BEA Tuxedo

BEA Tuxedo CORBA ネーム・サービス

BEA Tuxedo リリース 8.0
8.0 版
2001 年 10 月 31 日

BEA Tuxedo CORBA ネーム・サービス

Document Edition	Date	Software Version
8.0	2001 年 10 月 31 日	BEA Tuxedo 8.0

目次

5. CORBA Name Service サンプル・アプリケーションの使用

索引

このマニュアルについて

このマニュアルでは、BEA Tuxedo CORBA ネーム・サービスの使い方について説明します。

このマニュアルでは、以下の内容について説明します。

- 「第 1 章 CORBA ネーム・サービスの概要」では、BEA Tuxedo CORBA ネーム・サービスの機能および概念について説明します。
- 「第 2 章 CORBA ネーム・サービス・リファレンス」では、BEA Tuxedo CORBA ネーム・サービスのコマンドおよびアプリケーション・プログラミング・インターフェイス (API) について説明します。
- 「第 3 章 BEA Tuxedo 名前空間の管理」では、BEA Tuxedo CORBA ネーム・サービスに関連する管理タスクについて説明します。
- 「第 4 章 CORBA ネーム・サービスを使用するアプリケーションの開発」では、オブジェクトのリファレンスを格納するために名前空間を使用する BEA Tuxedo アプリケーションを開発する方法について説明します。
- 「第 5 章 CORBA Name Service サンプル・アプリケーションの使用」では、CORBA Name Service サンプル・アプリケーションをビルドおよび実行する方法について説明します。

対象読者

このマニュアルは、BEA Tuxedo 製品を使用してアプリケーションを開発し、ネーム・サービス機能を使用するプログラマを対象としています。

e-docs Web サイト

BEA Tuxedo 製品のマニュアルは BEA 社の Web サイトで参照することができます。BEA ホーム・ページの [製品のドキュメント] をクリックするか、または http://edocs.beasys.co.jp/e-docs/index.html に直接アクセスしてください。

マニュアルの印刷方法

このマニュアルは、ご使用の Web ブラウザで一度に 1 ファイルずつ印刷できます。Web ブラウザの [ファイル] メニューにある [印刷] オプションを使用してください。

このマニュアルの PDF 版は、Web サイト上にあります。また、マニュアルの CD-ROM にも収められています。この PDF を Adobe Acrobat Reader で開くと、マニュアル全体または一部をブック形式で印刷できます。PDF 形式を利用するには、BEA Tuxedo マニュアルのホーム・ページにある [PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をお持ちでない場合は、Adobe 社の Web サイト (http://www.adobe.co.jp/) から無償でダウンロードできます。

関連情報

CORBA、BEA Tuxedo、分散オブジェクト・コンピューティング、トランザクション処理、C++ プログラミング、および Java プログラミングの詳細については、BEA Tuxedo オンライン・マニュアルの「Bibliography」を参照してください。

サポート情報

皆様の BEA Tuxedo マニュアルに対するフィードバックをお待ちしています。ご意見やご質問がありましたら、電子メールで **docsupport-jp@bea.com** までお送りください。お寄せいただきましたご意見は、BEA Tuxedo マニュアルの作成および改訂を担当する BEA 社のスタッフが直接検討いたします。

電子メール メッセージには、BEA Tuxedo 8.0 リリースのマニュアルを使用していることを明記してください。

BEA Tuxedo に関するご質問、または BEA Tuxedo のインストールや使用に際して問題が発生した場合は、www.bea.com の BEA WebSUPPORT を通して BEA カスタマ・サポートにお問い合わせください。カスタマ・サポートへの問い合わせ方法は、製品パッケージに同梱されているカスタマ・サポート・カードにも記載されています。

カスタマ・サポートへお問い合わせの際には、以下の情報をご用意ください。

- お客様のお名前、電子メール・アドレス、電話番号、Fax 番号
- お客様の会社名と会社の住所
- ご使用のマシンの機種と認証コード
- ご使用の製品名とバージョン
- 問題の説明と関連するエラー・メッセージの内容

表記上の規則

このマニュアルでは、以下の表記規則が使用されています。

規則	項目
太字	用語集に定義されている用語を示します。
Ctrl + Tab	2 つ以上のキーを同時に押す操作を示します。

規則	項目
イタリック体	強調またはマニュアルのタイトルを示します。

等幅テキスト	コード・サンプル、コマンドとオプション、データ構造とメンバ、データ型、ディレクトリ、およびファイル名と拡張子を示します。また、キーボードから入力するテキストも等幅テキストで表示します。
	例 :
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な語を示します。 例 : `void **commit** ( )`
斜体の等幅テキスト	コード内の変数を示します。 例 : `String *expr*`
大文字のテキスト	デバイス名、環境変数、および論理演算子を示します。 例 : LPT1 SIGNON OR
`{ }`	構文の行で、選択肢の組み合わせを示します。かっこは入力しません。
`[ ]`	構文の行で、オプション項目を示します。かっこは入力しません。 例 : `buildobjclient [-v] [-o *name* ] [-f *file-list*]...[-l *file-list*]...`

