集の格納フォルダを指定する。

Solarisサーバの場合

Solarisサーバを使用してリモート開発する場合、ログインシェルとしてcshを使用する必要があります。各開発者の使用するホームディレクトリにある".cshrc"に以下のテキストを追加編集してください。

Solarisサーバでの".cshrc"への修正例を示します。

```
## COBOL環境設定
source /opt/FJSVcbl/config/cobol.csh
## Interstage Charset Managerのための環境設定
if(${?LD_LIBRARY_PATH}) then
setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib:${LD_LIBRARY_PATH}
else
setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib
endif
## 開発者共通の翻訳・リンク時設定
setenv COBOLOPTS "-dp ../list"
setenv COBCOPY ../COPYLIB:${COBCOPY}
## 開発対象アプリケーションの使用する文字コード
setenv LANG ja
```

Linuxサーバの場合

Linuxサーバを使用してリモート開発する場合、ログインシェルとしてcshまたはbashを使用できます。

ログインシェルとしてcshを使用する場合、各開発者の使用するホームディレクトリにある".cshrc"に以下のテキストを追加編集してください。

".cshrc"への修正例を示します。

```
## COBOL環境設定
source /opt/FJSVcbl/config/cobol.csh
## Interstage Charset Managerのための環境設定
if(${?LD_LIBRARY_PATH}) then
setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib:${LD_LIBRARY_PATH}
else
setenv LD_LIBRARY_PATH /opt/FSUNiconv/lib
endif
## 開発者共通の翻訳・リンク時設定
setenv COBOLOPTS "-dp ../list"
setenv COBCOPY ../COPYLIB:${COBCOPY}
## 開発対象アプリケーションの使用する文字コード
setenv LANG ja_JP.eucJP
```

ログインシェルとしてbashを使用する場合、各開発者の使用するホームディレクトリにある".bashrc"に以下のテキストを追加編集してください。

".bashrc"への修正例を示します。

```
## COBOL環境設定
source /opt/FJSVcbl/config/cobol.sh
```

```
## Interstage Charset Managerのための環境設定
if [ ${LD_LIBRARY_PATH:-""} = "" ] ; then
LD_LIBRARY_PATH=/opt/FSUNiconv/lib; export LD_LIBRARY_PATH
else
LD_LIBRARY_PATH=/opt/FSUNiconv/lib:${LD_LIBRARY_PATH};export LD_LIBRARY_PATH
fi
## 開発者共通の翻訳・リンク時設定
COBOLOPTS="-dp ../list";export COBOLOPTS
COBCOPY=../COPYLIB:${COBCOPY}; export COBCOPY
## 開発対象アプリケーションの使用する文字コード
LANG=ja_JP.eucJP; export LANG
```

5.2.3.2 Windowsサーバの場合

Windowsサーバを使用してリモート開発を行う場合、サーバ環境上のユーザアカウントを使って開発作業が実行されます。リモート開発作業のために既存のユーザアカウントを使用しない場合は、リモート開発用の新規ユーザアカウントをサーバ環境上に作成する必要があります。新規ユーザアカウントを作成するには、そのサーバの管理者に相談してください。一般に、サーバのローカルユーザアカウントを追加する場合は、管理者アカウントでサーバにログオンし、[スタート] > [管理ツール] > [コンピュータの管理]を選択して[コンピュータの管理]を表示し、その中の[ローカルユーザとグループ]を使用します。

リモート開発の作業のために追加の環境設定が必要になった場合は、そのユーザアカウントの環境に設定を追加してください。

注意

ユーザアカウントはユーザグループのメンバーとして登録してください。
ローカルPCからサーバへの接続時にパスワードの変更はできません。このため、ユーザアカウントの設定で、ローカルPCからサーバへ接続するときにパスワードを変更する設定はしないでください。

Windows(Itanium)の場合

ユーザアカウントの環境変数に、NetCOBOLおよびMicrosoft Platform SDKの情報を設定する必要があります。設定する情報は使用するソフトウェア開発キットにより違いがあります。

Microsoft Platform SDKの場合

環境変数と設定する値を、次の表に示します。

- "%NetCOBOL%"はNetCOBOLのインストール先フォルダ名です。
- "%MSSdk%"はMicrosoft Platform SDKのインストール先フォルダ名です。

環境変数名	設定する値
Path	%NetCOBOL% %MSSdk%¥Bin¥Win64¥IA64 %MSSdk%¥Bin¥Win64 %MSSdk%¥Bin %MSSdk%¥Bin¥WinNT
Lib	%MSSdk%¥Lib¥IA64 %MSSdk%¥Lib¥IA64¥mfc
Include	%MSSdk%¥Include¥crt %MSSdk%¥Include¥crt¥sys %MSSdk%¥Include¥mfc %MSSdk%¥Include¥atl %MSSdk%¥Include

Windows SDK for Windows Server 2008 and .NET Framework 3.5 (V6.1)の場合

環境変数と設定する値を、次の表に示します。

- "%NetCOBOL%"はNetCOBOLのインストール先フォルダ名です。

ー "%MSSdk%"と"%VCRoot%"はWindows SDK for Windows Server 2008 and .NET Framework 3.5 (V6.1)のインストール先フォルダ名です。"%MSSdk%"と"%VCRoot%"のデフォルトインストール先は以下になります。

```
%MSSdk%: C:¥Program Files¥Microsoft SDKs¥Windows¥v6.1
%VCRoot%: C:¥Program Files (x86)¥Microsoft Visual Studio 9.0¥VC
```

環境変数名	設定する値
Path	%NetCOBOL% %MSSdk%¥Bin %MSSdk%¥Bin¥IA64 %VCRoot%¥Bin¥IA64 %VCRoot%¥vcpackages
Lib	%MSSdk%¥Lib¥IA64 %VCRoot%¥Lib¥IA64
Include	%MSSdk%¥Include %MSSdk%¥Include¥gl %VCRoot%¥Include

注意

ソフトウェア開発キットのインストール先のパス名に含まれるフォルダ名"IA64"は、サーバのCPUアーキテクチャにより"x64"、"amd64"などになっている場合があります。このため、ソフトウェア開発キットのインストール先のパス名を確認してから設定してください。

Windows(x64)の場合

ユーザアカウントの"PATH"環境変数にNetCOBOLのインストール先フォルダ名を設定します。リモートビルドはサーバ側NetCOBOLの"NetCOBOLコマンドプロンプト"と同じ環境で実行されます。

5.2.4 ローカルPC側の環境設定

リモート開発する場合、各開発者が使用するワークベンチでサーバと連携するための情報を設定する必要があります。

なお、Windows XP SP2以降を適用済みのシステムで、かつ、サーバ側のNetCOBOLリモート開発サービスを使用せずにftpd/rexecサービスを使用する場合、"5.2.4.3 Windows XP SP2以降の適用時の設定"を先に行う必要があります。

5.2.4.1 サーバ情報

各開発者が使用するワークベンチでサーバと連携するための情報を設定する必要があります。

ここで設定した情報は、ワークスペース間で情報が共有されます。

サーバ情報の設定

以下の手順で[サーバ情報の新規作成]ダイアログボックスを表示し、サーバ情報を設定します。

1. メニューバーから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
2. [設定]ダイアログボックスの左のペインで[COBOL] > [リモート開発]を選択します。[リモート開発]ページが表示されます。
3. [リモート開発]ページで[サーバ情報の新規作成]をクリックします。[サーバ情報の新規作成]ダイアログボックスが表示されます。

項目	説明
サーバ名	[設定]ダイアログボックスの[リモート開発]ページの[サーバ名]に表示する任意の名前を指定します。 COBOLプロジェクトまたはCORBAサーバプロジェクトがどのサーバをターゲットとするかを指定するときに、ここで指定したサーバ名が利用されます。 定義済のサーバ名を指定することはできません。

項目	説明
サーバのOS	サーバのOSを選択します。
サーバのアドレス	ネットワーク上のサーバを識別するための名前（FQDN: Fully Qualified Domain Name）、またはIPアドレスを指定します。
常に以下のユーザ名とパスワードを使用する	サーバへ接続するときに、このダイアログボックスで指定したユーザ名とパスワードを使用するか否かを指定します。 選択すると、このダイアログボックスで指定したユーザ名とパスワードを使用します。 選択しないと、ワークベンチ起動後の最初のサーバへの接続時にサーバ接続時のユーザ名とパスワード入力が表示されます。 デフォルトではチェックされていません。
ユーザ名	サーバで使用するアカウントのユーザ名を指定します。 [常に以下のユーザ名とパスワードを使用する]が選択されている場合は、省略できません。
パスワード	ユーザ名に付与されたパスワードを指定します。 [常に以下のユーザ名とパスワードを使用する]が選択されている場合は、省略できません。
コード変換	サーバとのファイル送受信時におけるコード変換の情報です。 コード変換処理は、通常はシステムの提供するコード変換の機能を使用して行われます。しかし、ADJUST(Windowsだけ)またはInterstage Charset Managerが導入されている場合は、これらの製品を使用してコード変換が行われます。
サーバのコード系	サーバの日本語文字のコード系を選択します。
サーバでコード変換する ローカルでコード変換する	テキストファイルの送受信をする場合、サーバまたはローカルPCのどちらでコード変換するかを指定します。 初期値では"サーバでコード変換する"が選択されています。 [サーバのOS]で"Windows(Itanium)"または"Windows(x64)"を選択している場合は無効となります。
UNIX系サーバの情報	[サーバのOS]でSolaris、Linux(x86)またはLinux(Itanium)を選択したときに指定する情報です。
サーバ側NetCOBOLのバージョンがV10以前	サーバ側NetCOBOLの製品バージョンがV10以前の場合に選択します。 選択するとリモート開発のサーバ側のサービスとしてftpd/rexecサービスを使用します。
ファイル転送(FTP)にPASVモードを使用する	選択するとPASVモードでファイル転送されます。 ファイアウォールの外側にあるFTPサーバとファイル転送をするときに、サーバへ接続ができない場合があります。このような場合に、選択します。 デフォルトでは選択されていません。 サーバ側のftpd/rexecサービスを使用するリモート開発の場合に有効となります。
サーバのNetCOBOLリモート開発サービス	サーバ側のNetCOBOLリモート開発サービスの情報です。
ポート番号	NetCOBOLリモート開発サービスのTCP/IPのポート番号を指定します。 サーバ側でNetCOBOLリモート開発サービスのポート番号を変更した場合に、変更した値に合わせてください。 初期値では"61999"です。 サーバ側のNetCOBOLリモート開発サービスを使用するリモート開発の場合に有効となります。
接続確認	ボタンをクリックすると、現在の設定値でサーバに接続し、その結果をサーバへの接続確認に表示します。

サーバへの接続確認

サーバ情報の[サーバ情報の新規作成]ダイアログボックスまたは [サーバ情報の変更]ダイアログボックスで[接続確認]をクリックすると、ダイアログボックスの設定値でサーバへ接続し、その結果が[確認]ダイアログボックスに表示されます。

接続に成功した場合は、サーバの環境変数の設定情報が表示されます。

接続に失敗した場合は、エラー情報が表示されます。

サーバ接続時のユーザ名とパスワード入力

サーバ情報の[サーバ情報の新規作成]ダイアログボックスまたは [サーバ情報の変更]ダイアログボックスで[常に以下のユーザ名とパスワードを使用する]を選択していない場合、ワークベンチを起動後の最初のサーバへの接続で、ユーザ名とパスワードを指定するダイアログボックスが表示されます。以降のサーバへの接続はこのユーザ名とパスワードを使用します。

サーバ情報の変更

以下の手順で[サーバ情報の変更]ダイアログボックスを表示し、サーバ情報を変更します。

1. メニューバーから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
2. [設定]ダイアログボックスの左のペインで[COBOL] > [リモート開発]を選択します。[リモート開発]ページが表示されます。
3. [リモート開発]ページの[サーバ名]からサーバ情報を変更するサーバ名を選択し、[サーバ情報の変更]をクリックします。[サーバ情報の変更]ダイアログボックスが表示されます。

サーバ情報の削除

1. 以下の手順でサーバ情報を削除します。メニューバーから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
2. [設定]ダイアログボックスの左のペインで[COBOL] > [リモート開発]を選択します。[リモート開発]ページが表示されます。
3. [リモート開発]ページの[サーバ名]からサーバ情報を削除するサーバ名を選択し、[削除]をクリックするとサーバ情報が削除されます。

5.2.4.2 プロジェクトのリモート開発設定

新規作成したCOBOLプロジェクトまたはCORBAサーバプロジェクトはリモート開発のサーバ情報が設定されていないため、リモート開発の機能を使用できません。

以下の手順でプロジェクトにリモート開発の情報を設定します。

1. 依存ビューまたは構造ビューでリモート開発のサーバ情報を設定するプロジェクトを選択します。
2. コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. [プロパティ]ダイアログボックスの左のペインで[リモート開発]を選択すると、[リモート開発]ページが表示されます。

項目	説明
リモート開発機能を有効にする	このプロジェクトがリモート開発用のプロジェクトである場合に選択します。 5.2.4.1 サーバ情報が設定されていない場合は、無効となります。 デフォルトでは選択されていません。
サーバ名	プロジェクトが対象とするサーバ名を選択します。[設定]ダイアログボックスの[COBOL] > [リモート開発]で指定したサーバ名の一覧が表示されます。 [リモート開発機能を有効にする]が選択されていない、またはサーバ情報が設定されていない場合は、無効となります。
サーバディレクトリ	サーバ側でビルドするための資産を格納するディレクトリをフルパスで指定します。メイクファイル生成機能およびビルド機能は、このディレクトリをカレントディレクトリとして処理を実行します。 ・ サーバOSがSolaris、Linuxの場合：ルートディレクトリを指定することはできません。 ・ サーバOSがWindowsの場合：ドライブ直下を指定することはできません。 [参照]をクリックしてサーバのディレクトリを参照できます。 [リモート開発機能を有効にする]が選択されていない、またはサーバ名が定義されていない場合は、無効となります。

注意

[サーバディレクトリ]に指定したディレクトリがサーバ上に存在しない場合、メイクファイル生成時に作成されます。
[サーバディレクトリ]に指定するディレクトリは、ローカルPCのCOBOLプロジェクトまたはCORBAサーバプロジェクト単位で異なるディレクトリを指定する必要があります。複数のプロジェクトがサーバのディレクトリを共有すると、メイクファイル生成が正しく実行されません。

5.2.4.3 Windows XP SP2以降の適用時の設定

Windows XP SP2以降でセキュリティ強化のために追加された"Windowsファイアウォール"が有効となっている場合、リモート開発の機能が使用できなくなります。
この問題を回避するためには、次の表に示すプログラムを"Windowsファイアウォール"によるチェックの対象外とするように設定を変更します。

プログラム名	格納フォルダ	備考
COBRDC32.exe	NetCOBOLインストールフォルダ	リモートデバッガコネクタ

回避方法

以下の手順で、COBRDC32.exeを例外として登録します。
なお、この機能を使用しない場合は、対応するプログラムを登録する必要はありません。

1. [Windowsファイアウォール]設定画面の[例外]タブにおいて、[プログラムの追加]を選択します。
2. [プログラムの追加]ダイアログボックスの[参照]から、NetCOBOL製品のインストールフォルダに存在する"COBRDC32.exe"を選択し、[OK]をクリックして項目を追加します。

スコープの変更について

上記の方法で、必要なプログラムを"Windowsファイアウォール"によるチェックの対象外として登録した場合、そのプログラムに対するスコープを変更することによって、セキュリティを強化できます。スコープの変更は、次の手順で行います。

1. [Windowsファイアウォール]設定画面の[例外]タブに登録されているプログラムから、スコープを変更するプログラムを選択し、[編集]をクリックします。
2. [プログラムの編集]ダイアログボックスが表示されるので、[スコープの変更]をクリックします。
3. [スコープの変更]ダイアログボックスで、[ユーザのネットワーク(サブネットのみ)]を選択、または[カスタムの一覧]を選択します。
4. [カスタムの一覧]を選択した場合、対象コンピュータのIPアドレスを設定し、[OK]をクリックします。

Linuxサーバの場合

Linuxサーバとの連携で、"Windowsファイアウォール"が有効となっており、かつ、サーバ側のNetCOBOLリモート開発サービスではなくftpd/rexecサービスを使用する場合、Linuxサーバとの連携で処理時間が極端に遅くなる場合があります。

この現象を回避するには、以下の手順で113番ポートを例外として登録します。

1. [Windowsファイアウォール]設定画面の[例外]タブにおいて、[ポートの追加]をクリックします。[ポートの追加]ダイアログボックスが表示されます。
2. [ポートの追加]ダイアログボックスで以下の値を設定します。
   - [名前]:任意の名前を設定します。
   - [ポート番号]:"113"を設定します。
   - [TCP/UDP]:TCPを選択します。
3. 接続を受け付けるサーバを固定したい場合は、[スコープの変更]をクリックして[スコープの変更]ダイアログボックスを表示し、必要な設定を行います。

5.3 メイクファイル生成

リモートビルドによるサーバでのビルド処理ではメイクファイルが必要です。メイクファイル生成機能を使用することにより、サーバでのビルド処理で必要となるメイクファイルを生成できます。

COBOLプロジェクトの場合は、1つのメイクファイルが生成されます。CORBAサーバプロジェクトでは、実行ファイル用およびライブラリ用の2種類のメイクファイルが生成されます。ただし、CORBAサーバプロジェクトで主プログラムを指定したCOBOLソースファイルが存在しない場合には、ライブラリ用のメイクファイルだけが生成されます。

5.3.1 メイクファイルの生成

以下の手順でメイクファイルを生成します。

1. 依存ビューまたは構造ビューでメイクファイルを生成するプロジェクトを選択します。
2. メニューバーから[プロジェクト] > [リモート開発] > [メイクファイル生成]を選択するか、コンテキストメニューから[リモート開発] > [メイクファイル生成]を選択します。[メイクファイル生成]ダイアログボックスが表示されます。
3. [メイクファイル生成]ダイアログボックスの[生成条件]にメイクファイル生成時の条件が表示されます。表示された[生成条件]でメイクファイルを生成するのであれば、[OK]をクリックすることによりメイクファイルが生成されます。表示された[生成条件]を変更したい場合は、[オプション設定]をクリックして[生成条件]の内容を変更します。

COBOLプロジェクトの場合、生成されるメイクファイルのファイル名は"Makefile"で、サーバ側とローカルPC側の両方に格納されます。ローカルPC側のメイクファイルは、COBOLプロジェクトの[その他のファイル]フォルダに登録されます。サーバ側のメイクファイルは、プロジェクトのプロパティの[リモート開発]ページで指定した[サーバディレクトリ]に格納されます。
CORBAサーバプロジェクトの場合、生成されるメイクファイルのファイル名は以下のとおりとなります。

	サーバ側ファイル名 （サーバ側ディレクトリからの相対パス）	ローカルPC側ファイル名
実行ファイル用	Makefile	Makefile
共用ライブラリ／ダイナミックリンクライブラリ	lib/Makefile	Makefile_lib

ポイント

メイクファイルの生成処理はサーバ側で行われます。サーバ側での実行結果の詳細は、コンソールビューのツールバーのアイコン([コンソールを開く])から[COBOLリモート]を選択することにより確認できます。

注意

メイクファイル生成では、プロジェクトの[プロパティ] > [リモート開発]ページの[サーバディレクトリ]で指定したディレクトリ直下の拡張子が".cobol"または".cob"のファイルが、COBOLソースファイルとして扱われます。

[メイクファイル生成]ダイアログボックスの表示内容

[メイクファイル生成]ダイアログボックスの[生成条件]にメイクファイル生成時の、以下の生成条件が表示されます。

ターゲット名

メイクファイルのターゲットとなる実行ファイル名またはダイナミックリンクライブラリ名(共用ライブラリ名)が表示されます。

CORBAサーバプロジェクトの場合は、実行ファイル名およびライブラリ名が表示されます。ただし、主プログラム指定のCOBOLソースファイルが存在しない場合は、ライブラリ名だけが表示されます。

転送するファイル

メイクファイル生成時にサーバ側へ転送するファイルの一覧が表示されます。ファイルは以下の種別ごとに表示されます。

- COBOLソースファイル

ー COBOL登録集・定義体ファイル

ー プリコンパイラ入力ソース

翻訳オプション

メイクファイル中でCOBOLソースの翻訳時に使用する翻訳オプションが表示されます。

リンクオプション

メイクファイル中でCOBOLソースのリンク時に使用するリンクオプションが表示されます。

プリコンパイラ連携

プリコンパイラ連携をする場合、プリコンパイラの以下の情報が表示されます。

ー プリコンパイラ入力ソースの拡張子

ー プリコンパイラ出力ソースの拡張子

ー プリコンパイラコマンド名とパラメタ

ー COBOLコンパイラのエラーメッセージを、プリコンパイラ入力ソースの行番号で表示する場合、INSDBINFコマンドのパラメタ

[メイクファイル生成]ダイアログボックスの値

[メイクファイル生成]ダイアログボックスの[生成条件]に表示される値は、初回のメイクファイルと2回目以降の生成では次の違いがあります。

初回のメイクファイル生成

プロジェクトのプロパティの[ターゲット]ページおよび[ビルド]ページで設定されている値が参照されます。

ただし、[ビルド]ページで設定された情報は一部加工されます。[ビルド]ページの各タブの詳細を、以下に説明します。

ー [翻訳オプション]タブ

- システム間で共通の形式を持つ翻訳オプションの場合は、その情報のコピーが作成されます。
- サーバ側のOSでサポートされていない翻訳オプションの場合は、指定を無視します。

ー [登録集名]タブ

- [登録集名]ダブで指定された値はメイクファイルに反映されません。
- サーバ側の環境変数にIN/OFで指定した登録集名を環境変数名として、登録集ファイルの格納されているディレクトリを設定してください。

ー [リンクオプション]タブ

- InterstageおよびNetCOBOLの提供しているオブジェクトファイル、ライブラリファイルはサーバ側のNetCOBOLで提供されている格納パス、ファイル名に置き換えられます。
- InterstageおよびNetCOBOLの提供していないオブジェクトファイル、ライブラリファイルは、次のようになります。

  サーバ側のOSがWindows(Itanium)またはWindows(x64)の場合は、その情報のコピーが作成されます。

  サーバ側のOSがSolarisまたはLinuxの場合は、オブジェクトファイル、ライブラリファイルの情報は削除されます。

2回目以降のメイクファイル生成

前回のメイクファイル生成時の値となります。

5.3.2 メイクファイルの生成条件の変更

[メイクファイル生成]ダイアログボックスに表示されている、以下のメイクファイル生成時の条件を変更できます。

- ターゲット名
- プリコンパイラ
- 翻訳オプション

- リンクオプション

これらの条件を変更するには、[メイクファイル生成]ダイアログボックスで[オプション設定]をクリックしてください。[オプション設定]をクリックすると [オプション設定]ダイアログボックスが表示され、メイクファイル生成時のターゲット名、翻訳オプション、リンクオプションの生成条件を変更できます。

5.3.2.1 ターゲットオプションの変更

[オプション設定]ダイアログボックスで[ターゲット]タブを選択すると、ターゲット名を変更できます。

項目	説明
ターゲット名	ターゲット名を指定します。
初期化	プロジェクトのプロパティの[ターゲット]ページで指定された値に初期化します。

注意

実行ファイルまたはダイナミックリンクライブラリ(共用ライブラリ)の種別は、プロジェクトのプロパティの[ターゲット]ページの[ターゲット種別]で選択されている値になります。ターゲットの種別を変更する場合は、プロジェクトのプロパティの[ターゲット]ページの[ターゲット種別]の選択を変更してください。

CORBAサーバプロジェクトの場合、実際のターゲット名は以下のとおりです。

サーバ側システム	ターゲット種別	実際のターゲット名
UNIX系システム	実行ファイル	Target_name
	共用ライブラリ	libTarget_name.so
Windows系システム	実行ファイル	Target_name.exe
	ダイナミックリンクライブラリ	Target_name.dll

5.3.2.2 プリコンパイラ連携情報の変更

[オプション設定]ダイアログボックスで[プリコンパイラ]タブを選択すると、プリコンパイラ連携情報を変更できます。

項目	説明
プリコンパイラを使用する	プリコンパイラを使用するメイクファイルを生成する場合に選択します。 選択されていない場合、プリコンパイラの情報が設定されていてもプリコンパイラの情報はメイクファイルに反映されません。
プリコンパイラコマンド	プリコンパイラとして起動するコマンド名を指定します。
プリコンパイラのパラメタ	プリコンパイラコマンドのパラメタを指定します。
入力ソースの拡張子	プリコンパイラ入力ソースファイルの拡張子を指定します。 以下の拡張子を指定することはできません。 ・ cobol ・ cob ・ cbl ・ lcai
出力ソースの拡張子	プリコンパイラ出力ソースファイルの拡張子を選択します。
COBOLコンパイラのエラーメッセージをプリコンパイラ入力ソースの行番号で表示する	選択するとプリコンパイラ入力ソースの行対応情報をCOBOLソースファイルへ展開します(INSDBINFコマンドを呼び出します)。 初期値では選択されていません。

項目	説明
INSDBINFコマンドのパラメタ	プリコンパイルによって生成されたCOBOLソースファイルに、プリコンパイラ入力ソースに対する行補正情報を展開するINSDBINFコマンドのパラメタを指定します。ただし、入力ソースファイル名と出力ソースファイル名は、プリコンパイラ入力ソースファイル名から決定されるため、指定する必要はありません。
初期化	プロジェクトのプロパティの[プリコンパイラ]ページで設定されている値で初期化します。

プリコンパイラ連携情報の詳細は"6.1.4.1 プリコンパイラ連携情報の初期値の設定・変更"を参照してください。

注意

プリコンパイラ出力ソースの拡張子".cobol"はサーバ側のNetCOBOL製品が以下の場合、利用できません。

- Solaris、Linux(Itanium)の場合

  V9.1以前の製品

- Linux(x86)の場合

  V7.3以前の製品

- Windowsの場合

  V9.0L10以前の製品

5.3.2.3 翻訳オプションの変更

[オプション設定]ダイアログボックスで[翻訳オプション]タブを選択すると、翻訳オプションを変更できます。

項目	説明
翻訳オプション	メイクファイル中でCOBOLソースの翻訳時に使用する翻訳オプションが表示されます。
追加	翻訳オプションを追加します。 [翻訳オプションの追加]ダイアログボックスでは、[翻訳オプション]で追加したいオプションを選択し、[追加]をクリックすることで、翻訳オプションを追加します。
変更	[翻訳オプション]で選択された翻訳オプションを変更します。
削除	[翻訳オプション]で選択された翻訳オプションを削除します。
初期化	プロジェクトのプロパティの[ビルド]ページで指定された値に初期化します。
その他の翻訳オプション	[翻訳オプションの追加]ダイアログボックスで追加できない翻訳オプションを指定します。

注意

CORBAサーバプロジェクトの場合、ターゲットは静的プログラム構造にする必要があるため、DLOAD翻訳オプションは指定しないでください。

リモート開発で使用できない翻訳オプション

以下の翻訳オプションはローカル開発固有であり、リモート開発では使用できません。

- AIMLIB翻訳オプション
- GEN翻訳オプション

以下の翻訳オプションはターゲットOSがSolarisの場合に使用可能であり、他のOSのリモート開発では使用できません。

- FILELIB翻訳オプション

リモート開発固有の翻訳オプション

[オプション設定]ダイアログボックスの[翻訳オプション]タブでは、プロジェクトのプロパティの[ビルド]ページで扱えない、リモート開発固有の翻訳オプションを扱うことができます。

リモート開発固有の翻訳オプションと、それら翻訳オプションのターゲットOSごとの使用の可否を、以下の表に示します。

翻訳オプション	Solaris	Linux		Windows
		x86	Itanium	Itanium
CODECHECK	○	○	○	×
KANA	○	○	○	×
LALIGN	○	×	○	×

○: 使用可能
×: 使用不可

リモート開発固有の翻訳オプションの詳細について以下に説明します。

CODECHK翻訳オプション

実行時に翻訳時の日本語コード系のチェックを行う(CODECHK)か、行わない(NOCODECHK)かを指定します。
日本語のコード系に依存しないアプリケーション(Shift_JIS/EUC/Unicode共通アプリケーション)を作成する場合、NOCODECHKを指定する必要があります。

項目	説明
実行時の日本語コード系チェックの指定	実行時に翻訳時の日本語コード系のチェックを行うか、行わないかを指定します。 初期値では[実行時に翻訳時のコード系とのチェックを行う]が選択されます。
実行時に翻訳時のコード系とのチェックを行う	実行時に翻訳時の日本語コード系のチェックを行います。
実行時に翻訳時のコード系とのチェックを行わない	実行時に翻訳時の日本語コード系のチェックを行いません。

KANA翻訳オプション

文字定数および英字・英数字項目内のカナ文字のコード系を指定します。

項目	説明
文字コードの扱い	文字コードの扱いを指定します。 初期値では[EUC]が選択されます。
EUC	カナ文字の文字コードは、2バイトコード(EUC)となります。
JIS8	カナ文字の文字コードは、1バイトコード(JIS)となります。

LALIGN翻訳オプション

連絡節に宣言されたデータを参照する場合、8バイトの整列境界にあっていることを前提としたオブジェクトを生成する(LALIGN)か、前提としないオブジェクトを生成する(NOLALIGN)かを指定します。
なお、整列境界が8バイト境界にあっていることを前提としたオブジェクトを生成する場合、データの処理速度が向上します。

項目	説明
連絡節のデータ宣言の扱い	連絡節のデータ宣言の扱いを指定します。 初期値では[整列境界が8バイトの整列境界にあっていることを前提としない]が選択されます。
整列境界が8バイトの整列境界にあっていることを前提とする	整列境界が8バイトの整列境界にあっていることを前提とします。

項目		説明
	整列境界が8バイトの整列境界にあっていることを前提としない	整列境界が8バイトの整列境界にあっていることを前提としません。

リモート開発で指定形式が異なる翻訳オプション

ローカルPCとサーバで翻訳オプションの指定形式が異なる、以下の翻訳オプションがあります。

- RCS翻訳オプション

RCS翻訳オプションの指定形式が異なるのはLinux(Itanium)だけです。SolarisとLinux(x86)ではRCS翻訳オプションは指定できません。

リモート開発で指定形式が異なる翻訳オプションの詳細について以下に説明します。

RCS翻訳オプション

RCS翻訳オプションはローカルPCとLinux(Itanium)では指定形式が異なります。ここでの説明はサーバがLinux(Itanium)の場合に表示されるダイアログボックスの説明です。
Unicode環境での日本語項目の表現形式はUCS-2です。このときエンディアンをビッグエンディアンとするか、リトルエンディアンとするかを指定します。

項目		説明
Unicode環境での日本語項目の表現形式		UCS-2のエンディアンを指定します。 初期値では[LE]が選択されます。
	BE	UCS-2のエンディアンをビッグエンディアンとします。
	LE	UCS-2のエンディアンをリトルエンディアンとします。

5.3.2.4 登録集名の参照

[オプション設定]ダイアログボックスで[登録集名]タブを選択すると、プロジェクトのプロパティの[ビルド]ページの[登録集名]タブで指定した値を参照できます。

[登録集名]ダブで指定された値はメイクファイルに反映されません。サーバ側の環境変数にIN/OFで指定した登録集名を環境変数名として、登録集ファイルの格納されているディレクトリを設定してください。

5.3.2.5 リンクオプションの変更

リンクオプションは[オプション設定]ダイアログボックスの2つのタブで構成されています。

- [リンクオプション1]タブ

  サーバ側でCOBOLアプリケーションとリンクするライブラリ、オブジェクトファイルを指定できます。

- [リンクオプション2]タブ

  サーバ側のOSで固有なリンクオプションを指定できます。

リンクするライブラリおよびオブジェクトファイルの変更

[オプション設定]ダイアログボックスで[リンクオプション1]タブを選択すると、サーバ側でリンクするライブラリおよびオブジェクトを変更できます。

項目	説明
追加	COBOLアプリケーションとリンクするライブラリ／オブジェクトファイルを追加します。 [追加]をクリックすると、[リンクオプションの追加]ダイアログボックスが表示されます。 追加したライブラリ／オブジェクトファイルは[ライブラリ／オブジェクトファイル]に表示されます。 ライブラリ／オブジェクトファイルは複数追加できます。

項目	説明
変更	[ライブラリ／オブジェクトファイル]で選択されているライブラリ／オブジェクトファイルの指定を変更します。 [変更]をクリックすると、[リンクオプションの変更]ダイアログボックスが表示されます。
削除	[ライブラリ／オブジェクトファイル]で選択されているライブラリ/オブジェクトファイルを削除します。
すべて削除	[ライブラリ／オブジェクトファイル]にあるすべてのライブラリ/オブジェクトファイルを削除します。
Cランタイムライブラリ名	リンク時に結合するCランタイムライブラリのファイル名を指定します。 サーバのOSがWindows(Itanium)またはWindows(x64)の場合に有効になるオプションです。 Cランタイムライブラリ名を省略すると"LIBCMT.lib"が結合されます。
DLLエントリオブジェクト	COBOLで作成されたオブジェクトファイルだけでダイナミックリンクライブラリを作成するか、他言語で作成されたオブジェクトファイルと一緒にダイナミックリンクライブラリを作成するかを指定します。 サーバのOSがWindows(Itanium)またはWindows(x64)の場合に有効になるオプションです。
COBOL単体用	COBOLで作成されたオブジェクトファイルだけでダイナミックリンクライブラリを作成します。
他言語間結合用	他言語で作成されたオブジェクトファイルと一緒にダイナミックリンクライブラリを作成します。
初期化	[ライブラリ/オブジェクトファイル]にあるすべてのライブラリ／オブジェクトファイルを削除して、初回のメイクファイル生成時の値にします。

[オプション設定]ダイアログボックスの[リンクオプション1]タブで[追加]、[変更]をクリックすると、[リンクオプションの追加]ダイアログボックスが表示され、ライブラリ／オブジェクトファイルを追加・変更できます。
[ライブラリ／オブジェクトファイルの選択]に追加・変更するライブラリ・オブジェクトファイル名を指定してください。
ライブラリ名・オブジェクトファイル名は絶対パスまたは相対パスで指定します。ファイル名だけの指定はできません。
[参照]をクリックすると、サーバ側のファイルを参照するダイアログボックスが表示され、ライブラリ・オブジェクトファイルを選択できます。

サーバ側のOSで固有なリンクオプションの変更

[オプション設定]ダイアログボックスで[リンクオプション2]タブを選択すると、サーバ側のOSで固有なリンクオプションを変更できます。
サーバ側のリンクオプションの詳細は、サーバ側の"NetCOBOL 使用手引書"を参照してください。

項目	説明
結合モード	結合モードを指定します。 初期値では[動的結合]が選択されます。 サーバ側のOSがSolarisまたはLinuxの場合に有効となるオプションです。 CORBAサーバプロジェクトの場合、常に[動的結合]となります。
動的結合	COBOLアプリケーションを動的結合により作成します。
静的結合	COBOLアプリケーションを静的結合により作成します。
画面帳票定義体を使用するプログラム	画面帳票定義体を使用しているプログラムをリンクする場合に選択します。 デフォルトでは選択されていません。 サーバ側のOSがSolarisの場合に有効となるオプションです。 CORBAサーバプロジェクトでは無効となります。
スクリーン操作機能を使用するプログラム	スクリーン操作を使用しているプログラムをリンクする場合に選択します。 デフォルトでは選択されていません。 サーバ側のOSがSolarisの場合に有効となるオプションです。 CORBAサーバプロジェクトでは無効となります。
C-ISAMを使用するプログラム	C-ISAM使用しているプログラムをリンクする場合に選択します。 デフォルトでは選択されていません。 サーバ側のOSがSolarisの場合に有効となるオプションです。
C言語から呼び出されるプログラム	C言語から呼び出されるプログラムをリンクする場合に選択します。 デフォルトでは選択されていません。

項目	説明
	サーバ側のOSがSolarisの場合に有効となるオプションです。 CORBAサーバプロジェクトでは無効となります。
Cランタイムライブラリを使用する	ターゲット種別がダイナミックリンクライブラリで、かつ、Cランタイムライブラリを使用している場合に選択します。 デフォルトでは選択されていません。 サーバ側のOSがWindows(Itanium)またはWindows(x64)の場合に有効となるオプションです。
デバッグ情報を出力する	デバッグ情報を出力する場合に選択します。 デフォルトでは選択されています。 サーバ側のOSがWindows(Itanium)またはWindows(x64)の場合に有効となるオプションです。
リンクオプション[-Wl]の指定	ldコマンドが使用するリンクオプションを指定します。 サーバ側のOSがSolarisまたはLinuxの場合に有効となるオプションです。

注意

マルチスレッドモデルのプログラムをリンクするオプション("-Tm")は、プロジェクトのプロパティの[ビルド]ページの[翻訳オプション]タブで"THREAD(MULTI)"が指定されている場合に自動的に設定されます。

5.3.3 資産の転送

メイクファイル生成では、COBOLプロジェクトまたはCORBAサーバプロジェクトの以下のフォルダで管理しているファイルがサーバ側で必要となります。

- [ソースファイル]フォルダに登録されているCOBOLソースファイル・プリコンパイラ入力ソース
- [依存関係ファイル]フォルダに登録されているCOBOL登録集・定義体ファイル

これらファイルのサーバへの転送はメイクファイル生成時に自動的に行われます。メイクファイル生成時にサーバへ転送されるファイル名は、[メイクファイル生成]ダイアログボックスの[生成条件]に表示されます。

注意

[ソースファイル]フォルダに登録されている拡張子"cbl"のファイルはサーバへの転送対象とはなりません。
[依存関係ファイル]フォルダに登録されているCOBOL登録集・定義体ファイルで転送対象となるのはプロジェクト内にあるファイルだけです。他のプロジェクトまたは他のフォルダで管理されているCOBOL登録集・定義体ファイルは転送の対象となりません。また、サーバへの転送対象となる登録集ファイルは拡張子"cbl"のファイルだけとなります。

5.3.4 メイクファイルの編集

生成したメイクファイルは通常はそのままサーバ側のビルドに使用できますが、何らかの理由により問題がある場合は編集できます。ローカルPC側とサーバ側のどちらのメイクファイルを編集したかにより、次のように処理されます。

- ローカルPC側のメイクファイルを編集した場合

  メイクファイル編集後の初回サーバ側での[ビルド]・[再ビルド]の実行に先立って、編集したメイクファイルがサーバ側へ自動的に転送されます。

- サーバ側のメイクファイルを編集した場合

  メイクファイルの再生成時に、サーバ側の編集したメイクファイルを再生成するメイクファイルで置き換えてよいかの確認メッセージを出力します。確認メッセージで置き換えが拒否された場合は、メイクファイルの生成は中断されます。

5.3.5 メイクファイルの再生成

COBOLプロジェクトまたはCORBAサーバプロジェクトの構成が変更されてもメイクファイルの再生成は自動的に行われません。

以下のようなCOBOLプロジェクトまたはCORBAサーバプロジェクトの構成要素を変更した場合は、メイクファイルを再生成してください。

- [ソースファイル]フォルダに登録されているCOBOLソースファイルを追加、削除、改名
- [依存関係ファイル]フォルダに登録されている登録集ファイル、各種定義体ファイル、リポジトリファイルを追加、削除、改名
- [リンクファイル]フォルダのファイルの追加、削除、改名
- プリコンパイラ連携情報の追加、変更

以下の手順でメイクファイルを再生成します。

1. 依存ビューまたは構造ビューでメイクファイルを生成するプロジェクトを選択します。
2. メニューバーから[プロジェクト] > [リモート開発] > [メイクファイル生成]を選択するか、コンテキストメニューから[リモート開発] > [メイクファイル生成]を選択します。

メイクファイルの再生成ではターゲット名、プリコンパイラ連携情報、翻訳オプション、リンクオプションの値は、前回メイクファイルを生成したときの値となります。
現在生成されているメイクファイルのターゲット名、プリコンパイラ連携情報、翻訳オプション、リンクオプションの値を変更する場合もメイクファイルを再生成して、これらの値を変更してください。

5.4 リモートビルド

リモートビルドによりサーバ側で翻訳・リンクを実行し、サーバ側で動作するCOBOLアプリケーションを作成できます。

翻訳・リンクに必要となるメイクファイルはメイクファイル生成機能により生成されたものを使用します。

5.4.1 ビルドの実行

リモートビルドは以下の手順で実行します。

1. 依存ビューまたは構造ビューでリモートビルドするプロジェクトを選択します。
2. メニューバーから[プロジェクト] > [リモート開発] > [ビルド]を選択するか、コンテキストメニューから[リモート開発] > [ビルド]を選択します。

リモートビルドは前回のビルド以降に変更されたリソースがビルドされます。前回のビルド以降に変更されていないものも含め、すべてのリソースをビルドするには、以下の手順で再ビルドを実行します。

1. 依存ビューまたは構造ビューで再ビルドするプロジェクトを選択します。
2. メニューバーから[プロジェクト] > [リモート開発] > [再ビルド]を選択するか、コンテキストメニューから[リモート開発] > [再ビルド]を選択します。

ポイント

翻訳エラーは問題ビューに表示されます。サーバでのビルド結果は、コンソールビューのツールバーのアイコン([コンソールを開く])から[COBOLリモート]を選択することにより確認できます。

ビルドモードの設定

ビルドモードの設定により、リモートビルドで生成するCOBOLアプリケーションをリリース用またはデバッグ用にビルドできます。

ビルドモードの変更は以下の手順で実行します。

1. 依存ビューまたは構造ビューでビルドモードを変更するプロジェクトを選択します。
2. メニューバーから[プロジェクト] > [リモート開発] > [デバッグモードでビルド]の選択を変更するか、コンテキストメニューから[リモート開発] > [デバッグモードでビルド]の選択を変更します。[デバッグモードでビルド]が選択されていないとリリース用となり、選択されているとデバッグ用となります。

ビルドモードの情報は、プロジェクトごとの情報として保存されます。

注意

デバッグ用のビルドをした場合、TEST翻訳オプションに"対話型デバッガを使用しない"と指定していても"対話型を使用する"でビルドされます。

資産の転送

COBOLプロジェクトまたはCORBAサーバプロジェクトの以下のフォルダで管理しているファイルが、前回のリモートビルド後に更新されている場合は、リモートビルドの実行に先立ってサーバ側に自動的に転送されます。

- [ソースファイル]フォルダに登録されているCOBOLソースファイル・プリコンパイラ入力ソース
- [依存関係ファイル]フォルダに登録されているCOBOL登録集・定義体ファイル

注意

[ソースファイル]フォルダに登録されている拡張子"cbl"のファイルはサーバへの転送対象とはなりません。
[依存関係ファイル]フォルダに登録されているCOBOL登録集・定義体ファイルで転送対象となるのはプロジェクト内にあるファイルだけです。他のプロジェクトまたは他のフォルダで管理されているCOBOL登録集・定義体ファイルは転送の対象となりません。また、サーバへの転送対象となる登録集ファイルは拡張子"cbl"のファイルだけとなります。

5.4.2 翻訳エラーの修正

リモートビルドの翻訳エラー情報は問題ビューに表示されます。

翻訳エラーとなったローカルPC側のCOBOLソースファイルを編集するには、問題ビューで翻訳エラー情報をダブルクリックするか、コンテキストメニューから[ジャンプ]を選択します。COBOLソースファイルがエディタで開かれて翻訳エラーとなった行がカレント行となります。すでにエディタで開かれているCOBOLソースの場合は、翻訳エラー行がカレント行となります。

修正が完了したらリモートビルドを実行します。修正したファイルがサーバ側へ自動的に転送されてビルド処理が実行されます。

ポイント

プリコンパイラが検出したエラー情報は[問題]ビューには表示されません。エラー情報は、[コンソール]ビューのツールバーのアイコン([コンソールを開く])から[COBOLリモート]を選択することにより確認できます。

5.5 リモートデバッグ

サーバ側でビルドしたCOBOLアプリケーションをデバッグするには、リモートCOBOLアプリケーション起動構成を使用します。

リモートデバッグを開始するには、2つの方法があります。

- 通常デバッグ

  リモートデバッガをローカルPC側から起動し、デバッグを開始する起動方法です。あらかじめサーバ側でリモートデバッガコネクタを起動しておく必要があります。

- アタッチデバッグ

  サーバ側で実行したCOBOLアプリケーションからリモートデバッガを起動し、デバッグを開始する起動方法です。この起動方法は、Interstage Application ServerやWebサーバなどの環境下で動作するCOBOLアプリケーションをデバッグする場合に使用します。

リモートデバッグの詳細については"NetCOBOL 使用手引書"の"対話型リモートデバッガの使い方"も参照してください。

5.5.1 通常デバッグ

通常デバッグを開始する手順を以下に示します。

1. サーバ側リモートデバッガコネクタの起動

2. リモートデバッガの起動

サーバ側リモートデバッガコネクタの起動

リモートデバッグを行う場合、クライアント側のデバッガからの指示を監視するリモートデバッガコネクタをサーバ側で起動しておく必要があります。

サーバ側のリモートデバッガコネクタは以下のコマンド形式により起動します。

サーバ	起動コマンド
Windows(Itanium)	cobrds64 [ポート指定] [接続制限指定]
Windows(x64)	
Solaris	svdrds [ポート指定] [接続制限指定]
Linux	

起動コマンドの指定方法に関する詳細は、"NetCOBOL 使用手引書"を参照してください。

また、サーバ側のリモートデバッガコネクタはデバッガ終了時に自動終了しないため、リモートデバッグを終了した場合は、リモートデバッガコネクタも終了させる必要があります。サーバ側リモートデバッガコネクタを終了させるには、サーバ側リモートデバッガコネクタを起動したコマンド入力画面で [Ctrl + C]を実行してください。

COBOLアプリケーションをリモートデバッグする

以下にリモートデバッグを開始する手順を示します。

1. 依存ビューまたは構造ビューからCOBOLプロジェクトを選択します。
2. メニューバーから[実行] > [構成およびデバッグ]を選択します。またはツールバーで  の▼をクリックし[構成およびデバッグ]を選択します。[構成およびデバッグ]ダイアログボックスが表示されます。
3. 左のペインで[リモートCOBOLアプリケーション]を選択します。
4. 左のペイン上の  をクリックすると、右のペインに起動構成の設定ページが表示されます。
5. デフォルトで[名前]に起動構成名が表示されます。起動構成名は任意の名前に変更できます。
6. [メイン]タブをクリックし、各設定項目の確認と必要に応じて変更を行います。
   a. [プロジェクト名]には、選択したプロジェクト名が表示されています。
   b. [デバッグ方法]では、[通常デバッグ]を選択します。
   c. [サーバ名]には、プロジェクトのプロパティで設定したサーバ名が表示されています。
   d. [ポート番号]には、デバッグ時のサーバ側との通信に使用するポート番号を設定します。サーバ側のリモートデバッガコネクタの起動時に指定したポート番号と同じ値を設定してください。初期値では、サーバ側リモートデバッガコネクタの初期値である59998が設定されています。
   e. [実行ファイル]には、デバッグの対象となる実行ファイルを設定します。初期値には、メイクファイルの有無によって、以下が設定されます。
      - メイクファイルを生成している場合:メイクファイルに指定したターゲット名
      - メイクファイルを生成していない場合:プロジェクトのターゲット名

      デバッグ対象がダイナミックリンクライブラリの場合には、対象となるファイルを指定してください。
   f. [作業フォルダ]には、デバッグ対象アプリケーションの作業ディレクトリを設定します。初期値には、実行ファイルのディレクトリが設定されています。
   g. [プログラム引数]にコマンドラインで指定する形式でプログラムの引数を設定します。
7. [デバッグ]を選択することでデバッグが開始されます。一度デバッグ起動した起動構成は、[実行] > [ヒストリのデバッグ]およびツールバーのショートカットに登録され、そこから再度起動できます。

注意

プロジェクトにリモート開発の情報が設定されていない場合は、[デバッグ方法]で[通常デバッグ]を選択することはできません。

ポイント

依存ビューまたは構造ビューでプロジェクトを選択し、メニューバーから[実行] > [デバッグ] > [リモートCOBOLアプリケーション]を選択することにより、デフォルトの設定でデバッガを起動できます。

5.5.2 アタッチデバッグ

リモートデバッガを使用し、アタッチ形式のリモートデバッグを行うことができます。

アタッチ形式のデバッグ開始を指示するための環境変数をサーバ側で設定し、COBOLアプリケーションを実行することによりリモートデバッガが起動され、リモートデバッグを開始します。

ローカルPC側リモートデバッガコネクタ

アタッチ形式のリモートデバッグを行う場合、サーバ側からのリモートデバッガ起動に関する指示を監視するリモートデバッガコネクタがデバッガ起動時にローカルPC上で自動的に起動されます。

ローカルPC側リモートデバッガコネクタが起動されると、タスクトレイにアイコン  が表示されます。リモートデバッガコネクタはデバッガ終了時に自動終了しないため、リモートデバッグを終了した場合は、リモートデバッガコネクタも終了させる必要があります。リモートデバッガコネクタを終了させるには、アイコンのコンテキストメニューから[終了]を選択してください。

ローカルPC側リモートデバッガコネクタに関するその他の詳細については、"NetCOBOL 使用手引書"を参照してください。

リモートデバッガの起動

以下にリモートデバッガを起動する手順を示します。

1. 依存ビューまたは構造ビューからCOBOLプロジェクトを選択します。
2. メニューバーから[実行] > [構成およびデバッグ]を選択します。またはツールバーで  の▼をクリックし[構成およびデバッグ]を選択します。[構成およびデバッグ]ダイアログボックスが表示されます。
3. 左のペインで[リモートCOBOLアプリケーション]を選択します。
4. 左のペイン上の  をクリックすると、右のペインに起動構成の設定ページが表示されます。
5. デフォルトで[名前]に起動構成名が表示されます。起動構成名は任意の名前に変更できます。
6. [メイン]タブをクリックし、各設定項目の確認と必要に応じて変更を行います。
   a. [プロジェクト名]には、選択したプロジェクト名が表示されています。
   b. [デバッグ方法]では、[アタッチデバッグ]を選択します。
   c. [デバッグ情報フォルダ]には、プロジェクトのプロパティの[リモート開発]ページの[サーバディレクトリ]で指定したディレクトリと異なるディレクトリにデバッグ情報ファイルが格納されている場合に、その格納ディレクトリを指定します。

      複数のディレクトリを指定する場合には次のように指定します。

      - サーバがSolaris、Linuxの場合は":"(コロン)で区切って指定します。
      - サーバがWindowsの場合は";"(セミコロン)で区切って指定します。
7. [デバッグ]を選択することでデバッグが開始され、デバッグするアプリケーション側から起動が通知されるまで待ち状態となります。一度デバッグ起動した起動構成は、[実行] > [ヒストリのデバッグ]およびツールバーのショートカットに登録され、そこから再度起動できます。

注意

サーバがSolarisまたはLinuxであり、かつ、デバッグ対象のCOBOLアプリケーションが使用するダイナミックリンクライブラリが実行ファイルと異なるディレクトリに格納されている場合は、ダイナミックリンクライブラリのデバッグ情報ファイル(.svd)を実行ファイルの格納されているディレクトリにコピーしてからデバッグを開始してください。

ポイント

依存ビューまたは構造ビューでプロジェクトを選択し、メニューバーから[実行] > [デバッグ] > [リモートCOBOLアプリケーション]を選択することにより、デフォルトの設定でデバッガを起動できます。

サーバ側アプリケーションの実行

リモートデバッガに起動を通知するために必要となる環境変数の設定、およびデバッグ対象のアプリケーションの実行をサーバ側で行います。

設定する環境変数は以下のとおりです。"接続先"にはローカルPCのIPアドレスまたはホスト名および必要に応じてポート番号を指定します。

"追加パスリスト"には、起動するアプリケーションが動作し始めたときのカレントディレクトリと、起動するアプリケーションの格納ディレクトリは記述する必要はありません。以下の順序で、デバッグ情報のファイルが検索され、デバッグに利用されます。

1. 追加パスリストの指定順(各パスは、Windowsの場合は";"で区切って記述してください。Solaris、Linuxの場合は":"で区切って記述してください)
2. アプリケーションが動作し始めたときのカレントディレクトリ
3. 起動するアプリケーションの格納ディレクトリ

環境変数の設定方法に関する詳細は、"NetCOBOL 使用手引書"を参照してください。

サーバ	環境変数
Windows(Itanium)	@CBR_ATTACH_TOOL=接続先/APD 追加パスリスト
Windows(x64)	
Solaris	CBR_ATTACH_TOOL=接続先/APD 追加パスリスト
Linux	

接続待ち状態の解除

アタッチデバッグを実行したあとで、サーバ側アプリケーションからの接続待ち状態を解除する場合は、ウィンドウ右下の  をクリックし、表示された[進行状況]ビューで  をクリックしてください。