規則	項目
|	構文の行で、相互に排他的な選択肢の区切りとして使います。記号は入力しません。
. . .	コマンド・ラインで、以下のいずれかの場合を示します。 ■ コマンド・ラインで、同じ引数を繰り返し使用できることを示します。 ■ 文で、追加のオプション引数が省略されていることを示します。 ■ 追加のパラメータ、値、またはその他の情報を入力できることを示します。 記号は入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]...[-l file-list]...`
. . .	コード例または構文の行で、項目が省略されていることを示します。記号は入力しません。

1　CORBA ネーム・サービスの概要

ここでは、次の内容について説明します。

- CORBA ネーム・サービス
- CORBA ネーム・サービスについて

CORBA ネーム・サービス

BEA Tuxedo ネーム・サービス (以降 CORBA ネーム・サービスと呼びます) を使用すると、BEA Tuxedo CORBA サーバ・アプリケーションは、論理名でオブジェクト・リファレンスを宣言できるようになります。BEA Tuxedo CORBA クライアント・アプリケーションは、CORBA ネーム・サービスに名前の検索を依頼することによってオブジェクトをロケートできます。

CORBA ネーム・サービスは、以下の機能を提供します。

- Object Management Group (OMG) のインターオペラブル・ネーム・サービス (INS) 仕様の実装
- オブジェクト・リファレンスを階層的な命名構造 (名前空間) にマッピングするためのアプリケーション・プログラミング・インターフェイス (API)
- バインディングを表示し、ネーミング・コンテキスト・オブジェクトとアプリケーション・オブジェクトを名前空間にバインドおよびアンバインドするためのコマンド

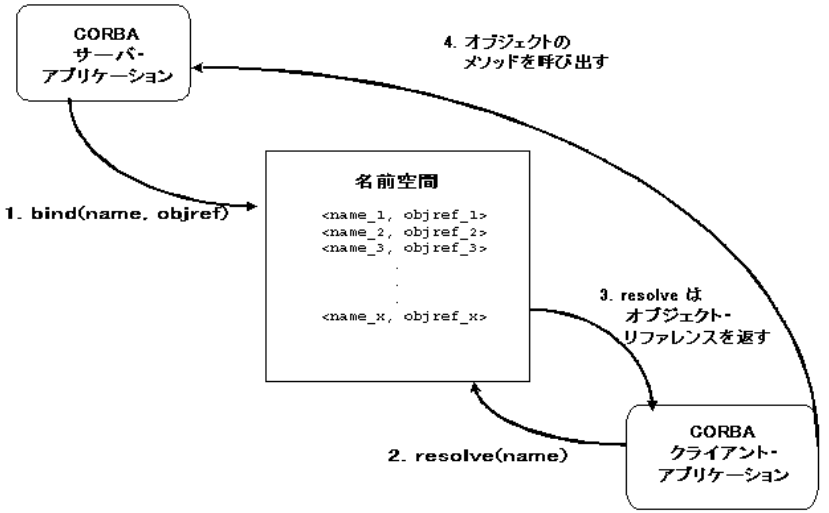
CORBA ネーム・サービスはレイヤ化された製品です。CORBA ネーム・サービスは、BEA Tuxedo 製品の一部としてインストールされます。サポートされているプラットフォームおよびインストール手順の詳細については、『BEA Tuxedo システムのインストール』を参照してください。

CORBA ネーム・サービスを使用する場合

1. BEA Tuxedo CORBA サーバ・アプリケーションは、名前空間内のアプリケーション・オブジェクトまたはネーミング・コンテキスト・オブジェクトのいずれか 1 つに名前をバインドします。
2. BEA Tuxedo CORBA クライアント・アプリケーションは、名前空間を使用して名前を解決し、アプリケーション・オブジェクトまたはネーミング・コンテキスト・オブジェクトのオブジェクト・リファレンスを取得できるようになります。

図 1-1 では、CORBA ネーム・サービスの概要を示します。

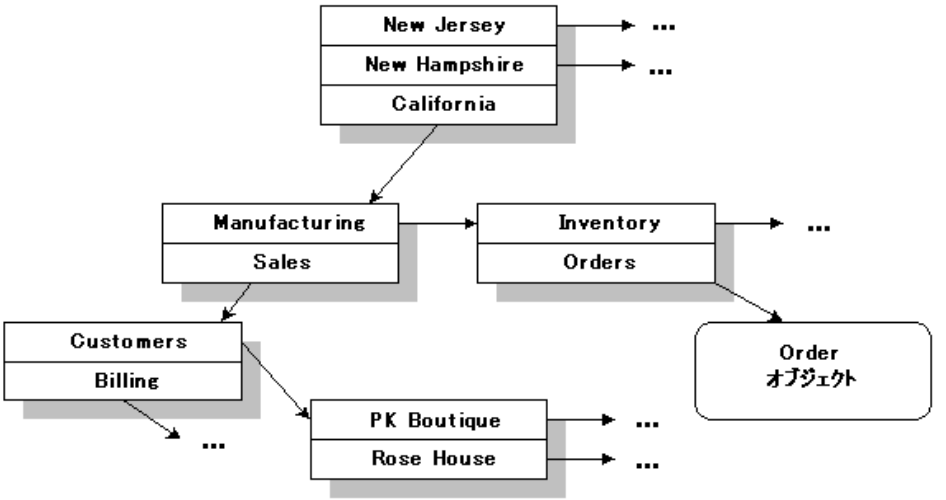
図 1-1　CORBA ネーム・サービス



CORBA ネーム・サービスについて

図 1-2 では、発注入力アプリケーションを構成するオブジェクトの格納に名前空間がどのように使用されるかを示します。

図 1-2 BEA Tuxedo 名前空間



図で示されたアプリケーションは、地域ごと、次に部門ごとに名前空間を構成します。CORBA ネーム・サービスでオブジェクトを使用して名前空間をインプリメントするために、影付きの四角い部分はそれぞれ NamingContext オブジェクトによってインプリメントされます。NamingContext オブジェクトは、アプリケーション・オブジェクトまたはその他の NamingContext オブジェクトにバインドされている CosNaming::Name データ構造体のリストを格納します。NamingContext オブジェクトをたどって、特定の名前をロケートします。たとえば、論理名 California.Manufacturing.Order を使用して、Order オブジェクトをロケートできます。

CosNaming::Name データ構造体は、英数字からなる文字列だけではありません。1 つまたは複数の CosNaming::NameComponent データ構造体のシーケンスです。各 CosNaming::NameComponent データ構造体は、id および kind の 2 種類の文字列を格納します。CORBA ネーム・サービスは、この文字列を解釈または管理しません。ただ、指定した NamingContext オブジェクト内で各 ID を固有のものにします。

BEA Tuxedo CORBA サーバ・アプリケーションは、NamingContext オブジェクトの bind() メソッドを使用して、サーバ・アプリケーションに含まれるアプリケーション・オブジェクトに名前をバインドします。BEA Tuxedo CORBA サーバ・アプリケーションは、NamingContext オブジェクトの resolve メソッドを使用して、バインドを使用するオブジェクトをロケートします。

また、CORBA ネーム・サービスは、BindingIterator オブジェクトと NamingContextExt オブジェクトも提供します。BindingIterator オブジェクトを使用すると、クライアント・アプリケーションは、各呼び出しで指定した数のバインドを取得できます。NamingContextExt オブジェクトは、URL (Uniform Resource Locator) と文字列化された名前を使用する方法を提供します。

CORBA ネーム・サービスとそのインターフェイスの詳細については、「第 2 章 CORBA ネーム・サービス・リファレンス」を参照してください。

2 CORBA ネーム・サービス・リファレンス

ここでは、次の内容について説明します。

- CORBA ネーム・サービスのコマンド
- CORBA ネーム・サービスの機能と制限事項
- NameService 環境オブジェクトへの初期リファレンスの取得
- CORBA ネーム・サービスで使用する CosNaming データ構造体
- NamingContext オブジェクト
- NamingContextExt オブジェクト
- BindingIterator オブジェクト
- CORBA ネーム・サービスで発生する例外

CORBA ネーム・サービスのコマンド

CORBA ネーム・サービスは、CORBA ネーム・サービスのサーバ・プロセスを管理し、名前空間内の名前にオブジェクトをバインドまたはアンバインドし、名前空間の内容を表示するために以下のコマンドを提供します。

- `cns`
- `cnsbind`
- `cnsls`
- `cnsunbind`

以下の節では、これらのコマンドについて説明します。

cns

概要 CORBA ネーム・サービスのサーバ・プロセスを制御します。

構文

```
cns CLOPT="[-A] [servopts options] --
     [-b bucketcount]
     [-c]
     [-d]
     [-f filename]
     [-M maxiterators]
     [-p [persiststoragefilename] ]"
```

説明 CORBA ネーム・サービスのサーバ・プロセスは、CORBA CosNaming 準拠のネーム・サービスを提供します。使用する BEA Tuxedo アプリケーションに対して、その BEA Tuxedo アプリケーションで使用するほかのサーバ・プロセスと同様に、CORBA ネーム・サービスのサーバ・プロセスとそのオプションを UBBCONFIG ファイルで定義する必要があります。UBBCONFIG ファイルの CLOPT パラメータで、ダブル・ハイフン (--) の後に、cns コマンド行オプションを入力します。コマンド行オプションには、以下のものがあります。

-b *bucketcount*

ネーミング・コンテキストをメモリ内にロケートするために、サーバ・プロセスが内部で使用するハッシュ・テーブルのバケット数を指定します。各ネーミング・コンテキストには、専用のハッシュ・テーブルがあります。BEA Tuxedo アプリケーションが各ネーミング・コンテキストでより少ない数のバインディングを使用する場合、小さなバケット数 (4 または 5 など) を使用します。BEA Tuxedo アプリケーションが各ネーミング・コンテキストでより大きな数 (1,000 など) のバインディングを使用する場合、大きなバケット数 (50 など) を使用します。