ローカルPC上のCOBOLアプリケーションをアタッチデバッグする

[リモートCOBOLアプリケーション]起動構成では、ローカルPC上でビルドしたCOBOLアプリケーションをアタッチデバッグすることもできます。

ローカルPC上でアタッチデバッグする場合には、デバッガにアプリケーションの起動を通知するために以下の環境変数をローカルPCに設定します。

ローカルPC	環境変数
Windows	@CBR_ATTACH_TOOL=localhost/APD

以下にローカルPC上でアタッチデバッグする手順を示します。

1. 依存ビューまたは構造ビューからCOBOLプロジェクトまたはCORBAサーバプロジェクトを選択します。

2. メニューバーから[実行] > [構成およびデバッグ]を選択します。またはツールバーで  の▼をクリックし[構成およびデバッグ]を選択します。[構成およびデバッグ]ダイアログボックスが表示されます。
3. 左のペインで[リモートCOBOLアプリケーション]を選択します。
4. 左のペイン上の  をクリックすると、右のペインに起動構成の設定ページが表示されます。
5. デフォルトで[名前]に起動構成名が表示されます。起動構成名は任意の名前に変更できます。
6. [メイン]タブをクリックし、各設定項目の確認と必要に応じて変更を行います。
    a. [プロジェクト名]には、選択したプロジェクト名が表示されています。
    b. [デバッグ方法]では、[アタッチデバッグ]を選択します。
7. [デバッグ]を選択することでデバッグが開始され、デバッグするアプリケーション側から起動が通知されるまで待ち状態となります。
8. COBOLアプリケーションを実行し、デバッグを開始します。

5.5.3 CORBAアプリケーションのデバッグ

CORBAアプリケーションをリモートデバッグする場合は、アタッチデバッグを使用します。ローカルPC側でデバッガが待機している状態で、サーバ側のCORBAアプリケーションを実行します。デバッガの起動手順については、アタッチデバッグと同様です。

第6章 プロジェクトをビルド・デバッグ・実行する

プロジェクトをビルドすることにより、実行に必要なファイルを作成します。作成したファイルは、実行機能を用いて実行できます。
アプリケーションの実行時に思うように動作しない場合は、デバッガを利用して問題点の調査を行い、アプリケーションの修正を行います。

6.1 プロジェクトをビルドする

ビルドとは、プロジェクトに定義された内容に従って、実行に必要なファイルを作成することです。プロジェクトを実行するためには、プロジェクトをビルドし、事前に実行に必要なファイルを作成しておく必要があります。
プロジェクトをビルドする場合、そのプロジェクトに関係する定義の内容に矛盾がないかどうかがチェックされます。そのプロジェクトに関係する定義の内容を修正した場合には、再度、ビルドする必要があります。
また、ビルド方法を変更する必要がある場合には、ビルドの前にビルドオプションを指定してください。

プロジェクトのビルドの実行方法には、自動的にビルドする方法と手動でビルドする方法の2通りがあります。

- 自動的にビルド
  自動的にビルドするように設定しておくと、リソースが変更されたあと(ファイルの保存など)にビルドが行われます。
- 手動でビルド
  ビルドを行うタイミングを細かくコントロールする必要がある場合に有効です。
  手動でビルドするには、メニューバーから[プロジェクト] > [プロジェクトのビルド]を選択するか、メニューバーから[プロジェクト] > [すべてビルド]を選択します。

ビルド方法はメニューの[プロジェクト]>[自動的にビルド]の選択によって切り替わります。[自動的にビルド]にチェックが付いている場合は自動的にビルドされます。[自動的にビルド]にチェックが付いていない場合は手動でビルドします。

ビルドは、前回のビルドから変更のあったリソースに対して行われます。すべてのリソースをビルドし直すには、メニューバーから[プロジェクト] > [クリーン]を選択します。

ビルドでコンパイルエラーなどの問題が発生した場合、問題ビューに問題が表示されます。

COBOLの場合はビルドの詳細なログをコンソールビューで確認できます。ビルドのログを表示するには、コンソールビューのツールバーのアイコン([コンソールを開く])から[ビルドコンソール]を選択します。

ポイント

プリコンパイラが検出したエラー情報は、問題ビューには表示されません。エラー情報は[コンソール]ビューの[ビルドコンソール]に表示されます。

注意

COBOLおよびCORBAプロジェクトのプロジェクトフォルダに生成されるファイルbuild.xmlはビルドで利用されるファイルです。編集およびAntビルド(Antスクリプトの実行)は行わないでください。

6.1.1 ビルドツールを設定する

プロジェクト単位にビルドツールのオプションを指定できます。

プロジェクト種別とデフォルトのビルドツールの一覧

プロジェクト種別がCOBOLおよびCORBAサーバの場合、ビルダには"COBOLビルダ"が設定されており、ビルドツールでビルドの詳細な設定を行います。ビルドツールでは追加、削除および実行順序を変更できます。ビルドツールの設定の変更は、プロジェクトを選択してメニューバーから[ファイル] > [プロパティ]を選択し、[プロパティ]ダイアログボックスの[ビルドツール]で設定します。

プロジェクト種別		ビルドツール				
		IDLコンパイラ	COBOLコンパイラ	リンカ	リソースコンパイラ	プリコンパイラ
COBOL	CORBAサーバ	デフォルトで設定	デフォルトで設定	デフォルトで設定		
	COBOL		デフォルトで設定	デフォルトで設定		

注意

プロジェクトの新規作成時にプリコンパイラを使用することを指示した場合、ビルドツールにプリコンパイラはデフォルトで設定されます。

ビルドツールの構成を設定する

以下の手順で[ビルドツール]ページを表示し、ビルドツールの構成を設定します。

1. ナビゲータビューなどからプロジェクトを選択します。
2. コンテキストメニューから[プロパティ]を選択するか、メニューバーから[ファイル] > [プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[ビルドツール]を選択すると、プロジェクトに設定されているビルドツールの一覧が表示されます。ビルドでは、表示された順序でビルドツールを実行します。

項目	説明
アプリケーションから追加	アプリケーション種別からビルドツールを追加します。 [アプリケーションから追加]をクリックすると[アプリケーションからのビルドツールの追加]ダイアログボックスが表示され、一覧にないビルドツールを持つアプリケーションが一覧で表示されます。追加したいビルドツールを持つアプリケーションを一覧から選択します。
追加	プロジェクトにビルドツールを追加します。 [追加]をクリックすると[ビルドツールの追加]ダイアログボックスが表示され、プロジェクトに設定されていないビルドツールが一覧で表示されます。追加したいビルドツールを一覧から選択します。
削除	選択されたビルドツールを削除します。 注)ビルドツールの内、プロジェクトのビルドに必須なビルドツールは削除できません。
上へ	選択されたビルドツールの実行順序を変更します。 選択されたビルドツールは、直前に実行するビルドツールより前に実行します。
下へ	選択されたビルドツールの実行順序を変更します。 選択されたビルドツールは、直後に実行するビルドツールより後に実行します。
デフォルトの復元	プロジェクトを新規に作成したときのビルドツール構成に戻します。
適用	設定した内容が適用されます。

注意

ビルドツールを追加すると、以下のチェックが行われます。

- 必要なビルドツールが利用可能であること
- ビルドツールが特定の順序で並んでいること

また、ビルドツールの追加や実行順序を変更する場合は、ビルドツールの実行順序に注意して設定を行ってください。

- 以下のビルドツールは、COBOLコンパイラより前に実行されるように設定してください。
  - IDLコンパイラ

— プリコンパイラ

- 以下のビルドツールは、COBOLコンパイラより後に実行されるように設定してください。

— リソースコンパイラ

— リンカ

ポイント

IDLコンパイラやリソースコンパイラなど、追加したビルドツールの詳細設定が必要な場合は、[適用]をクリックすることで、詳細設定が行えるようになります。

6.1.2 IDLコンパイラ

IDLコンパイラはIDL定義ファイルからソースコードを生成するために使用します。生成されるソースコードはクライアントスタブとオブジェクトスケルトンです。クライアントスタブはクライアントプログラムの開発に使用します。オブジェクトスケルトンはCORBAオブジェクトの実装に使用します。

COBOL用IDLコンパイラのオプションを設定する

以下の手順で[IDLコンパイラ]ページを表示し、ソースコードを生成するためのオプションを設定します。

1. ナビゲータビューなどからCOBOLプロジェクトまたはCORBAサーバプロジェクトを選択します。
2. メニューバーから[ファイル] > [プロパティ]を選択するか、コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[ビルドツール] > [IDLコンパイラ]を選択すると[IDLコンパイラ]ページが表示されます。

項目	説明
インクルードファイルフォルダ	IDLファイル内の“#include”文で指定されたファイルを検索するフォルダ名を指定します。
その他のオプション	追加の翻訳オプションを指定します。2つ以上のオプションを指定する場合は、区切り文字としてスペースを挿入します。
インタフェースリポジトリに登録する	情報をインタフェースリポジトリに登録する場合に指定します。

6.1.3 COBOLコンパイラ

COBOLコンパイラを使ってプロジェクトを翻訳できます。

6.1.3.1 翻訳に関係するファイル

COBOLコンパイラが使用するファイルを以下に示します。

COBOLコンパイラが使用するファイル
ソースファイル(*.cob、*.cobol、*.cbl)
登録集ファイル（*.cbl）
リポジトリファイル（*.rep）

依存ビューのソースファイル配下に表示されているソースファイルは、翻訳の対象になります。

COBOLコンパイラによって生成されるファイルを以下に示します。

COBOLコンパイラによって生成されるファイル
オブジェクトファイル（*.obj）
リポジトリファイル（*.rep）
デバッグ情報ファイル（*.svd）
翻訳リストファイル（*.lst）
実行ファイル（*.exe）
ダイナミックリンクライブラリ（*.dll）

以下に翻訳で使用されるファイルの詳細を示します。

ファイルの内容	ファイル名形式	入出力 (*1)	使用・作成条件	関連する翻訳オプション	関連する環境変数
ソースファイル	任意のファイル名.cob 任意のファイル名.cobol 任意のファイル名.cbl	I	必須	-	-
登録集	任意のファイル名.cbl	I	COPY文を使用するソースプログラムの翻訳時	LIB	COB_COBCOPY COB_LIBSUFFIX COB_ライブラリ名
オブジェクトファイル	ソースファイル名.obj	O	翻訳が正しく行われた場合に生成される	-	-
リポジトリファイル	クラス名.rep	I	REPOSITORY段落を持つソースプログラムの翻訳時	REPIN REP	COB_REPIN
		O	クラス定義が正しく翻訳された場合に生成される		
デバッグ情報ファイル	ソースファイル名.svd	O	翻訳オプションTESTを指定するか、ビルドモードにデバッグモードを設定した場合	-	-
翻訳リストファイル	*.lst	O	翻訳リストが出力される場合	PRINT	-

(*1) 入出力のIとOの意味は次のとおりです。

- I: 翻訳の入力ファイル
- O: 翻訳の結果出力されるファイル

6.1.3.2 主プログラムの設定

ターゲットの種別が実行ファイルの場合、COBOLプロジェクトのビルドを行う前にプロジェクトのソースファイルの中から1つを主プログラムとして指定する必要があります。
主プログラムを指定しない場合、リンクエラーが発生します。

主プログラムには作成するプログラムのタイプにより、以下の2種類があります。主プログラムの種別は、COBOLプロジェクト生成時にウィザードで指定します。

- COBOLのコンソールを使用するアプリケーション

  ACCEPT文およびDISPLAY文の入出力先にCOBOLが作成したコンソールウィンドウを使用し、かつ実行時エラーメッセージの出力先にメッセージボックスを使用する場合に指定します。

- システムのコンソールを使用するアプリケーション

  ACCEPT文、DISPLAY文および実行時エラーメッセージの入出力先としてシステムのコンソール(コマンドプロンプトウィンドウ)を使用する場合に指定します。

注意

主プログラムとして[システムのコンソールを使用するアプリケーション]を指定した場合、ワークベンチから実行した場合はシステムのコンソールではなくコンソールビューを使用します。

主プログラムを設定する

以下の手順で主プログラムを設定します。

1. 依存ビューまたは構造ビューから主プログラムとして設定するソースファイルを選択します。
2. コンテキストメニューから[主プログラム]を選択します。
3. 選択したソースファイルが主プログラムとして設定され、ファイルのアイコンが変更されます。

主プログラムの種別を変更する

以下の手順で主プログラムの種別を変更します。

1. 依存ビューまたは構造ビューからプロジェクトを選択します。
2. メニューバーから[ファイル] > [プロパティ]を選択するか、コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[ターゲット]を選択すると[ターゲット]ページが表示されます。
4. [作成するアプリケーションの形式]から[COBOLのコンソールを使用するアプリケーション]または[システムのコンソールを使用するアプリケーション]を選択し、[OK]をクリックします。

6.1.3.3 翻訳オプション

プロジェクトのビルドに必要な翻訳オプションの設定について説明します。

6.1.3.3.1 翻訳オプションの設定

翻訳オプションの設定はプロジェクトごとに行います。設定したオプションはパラメタとしてCOBOLコンパイラに渡されます。

以下の手順で[翻訳オプション]ページを表示し、翻訳オプションを設定します。

1. 依存ビューまたは構造ビューからCOBOLプロジェクトまたはCORBAサーバプロジェクトを選択します。
2. メニューバーから[ファイル] > [プロパティ]を選択するか、コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[ビルド]を選択すると[ビルド]ページが表示されます。
4. [翻訳オプション]タブを選択すると[翻訳オプション]ページが表示されます。

項目	説明
翻訳オプション	COBOLコンパイラに渡す翻訳オプションを表示します。
追加	翻訳オプションを追加します。 [翻訳オプションの追加]ダイアログボックスでは、[翻訳オプション]で追加したいオプションを選択し、[追加]をクリックすることで、翻訳オプションを追加します。 翻訳オプションについては"6.1.3.3.2 翻訳オプションの詳細"を参照してください。
変更	[翻訳オプション]で選択された翻訳オプションを変更します。
削除	[翻訳オプション]で選択された翻訳オプションを削除します。

項目	説明
ビルドモードがデバッグの場合にCHECK(ALL)翻訳オプションを付加する	ビルドモードがデバッグのときにCHECK(ALL)翻訳オプションを強制的に付加する場合にチェックします。 初期値ではチェックされていません。 ビルドモードがデバッグの場合に有効です。
オプションファイル	COBOLコンパイラに渡す翻訳オプションファイル(*.cbi)のファイル名を指定します。 [翻訳オプション]ダイアログボックスで指定する以外のその他のオプションを使用する場合には、翻訳オプションファイルにオプションを記述してプロジェクトフォルダに格納し、そのファイル名を[オプションファイル]に指定してください。なお、[翻訳オプション]ダイアログボックスと翻訳オプションファイルに同じオプションが指定された場合には、[翻訳オプション]ダイアログボックスの指定が有効になります。

注意

- ファイルの拡張子を指定する翻訳オプション

  以下の翻訳オプションは[翻訳オプションの追加]ダイアログボックスには表示されません。

  ー FILEEXT翻訳オプション

  ー FORMEXT翻訳オプション

  ー LIBEXT翻訳オプション

  これらの翻訳オプションを使用する場合は、6.1.3.3.3 既存の翻訳オプションファイルの利用か、または環境変数に設定してください。詳細については"NetCOBOL 使用手引書"を参照してください。

- ソースファイルのコード系を指定する翻訳オプション(NetCOBOL V10.0.0以上がインストールされている場合)

  COBOLソースファイルのコード系を指定するSCS翻訳オプションは[翻訳オプションの追加]ダイアログボックスには表示されません。SCS翻訳オプションはファイルのプロパティの[テキストファイルエンコード]から自動的に決定されます。

  SCS翻訳オプションのコード系は、ファイルのプロパティの [テキストファイルエンコード]で"UTF-8"が選択されている場合は"SCS(UTF8)"となり、"UTF-8"以外が選択されている場合は"SCS(SJIS)"となります。

6.1.3.3.2 翻訳オプションの詳細

[翻訳オプションの追加]ダイアログボックスの[翻訳オプション]の一覧で選択したオプションにしたがって、詳細な情報を指定するダイアログボックスが表示されます。

注意

翻訳オプションを指定するダイアログボックスで[参照]をクリックし、[プロジェクトから選択]ダイアログボックスからプロジェクトを選択した場合、パスの区切り文字が"¥"ではなく"/"と表示されますが、"¥"として扱われます。

翻訳オプションの一覧

翻訳時の資源に関するもの

- AIMLIB (サブスキーマ定義ファイルのフォルダの指定)
- FILELIB (ファイル定義体ファイルのフォルダの指定)
- FORMLIB (画面帳票定義体ファイルのフォルダの指定)
- LIB (登録集ファイルのフォルダの指定)
- REP (リポジトリファイルの入出力先フォルダの指定)
- REPIN (リポジトリファイルの入力先フォルダの指定)

翻訳リストに関するもの

- COPY (登録集原文の表示)
- GEN (FCOMおよびUWAの表示)
- LINECOUNT (翻訳リストの1ページあたりの行数)
- LINESIZE (翻訳リストの1行あたりの文字数)
- LIST (目的プログラムリストの出力の可否)
- MAP (データマップリスト、プログラム制御情報リストおよびセクションサイズリストの出力の可否)
- MESSAGE (オプション情報リスト、翻訳単位統計情報リストの出力の可否)
- NUMBER (ソースプログラムの一連番号領域の指定)
- PRINT (各種翻訳リストの出力の可否および出力先の指定)
- SOURCE (ソースプログラムリストの出力の可否)
- XREF (相互参照リストの出力の可否)

翻訳時メッセージに関するもの

- CONF (規格の違いによるメッセージの出力の可否)
- FLAG (診断メッセージのレベル)
- FLAGSW (COBOL文法の言語要素に対しての指摘メッセージ表示の可否)

COBOLプログラムの解釈に関するもの

- ALPHAL (英小文字の扱い)
- BINARY (2進項目の扱い)
- CURRENCY (通貨編集用文字の扱い)
- DUPCHAR (重複文字の扱い)
- INITVALUE (作業場所節でのVALUE句なし項目の扱い)
- LANGLVL (ANSI COBOL規格の指定)
- NCW (日本語利用者語の文字集合の指定)
- NSPCOMP (日本語空白の比較方法の指定)
- QUOTE/APOST (表意定数QUOTEの扱い)
- RSV (予約語の種類)
- SDS (符号付き10進項目の符号の整形の可否)
- SHREXT (マルチスレッドプログラムの外部属性に関する扱い)
- SQLGRP (SQLのホスト変数定義の拡張)
- SRF (正書法の種類)
- STD1 (英数字の文字の大小順序の指定)
- TAB (タブの扱い)
- ZWB (符号付き外部10進項目と英数字項目の比較)

ソースプログラムの解析に関するもの

- SAI (ソース解析情報ファイルの出力の可否)

目的プログラムの作成に関するもの

- ASCOMP5 (2進項目の解釈の指定)
- DLOAD (プログラム構造の指定)

- MODE (ACCEPT文の動作の指定)
- OBJECT (目的プログラムの出力の可否)
- OPTIMIZE (広域最適化の扱い)
- RCS (実行時コード系の指定)
- THREAD (マルチスレッドプログラム作成の指定)

実行時の処理に関するもの

- EQUALS (SORT文での同一キーデータの処理方法)
- TRUNC (桁落とし処理の可否)

実行時の資源に関するもの

- SMSIZE (PowerSORTが使用するメモリ容量を指定)
- SSIN (ACCEPT文のデータの入力先)
- SSOUT (DISPLAY文のデータの出力先)

実行時のデバッグ機能に関するもの

- CHECK (CHECK機能の使用の可否)
- COUNT (COUNT機能の使用の可否)
- TEST (対話型デバッガおよび診断機能の使用の可否)
- TRACE (TRACE機能の使用の可否)

AIMLIB翻訳オプション

サブスキーマ段落にサブスキーマ定義ファイルを記述した場合、サブスキーマ定義ファイルのフォルダを指定します。
使用するサブスキーマ定義ファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。フォルダを複数指定した場合、指定された順序でフォルダが検索されます。

項目	説明
サブスキーマ定義ファイルのフォルダ	サブスキーマ段落にサブスキーマ定義ファイルを記述した場合、サブスキーマ定義ファイルのフォルダを指定します。 フォルダはセミコロンで区切って複数追加できます。 [参照]をクリックすると、フォルダの指定方法を選択する[選択]ダイアログボックスが表示されます。[選択]ダイアログボックスに表示される一覧から、指定するフォルダのパスとなるパスの開始位置を指定してください。選択項目によっては、引き続き[フォルダの参照]ダイアログボックスが表示されます。指定するフォルダを選択してください。

ALPHAL翻訳オプション

ソースプログラム中の半角英小文字を半角英大文字と等価に扱う(ALPHAL)か、扱わない(NOALPHAL)かを指定します。

項目		説明
小文字の扱い		ソースプログラム中の英小文字の扱いを指定します。 初期値では[英大文字]が選択されます。
	英大文字	英小文字を英大文字として扱います。
	英小文字	英小文字と英大文字を区別して扱います。
文字定数の扱い		[小文字の扱い]に[英大文字]が指定された場合に選択できます。 初期値では[ALL]が選択されます。
	ALL	プログラム名定数、CALL文、CANCEL文、ENTRY文およびINVOKE文の定数中の英小文字も英大文字と等価に扱います。

項目		説明
	WORD	定数は記述どおりに扱います。

ASCOMP5翻訳オプション

2進項目の解釈を指定します。
データの内部表現が変わるため、内部表現を意識したコーディングが含まれている場合には注意してください。

項目		説明
2進項目の解釈の指定		2進項目の解釈を指定します。
	NONE	宣言されたとおりに解釈します。
	ALL	USAGE BINARYおよびUSAGE COMP、USAGE COMPUTATIONALと宣言された項目はUSAGE COMP-5が指定されたとみなします。
	BINARY	USAGE BINARYと宣言された項目はUSAGE COMP-5が指定されたとみなします。
	COMP	USAGE COMP、USAGE COMPUTATIONALと宣言された項目はUSAGE COMP-5が指定されたとみなします。

注意

PowerRW+を使用する場合、NONE以外は指定しないでください。

BINARY翻訳オプション

2進データの基本項目が、桁数より求められるワード単位の領域長(2,4,8)に割り付けられる(BINARY(WORD))か、バイト単位の領域長(1～8)に割り付けられる(BINARY(BYTE))かを指定します。
なお、符号なし2進項目の最左端ビットの扱いも指定できます。

項目		説明
2進項目の扱い		2進項目の扱いを指定します。 初期値では[BINARY(WORD,MLBON)]が選択されます。
	BINARY(WORD,MLBON)	ワード単位の領域長で最左端ビットは符号として扱われます。
	BINARY(WORD,MLBOFF)	ワード単位の領域長で最左端ビットは数値として扱われます。
	BINARY(BYTE)	バイト単位の領域長で最左端ビットは符号として扱われます。

CHECK翻訳オプション

CHECK機能を使用する(CHECK)か、しない(NOCHECK)かを指定します。
nには、メッセージを表示させる回数を0～999999の整数で指定します。省略した場合には、1が指定されたとみなします。

項目		説明
CHECK機能		CHECK機能の使用の可否を指定します。 初期値では[CHECK機能を使用しない]が選択されます。
	CHECK機能を使用する	CHECK機能を使用します。
	CHECK機能を使用しない	CHECK機能を使用しません。
表示メッセージ数		表示されるメッセージ数を指定します。
検査項目		

項目		説明
	ALL	NUMERIC、BOUND、ICONF、LINKAGEおよびPRMをすべて検査します。
	BOUND	添字・指標および部分参照の範囲外検査を行います。
	ICONF	INVOKE文のパラメタと呼び出すメソッドの仮パラメタの適合検査を行います。
	LINKAGE	以下のどちらかの場合に実行時メッセージを出力します。 ・ CALL文の呼び出し側と呼ばれる側でリンケージ規約が合っていない場合。 ・ 呼ばれる側がSTDCALLリンケージで、パラメタの個数が合っていない場合。
	NUMERIC	データの例外を検査します。データの例外は数値項目がその属性に適さない値を含む場合、または除算演算の除数が0(ゼロ)の場合に発生します。
	PRM	翻訳時に、内部プログラムを呼び出すCALL文(CALL一意名を除く)のUSING指定またはRETURNING指定に記述されたデータ項目と内部プログラムのUSING指定またはRETURNING指定に記述されたデータ項目に対して以下の検査を行います。 ・ USING指定のパラメタの個数の一致 ・ RETURNING指定のパラメタの有無の一致 ・ データ項目がオブジェクト参照以外の場合、データ項目の長さの一致。長さの検査は、翻訳時に長さが決定する場合だけ行う。 ・ データ項目がオブジェクト参照の場合、USAGE OBJECT REFERENCE句に指定されたクラス名、FACTORY指定およびONLY指定の一致 実行時に、外部プログラムを呼び出すCALL文のUSING指定またはRETURNING指定に記述されたデータ項目と外部プログラムのUSING指定またはRETURNING指定に記述されたデータ項目に対して以下の検査を行います。 ・ USING指定のパラメタの個数の一致、およびデータ項目の長さの一致。ただしUSING指定のパラメタの個数の不一致が4個以上の場合、誤りが検出されないことがあります。 ・ RETURING指定のパラメタの長さの一致。RETURNING指定がない場合、暗黙にPROGRAM-STATUSが受け渡されるため、長さ4バイトのデータ項目が指定されたものとみなします。 なお、実行時に長さが決定する場合は、翻訳時に記述した長さの最大値を使って、検査を行います。

注意

- CHECK機能使用時には、n回目のメッセージが出力されるまで、プログラムの処理が続行されますが、領域破壊などによりプログラムが期待どおり動作しない場合があります。なお、nに0を指定した場合には、メッセージの表示回数に関係なく、プログラムの処理が続行されます。
- CHECKを指定すると、上記の検査をするための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了時には、NOCHECKを指定して再翻訳してください。
- ON SIZE ERROR指定またはNOT ON SIZE ERROR指定の算術文では、ON SIZE ERRORの除数のゼロ検査が行われ、CHECK(NUMERIC)の除数のゼロ検査は行われません。
- CHECK(NUMERIC)のデータ例外検査は、外部10進項目または内部10進項目が参照で使用される場合、および英数字項目または集団項目から、外部10進項目または内部10進項目へ転記される場合に行われます。ただし次は、チェックの対象とはなりません。
  - ― 添字としてALLが指定されている表要素
  - ― SEARCH ALL文におけるキー項目(ただしキー項目に対する添字が1次元かつWHEN条件がひとつだけである場合は除く)
  - ― SORT/MERGE文におけるキー項目
  - ― SQL文中で使用されているホスト変数

ー CALL文、INVOKE文および行内呼び出しのBY REFERENCEパラメタ

ー 次の組み込み関数の引数

FUNCTION ADDR
FUNCTION LENG
FUNCTION LENGTH

ー 英数字項目または集団項目から、外部10進項目または内部10進項目のオブジェクトプロパティへの転記

- プロジェクトのプロパティのビルドページの[翻訳オプション]タブの[ビルドモードがデバッグの場合にCHECK(ALL)翻訳オプションを付加する]がチェックされていて、かつ、ビルドモードがデバッグの場合、CHECK(ALL)以外のオプションを指定するときはこのチェックを解除する必要があります。

CONF翻訳オプション

COBOLの旧規格と新規格の間の非互換を指摘させる(CONF)か、させない(NOCONF)かを指定します。
CONFを指定すると、非互換項目は、Iレベルの診断メッセージで指摘されます。

項目		説明
規格の違いによるメッセージの出力の可否		規格の違いによるメッセージの出力の可否を指定します。
	指摘する	COBOLの旧規格と新規格の間の非互換を指摘します。
	指摘しない	COBOLの旧規格と新規格の間の非互換を指摘しません。
指摘する非互換		[規格の違いによるメッセージの出力の可否]に[指摘する]が指定された場合に選択できます。 初期値では[68]が選択されます。
	68	'68 ANSI COBOLと'85 ANSI COBOLとで意味の解釈が異なる項目を指摘します。 翻訳オプションLANGLVL(85)を指定した場合にだけ意味を持ちます。
	74	'74 ANSI COBOLと'85 ANSI COBOLとで意味の解釈が異なる項目を指摘します。 翻訳オプションLANGLVL(85)を指定した場合にだけ意味を持ちます。
	OBS	廃要素である言語仕様および機能を指摘します。

注意

CONFは、従来の規格に従って作成したプログラムを、'85 ANSI COBOLの規格に従うように変更する場合に有効です。

COPY翻訳オプション

ソースプログラムリスト内に、COPY文によって組み込まれる登録集原文を表示する(COPY)か、しない(NOCOPY)かを指定します。
COPYは、翻訳オプションSOURCEを指定した場合だけ意味を持ちます。

項目		説明
登録集原文の表示		登録集原文を表示するか、しないかを指定します。 初期値では[NOCOPY]が選択されます。
	COPY	登録集原文を表示します。
	NOCOPY	登録集原文を表示しません。

COUNT翻訳オプション

COUNT機能を使用する(COUNT)か、使用しない(NOCOUNT)かを指定します。

項目		説明
COUNT機能の使用の可否		COUNT機能の使用の可否を指定します。 初期値では[使用しない]が選択されます。
	使用する	COUNT機能を使用します。
	使用しない	COUNT機能を使用しません。

注意

- COUNTを指定すると、COUNT情報を出力するための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了時には、NOCOUNTを指定して再翻訳してください。
- COUNTは、翻訳オプションTRACEオプションと同時に指定できません。同時に指定された場合、後に指定されたオプションが有効となります。

CURRENCY翻訳オプション

通貨編集用文字として使用している文字に、¥を使用する(CURRENCY(¥))か、$を使用する(CURRENCY($))かを指定します。

項目		説明
通貨編集用文字の扱い		通貨編集用文字の扱いを指定します。 初期値では[¥]が選択されます。
	¥	通貨編集用文字として¥を使用します。
	$	通貨編集用文字として$を使用します。

DLOAD翻訳オプション

プログラム構造を動的プログラム構造にする(DLOAD)か、しない(NODLOAD)かを指定します。

項目		説明
プログラム構造の指定		プログラム構造を指定します。 初期値では[NODLOAD]が選択されます。
	DLOAD	プログラム構造を動的プログラム構造にします。
	NODLOAD	プログラム構造を動的プログラム構造にしません。

DUPCHAR翻訳オプション

以下のソースプログラムをUnicode環境で翻訳したとき、コンパイラが付加／置換する全角ハイフンをシステム標準(DUPCHAR(STD))とするか、拡張文字(DUPCHAR(EXT))とするかを指定します。

- 3バイト項目制御部を指定した画面帳票定義体を取り込んでいる。
- COPY文の書き方2と3で日本語利用者語を使用している。

項目		説明
重複文字の扱い		重複文字の扱いを指定します。 初期値では[STD]が選択されています。
	STD	全角ハイフンをシステム標準とします。
	EXT	全角ハイフンを拡張文字とします。

注意

DUPCHAR翻訳オプションは、お使いになっているコンピュータにインストールされているNetCOBOLの製品バージョンがV10.0.0以降の場合に使用するこができます。

EQUALS翻訳オプション

実行時に、SORT文の入力中に同一キーを持つレコードが複数個存在する場合、それらに関して、SORT文の出力レコードの順序をSORT文の入力レコードの順序と同じにすることを保証する(EQUALS)か、しない(NOEQUALS)かを指定します。

項目		説明
SORT文での同一キーデータの処理方法		SORT文での同一キーデータの処理方法を指定します。 初期値では[順序を規定せず、実行を高速にする]が選択されます。
	入力した順序にすることを保証する	SORT文の入力中に同一キーを持つレコードが複数個存在する場合、それらに関して、SORT文の出力レコードの順序をSORT文の入力レコードの順序と同じにすることを保証します。
	順序を規定せず、実行を高速にする	SORT文の入力中に同一キーを持つレコードが複数個存在する場合、それらに関して、SORT文の出力レコードの順序をSORT文の入力レコードの順序と同じにすることを保証せず、実行速度を高速にします。

注意

EQUALSを指定すると、整列操作で入力順序を保証するための特別な処理が行われるために実行性能が低下します。

FILELIB翻訳オプション

IN/OF XFDLIB指定のCOPY文によりファイル定義体からレコード定義を取り込む場合、ファイル定義体ファイルのフォルダを指定します。使用するファイル定義体ファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。フォルダを複数指定した場合、指定された順序でフォルダが検索されます。

項目	説明
ファイル定義体ファイルのフォルダ	ファイル定義体ファイルの入力先フォルダを指定します。 フォルダはセミコロンで区切って複数追加できます。 [参照]をクリックすると、フォルダの指定方法を選択する[選択]ダイアログボックスが表示されます。[選択]ダイアログボックスに表示される一覧から、指定するフォルダのパスとなるパスの開始位置を指定してください。選択項目によっては、引き続き[フォルダの参照]ダイアログボックスが表示されます。指定するフォルダを選択してください。

FLAG翻訳オプション

表示する診断メッセージを指定します。

項目		説明
診断メッセージのレベル		診断メッセージのレベルを指定します。 初期値では[I]が選択されます。
	I	すべての診断メッセージを表示します。
	W	Wレベル以上の診断メッセージだけ表示します。
	E	Eレベル以上の診断メッセージだけ表示します。

注意

翻訳オプションCONFによる指摘メッセージは、FLAGの指定に関係なく表示されます。

FLAGSW翻訳オプション

COBOL文法の言語要素に対しての指摘メッセージを表示する(FLAGSW)か、しない(NOFLAGSW)かを指定します。

項目		説明
言語要素に対しての指摘メッセージ表示の可否		言語要素に対しての指摘メッセージ表示の可否を指定します。 初期値では[表示しない]が選択されます。
	表示する	COBOL文法の言語要素に対しての指摘メッセージを表示します。
	表示しない	COBOL文法の言語要素に対しての指摘メッセージを表示しません。
指摘する言語要素		指摘する言語要素を選択します。
	STDM	'85 ANSI COBOL 規格の下位レベル外の言語要素を指摘します。
	STDI	'85 ANSI COBOL 規格の中位レベル外の言語要素を指摘します。
	STDH	'85 ANSI COBOL 規格の上位レベル外の言語要素を指摘します。
	SIA	富士通システム統合アーキテクチャ (SIA) の範囲外の言語要素を指摘します。
	RPW	'85 ANSI COBOL 規格の報告書の言語要素を指摘します。 STDM、STDI、STDHが選択されている場合に選択できます。

注意

FLAGSW(SIA)は、他システムで動かすプログラムを作成するときに有効です。

FORMLIB翻訳オプション

IN/OF XMDLIB指定のCOPY文により画面帳票定義体からレコード定義を取り込む場合、画面帳票定義体ファイルのフォルダを指定します。
使用する画面帳票定義体ファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。フォルダを複数指定した場合、指定された順序でフォルダが検索されます。

項目	説明
画面帳票定義体ファイルのフォルダ	画面帳票定義体ファイルのフォルダを指定します。 フォルダはセミコロンで区切って複数追加できます。 [参照]をクリックすると、フォルダの指定方法を選択する[選択]ダイアログボックスが表示されます。[選択]ダイアログボックスに表示される一覧から、指定するフォルダのパスとなるパスの開始位置を指定してください。選択項目によっては、引き続き[フォルダの参照]ダイアログボックスが表示されます。指定するフォルダを選択してください。

GEN翻訳オプション

ソースプログラムリスト中に、AIM DBMSとの連絡領域(FCOM)およびAIM展開レコード域(UWA)を表示する(GEN)か、しない(NOGEN)かを指定します。

項目		説明
FCOMおよびUWAの表示		ソースプログラムリスト中に、AIM DBMSとの連絡領域およびAIM展開レコード域の表示の可否を指定します。 初期値では[表示しない]が選択されます。
	表示する	ソースプログラムリスト中に、AIM DBMSとの連絡領域およびAIM展開レコード域を表示します。

項目		説明
	表示しない	ソースプログラムリスト中に、AIM DBMSとの連絡領域およびAIM展開レコード域を表示しません。

INITVALUE翻訳オプション

作業場所節データのVALUE 句なし項目を指定値で初期化する(INITVALUE) か、しない(NOINITVALUE)かを指定します。

項目		説明
作業場所節でVALUE句なしの項目の扱い		作業場所節データのVALUE 句なし項目を指定値で初期化するか、しないかを指定します。 初期値では[初期化しない]が選択されます。
	初期化する	初期化します。
	初期化しない	初期化しません。
値		[初期化する]を選択したときに2桁の16進数を指定します。[初期化する]を選択したときには省略できません。

LANGLVL翻訳オプション

COBOLの旧規格と新規格との間で、ソースプログラムの解釈が異なる項目に対してどの規格に基づいて解釈するかを指定します。

項目		説明
ANSI COBOL規格の指定		ソースプログラムをどの規格に基づいて解釈するかを指定します。
	85	'85 ANSI COBOL規格として解釈します。
	74	'74 ANSI COBOL規格として解釈します。
	68	'68 ANSI COBOL規格として解釈します。

LIB翻訳オプション

登録集機能(COPY文)を使用する場合、登録集ファイルのフォルダを指定します。使用する登録集ファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。
フォルダを複数指定した場合、指定された順序でフォルダが検索されます。

項目	説明
登録集ファイルのフォルダ	登録集ファイルのフォルダを指定します。 フォルダはセミコロンで区切って複数追加できます。 [参照]をクリックすると、フォルダの指定方法を選択する[選択]ダイアログボックスが表示されます。[選択]ダイアログボックスに表示される一覧から、指定するフォルダのパスとなるパスの開始位置を指定してください。選択項目によっては、引き続き[フォルダの参照]ダイアログボックスが表示されます。指定するフォルダを選択してください。

LINECOUNT翻訳オプション

翻訳リストの1ページあたりの行数を指定します。
0から12までの値を指定すると、ページ替えのないベタ打ち表示となります。

項目	説明
翻訳リストの1ページあたりの行数	翻訳リストの1ページあたりの行数を指定します。 初期値には"60"が表示されます。 3桁以内の整数を指定してください。省略した場合、60が指定されたものとみなします。

LINESIZE翻訳オプション

翻訳リストの1行あたりの最大文字数(リスト上に表示されるA/N文字換算の値)を指定します。

項目	説明
翻訳リストの1行あたりの文字数	翻訳リストの1行あたりの最大文字数を指定します。 初期値には"136"が表示されます。 80 を指定するか、または 120 ～136の 3 けたの整数を指定できます。 文字数を省略した場合、136 が指定されたものとみなします。

注意

- ソースプログラムリスト、オプション情報リスト、診断メッセージリストおよび翻訳単位統計情報リストは、翻訳オプションLINESIZEに指定した最大文字数に関係なく固定の文字数(120)で出力されます。
- 文字数として有効な最大の値は136です。翻訳オプションLINESIZEに136より大きい値を指定した場合、136として扱われます。

LIST翻訳オプション

目的プログラムリストを出力する(LIST)か、しない(NOLIST)かを指定します。目的プログラムリストを出力する場合、目的プログラムリストは、翻訳オプションPRINTで指定したフォルダに出力されます。

項目		説明
目的プログラムリストの出力の可否		目的プログラムリストの出力の可否を指定します。 初期値では[出力しない]が選択されます。
	出力する	目的プログラムリストを出力します。
	出力しない	目的プログラムリストを出力しません。

注意

翻訳オプションPRINTが指定されていない場合、本オプションを指定しても、目的プログラムリストは出力されません。

MAP翻訳オプション

データマップリスト、プログラム制御情報リストおよびセクションサイズリストを出力する(MAP)か、しない(NOMAP)かを指定します。データマップリスト、プログラム制御情報リストおよびセクションサイズリストを出力する場合、これらのリストは翻訳オプションPRINTで指定したファイルに出力されます。

項目		説明
データマップリストの出力の可否		データマップリスト、プログラム制御情報リストおよびセクションサイズリストの出力の可否を指定します。 初期値では[出力しない]が選択されます。
	出力する	データマップリスト、プログラム制御情報リストおよびセクションサイズリストを出力します。
	出力しない	データマップリスト、プログラム制御情報リストおよびセクションサイズリストを出力しません。

注意

データマップリスト、プログラム制御情報リストおよびセクションサイズリストを出力するためには、本オプションのほかに翻訳オプションPRINTを指定する必要があります。

MESSAGE翻訳オプション

オプション情報リストおよび翻訳単位統計情報リストを出力する(MESSAGE) か、しない(NOMESSAGE)かを指定します。

項目		説明
オプション情報リスト、翻訳単位統計情報リストの出力の可否		オプション情報リスト、翻訳単位統計情報リストの出力の可否を指定します。 初期値では[出力しない]が選択されます。
	出力する	オプション情報リスト、翻訳単位統計情報リストを出力します。
	出力しない	オプション情報リスト、翻訳単位統計情報リストを出力しません。

MODE翻訳オプション

ACCEPT文の"ACCEPT 一意名 [FROM 呼び名]"の書き方で、受取り側項目に数字項目を指定したACCEPT文を実行する場合、受取り側項目に右詰めの数字転記を行う(MODE(STD))か、左詰めの文字転記を行う(MODE(CCVS))かを指定します。

項目		説明
ACCEPT文の動作の指定		ACCEPT文の動作を指定します。 初期値では[STD]が選択されます。
	STD	右詰めの数字転記を行います。
	CCVS	左詰めの数字転記を行います。

NCW翻訳オプション

利用者語に指定できる日本語文字集合をシステム共通な日本語文字集合とする(NCW(STD))か、計算機の日本語文字集合とする(NCW(SYS))かを指定します。

項目		説明
日本語利用者語の文字集合の指定		日本語利用者語の文字集合を指定します。 初期値では[STD]が選択されます。
	STD	システム共通な日本語文字集合とします。
	SYS	計算機の日本語文字集合とします。

STDを指定すると、次の日本語文字集合が日本語利用者語として利用できます。

- JIS第一水準
- JIS第二水準
- JIS非漢字(以下の文字)
  - ０、１、…、９
  - Ａ、Ｂ、…、Ｚ
  - ａ、ｂ、…、ｚ
  - ぁ、あ、ぃ、い、…、ん
  - ァ、ア、ィ、イ、…、ン、ヴ、ヵ、ヶ
  - ー（長音）、-（ハイフン）、－（負号）、々

SYSを指定すると、次の日本語文字集合が日本語利用者語として使用できます。

- STD指定の文字集合
- 拡張文字
- 拡張非漢字

・ 利用者定義文字

・ JIS非漢字(以下の文字は使用不可)

ー 、。，．・：；？ ！゛

ー ゜`＾￣＿／＼｜（）

ー ［］｛｝「」＋＝＜＞

ー ￥ ＄ ¢ £ ％ ＃ ＆ ＊ @

NSPCOMP翻訳オプション

後述する比較において、日本語空白の扱いを、日本語空白として扱う(NSPCOMP(NSP))か、ANK空白とみなす(NSPCOMP(ASP))かを指定します。日本語空白をANK空白とみなす場合には、日本語空白は2バイトのANK空白と等価に扱われます。

項目		説明
日本語空白とANK空白の扱い		日本語空白とANK空白の扱いを指定します。 初期値では[NSP]が選択されます。
	NSP	日本語空白とみなします。
	ASP	ANK空白とみなします。

NSPCOMP(ASP)オプションは、以下の比較に対して有効となります。

・ 日本語項目を作用対象とする日本語文字比較

・ 集団項目を作用対象とする文字比較

以下の比較に対しては無効です。

・ 日本語項目を含まない集団項目どうしの比較

・ 明または暗に属性が表示用でない項目を含む集団項目の比較

注意

・ INSPECT文、STRING文、UNSTRING文および索引ファイルのキー操作で行われる文字比較および日本語文字比較では、NSPCOMP(ASP)オプションを指定しても日本語空白はANK空白と等価に扱われません。

・ NSPCOMP(ASP)が指定された場合、字類条件JAPANESEでANK空白が日本語として扱われます。

・ RCS(UCS2)オプションが有効な場合、NSPCOMP(ASP)オプションは指定できません。

NUMBER翻訳オプション

翻訳時および実行時の各種リストで、ソースプログラム中の各行を識別するための行情報の行番号に、ソースプログラムの一連番号領域の値を使用する(NUMBER)か、コンパイラが生成した値を使用する(NONUMBER)かを指定します。

項目		説明
ソースプログラムの一連番号域の指定		ソースプログラムの一連番号域の扱いを指定します。 初期値では[コンパイラが生成した行番号とする]が選択されます。
	ソースプログラム中の行番号とする	一連番号領域に数字以外の文字が含まれている場合および一連番号が昇順になっていない場合、その行の行番号は、直前の正しい一連番号に1を加えた値に変更されます。ただし、一連番号が降順となる場合、一意の補正された番号がCOPY修飾値と同じ形式で付加されます。
	コンパイラが生成した行番号とする	行番号は、1から1きざみに昇順に与えられます。

注意

- NUMBERが指定されているときには、同一の一連番号が連続していても誤りではないものとみなされます。
- NUMBERを指定した場合、[問題]ビューのコンテキストメニュー[ジャンプ]は使用できません。

OBJECT翻訳オプション

目的プログラムを出力する(OBJECT)か、しない(NOOBJECT)かを指定します。目的プログラムを出力する場合、通常、ソースプログラムと同じフォルダにファイルが作成されます。変更したい場合には、出力先を指定してください。

項目	説明
目的プログラムの出力の可否	目的プログラムの出力の可否を指定します。 初期値では[出力する]が選択されます。
出力する	目的プログラムを出力します。
出力しない	目的プログラムを出力しません。
目的プログラムの出力先フォルダ	目的プログラムの出力先をソースプログラムと異なるフォルダにしたい場合、出力先のフォルダを指定します。 [参照]をクリックすると、フォルダの指定方法を選択する[選択]ダイアログボックスが表示されます。[選択]ダイアログボックスに表示される一覧から、指定するフォルダのパスとなるパスの開始位置を指定してください。選択項目によっては、引き続き[フォルダの参照]ダイアログボックスが表示されます。指定するフォルダを選択してください。

OPTIMIZE翻訳オプション

広域最適化された目的プログラムを作成する(OPTIMIZE)か、しない(NOOPTIMIZE)かを指定します。ただし、TEST翻訳オプションと同時に指定された場合は広域最適化されたターゲットプログラムは作成されません。

項目	説明
広域最適化の扱い	広域最適化の扱いを指定します。 初期値では[NOOPTIMIZE]が選択されます。
OPTIMIZE	広域最適化された目的プログラムを作成します。
NOOPTIMIZE	広域最適化された目的プログラムを作成しません。

PRINT翻訳オプション

翻訳リストを出力する場合に指定します。翻訳リストを出力する場合、通常、ソースプログラムと同じフォルダにファイルが作成されます。変更したい場合には、出力先を指定してください。

項目	説明
翻訳リストの出力フォルダ	翻訳リストの出力先フォルダ名を入力します。 [参照]をクリックすると、フォルダの指定方法を選択する[選択]ダイアログボックスが表示されます。[選択]ダイアログボックスに表示される一覧から、指定するフォルダのパスとなるパスの開始位置を指定してください。選択項目によっては、引き続き[フォルダの参照]ダイアログボックスが表示されます。指定するフォルダを選択してください。

QUOTE/APOST翻訳オプション

表意定数QUOTEおよびQUOTESとしてクォーテーションマーク(") を使う(QUOTE)か、アポストロフィ(')を使う(APOST)かを指定します。

項目	説明
表意定数QUOTEの扱い	表意定数QUOTEおよびQUOTESの扱いを指定します。 初期値では[QUOTE]が選択されます。

項目		説明
	QUOTE	クォーテーションマーク(") を使用します。
	APOST	アポストロフィ(')を使用します。

注意

ソースプログラム中の引用符は、このオプションの指定に関係なく、クォーテーションマークとアポストロフィのどちらでも使用できます。ただし、左側の引用符と右側の引用符は、同じでなくてはなりません。

RCS翻訳オプション

実行時のコード系をShift_JISにする(RCS(SJIS))か、Unicodeにする(RCS(UCS2))かを指定します。
RCS翻訳オプションダイアログボックスで指定できるオプションは、お使いになっているコンピュータにインストールされているNetCOBOLの製品バージョンにより異なります。

NetCOBOL V9.0L20以前がインストールされている場合

項目		説明
実行時コード系の指定		初期値では[SJIS]が選択されます。
	SJIS	実行時のコード系をShift_JISとします。
	UCS2	実行時のコード系をUnicodeとします。

NetCOBOL V10.0.0以降がインストールされている場合

項目		説明
実行時コード系の指定		初期値では[SJIS]が選択されます。
	SJIS	実行時のコード系をShift_JISとします。
	UTF16	実行時のコード系をUnicodeとします。
	UCS2	実行時のコード系をUnicodeとします。
Unicode環境でのエンディアン		実行時のコード系がUnicodeの場合のエンディアンを指定します。 初期値では[LE]が選択されます。
	LE	Unicode環境でのエンディアンをリトルエンディアンとします。
	BE	Unicode環境でのエンディアンをビッグエンディアンとします。

UTF16とUCS2は同義ですが、UTF16を使用することをお勧めします。

REP翻訳オプション

通常、リポジトリファイルは、ソースプログラムと同じフォルダに作成されます。変更したい場合には、出力先を指定してください。
また、指定されたフォルダは、リポジトリファイルの入力先フォルダとしても使用されます。

項目	説明
リポジトリファイルのフォルダ	リポジトリファイルの入出力先フォルダを指定します。 [参照]をクリックすると、フォルダの指定方法を選択する[選択]ダイアログボックスが表示されます。[選択]ダイアログボックスに表示される一覧から、指定するフォルダのパスとなるパスの開始位置を指定してください。選択項目によっては、引き続き[フォルダの参照]ダイアログボックスが表示されます。指定するフォルダを選択してください。

REPIN翻訳オプション

リポジトリファイルの入力先フォルダを指定します。リポジトリファイルが複数のフォルダに存在する場合、フォルダをセミコロンで区切って複数指定します。フォルダを複数指定した場合、指定された順番でフォルダが検索されます。

項目	説明
リポジトリファイルの入力先フォルダ	リポジトリファイルの入力先フォルダを指定します。 フォルダはセミコロンで区切って複数追加できます。 [参照]をクリックすると、フォルダの指定方法を選択する[選択]ダイアログボックスが表示されます。[選択]ダイアログボックスに表示される一覧から、指定するフォルダのパスとなるパスの開始位置を指定してください。選択項目によっては、引き続き[フォルダの参照]ダイアログボックスが表示されます。指定するフォルダを選択してください。

RSV翻訳オプション

予約語の種類を指定します。

項目		説明
予約語の種類		予約語の種類を指定します。 初期値では[ALL]が選択されます。
	ALL	使用するNetCOBOLのバージョン用です。
	V111	OSIV COBOL85 V11L11用です。
	V112	OSIV COBOL85 V11L20用です。
	V122	OSIV COBOL85 V12L20用です。
	V125	COBOL85 V12L50用です。
	V30	COBOL85 V30用です。
	V40	COBOL97 V40用です。
	V61	COBOL97 V61用です。
	V70	NetCOBOL V7.0用です。
	V81	NetCOBOL V8.1用です。
	V90	NetCOBOL V9.0用です。
	VSR2	VS COBOLII REL2.0用です。
	VSR3	VS COBOLII REL3.0用です。

SAI翻訳オプション

ソース解析情報ファイルを出力する(SAI)か、出力しない(NOSAI)かを指定します。ソース解析情報ファイルを出力する場合、通常、ソースプログラムと同じフォルダにファイルが作成されます。変更したい場合には、出力先を指定してください。

項目		説明
ソース解析情報ファイルの出力の可否		ソース解析情報ファイルの出力の可否を指定します。 初期値では[NOSAI]が選択されます。
	SAI	ソース解析情報ファイルを出力します。
	NOSAI	ソース解析情報ファイルを出力しません。
ソース解析情報の出力先フォルダ		ソース解析情報ファイルを出力するフォルダを指定します。[ソース解析情報ファイルの出力の可否]に[SAI]が選択された場合に指定できます。 [参照]をクリックすると、フォルダの指定方法を選択する[選択]ダイアログボックスが表示されます。[選択]ダイアログボックスに表示される一覧から、指定するフォルダのパスとなるパスの開始位置を指定してください。選択項目によっては、引き続き[フォルダの参照]ダイアログボックスが表示されます。指定するフォルダを選択してください。

SDS翻訳オプション

符号付き内部10進項目から符号付き内部10進項目への転記で、送り出し側項目の符号をそのまま転記する(SDS)か、整形された符号を転記する(NOSDS)かを指定します。
負号にはX'B'およびX'D'の2種類があり、その他は正号として扱われます。ここでいう整形された符号とは、送り出し側項目の符号が正ならばX'C'に、負ならばX'D'に変換することです。

項目		説明
符号付き10進項目の符号の整形の可否		符号付き10進項目の符号の整形の可否を指定します。 初期値では[そのまま転記する]が選択されます。
	そのまま転記する	そのまま転記します。
	符号整形して転記する	符号整形して転記します。

SHREXT翻訳オプション

マルチスレッドとなるオブジェクト形式の場合(THREAD(MULTI)指定)に、外部属性(EXTERNAL指定)のデータおよびファイルをスレッド間で共有する(SHREXT)か、共有しない(NOSHREXT)かを指定します。

項目	説明
マルチスレッドプログラムの外部属性に関する扱い	[スレッド間で外部データを共有する]または[スレッド間で外部データを共有しない]のどちらかを選択します。 初期値では[スレッド間で外部データを共有しない]が選択されます。

注意

オブジェクト形式がシングルスレッド(THREAD(SINGLE))の場合、確定翻訳オプションにはSHREXTと表示されますが、NOSHREXTとして翻訳が行われます。

SMSIZE翻訳オプション

PowerSORTが使用するメモリ容量をキロバイト単位の数字で指定します。

項目	説明
PowerSORTが使用するメモリ容量の指定	PowerSORTが使用するメモリ容量を指定します。

注意

- このオプションは、別製品PowerSORTをインストールしている場合に有効であり、インストールしていない場合は無効です。
- SORT文およびマージ文から呼び出されるPowerSORTが使用するメモリ空間の容量を限定したい場合に指定します。指定する値は、キロバイト単位の数字です。指定された値が実際に有効になるかについては、PowerSORTのオンラインマニュアルをお読みください。
- このオプションは、実行時オプションsmsizeおよび特殊レジスタSORT-CORE-SIZEに指定する値の意味と等価ですが、同時に指定された場合の優先順位は、特殊レジスタSORT-CORE-SIZEが一番強く、以降、実行時オプションsmsize、翻訳オプションSMSIZE()の順で弱くなります。

例

ー 特殊レジスタ MOVE 102400 TO SORT-CORE-SIZE
(102400=100キロバイトです)

ー 翻訳オプション SMSIZE(500K)

ー 実行時オプション smsize300k

この場合、一番強い特殊レジスタSORT-CORE-SIZEの値 100キロバイトを優先します。

SOURCE翻訳オプション

ソースプログラムリストを出力する(SOURCE)か、しない(NOSOURCE)かを指定します。ソースプログラムリストを出力する場合、ソースプログラムリストは、翻訳オプションPRINTで指定したフォルダに出力されます。

項目		説明
ソースプログラムリストの出力の可否		ソースプログラムリストの出力の可否を指定します。 初期値では[出力しない]が選択されます。
	出力する	ソースプログラムリストを出力します。
	出力しない	ソースプログラムリストを出力しません。

注意

翻訳オプションPRINTが指定されていない場合、本オプションを指定しても、ソースプログラムリストは出力されません。

SQLGRP翻訳オプション

SQLのホスト変数の定義方法を拡張する(SQLGRP)か、しない(NOSQLGRP)かを指定します。

項目		説明
SQLのホスト変数定義の拡張		SQLのホスト変数の定義方法の拡張の可否を指定します。 初期値では[拡張する]が選択されます。
	拡張する	拡張します。
	拡張しない	拡張しません。

SRF翻訳オプション

COBOLソースプログラムおよび登録集ファイルの正書法の種類を、固定形式にする(FIX)か、可変形式にする(VAR)かを指定します。

項目	説明
正書法の指定	[ソースプログラム形式]にはCOBOLソースプログラムに指定する正書法を[固定]、[可変]から選択します。 [ライブラリテキスト形式]には登録集に指定する正書法を[固定]、[可変]から選択します。 SRFの値はエディタの設定にしたがって初期設定されています。

SSIN翻訳オプション

小入出力機能のACCEPT文のデータの入力先を指定します。

項目		説明
ACCEPT文のデータの入力先		ACCEPT文のデータの入力先を指定します。 初期値では[SSIN(SYSIN)]が選択されます。
	SSIN	データの入力先としてファイルを使用します。
	SSIN(SYSIN)	データの入力先としてコンソールウィンドウを使用します。
実行環境変数の指定		入力元ファイルが設定されている環境変数情報名を指定します。

注意

環境変数情報名は英大文字(A～Z)で始まる8文字以内の英大文字および数字でなければなりません。また、環境変数情報名は、他のファイルで使用する環境変数情報名(ファイル識別名)と一致しないようにする必要があります。

SSOUT翻訳オプション

小入出力機能のDISPLAY文のデータの出力先を指定します。

項目		説明
DISPLAY文のデータの出力先		DISPLAY文のデータの出力先を指定します。 初期値では[SSOUT(SYSOUT)]が選択されます。
	SSOUT	データの出力先としてファイルを使用します。
	SSOUT(SYSOUT)	データの出力先としてコンソールウィンドウを使用します。
実行環境変数の指定		出力元ファイルが設定されている環境変数情報名を指定します。

注意

環境変数情報名は英大文字(A～Z)で始まる8文字以内の英大文字または数字でなければなりません。また、環境変数情報名は、他のファイルで使用する環境変数情報名(ファイル識別名)と一致しないようにする必要があります。

STD1翻訳オプション

ALPHABET句のEBCDIC指定で、英数字のコード(1バイト文字の標準コード)をASCII(ASCII)として取り扱うか、JIS8単位コード(JIS1)として取り扱うか、またはJIS7単位ローマ字コード(JIS2)として取り扱うかを指定します。

項目		説明
ALPHABET句のEBCDIC指定での英数字コードの扱い		ALPHABET句のEBCDIC指定での英数字コードの扱いを指定します。 初期値では[JIS2(JIS7単位コード)]が選択されます。
	STD1(ASCII)	文字符号系として"EBCDIC(ASCII)"を採用します。
	JIS1(JIS8単位コード)	文字符号系として"EBCDIC(カナ)"を採用します。
	JIS2(JIS7単位コード)	文字符号系として"EBCDIC(英小文字)"を採用します。

TAB翻訳オプション

タブの扱いを4カラム単位にする(TAB(4))か8カラム単位にする(TAB(8))かを指定します。

項目	説明
タブの扱い	タブの扱いを4カラムか、8カラムのどちらかを指定します。 初期値はエディタで設定されている値です。

TEST翻訳オプション

実行時に対話型デバッガおよび診断機能の使用を可能にする(TEST)か、しない(NOTEST)かを指定します。TESTを指定すると、対話型デバッガや診断機能で使用するデバッグ情報ファイルが作成され、通常はソースプログラムと同じフォルダに格納されます。変更したい場合には、出力先を指定してください。

項目		説明
対話型デバッガの使用の可否		対話型デバッガの使用の可否を指定します。 初期値では[使用しない]が選択されます。 ターゲットがデバッグの場合、[使用しない]は非活性となり[使用する]が選択されます。
	使用する	対話型デバッガを使用します。
	使用しない	対話型デバッガを使用しません。
デバッグ情報の出力先フォルダ		デバッグ情報ファイルの出力先フォルダを指定します。

注意

OPTIMIZEと同時に指定した場合、作成されたデバッグ情報ファイルは診断機能では使用できますが、デバッガでは使用できません。

THREAD翻訳オプション

オブジェクトの形式をマルチスレッドとする(THREAD(MULTI))か、シングルスレッドとする(THREAD(SINGLE))かを指定します。

項目	説明
マルチスレッドプログラム作成の指定	オブジェクトファイルの属性を[MULTI - マルチスレッド]または[SINGLE - シングルスレッド]で指定します。 初期値では[SINGLE -シングルスレッド]が選択されます。

TRACE翻訳オプション

TRACE機能を使用する(TRACE)か、しない(NOTRACE)かを指定します。

項目		説明
TRACE機能の使用の可否		TRACE機能の使用の可否を指定します。 初期値では[使用しない]が選択されます。
	使用する	TRACE機能を使用します。
	使用しない	TRACE機能を使用しません。
出力するトレース情報の個数		出力するトレース情報の個数を1～999999の整数で指定します。省略した場合、出力するトレース情報の個数は200個になります。

注意

- TRACEを指定すると、トレース情報を表示するための処理が目的プログラム中に組み込まれるため、実行性能が低下します。デバッグ終了時には、NOTRACEを指定して再翻訳してください。
- TRACEは、翻訳オプションCOUNTと同時に指定できません。同時に指定された場合、後に指定された方が有効となります。

TRUNC翻訳オプション

2進項目を受取り側項目とする数字転記で、上位桁の桁落としをする(TRUNC)か、しない(NOTRUNC)かを指定します。

項目		説明
けた落とし処理の可否		けた落とし処理の可否を指定します。 初期値では[けた落としをしない]が選択されます。
	けた落としをする	結果の値が受取り側項目のPICTURE句の記述に従って、上位桁が桁落としされ、受取り側項目に格納されます。翻訳オプションOPTIMIZEを同時に指定した場合、最適化によって外部10

項目		説明
		進項目または内部10進項目から導入されたデータ項目に対しても上位の桁落としが行われます。なお、送り出し側項目の整数部の桁数が、受取り側項目の整数部の桁数よりも大きい場合だけ、上記のような桁落としが行われます。
	けた落としをしない	目的プログラムの実行速度を優先します。桁落としを行わないほうが速く実行できる場合には、桁落としは行われません。

ポイント

PICTURE 句の記述によって桁落としされる場合と、桁落としされない場合の例を以下に示します。

- S999V9（整数部3桁）をS99V99（整数部2桁）に転記：桁落としあり
- S9V999（整数部1桁）をS99V99（整数部2桁）に転記：桁落としなし

注意

- NOTRUNCで、送り出し側項目の整数部の桁数が、受け取り側項目の整数部の桁数より大きい場合の結果は規定されません。
- NOTRUNCを指定する場合には、桁落としが行われなくても、受取り側項目にPICTURE句に記述した桁を超える値が格納されないように、プログラムを設計しなければなりません。
- NOTRUNCで桁落としを行うか行わないかの基準は、コンパイラによって異なります。したがって、NOTRUNCによって桁落としが行われないことを利用したプログラムは他システムへの互換が保証されないので注意してください。

XREF翻訳オプション

相互参照リストを出力する(XREF)か、しない(NOXREF)かを指定します。
相互参照リストには、利用者語や特殊レジスタなどが文字の大小順序の昇順に表示されます。相互参照リストは、翻訳オプションPRINTに指定したフォルダに出力されます。

項目		説明
相互参照リストの出力の可否		相互参照リストの出力の可否を指定します。 初期値では[出力しない]が選択されます。
	出力する	相互参照リストを出力します。
	出力しない	相互参照リストを出力しません。

注意

- 翻訳オプションPRINTが指定されていない場合、本オプションを指定しても、相互参照リストは出力されません。
- 翻訳オプションXREFが指定されている場合で、翻訳の結果、最大重大度コードがSレベル以上の場合、相互参照リストの出力は抑止されます。

ZWB翻訳オプション

符号付き外部10進項目を英数字フィールドと比較するときに、外部10進項目の符号部を無視して比較する(ZWB)か、符号部を含めて比較する(NOZWB)かを指定します。ここで、英数字とは、英数字項目、英字項目、英数字編集項目、数字編集項目、文字定数およびZERO以外の表意定数のことです。

項目	説明
符号付き外部10進項目と英数字項目の比較	符号付き外部10進項目を英数字フィールドと比較するときの比較方法を指定します。 初期値では[符号部を無視して比較する]が選択されます。

項目		説明
	符号部を無視して比較する	符号部を無視して比較します。
	符号部を含めて比較する	符号部を含めて比較します。

ポイント

```
77  ED  PIC  S9(3)  VALUE  +123.
77  AN  PIC  X(3)  VALUE  "123".
```

上記で、条件式 ED = AN の真偽は、以下のようになります。

- ZWB を指定した場合：真
- NOZWB を指定した場合：偽

6.1.3.3.3 既存の翻訳オプションファイルの利用

既存の翻訳オプションファイル（CBIファイル）を利用する場合は、以下の手順で行います。

1. Windowsのエクスプローラで既存の翻訳オプションファイルを選択し、コンテキストメニューから[コピー]を選択します。
2. 依存ビューまたは構造ビューで[その他のファイル]フォルダを選択し、[貼り付け]を選択します。プロジェクトに翻訳オプションファイルがコピーされ、[その他のファイル]フォルダに登録されます。
3. 依存ビューまたは構造ビューでプロジェクトを選択し、コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
4. 左のペインで[ビルド]を選択し、右のペインで[翻訳オプション]タブを選択します。
5. [オプションファイル]に、2.で貼り付けた翻訳オプションファイル名を入力し、[OK]をクリックします。

翻訳オプションファイルの変更

依存ビューまたは構造ビューで翻訳オプションファイルをダブルクリックすると、[翻訳オプション]ダイアログボックスが表示されます。このダイアログボックスで、翻訳オプションファイルの内容を変更できます。[翻訳オプション]ダイアログボックスの詳細については"NetCOBOL使用手引書"を参照してください。

注意

- プロジェクトのプロパティと翻訳オプションファイルで同じ翻訳オプションを指定した場合は、プロジェクトのプロパティで指定した値が優先されます。
- 翻訳オプションを変更した場合は、プロジェクトを再ビルドする必要があります。ビルドでは変更したオプションが有効にならない場合があります。

6.1.3.4 登録集名

登録集名を指定するCOPY文を含むCOBOLソースプログラムをビルドする場合、[登録集名]オプションを使用して、登録集名と登録集ファイルが格納されているフォルダを指定します。

登録集名を追加する

以下の手順で[登録集名]ページを表示し、登録集名を設定します。

1. 依存ビューまたは構造ビューからCOBOLプロジェクトまたはCORBAサーバプロジェクトを選択します。
2. メニューバーから[ファイル] > [プロパティ]を選択するか、コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。

3. 左のペインで[ビルド]を選択すると[ビルド]ページが表示されます。
4. [登録集名]タブを選択すると[登録集名]ページが表示されます。

項目	説明
追加	登録集オプションを追加します。 クリックすると[登録集名の追加]ダイアログボックスが表示されます。 [登録集名]に登録集名を指定します。[フォルダの選択]に登録集ファイルが格納されているフォルダを指定します。[参照]をクリックしてフォルダを選択することもできます。
変更	選択した登録集オプションを変更します。 クリックすると[登録集名の変更]ダイアログボックスが表示されます。 [登録集名の変更]ダイアログボックスでは登録集名を変更することはできません。フォルダを変更します。
削除	選択した登録集オプションを削除します。

注意

ソースプログラム中に登録集名が小文字で記述されているとき、翻訳オプションALPHAL(ALL)またはALPHAL(WORD)を指定して翻訳すると、登録集名は大文字として扱われます。そのため、ここで指定した登録集名が小文字の場合、意図した登録集ファイルが読み込めません。翻訳オプションALPHAL(ALL)またはALPHAL(WORD)を指定して翻訳する場合、ここで指定する登録集名は大文字で指定してください。

登録集名を指定しないCOPY文を使ったCOBOLソースプログラムを翻訳する場合には、翻訳オプションLIBに登録集ファイルの格納フォルダを指定します。

6.1.4 プリコンパイラ

プリコンパイラ機能によりプリコンパイラ入力ソースを含むCOBOLプログラムの開発ができます。

6.1.4.1 プリコンパイラ連携情報の初期値の設定・変更

プリコンパイラコマンドを呼び出す情報の初期値を設定・変更できます。
ここで設定した値は、ワークスペース内のCOBOLプロジェクトおよびCORBAサーバプロジェクトにプリコンパイラ連携情報を設定するときの初期値として共有されます。

以下の手順でプリコンパイラ連携情報の初期値を設定・変更します。

1. メニューバーから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
2. [設定]ダイアログボックスの左のペインで[COBOL] > [プリコンパイラ]を選択すると[プリコンパイラ]ページが表示されるので、初期値を設定・変更します。

項目	説明
プリコンパイラを使用する	現在使用しているワークスペース内のCOBOLプロジェクトまたはCORBAサーバプロジェクトでプリコンパイラを使用する場合に選択します。
プリコンパイラコマンド	プリコンパイラとして起動するコマンド名を指定します。
プリコンパイラのパラメタ	プリコンパイラコマンドのパラメタを指定します。
入力ソースの拡張子	プリコンパイラ入力ソースファイルの拡張子を指定します。 以下の拡張子を指定することはできません。 ・ cobol ・ cob ・ cbl ・ lcai

項目	説明
出力ソースの拡張子	プリコンパイラ出力ソースファイルの拡張子を選択します。
COBOLコンパイラのエラーメッセージをプリコンパイラ入力ソースの行番号で表示する	選択するとプリコンパイラ入力ソースの行対応情報をCOBOLソースファイルへ展開します(INSDBINFコマンドを呼び出します)。 初期値では選択されていません。
INSDBINFコマンドのパラメタ	プリコンパイルによって生成されたCOBOLソースファイルに、プリコンパイラ入力ソースに対する行補正情報を展開するINSDBINFコマンドのパラメタを指定します。 ただし、入力ソースファイル名と出力ソースファイル名は、プリコンパイラ入力ソースファイル名から決定されるため、指定する必要はありません。

プリコンパイラのコマンドパラメタへのファイル名の指定方法

プリコンパイラのコマンドパラメタへファイル名を指定する必要がある場合は、次のマクロを指定してください。プリコンパイラコマンド呼び出し時に、ファイル名が自動的に展開されます。

マクロ名	マクロの意味
%INFILE%	プリコンパイラ入力ソースのファイル名
%INFILE_BASE%	プリコンパイラ入力ソースの拡張子を取り除いたファイル名
%OUTFILE%	プリコンパイラ出力ソースのファイル名

INSDBINFコマンド

Oracle連携時には、INSDBINFコマンドを使用することによりCOBOLコンパイラとプリコンパイラ連携での、以下のような問題を解決します。

- コンパイラの出力する翻訳エラー検出行番号は、中間ファイル(プリコンパイル後のソースファイル)の行番号である。このため、利用者は中間ファイルを参照しながらプリコンパイラ入力ソース(プリコンパイル前の、埋込みSQL文が書かれたCOBOLソースプログラム)の修正を行わなければならない。
- 問題ビューの[ジャンプ]で、オリジナルソースプログラムの正しいエラー行にジャンプできない。
- オリジナルソースプログラムを被デバッグプログラムとして、デバッガでデバッグできない。

INSDBINFコマンドは行番号制御情報およびファイル名制御情報を埋め込んだ中間ファイルを生成します。

行番号制御情報(#LINE情報)は、プリコンパイルを行う前のソースの行番号を、それ以降に動作するコンパイラまたはプリコンパイラに通知するための情報です。

ファイル名制御情報(#FILE情報)は、オリジナルソースプログラムのファイル名や、プリコンパイラがインクルードしたファイルのファイル名を、それ以降に動作するコンパイラまたはプリコンパイラに通知するための情報です。

COBOLコンパイラは、INSDBINFコマンドが生成した中間ファイルを入力ファイルとすることにより、行番号制御情報およびファイル名制御情報を参照できます。それらの情報からオリジナルソースファイルとプリコンパイル後のソースファイル行番号の対応付けたオリジナルソースプログラムおよびインクルードファイルのファイル名の取得が可能となります。これにより、COBOLコンパイラとプリコンパイラの連携時の問題が解決されることになります。

INSDBINFコマンドの詳細は、"NetCOBOL 使用手引書"を参照してください。

6.1.4.2 プリコンパイラを使用したCOBOLプログラムの作成

プリコンパイラを使用するCOBOLプログラム開発は、COBOLプロジェクトまたはCORBAサーバプロジェクトの新規作成時にプリコンパイラを使用するプロジェクトであることを指示すれば、プリコンパイラを使用するためのビルド環境が設定されます。

既存のCOBOLプロジェクトまたはCORBAサーバプロジェクトでプリコンパイラを使用できるようにビルド環境を変更するには、以下の手順で設定します。

1. [ビルドツール]にプリコンパイラを追加
2. プリコンパイラ連携情報の設定

3. プリコンパイラ入力ソースの生成・追加

6.1.4.2.1 ビルドツールの設定

COBOLプロジェクトおよびCORBAサーバプロジェクトのビルド時にプリコンパイラを呼び出すためには、[ビルドツール]に[プリコンパイラ]が設定されている必要があります。

ビルドツールへ[プリコンパイラ]を追加する

COBOLプロジェクトまたはCORBAサーバプロジェクトのビルドツールに[プリコンパイラ]が設定されていない場合、以下の手順で[プリコンパイラ]を追加します。

1. [依存]または[構造]ビューから[プリコンパイラ]を追加するプロジェクトを選択します。
2. コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[ビルドツール]を選択すると[ビルドツール]ページが表示されます。
4. [追加]を選択すると[ビルドツール選択]ダイアログボックスが表示されます。
5. [プリコンパイラ]を選択して、[OK]をクリックします。
6. [ビルドツール]ページの[COBOLビルダに関連付けられたビルドツールの一覧]に[プリコンパイラ]が追加されていることを確認して、[プロパティ]ダイアログボックスの[OK]をクリックします。

ビルドツールから [プリコンパイラ]を削除する

以下の手順でCOBOLプロジェクトまたはCORBAサーバプロジェクトから[プリコンパイラ]を削除します。ただし、[ソースファイル]フォルダにプリコンパイラ入力ソースが登録されている場合は、削除することはできません。

1. 依存ビューまたは構造ビューから[プリコンパイラ]を削除するプロジェクトを選択します。
2. コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[ビルドツール]を選択すると[ビルドツール]ページが表示されます。
4. [ビルドツール]ページの[COBOLビルダに関連付けられたビルドツールの一覧]から[プリコンパイラ]を選択し、[削除]をクリックします。
5. [COBOLビルダに関連付けられたビルドツールの一覧]から[プリコンパイラ]が削除されていることを確認して、[プロパティ]ダイアログボックスの[OK]をクリックします。

注意

[ソースファイル]フォルダにプリコンパイラ入力ソースが登録された状態ではビルドツールから[プリコンパイラ]を削除することはできません。[ソースファイル]フォルダからプリコンパイラ入力ソースを削除してから実行してください。

6.1.4.2.2 プリコンパイラ連携情報の設定・変更

プリコンパイラ入力ソースを含むCOBOLプロジェクトまたはCORBAサーバプロジェクトをビルドする際に呼び出すプリコンパイラコマンドの情報を、設定・変更できます。

ビルドツールに[プリコンパイラ]を追加した直後は、[設定]ダイアログボックスの[プリコンパイラ]ページでプリコンパイラ連携情報の初期値が設定されていればその値が初期値となります。

プリコンパイラ連携情報を設定・変更する

以下の手順でCOBOLプロジェクトまたはCORBAサーバプロジェクトにプリコンパイラの情報を設定・変更します。

1. 依存ビューまたは構造ビューでプリコンパイラ連携情報を設定するプロジェクトを選択します。
2. コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[ビルドツール] > [プリコンパイラ]を選択すると[プリコンパイラ]ページが表示されます。

プリコンパイラページの詳細については"6.1.4.1 プリコンパイラ連携情報の初期値の設定・変更"を参照してください。

注意

[ソースファイル]フォルダにプリコンパイラ入力ソースが登録されている場合、[プリコンパイラ]ページの[入力ソースの拡張子]は変更できません。プリコンパイラ入力ソースの拡張子を変更する場合は、[ソースファイル]フォルダからプリコンパイラ入力ソースを削除してから再実行してください。

6.1.4.2.3 プリコンパイラ入力ソースの生成・追加

プリコンパイラ入力ソースをプロジェクトに追加するには、あらかじめプロジェクトにプリコンパイラ連携情報が設定されていなければなりません。

プリコンパイラ入力ソースは、COBOLプロジェクトまたはCORBAサーバプロジェクトの[ソースファイル]フォルダに登録されていなければなりません。[ソースファイル]フォルダに登録するには、次の2つの方法があります。

- プリコンパイラ入力ソースを新規に作成して登録する
- 既存のプリコンパイラ入力ソースを登録する

プリコンパイラ入力ソースを新規作成して[ソースファイル]フォルダへ登録するには、[COBOLソース生成ウィザード]または[オブジェクト指向COBOLソース生成ウィザード]を使用します。

ウィザードの1枚目にある[プリコンパイラを使用する]チェックボックスをチェックすると、生成したCOBOLソースまたはオブジェクト指向COBOLソースが、プリコンパイラ入力ソースとして[ソースファイル]フォルダに登録されます。

以下の手順でウィザードを起動します。

1. 依存ビューまたは構造ビューでプリコンパイラ入力ファイルを生成するCOBOLプロジェクトの[ソースファイル]フォルダを選択します。
2. メニューバーから[ファイル] > [新規] > [COBOLソース] または[オブジェクト指向COBOLソース]を選択するか、コンテキストメニューから[新規] > [COBOLソース] または[オブジェクト指向COBOLソース]を選択すると、ウィザードが起動します。

注意

プロジェクトにプリコンパイラ連携情報が設定されていない場合は、[プリコンパイラを使用する]チェックボックスが無効となります。プリコンパイラ連携情報を設定してから、再度ウィザードを起動してください。

既存のプリコンパイラ入力ソースを、COBOLプロジェクトまたはCORBAサーバプロジェクトの[ソースファイル]フォルダへプリコンパイラ入力ソースとして登録できます。

- プロジェクト内のプリコンパイラ入力ソースの場合

  以下の手順でプロジェクト内のプリコンパイラ入力ソースを追加します。

  1. 依存ビューまたは構造ビューで[ソースファイル]フォルダを選択します。
  2. コンテキストメニューから[ファイルの追加]を選択します。
  3. 選択したプロジェクトに対応する[ファイル一覧]ダイアログボックスが表示されます。
  4. [ソースファイル]フォルダに追加するプリコンパイラ入力ソースを選択します。
  5. [OK]をクリックすると、選択したプリコンパイラ入力ソースが[ソースファイル]フォルダに登録されます。

- プロジェクト外のプリコンパイラ入力ソースの場合

  プロジェクト外に存在するプリコンパイラ入力ソースは以下のいずれかの方法で登録します。

  ー Windowsのエクスプローラで登録するプリコンパイラ入力ソースを選択し、コンテキストメニューから[コピー]を選択します。[ソースファイル]フォルダを選択し、コンテキストメニューから[貼り付け]を選択します。

  ー Windowsのエクスプローラから[ソースファイル]フォルダへ、プリコンパイラ入力ソースをドラッグ&ドロップします。

注意

プロジェクトにプリコンパイラ連携情報が登録されていない場合はエラーとなります。[ソースファイル]フォルダにファイルは登録されますが、プリコンパイラ入力ソースとして登録はされません。このままの状態でプロジェクトにプリコンパイラ連携情報を設定しても上記の操作で追加したファイルがプリコンパイラ入力ソースとして自動的に再登録はされません。[ソースファイル]フォルダから[その他のファイル]フォルダへ移動して、再度[ソースファイル]フォルダへ登録してください。

6.1.4.2.4 プリコンパイラ入力ソースの編集

プリコンパイラ入力ソースの内容をCOBOLエディタで編集するには、プリコンパイラ入力ソースファイルの拡張子をコンテンツタイプおよびCOBOLエディタに関連付ける必要があります。プリコンパイラ入力ソースファイルの拡張子のコンテンツタイプおよびCOBOLエディタへの関連付けは、プリコンパイラ連携情報を設定したときに自動的に行われます。

プリコンパイラ入力ソースファイルがCOBOLエディタで開かれない場合は、以下の手順でファイルの拡張子をコンテンツタイプおよびCOBOLエディタに関連付けます。

1. メニューバーから[ウィンドウ] > [設定]を選択すると、[設定]ダイアログボックスが表示されます。
2. [設定]ダイアログボックスの左のペインで[一般] > [コンテンツタイプ]を選択すると、[コンテンツタイプ]ページが表示されます。
3. [コンテンツタイプ]から[テキスト] > [COBOLソースファイル]を選択します。
4. [追加]をクリックすると、[新規ファイルタイプ]ダイアログボックスが表示されます。
5. [新規ファイルタイプ]ダイアログボックスの[ファイルタイプ]にプリコンパイラ入力ソースファイルの拡張子を入力して、[OK]をクリックします。
6. [コンテンツタイプ]ページの[ファイルの関連付け]にプリコンパイラ入力ソースファイルの拡張子が表示されていることを確認し、[OK]をクリックします。
7. [設定]ダイアログボックスの左のペインで[一般] > [エディタ] > [ファイルの関連付け]を選択すると、[ファイルの関連付け]ページが表示されます。
8. [ファイルの関連付け]ページの[ファイルタイプ]の右側の[追加]をクリックすると、[新規ファイルタイプ]ダイアログボックスが表示されます。
9. [新規ファイルタイプ]ダイアログボックスの[ファイルタイプ]に、プリコンパイラ入力ソースファイルの拡張子を入力して、[OK]をクリックします。[ファイルの関連付け]ページの[ファイルタイプ]に指定した拡張子が追加されます。
10. [ファイルの関連付け]ページの[ファイルタイプ]から追加した拡張子を選択して、[関連付けられたエディタ]の右側の[追加]をクリックすると、[エディタの選択]ダイアログボックスが表示されます。
11. [エディタの選択]ダイアログボックスの[内部エディタ]が選択されていることを確認し、[COBOLエディタ]を選択します。
12. [ファイルの関連付け]ページの[関連付けられたエディタ]に[COBOLエディタ]が追加されます。

6.1.5 リンカ

オブジェクトプログラムをリンクして、実行ファイルまたはダイナミックリンクライブラリを作成できます。

6.1.5.1 リンクオプションの設定

リンクする際のオプションを設定できます。

リンクオプションを設定する

以下の手順で[リンクオプション]ページを表示し、リンクオプションを設定します。

1. 依存ビューまたは構造ビューからCOBOLプロジェクトまたはCORBAサーバプロジェクトを選択します。
2. メニューバーから[ファイル] > [プロパティ]を選択するか、コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインから[ビルド]を選択すると[ビルド]ページが表示されます。
4. [リンクオプション]タブを選択すると[リンクオプション]ページが表示されます。

項目	説明
ライブラリ/オブジェクトファイル	設定されているライブラリ/オブジェクトファイルの一覧が表示されます。
追加	リンクするライブラリ/オブジェクトファイルを追加します。 クリックすると[リンクオプションの変更]ダイアログボックスが表示され、ライブラリを選択します。 [参照]をクリックすると[選択]ダイアログボックスが表示され、ファイルが選択できます。 ライブラリファイルは複数追加できます。
変更	選択したライブラリファイルを変更します。
削除	選択したライブラリファイルを削除します。
すべて削除	[ライブラリ/オブジェクトファイル]に表示されているライブラリ/オブジェクトファイルをすべて削除します。
上へ	選択したライブラリファイルのリンク順序を変更します。
下へ	選択したライブラリファイルのリンク順序を変更します。
Cランタイムライブラリ名	リンク時に結合するCランタイムライブラリのファイル名を指定します。 Cランタイムライブラリ名を省略すると"LIBC.lib"が結合されます。
DLLエントリオブジェクト	COBOLで作成されたオブジェクトファイルだけでダイナミックリンクライブラリを作成するか、他言語で作成されたオブジェクトファイルと一緒にダイナミックリンクライブラリを作成するかを指定します。
COBOL単体用	COBOLで作成されたオブジェクトファイルだけでダイナミックリンクライブラリを作成します。
他言語間結合用	他言語で作成されたオブジェクトファイルと一緒にダイナミックリンクライブラリを作成します。
その他のオプション	追加のリンクオプションを入力します。2つ以上のオプションを指定する場合は、区切り文字としてスペースを挿入します。

注意

リンクオプションを指定するダイアログボックスで[参照]を選択し、[プロジェクトから選択]ダイアログボックスからファイルを選択した場合、パスの区切り文字は"¥"ではなく"/"となりますが、ビルドには問題ありません。

その他のオプションに指定できるリンクオプションを以下に示します。リンクオプションの詳細については"NetCOBOL 使用手引書"を参照してください。

指定形式	説明
/DEF:モジュール定義ファイル名	モジュール定義ファイルを指定します。
/EXPORT:外部参照名	外部参照情報を生成します。
/OUT:filename	メイン出力ファイルの名前を指定します。
/STACK:スタックサイズ	スタックサイズの変更を指定します。 省略した場合のスタックサイズは1Mバイトになります。スタックサイズはバイト単位で指定してください。
/MAP:filename	マップファイルを生成する場合に指定します。

6.1.5.2 ターゲットオプションの設定

プロジェクトごとに、以下のターゲットオプションを設定できます。

- ターゲット名
- ターゲット種別
- 作成するアプリケーションの形式

・ ビルドモード

ターゲットオプションを設定する

以下の手順で[ターゲット]ページを表示し、ターゲットオプションを設定します。

1. 依存ビューまたは構造ビューからCOBOLプロジェクトまたはCORBAサーバプロジェクトを選択します。
2. メニューバーから[ファイル] > [プロパティ]を選択するか、コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[ターゲット]を選択すると[ターゲット]ページが表示されます。

項目	説明
ターゲット名	ターゲット名を指定します。 初期値として「現在のプロジェクト名」が設定されています。
ターゲット種別	出力するターゲット種別を指定します。[実行ファイル]、[ダイナミックリンクライブラリ]、[CORBAサーバ]が選択できます。初期値では[実行ファイル]が設定されています。
DLL固有の実行用の初期化ファイル(COBOL85.CBR)を使用する	ターゲット種別がダイナミックリンクライブラリの場合、DLL固有の実行用の初期化ファイルを使用するか否かを指定します。 選択すると、DLL固有の実行用の初期化ファイルを使用するダイナミックリンクライブラリになります。
作成するアプリケーションの形式	作成するアプリケーションの形式を指定します。[COBOLのコンソールを使用するアプリケーション]、[システムのコンソールを使用するアプリケーション]が選択できます。初期値では[COBOLのコンソールを使用するアプリケーション]が設定されています。
ビルドモード	ビルドモードとして[リリース]または[デバッグ]を指定します。 初期値ではデバッグモードが設定されています。

6.1.6 リソースコンパイラ

リソースコンパイラは、アプリケーションにバージョン情報を付加する場合や、アイコンを設定する場合に利用するビルドツールです。
リソースの定義をリソース定義ファイル(.rcファイル)に記述し、[ソースフォルダ]に追加した場合に本ビルドツールが有効になります。

リソースコンパイラの設定

以下の手順で[リソースコンパイラ]ページを表示し、[リソースコンパイラ(RC)オプション]にリソースコンパイラオプションを指定します。

1. 依存ビューまたは構造ビューからCOBOLプロジェクトまたはCORBAサーバプロジェクトを選択します。
2. メニューバーから[ファイル] > [プロパティ]を選択するか、コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[ビルドツール] > [リソースコンパイラ]を選択すると[リソースコンパイラ]ページが表示されます。

注意

リソースコンパイラオプションについては、Microsoft(R) Resource Compilerの情報を参照してください。

6.2 プロジェクトをデバッグする

デバッガでは、開発したアプリケーションを実行させることにより、アプリケーションの論理的な誤りを検出できます。
ここではデバッガの概要、デバッガを使用する場合の留意事項およびデバッガの起動方法について説明します。

6.2.1 デバッグのビュー

デバッグパースペクティブを構成する各ビューとエディタ領域について、デバッグ時の用途とCOBOLのデバッグ時に使用可能かどうかを以下に示します。

画面	COBOLでの使用	用途
デバッグビュー	○	プログラムの実行状態(スレッドやスタックの状態)を表示します。 ただし、[ステップリターン]、[フレームにドロップ]、[ステップフィルタ/ステップデバッグの使用]は使用することはできません。
変数ビュー	×	COBOLでは機能しません。
ブレークポイントビュー	○	設定されているブレークポイントの一覧が表示されます。ブレークポイントに関する操作(追加、削除など)が行えます。 ただし、[Java例外ブレークポイントの追加]は使用できません。
式ビュー	×	COBOLでは機能しません。
ウォッチビュー	○	COBOLのデータ項目に関する情報が表示されます。プログラムが中断している際にデータ項目の値の参照、設定が行えます。
エディタ領域	○	プログラムの中断している行がソース上で表示されます。ブレークポイントの追加や、データチップによる変数値の参照を行うことができます。 また、実行を再開したい行にカーソル位置を合わせ、コンテキストメニューから[実行開始位置変更]を選択することにより、誤りのある文などを迂回できます。
アウトラインビュー	○	エディタで表示しているソースの構造を表示します。
コンソールビュー	×	COBOLでは機能しません。

6.2.2 デバッガの機能

COBOLデバッガには以下の機能があります。

ブレークポイントの設定

- COBOL処理記述にブレークポイントを設定
- ヒットカウントの指定

データの表示と変更

- データ項目の値の表示と変更

ブレークポイントの表示

- ブレークポイントをソース上で表示
- コールスタックの表示
- 設定したブレークポイント一覧の表示と設定の変更

スレッド

- スレッドの状態表示
- スレッドの状態変更

データ項目監視ポイントの設定

- データ項目の値変更時に中断

6.2.3 デバッガ使用時の留意事項

COBOLデバッガを使用する場合の留意事項を以下に示します。

COBOLを使用する場合の共通事項

- COBOL処理記述をデバッグする場合、デバッグ情報を生成するように設定してビルドを行います。
- デバッグする場合は、ビルド時と同じバージョンのCOBOL製品を使用してください。

- サーバアプリケーションをデバッグする場合、サーバ側での応答の監視によってタイムアウトが発生し、デバッグを行えないことがあります。デバッグが行えない場合には、監視時間の設定を長くするか、監視そのものをはずしてからデバッグを行ってください。
- COBOLデバッガは複数のデバッグセッションをサポートしません。

  COBOLプログラムのデバッグ中に、他のプログラム(COBOL、Java等)のデバッグを開始することはできません。

リモートデバッグする場合の留意事項

- CORBAサーバなどをデバッグするためには、Interstage Application Serverが同一端末にインストールされている必要があります。
- サーバアプリケーションをデバッグする場合、サーバ側での応答の監視によってタイムアウトが発生し、デバッグを行えないことがあります。デバッグが行えない場合には、監視時間の設定を長くするか、監視そのものをはずしてからデバッグを行ってください。

6.2.4 デバッグ情報を付加してビルドする

COBOL処理記述をデバッグする場合、デバッグ情報を生成する設定でビルドする必要があります。

ビルド時にデバッグ情報を生成するかどうかについては、プロジェクトの[プロパティ]ダイアログボックスより[ターゲット]を選択し、[ビルドモード]で設定します。

指定した設定は、指定したプロジェクトに対してだけ有効になります。

ポイント

デバッグ情報(.SVD)は、COBOLソースファイル単位に生成されます。デバッグ時には、デバッグ対象の実行ファイル(.EXE)と同じフォルダに格納しておく必要があります。そのため、リモートデバッグでサーバとして[ローカルマシン]を選択した場合にはデバッグ情報ファイルも実行ファイルと一緒にコピーする必要があります。

6.2.5 デバッガを起動する

起動構成との組み合わせで、各アプリケーションのローカルデバッグとリモートデバッグをサポートします。

起動構成

以下に、デバッグを開始するまでの手順を説明します。

1. デバッグするアプリケーションに対応した起動構成を作成します。
2. 起動構成に必要な情報を設定します。
3. デバッグを開始します。

起動構成の種別により、起動構成の作成方法が異なります。
起動構成による違いについて、説明します。

- COBOLアプリケーション起動構成を使う
- リモートCOBOLアプリケーション起動構成を使う
- CORBAワークユニット起動構成を使う

注意

作成した起動構成は、デバッグ時に毎回作成する必要はありません。
同じ起動構成でデバッグする場合は、[構成およびデバッグ]ダイアログボックスの[構成]から作成した構成ファイルを選択し、[デバッグ]をクリックしてデバッグを開始してください。ツールバー上のデバッグアイコンの▼部分をクリックして、作成したときに[名前]に指定した名前を選択することで素早くデバッグを開始することもできます。

COBOLアプリケーション起動構成を使う

作成したCOBOLアプリケーションをデバッグする場合、起動構成はCOBOLアプリケーションを使用します。

1. メニューバーから[実行] > [構成およびデバッグ]を選択し、[構成およびデバッグ]ダイアログボックスを表示します。
2. [構成およびデバッグ]ダイアログボックスより、[COBOLアプリケーション]を選択し、ダブルクリックします。
3. 右のペインに表示された所定のフィールドにそれぞれの値を指定します。
   [名前]には、任意の名前を指定します。たとえば、プロジェクトの名前を指定します。[メイン]タブでは、以下の値を指定します。
   [プロジェクト名]:プロジェクトの名前を指定します。
   [実行ファイル]:起動する実行ファイル名を指定します。
   [作業フォルダ]:実行時のカレントフォルダ名を指定します。
   [プログラム引数]:実行ファイルに渡すプログラム引数などを指定します。
   その他のタブについても必要に応じて値の指定、変更を行います。
4. [デバッグ]をクリックして、デバッグを開始します。

リモートCOBOLアプリケーション起動構成を使う

リモートCOBOLアプリケーション起動構成を使うことにより、サーバ上でビルドしたCOBOLアプリケーションをワークベンチからデバッグできます。また、アタッチデバッグを使用することによりワークベンチから直接実行できないCOBOLアプリケーションおよびCORBAサーバアプリケーションをデバッグできます。
メニューバーから[実行] > [構成およびデバッグ]を選択し、[構成およびデバッグ]ダイアログボックスを表示します。

1. [構成およびデバッグ]ダイアログボックスより、[リモートCOBOLアプリケーション]を選択し、ダブルクリックします。
2. 右のペインに表示された所定のフィールドにそれぞれの値を指定します。
   [名前]には、任意の名前を指定します。たとえば、プロジェクトの名前を指定します。
   [メイン]タブでは以下の値を指定します。

   通常デバッグの場合

   [プロジェクト名]:プロジェクトの名前を指定します。
   [デバッグ方法]:[通常デバッグ]を指定します。
   [サーバ名]:リモートデバッグで使用するサーバ情報の名前を指定します。
   [ポート番号]:サーバとの通信で使用するリモートデバッガコネクタのポート番号を指定します。
   [実行ファイル]:起動する実行ファイル名を指定します。
   [作業フォルダ]:実行時のカレントディレクトリ名を指定します。
   [プログラム引数]:実行ファイルに渡すプログラム引数などを指定します。
   その他のタブについても必要に応じて値の指定、変更を行います。

   アタッチデバッグの場合

   [プロジェクト名]:プロジェクトの名前を指定します。
   [デバッグ方法]:[アタッチデバッグ]を指定します。
   [デバッグ情報フォルダ]:デバッグするCOBOLアプリケーションと異なるディレクトリにデバッグ情報ファイルが格納されている場合、そのディレクトリのフルパス名を指定します。
   その他のタブについても必要に応じて値の指定、変更を行います。
3. [デバッグ]をクリックして、デバッガを起動します。
4. アタッチデバッグの場合は、デバッグするプログラムを起動します。アタッチデバッグするプログラムは、デバッガにプログラムの起動を通知するために環境変数を設定する必要があります。詳細は"5.5.2 アタッチデバッグ"の"サーバ側アプリケーションの実行"または"ローカルPC上のCOBOLアプリケーションをアタッチデバッグする"を参照してください。

CORBAワークユニット起動構成を使う

CORBAワークユニット起動構成を使うことにより、ローカルPC上のCORBAサーバアプリケーションをデバッグできます

1. メニューバーから[実行] > [構成およびデバッグ]を選択し、[構成およびデバッグ]ダイアログボックスを表示します。
2. [構成およびデバッグ]ダイアログボックスより、[CORBAワークユニット]を選択し、ダブルクリックします。
3. 右のペインに表示された所定のフィールドにそれぞれの値を指定します。
   [名前]には、任意の名前を指定します。たとえば、プロジェクトの名前を指定します。
   [メイン]タブでは以下の値を指定します。
   [CORBAワークユニット名]:デバッグに使用するCORBAワークユニット名を指定します。
   [起動前に配備する]:資産を自動的に配備する場合チェックします。

[配備するプロジェクト名]:プロジェクトの名前を指定します。
その他のタブについても必要に応じて値の指定、変更を行います。

4. [デバッグ]をクリックして、デバッガを起動します。

CORBAワークユニットの作成

CORBAワークユニット起動構成を使ってデバッグを行うためには、[構成およびデバッグ]ダイアログボックスの[メイン]タブの[CORBAワークユニット名]に指定するCORBAワークユニットがInterstage Application Serverに作成されている必要があります。CORBAワークユニットはInterstage管理コンソールで作成します。

注意

Interstageの管理に必要なサービス(基盤サービス)が起動されていない場合、[CORBAワークユニット名]にCORBAワークユニットの一覧が正しく表示されません。このような場合には基盤サービスを起動してからCORBAワークユニット起動構成を作成してください。
基盤サービスの起動にはInterstage基盤サービス操作ツールを使用します。ツールを起動するには、スタートメニューから [Interstage] > [Studio] > [Interstage基盤サービス操作ツール]を選択してください。

注意

CORBAワークユニット起動構成を用いたデバッグでは、ローカルのInterstage Application Serverへの接続に必要なサービスが停止している場合にはそれらのサービスの起動を行います。Windows VistaなどのUser Account Control(UAC)機能を持つOSでは、サービスの起動を行う際に管理者権限への昇格を求めるダイアログボックスが表示されます。その場合には表示に従って管理者権限への昇格を行ってください。

ポイント

依存ビューまたは構造ビューでCORBAサーバプロジェクトを選択し、メニューバーから[実行] > [デバッグ] > [CORBAワークユニット]を選択することにより、デフォルトの設定でデバッガを起動できます。

ポイント

.deploymentファイルの設定

CORBAワークユニットでは、デバッグを行う前に資産を自動的に配備できますが、配備時に必要な情報は.deploymentファイルに記述されているものを使用します。.deploymentファイルの内容を以下に示します。

タグ	説明
implementation-repository-id	インプリメンテーションリポジトリIDを指定します。
interface-repository-id	インタフェースリポジトリIDを指定します。
binding-name	ネーミングサービス登録名を指定します。
workunit-name	CORBAワークユニット名を指定します。

.deploymentファイルは[CORBAサーバアプリケーション生成ウィザード]の実行時に、ウィザードで指定されたモジュール名とインタフェース名の情報をもとに生成されます。

6.3 プロジェクトを実行する

ビルドが完了したら、アプリケーションを実行できます。

ここでは、実行する方法について説明します。

6.3.1 実行環境情報

実行環境情報は、環境変数情報とエントリ情報の2種類に分けられます。

環境変数情報は、COBOLランタイムが処理の開始時に環境固有情報を得るために使用されます。エントリ情報は、動的プログラム構造の情報を指定するために使用されます。

ワークベンチでは、プロジェクト新規作成時に生成される実行用初期化ファイル(COBOL85.CBR)を、依存ビューやナビゲータビューでダブルクリックすることで、実行環境設定ツールを起動し、実行環境情報を設定できます。
実行環境情報の詳細については、"NetCOBOL 使用手引書"を参照してください。

6.3.2 プロジェクトの実行方法

アプリケーションの実行方法について説明します。

COBOLアプリケーションの実行方法

ワークベンチからの実行

1. 構造ビューまたは依存ビューでCOBOLプロジェクトを選択します。
2. メニューバーから[実行] > [構成および実行]を選択します。[構成および実行]ダイアログボックスが表示されます。
3. 左のペインで[COBOLアプリケーション]を選択し、ダブルクリックします。
4. 右のペインに起動構成の設定ページが表示されます。
5. デフォルトで起動構成名が[名前]に表示されます。起動構成名は任意の名前に変更できます。
6. [メイン]タブをクリックします。
7. [プロジェクト名]にCOBOLプロジェクト名を入力します。または[参照]をクリックしてCOBOLプロジェクトを選択します。
8. 指定したプロジェクトのターゲットがダイナミックリンクライブラリの場合には、[実行ファイル]を指定します。
9. [作業フォルダ]には、実行時のカレントフォルダを指定します。実行に必要なダイナミックリンクライブラリをそのフォルダに格納しておくことで、実行時にロードできます。
10. [プログラム引数]にコマンドラインで指定する形式でパラメタを入力します。
11. [実行]をクリックすることでプログラムが実行されます。

注意

作成した起動構成は、アプリケーションの実行時に毎回作成する必要はありません。
同じ起動構成で実行する場合は、[構成および実行]ダイアログボックスの[構成]から作成した構成ファイルを選択し、[実行]をクリックしてCOBOLアプリケーションを実行してください。ツールバー上の実行アイコンの▼部分をクリックして、作成したときに[名前]に指定した名前を選択することで素早くアプリケーションを実行することもできます。

コマンドによる実行

実行ファイル名を直接起動します。プログラム引数が必要な場合には、コマンドラインに引数を指定します。
実行に必要なDLLについては、実行ファイルと同じファイルまたは環境変数PATHで参照可能なフォルダに格納する必要があります。

6.4 COBOLアプリケーション起動構成の設定項目

COBOLアプリケーション起動構成のメインタブ以外の設定項目について説明します。

[ソース]タブ

デバッグ時に使用するソース検索パスを指定します。[ソース検索パス]に表示されている順にソースファイルを検索します。検索パスには、プロジェクトやフォルダを追加できます。追加された検索パスは[上へ]、[下へ]を使用して、順番を入れ替えることができます。

注意

ソース検索パスに外部フォルダを指定した場合には、外部フォルダのCOBOLソースに対して以下の機能が正しく動作しません。

- ブレークポイント操作
- 編集画面のコンテキストメニューの[指定行まで実行]

上記機能を使用する場合、ソース検索パスにはプロジェクトを指定することをお勧めします。なお、外部フォルダのCOBOLソースをデバッグする場合は、ステップイン/ステップオーバーなどで、各行をデバッグしてください。

[環境]タブ

COBOLランタイムが処理の開始時に環境固有情報を得るために使用する情報を設定します。

注意

実行用初期化ファイルと[環境]タブで設定した環境変数が重複した場合は、実行用初期化ファイルで設定した情報が有効となります。実行環境情報の詳細については"NetCOBOL 使用手引書"を参照してください。

[共通]タブ

起動構成の情報の保存方法や、実行・デバッグ起動後に開かれるパースペクティブなどを指定します。

[別名保存]

起動構成のタイプに[ローカルファイル]または[共用ファイル]を選択します。[ローカルファイル]を選択した場合には、ワークスペースメタデータに起動構成の内容がローカルに保存されます。[共用ファイル]を選択した場合、指定した場所に起動構成の内容を保存し、共用できます。

[お気に入りメニューで表示]

お気に入りメニューに[構成および実行]および[構成およびデバッグ]を追加する場合は、[構成および実行]および[構成およびデバッグ]をチェックします。デフォルトでこれらの項目は選択されません。

[標準入出力]

COBOLアプリケーション起動構成では、標準入出力の設定変更は無効です。

[バックグラウンドでの起動]

COBOLアプリケーションをバックグラウンドで起動する場合には、[バックグラウンドの起動]をチェックします。デフォルトではバックグラウンドでの起動が選択されています。

第7章 アプリケーションを運用する

開発したアプリケーションを運用して業務を行うときの環境を、運用環境といいます。ここでは、運用環境について説明します。

7.1 アプリケーションの運用環境

開発したアプリケーションを、実際の運用環境で独立して実行させるためには、開発環境から運用環境に必要な資産を配布する必要があります。運用に必要な資産のことを、運用資産といいます。
ここでは、運用資産について説明します。

運用資産

アプリケーションの運用に必要な運用資産を構成するファイルの説明およびプロジェクト資産の格納フォルダに格納される運用資産を以下に示します。

表7.1 運用資産を構成するファイルの説明

運用資産を構成するファイル		意味	作成されるタイミング
ファイル名	拡張子		
プロジェクト名(デフォルト)	jar	プロジェクトのJavaアーカイブファイル	ビルド時
	class	プロジェクトのJavaクラスファイル	ビルド時
	war	Webアプリケーションのアーカイブファイル	ビルド時
	ear	エンタープライズアプリケーションのアーカイブファイル	ビルド時、またはAntビルドファイル(earbuild.xml)実行時
	exe	COBOLアプリケーションの実行ファイル	ビルド時
	dll	COBOLアプリケーションのダイナミックリンクライブラリ	ビルド時
USプロジェクト名	dll	CORBAサーバアプリケーションのダイナミックリンクライブラリ	ビルド時
電子フォーム名	fae	電子フォームデザイナで作成した電子フォームの大きさや電子フォーム内の項目のレイアウトおよびオーバレイなどを定義したファイル	電子フォーム定義時
	pmd		
	ovd		
	psf		
	bip		
Javaクラス名	class	ユーザがメインプログラム/サブプログラム/イベント処理に追加したJavaクラスのJavaクラスファイル	ビルド時
任意名	ser	Beanのシリアライズファイル	Javaフォーム定義でBeanに設定したプロパティ値を保存したとき
	map	関係定義ファイル	Webアプリケーション(Apcoordinator)新規作成時など
	tld	タグライブラリディスクリプタファイル	ユーザが作成
	css	スタイルシートファイル	ユーザが作成
	js	JavaScriptファイル	ユーザが作成。電子フォームで関数を定義した場合にも作成される。
ユーザが指定したリソースファイル名	任意	・ ダイアログなどのウィンドウに表示するイメージファイル ・ その他(任意のファイル)	ユーザが作成