-c

CORBA ネーム・サービスのサーバ・プロセスが開始されたときに、永続ストレージ・ファイルを圧縮します。ネーミング・コンテキストおよびアプリケーション・オブジェクトが名前空間に追加され、名前空間から削除されるたびに、永続ストレージ・ファイルのサイズは増え続けます。圧縮によって、永続ストレージ・ファイルのサイズを可能な限り小さくすることができます。バインドされていないバインディングは圧縮時に削除されます。バインドされていないバインディングは、バインディングが関連付けられているオブジェクトが名前空間から削除されても名前空間内に残ります。-p コマンド行オプションは、-c コマンド行オプションの指定時に指定する必要があります。

-d

CORBA ネーム・サービスのサーバ・プロセスで、サーバ・プロセスの開始時に、関連付けのないコンテキストが削除されるようにします。関連付けのないコンテキストは、ほかのどのオブジェクトにもバインドされていないコンテキストです。一度もバインドされていない可能性もありますし、コンテキストにバインドされた後で明示的に、または再バインドの影響で破棄された可能性もあります。-p コマンド行オプションは、-d コマンド行オプションの指定時に指定する必要があります。

-f *filename*

CORBA ネーム・サービスのサーバ・プロセスが、名前空間のルートのインターオペラブル・オブジェクト・リファレンス (IOR) を書き込む先となるファイルを指定します。

-M *maxiterators*

任意の時点での未処理のバインディング・イテレータの最大数を定義します。

バインディング・イテレータは、クライアント・アプリケーションが CosNaming::NamingContext::list() メソッドを使用したときに作成されます。クライアント・アプリケーションは、使用を完了したバインディング・イテレータを削除するために、CosNaming::BindingIterator::destroy() メソッドを使用する必要があります。

クライアント・アプリケーションがバインディング・イテレータを特に削除しなかった場合、CORBA ネーム・サービスのサーバ・プロセスは、-M コマンド行オプションで指定した数に達したときにバインディング・イテレータを削除します。バインディング・イテレータが最大数に達してから新しいバインディング・イテレータを作成しようとすると、CORBA ネーム・サービスのサーバ・プロセスは、クライアント・アプリケーションで使用中のバインディング・イテレータを破棄します。

バインディング・イテレータは、使用履歴の最も古いアルゴリズムを基に削除されます。デフォルト値は 20 です。値を 0 にすると、バインディング・イテレータの数の制約がないことを示します。つまり、CORBA ネーム・サービスのサーバ・プロセスはバインディング・イテレータを破棄せず、関連付けられているメモリは解放されません。0 を指定した場合、クライアント・アプリケーションは、CosNaming::BindingIterator::destroy() メソッドを使用して、未使用のバインディング・イテレータを明示的に削除する必要があります。

-p [*persistentstoragefilename*]

CORBA ネーム・サービスのサーバ・プロセスで、指定したファイルを使用して現在の名前空間のコピーが永続ストレージに保存されるようにします。ファイル名を指定しなかった場合、CNS_PERSIST_FILE 環境変数の値が使用されます。CNS_PERSIST_FILE 環境変数を設定しなかった場合、以下のファイルが使用されます。

Windows

```
%APPDIR%\cnspersist.dat
```

UNIX

```
$APPDIR/cnspersist.dat
```

永続ストレージ・ファイルは、CORBA ネーム・サービスのサーバ・プロセスが開始されたときに読み取られます。永続ストレージ・ファイルは、名前空間に変更が加えられるたびに追加されます。新しい名前空間を作成する場合、既存の永続ストレージ・ファイルを削除するか、CORBA ネーム・サービスのサーバ・プロセスに対して新しい永続ストレージ・ファイルを作成する必要があります。

cnsbind

概要 アプリケーション・オブジェクトおよびネーミング・コンテキストを名前空間にバインドします。

注記 cnsbind コマンドは、CosNaming インターフェイスとやり取りします。このコマンドを使用して CORBA ネーム・サービスのサーバ・プロセスを実行する必要があります。

構文

```
cnsbind
    [-C]
    [-f root_context_filename]
    [-h]
    [-N]
    [-o ior_filename]
    [-r]
    [-T TObjAddr]
    bind_name
```

説明 cnsbind コマンドは、CORBA CosNaming インターフェイスを使用して新しいアプリケーション・オブジェクトとネーミング・コンテキスト・オブジェクトを名前空間にバインドします。このコマンドを使用すると、連合型名前空間を簡単に作成できるようになります。cnsbind コマンドが呼び出されたときに例外が返されると、このコマンドは終了し、対応するメッセージが表示されます。

cnsbind コマンドのコマンド行オプションは以下のとおりです。

-C

cnsbind コマンドが、*bind_name* (名前)、および -o コマンド行オプションで指定した *ior_filename* を使用してコンテキストを作成するように指定します。-C コマンド行オプションを使用して、ある名前空間のネーミング・コンテキスト・オブジェクトを、指定した名前空間に連合させます。

-f *root_context_filename*

名前空間の内容を変更するためにコマンドがやり取りする、CORBA ネーム・サービスのサーバ・プロセスの IOR を格納するファイルを指定します。このコマンド行オプションを指定しない場合、コマンドは、NameService 環境オブジェクトで Tobj_Bootstrap::resolve_initial_references() メソッドを使用し、指定した BEA Tuxedo ドメインに対する CORBA ネーム・サービスのサーバ・プロセスをロケートします。IOR のホストとポートは、TOBJADDR の値と一致していなければなりません。コマンド行オプショ