運用資産を構成するファイル		意味	作成されるタイミング
ファイル名	拡張子		
COBOL85	CBR	COBOLアプリケーションの実行環境情報設定ファイル	COBOLプロジェクト新規作成時

7.1.1 運用環境へ資産を配布する

アプリケーションを運用環境で動作させるためには、開発したプロジェクト資産の中から運用時に必要な資産を、運用資産として抽出し、運用環境に配布する必要があります。
J2EEアプリケーションの場合には、資産の配布とアプリケーションのインストールを兼ねた配備を行うことで運用可能になります。

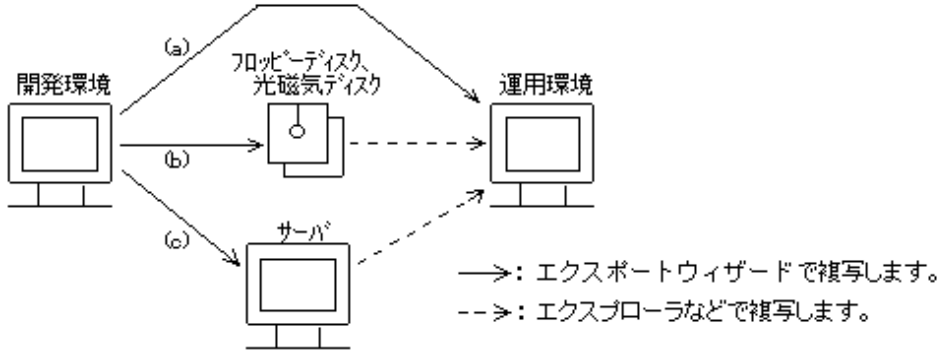
運用資産を配布する

開発したアプリケーションは、エクスポートウィザードを使用して、開発環境から運用環境に運用資産を配布できます。

運用資産を配布する方法

メニューから[ファイル] > [エクスポート]を選択してエクスポートウィザードを起動します。最初に表示される画面でエクスポート先に[ファイルシステム]を選択します(圧縮したい場合には[アーカイブファイル])。続いて表示される画面で運用時に必要な資産を抽出し、運用環境、媒体(フロッピーディスク、光磁気ディスクなど)またはサーバ上のハードディスクに、抽出した運用資産を複写します。
媒体(フロッピーディスク、光磁気ディスクなど)またはサーバ上のハードディスクに複写した場合は、エクスプローラなどで運用環境に運用資産を複写します。
資産配布の流れを下図に示します。

図7.1 資産配布の流れ



(a): ネットワークを経由して、運用環境に直接配布します。

(b): 媒体(フロッピーディスク、光磁気ディスクなど)を経由して、運用環境に配布します。

(c): サーバ上のハードディスクを経由して、運用環境に配布します。

7.1.2 運用資産を配備する

EARファイル、WARファイル、EJB JARファイルなどのJ2EEアプリケーションの運用資産は、配備を行うことで運用可能になります。
Interstage Application Serverの場合、配備にはInterstage管理コンソールを使用します。
Interstage管理コンソールでは、運用環境にログインし、配備するファイルを指定することにより、リモート環境から運用環境に配備を行うことができます。
配備やInterstage管理コンソールの詳細については、Interstage Application Serverのマニュアルを参照してください。

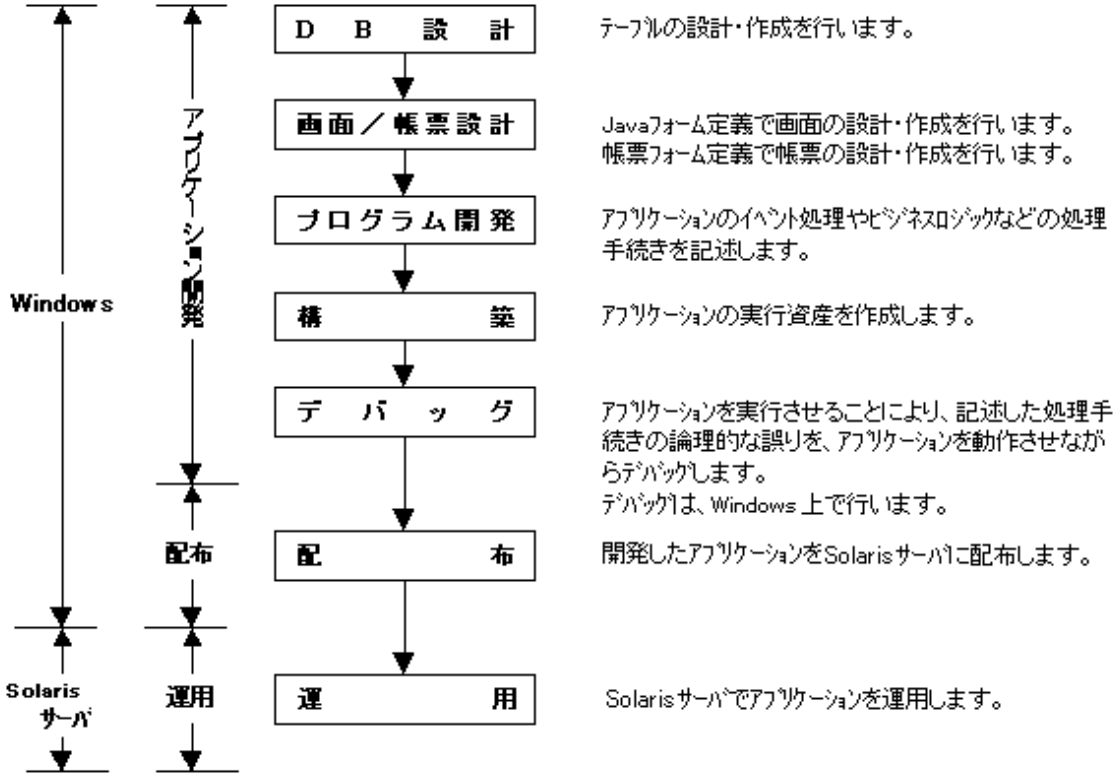
7.1.3 運用環境をセットアップする

運用資産の配布を行ったあと、運用環境をセットアップする作業が必要です。セットアップ作業は、配布先コンピュータごとに実施します。
セットアップ作業は、アプリケーション種別によって異なります。各アプリケーションのセットアップ方法については、各アプリケーションに関する開発方法、チュートリアル、Interstage Application Serverのマニュアルを参照してください。

7.2 アプリケーションの開発から運用までの流れ

ワークベンチで開発したアプリケーションをSolarisサーバで運用する手順を以下に示します。

図7.2 Solarisサーバでの運用手順



注意

COBOL関連のアプリケーションは、そのアプリケーションを運用する環境で実行可能な形式で実行ファイルを作成する必要があります。SolarisなどのWindows以外の環境で運用する場合には、リモート開発機能を利用して実行ファイルを作成します。

第3部 ワークベンチ操作編

第8章 ウィザード

プロジェクトやソースファイルを作成する場合に使用する、ウィザードについて説明します。

8.1 プロジェクトウィザード

新規プロジェクトを作成する場合に使用する、ウィザードについて説明します。

プロジェクトウィザードは、以下の手順で起動します。

1. メニューバーから[ファイル] > [新規] > [プロジェクト]を選択します。
2. [新規プロジェクト]ウィザードで、作成するプロジェクト種別を選択します。

[新規プロジェクト]ウィザード

カテゴリ	プロジェクト種別
COBOL	COBOLプロジェクト
	CORBAサーバプロジェクト

8.1.1 COBOLプロジェクトウィザード

COBOLプロジェクトウィザードを使用すると、COBOL言語記述によるCORBAクライアントなどのアプリケーションを作成できます。
COBOLプロジェクトウィザードでは、ターゲットの定義やビルド環境の設定を行います。

COBOLプロジェクトは、以下の手順で作成します。

1. メニューバーから[ファイル] > [新規] > [COBOLプロジェクト]を選択します。
2. [新規]ウィザードで、以下の情報を設定します。
   - [COBOLプロジェクト]ページで、プロジェクト基本情報を指定します。
   - [COBOLプロジェクト]ページで、ターゲットを定義します。
   - ターゲットの定義でプリコンパイラを使用するプロジェクトの生成を指示した場合、プリコンパイラ連携情報を指定します。
   - [ビルド環境]ページで、CORBAクライアント用のビルド環境を定義します。
   - [選択]ページで[コード生成を行う]を選択し、[使用可能なコード生成ウィザード]から作成するソース種別を選択します。
     - COBOLソース
     - オブジェクト指向COBOLソース

COBOLソース生成ウィザード、オブジェクト指向COBOLソース生成ウィザードの詳細については、"8.2 ソース生成ウィザード"を参照してください。

8.1.1.1 プロジェクト基本情報

プロジェクトに関する基本情報を指定します。

項目		説明
プロジェクト名		プロジェクト名を入力します。
プロジェクトコンテンツ		プロジェクト資産を保存する場所を指定します。
	ワークスペース内に新規プロジェクトを作成	プロジェクト資産をワークスペースフォルダの配下に保存します。
	外部ロケーションに新規プロジェクトを作成	プロジェクト資産をワークスペースフォルダの外に保存します。[参照]をクリックして保存先フォルダを選択できます。

8.1.1.2 ターゲットの定義

作成するCOBOLアプリケーションのターゲット種別とターゲットファイルの名前を定義します。

項目	説明
ターゲット種別	作成するCOBOLアプリケーションのターゲット種別を指定します。 実行可能ファイル(exe)を作成する場合は[実行ファイル]を選択し、ダイナミックリンクライブラリ(dll)を作成する場合は[ダイナミックリンクライブラリ]を選択します。
DLL固有の実行用の初期化ファイル(COBOL85.CBR)を使用する	ターゲット種別がダイナミックリンクライブラリの場合、DLL固有の実行用の初期化ファイルを使用するか否かを指定します。 選択すると、DLL固有の実行用の初期化ファイルを使用するダイナミックリンクライブラリになります。
ターゲット名	リンク後に作成するターゲットファイル(exe/dllファイル)のファイル名を指定します。
作成するアプリケーションの形式	実行可能ファイル(exe)を作成する場合に、アプリケーションが使用するコンソールの種別(COBOLのコンソール/システムのコンソール)を指定します。
プリコンパイラを使用する	プリコンパイラを使用するプロジェクトを生成する場合に選択します。
テキストファイルエンコード	プロジェクトにファイルを新規作成するときのテキストファイルエンコードを選択します。

[テキストファイルエンコード]は以下の手順でプロジェクト作成後に変更できます。

1. 依存ビューまたは構造ビューから[テキストファイルエンコード]を変更するプロジェクトを選択します。
2. コンテキストメニューから[プロパティ]を選択します。[プロパティ]ダイアログボックスが表示されます。
3. 左のペインで[情報]を選択すると[情報]ページが表示されます。
4. [情報]ページの[テキストファイルエンコード]から変更するエンコードを選択して、[プロパティ]ダイアログボックスの[OK]ボタンをクリックします。

注意

- [情報]ページの[テキストファイルエンコード]でエンコードを変更しても、プロジェクト内の既存ファイルのエンコードは変更されません。
- テキストファイルエンコード"UTF-8"のCOBOLソースファイルをビルドするには、NetCOBOL V10.0.0以降がインストールされている必要があります。

8.1.1.3 プリコンパイラ連携情報

生成するCOBOLプロジェクトのプリコンパイラ連携情報を指定します。

項目	説明
プリコンパイラコマンド	プリコンパイラとして起動するコマンド名を指定します。
プリコンパイラのパラメタ	プリコンパイラコマンドのパラメタを指定します。
入力ソースの拡張子	プリコンパイラ入力ソースファイルの拡張子を指定します。 以下の拡張子を指定することはできません。 ・ cobol ・ cob ・ cbl ・ lcai
出力ソースの拡張子	プリコンパイラ出力ソースファイルの拡張子を選択します。

項目	説明
COBOLコンパイラのエラーメッセージをプリコンパイラ入力ソースの行番号で表示する	選択するとプリコンパイラ入力ソースの行対応情報をCOBOLソースファイルへ展開します(INSDBINFコマンドを呼び出します)。 初期値では選択されていません。
INSDBINFコマンドのパラメタ	プリコンパイルによって生成されたCOBOLソースファイルに、プリコンパイラ入力ソースに対する行補正情報を展開するINSDBINFコマンドのパラメタを指定します。 ただし、入力ソースファイル名と出力ソースファイル名は、プリコンパイラ入力ソースファイル名から決定されるため、指定する必要はありません。

プリコンパイラ連携情報の詳細は"6.1.4.1 プリコンパイラ連携情報の初期値の設定・変更"を参照してください。

8.1.1.4 ビルド環境

CORBAクライアント用のビルド環境を定義します。

項目		説明
CORBAクライアントのビルド環境を設定		このプロジェクトをCORBAクライアントとして使用する場合に指定します。
	記述言語	CORBAクライアントの記述言語を指定します。 [オブジェクト指向COBOL]または[COBOL]のどちらかを選択します。
	オブジェクトの形式	オブジェクトのスレッド形式を指定します。 [マルチスレッド]または[シングルスレッド]のどちらかを選択します。
CDCORBAクラスを使用する		CORBAクライアント作成支援クラス(CDCORBA)を使用する場合に指定します。

8.1.1.5 選択

ソースコードを生成するかどうかを指定します。
ソースコードを生成してプロジェクトを作成する場合は、[使用可能なコード生成ウィザード]で、作成するソースコードを選択します。

項目	説明
コード生成は行わない	ソースコードを生成しないプロジェクトを作成します。
コード生成を行う	ソースコードのひな型を生成するウィザードを使用して、プロジェクトを作成します。 [使用可能なコード生成ウィザード]一覧からウィザードを選択します。
使用可能なコード生成ウィザード	[コード生成を行う]を指定した場合に、ソースコードを生成するためのウィザードを選択します。

8.1.2 CORBAサーバプロジェクトウィザード

CORBAサーバプロジェクトウィザードを使用すると、オブジェクト指向COBOL言語記述によるCORBAサーバアプリケーションを開発できます。
CORBAサーバプロジェクトウィザードでは、ターゲットの定義やビルド環境の設定を行います。

CORBAサーバプロジェクトは、以下の手順で作成します。

1. メニューバーから[ファイル] > [新規] > [CORBAサーバプロジェクト]を選択します。
2. [新規]ウィザードで、以下の情報を設定します。
   - [CORBAサーバプロジェクト]ページで、プロジェクト基本情報を指定します。
   - [CORBAサーバプロジェクト]ページで、ターゲットを定義します。
   - ターゲットの定義でプリコンパイラを使用するプロジェクトを指示した場合、プリコンパイラ連携情報を指定します。
   - [ビルド環境]ページで、CORBAサーバ用のビルド環境を定義します。

ー [選択]ページで[コード生成を行う]を選択し、[使用可能なコード生成ウィザード]から作成するソース種別を選択します。

- CORBAサーバアプリケーション

CORBAサーバアプリケーション生成ウィザードの詳細については、"8.2 ソース生成ウィザード"を参照してください。

8.1.2.1 ターゲットの定義

作成するCORBAサーバアプリケーションのターゲットファイルの名前を定義します。

項目	説明
ターゲット種別	作成するCOBOLアプリケーションのターゲット種別を指定します。 CORBAサーバプロジェクトでは、[CORBAサーバ]が選択されています。
ターゲット名	リンク後に作成するターゲットファイル（exe/dllファイル）のファイル名を指定します。
作成するアプリケーションの形式	アプリケーションが使用するコンソールの種別（COBOLのコンソール/システムのコンソール）を指定します。
プリコンパイラを使用する	プリコンパイラを使用するプロジェクトを生成する場合に選択します。

8.1.2.2 プリコンパイラ連携情報

生成するCORBAサーバプロジェクトのプリコンパイラ連携情報を指定します。

項目	説明
プリコンパイラコマンド	プリコンパイラとして起動するコマンド名を指定します。
プリコンパイラのパラメタ	プリコンパイラコマンドのパラメタを指定します。
入力ソースの拡張子	プリコンパイラ入力ソースファイルの拡張子を指定します。 以下の拡張子を指定することはできません。 ・ cobol ・ cob ・ cbl ・ lcai
出力ソースの拡張子	プリコンパイラ出力ソースファイルの拡張子を選択します。
COBOLコンパイラのエラーメッセージをプリコンパイラ入力ソースの行番号で表示する	選択するとプリコンパイラ入力ソースの行対応情報をCOBOLソースファイルへ展開します(INSDBINFコマンドを呼び出します)。 初期値では選択されていません。
INSDBINFコマンドのパラメタ	プリコンパイルによって生成されたCOBOLソースファイルに、プリコンパイラ入力ソースに対する行補正情報を展開するINSDBINFコマンドのパラメタを指定します。 ただし、入力ソースファイル名と出力ソースファイル名は、プリコンパイラ入力ソースファイル名から決定されるため、指定する必要はありません。

プリコンパイラ連携情報の詳細は"6.1.4.1 プリコンパイラ連携情報の初期値の設定・変更"を参照してください。

8.1.2.3 ビルド環境

CORBAサーバ用のビルド環境を定義します。

項目	説明
Unicodeを使用する	Unicodeを使用する場合に指定します。

8.2 ソース生成ウィザード

ソースファイルを作成する場合に使用する、ウィザードについて説明します。

ソース生成ウィザードは、以下の手順で起動します。

1. メニューバーから[ファイル] > [新規] > [その他]を選択します。
2. [新規]ウィザードで、作成するソースファイルを選択します。

[新規]ウィザード

カテゴリ		リソース種別
COBOL	ソース	COBOLソース
		CORBAサーバアプリケーション
		CORBAスタブファイル
		IDLファイル
		オブジェクト指向COBOLソース
		COBOL登録集

8.2.1 COBOLソース

COBOLソースファイルを作成できます。

COBOLソース生成ウィザードは、以下の手順で起動します。

1. メニューバーから[ファイル] > [新規] > [その他]を選択します。
2. [新規]ウィザードで[COBOL] > [ソース] > [COBOLソース]を選択します。

[COBOLソース生成ウィザード]では、[COBOLソースファイルの作成]ページでCOBOLソースの基本情報を指定します。

8.2.1.1 COBOLソースファイルの作成

COBOLソースファイルを生成するための基本情報を指定します。

項目	説明
プロジェクト名	ソースの生成先となるプロジェクトのプロジェクト名を入力します。
ファイル名	生成されるCOBOLソースのファイル名を入力します。
PROGRAM-ID	PROGRAM-IDを入力します。 デフォルトは[ファイル名]と同じです。PROGRAM-IDを変更する場合は[PROGRAM-ID]にそのPROGRAM-IDを入力します。
ファイルコメント	生成されるCOBOLソースの先頭に付加されるコメントを入力します。ファイルコメントは省略可能です。
プリコンパイラを使用する	プリコンパイラ入力ソースを生成する場合に選択します。 ワークスペースのプリコンパイラ連携情報が設定されている場合に有効となります。ワークスペースのプリコンパイラ連携情報設定については、"6.1.4.1 プリコンパイラ連携情報の初期値の設定・変更"を参照してください。

8.2.2 CORBAサーバアプリケーション

CORBAサーバアプリケーション生成ウィザードは、IDLファイル（インタフェース定義ファイル）とCORBAサーバアプリケーションのひな型を生成します。

CORBAサーバアプリケーション生成ウィザードは、以下の手順で起動します。

1. メニューバーから[ファイル] > [新規] > [その他]を選択します。

2. [新規]ウィザードで[COBOL] > [ソース] > [CORBAサーバアプリケーション]を選択します。

[CORBAサーバアプリケーション生成ウィザード]では、以下の情報を指定します。

1. [モジュール宣言]ページで、モジュール名や例外宣言などの情報を指定します。
2. [定数宣言]ページで、インタフェース宣言に必要な定数の情報を指定します。
3. [型宣言]ページで、インタフェース宣言に必要な型の情報を指定します。
4. [構造体宣言]ページで、インタフェース宣言で使用する構造体の情報を指定します。
5. [メソッド宣言]ページで、インタフェース宣言で使用するメソッドの情報を指定します。

8.2.2.1 モジュール宣言

作成するCORBAサーバアプリケーションのモジュール名、例外宣言、デフォルト処理の生成、コメントの生成の情報を指定します。

項目	説明
プロジェクト名	作成するCORBAサーバアプリケーションソースを格納するプロジェクトを指定します。プロジェクト名は必須入力です。
モジュール名	作成するCORBAサーバアプリケーションのモジュール名を指定します。モジュール名は、必須入力です。
クラス名	作成するCORBAサーバアプリケーションのクラス名が表示されます。 クラス名はプロジェクト名と同じになります。また、ウィザード終了後のIDLファイルでは、インタフェース名として生成されます。

例外の生成	作成するCORBAサーバアプリケーションで、デフォルトの例外宣言をするかどうかを指定します。
	チェックした場合は、以下の宣言がIDLファイル中に生成されます。
	`exception CDException {`
	`  string CDExceptionMsg ;`
	`  long  CDExceptionCode ;`
	`} ;`

デフォルト処理の生成	作成するCORBAサーバアプリケーションで、定型処理を生成するかどうかを指定します。 チェックした場合は、プログラムソース中にCORBAサーバアプリケーションの定型処理が生成されます。
コメントの生成	作成するCORBAサーバアプリケーションで、コメントを生成するかどうかを指定します。 チェックした場合は、IDLファイル中にコメントが生成されます。
プリコンパイラを使用する	作成するCOBOLソースファイルの拡張子を、プリコンパイラ連携情報で設定した入力ソースファイルの拡張子にするかどうかを指定します。プリコンパイラ連携情報設定については、"6.1.4.1 プリコンパイラ連携情報の初期値の設定・変更"を参照してください。 ターゲットの定義で[プリコンパイラを使用する]チェックボックスをチェックした場合にだけ有効です。

注意

ソース生成中に実行されるIDLの翻訳にはあらかじめInterstageのサービスが起動されている必要があります。ソース生成時にウィザードの[モジュール宣言]ページに「IDLを翻訳する環境が整っていません」というメッセージが表示された場合には、一度[キャンセル]をクリックしてウィザードを終了し、Interstageのサービスを起動してから再度、ソース生成ウィザードを起動してください。

8.2.2.2 定数宣言

CORBAサーバアプリケーションのインタフェース宣言に必要な定数を宣言します。

項目	説明
型	宣言する定数の型を選択します。 定義可能な型およびCOBOLとIDLのマッピングは、定数の型を参照してください。

項目	説明
定数名	定義する定数名を指定します。
初期値	定数の初期値を指定します。 初期値は、IDLファイル中に生成されます。定数の初期値は、IDLの形式で記述してください。
追加	定数宣言を新規に追加します。
削除	選択した定数宣言を削除します。

定数の型

定義可能な型	COBOL	IDL
2バイト整数	PIC S9(4) COMP-5	short
2バイト整数(符号無)	PIC 9(4) COMP-5	unsigned short
4バイト整数	PIC S9(9) COMP-5	long
4バイト整数(符号無)	PIC 9(9) COMP-5	unsigned long
単精度浮動小数点	COMP-1	float
倍精度浮動小数点	COMP-2	double
英数文字	PIC X(1)	char
ブール	PIC 1(1)	boolean
英数文字列	PIC X(n)	string

定数の初期値

文字定数の場合	文字をシングルクォーテーションで囲む(例:'A')
文字列定数の場合	文字列をダブルクォーテーションで囲む(例:"ABC")
4バイト整数の場合	数値(例:1)
論理値の場合	TRUEまたはFALSE

8.2.2.3 型宣言

CORBAサーバアプリケーションのインタフェース宣言に必要な型(繰返し項目)を宣言します。

項目	説明
型	宣言する変数名の型を選択します。 定義可能な型およびCOBOLとIDLのマッピングは、変数名の型を参照してください。
変数名	定義する変数名を指定します。
全体桁	型が英数文字列、日本語文字列または内部10進数の場合に、全体桁を指定します。
小数桁	型が内部10進数の場合に、小数桁を指定します。
繰返し数	1次元要素の繰返し数を指定します。
追加	型宣言を新規に追加します。
削除	選択した型宣言を削除します。

変数名の型

定義可能な型	COBOL	IDL
2バイト整数	PIC S9(4) COMP-5	short
2バイト整数(符号無)	PIC 9(4) COMP-5	unsigned short

定義可能な型	COBOL	IDL
4バイト整数	PIC S9(9) COMP-5	long
4バイト整数(符号無)	PIC 9(9) COMP-5	unsigned long
8バイト整数	PIC S9(18) COMP-5	long long
内部10進数	PIC xx(n) PACKED-DECIMAL 注1)	fixed
単精度浮動小数点	COMP-1	float
倍精度浮動小数点	COMP-2	double
英数文字	PIC X(1)	char
日本語文字	PIC N(1)	wchar
ブール	PIC 1(1)	boolean
英数文字列	PIC X(n)	string
日本語文字列	PIC N(n)	wstring

注1) 内部10進数のCOBOL型は全体桁数と小数桁数の組合せによって変わります。詳細は、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"の"アプリケーションの開発(OOCOBOL)"を参照してください。

8.2.2.4 構造体宣言

CORBAサーバアプリケーションのインタフェース宣言で使用する構造体を宣言します。

項目	説明
構造体の定義	定義済みの構造体の一覧が表示されます。 この一覧で選択中のメソッドが、[編集]、[削除]の対象になります。
追加	構造体を新規に追加します。 [追加]をクリックすると、"構造体の定義"画面が表示されます。詳細は"8.2.2.4.1 構造体の定義"を参照してください。
編集	選択した構造体の定義情報を変更します。 [編集]をクリックすると、"構造体の定義"画面が表示されます。詳細は"8.2.2.4.1 構造体の定義"を参照してください。
削除	選択した構造体を削除します。

8.2.2.4.1 構造体の定義

CORBAサーバアプリケーションで使用する構造体を定義します。

項目	説明
構造体名	定義する構造体名を指定します。
変数名	定義する変数名を指定します。
型	宣言する変数名の型を選択します。 定義可能な型およびCOBOLとIDLのマッピングは、変数名の型を参照してください。
全体桁	型が英数文字列、日本語文字列または内部10進数の場合に、全体桁を指定します。
小数桁	型が内部10進数の場合に、小数桁を指定します。
繰返し数	1次元要素の繰返し数を指定します。
追加	構造体に要素を新規に追加します。
削除	選択した構造体の要素を削除します。

変数名の型

定義可能な型	COBOL	IDL
2バイト整数	PIC S9(4) COMP-5	short
2バイト整数(符号無)	PIC 9(4) COMP-5	unsigned short
4バイト整数	PIC S9(9) COMP-5	long
4バイト整数(符号無)	PIC 9(9) COMP-5	unsigned long
8バイト整数	PIC S9(18) COMP-5	long long
内部10進数	PIC xx(n) PACKED-DECIMAL 注1)	fixed
単精度浮動小数点	COMP-1	float
倍精度浮動小数点	COMP-2	double
英数文字	PIC X(1)	char
日本語文字	PIC N(1)	wchar
ブール	PIC 1(1)	boolean
英数文字列	PIC X(n)	string
日本語文字列	PIC N(n)	wstring
任意の型 注2)	任意の型	任意の型

注1) 内部10進数のCOBOL型は全体桁数と小数桁数の組合せによって変わります。詳細は、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"の"アプリケーションの開発(OOCOBOL)"を参照してください。

注2) 型定義で指定した型(繰返し項目の型)を入力します。任意の型はすべて型定義で定義された型でなければなりません。

8.2.2.5 メソッド宣言

CORBAサーバアプリケーションのインタフェース宣言で使用する利用者メソッド(ビジネスメソッド)を宣言します。

項目	説明
利用者メソッドの定義	定義済みの利用者メソッドの一覧が表示されます。 この一覧で選択中のメソッドが、[編集]、[削除]の対象になります。
詳細	[利用者メソッドの定義]で選択した利用者メソッドの詳細情報が表示されます。
追加	利用者メソッド(ビジネスメソッド)を新規に追加します。 [追加]をクリックすると、"利用者メソッドの定義"画面が表示されます。詳細は"8.2.2.5.1 利用者メソッドの定義"を参照してください。
編集	[利用者メソッドの定義]で選択しているメソッドの定義情報を変更します。 [編集]をクリックすると、"利用者メソッドの定義"画面が表示されます。詳細は"8.2.2.5.1 利用者メソッドの定義"を参照してください。
削除	[利用者メソッドの定義]で選択している利用者メソッド(ビジネスメソッド)を削除します。

8.2.2.5.1 利用者メソッドの定義

CORBAサーバアプリケーションで使用する利用者メソッドを定義します。

項目	説明
メソッド名	定義する利用者メソッド名を指定します。
戻り値の型	メソッドの戻り値の型を選択します。 定義可能な型およびCOBOLとIDLのマッピングは、戻り値の型を参照してください。
全体桁数	戻り値の全体桁数を指定します。 戻り値の型が英数文字列、日本語文字列および内部10進数の場合に全体桁数を指定します。

項目	説明
小数部桁数	戻り値の型の小数部桁数を指定します。 戻り値の型が内部10進数の場合に小数部桁数を指定します。
例外を発生させる	作成するCORBAサーバアプリケーションで、乗算、除算の例外を通知する処理を生成するかどうかを指定します。 チェックした場合は、プログラムソース中に乗算、除算の例外を通知する処理が生成されます。
パラメタリスト	パラメタの追加および削除、各パラメタの変数および型の編集ができます。パラメタの型は、型の一覧から選択することも、直接、値を入力することもできます。また、変数名には、全角文字と半角文字を組み合わせた文字列は指定できません。型には、全角文字は指定できません。
変数名	定義する変数名を指定します。
型	宣言する変数名の型を選択します。 定義可能な型およびCOBOLとIDLのマッピングは、変数名の型を参照してください。
全体桁	型が英数文字列、日本語文字列および内部10進数の場合に、全体桁を指定します。
小数桁	型が内部10進数の場合に、小数桁を指定します。
パラメタタイプ	パラメタのタイプを選択します。 inの場合は、入力用のパラメタとしてプログラムソース中に生成されます。 outの場合は、出力用のパラメタとしてプログラムソース中に生成されます。 inoutの場合は、入出力用のパラメタとしてプログラムソース中に生成されます。
追加	利用者メソッドにパラメタを新規に追加します。
削除	選択した利用者メソッドのパラメタを削除します。

戻り値の型

定義可能な型	COBOL	IDL
なし	-	oneway void
2バイト整数	PIC S9(4) COMP-5	short
2バイト整数(符号無)	PIC 9(4) COMP-5	unsigned short
4バイト整数	PIC S9(9) COMP-5	long
4バイト整数(符号無)	PIC 9(9) COMP-5	unsigned long
8バイト整数	PIC S9(18) COMP-5	long long
内部10進数	PIC xx(n) PACKED-DECIMAL 注1)	fixed
単精度浮動小数点	COMP-1	float
倍精度浮動小数点	COMP-2	double
英数文字	PIC X(1)	char
日本語文字	PIC N(1)	wchar
ブール	PIC 1(1)	boolean
英数文字列	PIC X(n)	string
日本語文字列	PIC N(n)	wstring
任意の型 注2)	入力された型	入力された型

注1) 内部10進数のCOBOL型は全体桁数と小数桁数の組合せによって変わります。詳細は、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"の"アプリケーションの開発(OOCOBOL)"を参照してください。

注2) 型定義または構造体定義で指定した型を入力します。

変数名の型

定義可能な型	COBOL	IDL
2バイト整数	PIC S9(4) COMP-5	short
2バイト整数(符号無)	PIC 9(4) COMP-5	unsigned short
4バイト整数	PIC S9(9) COMP-5	long
4バイト整数(符号無)	PIC 9(9) COMP-5	unsigned long
8バイト整数	PIC S9(18) COMP-5	long long
内部10進数	PIC xx(n) PACKED-DECIMAL 注1)	fixed
単精度浮動小数点	COMP-1	float
倍精度浮動小数点	COMP-2	double
英数文字	PIC X(1)	char
日本語文字	PIC N(1)	wchar
ブール	PIC 1(1)	boolean
英数文字列	PIC X(n)	string
日本語文字列	PIC N(n)	wstring
任意の型 注2)	入力された型	入力された型

注1) 内部10進数のCOBOL型は全体桁数と小数桁数の組合せによって変わります。詳細は、"Interstage Application Server アプリケーション作成ガイド（CORBAサービス編）"の"アプリケーションの開発(OOCOBOL)"を参照してください。

注2) 型定義または構造体定義で指定した型を入力します。

8.2.3 CORBAスタブファイル

CORBAクライアントアプリケーションを作成する場合には、CORBAサーバアプリケーションで作成したIDLファイルを利用してCORBAクライアントに必要なスタブファイルおよびその他のファイルを生成し、プロジェクトに登録します。

CORBAスタブファイル生成ウィザードは、以下の手順で起動します。

1. メニューバーから[ファイル] > [新規] > [その他]を選択します。
2. [新規]ウィザードで[COBOL] > [ソース] > [CORBAスタブファイル]を選択します。

[CORBAスタブファイル生成ウィザード]では、[CORBAスタブファイル生成]ページでCORBAスタブファイルの基本情報を指定します。

8.2.3.1 CORBAスタブファイル生成

CORBAスタブファイルを生成するための基本情報を指定します。

項目	説明
IDLファイル	CORBAサーバアプリケーションで作成したIDLファイルのパスを入力します。絶対パスで指定してください。
登録先プロジェクト	生成したスタブファイルを登録するプロジェクトを入力します。
生成言語	生成言語を指定します。 [COBOL]または[オブジェクト指向COBOL]のどちらかを選択します。

8.2.4 IDLファイル

CORBA/IDLのオブジェクト指向COBOLのマッピングに従い、構造を記述したクライアントインタフェース(IDL)ファイルを作成できます。

IDLファイル生成ウィザードは、以下の手順で起動します。

1. メニューバーから[ファイル] > [新規] > [その他]を選択します。
2. [新規]ウィザードで[COBOL] > [ソース] > [IDLファイル]を選択します。

[IDLファイル生成ウィザード]では、[IDLファイル]ページでIDLファイルの基本情報を指定します。

8.2.4.1 IDLファイルの基本情報

IDLファイルを生成するための基本情報を指定します。

項目	説明
プロジェクト名	IDLファイルを作成するプロジェクトのプロジェクト名を指定します。
ファイル名	ファイル名を入力します。

8.2.5 オブジェクト指向COBOLソース

オブジェクト指向COBOLソースファイルを作成できます。

オブジェクト指向COBOLソースの生成ウィザードは、以下の手順で起動します。

1. メニューバーから[ファイル] > [新規] > [その他]を選択します。
2. [新規]ウィザードで[COBOL] > [ソース] > [オブジェクト指向COBOLソース]を選択します。

[オブジェクト指向COBOLソースプロジェクトの作成]ウィザードでは、[オブジェクト指向COBOLソースファイルの作成]ページでオブジェクト指向COBOLソースの基本情報を指定します。

8.2.5.1 オブジェクト指向COBOLソースファイルの作成

オブジェクト指向COBOLソースファイルを生成するための基本情報を指定します。

項目	説明
プロジェクト名	ソースの生成先となるプロジェクトのプロジェクト名を入力します。
ファイル名	生成されるCOBOLソースのファイル名を入力します。
CLASS-ID	CLASS-IDを入力します。 デフォルトは[ファイル名]と同じです。CLASS-IDを変更する場合は[CLASS-ID]にそのCLASS-IDを入力します。
親クラス	生成されるクラスの親クラス名を指定します。 親クラス名は省略可能です。この場合は、親クラスにFJBASEを指定したことになります。
ファイルコメント	生成されるCOBOLソースの先頭に付加されるコメントを入力します。ファイルコメントは省略可能です。
プリコンパイラを使用する	プリコンパイラ入力ソースを生成する場合に選択します。 ワークスペースのプリコンパイラ連携情報が設定されている場合に有効となります。ワークスペースのプリコンパイラ連携情報設定については、"6.1.4.1 プリコンパイラ連携情報の初期値の設定・変更"を参照してください。

8.2.6 COBOL登録集

COBOL登録集ファイルを作成できます。

COBOL登録集生成ウィザードは、以下の手順で起動します。

1. メニューバーから[ファイル] > [新規] > [その他]を選択します。
2. [新規]ウィザードで[COBOL] > [ソース] > [COBOL登録集]を選択します。

8.2.6.1 COBOL登録集ファイルの作成

COBOL登録集ファイルを生成するための基本情報を指定します。

項目	説明
COBOL登録集ファイル名	生成されるCOBOL登録集のファイル名を入力します。
プロジェクト	COBOL登録集を既存のCOBOLプロジェクトに生成する場合に選択します。
プロジェクト名	COBOL登録集の生成先となるプロジェクトのプロジェクト名を入力します。
外部フォルダ	COBOL登録集を外部フォルダに生成する場合に選択します。
フォルダ	COBOL登録集の生成先となるフォルダ名を入力します。

第9章 パースペクティブ

COBOLパースペクティブは、CORBAサーバアプリケーションなどのCOBOLアプリケーションを開発するためのパースペクティブです。CORBAサーバアプリケーションプロジェクトやCOBOLプロジェクトの新規作成後に、自動的にCOBOLパースペクティブに切り替わります。

このパースペクティブは、エディタ領域と以下のビューで構成されています。

- 依存
- 構造
- ナビゲータ
- プロパティ
- アウトライン
- タスク
- コンソール
- 問題

また、メニューバーの[ファイル] > [新規] 配下に、COBOLアプリケーション開発時に使用頻度が高い以下のウィザードのエントリが追加されています。そのため、新規ダイアログボックスを表示することなくウィザードを起動できます。

- COBOLプロジェクト
- CORBAサーバプロジェクト
- COBOLソース
- オブジェクト指向COBOLソース
- COBOL登録集

第10章　エディタ

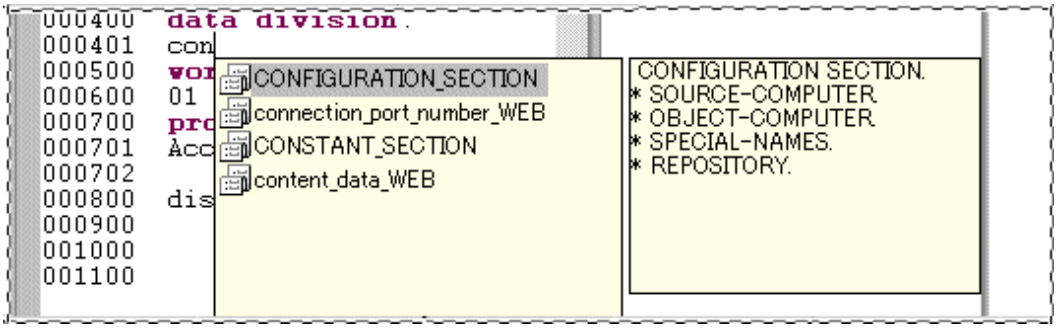
ここでは、エディタについて説明します。

10.1 COBOLエディタ

COBOLエディタは、COBOLソースの編集に使用します。
エディタのタブにファイル名が表示されます。エディタでの変更を保存していないとき、ファイル名のうしろにアスタリスク(*)が表示されます。

10.1.1 入力支援候補一覧(コンテンツアシスト)

入力支援候補一覧にはCLASS-ID、METHOD-ID(ファクトリMETHOD-IDまたはオブジェクトMETHOD-ID)、PROGRAM-ID、SUB-PROGRAM-IDおよび構文テンプレートのキーワードが表示されます。



PROGRAM-IDおよびMETHOD-IDの表示内容は、下記の条件に基づきます。

- ファイル内で[Ctrl+Space]キーを押すと、候補一覧にCLASS-IDおよびテンプレートキーワードが表示されます。

  例：Move <Ctrl+Space>

  最後に入力した文字の後にスペースを置かないで[Ctrl+Space]キーを押すと、一覧には最後に入力した文字で始まるCLASS-IDおよびテンプレートキーワードが表示されます。(以下はDaで始まる一覧を表示する場合の例です。)

  例：Move Da<Ctrl+Space>.

- キーワード[INVOKE]の後に1スペース置いて[Ctrl+Space]キーを押すと、候補一覧にCLASS-ID、METHOD-IDおよびテンプレートキーワードが表示されます。

  例：INVOKE <Ctrl+Space>

  最後に入力した文字の後にスペースを置かないで[Ctrl+Space]キーを押すと、一覧には最後に入力した文字で始まるCLASS-ID、METHOD-IDおよびテンプレートキーワードが表示されます。(以下はDaで始まる一覧を表示する場合の例です。)

  例：INVOKE Da<Ctrl+Space>

- キーワード[CALL]の後で[Ctrl+Space]キーを押すと、候補一覧にPROGRAM-ID、SUB-PROGRAM-IDおよびテンプレートキーワードが表示されます。

  例：CALL <Ctrl+Space>

  最後に入力した文字の後にスペースを置かないで[Ctrl+Space]キーを押すと、一覧には最後に入力された文字で始まるPROGRAM-IDおよびテンプレートキーワードが表示されます。(以下はDaで始まる一覧を表示する場合の例です。)

  例：CALL Da<Ctrl+Space>

テンプレートキーワードを選択した場合、テンプレートのパターンを表示する補助ウィンドウが表示されます。

10.1.2 強調色の設定

色およびフォントスタイル(太字)を使用して、キーワードを強調表示します。
以下の項目について、強調色が設定できます。

項目	説明
行コメント	標識領域に"*"または"/"がある場合、その行全体が行コメントとみなされます。また、標識領域に"D"または"d"がある行は、WITH DEBUGGING MODE句の有無にかかわらず、表示上は行コメントとしてみなされます。
行内コメント	文中に"*>"がある場合、その位置からその行末までの部分が行内コメントとみなされます。 注)固定形式にも適用されます。
予約語	予約語の一覧はCOBOLのバージョン単位に定義されます。予約語については、"NetCOBOL 文法書"を参照してください。 大文字、小文字の区別はありません。
表意定数	表意定数の一覧は次のとおりです。 SPACE、SPACES、ZERO、ZEROS、ZEROES、HIGH-VALUE、HIGH-VALUES、LOW-VALUE、LOW-VALUES、QUOTE、QUOTES、ALL 大文字、小文字の区別はありません。
特殊レジスタ	特殊レジスタ定数の一覧は次のとおりです。 LINAGE-COUNTER、PROGRAM-STATUS、RETURN-CODE、SORT-STATUS、EDIT-MODE、EDIT-OPTION、EDIT-OPTION2、EDIT-OPTION3、EDIT-COLOR、EDIT-STATUS、EDIT-CURSOR、LINE-COUNTER、PAGE-COUNTER、SORT-CORE-SIZE 大文字、小文字の区別はありません。
文字列	文字列は以下の形式で定義されます。 " "、B" "、X" "、N" "、NC" "、NX" " 大文字、小文字の区別はありません。 一重引用符または二重引用符のどちらを使用してもかまいません。

強調色の設定を表示する

上記の項目について、色設定を表示できます。

1. メニューバーから[ウィンドウ] > [設定]を選択すると[設定]ダイアログボックスが表示されます。
2. 左ペインから[COBOL] > [エディタ]を選択すると[エディタ]ページが表示されます。
3. [色]タブをクリックすると[項目]、[色]、[太字]および[プレビュー]領域が表示されます。
4. 項目を選択すると設定されている[色]、[太字]にスタイルが表示されます。[プレビュー]には現在の設定内容が表示されます。

注意

デフォルトで予約語は太字、他のすべての項目は通常スタイルで表示されます。

強調色の設定を変更する

上記の項目について、色設定を変更できます。

1. [項目]から表示色を変更したい項目を選択します。
2. [色]をクリックすると[色の設定]ダイアログボックスが表示されます。
3. [基本色]パレットから適用したい色をクリックします。[色の作成]をクリックして色を作成することもできます。

4. 選択した色を適用してダイアログボックスを閉じる場合、[OK]をクリックします。選択した色が[プレビュー]領域に反映されます。選択した色を適用しないで閉じる場合は、[キャンセル]をクリックします。
5. テキストを太字で表示する場合は[太字]をチェックします。

注意

設定内容は、次回以降にエディタを起動した場合でも有効です。

10.1.3 フォントの設定

COBOLエディタのフォントは[設定]ダイアログボックスで変更できます。デフォルトでは、ワークベンチのフォントで設定されている内容になります。

フォントの設定を変更する

1. メニューバーから[ウィンドウ] > [設定]を選択すると[設定]ダイアログボックスが表示されます。
2. 左ペインから[COBOL] > [エディタ]を選択すると[エディタ]ページが表示されます。
3. [全般]タブをクリックします。
4. [変更]をクリックすると[フォント]ダイアログボックスが表示されます。
5. 必要に応じて[フォント]、[スタイル]、[サイズ]などを変更します。

10.1.4 正書法サポート

COBOLエディタでCOBOLソースプログラムを作成、または変更する際、ファイルの形式は正書法で指定された規則に従います。

COBOLエディタでサポートする正書法には以下の2つの形式があります。

- 固定形式
- 可変形式

正書法の形式を設定する

1. メニューバーから[ウィンドウ] > [設定]を選択すると[設定]ダイアログボックスが表示されます。
2. 左ペインから[COBOL] > [エディタ]を選択すると[エディタ]ページが表示されます。
3. [正書法]タブを選択します。[正書法]ページが表示されます。
4. [正書法の設定]で[固定形式]または[可変形式]を選択します。

固定形式でのエディタの動作

- エディタの1～6列目は行番号に使用されます。これを一連番号と言います。
- 7列目は標識領域として使用されます。
- 標識領域に"/"、"*"、"D"、"d"のいずれかの文字が存在するとき、その行がコメント行であることを表します。標識領域に"D"または"d"を書いた行を、「デバッグ行」といいます。デバッグ行は、デバッグのための情報を原始プログラムに残すために使います。
- 8～11列目は"A領域"、12～72列目は"B領域"として使用されます。
- 73列目以降は"プログラム識別領域"として使用されます。

可変形式でのエディタの動作

- エディタの1～6列目は行番号に使用されます。これを一連番号と言います。
- 7列目は標識領域として使用されます。

- 標識領域に"/"、"*"、"D"、"d"のいずれかの文字が存在するとき、その行がコメント行であることを表します。標識領域に"D"または"d"を書いた行を、「デバッグ行」といいます。デバッグ行は、デバッグのための情報を原始プログラムに残すために使います。
- 8～11列目は"A領域"、12～250列目は"B領域"として使用されます。
- 251列目以降は入力したテキストはコメントとして扱われます。

注意

- 詳細については、"10.1.7 異なる形式間の編集"を参照してください。
- 正書法は変更が可能です。ファイルの変更内容を保存していない場合、正書法を変更する前にファイルの保存を指示するメッセージが表示されます。
  ファイルの変更を保存してから正書法を変更する場合は[はい]を、ファイルの変更を保存しないで正書法を変更する場合は[いいえ]を選択します。

10.1.5 コメントのスタイル

COBOLエディタは、以下のコメントスタイルをサポートします。

スタイル	説明
スタイルA	行中の標識領域（7列目）に"*"、"/"、"D"、"d"のいずれかの文字が存在する場合、その行全体がコメント行として扱われる。
スタイルB	文中に"*>"が存在する場合、その位置から行末にかけて行内コメントとして扱われる。
スタイルC	プログラム識別領域（73列目以降）に記述された文字もコメントとして扱われる。

注意

- 固定形式は上記のすべてのコメントスタイルをサポートします。
- 可変形式はスタイルAとスタイルBをサポートします。（可変形式では、251列目以降がコメントとして扱われます。）

10.1.6 一連番号の初期値および増加値の変更

一連番号の初期値と増加値を変更できます。

一連番号の初期値、増加値を変更する

1. メニューバーから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
2. 左ペインから[COBOL] > [エディタ]を選択すると[エディタ]ページが表示されます。
3. [正書法]タブを選択します。[正書法]ページが表示されます。
4. [一連番号の初期値および増加値の設定]で[初期値]と[増加値]を指定します。

注意

デフォルトで[初期値]は10、[増加値]は10に設定されます。

10.1.7 異なる形式間の編集

ワークベンチは異なる形式間のファイル編集をサポートします。下表は、異なる形式のファイルが開かれた場合のエディタの動作を説明しています。

COBOLエディタに設定されている正書法	ファイルの正書法	説明
固定形式	可変形式	ファイルは固定形式で開かれる
固定形式	固定形式	ファイルは固定形式で開かれる
可変形式	可変形式	ファイルは可変形式で開かれる
可変形式	固定形式	ファイルは可変形式で開かれる

[設定]ダイアログボックスの[COBOL] > [エディタ]ページで[正書法の設定]を変更する場合、その形式変更はエディタで現在開いているファイルには適用されません。変更後、エディタで開かれるファイルにだけ適用されます。

10.1.8 一連番号サポート

エディタは自動番号付加機能をサポートします。ワークベンチの番号付加は、以下の2つのパターンをサポートします。

1. パターンA
   ファイル中のすべての行が6桁の数字(0 ～ 9)の一連番号を持ち、それらが昇順に並んでいる場合
2. パターンB
   パターンA以外のすべてのパターン

一連番号領域は編集不可能です。下の表に、手動で番号付加する際のエディタの動作を示します。

ファイルの最後に新しい行を追加する	パターンBの番号付加の場合、新しく追加された行の一連番号領域用に6バイトのスペースが割当てられる。パターンAの番号付加の場合、新しく追加された行番号は最後の一連番号に増加値を加えた値となる。その値が999999を超える場合、増加値は1に設定される。一連番号999999の次に行が追加される場合、パターンBに変更される。
新しい行を挿入する	パターンBの番号付加の場合、新しく追加された行の一連番号領域に6バイトのスペースが割当てられる。パターンAの番号付加の場合、新しく追加された行の一連番号は前行の一連番号に増加値を加えた値となる。この値が次の一連番号以上となる場合、増加値は1に設定される。ただし、増加値1を加えた値が次の一連番号以上になる場合、後続の行に対して番号の振り直しが行われる。
行を削除する	行が削除される場合、一連番号の振り直しは行われません。
行を貼り付ける	新しい行を挿入する処理と同様です。

注意

修正履歴の比較では一連番号は対象になりません。このため、一連番号を振り直しただけでは修正されたことになりません。ただし、修正履歴から置換する場合は一連番号も置換の対象になります。

10.1.9 一連番号を振り直す

ファイルで初期値および増加値を変更して一連番号を振り直すことができます。増加値は最大999999です。

一連番号を振り直す

1. メニューバーから[編集] > [リナンバ]を選択するか、[Ctrl+R]キーを押します。[リナンバ]ダイアログボックスが表示されます。
2. [初期値]と[増加値]を指定します。
3. [リナンバ]をクリックして一連番号を振り直し、ダイアログボックスを閉じます。[キャンセル]をクリックすると、番号を振り直さないでダイアログボックスを閉じます。

注意

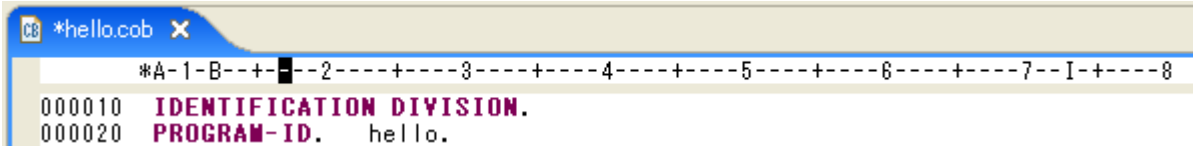
- デフォルトで[初期値]は10、[増加値]は10に設定されます。

- 設定された[初期値]および[増加値]については、ファイル内の行数が999999以上になる場合、[リナンバ]ダイアログボックスにエラーメッセージが表示されます。

10.1.10 水平ルーラの表示

水平ルーラはエディタのウィンドウ上部に表示されます。水平ルーラの特徴を以下に示します。

- 列番号は10列ごとに表示されており、ルーラ上では10列目を1、15列目を+、20列目を2、(以下同様)という規則で表示される
- 固定形式ファイルの場合
  - ルーラは7列目から開始し、"*"記号で示される



  - ルーラ上8列目のAは、A領域の始まりの列であることを示す(8列目から11列目)
  - ルーラ上12列目のBは、B領域の始まりの列であることを示す(12列目から72列目)
  - ルーラ上73列目のIは、プログラム識別領域の始まりの列であることを示す(73列目以降)
  - ルーラは80列目まで表示される
- 可変形式ファイルの場合
  - ルーラは7列目から開始し、"*"記号で示される
  - ルーラ上8列目のAは、A領域の始まりの列であることを示す(8列目から11列目)
  - ルーラ上12列目のBは、B領域の始まりの列であることを示す(12列目から250列目)
  - ルーラは250列目まで表示される

タブを使用するとカーソルが設定ページで設定されたタブ幅の値に従って移動します。タブ幅の値は4または8が選択可能です。

タブ設定を変更する

1. メニューバーから[ウィンドウ] > [設定]を選択すると、[設定]ダイアログボックスが表示されます。
2. 左ペインから[COBOL] > [エディタ]を選択します。[エディタ]ページが表示されます。
3. [正書法]タブをクリックします。
4. [タブ幅]でタブ値として[4]または[8]を選択します。