ンは、TOBJADDR 環境変数の設定をオーバーライドします。コマンド行オプションを指定しなかった場合、TOBJADDR 環境変数が使用されます。

-h

コマンドの構文を出力します。

-N

新しいコンテキストを作成し、指定した名前を使用する名前空間にそれをバインドします。cnsbind コマンドが新しいコンテキストを作成するので、-N コマンド行オプションとともに -o コマンド行オプションを使用する必要はありません。-N コマンド行オプションとともに -o コマンド行オプションを使用しても、-o コマンド行オプションの情報は無視されます。

-o *ior_filename*

-f コマンド行オプションで指定した名前空間にバインドするオブジェクトの IOR を格納するファイルを指定します。-C コマンド行オプションを指定した場合は ncontext 型のオブジェクトが作成され、指定しなかった場合は nobject 型のオブジェクトが作成されます。

-r

名前に既にバインディングがある場合でも、アプリケーション・オブジェクトまたはネーミング・コンテキスト・オブジェクトのバインディングを作成します。-r コマンド行オプションを指定しない cnsbind コマンドのデフォルトの動作では、指定したオブジェクトのバインディングが既に存在する場合に AlreadyBound 例外が発生します。cnsbind コマンドが呼び出されたときに AlreadyBound その他の例外が返された場合、コマンドは終了し、「Error, already bound」というメッセージが表示されます。

-T *TObjAddr*

BEA Tuxedo ドメイン用のホストとポートを指定します。CORBA ネーム・サービスのサーバ・プロセスに接続する前に、cnsbind コマンドは、サーバ・プロセスが実行される BEA Tuxedo ドメインにログインする必要があります。コマンド行オプションは、TOBJADDR 環境変数の設定をオーバーライドします。コマンド行オプションを指定しなかった場合、TOBJADDR 環境変数の値が使用されます。*TObjAddr* 指定の有効な形式は、*//hostname:port_number* です。

bind_name

Tobj_Bootstrap::resolve_initial_references メソッドから取り出したルート・ネーミング・コンテキスト、または -f コマンド行オプ

ションから取得した文字列の IOR に示されているネーミング・コンテキスト・オブジェクトを基に名前空間に追加されたアプリケーション・オブジェクトまたはネーミング・コンテキスト・オブジェクトにバインドする名前を指定します。*bind_name* 文字列は、Object Management Group (OMG) インターオペラブル・ネーム・サービス (INS) 仕様で指定した名前の文字列の形式に従っている必要があります。

例 次の例は、アプリケーション・オブジェクトのバインディングを示しています。

```
cnsbind -o ./app_obj_ior.txt MyContext/AppObject1
```

次の例は、ネーミング・コンテキスト・オブジェクトのバインディングを示しています。

```
cnsbind -N MyContext/CtxObject1
```

次の例は、フェデレーション・ポイントのほかの名前空間へのバインディングを示しています。

```
cnsbind -C -o ./remote_ior.txt MyContext/RemoteNSCtx1
```

cnsls

概要 名前空間の内容を表示します。

注記 cnsls コマンドは、CosNaming インターフェイスとやり取りします。このコマンドを使用して CORBA ネーム・サービスのサーバ・プロセスを実行する必要があります。

構文

```
cnsls
    [-f root_context_filename]
    [-h]
    [-s]
    [-R]
    [-T TobjAddr]
    [resolve_name]
```

説明 cnsls コマンドは、CORBA CosNaming インターフェイスを使用して名前空間の内容を表示します。非出力文字を NameComponent データ構造の一部として使用する場合、cnsls コマンドの動作は未定義です。cnsls コマンドが呼び出されたときに例外が返されると、このコマンドは終了し、対応するメッセージが表示されます。

cnsls コマンドのコマンド行オプションは以下のとおりです。

-f *root_context_filename*
: 名前空間の内容を変更するためにコマンドがやり取りする、CORBA ネーム・サービスのサーバ・プロセスの IOR を格納するファイルを指定します。このコマンド行オプションを指定しない場合、コマンドは、NameService 環境オブジェクトで Tobj_Bootstrap::resolve_initial_references() メソッドを使用し、指定した BEA Tuxedo ドメインに対する CORBA ネーム・サービスのサーバ・プロセスをロケートします。IOR のホストとポートは、*TOBjAddr* の値と一致していなければなりません。コマンド行オプションは、TOBJADDR 環境変数の設定をオーバーライドします。コマンド行オプションを指定しなかった場合、TOBJADDR 環境変数の値が使用されます。

-h
: コマンドの構文を出力します。

-S
: *resolve_name* コマンド行オプションで指定した名前空間の名前の文字列化された IOR を表示します。

-R

resolve_name で始まる名前空間のバインディングを再帰的に表示します。このコマンド行オプションによって、cnsls コマンドは、フェデレーションの境界が交差している場合にも何も示されずに境界を越える可能性があります。また、名前空間の情報に循環が存在する場合、このコマンド行オプションによって cnsls コマンドがループに入る可能性もあります。

-T *TObjAddr*

BEA Tuxedo ドメイン用のホストとポートを指定します。CORBA ネーム・サービスのサーバ・プロセスに接続する前に、cnsls コマンドは、サーバ・プロセスが実行される BEA Tuxedo ドメインにログインする必要があります。コマンド行オプションは、TOBJADDR 環境変数の設定をオーバーライドします。コマンド行オプションを指定しなかった場合、TOBJADDR 環境変数が使用されます。

resolve_name

Tobj_Bootstrap::resolve_initial_references メソッドを介して取り出したルート・ネーミング・コンテキスト、または -f コマンド行オプションから取得した文字列の IOR に示されているネーミング・コンテキスト・オブジェクトを基にしたネーム・サービスで解決する名前を指定します。*resolve_name* 文字列は、OMG INS 仕様で指定した名前の文字列の形式に従っている必要があります。名前コンポーネントの区切りにはバックスラッシュ (\)、id フィールドと kind フィールドの区切りにはピリオド (.) を使用します。

このコマンド行オプションを指定しなかった場合、ルート・コンテキストが解決されます。

例

```
cnsls -R MyContext.kind/AnotherContext
[context] MyContext.kind/AnotherContext
    [object] Obj1
    [object] Obj2
    [context] Ctx1
            [object] AnotherObject
```

cnsunbind

概要 名前空間のバインディングを解除します。

構文

```
cnsunbind
    [-D]
    [-f root_context_filename]
    [-h]
    [-T TObjAddr]
    bind_name
```

説明 cnsunbind コマンドは、名前空間のバインディングを解除します。cnsunbind コマンドが呼び出されたときに例外が返されると、このコマンドは終了し、対応するメッセージが表示されます。

cnsunbind コマンド行オプションは、以下のとおりです。

`-D`
: バインディングを解除した後に *bind_name* にバインドされているネーミング・コンテキストを破棄します。コンテキストを破棄するときに -D コマンド行オプションを指定しても、コンテキストがほかのバインディングの一部になっている場合は孤立化されません。同時に複数のネーミング・コンテキスト・オブジェクトにバインドされた場合などにバインディングが保留される可能性があるので、このコマンド行オプションの使用には注意する必要があります。

`-f root_context_filename`
: 名前空間の内容を変更するためにコマンドがやり取りする、CORBA ネーム・サービスのサーバ・プロセスの IOR を格納するファイルを指定します。このコマンド行オプションを指定しない場合、コマンドは、NameService 環境オブジェクトで Tobj_Bootstrap::resolve_initial_references() メソッドを使用し、指定した BEA Tuxedo ドメインに対するサーバ・プロセスをロケートします。

`-h`
: コマンドの構文を出力します。

`-T TObjAddr`
: BEA Tuxedo ドメイン用のホストとポートを指定します。CORBA ネーム・サービスのサーバ・プロセスに接続する前に、cnsbind コマンドは、サーバ・プロセスが実行される BEA Tuxedo ドメインにログインする必要があります。コマンド行オプションは、TOBJADDR 環境変数の設

定をオーバーライドします。コマンド行オプションを指定しなかった場合、TOBJADDR 環境変数が使用されます。

bind_name

Tobj_Bootstrap::resolve_initial_references() メソッドから取り出したルート・ネーミング・コンテキスト、または -f コマンド行オプションから取得した文字列の IOR に示されているネーミング・コンテキストを基にした名前空間から解除するバインディングの名前を指定します。*bind_name* 文字列は、OMG INS 仕様で指定した名前の文字列の形式に従っている必要があります。

例 次の例は、名前空間からのバインディングの解除を示しています。

```
cnsunbind MyContext/CtxObject1
```

次の例は、名前空間からのバインディングの解除、およびバインドされているオブジェクトの破棄を示しています。

```
cnsunbind -D MyContext/CtxObject1
```

CORBA ネーム・サービスの機能と制限事項

CORBA ネーム・サービスには、以下の機能と制限があります。

- NULL 文字は、id 文字列と kind 文字列の終端としてのみ使用します (空の文字列は有効と見なされます)。
- ワイド文字はサポートされていません。
- CORBA ネーム・サービスは、名前コンポーネントの文字列の長さに制限を設けていません。
- CORBA ネーム・サービスは、名前内のコンポーネント数に制限を設けていません。0 は不正な値です。
- CORBA ネーム・サービスは、コンテキスト内のバインディング数に制限を設けていません。
- CORBA ネーム・サービスは、インプリメンテーション全体のバインディング数に制限を設けていません。

- CORBA ネーム・サービスは、コンテキスト数に制限を設けていません。
- CORBA ネーム・サービスは、サーバ・プロセスの開始時に、関連付けのないネーミング・コンテキストとバインドされていないバインディングを削除します。
- CORBA ネーム・サービスは、サーバ・プロセスの開始時に、関連付けのないネーミング・コンテキストを削除します。
- CORBA ネーム・サービスでは、必要に応じて、使用履歴の最も古いアルゴリズムを基にバインディング・イテレータをクリーンアップできます。詳細については、第 3 章の 9 ページ「バインディング・イテレータの管理」を参照してください。
- CORBA ネーム・サービスは、`CannotProceed` 例外をスローしません。