注意

デフォルトでタブ値は[4]に設定されます。

10.1.11 カーソル位置表示

アクティブなファイルのカーソル位置に対応する行番号および列番号がステータスバーに表示されます。
固定形式および可変形式ファイルの場合、カーソルは7列目から始まります。

10.1.12 複数のソースファイルの表示

複数のCOBOLソースファイルをエディタにロードできます。エディタのタブをクリックして、エディタウィンドウの間で切り替えができます。

10.1.13 コードフォーマッタ

エディタは以下の場合における自動インデントをサポートします。

[ENTER]キーを押した場合	カーソル位置以降のスペースおよびタブはすべてテキストと共に次の行にコピーされ、カーソルは行頭に置かれる。
テキストを複数行貼り付ける場合	現在のカーソル位置からスタートし、続く行頭に余分なスペースやタブを入れないで貼り付ける。
入力支援候補一覧からテキストを挿入する場合	現在の行頭スペースおよびタブはすべて新規挿入行にも挿入される。

10.1.14 挿入/上書きモード

エディタ上のテキストを挿入または上書きできます。
選択された挿入または上書きモードはステータスバーに表示されます。

10.1.15 選択した行の強調表示

エディタ上で強調表示された行/語は、異なる背景色に変わります。

10.1.16 すべて選択

[編集]メニューから[すべて選択]を選択して、エディタのアクティブなファイル内のテキストをすべて選択できます。[Ctrl+A]キーを押しても同様です。

ただし、一連番号領域に存在する一連番号を選択することはできません。

10.1.17 元に戻す/やり直し

元に戻す

[編集]メニューまたはコンテキストメニューの[元に戻す]を使用して、以前に実行された編集操作(最大25操作まで)を取り消すことができます。[Ctrl+Z]またはツールバーの  をクリックしても同様です。

やり直し

[編集]メニューの[やり直し]を使用して、元に戻された編集操作をやり直すことができます。[Ctrl+Y]またはツールバーの  をクリックしても同様です。

10.1.18 左へシフト/右へシフト

選択した行のテキスト全体をエディタ上で移動できます。

左へシフト

[編集]メニューの[左へシフト]を使用して、選択したテキストをエディタ上で左に移動できます。[Shift+Tab]を押しても同様です。テキストの移動する列数は、[設定]ダイアログボックスの[COBOL] > [エディタ]ページで設定された[タブ幅]の値に基づきます。

右へシフト

[編集]メニューの[右へシフト]を使用して、選択したテキストをエディタ上で右に移動できます。[Tab]を押しても同様です。テキストの移動する列数は、[設定] ダイアログボックスの[COBOL] > [エディタ]ページで設定された[タブ幅]の値に基づきます。

注意

この操作を一連番号領域で行うことはできません。

10.1.19 切り取り/コピー/貼り付け

切り取り

[編集]メニューまたはコンテキストメニューの[切り取り]を使用して、選択したテキストを切り取ることができます。
[Ctrl+X]キーを押す、またはツールバーの  をクリックしても同様です。

コピー

[編集]メニューまたはコンテキストメニューの[コピー]を使用して、選択したテキストをコピーできます。
[Ctrl+C]キーを押す、またはツールバーの  をクリックしても同様です。

貼り付け

[編集]メニューまたはコンテキストメニューの[貼り付け]を使用して、切り取ったまたはコピーしたテキストを貼り付けることができます。
[Ctrl+V]キーを押す、またはツールバーの  をクリックしても同様です。

10.1.20 検索/置換

[編集]メニューの[検索/置換]を使用してキーワードの検索/置換ができます。
[Ctrl+F]キーを押す、またはツールバーの  をクリックしても同様です。

キーワード検索

1. メニューバーから[編集] > [検索/置換]を選択します。[検索/置換]ダイアログボックスが表示されます。
2. [検索]に検索したいキーワードを入力します。
3. [検索]をクリックします。キーワードが存在する場合はエディタ上で強調表示されます。

文字列の置換

1. メニューバーから[編集] > [検索/置換]を選択します。[検索/置換]ダイアログボックス が表示されます。
2. [検索]に検索したいキーワードを、[置換後]に置換後の文字列を入力します。
3. [置換]をクリックして検索されたテキストを置換します。

 注意

- [検索/置換]ダイアログボックスには下の詳細設定項目があります。

項目	説明
下へ	カーソル位置から下方向へと検索を進める(前進)
上へ	カーソル位置から上方向へと検索を進める(逆行)
すべて	ファイル全体から検索する。
選択された行	ファイルの選択した範囲から検索する。
大文字小文字の区別	大文字と小文字の区別をつけてキーワード検索する。
循環検索	ファイルの最後まで検索し終わるとファイルのはじめに戻って、検索をキャンセルするまで繰り返し検索し続ける。
単語全体	完全に一致する単語だけを検索する。
インクリメンタル	検索文字列の入力にあわせて、ファイル内の該当する文字列に位置付ける。

- [置換/検索]をクリックすると、一致する文字列を置換して次の一致する文字列へ移動します。
- [すべて置換]をクリックすると、一致する文字列がすべて置換されます。

10.1.21 指定行/一連番号へジャンプ

[ナビゲート]メニューの[指定行へジャンプ]を使って、一連番号または行番号に基づいて指定行に移動できます。[Ctrl+L]キーを押した場合も同様です。

固定形式および可変形式ファイルの場合の指定行/一連番号へジャンプ

1. メニューバーから[ナビゲート] > [指定行へジャンプ]を選択します。[指定行へジャンプ]ダイアログボックスが表示されます。
2. [行番号]を選択して[行番号の入力]に行番号を入力するか、[一連番号]を選択して[一連番号の入力]に一連番号を入力します。デフォルトでは、[一連番号]が選択されています。
3. 一連番号で実数の前にゼロがある場合は、実数だけ入力してもジャンプできます。例えば一連番号が000100の場合、000100と入力しないで100とだけ入力してもかまいません。
4. 指定行に移動してダイアログボックスを閉じる場合は[OK]をクリックします。カーソルが入力した行番号の開始位置に置かれます。
5. ダイアログボックスを閉じる場合は[キャンセル]をクリックします。

注意

- 固定形式または可変形式ファイルの場合、[一連番号の入力]の見出しに、指定可能な一連番号の範囲が表示されます。
- 入力した一連番号または行番号が無効である場合、メッセージが[指定行へジャンプ]ダイアログボックスに表示されます。

10.1.22 ブックマーク

[編集]メニューで[ブックマークの追加]を選択して、ソースコードの行にブックマークを付けることができます。またエディタのマーカバー上でコンテキストメニューを使用してもブックマークを付けることができます。

注意

ブックマーク一覧を表示するには、メニューバーから[ウィンドウ] > [ビューの表示] > [その他] を選択します。[ビューの表示]ダイアログボックスから[基本] > [ブックマーク]をクリックします。

10.1.23 タスク

[編集]メニューまたはエディタのコンテキストメニューで[タスクの追加]を選択して、タスクの設定が可能です。タスクは、例えばそのファイルに対する覚え書きを記録しておくために使用できます。設定したタスクはタスクビューに表示され、タスクが完了したかどうかを管理できます。タスクの詳細については、"ワークベンチユーザガイド"の"概念"の"ビュー"の"タスクビュー"を参照してください。

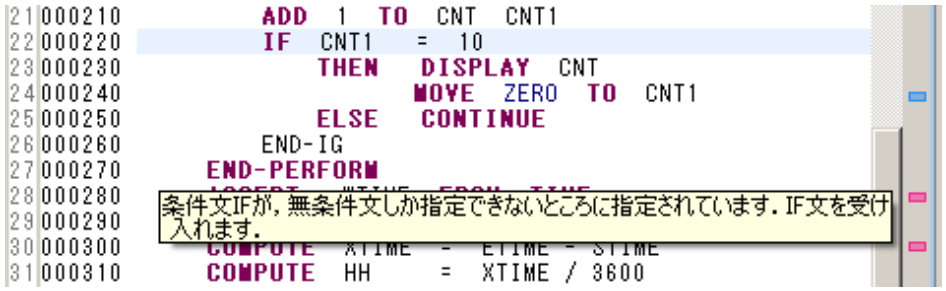
以下の手順でソースコードにタスクを挿入します。

1. マーカバーのコンテキストメニューから[タスクの追加]を選択します。
2. [新規タスク]ダイアログボックスの[説明]にタスクの説明を記述します。
3. 優先順位を選択し、タスクが完了している場合は[完了]をチェックします。
4. [リソース]、[フォルダ内]、[ロケーション]の値はデフォルトで表示されます。

10.1.24 概説ルーラ表示

ビルド時のエラー箇所などの位置を、エディタ右側の概説ルーラ上に色分けされた矩形で表示します。

ルーラ上の矩形をクリックすることによって、該当するソース行へジャンプできます。



COBOLエディタでサポートする表示項目を以下に示します。

- エラー
- タスク
- ブックマーク
- ブレークポイント
- 警告
- 検索結果
- 命令ポイントのデバッグ
- 呼出しスタックのデバッグ

表示設定の変更

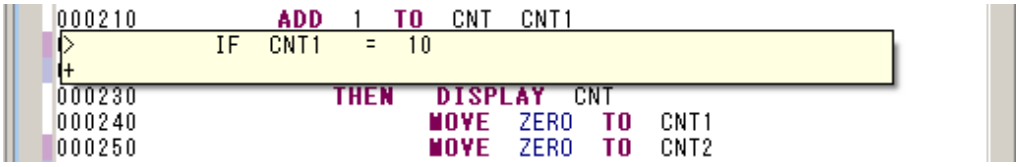
1. メニューバーから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
2. 左ペインから[一般] > [エディタ] > [テキストエディタ] > [注釈]を選択し、[注釈]ウィンドウを表示します。
3. [注釈型]の一覧から設定項目を選択し、概説ルーラへの表示／非表示の設定、および表示色を変更できます。

注意

表示設定は、他のエディタと共通となっているため、設定の変更は他のエディタの設定内容に影響を与えることに注意してください。

10.1.25 クイックDiff表示

エディタの垂直方向ルーラ横に現在編集中のソースコードと保存済みソースコードとの差分を色分けして表示できます。

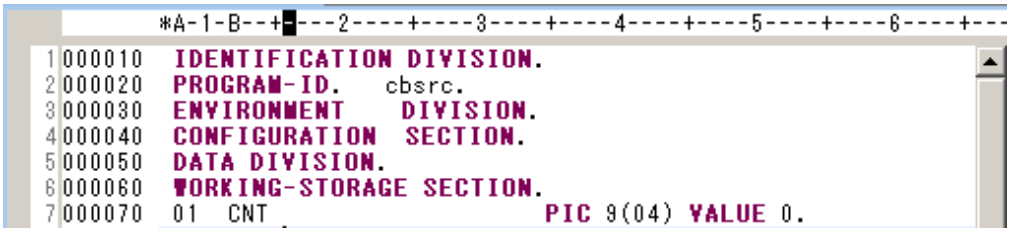


設定を変更する

1. メニューバーから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
2. 左ペインから[一般] > [エディタ] > [テキストエディタ] > [クイックDiff]を選択すると[クイックDiff]ウィンドウが表示されます。
3. ウィンドウ内の項目を操作して設定を変更します。

10.1.26 行番号表示

エディタの垂直方向ルーラ横に行番号を表示できます。



行番号を表示する

1. メニューバーから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。

2. 左ペインから[一般] > [エディタ] > [テキストエディタ]を選択すると、[テキストエディタ] ページが表示されます。
3. 行番号を表示する場合は、[行番号の表示]をチェックします。

注意

設定は、他のエディタと共通となっているため、設定の変更は他のエディタの設定内容に影響を与えることに注意してください。

10.2 IDLエディタ

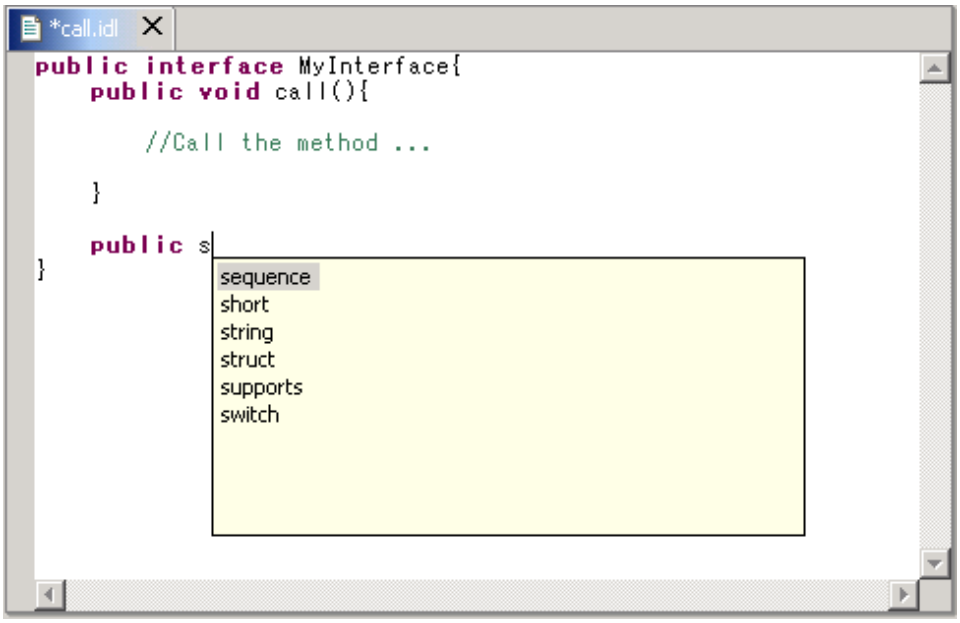
IDLエディタは、IDL(インタフェース定義言語)で記述されたファイルを編集するためのエディタです。IDLエディタにはIDLを記述するうえで便利なさまざまな機能があります。

10.2.1 入力支援候補一覧(コンテンツアシスト)

コードの作成時に、コンテンツアシスト(コードアシストとも呼ばれる)を使用できます。
行の有効な位置にカーソルを置き、以下のどれかの操作を行うと、使用可能な入力候補のリストを補助ウィンドウに表示します。

- メニューバーから[編集] > [コンテンツアシスト]を選択する。
- エディタのコンテキストメニューから[コンテンツアシスト]を選択する。
- [Ctrl+Space]キーを押す。

下図に、IDLエディタのコンテンツアシストの例を示します。



IDLエディタは有効な候補を見つけると、使用可能な入力候補のリストを補助ウィンドウに表示します。さらに入力し、そのリストを絞り込むことができます。

例えば、“s”とタイプして[Ctrl+Space]キーを押すと、“s”で始まるキーワードだけが補助ウィンドウに表示されます。

注意

カーソルが以下の位置にある場合は、コンテンツアシストを使用できません。

- 行コメント
- 複数行コメント

- 文字列や文字定数の途中

10.2.2 構文の強調表示

色とフォントスタイル(太字)で、キーワードを強調表示します。

10.2.3 強調色の設定

色およびフォントスタイル(太字)を使用して、キーワードを強調表示します。
以下の項目について、強調色が設定できます。

項目	説明
一行コメント	“//”で始まる行全体が一行コメントとして扱われます。
複数行コメント	“/*”と“*/”で囲まれるテキストは複数行コメントとして扱われます。
予約語	予約語は、大文字と小文字の区別をします。 予約語については、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。
文字列リテラル	' 'または" "で囲まれる文字が文字列リテラルとして扱われます。

強調色の設定を表示する

上記の項目について、色設定を表示できます。

1. メニューバーから[ウィンドウ] > [設定]を選択すると [設定]ダイアログボックスが表示されます。
2. 左ペインから[IDL] > [エディタ]を選択すると、[エディタ]ページが表示されます。
3. [色]タブをクリックすると[項目]、[色]、[太字]および[プレビュー]領域が表示されます。
4. 項目を選択すると設定されている[色]、[太字]にスタイルが表示されます。[プレビュー]には現在の設定内容が表示されます。

強調色の設定を変更する

上記の項目について、色設定を変更できます。

1. [項目]から表示色を変更したい項目を選択します。
2. [色]をクリックすると[色の設定]ダイアログボックスが表示されます。
3. [基本色]パレットから適用したい色をクリックします。[色の作成]をクリックして色を作成することもできます。
4. 選択した色を適用してダイアログボックスを閉じる場合、[OK]をクリックします。選択した色が[プレビュー]領域に反映されます。
   選択した色を適用しないで閉じる場合は、[キャンセル]をクリックします。
5. テキストを太字で表示する場合は、[太字]をチェックします。

注意

- デフォルトで予約語は太字、他のすべての項目は通常スタイルで表示されます。
- 設定内容は、次回以降にエディタを起動した場合でも有効です。

10.2.4 フォントの設定

IDLエディタのフォントは[設定]ダイアログボックスで変更できます。デフォルトでは、ワークベンチのフォントで設定されている内容になります。

フォントの設定を変更する

1. メニューバーから[ウィンドウ] > [設定]を選択すると[設定]ダイアログボックスが表示されます。
2. 左ペインから[IDL] > [エディタ]を選択すると[エディタ]ページが表示されます。
3. [全般]タブをクリックします。
4. [変更]をクリックすると[フォント]ダイアログボックスが表示されます。
5. 必要に応じて[フォント]、[スタイル]、[サイズ]などを変更します。

10.2.5 行番号の表示

エディタで行番号を表示または非表示にできます。

行番号を表示または非表示する

1. メニューバーから[ウィンドウ] > [設定]を選択すると[設定]ダイアログボックスが表示されます。
2. 左ペインから[一般] > [エディタ] > [テキストエディタ]を選択すると[テキストエディタ]ページが表示されます。
3. 行番号を表示する場合は、[行番号の表示]をチェックします。

10.2.6 現在行の強調表示

エディタで現在行を強調表示できます。

現在行を強調表示する

1. メニューバーから[ウィンドウ] > [設定]を選択すると[設定]ページが表示されます。
2. 左ペインから[一般] > [エディタ] > [テキストエディタ]を選択すると[テキストエディタ]ページが表示されます。
3. 現在行を強調表示する場合は、[現在行の強調表示]をチェックします。

10.2.7 カーソル位置表示

アクティブなファイルのカーソル位置を示す行・列番号が、ステータスバーに表示されます。

10.2.8 [編集]メニューのコマンド

IDLエディタが可視のときに表示される[編集]メニューのコマンドについて説明します。

名前	機能	キーボード ショートカット
元に戻す	編集操作(最大25操作まで)を取り消します。 コンテキストメニューで[元に戻す]を選択するか、またはツールバーの  をクリックします。	Ctrl+Z
やり直し	取り消した編集操作(最大25操作まで)をやり直します。 コンテキストメニューで[繰り返し]をクリックするか、またはツールバーの  をクリックします。	Ctrl+Y
切り取り	選択した範囲を切り取って、クリップボードに保存します。 コンテキストメニューで[切り取り]を選択するか、またはツールバーの  をクリックします。	Ctrl+X
コピー	選択した範囲をコピーして、クリップボードに保存します。 コンテキストメニューで[コピー]を選択するか、またはツールバーの  をクリックします。	Ctrl+C
貼り付け	クリップボードの内容を貼り付けます。 コンテキストメニューで[貼り付け]を選択するか、またはツールバーの  をクリックします。	Ctrl+V
削除	選択した範囲を削除します。	Delete

名前	機能	キーボード ショートカット
すべて選択	編集中のファイルの内容をすべて選択します。または、コンテキストメニューで[すべて選択]を選択します。	Ctrl+A
検索/置換	[検索/置換]ダイアログボックスを表示し、指定文字列を検索、または置換します。または、ツールバーの  をクリックします。	Ctrl+F
次を検索	直前に[検索/置換]ダイアログボックスで指定した文字列が、次に表示される位置を検索します。	Ctrl+K
前を検索	直前に[検索/置換]ダイアログボックスで指定した文字列が、前に表示される位置を検索します。	Ctrl+Shift+K
次をインクリメンタル検索	次をインクリメンタル検索モードで開始します。 呼び出したら、ステータスバーの指示どおりに検索テキストを入力してください。	Ctrl+J
前をインクリメンタル検索	前をインクリメンタル検索モードで開始します。 呼び出したら、ステータスバーの指示どおりに検索テキストを入力してください。	Ctrl+Shift+J
ブックマークの追加	ブックマークを現在のテキスト選択に追加します。	
タスクの追加	ユーザ定義タスクを現在のテキスト選択に追加します。	
単語補完	IDLエディタのカーソル位置で単語の入力を補完します。	Alt+/
コンテンツアシスト	コンテンツアシストを行います。	Ctrl+Space
エンコードの設定	エンコードを設定します。未保管の変更がある場合には選択できません。	

第11章　ビュー

ビューではファイル体系が表示されます。ここでは、ワークベンチでサポートされるビューについて説明します。

11.1 アウトラインビュー

アウトラインビューには、エディタ領域で開いている構造化されたファイルの概略および要素の一覧が表示されます。

COBOLエディタ上で現在アクティブなCOBOLソースファイルの構造の概略が一覧に表示されます。アウトラインビューで表示される内容は、COBOLプログラムの種別によって異なります。

アウトラインビューのツリー構造には、以下の内容が表示されます。

- 通常のCOBOLプログラムの場合
  - PROGRAM-ID
  - 環境部および節名
  - データ部および節名
  - 手続き部および節名と段落名
- オブジェクト指向COBOLプログラムの場合
  - CLASS-ID
  - FACTORY
  - OBJECT
  - METHOD-ID
  - 環境部および節名
  - データ部および節名
  - 手続き部および節名と段落名

COBOL用アウトラインビューの機能

COBOL用アウトラインビューには、以下の機能があります。

要素への移動

プログラム名、メソッド名などの要素を選択すると、エディタ上のカーソルが該当する要素の位置に移動します。

ツールバー

操作	アイコン	説明
ソート		アウトラインビュー上の要素のうち、節名、段落名、METHOD-IDの並び順を、ソースファイルに記述された順とアルファベット順で切り替えます。部の並び順とFACTORY・OBJECTの並び順はソートの対象とはなりません。
手続き部		アウトラインビュー上の要素の表示内容をすべて表示するか、環境部やデータ部の要素を表示しないで手続き部の情報だけを表示するかを切り替えます。

11.2 テンプレートビュー

テンプレートビューはCOBOL、CORBAサーバオブジェクトのルートカテゴリ、カテゴリおよびテンプレートを階層構造で表示します。テンプレートビューに表示されるテンプレートの内容は、アクティブなエディタで開いているファイルの拡張子により異なります。

テンプレートにはルートカテゴリ、カテゴリ、テンプレートがあります。ルートカテゴリはファイル拡張子と関連付けられます。ルートカテゴリと関連付けられた拡張子を持つファイルがエディタで開いているとき、テンプレートビューには対応するルートカテゴリが表示されます。

定義済みルートカテゴリの「COBOL」の場合は、次のファイル拡張子と関連付けられています。

- .cbl
- .cobol
- .cob

上記の拡張子を持つファイルがエディタで開いているとき、テンプレートビューにはルートカテゴリの「COBOL」が表示されます。

ルートカテゴリと関連付けられていない拡張子を持つファイルを開いたとき、またはエディタが開いていないときは、テンプレートビューにはルートカテゴリのないメインルートだけが表示されます。

設定の[テンプレート]ページまたはテンプレートビューを使用して、新しいテンプレートの作成や既存テンプレートの編集を行うことができます。

テンプレートビューの詳細は、ヘルプの"Interstage Studio ユーザーズガイド(互換ワークベンチ)"の"5.6 テンプレートビュー"を参照してください。

注意

「COBOL」は定義済みルートカテゴリであり、削除および名前を変更できません。

11.2.1 CORBAサーバオブジェクト一覧

CORBAサーバオブジェクト一覧の特徴を以下に示します。

- CORBAサーバオブジェクト一覧は、Interstageのインタフェースリポジトリに登録されているCORBAサーバオブジェクトの一覧が表示されます。
- CORBAサーバオブジェクト一覧にカテゴリやテンプレートを追加、編集、削除することはできません。
- CORBAサーバオブジェクトのルートカテゴリのツリー構造を示します。
  - モジュール
    CORBAサーバオブジェクト一覧フォルダの下にモジュールの一覧が表示されます。
  - インタフェース
    各モジュールフォルダの下に、そのモジュールで定義されているインタフェースの一覧が表示されます。
  - オペレータ
    各インタフェースフォルダの下に、そのインタフェースで定義されているオペレータの一覧が表示されます。

注意

CORBAサーバオブジェクト一覧は、インタフェースリポジトリを参照できる環境の場合だけ、各要素が表示されます。CORBAサーバオブジェクト一覧に何も要素が表示されない場合は、環境を見直してください。Interstageのサーバ機能をインストールしている場合は、インタフェースリポジトリを参照するのに必要なサービスが実行されているか確認してください。Interstageのクライアント機能をインストールしている場合は、"Interstageインストールフォルダ¥ODWIN¥etc¥INITHOST"に適切なサーバ名が設定されており、サーバが動作していることを確認してください。

11.2.1.1 CORBAサーバオブジェクト一覧固有のコンテキストメニュー

要素の種類	メニュー	説明
インタフェース	オブジェクト検索の挿入	選択しているインタフェースのサーバオブジェクトの検索処理をアクティブなCOBOLエディタのカーソル位置に挿入します。
オペレータ	挿入	オペレータを呼び出す処理をアクティブなCOBOLエディタのカーソル位置に展開します。引数は、param1, param2というデータ項目名で展開され、復帰値は、復帰値というデータ項目名で展開されます。

要素の種類	メニュー	説明
	引数の入力	オペレータの引数と復帰値のデータ項目名を入力するダイアログボックスを表示し、オペレータを呼び出す処理をアクティブなCOBOLエディタのカーソル位置に展開します。データ項目名はダイアログボックスで指定できます。
	引数を繰り返し入力する	オペレータを呼び出す処理を指定した数だけアクティブなCOBOLエディタのカーソル位置に展開します。引数を入力するダイアログボックスが表示され、データ項目名を入力できます。
	コピー	オペレータを呼び出す処理をクリップボードにコピーします。

注意

- 最新表示では、選択されているノードの配下しか最新状態に更新しません。
- インタフェースリポジトリの登録を行っても自動的にはCORBAサーバオブジェクト一覧には反映されません。最新表示を行ってください。

11.3 依存ビュー

依存ビューではCOBOL言語のプロジェクトをビルドする際に、翻訳するファイル、リンクするファイルおよび翻訳するファイルの依存関係をツリーで表示します。ナビゲータビューに表示されているファイルであっても、依存ビューの適切なフォルダに追加していないファイルは、翻訳、リンクの対象とはなりません。
依存ビューには、以下の3つのサブフォルダが表示されます。

- [ソースファイル]フォルダ

  翻訳対象となるCOBOLソースファイル(.cob, .cbl, .cobol)、プリコンパイラ入力ソースファイル、IDLファイル(.idl)、リソースファイル(.rc)を指定します。このフォルダに追加したファイルだけがそれぞれのコンパイラで翻訳されます。

- [リンクファイル]フォルダ

  リンクするファイルを表示します。翻訳対象ファイル以外に、追加でリンクするライブラリファイル(.lib)やオブジェクトファイル(.obj)を指定します。

- [その他のファイル]フォルダ

  [ソースファイル]フォルダにも[リンクファイル]フォルダにも追加していないプロジェクトのファイルが表示されます。

11.3.1 依存ビューの構造

COBOLの依存ビューにはプロジェクト内の依存関係が表示されます。COBOLソースファイルの構造、COBOLクラスリポジトリのクラス構造は表示されません。COBOLソースファイルの構造およびCOBOLクラスリポジトリのクラス構造は、構造ビューで表示されます。

COBOLプロジェクト生成時に、以下のフォルダが依存ビューに作成されます。

- ソースファイル
- リンクファイル
- その他のファイル

また、ソースファイルフォルダ内のCOBOLソースファイルには、ファイルの内容や指定されたオプションに従い、以下のフォルダが表示されます。

- ターゲットリポジトリ
- 依存関係ファイル
- ターゲットオブジェクト

これらのフォルダは、各ソースファイルの依存関係を指定するのに使用され、COBOLソースファイルの翻訳順番を自動的に決定するのに利用されます。

ソースファイル

プロジェクトで翻訳対象となるファイルを指定します。このフォルダに表示されているファイルは、ビルド時に翻訳され、リンクされます。拡張子が".cob"、".cbl"、".cobol"のファイルは、COBOLソースファイルとして扱われ、COBOLの翻訳が行われます。プリコンパイラ入力ソースとして登録されているファイルは、COBOLの翻訳前にプリコンパイラコマンドが呼び出されます。拡張子が".idl"のファイルは、IDLコンパイラビルドツールがプロジェクトに追加されている場合にだけ、IDL翻訳されます。拡張子が".rc"のファイルは、リソースコンパイラビルドツールがプロジェクトに追加されている場合にだけリソースコンパイラで翻訳されます。このフォルダにあるファイルは依存関係を考慮した上で、表示されている順番に翻訳されます。翻訳順番を変更するには、順番を変更するファイルを選択してから、ツールバーの  および  をクリックして上下にファイルを移動します。

注意

- リモート開発の場合、[ソースファイル]フォルダに登録されているファイルのうち、拡張子が".cobol"または".cob"のファイルはCOBOLソースファイルと扱われ、拡張子が".cbl"のファイルは登録集ファイルとして扱われます。
- 登録するCOBOLソースファイルのファイル名は、他のCOBOLソースファイルと重複しないよう指定してください。拡張子だけが異なる同名のCOBOLソースファイルを[ソースファイル]フォルダに登録すると、ビルド時にエラーとなります。

COBOLソースファイルの依存関係を指定するために、ターゲットリポジトリと依存関係ファイルをCOBOLソースファイルのサブフォルダで指定できます。これらの情報は、翻訳順番を自動的に決定する際に使用されます。

- ターゲットリポジトリ

  ターゲットリポジトリフォルダには、COBOLソースファイルが生成するリポジトリファイル(.rep)が表示されます。

- 依存関係ファイル

  このソースファイルが依存するファイルを指定します。ソースファイルが参照する登録集ファイルおよびソースファイル翻訳で必要となるリポジトリファイル(.rep)を指定します。また、クラス間で相互参照している場合は、継承しているファイルを指定します。

単一のCOBOLソースファイル中に、複数の翻訳単位(外部プログラムや外部クラス)がある場合、リンクするオブジェクトファイルがターゲットオブジェクトファイルに表示されます。

- ターゲットオブジェクトファイル

  ソースファイルに対して[ターゲットオブジェクトを指定する]と指定するとターゲットオブジェクトファイルフォルダが表示され、フォルダ内にターゲットオブジェクトファイルが表示されます。

リンクファイル

ビルド時にリンクするライブラリファイル(.lib)およびオブジェクトファイル(.obj)を指定します。ソースファイルフォルダで指定したソースファイルから生成されるオブジェクトファイルを指定する必要はありません。

その他のファイル

ソースファイルフォルダにもリンクファイルフォルダにも表示されていないプロジェクト内のファイルが表示されます。

ターゲットリポジトリ

COBOLソースファイルを翻訳したときに生成されるリポジトリファイルが、ソースファイルを保存したときに自動的に解析されて表示されます。リポジトリファイルは、COBOLソースファイル内で定義されているクラスに対応して生成されます。生成されるリポジトリファイル名は、"クラス名.rep"になります。このフォルダに表示されたリポジトリファイルは、COBOLソースファイルの翻訳順番を自動的に決定するとき、およびファイルが更新されたときにコンパイルする必要があるファイルの検索に使用されます。

依存関係ファイル

COBOLソースファイルが依存しているファイルを指定します。COBOLソースが参照しているCOBOL登録集や参照しているクラスのリポジトリファイルを指定します。ここで指定したファイルは、COBOLソースファイルの翻訳順番を自動的に決定するとき、およびファイルが更新されたときにコンパイルする必要があるファイルの検索に使用されます。[ファイルの追加]でファイルを追加します。また、[依存関係の解析]を実行することで、自動的にソースファイルを解析し、依存するファイルを追加することもできます。

ターゲットオブジェクト

[ターゲットオブジェクトを指定する]を指定した場合に表示されます。ソースファイルから生成されるオブジェクトファイルが表示され、ここで表示されているオブジェクトファイルがリンクされます。表示されるオブジェクトファイルは、ソースファイルを保存したときに更新されます。

注意

リモート開発の場合、[依存関係ファイル]フォルダに登録されているCOBOL登録集・定義体ファイルのうち転送対象となるのはプロジェクト内にあるファイルだけです。他のプロジェクトまたは他のフォルダで管理されているCOBOL登録集・定義体ファイルは転送の対象となりません。また、サーバへの転送対象となる登録集ファイルは拡張子"cbl"のファイルのみとなります。

11.3.2 依存ビューのコンテキストメニュー

依存ビューに固有のコンテキストメニューを以下に示します。

要素	メニュー	説明
プロジェクト	依存関係の解析	ソースファイルフォルダ内にあるすべてのCOBOLソースを解析し、ターゲットリポジトリおよび依存ファイルを抽出します。
[ソースファイル]フォルダ	ファイルの追加	既存のファイルをソースファイルフォルダに追加します。 詳細は、"依存ビューでソースファイルを追加する"を参照してください。
	依存関係の解析	ソースファイルフォルダ内にあるすべてのCOBOLソースを解析し、ターゲットリポジトリおよび依存ファイルを抽出します。
[リンクファイル]フォルダ	ファイルの追加	既存のファイルをリンクファイルフォルダに追加します。 詳細は、"依存ビューでリンクファイルを追加する"を参照してください。
[依存関係ファイル]フォルダ	ファイルの追加	依存関係ファイルに登録集ファイルやリポジトリファイルなどの依存ファイルを追加します。
	クリア	依存関係ファイルフォルダ内のすべてのファイルを削除します。
[ソースファイル]フォルダ内のファイル選択時	ファイルの追加	既存のファイルをソースファイルフォルダに追加します。 詳細は、"依存ビューでソースファイルを追加する"を参照してください。
	ソースファイルから削除	ソースファイルフォルダからファイルを削除し、[その他のファイル]フォルダに表示します。ファイルシステム上のファイルは削除しません。
	主プログラム	選択したソースプログラムをこのプロジェクトの主プログラムとして設定します。 主プログラムは、プロジェクトに1つしか設定できません。詳細は、"6.1.3.2 主プログラムの設定"を参照してください。
	ターゲットオブジェクトを指定する	ソースファイルに複数の翻訳単位(外部プログラム、外部クラス)がある場合に指定します。このメニューをチェックすると、チェックされたファイルは、NAME翻訳オプションが追加されて翻訳されます。
	依存関係の解析	選択されているCOBOLソースファイルを解析し、ターゲットリポジトリおよび依存ファイルを抽出します。
	インタフェースリポジトリの登録	選択されたIDLファイルをインタフェースリポジトリに登録します。
[依存関係ファイル]フォルダ内のファイル選択時	ファイルの追加	依存関係ファイルに登録集ファイルやリポジトリファイルなどの依存ファイルを追加します。
[リンクファイル]フォルダ内のファイル選択時	ファイルの追加	既存のファイルをリンクファイルフォルダに追加します。 詳細は、"依存ビューでリンクファイルを追加する"を参照してください。
[その他のファイル]フォルダ内のファイル選択時	ソースファイルへ追加	選択されているファイルを[ソースファイル]フォルダに追加します。[ソースファイル]フォルダにファイルが追加されると、[その他のファイル]フォルダからはそのファイルが消えます。

要素	メニュー	説明
		プロジェクトの格納先と異なるフォルダにあるファイルは、プロジェクトの格納先へファイルがコピーされます。

11.3.3 依存ビューのファイル関連手順

依存ビューにおけるファイル関連の操作手順について説明します。依存ビュー内でのファイル関連の操作は、構造ビューにも反映されます。

依存ビューでソースファイルを追加する

・ プロジェクト内のCOBOLソースファイルの場合

以下の手順でプロジェクト内のCOBOLソースファイルを追加します。

1. 依存ビューで[ソースファイル]フォルダを選択します。
2. コンテキストメニューから[ファイルの追加]を選択します。
3. 選択したプロジェクトに対応するファイル一覧のダイアログボックスが表示されます。
4. ソースファイルフォルダに追加するファイルを選択します。
5. [OK]をクリックすると、選択したCOBOLソースファイルが依存ビューのソースファイルフォルダに追加されます。追加されたファイルは、構造ビューにも反映されます。

・ プロジェクト外のCOBOLソースファイルの場合

プロジェクト外に存在するCOBOLソースファイルは以下のどちらかの方法で追加します。

- Windowsのエクスプローラで登録するCOBOLソースファイルを選択し、コンテキストメニューから[コピー]を選択します。ソースファイルフォルダを選択し、コンテキストメニューから[貼り付け]を選択します。
- Windowsのエクスプローラからソースファイルフォルダへ、COBOLソースファイルをドラッグ&ドロップします。

依存ビューでリンクファイルを追加する

1. ファイルを追加したい[リンクファイル]フォルダを選択します。
2. コンテキストメニューから[ファイルの追加]を選択します。ファイル選択ダイアログボックスが表示されます。
3. リンクファイルフォルダに追加したいファイルを選択します。
4. [OK]をクリックすると、依存ビューのリンクファイルフォルダに選択されたファイルが追加されます。追加されたファイルは、構造ビューにも反映されます。

依存ファイルを追加する

1. ファイルを追加したい[依存関係ファイル]フォルダを選択します。
2. コンテキストメニューから[ファイルの追加]を選択します。ファイル選択ダイアログボックスが表示されます。
3. 依存ファイルフォルダに追加したいファイルを選択します。
4. [OK]をクリックすると、依存関係ファイルフォルダに選択されたファイルが追加されます。

依存関係ファイルを削除する

1. 依存関係ファイルフォルダから削除したいファイルを選択します。
2. コンテキストメニューから[削除]を選択します。
3. ファイルの削除を確認するダイアログボックスが表示されますので[OK]をクリックします。依存関係ファイルフォルダからファイルが削除されます。

注意

依存関係ファイルの削除は依存関係からファイルを削除するものです。依存関係ファイルフォルダからファイルを削除しても、システム上のファイルは削除されません。

11.4 構造ビュー

構造ビューには、ソースファイル、リンクファイルおよびその他のファイルを表示するサブフォルダがあります。ソースファイルに関しては、内部の構造を階層的に表示します。

ワークスペース内のCOBOLに関係する以下の情報が表示されます。

- ソースファイル

  翻訳対象となるソースファイルを表示します。この情報は依存ビューの[ソースファイル]と同等です。COBOLソースファイル内の構造は各ソースファイル配下に階層表示されます。

  ソースプログラム配下に表示される構造は以下のようになります。

  ー 通常のCOBOLプログラムの場合

  - PROGRAM-ID
  - 環境部および節名
  - データ部および節名
  - 手続き部および節名と段落名

  ー オブジェクト指向COBOLプログラムの場合

  - CLASS-ID
  - FACTORY
  - OBJECT
  - METHOD-ID
  - 環境部および節名
  - データ部および節名
  - 手続き部および節名と段落名

- リンクファイル

  リンク対象とするファイルを表示します。翻訳対象ファイル以外で、リンク対象となるファイルを表示します。この情報は依存ビューの[リンクファイル]と同等です。

- その他のファイル

  翻訳対象でもなく、リンク対象でもないファイルを表示します。この情報は依存ビューの[その他のファイル]と同等です。

11.4.1 構造ビューのコンテキストメニュー

構造ビューに固有のコンテキストメニューを以下に示します。

要素	メニュー	説明
[ソースファイル]フォルダ	ファイルの追加	既存のファイルを[ソースファイル]フォルダに追加します。
[リンクファイル]フォルダ	ファイルの追加	既存のファイルを[リンクファイル]フォルダに追加します。
[ソースファイル]フォルダ内のファイル選択時	ファイルの追加	既存のファイルを[ソースファイル]フォルダに追加します。

要素	メニュー	説明
	ソースファイルから削除	[ソースファイル]フォルダからファイルを削除し、[その他のファイル]フォルダに表示します。ファイルシステム上のファイルは削除しません。
	主プログラム	選択したソースプログラムをこのプロジェクトの主プログラムとして設定します。 主プログラムは、プロジェクトに1つしか設定できません。詳細は、"6.1.3.2 主プログラムの設定"を参照してください。
	ターゲットオブジェクトを指定する	ソースファイルに複数の翻訳単位(外部プログラム、外部クラス)がある場合に指定します。このメニューをチェックすると、チェックされたファイルは、NAME翻訳オプションが他の翻訳オプションに追加されてCOBOL翻訳されます。
	インタフェースリポジトリの登録	選択されたIDLファイルをインタフェースリポジトリに登録します。
[リンクファイル]フォルダ内のファイル選択時	ファイルの追加	既存のファイルを[リンクファイル]フォルダに追加します。
[その他のファイル]フォルダ内のファイル選択時	ソースファイルへ追加	選択されているファイルを[ソースファイル]フォルダに追加します。[ソースファイル]フォルダにファイルが追加されると、[その他のファイル]フォルダからはそのファイルが消えます。 プロジェクトの格納先と異なるフォルダにあるファイルは、プロジェクトの格納先へファイルがコピーされます。

11.5 ウォッチビュー

ウォッチビューはCOBOLアプリケーションデバッグ中に、データ項目の値を表示するために使用します。

コンテキストメニューおよびツールバー

操作	説明
データ項目の追加	データ項目の値を表示するために、ウォッチビューにデータ項目を追加します。
値の変更	選択したデータ項目の値を変更します。データ項目をダブルクリックしても同様の操作となります。
16進入力	選択したデータ項目の値を16進で変更します。
削除	選択したデータ項目をウォッチビューから削除します。
値変更時に中断	データ項目の値が変更された場合にプログラムを中断するかどうかを指定します。データ項目単位に指定できます。
すべて削除	すべてのデータ項目をウォッチビューから削除します。
データ型の表示	ウォッチビューにデータ名だけでなくデータ型を表示するかどうかを指定します。
すべて縮小表示	すべての集団項目の表示を縮小します。
垂直方向のビュー	詳細ペインを垂直方向のビューで表示します。詳細ペインには、プログラム名、データ型、値、値変更時に中断の情報が表示されます。
水平方向のビュー	詳細ペインを水平方向のビューで表示します。詳細ペインには、プログラム名、データ型、値、値変更時に中断の情報が表示されます。
変数ビューのみ	詳細ペインを表示しません。

ウォッチビューへのデータ項目の追加

コンテキストメニューやツールバーから[データ項目の追加]を選択することにより、ウォッチビューにデータ項目を追加できます。

1. コンテキストメニューやツールバーから[データ項目の追加]を選択します。
2. [データ項目の追加]ダイアログボックスが表示されます。

3. データ名、プログラム名、値変更時に中断などを指定し、ウォッチビューにデータ項目を追加します。
   クラスのメソッドに定義されているデータ項目を追加する場合には、プログラム名に「クラス名:メソッド名」を指定します。

COBOLエディタ上でデータ項目をダブルクリックし、コンテキストメニューから[ウォッチビューへ追加]を選択した場合もウォッチビューへの追加が行えます。

注意

ウォッチビューへのデータ項目の追加は、プログラムのデバッグ実行中にだけ行えます。プログラムをデバッグ実行し、ブレークポイントなどでその実行が中断している時に、ウォッチビューへのデータ項目の追加を行ってください。

11.6 ブレークポイントビュー

ここでは、COBOLに関する機能について説明します。

ブレークポイントのヒットカウント

ブレークポイントのヒットカウントを使用することにより、ブレークポイントの行が指定されたヒットカウント数の回数実行された場合に中断するようにできます。

1. COBOLエディタでブレークポイントを設定します。
2. ブレークポイントビューでブレークポイントを選択してコンテキストメニューから[ヒットカウント]を選択します。
3. [ブレークポイントヒットカウントの設定]ダイアログボックスで、ブレークポイントのヒットカウントの設定、およびブレークポイントのヒットカウント有効無効を設定します。

ブレークポイントのプロパティ

ブレークポイントの有効無効の設定、およびブレークポイントのヒットカウントとブレークポイントのヒットカウントの有効無効を設定できます。

1. COBOLエディタでブレークポイントを設定します。
2. ブレークポイントビューでブレークポイントを選択してコンテキストメニューから[プロパティ]を選択します。
3. [COBOL行ブレークポイントのプロパティ]ダイアログボックスで、ブレークポイントの有効無効の設定、およびブレークポイントのヒットカウントとブレークポイントのヒットカウントの有効無効を設定します。

第4部 チュートリアル編

第12章　CORBAアプリケーションの開発について

COBOLプラグインを組み込んだInterstage Studio 互換ワークベンチを使用してCORBAアプリケーションを作成できます。

このチュートリアルでは、Interstage Application Server を便宜的にInterstage と記述している箇所があります。また、Interstageのマニュアルの参照を指示している箇所では、特に指定がない限り以下を参照してください。

インストール、環境設定について

Interstage Application Server インストールガイド

Interstage Application Server 運用ガイド（基本編）

CORBA仕様全般、CORBA通信について

Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)

Interstageの提供するコマンド、IDL書式、ワークユニット定義書式について

Interstage Application Server リファレンスマニュアル

メッセージについて

Interstage Application Server メッセージ集

CORBAアプリケーションとは

CORBA(Common Object Request Broker Architecture)とは、OMG(Object Management Group : オブジェクト指向技術の標準化と普及を目的として1989年に設立された非営利団体)によって規定されたオブジェクト指向技術の仕様で、異機種、異言語間での接続可能な分散アプリケーションを作成できます。

富士通では、Interstage Application ServerでCORBA準拠の分散通信基盤やサービスを提供しています。Interstage Studioにおいては、CORBAアプリケーションとは、Interstage Application Serverを利用して作成したアプリケーションを意味します。また、COBOLプラグインを組み込んだInterstage Studio 互換ワークベンチでは、CORBAサーバアプリケーションおよびCORBAクライアントアプリケーションの両方を作成できます。

CORBAアプリケーションの種類とサポート範囲

アプリケーションの種類	説明	サーバ	クライアント
静的インタフェース	IDLファイルから作成したスタブ、スケルトンファイルをアプリケーションに結合してプログラムを作成します。スタブ、スケルトンは、サーバ、クライアント間で使用されるデータをCORBA通信基盤のプロトコル(IIOP)から各言語タイプに変換する機能を持っています。	オブジェクト指向COBOL	COBOL
CORBA-Gateway	WindowsシステムのOLEアクセスにより、サーバアプリケーションの提供する関数を呼び出します。	-	×
動的インタフェース	スタブファイル等は必要なく、インタフェースリポジトリから情報を取り出し、サーバのメソッドを呼び出すパラメタをプログラム中で組み立てて、サーバアプリケーションの関数を呼び出します。	×	×

CORBAアプリケーション作成のための環境設定

CORBAアプリケーション開発に必要なプログラム

- Interstage Studio 互換ワークベンチ
- COBOLプラグイン(Interstage Business Application Server 開発環境パッケージのCOBOL開発支援ツールでインストール)
- Interstage Application Serverサーバ機能 もしくは クライアント機能
- NetCOBOL V8.0L10 以降

Interstage Application Serverの環境設定の確認

CORBAアプリケーションの開発では、IDLファイルをコンパイルしますが、その場合は必ずCORBAサーバにアクセスできる環境が設定されていなければなりません。
Interstage Application Serverサーバ機能と組み合わせて開発する場合には環境設定は必要ありませんが、クライアント機能と組み合わせて開発する場合には環境設定が必要です。
クライアント機能と組み合わせる場合には、Interstage Application Serverインストールフォルダ¥odwin¥etc¥inithostファイルに、アクセスするサーバを指定する必要があります。
CORBAアプリケーションのビルド時にIDLコンパイラでエラーが発生する場合には、inithostの記述内容を確認してください。

Interstage StudioによるCORBAアプリケーション開発の概要

以下のような手法でCORBAアプリケーションを開発できます。

CORBAサーバアプリケーション開発

CORBAサーバプロジェクトを使用することで、必要なビルドオプションが設定されます。
CORBAサーバアプリケーション作成ウィザードで、CORBAサーバアプリケーションのひな型(IDLファイルおよびプログラムソース)を作成します。
ひな型作成後は、IDLファイルの編集、プログラムソースの編集は自由に行うことができますが、その場合は必ずIDLファイルとプログラムソースの内容の同期をとる必要があります。例えば、すでに定義済のメソッドの定義を変更した場合は、IDLファイル、プログラムソースの両方を修正する必要があります。

CORBAクライアントアプリケーション開発

COBOLプロジェクトを使用し、プロジェクトウィザードで[CORBAクライアントのビルド環境を設定]オプションを指定することで、必要なビルド環境を自動的に設定できます。
CORBAサーバにアクセスする処理については、テンプレートを利用して作成します。
CORBAスタブファイル生成ウィザードで、CORBAサーバアプリケーションのIDLファイルからスタブファイルを作成し、ビルドすることでクライアントアプリケーションを作成します。

Interstage Studioによる生成物

ウィザードで生成されるファイルおよびビルド時に生成されるターゲットファイルの一覧を以下の表に示します(IDLファイルから生成されるスタブおよびスケルトンファイルは含みません)。

CORBAサーバアプリケーション開発

プロジェクト名 = intf1、モジュール名 = module1 の場合(※プロジェクト名は自動的にインタフェース名として採用されます。)

説明	ファイル名
メインソース(COBOL)	intf1.cob
ビジネスメソッドソース(COBOL)	USintf1.cob
サーバアプリケーション登録ソース(COBOL)	USINITintf1.cob
IDLファイル	USintf1.idl
ターゲット	intf1.exe USintf1.dll

IDLファイルは以下の形式で生成されます。

```
// モジュール宣言
module MDSample {

    // 定数宣言
    const unsigned long const1 = 1 ;

    // 型宣言
    typedef unsigned long type1 ;

    // 構造体宣言
    struct S1 {
```

```
        unsigned long item1;
        unsigned long item2;
    };

    // 例外宣言
    exception CDException{
        string CDExceptionMsg;
        long CDExceptionCode;
    };
    // ユーザインタフェース宣言
    interface SAMPLE {
        void OP1(in unsigned long param1)
            raises (CDException);
    };

};
```

- モジュール宣言
  一番外側の宣言は、必ずmodule宣言です。その中に、ウィザードの各ダイアログで入力した宣言が生成されます。
- 定数宣言
  ウィザード中の定数宣言で行った宣言の内容が宣言されます。
- 型宣言
  ウィザード中の型宣言で行った宣言の内容が宣言されます。通常は繰り返し項目等を宣言します。
- 構造体宣言
  ウィザード中の構造体宣言で行った宣言の内容が宣言されます。
- 例外宣言
  ウィザードの最初で[例外の生成]を選択したとき、標準的な例外宣言の型としてCDExceptionを定義します。他に情報等を追加したい場合や、変更したい場合は、ウィザード終了後、直接IDLファイルを修正してください。
- ユーザインタフェース宣言
  ウィザードで定義したビジネスメソッドをオペレータとして宣言したインタフェースを宣言します。

その他の開発機能

COBOLプラグインを組み込んだInterstage Studio 互換ワークベンチでは、CORBAサーバアプリケーション生成ウィザードの他に以下の開発機能を提供します。

インタフェースリポジトリの登録

IDLコンパイラビルドツールのオプション[インタフェースリポジトリに登録する]をチェックしている場合、ビルド時に、プロジェクトに登録してあるIDLファイルをインタフェースリポジトリに登録します。
また、プロジェクトに登録してあるIDLファイルをエディタで開き、コンテキストメニューの[インタフェースリポジトリの登録]を選択した場合も、CORBAサーバのインタフェースリポジトリにIDLの内容を登録できます。登録した内容は、Interstage Studioのテンプレートの CORBAサーバオブジェクト一覧で参照できます。

サーバオブジェクト一覧機能

クライアントアプリケーション作成時、テンプレートのCORBAサーバオブジェクト一覧で、CORBAサーバのインタフェースリポジトリからモジュール宣言、インタフェース宣言、オペレータ宣言の内容を表示できます。また、オペレータ宣言を選択して、ソースにオペレータ(ビジネスメソッド)の呼び出し処理を挿入できます。

定型処理入力

クライアントアプリケーション作成時の定型処理をテンプレートからソースに挿入できます。テンプレートビューから定型処理を選択し、コンテキストメニューから[編集]を選択することで、定型処理をカスタマイズできます。

CORBAスタブファイル生成ウィザード

IDLファイルからスタブファイルを生成し、クライアントアプリケーションのプロジェクトにソースを登録できます。

CORBAクラス

CORBAクライアントアプリケーションを開発時に、CORBAの初期化処理やオブジェクトの検索処理を簡潔に記述するためにCDCORBAクラスを提供しています。

CORBAアプリケーションの実行

CORBAサーバアプリケーションはワークユニット上で実行します。以下に、CORBAサーバアプリケーションの動作確認の手順の概要を示します。

1. 実行資産のコピー
   ビルドで生成された実行ファイルとダイナミックリンクライブラリを実行環境にコピーします。
2. ワークユニット作成
   Interstage管理コンソールを使い、実行環境フォルダなどを指定してワークユニットを作成します。
3. CORBAサーバアプリケーションの配備
   Interstage管理コンソールから、コピーした実行資産の配備を行います。
4. ワークユニットの起動
   作成したワークユニットを、Interstage管理コンソールから起動します。
5. クライアントアプリケーションの実行
   クライアントアプリケーションを実行し、CORBAサーバアプリケーションにアクセスします。

データベースアクセスする場合のCORBAサーバアプリケーション

データベースアクセスを行うCORBAサーバアプリケーションを作成する場合は、以下の2つの方法があります。

- 事前にデータベースをアクセスするためのクラスライブラリを作成し、そのクラスを使用するCORBAサーバアプリケーションを作成する。クラスライブラリ化することで、部品として他アプリケーションでも流用できます。
- CORBAサーバアプリケーションのビジネスメソッド内に、直接データベースアクセスする命令を記述する。

COBOLの提供するデータベースアクセス方法(ESQL/COBOLなど)の詳細は、NetCOBOLのソフトウェア説明書を参照してください。

プログラム作成の留意事項

CORBAサーバアプリケーションのターゲット生成規則について

CORBAサーバアプリケーションでは、ビルド時に、メインプログラム(.EXE)、ビジネスロジックのライブラリ(.DLL)のターゲットが作成されます。
依存ビューでメインプログラムとして設定しているCOBOLソースからメインプログラムは生成され、その他のCOBOLソースはすべてビジネスロジックのライブラリに結合されます。

UNICODEオブジェクトを作成する場合

Interstage Application ServerにはUNICODE対応用のリポジトリファイル、ライブラリファイルが提供されています。詳細は、NetCOBOLのマニュアル、Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)を参照してください。
CORBAサーバプロジェクトウィザードで[Unicodeを使用する]を指定することにより、UNICODE対応用のファイルを使用してビルドを行うことができます。

第13章　CORBAサーバアプリケーション開発

ここでは、実際にCORBAサーバアプリケーションを作成しながら説明します。

1. 作成するプログラムの内容

ここでは、2項演算プログラムを作成します。

- 2つのパラメタの四則演算を行うプログラム
- メソッドは、加減乗除に対してそれぞれ、addop、subop、mltop、divopというメソッド名にする
- それぞれのメソッドは復帰値なしで、パラメタを構造体形式にして、その構造体の中に演算結果を設定しクライアント側で参照可能にする。(パラメタはクライアントで設定し、かつ、演算結果はサーバアプリで設定するため、パラメタのモードはinoutにする必要がある)
- 乗算、除算の場合は、パラメタをチェックしどちらか一方が0である場合、例外情報を返却する。

2. Interstage基盤サービスの起動

CORBAサーバアプリケーションのプロジェクトを作成する前に、Interstage基盤サービス操作ツールによってJ2EE実行環境のためのサービスをあらかじめ起動しておく必要があります。デフォルトの状態では必要なサービスは起動されていないため、以下の手順でサービスを起動してください。

1. スタートメニューから[Interstage] > [Studio] > [Interstage基盤サービス操作ツール]を選択します。
2. Interstage基盤サービス操作ツールのウィンドウの[J2EE実行環境を使用する]チェックボックスをチェック状態にします。
3. [必須サービスのみ起動状態にする]ボタンが有効の場合、ボタンをクリックしてサービスを起動します。ボタンが無効の場合は操作は不要です。
4. <必要なサービス>が全て実行中となっていることを確認してから[閉じる]ボタンをクリックしてInterstage基盤サービス操作ツールを終了します。

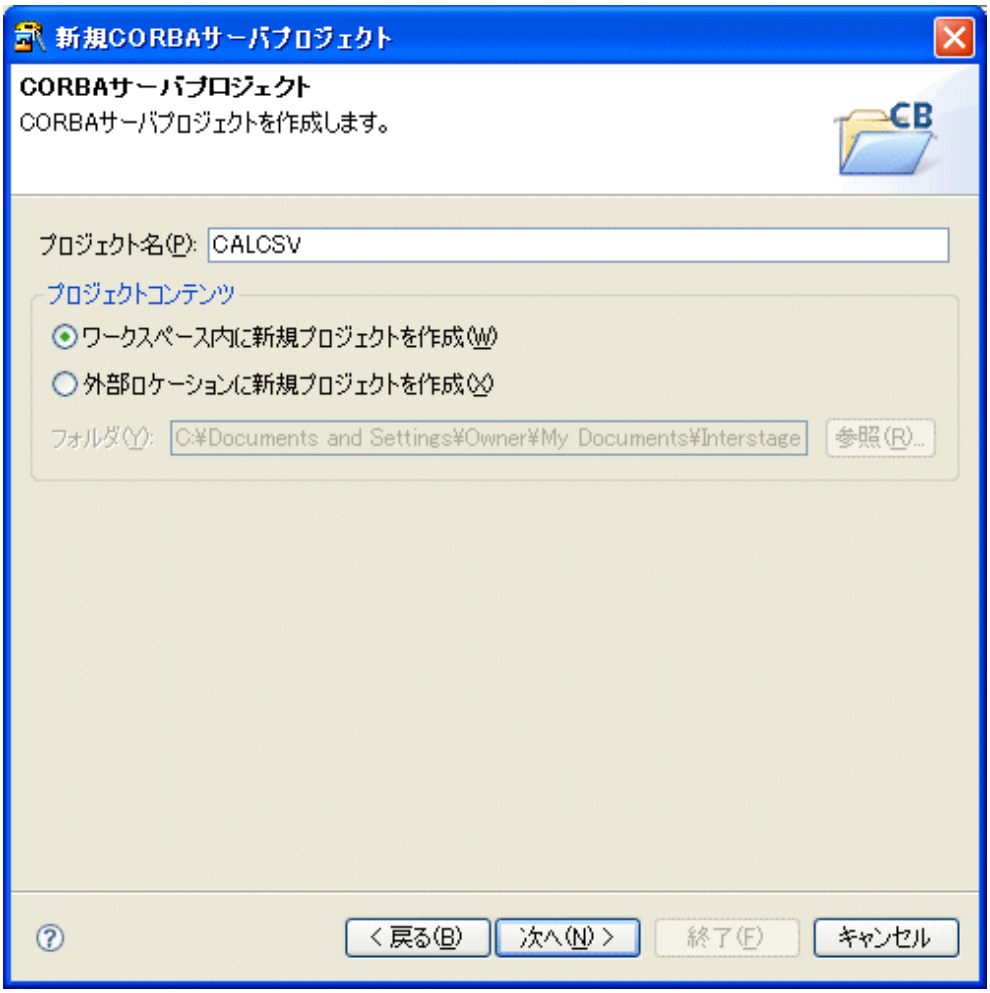
3. プロジェクトの作成

COBOLパースペクティブを表示していない場合は、以下の手順でCOBOLパースペクティブを表示してください。

1. メニューバーから[ウィンドウ] > [パースペクティブを開く] > [その他]を選択すると[パースペクティブを開く]ダイアログボックスが表示されます。
2. [COBOL]を選択して[OK]ボタンをクリックします。

ワークベンチのメニューバーから[ファイル] > [新規] > [CORBAサーバプロジェクト]を選択すると、新規CORBAサーバプロジェクトが表示されます。

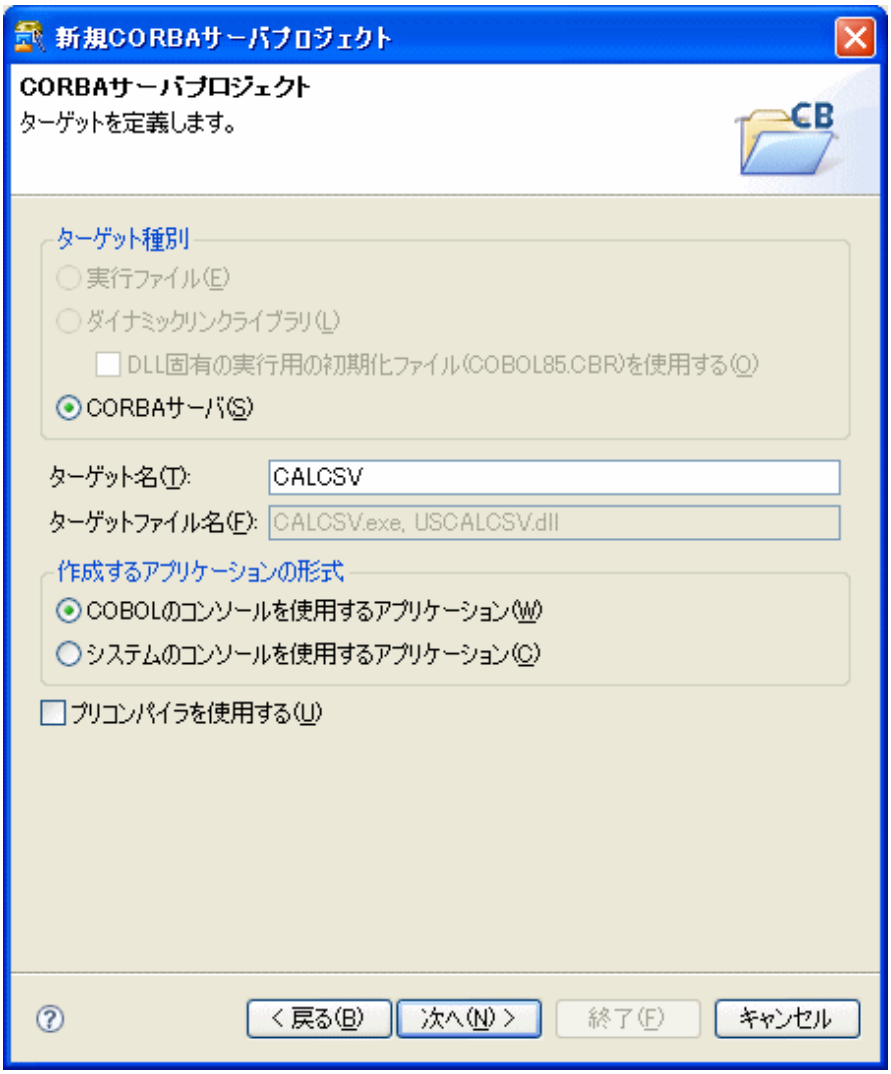
[1枚目]



プロジェクト名と保存フォルダを入力します。

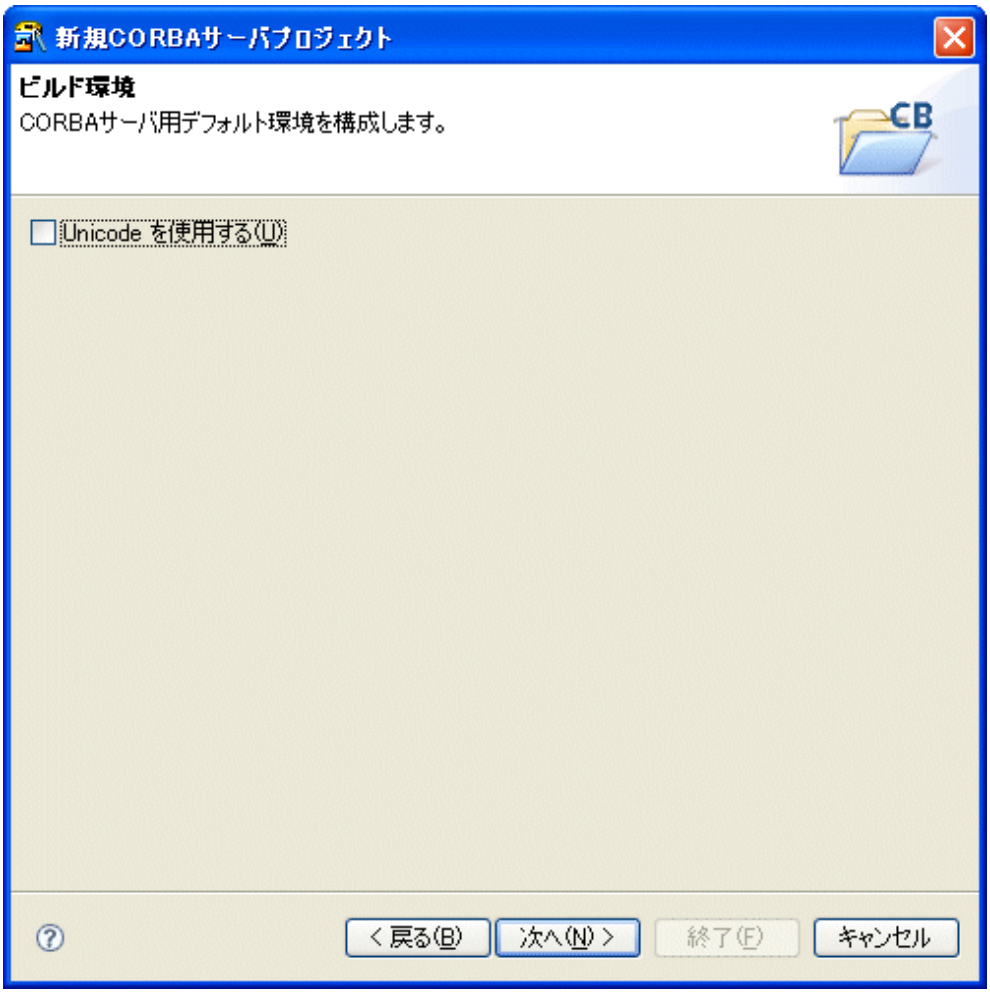
設定項目	設定内容
プロジェクト名	CALCSV
プロジェクトコンテンツ	[ワークスペース内に新規プロジェクトを作成]を選択

[2枚目]



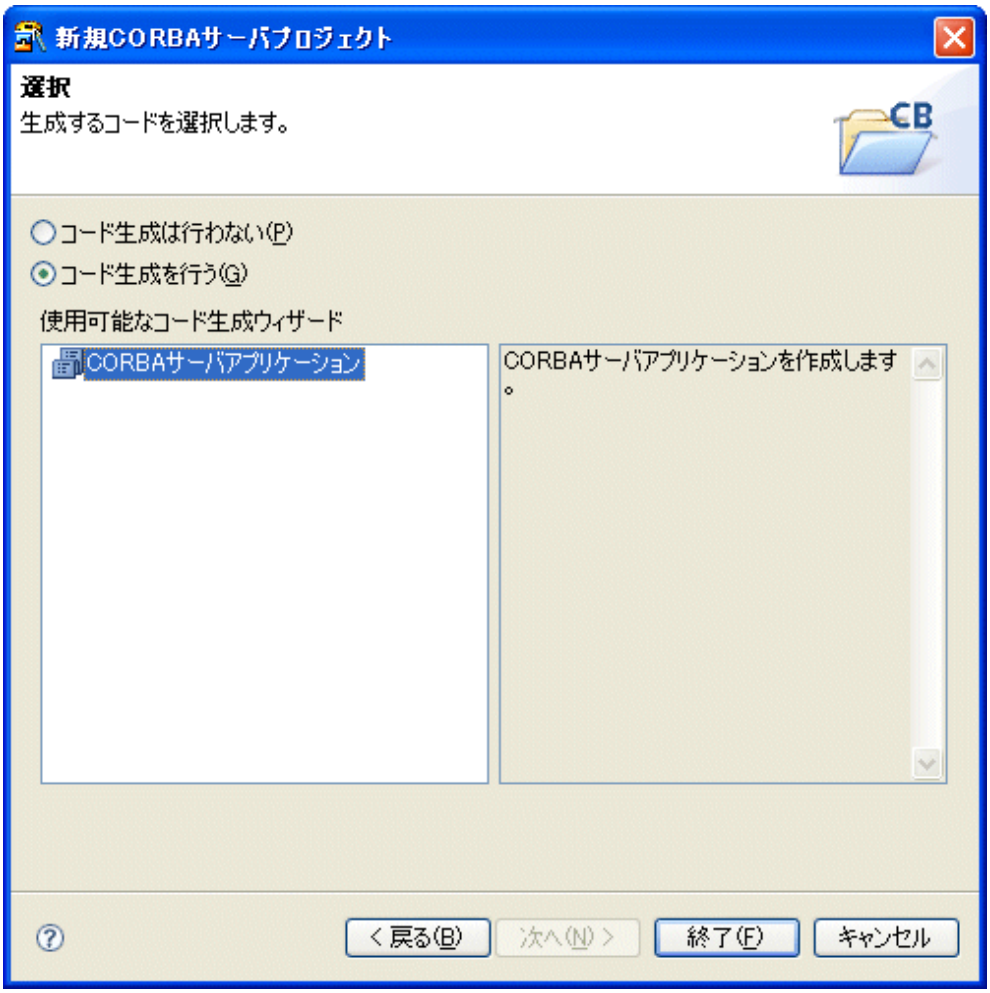
ターゲットを指定する画面です。ここでは何も変更せずに次に進みます。

[3枚目]



ビルド環境を指定する画面です。ここでは何も変更せずに次に進みます。

[4枚目]

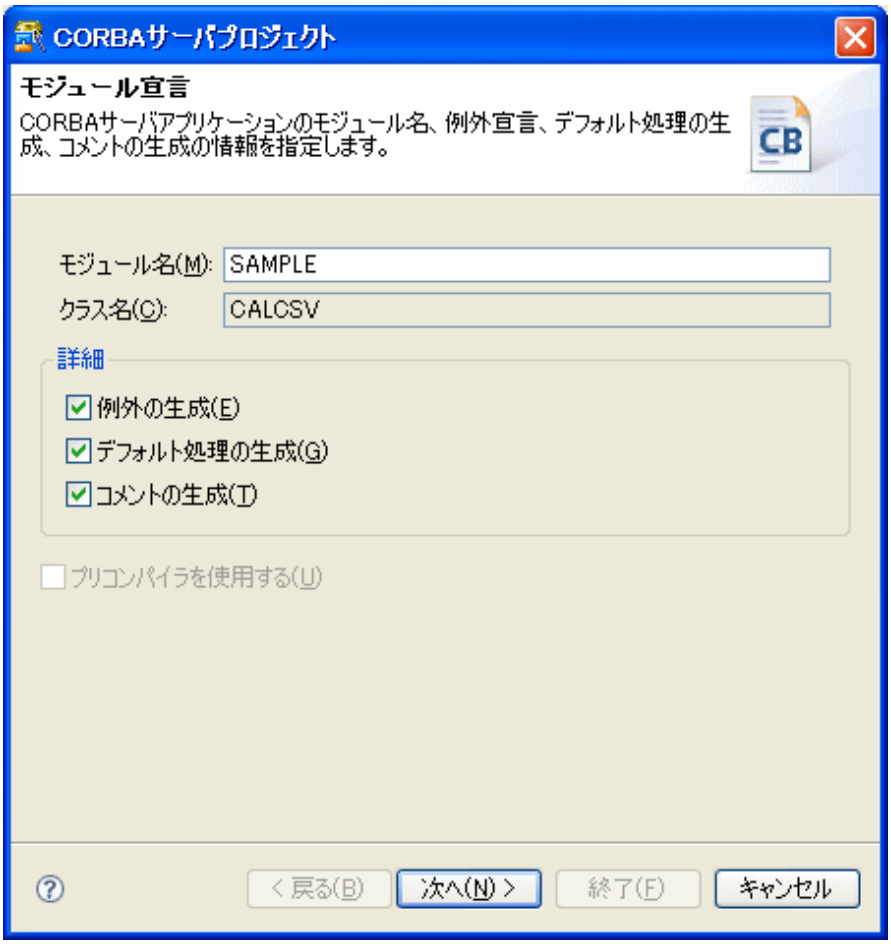


生成するコードを選択する画面です。コード生成ウィザードとして[CORBAサーバアプリケーション]が選択されていることを確認し、[終了]をクリックします。

4. CORBAサーバアプリケーション生成ウィザードによるひな型作成

プロジェクトが作成された後に、CORBAサーバアプリケーション生成ウィザードが起動されます。

[1枚目]



プロジェクト名と保存フォルダを入力します。

設定項目	設定内容
モジュール名	SAMPLE(任意)
クラス名	CALCSV(プロジェクト名固定)
例外の生成	チェックする
デフォルト処理の生成	チェックする
コメントの生成	チェックする

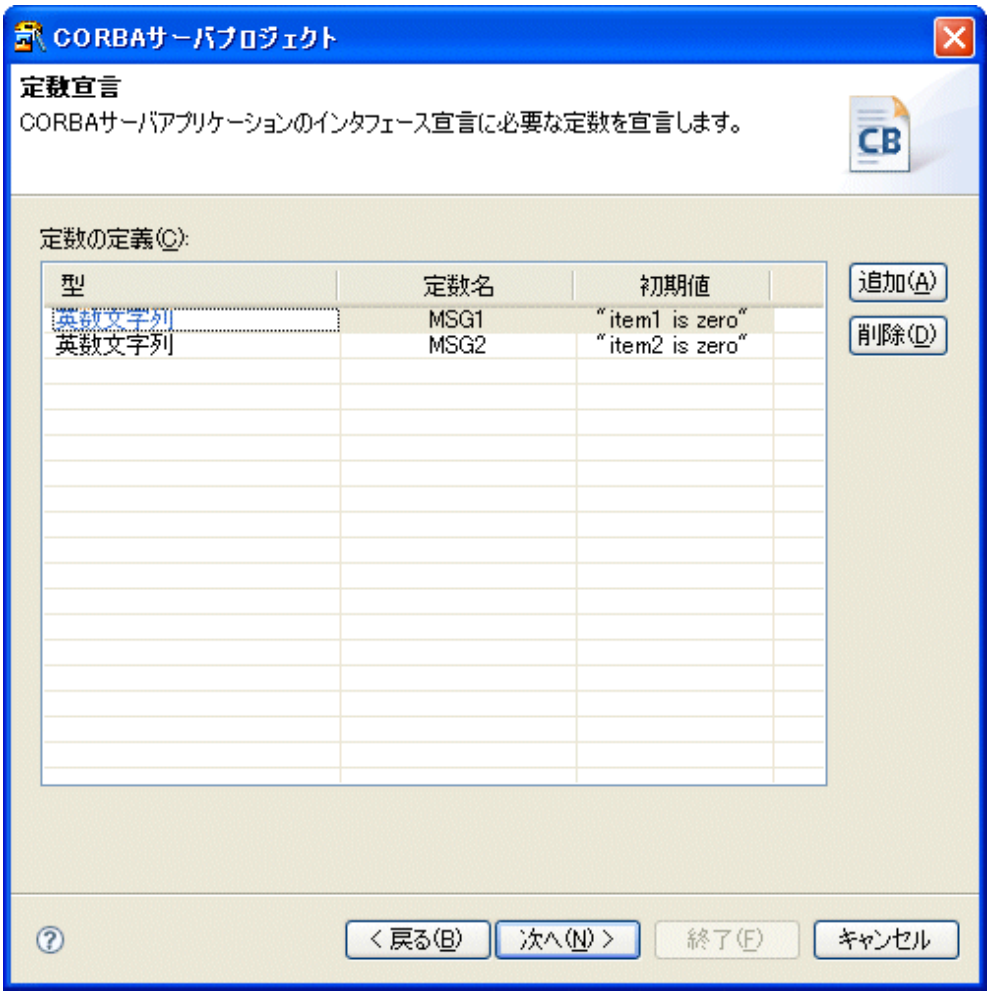
乗算、除算の例外情報のために[例外の生成]をチェックします。チェックした場合は、以下の例外宣言がIDL中に生成されます。

```
exception CDException{
    string CDExceptionMsg;
    long CDExceptionCode;
};
```

デフォルト処理の生成をチェックすると、サーバアプリケーションとして実装すべきメソッド定義と、そのメソッドの処理を生成します。チェックしない場合は、メソッドの宣言のみが生成されます。

コメントの生成をチェックした場合は、生成ひな型ソース中にコメントを生成します。

[2枚目]



2枚目では、定数を宣言します。ここでは、例外発生時に返却するエラーメッセージを2個定義します。

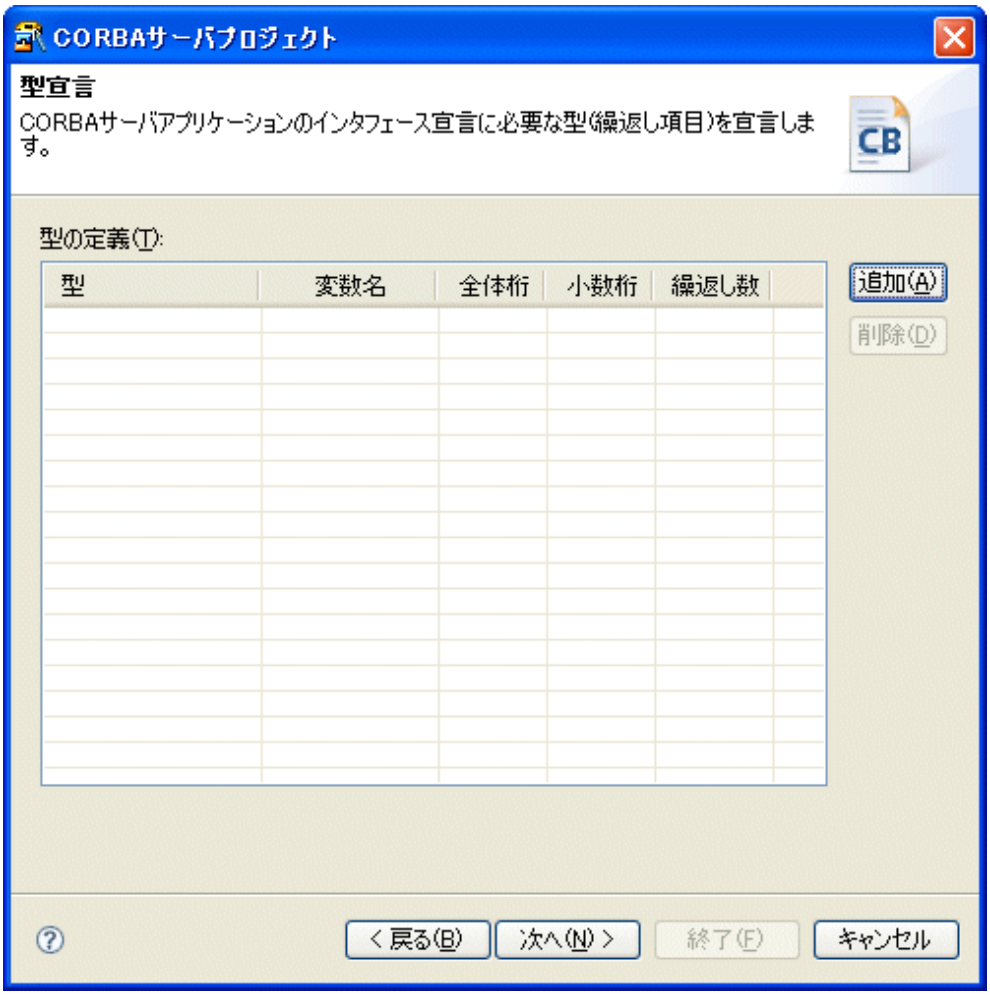
＜定数1＞

設定項目	設定内容
型	英数文字列
定数名	MSG1
初期値	"item1 is zero"

＜定数2＞

設定項目	設定内容
型	英数文字列
定数名	MSG2
初期値	"item2 is zero"

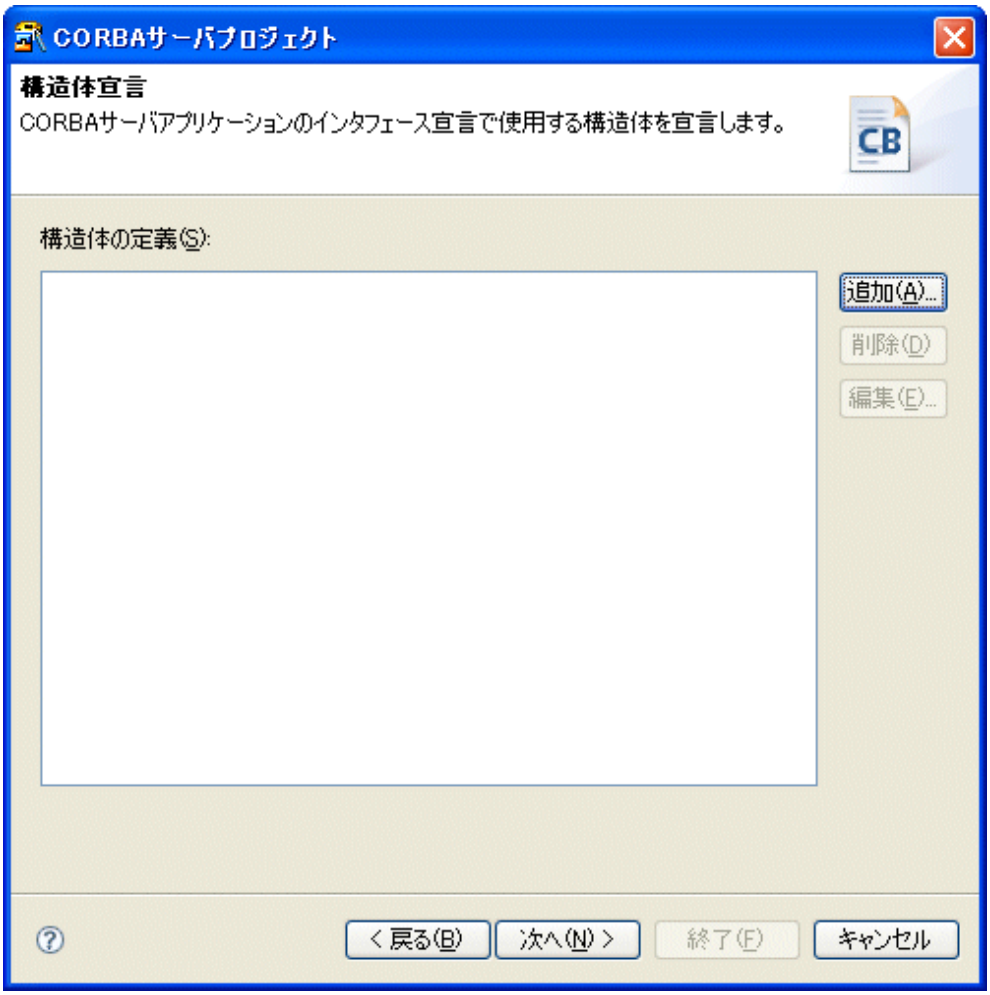
[3枚目]



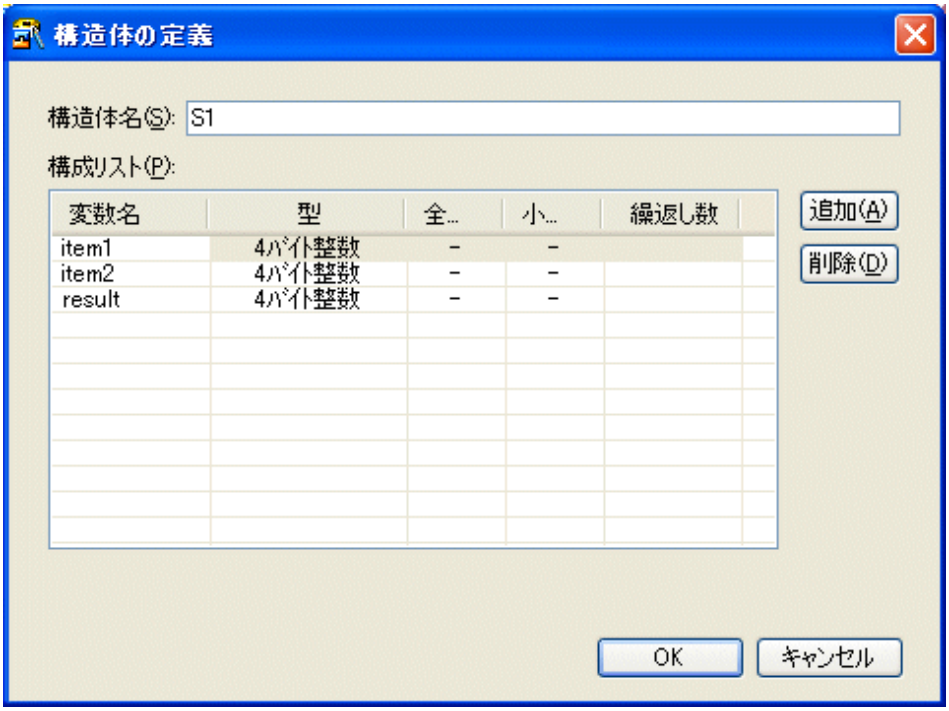
3枚目では型宣言を行います。今回は必要ありませんが、ここでは繰り返し項目などを定義します。
なにもせず、次に進んでください。

[4枚目]

ここで、構造体を宣言します。今回のプログラムでは、クライアントからの入力となる項目が2個、演算結果を返却する領域が1個必要であるため、それらの領域を1つの構造体として宣言することにします。



この画面で、[追加]ボタンを押して、構造体定義ダイアログで構造体を宣言します。

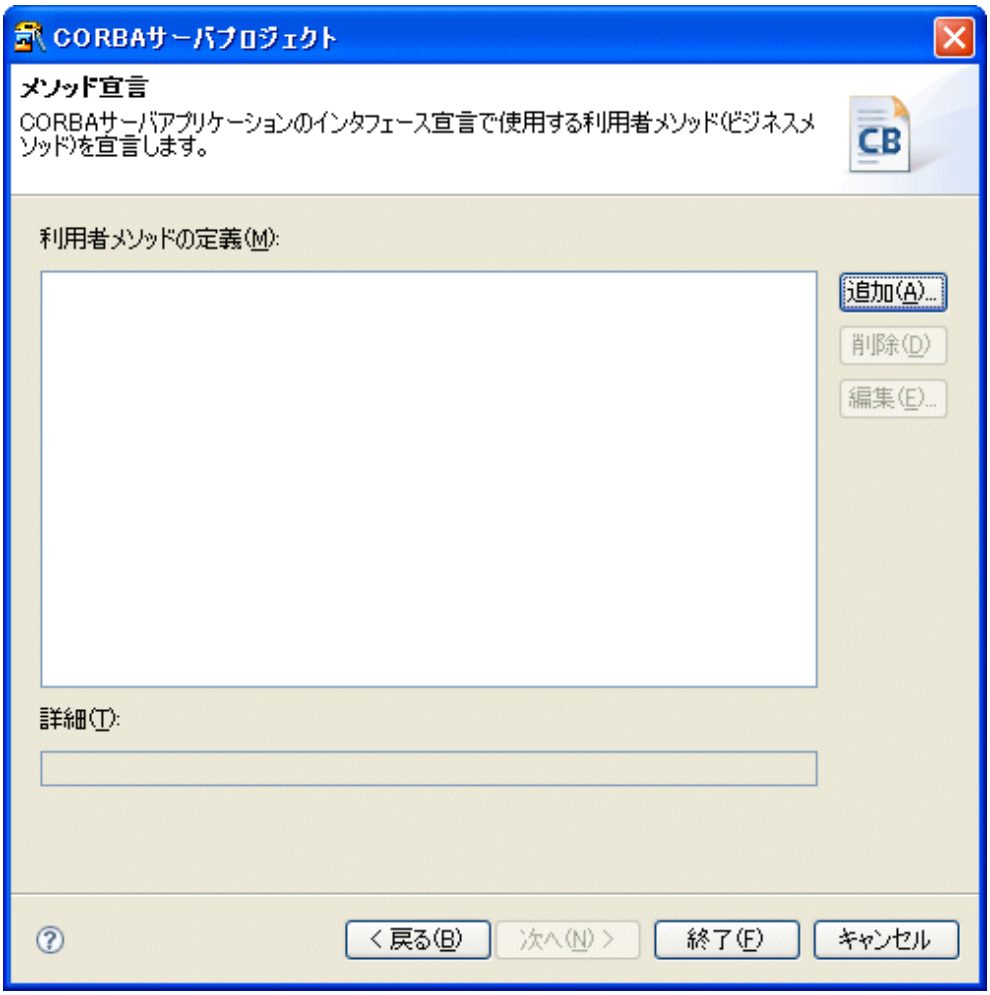


ここでは、構造体「S1」を以下の要素を持つ構造体として宣言します。

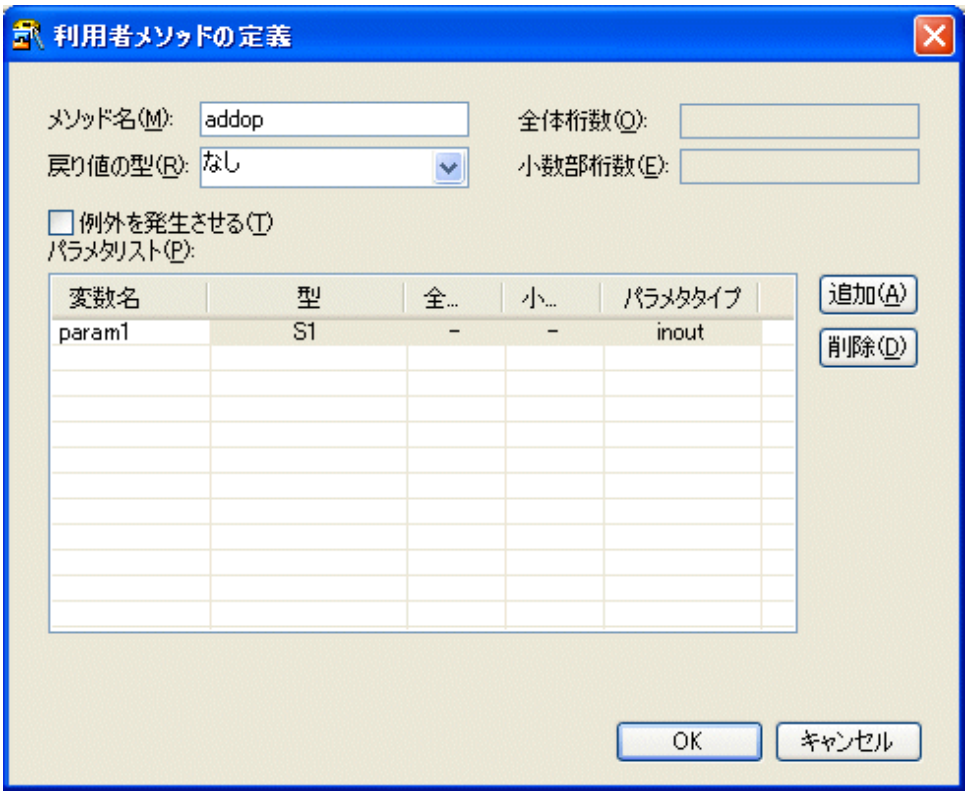
変数名	型と使用目的
item1	4バイト整数 , 2項演算のうちの1項目(クライアントから値設定)
item2	4バイト整数 , 2項演算のうちの1項目(クライアントから値設定)
result	4バイト整数 , 2項演算結果 (サーバが値設定)

[5枚目]

次に、ビジネスメソッドを宣言します。ここでは、「1. 作成するプログラムの内容」に従い、メソッド「addop」、「subop」、「mltop」、「divop」を宣言します。



この画面で、[追加]ボタンを押して、利用者メソッドの定義ダイアログでメソッドを宣言します。
下記のダイアログは、メソッド addopを設定している様子です。



各メソッドの定義情報を説明します。

＜メソッドaddop＞

設定項目	設定内容
メソッド名	addop
戻り値の型	なし(void)
例外を発生させる	チェックしない
パラメタリスト	変数名：param1 型：S1 (※1) パラメタタイプ：inout (※2)

＜メソッドsubop＞

設定項目	設定内容
メソッド名	subop
戻り値の型	なし(void)
例外を発生させる	チェックしない
パラメタリスト	変数名：param1 型：S1 パラメタタイプ：inout

＜メソッドmltop＞

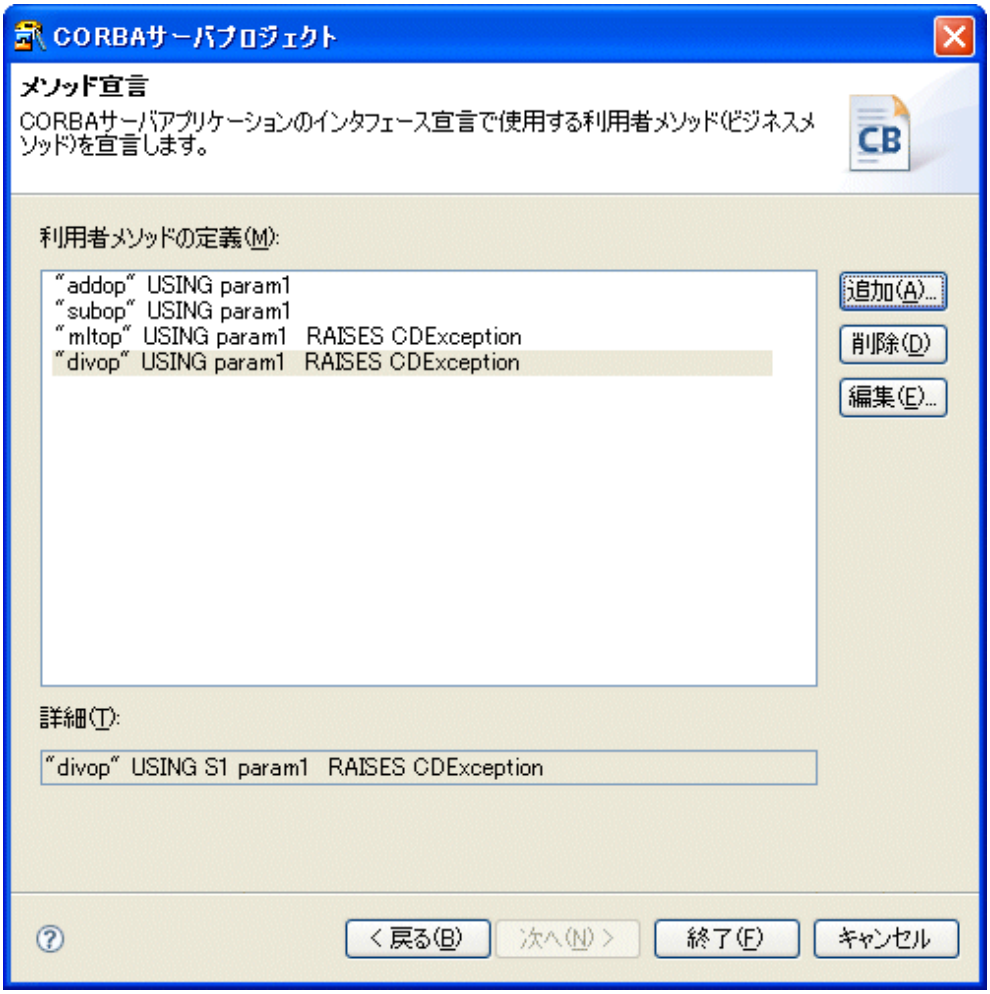
設定項目	設定内容
メソッド名	mltop
戻り値の型	なし(void)
例外を発生させる	チェックする (※3)

設定項目	設定内容
パラメタリスト	変数名：param1 型：S1 パラメタタイプ：inout

＜メソッドdivop＞

設定項目	設定内容
メソッド名	divop
戻り値の型	なし(void)
例外を発生させる	チェックする (※3)
パラメタリスト	変数名：param1 型：S1 パラメタタイプ：inout

(※1)「S1」は、構造体定義で宣言した構造体名です。
(※2) 構造体「S1」は、クライアント、サーバの両アプリケーションで値を設定するため、inoutにします。
(※3) 乗算、除算の場合は、例外を通知します。



[終了]ボタンを押すと、ソースファイルフォルダにひな型ソースが生成されます。この例の場合は、以下のファイルが生成されます。

ファイル名	説明
CALCSV.cob	メインソースプログラム(サーバアプリケーションのフレームワーク)

ファイル名	説明
SAMPLE-CALCSV-IMPL.cob	ビジネスメソッドプログラム
SAMPLE-CALCSV--INIT.cob	サーバアプリケーション登録プログラム
USCALCSV.idl	インタフェースファイル(IDLファイル)
"SAMPLE"で始まるCOBOLソースファイル	IDLコンパイラが生成したスケルトン用ファイル

5. プログラムの編集

IDLファイル (USCALCSV.idl)

```
// モジュール宣言
module SAMPLE {

  // 定数宣言
  const string MSG1 = "item1 is zero" ;
  const string MSG2 = "item2 is zero" ;

  // 型宣言

  // 構造体宣言
  struct S1 {
    long item1;
    long item2;
    long result;
  };

  // 例外宣言
  exception CDException{
    string CDExceptionMsg;
    long CDExceptionCode;
  };
  // ユーザインタフェース宣言
  interface CALCSV {
    void addop(inout S1 param1);
    void subop(inout S1 param1);
    void mltop(inout S1 param1)
      raises (CDException);
    void divop(inout S1 param1)
      raises (CDException);
  };

};
```

今回は、特に修正の必要がありませんが、ここで、IDLファイルの内容を編集することも可能です。
※インタフェースの内容を変更した場合は、必ずビジネスメソッドのプログラムソースと等価になるように両方を修正する必要があります。

メインソースプログラム (CALCSV.cob)

ひな型は、選択されたアプリケーション形態に対して標準的なものを生成しているため、カスタマイズが必要な場合は修正してください。今回は、特に修正の必要がありません。

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. CALCSV.
000030 ENVIRONMENT DIVISION.
000040 CONFIGURATION SECTION.
000050 REPOSITORY.
000060       COPY CORBA--REP.
000070       COPY USCALCSV--REP.
000080 .
```

```
000090 SPECIAL-NAMES.
000100      SYMBOLIC CONSTANT
000110      COPY CORBA--CONST.
000120  .
000130 DATA DIVISION.
000140 WORKING-STORAGE SECTION.
000150 COPY CORBA--COPY.
000160 COPY USCALCSV--COPY.
000170 01  API-NAME        PIC   X(50).
000180 01  APL-NAME        PIC   X(64)   VALUE     "CALCSV".
000190 01  ORB             USAGE OBJECT REFERENCE CORBA-ORB.
000200 01  BOA             USAGE OBJECT REFERENCE CORBA-BOA.
000210 01  IMPL-REP        USAGE OBJECT REFERENCE FJ-IMPLEMENTATIONREP.
000220 01  IMPL            USAGE OBJECT REFERENCE CORBA-IMPLEMENTATIONDEF.
000230 01  REP-ID          PIC   X(128) VALUE     "IDL:SAMPLE/CALCSV:1.0".(※1)
000240 01  OBJ             USAGE OBJECT REFERENCE CORBA-OBJECT.
000250 01  EXCEPT-ID       USAGE OBJECT REFERENCE CORBA-STRING.
000260 01  EXCEPT-ID-VALUE PIC   X(50).
000270 LINKAGE SECTION.
000280 PROCEDURE DIVISION
000290                   .
000300*
000310*  ORBの初期化
000320*
000330      INVOKE CORBA "ORB_INIT" USING APL-NAME FJ-OM_ORBID RETURNING ORB.
000340*
000350*  BOAの初期化
000360*
000370      INVOKE ORB "BOA_INIT" USING APL-NAME CORBA-BOA_OAID RETURNING BOA.
000380*
000390*  インプリメンテーションリポジトリオブジェクトの取得
000400*
000410      INVOKE ORB "RESOLVE_INITIAL_REFERENCES" USING CORBA-OBJECTID_IMPLEMENTAT-001 RETURNING OBJ.
000420      INVOKE FJ-IMPLEMENTATIONREP "NARROW" USING OBJ RETURNING IMPL-REP.
000430*
000440*  インプリメンテーション情報の取得
000450*
000460      INVOKE IMPL-REP "LOOKUP_ID" USING REP-ID RETURNING OBJ.
000470      INVOKE CORBA-IMPLEMENTATIONDEF "NARROW" USING OBJ RETURNING IMPL.
000480*
000490      SET OBJ TO NULL.
000500*
000510*  サーバの活性化をODに通知する
000520*
000530      MOVE "CORBA::BOA::IMPL_IS_READY" TO API-NAME.
000540      INVOKE BOA "IMPL_IS_READY" USING IMPL.
000550*
000560      STOP RUN.
000570 END PROGRAM CALCSV.
```

(※1) リポジトリID
CORBAサーバアプリケーションを一意に決定するためのIDです。デフォルトは、「IDL:モジュール名/インタフェース名:1.0」です。
詳細は、Interstage Application Serverのマニュアルを参照してください。

サーバアプリケーション登録プログラム (SAMPLE-CALCSV--INIT.cob)

このプログラムは、サーバアプリケーションがCORBAサーバとして使用できるように登録する処理です。特に修正の必要はありません。

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. SAMPLE-CALCSV--INIT.
000030 ENVIRONMENT DIVISION.
000040 CONFIGURATION SECTION.
000050 REPOSITORY.
```

```
000060      CLASS SAMPLE-CALCSV
000070      CLASS SAMPLE-CALCSV-IMPL
000080 .
000090 DATA DIVISION.
000100 WORKING-STORAGE SECTION.
000110 LINKAGE SECTION.
000120 01 RET USAGE OBJECT REFERENCE SAMPLE-CALCSV.
000130 PROCEDURE DIVISION
000140         RETURNING RET
000150                   .
000160      INVOKE SAMPLE-CALCSV-impl "new" RETURNING RET.
000170      EXIT PROGRAM.
000180 END PROGRAM SAMPLE-CALCSV-- INIT.
```

ビジネスメソッドの実装 (SAMPLE-CALCSV-IMPL.cob)

ウィザードで入力したメソッド「addop」、「subop」、「mltop」、「divop」の宣言がされているので、処理を実装します。**太字(**青色)の部分が追加したコードです。

```
000010 CLASS-ID. SAMPLE-CALCSV-IMPL AS "SAMPLE-CALCSV-IMPL" INHERITS
000020     SAMPLE-CALCSV
000030  .
000040 ENVIRONMENT DIVISION.
000050 CONFIGURATION SECTION.
000060 REPOSITORY.
000070      COPY CORBA--REP.
000080      COPY USCALCSV--REP.
000090  .
000100 SPECIAL-NAMES.
000110      SYMBOLIC CONSTANT
000120      COPY CORBA--CONST.
000121      COPY USCALCSV--CONST.
000130  .
000140 OBJECT.
000150 DATA DIVISION.
000160 WORKING-STORAGE SECTION.
000170 COPY CORBA--COPY.
000180 COPY USCALCSV--COPY.
000190 PROCEDURE DIVISION
000200                   .
000210 METHOD-ID. ADDOP AS "ADDOP" OVERRIDE.
000220* <IDL-INFO-START>
000230* void addop(inout S1 param1)
000240* <IDL-INFO-END>
000250 DATA DIVISION.
000260 WORKING-STORAGE SECTION.
000270 LINKAGE SECTION.
000280 01 PARAM1  TYPE SAMPLE-S1.
000290 PROCEDURE DIVISION
000300         USING
000310                   PARAM1
000320                   .
000321      COMPUTE result OF param1 = item1 OF param1 + item2 OF param1.
000330 END METHOD ADDOP.
000340 METHOD-ID. SUBOP AS "SUBOP" OVERRIDE.
000350* <IDL-INFO-START>
000360* void subop(inout S1 param1)
000370* <IDL-INFO-END>
000380 DATA DIVISION.
000390 WORKING-STORAGE SECTION.
000400 LINKAGE SECTION.
000410 01 PARAM1  TYPE SAMPLE-S1.
000420 PROCEDURE DIVISION
000430         USING
```

```
000440                  PARAM1
000450                  .
000451      COMPUTE result OF param1 = item1 OF param1 - item2 OF param1.
000460 END METHOD SUBOP.
000470 METHOD-ID. MLTOP AS "MLTOP" OVERRIDE.
000480* <IDL-INFO-START>
000490* void mltop(inout S1 param1)
000500* <IDL-INFO-END>
000510 DATA DIVISION.
000520 WORKING-STORAGE SECTION.
000521 01 W-EXCEPTION OBJECT REFERENCE SAMPLE-CDException.
000522 01 W-STRING OBJECT REFERENCE CORBA-STRING.
000530 LINKAGE SECTION.
000540 01 PARAM1  TYPE SAMPLE-S1.
000550 PROCEDURE DIVISION
000560         USING
000570                  PARAM1
000580         RAISING
000590                  SAMPLE-CDEXCEPTION
000600                  .
000601* 例外処理
000602       IF item1 OF param1 = 0
000603       THEN
000604          INVOKE SAMPLE-CDException "NEW" RETURNING W-EXCEPTION
000605          MOVE -1 TO CDEXCEPTIONCODE OF W-EXCEPTION
000606          INVOKE CORBA-STRING "NEW" RETURNING W-STRING
000607          INVOKE W-STRING "SET-VALUE" USING SAMPLE-MSG1
000608          SET CDEXCEPTIONMSG OF W-EXCEPTION TO W-STRING
000609          EXIT METHOD RAISING W-EXCEPTION
000610       END-IF
000611       IF item2 OF param1 = 0
000612       THEN
000613          INVOKE SAMPLE-CDException "NEW" RETURNING W-EXCEPTION
000614          MOVE -1 TO CDEXCEPTIONCODE OF W-EXCEPTION
000615          INVOKE CORBA-STRING "NEW" RETURNING W-STRING
000616          INVOKE W-STRING "SET-VALUE" USING SAMPLE-MSG2
000617          SET CDEXCEPTIONMSG OF W-EXCEPTION TO W-STRING
000618          EXIT METHOD RAISING W-EXCEPTION
000619       END-IF
000620       COMPUTE result OF param1 = item1 OF param1 * item2 OF param1.
000621
000622 END METHOD MLTOP.
000623 METHOD-ID. DIVOP AS "DIVOP" OVERRIDE.
000630* <IDL-INFO-START>
000640* void divop(inout S1 param1)
000650* <IDL-INFO-END>
000660 DATA DIVISION.
000670 WORKING-STORAGE SECTION.
000671 01 W-EXCEPTION OBJECT REFERENCE SAMPLE-CDException.
000672 01 W-STRING OBJECT REFERENCE CORBA-STRING.
000680 LINKAGE SECTION.
000690 01 PARAM1  TYPE SAMPLE-S1.
000700 PROCEDURE DIVISION
000710         USING
000720                  PARAM1
000730         RAISING
000740                  SAMPLE-CDEXCEPTION
000750                  .
000751* 例外処理
000752       IF item1 OF param1 = 0
000753       THEN
000754          INVOKE SAMPLE-CDException "NEW" RETURNING W-EXCEPTION
000755          MOVE -1 TO CDEXCEPTIONCODE OF W-EXCEPTION
```

```
000756          INVOKE CORBA-STRING "NEW" RETURNING W-STRING
000757          INVOKE W-STRING "SET-VALUE" USING SAMPLE-MSG1
000758          SET CDEXCEPTIONMSG OF W-EXCEPTION TO W-STRING
000759          EXIT METHOD RAISING W-EXCEPTION
000760       END-IF
000761       IF item2 OF param1 = 0
000762       THEN
000763          INVOKE SAMPLE-CDException "NEW" RETURNING W-EXCEPTION
000764          MOVE -1 TO CDEXCEPTIONCODE OF W-EXCEPTION
000765          INVOKE CORBA-STRING "NEW" RETURNING W-STRING
000766          INVOKE W-STRING "SET-VALUE" USING SAMPLE-MSG2
000767          SET CDEXCEPTIONMSG OF W-EXCEPTION TO W-STRING
000768          EXIT METHOD RAISING W-EXCEPTION
000769       END-IF
000770       COMPUTE result OF param1 = item1 OF param1 / item2 OF param1.
000771
000772 END METHOD DIVOP.
000773 END OBJECT.
000780 END CLASS SAMPLE-CALCSV-IMPL.
```