NameService 環境オブジェクトへの初期リファレンスの取得

`NameService` 環境オブジェクトは、名前空間のルートへの接続に使用できます。`NameService` 環境オブジェクトを使用する場合、オブジェクト・リクエスト・ブローカ (ORB) が名前空間のルートをロケートします。Bootstrap オブジェクトまたは CORBA インターオペラブル・ネーミング・サービス (INS) のブートストラップ処理メカニズムを使用すると、NameService 環境オブジェクトへの初期リファレンスを取得できます。BEA クライアント ORB を使用する場合は、BEA 社固有のメカニズムを使用します。別のベンダのクライアント ORB を使用する場合は、CORBA INS メカニズムを使用します。

名前空間への接続の詳細については、第 4 章の「ステップ 3: BEA Tuxedo 名前空間への接続」を参照してください。BEA Tuxedo ドメインのブートストラップ処理の詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA プログラミング・リファレンス』の「第 4 章 CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

CORBA ネーム・サービスで使用する CosNaming データ構造体

CORBA ネーム・サービスは、以下の CosNaming データ構造体を使用します。

- `CosNaming::BindingList`
- `CosNaming::BindingType`
- `CosNaming::Istring`
- `CosNaming::Name`
- `CosNaming::NameComponent`

NamingContext オブジェクト

`NamingContext` オブジェクトは、オブジェクト・リクエスト・ブローカ (ORB) オブジェクトまたはほかの `NamingContext` オブジェクトにバインドされている名前のリストを格納および操作するのに使用します。BEA Tuxedo CORBA クライアント・アプリケーションは、このインターフェイスを使用して、コンテキスト内のすべての名前を解決またはリストします。BEA Tuxedo CORBA サーバ・アプリケーションは、このオブジェクトを使用して、アプリケーション・オブジェクトまたはネーミング・コンテキスト・オブジェクトに名前をバインドします。リスト 2-1 は、`NamingContext` オブジェクトの OMG IDL を示しています。

リスト 2-1　NamingContext オブジェクトの OMG IDL

```
module CosNaming {
  interface NamingContext {
    void bind(in Name, in Object obj)
     raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
    void rebind(in Name, in Object obj)
     raises(NotFound, CannotProceed, InvalidName);
    void bind_context(in Name n, in NamingContext nc)
     raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
    void rebind_context(in Name n, in NamingContext nc)
     raises(NotFound, CannotProceed, InvalidName);
```

```
    Object resolve(in Name n)
     raises(NotFound, CannotProceed, InvalidName);
    void unbind(in Name n)
     raises(NotFound, CannotProceed, InvalidName);
    NamingContext new_context
    NamingContext bind_new_context(in Name n)
     raises(NotFound, CannotProceed, InvalidName, AlreadyBound);
    void destroy()
     raises(NotEmpty);
    void list(in unsigned long how_many,
              out BindingList bl,
              out BindingIterator bi);
    }
}
```

CosNaming::NamingContext::bind()

概要 最初の NameComponent データ構造体に関連付けられているコンテキストを解決し、オブジェクトを新しいコンテキストにバインドすることで、指定したオブジェクトを指定した名前にバインドしようとします。

C++ マッピング

```
void bind(in Name n, in Object obj);
```

Java マッピング

```
void bind (NameComponent [] n, Object obj);
```

パラメータ

n
目的のオブジェクト名で初期化される Name データ構造体。

obj
指定した名前にバインドするオブジェクト。

例外

AlreadyBound
bind() メソッドまたは bind_context() メソッドの Name が、ネーミング・コンテキスト内のほかのオブジェクトに既にバインドされていることを示します。

InvalidName
指定した Name に 0 名前コンポーネントがあるか、最初の名前コンポーネントのいずれか 1 つがネーミング・コンテキストに対して解決されなかったことを示します。

NotFound
Name またはそのコンポーネントのいずれか 1 つが見つからなかったことを示します。

説明 bind にバインドされたネーミング・コンテキストは、複数の名前を解決のために渡すときに名前解決に参加しません。

戻り値 特にありません。

CosNaming::NamingContext::bind_context()

概要 このメソッドは、bind() メソッドとほぼ同じですが、指定した Name が NamingContext オブジェクトに関連付けられています。

C++ マッピング

```
void bind_context(in Name n, in NamingContext nc);
```

Java マッピング

```
void bind_context (NameComponent [] n, NamingContext nc);
```

パラメータ

n

目的のネーミング・コンテキスト名で初期化される Name データ構造体。シーケンス内の最初の NameComponent データ構造体がネーミング・コンテキストに対して解決される必要があります。

nc

指定した名前にバインドする NamingContext オブジェクト。

例外

AlreadyBound

bind() メソッドまたは bind_context() メソッドの Name が、ネーミング・コンテキスト内のほかのオブジェクトに既にバインドされていることを示します。