ポイント

- 構造体「S1」は、固定長であるため、TYPE宣言されます。
- 構造体「S1」の要素は集団項目として定義されているため、集団項目のように使用できます。
- 例外を通知する場合は、例外クラスのインスタンスを生成(new)し、メンバに値を設定します。

6. プロジェクトのビルド

SAMPLE-CALCSV-IMPL.cobの編集画面のコンテキストメニューから[保存]を選択してください。CORBAサーバアプリケーションが自動的にビルドされます。
ただし、メニューの[プロジェクト] > [自動的にビルド]のチェックを解除している場合はビルドがされないため、チェックしなおしてください。チェックをするとにCORBAサーバアプリケーションが自動的にビルドされます。

7. プログラムの実行

CORBAサーバアプリケーションはワークユニット上で実行します。以下に、CORBAサーバアプリケーションの動作確認の手順の概要を示します。

ポイント

MyCORBADebugについて

本チュートリアルでは、MyCORBADebugという名前のワークユニットを使用します。MyCORBADebugワークユニットを作成する方法は、"B.1 CORBAワークユニット(MyCORBADebug)を作成するには"を参照してください。

1. 実行資産のコピー

   ビルドで生成された実行ファイルとダイナミックリンクライブラリを実行環境にコピーします。
   ここでは、以下のようにコピーします。

   ```
   ファイル：CALCSV.exe、USCALCSV.dll
   コピー先フォルダ：C:¥Interstage¥APS¥var¥CORBA_WU¥MyCORBADebug
   ```

   任意のフォルダにコピー可能ですが、このチュートリアルでは、MyCORBADebugのデフォルトのアプリケーション格納フォルダにコピーします。

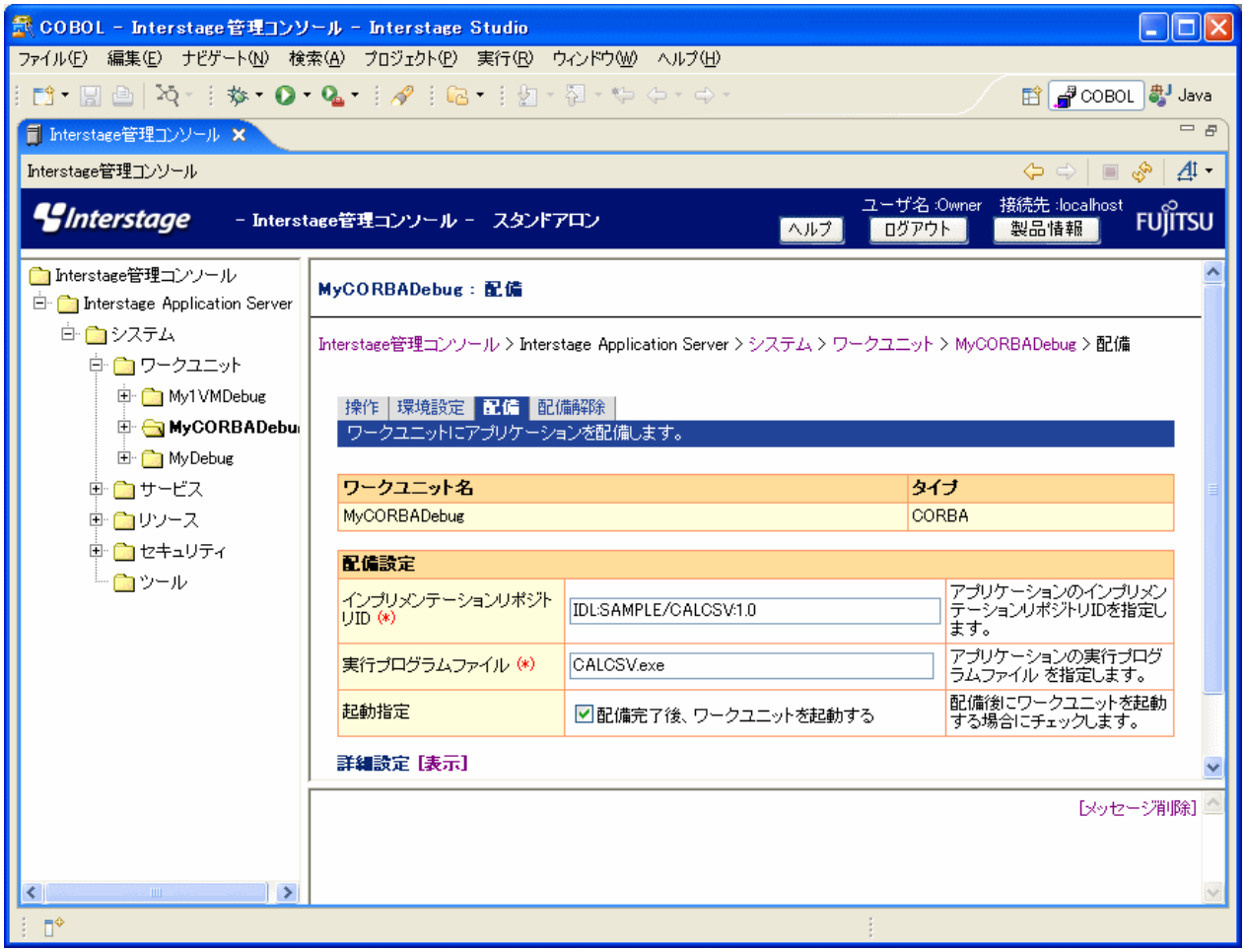
2. CORBAサーバアプリケーションの配備

IJServerビューで右ボタンをクリックし、[Interstage管理コンソール]を選択します。Interstage Application Serverに未接続状態の場合には、Interstage Application Serverのコンテキストメニューから[接続/ログイン]を選択して、Interstage Application Serverに接続あるいはログインしてから[Interstage管理コンソール]を選択してください。
Interstage管理コンソールの左側のツリーから配備先のワークユニットを選択し、画面右側の[配備]タブを選択します。
インプリメンテーションリポジトリIDと実行プログラムファイルを指定します。

3. アプリケーション動作カレントディレクトリの作成

ワークユニットの起動で必要となるアプリケーション動作カレントディレクトリを作成します。
このチュートリアルでは、ワークユニット作成時にデフォルト値を使用しているため、以下のディレクトリを用意しておく必要があります。

```
C:¥Interstage¥APS¥var¥CORBA_WU¥MyCORBADebug¥work
```

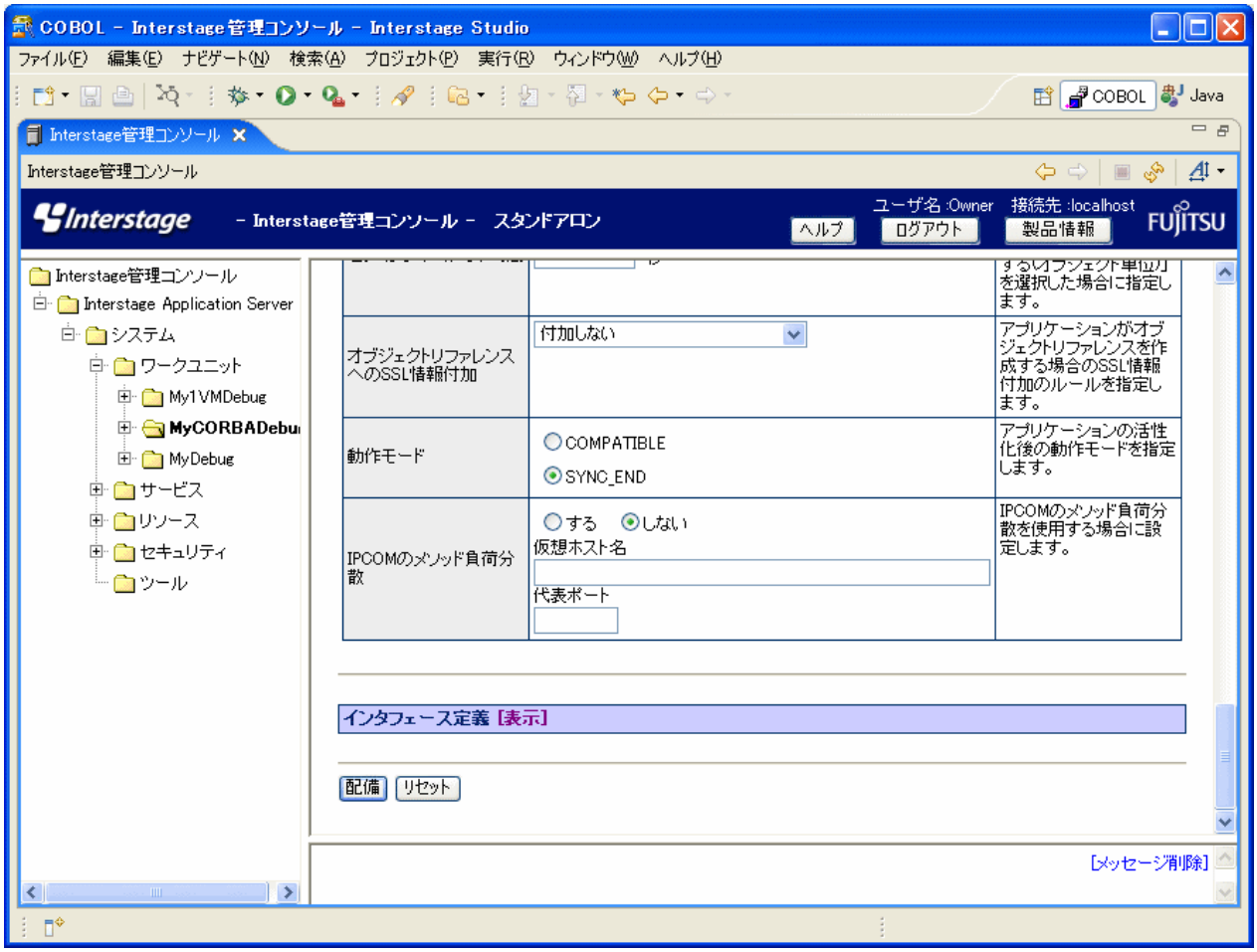


インプリメンテーションリポジトリIDは、以下の形式で指定します。

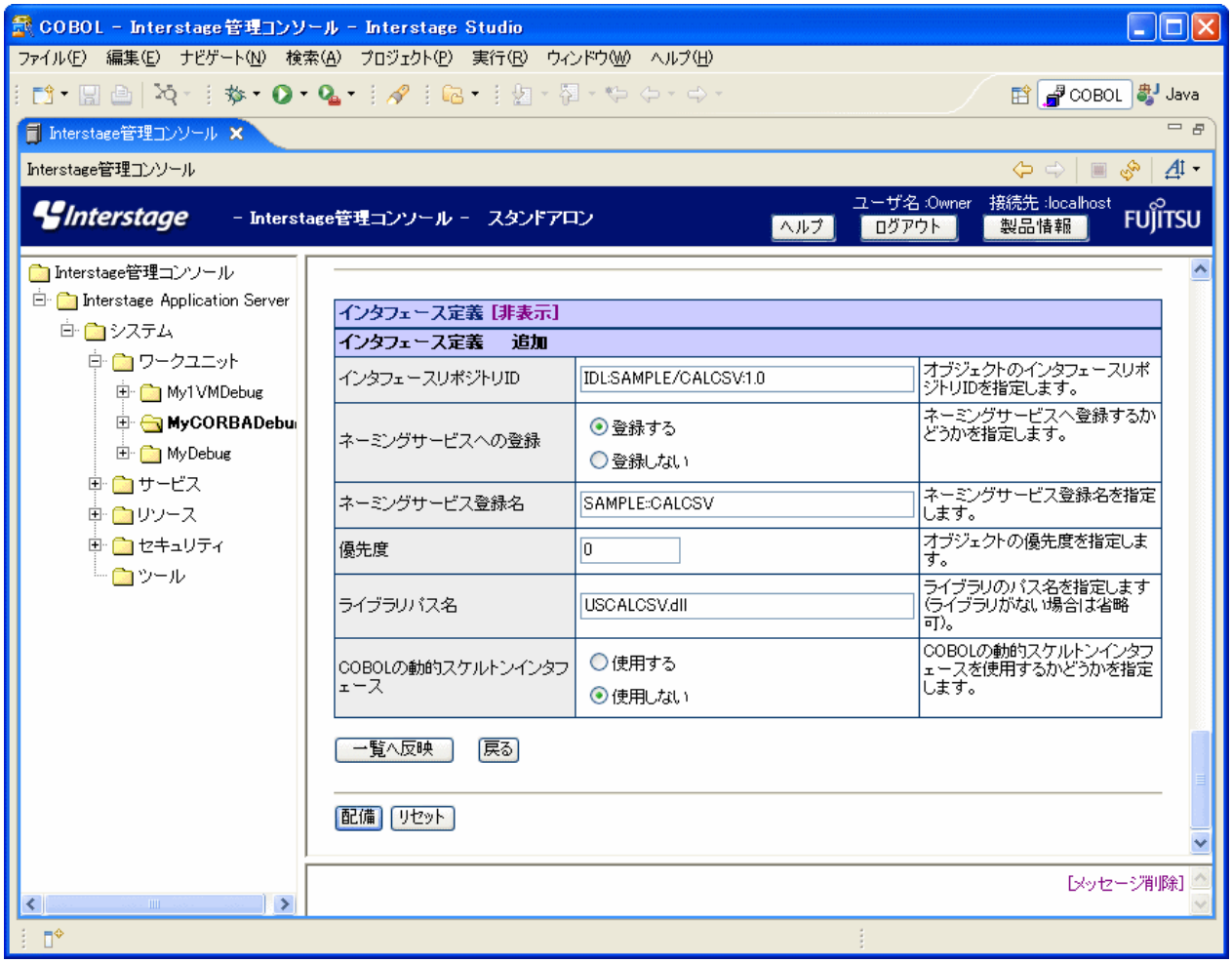
```
IDL:モジュール名/インタフェース名:1.0
```

実行ファイル名は、最初にコピーしたEXEファイルを指定します。

詳細設定の[表示]を選択し、CORBAアプリケーションの[表示]を選択します。表示された詳細設定画面の[動作モード]で、[SYNC_END]を選択します。



インタフェース定義の[表示]を選択します。[追加]ボタンを押し、以下のようにインタフェースリポジトリID、ネーミングサービス登録名、ライブラリ名を指定します。

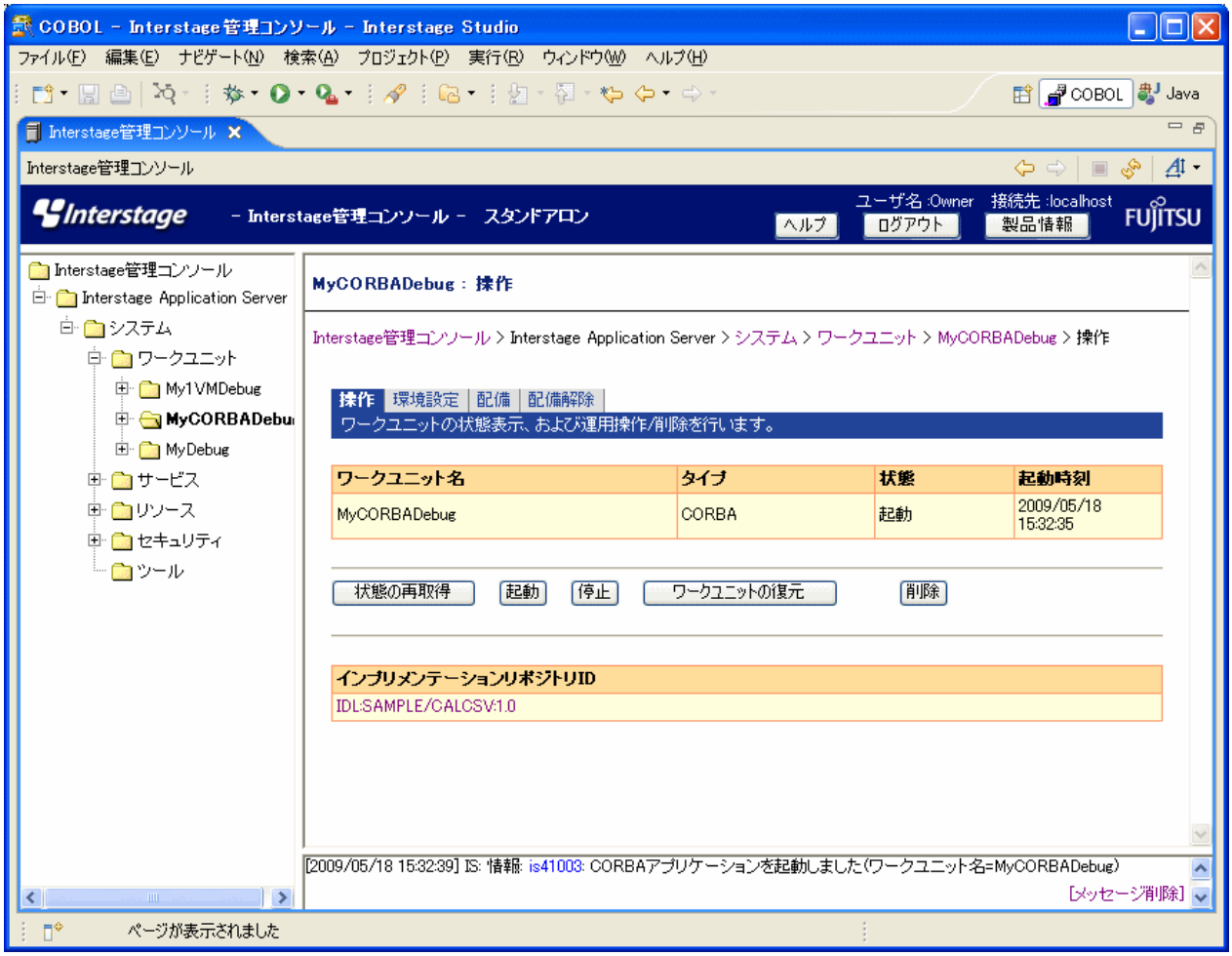


[一覧へ反映]ボタンを押した後、[配備]ボタンを押して、配備を行います。

4. ワークユニットの起動

配備時に起動指定で、[配備完了後、ワークユニットを起動する]をチェックしていない場合は、以下のようにして作成したワークユニットを起動します。

Interstage管理コンソールの左側のツリーからワークユニットを選択し、画面右側の[操作]タブを選択します。[起動]ボタンを押します。



アプリケーション動作カレントディレクトリが存在しない場合、起動時にエラーが発生します。アプリケーション動作カレントディレクトリは、ワークユニットの[環境設定]タブで確認できます。

5. クライアントアプリケーションの実行

   クライアントアプリケーションを実行し、CORBAサーバアプリケーションにアクセスします。
   クライアントアプリケーションの作成、実行方法の詳細については、"第14章 CORBAクライアントアプリケーション開発"を参照してください。

8. デバッグ

ワークベンチではCORBAワークユニット起動構成を使用することで、ワークベンチと同一端末にインストールされているInterstage Application Serverのワークユニット上で動作するCORBAサーバアプリケーションをデバッグできます。デバッグの手順の詳細については、"6.2 プロジェクトをデバッグする"を参照してください。

第14章　CORBAクライアントアプリケーション開発

ここでは、CORBAサーバアプリケーションで説明した2項演算プログラムを呼び出すクライアントアプリケーションの作成手順について説明します。

1. プロジェクトの作成

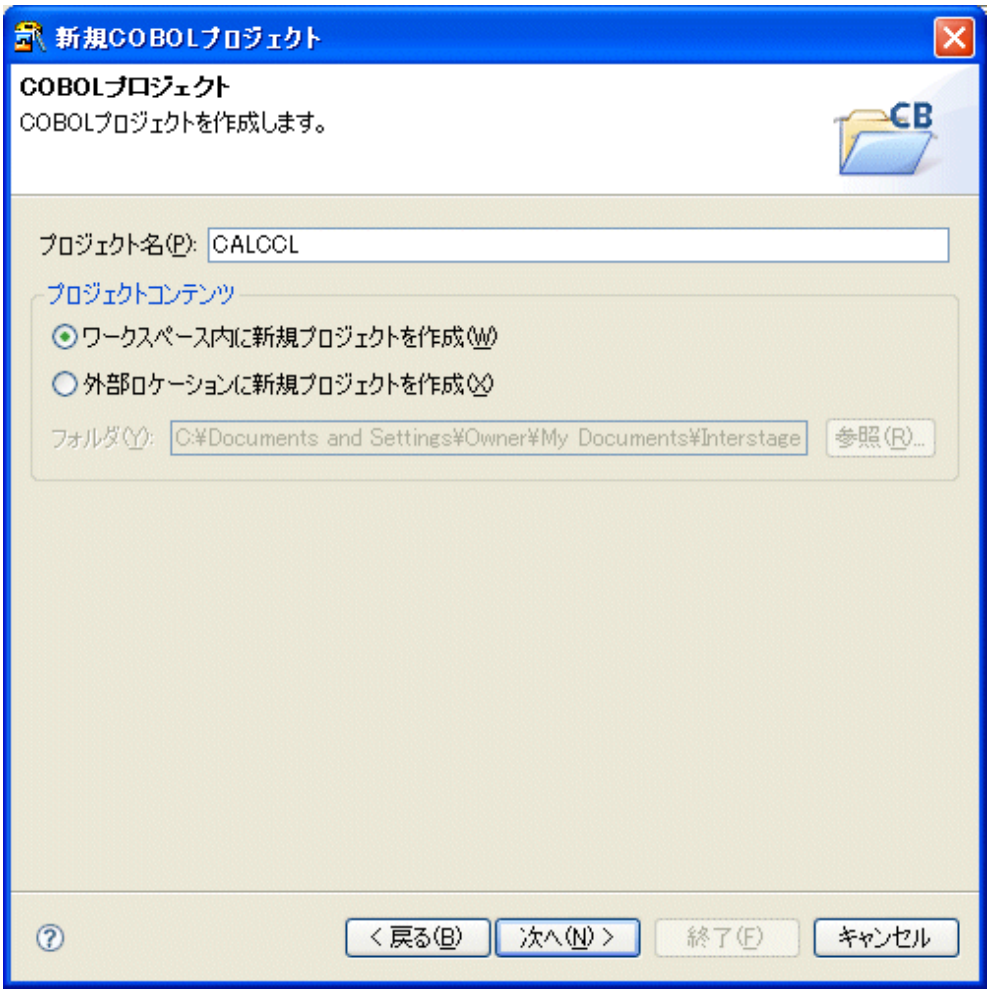
ここでは、オブジェクト指向COBOL言語を使用してCORBAアプリケーションを呼び出す方法を説明します。オブジェクト指向COBOLのプログラミング方法の詳細は、Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)を参照してください。

COBOLパースペクティブを表示していない場合は、以下の手順でCOBOLパースペクティブを表示してください。

1. メニューバーから[ウィンドウ] > [パースペクティブを開く] > [その他]を選択すると[パースペクティブを開く]ダイアログボックスが表示されます。
2. [COBOL]を選択して[OK]ボタンをクリックします。

ワークベンチ のメニューバーから[ファイル] > [新規] > [COBOLプロジェクト]を選択すると、新規COBOLプロジェクトが表示されます。

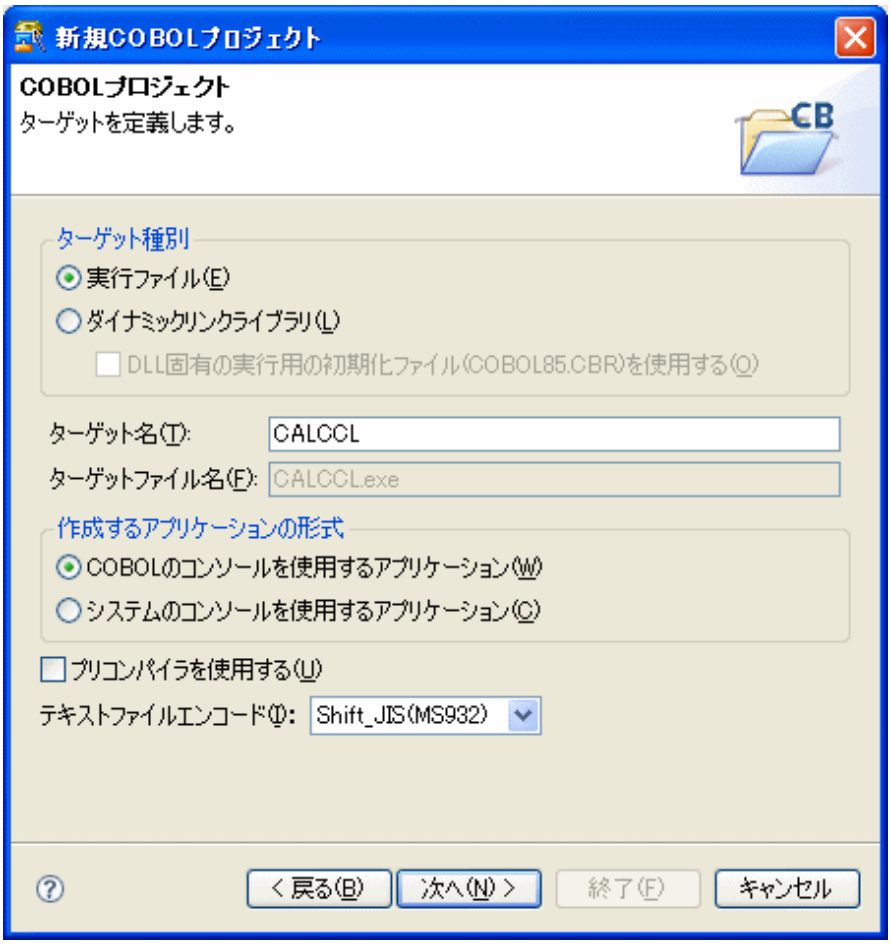
[1枚目]



プロジェクト名と保存フォルダを入力します。

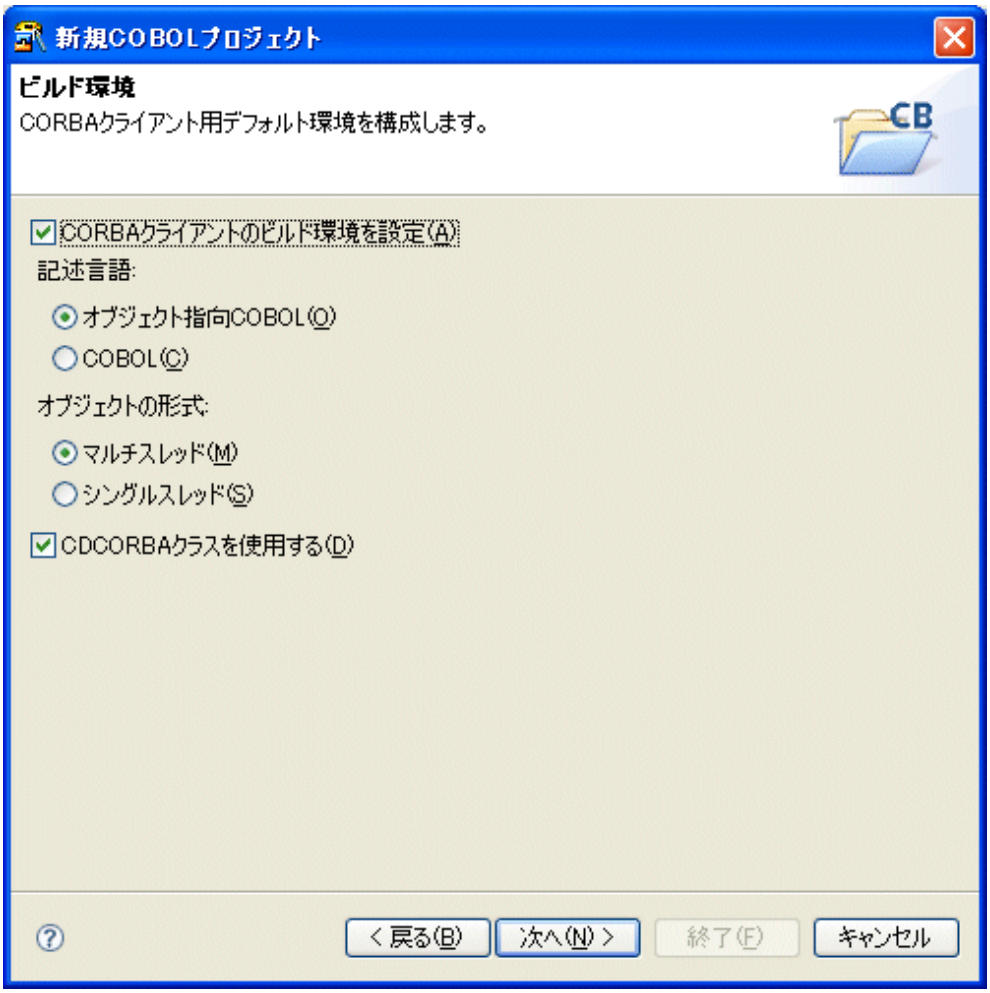
設定項目	設定内容
プロジェクト名	CALCCL
プロジェクトコンテンツ	[ワークスペース内に新規プロジェクトを作成]をチェック

[2枚目]



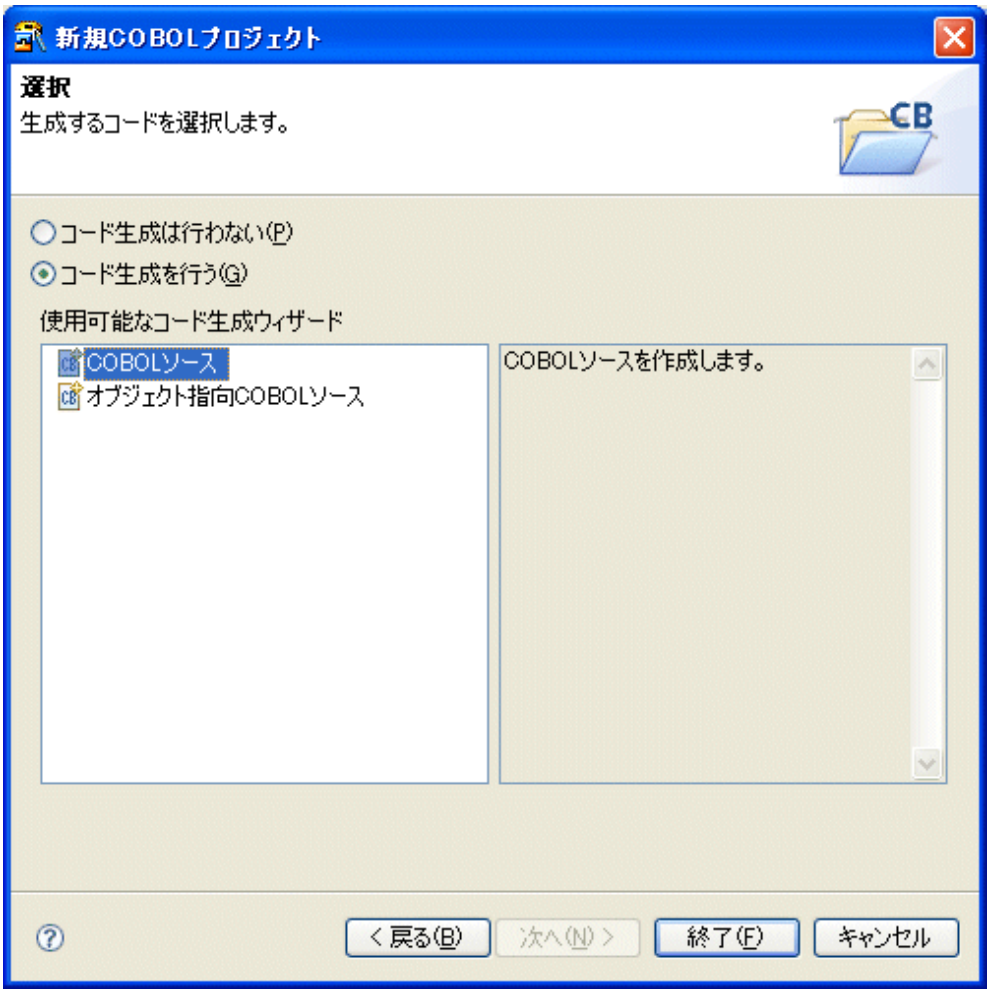
ターゲットに関する情報を指定する画面です。ここでは何も指定せずに次に進みます。

[3枚目]



ビルド環境を指定する画面です。[CORBAクライアントのビルド環境を設定]をチェックします。それ以外についてはデフォルトの指定のままとします。

[4枚目]

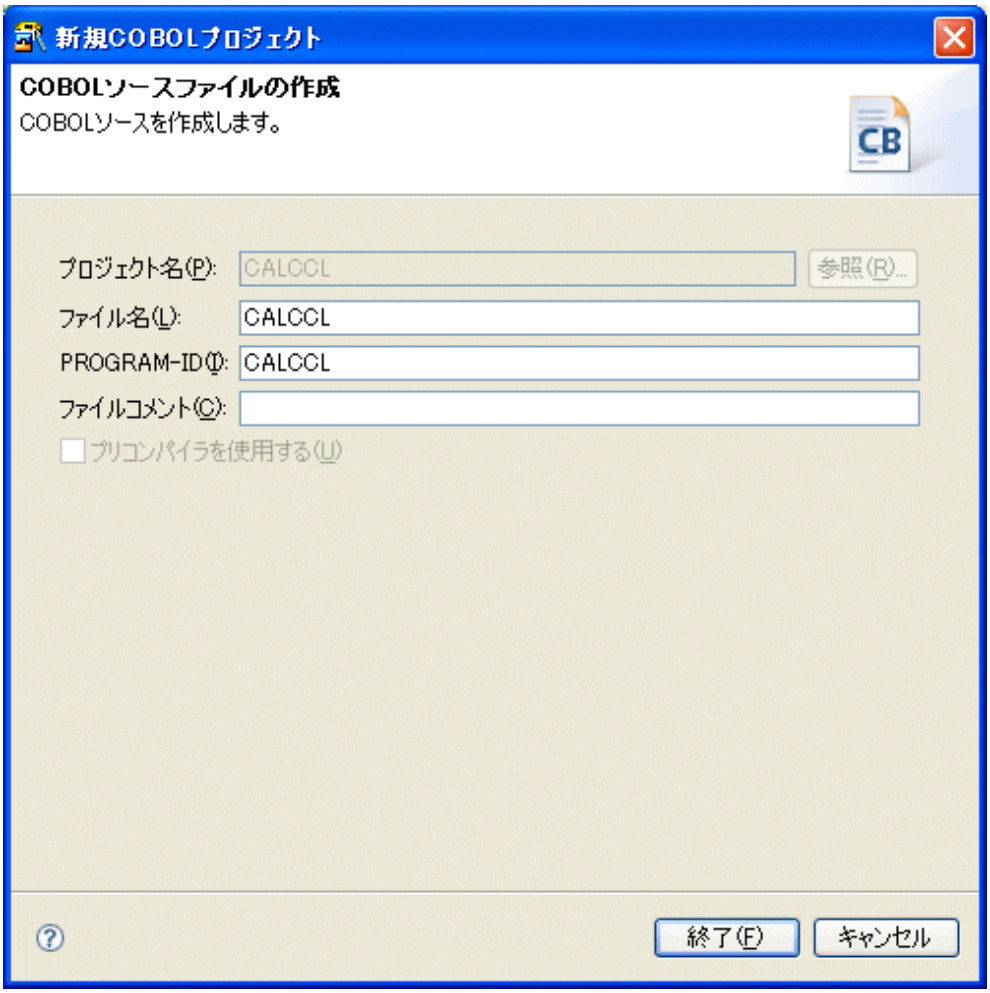


生成するコードを選択する画面です。コード生成ウィザードとして[COBOLソース]が選択されることを確認し、[終了]をクリックします。

2. COBOLソース生成ウィザードによるひな型作成

生成するコードを選択する画面です。コード生成ウィザードとして[COBOLソース]が選択されることを確認し、[終了]をクリックします。

[1枚目]



ここではCOBOLソースの情報を以下のように指定します。

設定項目	設定内容
プロジェクト名	CALCCL
ファイル名	CALCCL
PROGRAM-ID	CALCCL
ファイルコメント	(任意のコメントを記載してください)

[終了]ボタンを押すと、"CALCCL.cob"ファイルが作成されます。

3. Interstage基盤サービスの起動

テンプレートからCORBAサーバオブジェクト一覧を取得する場合や、CORBAスタブファイル生成ウィザードによってスタブファイルを作成する場合には、Interstage基盤サービス操作ツールによってJ2EE実行環境のためのサービスをあらかじめ起動しておく必要があります。デフォルトの状態では必要なサービスは起動されていないため、以下の手順でサービスを起動してください。

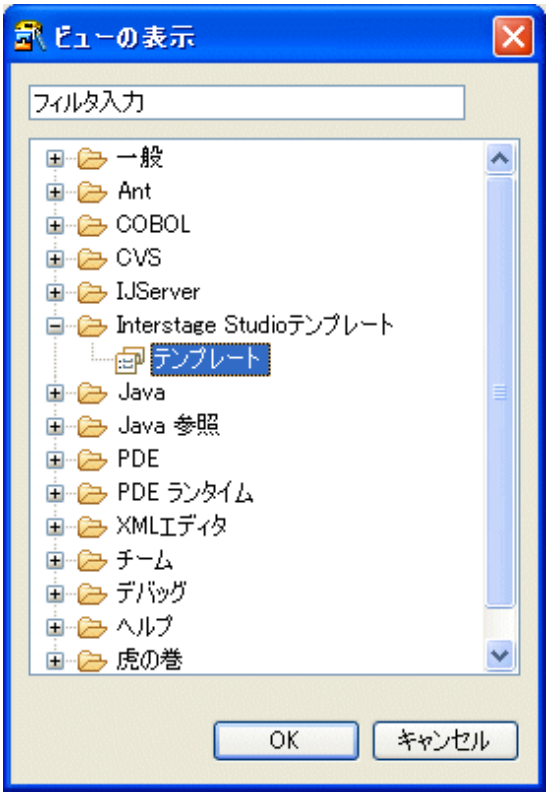
1. スタートメニューから[Interstage] > [Studio] > [Interstage基盤サービス操作ツール]を選択します。
2. Interstage基盤サービス操作ツールのウィンドウの[J2EE実行環境を使用する]チェックボックスをチェック状態にします。
3. [必須サービスのみ起動状態にする]ボタンが有効の場合、ボタンをクリックしてサービスを起動します。ボタンが無効の場合は操作は不要です。
4. <必要なサービス>が全て実行中となっていることを確認してから[閉じる]ボタンをクリックしてInterstage基盤サービス操作ツールを終了します。

4. プログラムの編集

ウィザードで生成されたCOBOLソースを修正します。

テンプレートビューを表示

ワークベンチ のメニューより[ウィンドウ] > [ビューの表示] > [その他]を選択すると[ビューの表示]が開きます。
[Interstage Studioテンプレート] > [テンプレート]を選択して[OK]ボタンを押してください。

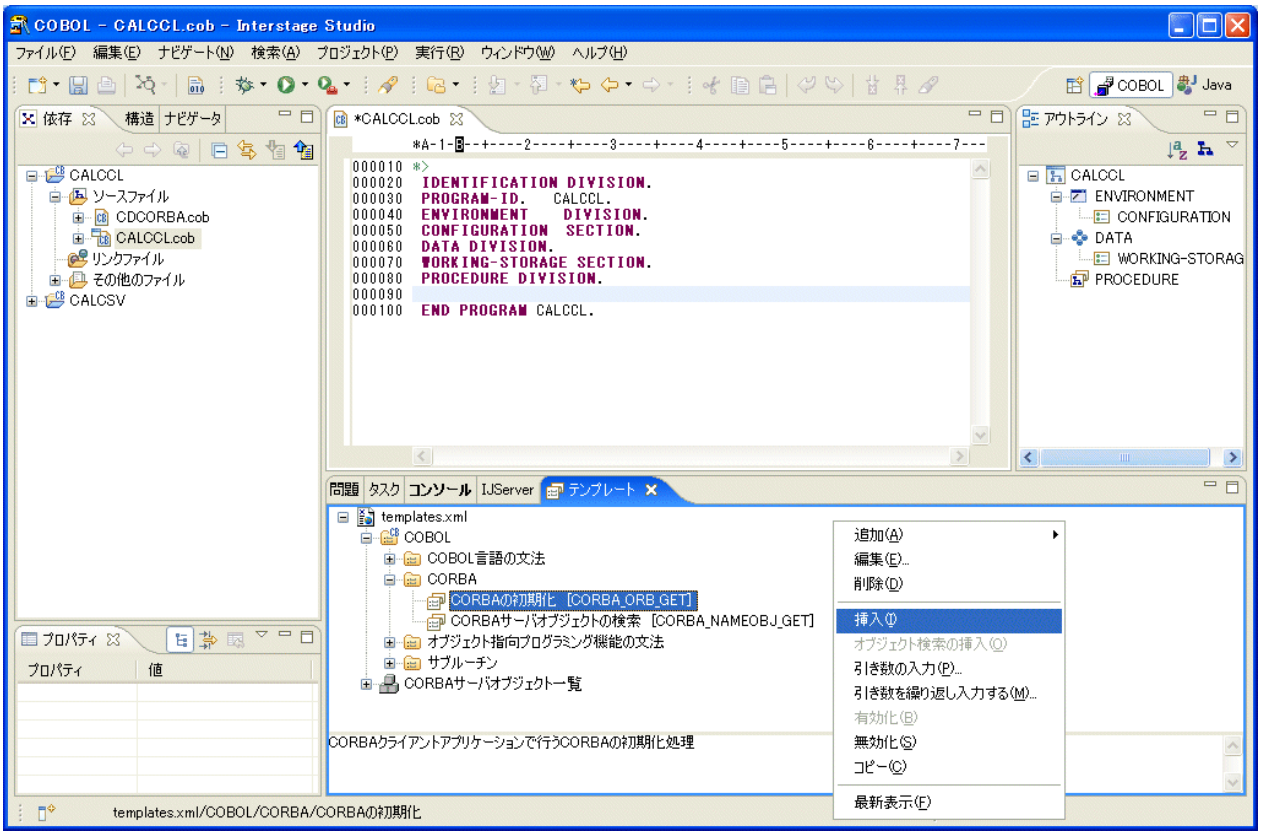


開いたビューは任意の箇所に移動できます。以降では、タスクビューやコンソールビューと同じ場所に移動して利用しています。

CORBA初期化処理を追加

エディタ上で処理を追加する位置にカーソルを位置付けます。テンプレートビューから、[COBOL] > [CORBA] > [CORBAの初期化]を選択し、コンテキストメニューから[挿入]を選択して、処理を追加します。

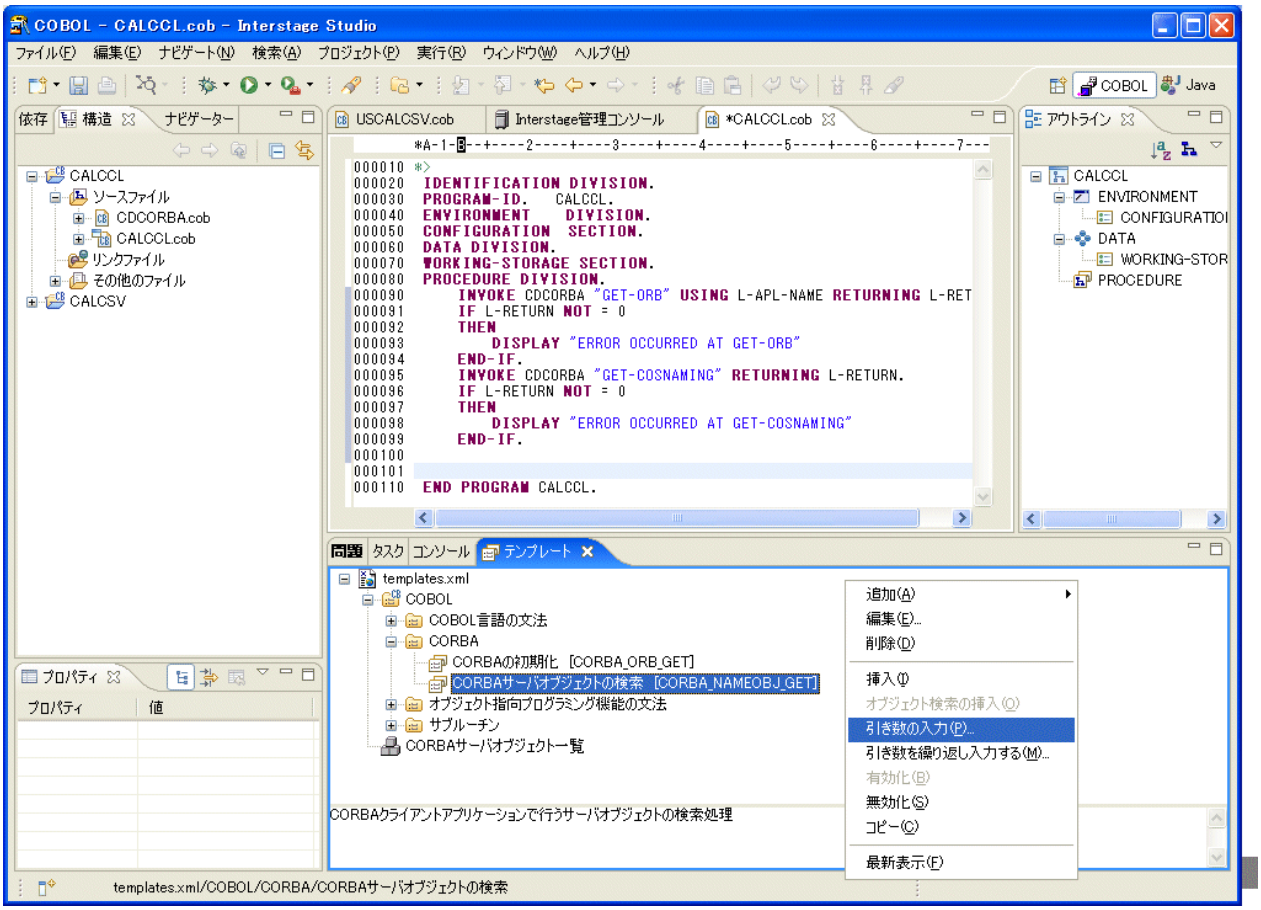
この例では、CALCCL.cobの9行目（行番号90）に[CORBAの初期化]の処理を挿入します。



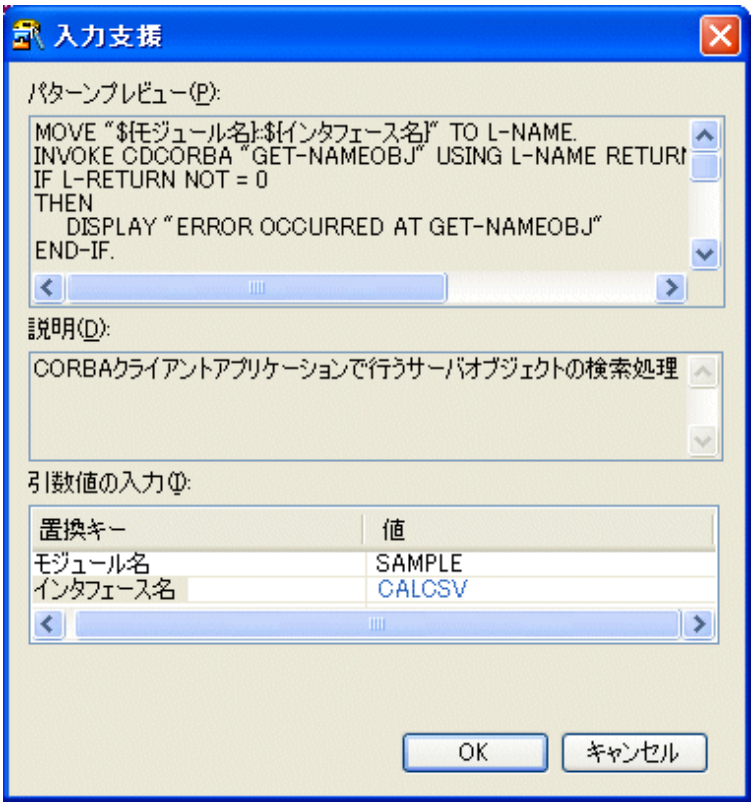
CORBAサーバオブジェクト検索処理を追加

エディタ上で処理を追加する位置にカーソルを位置付けます。テンプレートビューから、[COBOL] > [CORBA] > [CORBAサーバオブジェクトの検索]を選択し、コンテキストメニューから[引き数の入力]を選択します。

この例では、CALCCL.cobの19行目（行番号100）の下に空行を1行作成し、そこに[CORBAサーバオブジェクトの検索]の処理を挿入します。



表示された[入力支援]ダイアログボックスで、置換キーに対する値を以下のように指定し、[OK]ボタンを押して処理を追加します。



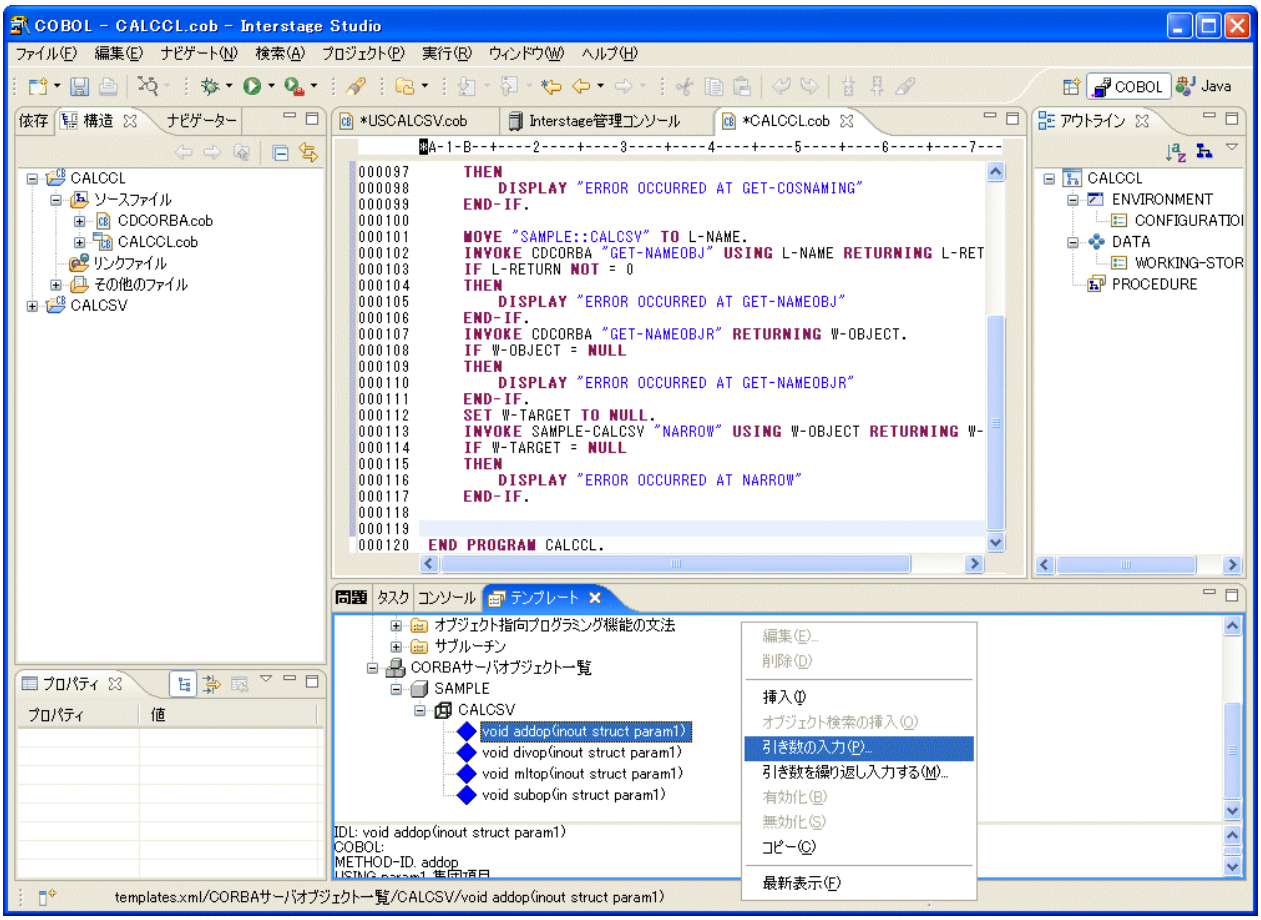
メソッド呼び出し処理の追加

エディタ上で処理を追加する位置にカーソルを位置付けます。テンプレートビューの[CORBAサーバオブジェクト一覧]から、[モジュール名] > [インタフェース名] > [メソッド]を選択し、コンテキストメニューから[引き数の入力]を選択します。

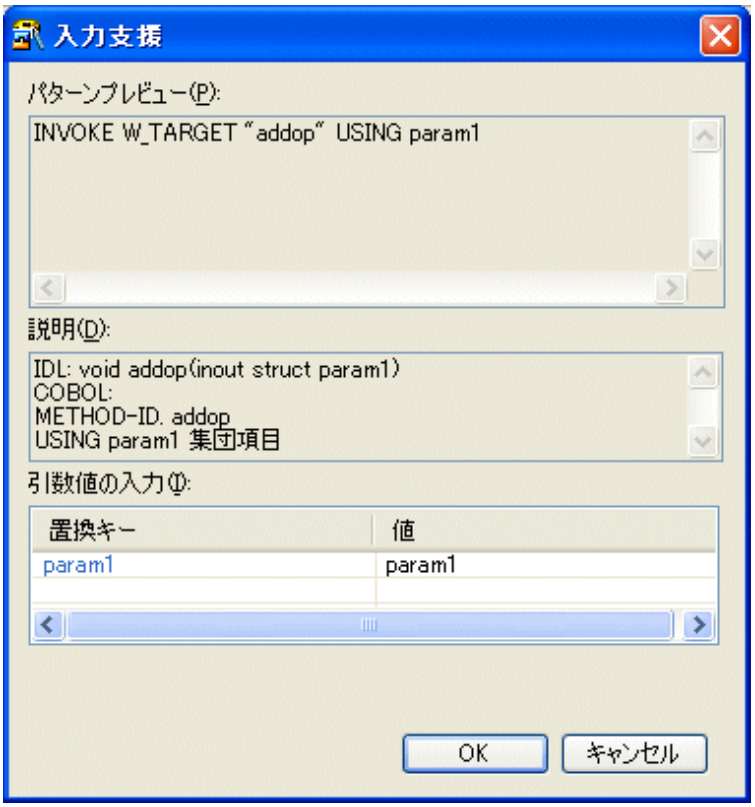
この例では、CALCCL.cobの37行目（行番号118）の下に空行を1行作成し、そこに[addop]メソッドの呼び出し処理を挿入します。

注意

テンプレートからCORBAサーバオブジェクト一覧を取得する場合は、InterstageおよびInterstageの管理に必要なサービス（基盤サービス）が起動している必要があります。
なお、基盤サービスの起動にはInterstage基盤サービス操作ツールを使用します。ツールを起動するには、スタートメニューから[Interstage] > [Studio] > [Interstage基盤サービス操作ツール]を選択してください。Interstageを起動する場合は、IJServerビューで右クリックし、[起動]を選択します。IJServerビューを表示する場合は、メニューバーから[ウィンドウ] > [ビューの表示] > [その他]を選択し、ビューの表示ダイアログで、[IJServer] > [IJServer]を選択して[OK]ボタンをクリックします。



置換キーに対応する値を設定し、[OK]ボタンを押して処理を追加します。



必要なデータ項目や処理記述を追加・修正し、以下のようなソースにします。

注意

最後に追加したメソッド呼び出し処理は、以下のソースでは修正して利用しています。ソースをコピーする場合は、あらかじめ追加したメソッド呼び出し処理を削除してください。

```
000010*>
000020 IDENTIFICATION DIVISION.
000030 PROGRAM-ID.    CALCCL.
000040 ENVIRONMENT    DIVISION.
000050 CONFIGURATION  SECTION.
000060   REPOSITORY.
000070* ObjectDirectorの標準登録集（リポジトリ宣言用）
000080     COPY CORBA--REP.
000090* ネーミングサービスの標準登録集（リポジトリ宣言用）
000100     COPY CosNaming--REP.
000110* IDLコンパイラが出力した登録集（リポジトリ宣言用）
000120     COPY USCALCSV--REP.
000130* COBOLプラグインが提供しているCORBAクライアント開発用クラス
000140     CLASS CDCORBA.
000150
000160  SPECIAL-NAMES.
000170     ARGUMENT-NUMBER IS 引数番号
000180     ARGUMENT-VALUE  IS 引数内容
000190     SYMBOLIC CONSTANT
000200* ObjectDirectorの標準登録集（定数宣言用）
000210     COPY CORBA--CONST.
000220* ネーミングサービスの標準登録集（定数宣言用）
000230     COPY COSNAMING--CONST.
000240* IDLコンパイラが出力した登録集（定数宣言用）
000250     COPY USCALCSV--CONST.
000260     .
```

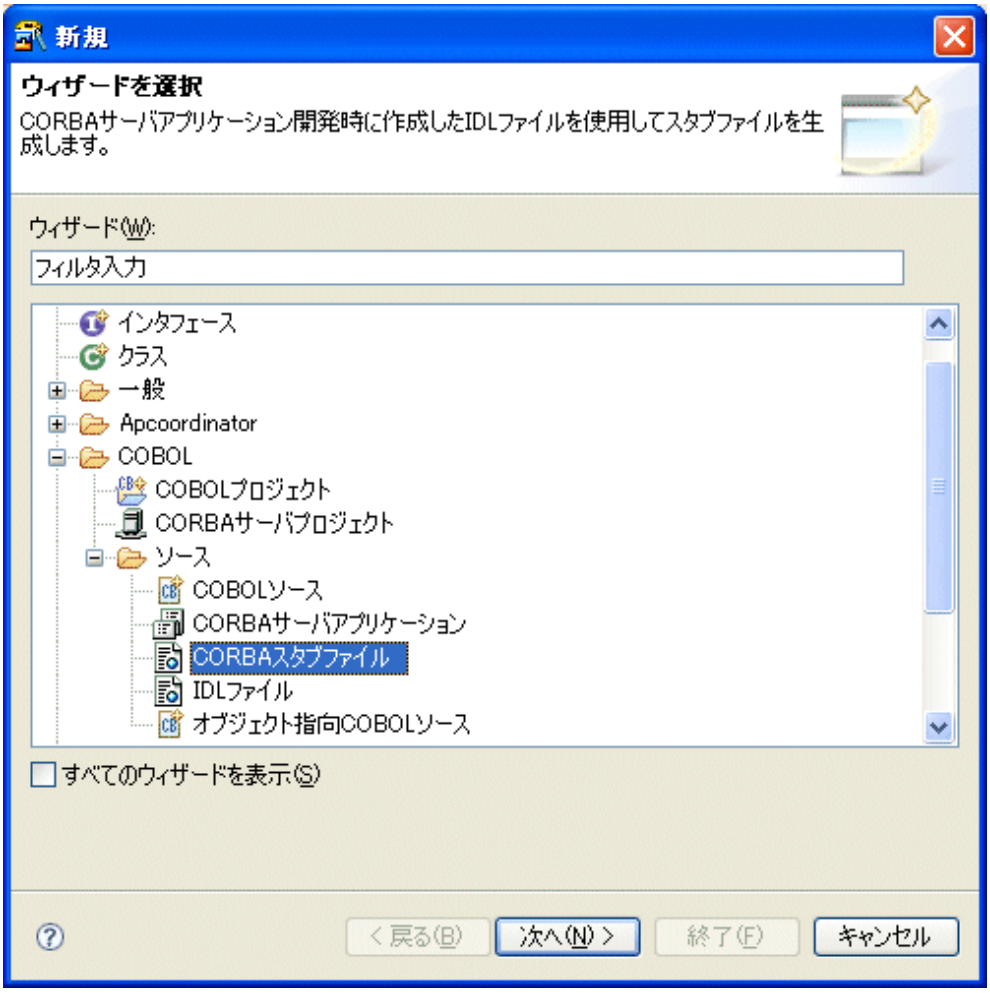
```
000270 DATA DIVISION.
000280 WORKING-STORAGE SECTION.
000290 COPY CORBA--COPY.
000300 COPY USCALCSV--COPY.
000310 01 L-APL-NAME PIC X(50) VALUE "CALCCL".
000320 01 W-OBJECT OBJECT REFERENCE CORBA-OBJECT.
000330 01 W-TARGET OBJECT REFERENCE SAMPLE-CALCSV.
000340 01 STRUCT1  TYPE SAMPLE-S1.
000350 01 L-RETURN PIC S9(9) COMP-5.
000360 01 L-NAME PIC X(128) VALUE "SAMPLE::CALCSV".
000370 01 ERR-MSG PIC X(128).
000380 01 CDEXCEPTIONMSG OBJECT REFERENCE CORBA-STRING.
000390 01 CDEXCEPTIONCODE TYPE CORBA-LONG.
000400
000410 PROCEDURE DIVISION.
000420  DECLARATIVES.
000430   ERR SECTION.
000440     USE AFTER EXCEPTION SAMPLE-CDEXCEPTION.
000450     MOVE CDEXCEPTIONCODE OF EXCEPTION-OBJECT AS SAMPLE-CDEXCEPTION TO CDEXCEPTIONCODE.
000460     SET CDEXCEPTIONMSG TO CDEXCEPTIONMSG OF EXCEPTION-OBJECT AS SAMPLE-CDEXCEPTION.
000470     INVOKE CDEXCEPTIONMSG "GET-VALUE" RETURNING ERR-MSG.
000480     DISPLAY ERR-MSG.
000490  END DECLARATIVES.
000500     INVOKE CDCORBA "GET-ORB" USING L-APL-NAME RETURNING L-RETURN.
000510     IF L-RETURN NOT = 0
000520     THEN
000530         DISPLAY "ERROR OCCURRED AT GET-ORB"
000540     END-IF
000550     INVOKE CDCORBA "GET-COSNAMING" RETURNING L-RETURN.
000560         IF L-RETURN NOT = 0
000570     THEN
000580          DISPLAY "ERROR OCCURRED AT GET-COSNAMING"
000590     END-IF
000600
000610     MOVE "SAMPLE::CALCSV" TO L-NAME.
000620     INVOKE CDCORBA "GET-NAMEOBJ" USING L-NAME RETURNING L-RETURN.
000630     IF L-RETURN NOT = 0
000640     THEN
000650        DISPLAY "ERROR OCCURRED AT GET-NAMEOBJ"
000660     END-IF
000670     INVOKE CDCORBA "GET-NAMEOBJR" RETURNING W-OBJECT.
000680     IF W-OBJECT = NULL
000690     THEN
000700        DISPLAY "ERROR OCCURRED AT GET-NAMEOBJR"
000710     END-IF
000720
000730     SET W-TARGET TO NULL.
000740     INVOKE SAMPLE-CALCSV "NARROW" USING W-OBJECT RETURNING W-TARGET.
000750     IF W-TARGET = NULL
000760     THEN
000770        DISPLAY "ERROR OCCURRED AT NARROW"
000780     END-IF
000790
000800*    サーバアプリケーションのメソッドの呼出し
000810     DISPLAY "第一引数を入力してください：" WITH NO ADVANCING.
000820     ACCEPT  item1 OF STRUCT1.
000830     DISPLAY "第二引数を入力してください：" WITH NO ADVANCING.
000840     ACCEPT  item2 OF STRUCT1.
000850
000860     INVOKE W-TARGET "addop" USING STRUCT1
000870     DISPLAY "加算結果：" result OF STRUCT1.
000880
000890     INVOKE W-TARGET "subop" USING STRUCT1
```

```
000900     DISPLAY "減算結果：" result OF STRUCT1.
000910
000920     MOVE 0 TO CDEXCEPTIONCODE.
000930     DISPLAY "乗算結果：" WITH NO ADVANCING.
000940     INVOKE W-TARGET "mltop" USING STRUCT1
000950     IF CDEXCEPTIONCODE NOT = -1 THEN
000960       DISPLAY result OF STRUCT1
000970     END-IF.
000980
000990     MOVE 0 TO CDEXCEPTIONCODE.
001000     DISPLAY "除算結果：" WITH NO ADVANCING.
001010     INVOKE W-TARGET "divop" USING STRUCT1
001020     IF CDEXCEPTIONCODE NOT = -1 THEN
001030       DISPLAY result OF STRUCT1
001040     END-IF.
001050
001060 END PROGRAM CALCCL.
```

5. スタブの追加

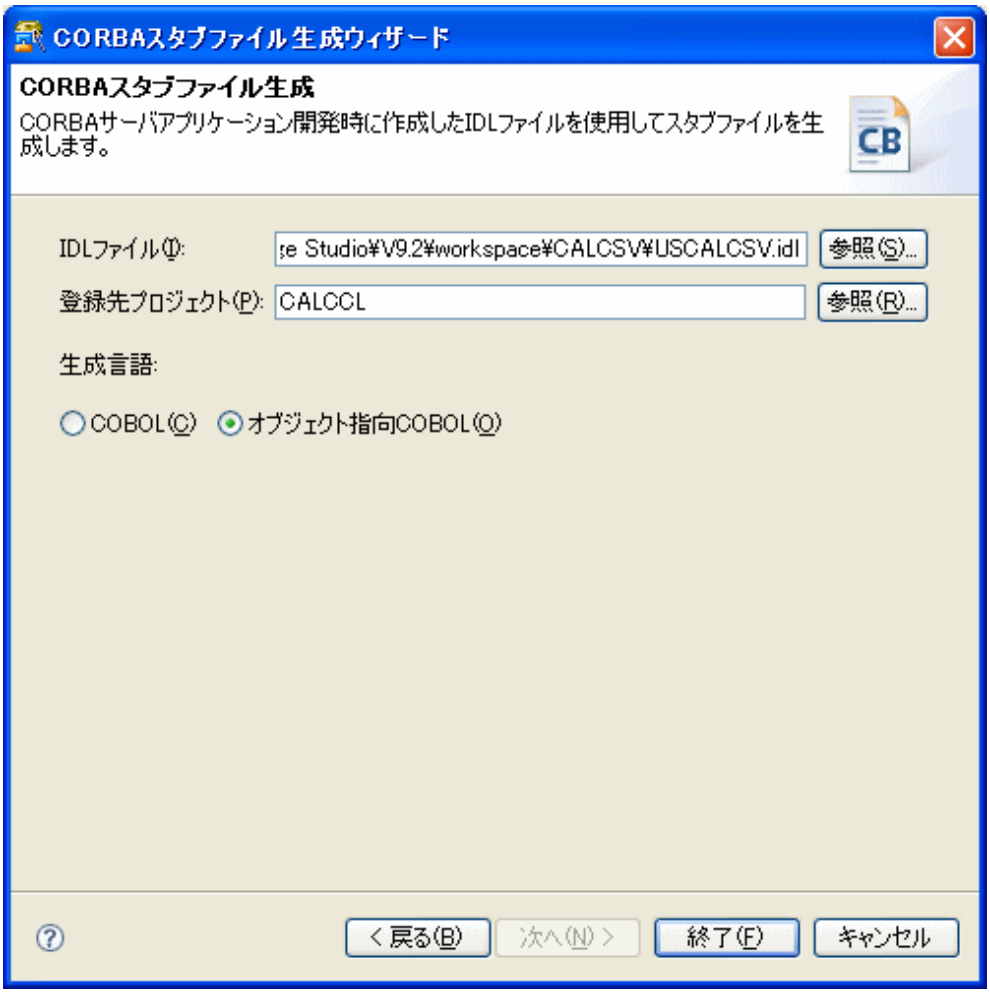
CORBAサーバアプリケーションを呼び出すには、IDLファイルからCORBAクライアントに必要なスタブファイルおよびその他のファイルを生成し、プロジェクトに追加する必要があります。CORBAスタブファイル生成ウィザードを利用してスタブファイルおよびその他のファイルを生成し、プロジェクトに追加してください。

ワークベンチ のメニューより[ファイル] > [新規] > [その他]を選択すると、新規ウィザードが表示されます。



[COBOL] > [ソース] > [CORBAスタブファイル]を選択します。

CORBAスタブファイル生成ウィザードが表示されます。



IDLファイルと登録先プロジェクト、および生成言語を入力します。

設定項目	設定内容
IDLファイル	CALCSVプロジェクトのUSCALCSV.idlを指定してください
登録先プロジェクト	CALCCL
生成言語	オブジェクト指向COBOL

[終了]ボタンを押すと、プロジェクトに以下のファイルが生成されます。

ファイル種別	ファイル名
リポジトリ段落宣言用登録集	USCALCSV--REP.cbl
定数宣言用登録集	USCALCSV--CONST.cbl
TYPEDEF型宣言用登録集	USCALCSV--COPY.cbl
インタフェースファイル	SAMPLE-CALCSV.cob
Helperクラスファイル	SAMPLE-CALCSV--HELPER.cob
スタブファイル	SAMPLE-CALCSV--STUB.cob
Narrowスタブファイル	SAMPLE-CALCSV_NARROW.cob
データ型クラスファイル	SAMPLE-CDEXCEPTION.cob
データ型Helperファイル	SAMPLE-S1--HELPER.cob SAMPLE-CDEXCEPTION--HELPER.cob

生成されたファイルのうち、登録集以外のCOBOLソースが[依存]ビューにソースファイルとして登録されます。

6. 依存関係の解析

オブジェクト指向COBOLのソースファイルを含んだプロジェクトをビルドする場合、ビルドを行う前にCOBOLソースファイル間の依存関係を設定する必要があります。依存関係を設定することでCOBOLソースファイルが正しい順番でビルドされるようになります。依存関係が正しく設定されていないとビルド時に翻訳エラーが発生することがあります。

プロジェクトの新規作成時および[ソースフォルダ]にCOBOLソースファイルを追加した時には、追加されたソースファイルに対して依存関係が自動的に解析されて設定されます。しかし今回の例のようにCOBOLソースファイルのREPOSITORY段落を編集した場合には、ビルドを行う前に手動で依存関係の解析を実行する必要があります。

ワークベンチの依存ビューでプロジェクトを選択し、コンテキストメニューから [依存関係の解析] > [すべて]を選択して、依存関係の解析を行ってください。

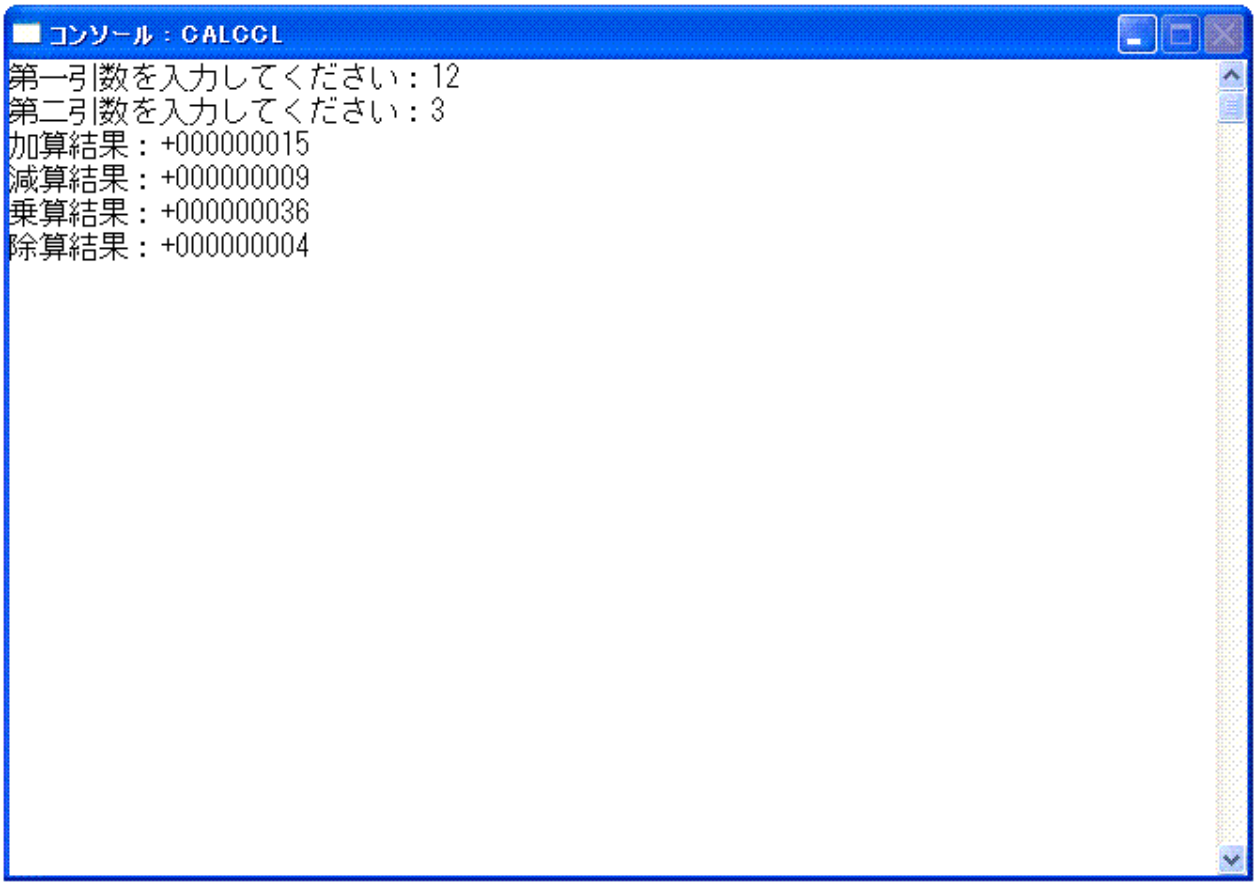
依存関係の解析を行うと、コンソールに“xxx.rep: は存在しません。”というメッセージが表示されます。プロジェクト内のオブジェクト指向COBOLソースファイルに対するリポジトリファイル（*.rep）はそのソースファイルを翻訳して初めて生成されるもののため、ビルドを行う前に依存関係の解析を行うとこのようなメッセージが表示されます。依存関係の解析自体は正しく行われていますので、そのままプロジェクトのビルドを行って構いません。

7. プロジェクトのビルド

ここまでの手順でCORBAクライアントアプリケーションのビルドは完了しています。
ただし、メニューの[プロジェクト] > [自動的にビルド]のチェックを解除している場合はビルドがされないため、チェックしなおしてください。チェックをするとにCORBAクライアントアプリケーションが自動的にビルドされます。

8. 動作確認

CORBAサーバアプリケーションを配備したワークユニットを起動後に、ワークベンチのビューでプロジェクトを選択して、メニューより[実行] > [実行] > [COBOLアプリケーション]を選択して、クライアントアプリケーションを実行します。


コンソール：CALCCL
第一引数を入力してください：12
第二引数を入力してください：3
加算結果：+000000015
減算結果：+000000009
乗算結果：+000000036
除算結果：+000000004

第5部 リファレンス編

第15章　COBOLサービスクラスの概要

COBOLサービスクラスは、COBOLプラグインが提供しているクラスで、COBOLアプリケーションを開発するときに利用します。COBOLプラグインで提供しているCOBOLサービスクラスには、CDCORBAクラスがあります。

第16章　CDCORBAクラス

CDCORBAクラスは、CORBAクライアントアプリケーションを開発するためのクラスです。
ここでは、CDCORBAクラスの各メソッドについて説明します。

16.1 メソッド一覧

メソッド名	説明
CREATE	メンバ変数の初期化を行います。
GET-ORB	ORBの初期化を行い、ORBのオブジェクトリファレンスを取得します。
GET-COSNAMING	ネーミングサービスのオブジェクトリファレンスを取得します。
GET-NAMEOBJ	使用するサーバアプリケーションのオブジェクトリファレンスを取得します。
GET-ERROR-MSG	エラー発生時に、エラーメッセージの内容を参照します。
GET-ORBR	取得済みのORBのオブジェクトリファレンスを参照します。
SET-ORBR	ORBのオブジェクトリファレンスを設定します。
GET-COSNAMINGR	取得済みのネーミングサービスのオブジェクトリファレンスを参照します。
SET-COSNAMINGR	ネーミングサービスのオブジェクトリファレンスの設定を行います。
GET-NAMEOBJR	取得済みのサーバオブジェクトリファレンスを参照します。
SET-NAMEOBJR	サーバオブジェクトリファレンスの設定を行います。

16.2 メソッド詳細

メソッドの詳細について説明します。

16.2.1 CREATEメソッド

対象クラス

CDCORBAクラス

説明

CREATEメソッドの処理を行い、CDCORBAクラスのメンバ変数の初期化を行います。
CREATEメソッドの詳細は、"COBOL文法書"を参照してください。

記述形式

```
INVOKE CDCORBA “CREATE” RETURNING L-RETURN.
```

パラメタ

なし

復帰値

```
L-RETURN [属性: OBJECT REFERENCE SELF]
```

CDCORBA自身のオブジェクトリファレンスを返却します。

参照

COBOLのFJBASEクラスのCREATEメソッド

16.2.2 GET-ORBメソッド

対象クラス

CDCORBAクラス

説明

ORBの 初期化処理を行い、ORBのオブジェクトリファレンスを取得します。
取得したオブジェクトリファレンスを参照する場合は、GET-ORBRメソッドを使用します。

記述形式

```
INVOKE CDCORBA “GET-ORB” USING L-PARAM RETURNING L-RETURN.
```

パラメタ

```
L-PARAM [属性 : PIC X(50)]
```

クライアントアプリケーションの名前

復帰値

```
L-RETURN [属性: PIC S9(9) COMP-5]
```

関数の処理結果を返却します。

0 : 正常終了

負数 : 異常終了(続行不可能)

正数 : すでにORBが取得されている

参照

GET-ORBRメソッド

16.2.3 GET-COSNAMINGメソッド

対象クラス

CDCORBAクラス

説明

ネーミングサービスのオブジェクトリファレンスを取得します。
取得したオブジェクトリファレンスを参照する場合は、GET-COSNAMINGRメソッドを使用します。

記述形式

```
INVOKE CDCORBA “GET-COSNAMING” RETURNING L-RETURN.
```

パラメタ

なし

復帰値

```
L-RETURN [ 属性 : S9(9) COMP-5 ]
```

関数の処理結果を返却します。

0 : 正常終了

負数 : 異常終了

参照

GET-COSNAMINGRメソッド

16.2.4 GET-NAMEOBJメソッド

対象クラス

CDCORBAクラス

説明

パラメタで指定されたネーミングサービス名のサーバオブジェクトリファレンスを取得します。
取得したオブジェクトリファレンスを参照する場合は、GET-NAMEOBJRメソッドを使用します。

記述形式

```
INVOKE CDCORBA “GET-NAMEOBJ” USING L-PARAM RETURNING L-RETURN.
```

パラメタ

```
L-PARAM [ 属性 : X(128) ]
```

取得したい、サーバアプリケーションのネーミングサービス名。

ネーミングサービス名の詳細は、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。

復帰値

```
L-RETURN [ 属性 : S9(9) COMP-5 ]
```

関数の処理結果を返却します。

0 : 正常終了

負数 : 異常終了

参照

GET-NAMEOBJRメソッド

16.2.5 GET-ERROR-MSGメソッド

対象クラス

CDCORBAクラス

説明

エラー発生時のエラーの内容を返却します。
返却内容は、CORBA-Exceptionに返却されるEXCEPTION-IDを返却しています。詳細は、"Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。

記述形式

```
INVOKE CDCORBA “GET-ERROR-MSG” RETURNING L-RETURN.
```

パラメタ

なし

復帰値

```
L-RETURN [ 属性 : X(128) ]
```

CORBA-Exceptionより返却された内容。

参照

CORBA-Exception ("Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。)

16.2.6 GET-ORBRメソッド

対象クラス

CDCORBAクラス

説明

取得済みのORBのオブジェクトリファレンスを参照します。

記述形式

```
INVOKE CDCORBA “GET-ORBR” RETURNING L-RETURN.
```

パラメタ

なし

復帰値

```
L-RETURN [ 属性 : OBJECT REFERENCE CORBA-ORB ]
```

GET-ORBメソッドにより取得されたORBオブジェクトリファレンスを返却します。

参照

GET-ORBメソッド

CORBA-ORB("Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。)

16.2.7 SET-ORBRメソッド

対象クラス

CDCORBAクラス

説明

ORBのオブジェクトリファレンスを設定します。

記述形式

```
INVOKE CDCORBA “SET-ORBR” USING L-PARAM.
```

パラメタ

```
L-PARAM [ 属性 : OBJECT REFERENCE CORBA-ORB ]
```

ORBのオブジェクトリファレンス。

復帰値

なし

参照

GET-ORBメソッド

GET-ORBRメソッド

CORBA-ORB("Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。)

16.2.8 GET-COSNAMINGRメソッド

対象クラス

CDCORBAクラス

説明

取得済みのネーミングサービスのオブジェクトリファレンスを参照します。

記述形式

```
INVOKE CDCORBA “GET-COSNAMINGR” RETURNING L-RETURN.
```

パラメタ

なし

復帰値

```
L-RETURN [ 属性 : OBJECT REFERENCE COSNAMING-NAMINGCONTEXT ]
```

取得済みのネーミングサービスのオブジェクトリファレンス。

参照

GET-COSNAMINGメソッド

COSNAMING-NAMINGCONTEXT("Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。)

16.2.9 SET-COSNAMINGRメソッド

対象クラス

CDCORBAクラス

説明

ネーミングサービスのオブジェクトリファレンスの設定を行います。

記述形式

```
INVOKE CDCORBA “SET-COSNAMINGR” USING L-PARAM.
```

パラメタ

```
L-PARAM [ 属性 : OBJECT REFERENCE COSNAMING-NAMINGCONTEXT ]
```

ネーミングサービスのオブジェクトリファレンス。

復帰値

なし

参照

GET-COSNAMINGメソッド

GET-COSNAMINGRメソッド

COSNAMING-NAMINGCONTEXT("Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。)

16.2.10 GET-NAMEOBJRメソッド

対象クラス

CDCORBAクラス

説明

サーバアプリケーションのオブジェクトリファレンスを参照します。

記述形式

```
INVOKE CDCORBA “GET-NAMEOBJR” RETURNING L-RETURN.
```

パラメタ

なし

復帰値

```
L-RETURN [ 属性 : OBJECT REFERENCE CORBA-OBJECT ]
```

サーバアプリケーションのオブジェクトリファレンス。

参照

GET-NAMEOBJメソッド

CORBA-OBJECT("Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。)

16.2.11 SET-NAMEOBJRメソッド

対象クラス

CDCORBAクラス

説明

サーバアプリケーションのオブジェクトリファレンスを設定します。

記述形式

```
INVOKE CDCORBA “SET-NAMEOBJR” USING L-PARAM
```

パラメタ

```
L-PARAM [ 属性 : OBJECT REFERENCE CORBA-OBJECT ]
```

サーバアプリケーションのオブジェクトリファレンス。

復帰値

なし

参照

GET-NAMEOBJメソッド

GET-NAMEOBJRメソッド

CORBA-OBJECT("Interstage Application Server アプリケーション作成ガイド(CORBAサービス編)"を参照してください。)

付録A　サンプルの使用方法

ここでは、提供しているサンプルについて説明します。

サンプルの格納先

サンプルアプリケーションは、以下に格納されています。

```
本製品のインストールフォルダ¥IDE¥1000¥manual¥1¥tutorial¥samples¥cobol
```

サンプルの一覧

ファイル名	プロジェクト名	概要
calcsv.zip	calcsv	Interstage CORBAサービスを使用したサーバアプリケーションです。 ※"第4部 チュートリアル編"の"第13章 CORBAサーバアプリケーション開発"で作成するアプリケーションです。
calccl.zip	calccl	Interstage CORBAサービスを使用したクライアントアプリケーションです。 ※"第4部 チュートリアル編"の"第14章 CORBAクライアントアプリケーション開発"で作成するアプリケーションです。

サンプルの利用手順

サンプルは以下の手順でワークスペースにインポートします。

1. ワークベンチを起動します。
2. メニューから[ファイル] > [インポート]を選択します。
3. インポートソースの選択で[一般] > [既存プロジェクトをワークスペースへ]を選択し、[次へ]をクリックします。
4. 次画面で[アーカイブファイルの選択]を選択し、[参照]をクリックしてインポートするアーカイブファイルを指定します。
5. [終了]をクリックします。

付録B　Tips

ここでは、ワークベンチが更に使いやすくなる便利な機能について紹介します。

B.1　CORBAワークユニット(MyCORBADebug)を作成するには

CORBAアプリケーションのローカルデバッグで使用するデフォルトのCORBAワークユニット(MyCORBADebug)を作成する手順を以下に示します。

注意

CORBAワークユニットの作成には、PATH、CLASSPATHの設定が必要なため、インストール時にPATH、CLASSPATHの設定時にエラーが発生した場合は、PATH、CLASSPATHを設定して、コンピュータを再起動してください。

CORBAワークユニット(MyCORBADebug)の作成

CORBAワークユニットを作成するには、通常は、Interstage管理コンソールを使用しますが、ここでは、ローカルデバッグ向けのCORBAワークユニット用の定義ファイルを利用して、コマンドで作成する方法を示します。

Interstage管理コンソールからCORBAワークユニットの作成方法については、"Interstage Application Server 運用ガイド(基本編)"を参照してください。

- ワークユニット生成コマンド(isaddwudef)の実行

  CORBAワークユニット定義ファイルを引数にして、以下のコマンドを実行します。
  isaddwudefコマンドの詳細については、"Interstage Application Server リファレンスマニュアル(コマンド編)"を参照してください。

```
isaddwudef Java統合開発環境のインストールフォルダ¥etc¥ijserver¥MyCORBADebug.wu
```

注意

CORBAワークユニット定義ファイル内の"Path"、"Current Directory"プロパティには、作業フォルダなどの情報が設定されています。通常は、インストール時にインストール先フォルダの情報から値が設定されますが、パス構成に誤りがある場合は、適宜インストール環境に合わせて、設定を変更してください。

付録C　トラブルシューティング

ここでは、ワークベンチで発生する問題を解決する方法を説明します。

C.1　COBOLアプリケーションに関する問題

1. 依存関係解析を実行すると、リポジトリファイル（*.rep）が無いというエラーメッセージが表示される。

依存関係解析を実行すると、コンソールビューに以下のエラーメッセージが表示される。

```
COBOLソースファイルのフルパス名:行位置:リポジトリファイル名: は存在しません。
```

【対処】

以下を確認してください。

- プロジェクト内のCOBOLソースファイルに対するリポジトリファイルの場合

  プロジェクトの新規作成時またはプロジェクトをクリーンした後は、プロジェクト内にリポジトリファイルが存在しないため、このエラーメッセージが表示されます。これらのリポジトリファイルはプロジェクトをビルドすると生成されますので、このエラーメッセージはそのままにしておいてかまいません。

- プロジェクト外のリポジトリファイルの場合

  COBOLソースファイル内の記述が正しいか、または必要なリポジトリファイルがすべて揃っているかを確認してください。

2. COBOLエディタで、漢字やカナが混在した変数名をダブルクリックで正しく選択できない。

COBOLソース内に"ローカル変数"のようにカナと漢字が混在した変数名がある場合、COBOLエディタ上でその変数名をダブルクリックしても、"ローカル"または"変数"だけが選択されてしまい、変数名全体を選択できない。

【対処】

このような変数名は、マウスによるドラッグまたはShift+カーソルキーを用いた選択方式で選択してください。

3. COBOL登録集内をデバッグできない。

【対処】

COBOLの登録集内はデバッグできません。

NetCOBOLのプロジェクトマネージャのデバッガを使用することで登録集内のデバッグを行うことができます。

4. デバッグ時に[実行]メニュー内のいくつかのメニュー項目が使用できない。

COBOLアプリケーションのデバッグ時には、[実行]メニュー内の以下のメニュー項目が常に無効表示となる。

- 指定行まで実行
- ステップフィルタの使用
- 監視
- インスペクション
- 表示
- 実行
- 選択項目にステップイン
- 行ブレークポイントの切り替え
- メソッドブレークポイントの切り替え
- 監視ポイントの切り替え

【対処】

これらのメニュー項目はJavaアプリケーションのデバッグ時に使用するもので、COBOLアプリケーションのデバッグ時には使用できません。

- ブレークポイントの追加や削除を行う際には、COBOLエディタの垂直方向ルーラ上でマウスを右クリックし、コンテキストメニューから[ブレークポイントの追加]または[ブレークポイントの削除]を選択してください。

指定行まで実行させる際には、COBOLエディタのコンテキストメニューで[指定行まで実行]を選択してください。

5. デバッグ時にスコープ外の変数も値が表示される。

【対処】

スコープ外の変数も値が表示されますが、正しい値ではありません。スコープ外の変数に表示されている値は使用しないでください。

6. ターゲット実行ファイルが見つかりません。次へ進めません。」というエラーダイアログボックスが表示される。

COBOLプロジェクトを含むワークスペースフォルダをコピーまたは移動して作成したワークスペースでCOBOLアプリケーションを実行またはデバッグすると、「ターゲット実行ファイルが見つかりません。次へ進めません。」というエラーダイアログボックスが表示される。

【対処】

COBOLプロジェクトを含むワークスペースフォルダをコピーまたは移動した場合、COBOLアプリケーションの[構成およびデバッグ]または[構成および実行]の[メイン]タブにある[実行ファイル]のパス名が、コピーまたは移動する前のパス名になっていることが原因です。

この場合、[構成およびデバッグ]または[構成および実行]の[メイン]タブの[プロジェクト名]の[参照]をクリックして、[プロジェクトの選択]ダイアログボックスでプロジェクト名を選択し直してください。正しいパス名に変更されます。

7. 外部エディタでリソースを変更した後に[プロジェクトのビルド]や[すべてビルド]を選択しても、ビルドが行われない。

【対処】

リソースの変更がワークスペースに反映されていません。

以下のどちらかを行ってください。

- メニューバーから[ウィンドウ] > [設定] > [一般] > [ワークスペース] を選択し、[自動で更新]を選択してください。

  このオプションが選択されていると、ワークスペースのリソースは、自動的にファイルシステム内の対応するリソースと同期化されます。

- 外部エディタでリソースを更新したときは、ビルドを行う前に[依存]または[構造]ビューで、更新したリソースまたはそのリソースを含むフォルダを選択し、メニューバーの[ファイル] > [最新表示]を選択してください。

8. リモート開発のサーバ接続確認時にsttyコマンドがエラーとなる。

SolarisまたはLinuxサーバとの連携時にsttyコマンドがエラーとなり、失敗する。

- Solarisのエラーメッセージ

```
ホストでのコマンド処理中にエラーが発生しました。
stty: : 引数が正しくありません。
```

- Linuxのエラーメッセージ

```
ホストでのコマンド処理中にエラーが発生しました。
stty:標準入力:無効な引数です。
```

【対処】

.bashrcファイル(bash使用時)または.cshrcファイル(csh使用時)に記述している、sttyコマンドをコメントにしてください。

例えば、以下のように記述している場合、

```
stty erase ^H
```

次のようにコメント化します。

```
#stty erase ^H
```

9. リモート開発のサーバ接続確認時にフリーズする。

SolarisまたはLinuxサーバとの連携時にサーバからの応答がなく、ワークベンチがフリーズする。

【対処】

.bashrcファイル(bash使用時)または.cshrcファイル(csh使用時)に、ユーザの入力を要求して応答待ちを引き起こすスクリプトなどが記述されている場合に発生します。

ユーザの入力を要求するなど応答待ちになるスクリプトを、ファイルから削除します。

10. リモート開発時に「コマンドの送信中にエラーが発生しました。リモート側から接続がリセットされました。」のエラーが表示される。

SolarisまたはLinuxサーバとの連携時に、「コマンドの送信中にエラーが発生しました。リモート側から接続がリセットされました。」のエラーとなり、サーバへ接続できない。

【対処】

ドメイン名とIPアドレスを管理するDNS(Domain Name System)の設定によっては、rexecデーモンが行う接続元PCのホスト名の逆引きに失敗することがあります。

サーバの/etc/hostsファイルを編集して、クライアント側PCの、以下の情報を追加してください。

```
IPアドレス　ホスト名
```

11. リモート開発時にテキストファイルの転送で改行コードが変換されない。

Linuxのファイル転送(FTP)にvsftpdを使用している場合、テキストファイルを転送すると改行コードがWindowsの改行コード(0x0d0a)のままとなり、Unixの改行コード(0x0a)にならない。

【対処】

vsftpdの設定が改行コードを変換しないようになっていることが原因です。

改行コードの変換を有効にするには、/etc/vsftpd/vsftpd.confファイル内でコメントアウトされている次の行の先頭の#を削除する必要があります。

- 修正前

```
#ascii_upload_enable=YES
#ascii_download_enable=YES
```

- 修正後

```
ascii_upload_enable=YES
ascii_download_enable=YES
```

12. リモート開発の機能を使用すると「サーバのコマンドパラメタの指定に誤りがあります。」のエラーになる。

SolarisまたはLinuxサーバとの連携時にリモート開発の機能を使用すると、「サーバのコマンドパラメタの指定に誤りがあります。」のエラーになり、リモート開発の機能を使用することかできない。

【対処】

vsftpdの設定が改行コードを変換しないようになっていることが原因です。

改行コードの変換を有効にするには、/etc/vsftpd/vsftpd.confファイル内でコメントアウトされている次の行の先頭の#を削除する必要があります。

・ 修正前

```
#ascii_upload_enable=YES
#ascii_download_enable=YES
```

・ 修正後

```
ascii_upload_enable=YES
ascii_download_enable=YES
```

13. リモートデバッグを開始するとローカル側のプロジェクトのビルドが実行されることがある。

[起動前に(必要に応じて)ビルド]が指示されているため、ローカル側でのビルドが実行されてしまうことがあります。

【対処】

以下の操作を行って[起動前に(必要に応じて)ビルド]を解除してください。

1. メニューバーから[ウィンドウ] > [設定]を選択します。[設定]ダイアログボックスが表示されます。
2. 左のペインで[実行/デバッグ] > [起動]を選択すると[起動]ページが表示されます。
3. [汎用オプション]グループボックスの[起動前に(必要に応じて)ビルド]のチェックを解除してください。

C.2 CORBAアプリケーションに関する問題

1. IDLファイルがコンパイルされない。

【対処】

IDLファイルのコンパイルを行うには、あらかじめInterstage Application ServerおよびInterstageの管理に必要なサービス(基盤サービス)を起動しておく必要があります。起動していない場合にはあらかじめ起動しておいてください。
基盤サービスの起動にはInterstage基盤サービス操作ツールを使用します。ツールを起動するには、スタートメニューから[Interstage] > [Studio] > [Interstage基盤サービス操作ツール]を選択してください。Interstage Application Serverの起動にはInterstage管理コンソールを使用してください。

2. ビルド時にIDLコンパイラでエラーが発生する。

【対処】

Interstage Application Serverサーバ機能と組み合わせて開発を行う場合には環境設定は必要ありませんが、クライアントパッケージと組み合わせて開発を行う場合には環境設定が必要です。
クライアントパッケージと組み合わせる場合には、Interstage Application Serverインストールフォルダ¥odwin¥etc¥inithostファイルに、アクセスするサーバを指定する必要があります。

3. 接続先ホストを変更してもCORBAサーバオブジェクト一覧の表示が更新されない。

【対処】

ワークベンチを実行中に、Interface Repositoryサービスのホストを変更した場合、ワークベンチが起動されている間は、CORBAサーバオブジェクト一覧に変更は反映されません。Interface Repositoryサービスのホスト変更した場合は、ワークベンチを再起動してください。

C.3 ビルドに関する問題

1. CORBAサーバプロジェクトでメインプログラムをデフォルトのファイルから変更すると、ビルドに失敗する。

【対処】

CORBAサーバプロジェクトでは、メインプログラム名はプロジェクト名と一致していなければなりません。メインプログラムをデフォルトのファイルから変更しないでください。

2. Interstage Application Serverが起動していない場合にCORBAサーバプロジェクトのビルドを行うと、ビルドに失敗する場合がある。

コンソールビューに以下のエラーが表示されている。

```
OD: エラー: od51401:IDLparser:CORBA_Container_lookup関数で例外情報にIDL:CORBA/StExcep/COMM_FAILURE:1.0が発生しました。
マイナーコードは0x464a010aです。
OD: エラー: od51221:IDLc:IDLparserコマンドエラー. エラーコード=4
```

【対処】

IDLファイルを更新している場合、Interstage Application Serverが強制停止状態の時にはIDLコンパイラの実行がエラーとなるため、ビルドに失敗します。このような場合には、Interstage管理コンソールを用いてInterstage Application Serverを起動してから、再度ビルドを実行してください。

3. CORBAサーバプロジェクトの初回ビルド時に、インタフェースリポジトリからの削除失敗を示す警告メッセージが出力される。

コンソールビューに以下のエラーが表示されている。

```
OD: 警告: od51006:IDLinst:削除すべきオブジェクト::SAMPLEが存在しません。
```

【対処】

IDLコンパイラの設定画面で[インタフェースリポジトリへの登録を行う]がチェックされている場合、IDLコンパイラは、まずインタフェースリポジトリから既存の登録情報を削除し、その後に新しく情報を登録します。初回ビルド時には、まだインタフェースリポジトリに情報が登録されていないため、登録情報の削除に失敗したことを示す警告メッセージが表示されます。この警告メッセージが表示されてもビルドは問題なく実行できます。

付録D　旧資産からの移行

ここではCOBOLプラグイン固有の旧資産の移行について説明します。その他の移行の手順については、ヘルプの"Interstage Studio ユーザーズガイド(互換ワークベンチ)"の"付録D 旧資産からの移行"を参照してください。

D.1 コンポーネントデザイナからの移行について

COBOL言語で記述されたCORBAサーバプロジェクトをワークベンチに移行するには

COBOL言語で記述されたCORBAサーバプロジェクトは、以下の手順でワークベンチに移行できます。

1. 空のCORBAサーバプロジェクトを新規に作成します。

   メニューバーから[ファイル] > [新規] > [プロジェクト]を選択し、[新規プロジェクト]ダイアログボックスの[COBOL] > [CORBAサーバプロジェクト]を選択してプロジェクトを作成します。プロジェクト名およびターゲット名は移行元のプロジェクト名と同じ名前に設定します。生成するコードを選択する画面では、[コード生成は行わない]を選択して[終了]をクリックします。

2. 作成したプロジェクトに翻訳オプションおよびリンクオプションを設定します。

   移行元のプロジェクトで翻訳オプション、登録集名、リンクオプションおよびIDLコンパイラのオプションを独自に設定していた場合には、プロジェクトのプロパティ画面で同じ設定を行います。

3. 移行元のプロジェクトからソースファイルをコピーします。

   コンポーネントデザイナで移行元のプロジェクトを開き、プロジェクト表示域に表示されているソースファイル（COBOLソースファイルおよびIDLファイル）を確認します。そこに表示されているソースファイルを移行元のプロジェクトのフォルダから移行先のプロジェクトのフォルダにコピーします。ソースファイルの確認後はコンポーネントデザイナを終了してかまいません。

   - 移行元：<旧製品のインストールフォルダ>¥APW¥Projects¥<プロジェクト名>
   - 移行先：<ユーザのドキュメントフォルダ>¥Interstage Studio¥<製品バージョン>¥workspace¥<プロジェクト名>

   注意

   - 移行元のフォルダ内にはプロジェクト表示域に表示されている以外のファイルも存在しますが、それらのファイルはコピーする必要はありません。
   - ワークベンチのワークスペースフォルダをデフォルトの位置から変更している場合には、そのワークスペースフォルダ配下の（プロジェクト名）フォルダが移行先になります。

4. コピーしたファイルをプロジェクトに追加します。

   依存ビューでプロジェクトを選択し、コンテキストメニューから[最新表示]を実行します。コピーしたファイルが[その他のファイル]フォルダに追加されます。

5. コピーしたファイルを[ソースファイル]フォルダに移動します。

   コピーしたファイルを[その他のファイル]フォルダの中で選択し、コンテキストメニューから[ソースファイルへ追加]を実行します。ファイルが[その他のファイル]フォルダから[ソースファイル]フォルダに移動します。

6. メインプログラムを設定します。

   [ソースファイル]フォルダを開き、その中から（プロジェクト名）.cobというファイルを選択し、コンテキストメニューから[メインプログラム]を選択します。

7. IDLファイルを翻訳します。

   [ソースファイル]フォルダの中にあるIDLファイルを選択し、コンテキストメニューから[ファイルの翻訳]を実行します。なお、IDLファイルの翻訳にはあらかじめInterstage Application ServerおよびInterstageの管理に必要なサービス（基盤サービス）を起動しておく必要があります。起動していない場合にはあらかじめ起動しておいてください。

   基盤サービスの起動にはInterstage基盤サービス操作ツールを使用します。ツールを起動するには、スタートメニューから[Interstage] > [Studio] > [Interstage基盤サービス操作ツール]を選択してください。Interstage Application Serverの起動にはInterstage管理コンソールを使用してください。

8. スケルトンファイルを[ソースファイル]フォルダに移動します。

   IDLファイルが翻訳されると、[その他のファイル]フォルダにスタブ・スケルトンとなるCOBOLソースファイルおよびCOBOL登録集ファイルが生成されます。この中でスケルトンとなるファイルを[ソースファイル]フォルダに移動します。以下のファイルを先と同じ手順で[ソースファイル]フォルダに移動します。

   - インタフェースファイル：（モジュール名）-（インタフェース名）.cob
   - Helperクラスファイル：（モジュール名）-（インタフェース名）--HELPER.cob
   - Narrowスケルトンファイル：（モジュール名）-（インタフェース名）_NARROW.cob
   - tieクラスファイル：（モジュール名）-（インタフェース名）--TIE.cob
   - インプリメンテーション登録ファイル：（モジュール名）-（インタフェース名）--NEW.cob
   - データ型クラスファイル：（データ型名）.cob
   - データ型Helperファイル：（データ型名）--HELPER.cob

   注意

   登録集ファイル（*.cbl）およびスタブ用のファイル（*--stub.cob、*--NarrowStub.cob）は[ソースファイル]フォルダに移動しないでください。

付録E　名前に関する規則

COBOLプラグイン固有の名前に関する規則について説明します。

名前の種類	名前の長さ	拡張子	命名規約
COBOLソース名	ファイルのフルパスで255バイト以内	cob、cbl、cobol	ヘルプの"Interstage Studio ユーザーズガイド(互換ワークベンチ)"の"付録A 名前に関する規則"のファイル名の規約を参照してください。
IDLファイル名		idl	

索　引