InvalidName

指定した Name に 0 名前コンポーネントがあるか、最初の名前コンポーネントのいずれか 1 つがネーミング・コンテキストに対して解決されなかったことを示します。

NotFound

Name またはそのコンポーネントのいずれか 1 つが見つからなかったことを示します。

BAD_PARAM

NULL コンテキストをバインドするための呼び出しが試行されたことを示します。

説明 bind_context() にバインドされたネーミング・コンテキストは、複数の名前を解決のために渡すときに名前解決に参加しません。

戻り値 特にありません。

CosNaming::NamingContext::bind_new_context()

概要 新しいコンテキストを作成し、コンテキスト内で指定した Name にバインドします。

C++ マッピング

```
NamingContext bind_new_context(in Name n);
```

Java マッピング

```
bind_new_context (NameComponent [] n);
```

パラメータ

n

新しく作成された NamingContext オブジェクト名で初期化される Name データ構造体。

例外

AlreadyBound

bind() メソッドまたは bind_context() メソッドの Name が、ネーミング・コンテキスト内のほかのオブジェクトに既にバインドされていることを示します。

InvalidName

指定した Name に 0 名前コンポーネントがあるか、最初の名前コンポーネントのいずれか 1 つがネーミング・コンテキストに対して解決されなかったことを示します。

NotFound

Name またはそのコンポーネントのいずれか 1 つが見つからなかったことを示します。

説明 このメソッドは、CosNaming::NamingContext::new_context() メソッドと CosNaming::NamingContext::bind_context() メソッドを 1 つにまとめたものです。

戻り値 新しい NamingContext オブジェクトに対するリファレンスを返します。

CosNaming::NamingContext::destroy()

概要　NamingContext オブジェクトを削除します。削除した後で NamingContext オブジェクトに対する呼び出しを実行すると、CORBA::NO_IMPLEMENT 例外が発生します。

C++ マッピング

```
void destroy();
```

Java マッピング

```
void destroy();
```

パラメータ　特にありません。

例外　NotEmpty

NamingContext オブジェクトにバインディングがある場合、このメソッドで NotEmpty が発生します。

説明　このメソッドを使用する前に、NamingContext　オブジェクトにバインドされているすべてのネーム・オブジェクトに対して、CosNaming::NamingContext::unbind() メソッドでアンバウンドする必要があります。

戻り値　特にありません。

CosNaming::NamingContext::list()

概要　このネーミング・コンテキストによって格納されているすべてのバインディングを返します。

C++ マッピング

```
void list(in unsigned long how_many,
    out BindingList bl,
    out BindingIterator bi);
```

Java マッピング

```
void list(int how_many,
    BindingListHolder bl,
    BindingIteratorHolder bi);
```

パラメータ

how_many
リストに返されるバインディングの最大数。

bl
返されたバインディングのリスト。各要素は、NameComponent オブジェクトを表す Name を含むバインディングです。それぞれの Name は単純な名前です。つまり、長さ 1 の名前のシーケンスです。リスト内のバインディングの数は、*how_many* の値を超えません。

bi
残りのバインディングをたどるための BindingIterator オブジェクトに対するリファレンス。

例外

InvalidName
指定した Name に 0 名前コンポーネントがあるか、最初の名前コンポーネントのいずれか 1 つがネーミング・コンテキストに対して解決されなかったことを示します。

NotFound
Name またはそのコンポーネントのいずれか 1 つが見つからなかったことを示します。

説明　このメソッドは、名前のバインディングのシーケンスを返します。*bl* リストに当てはまるよりも多くの名前のバインディングが存在する場合、BindingIterator オブジェクトが返されます。BindingIterator オブジェクトを使用して、バインディングの次のセットを取得できます。BindingList オブジェクト (C++) または BindingListHolder オブジェクト (Java) は、要求された番号よりも少ないバインディングしかない場合、リストの最後にあるバインディングの番号を返します。*bi* が NULL リファレンスを返した場合、*bl* にはすべてのバインディングが含まれています。

戻り値　特にありません。

CosNaming::NamingContext::new_context()

概要 新しいネーミング・コンテキストを作成します。新しく作成されたコンテキストは、最初はどの Name にもバインドされていません。

C++ マッピング

```
NamingContext new_context();
```

Java マッピング

```
NamingContext new_context();
```

パラメータ 特にありません。

例外 特にありません。

説明 CosNaming::NamingContext::bind_context() メソッドを使用して、新しいネーミング・コンテキストを Name にバインドします。

戻り値 新しいネーミング・コンテキストに対するリファレンスを返します。

CosNaming::NamingContext::rebind()

概要 このメソッドは、bind() メソッドとほぼ同じです。違いは、rebind メソッドでは AlreadyBound 例外が発生しないという点です。指定した Name が既にほかのオブジェクトにバインドされている場合、そのバインディングは新しいバインディングに置き換えられます。

C++ マッピング

```
void rebind(in Name n, in Object obj);
```

Java マッピング

```
void rebind(NameComponent [] n, Object obj);
```

パラメータ

n

目的のオブジェクト名で初期化される Name データ構造体。

obj

命名対象のオブジェクト。

例外

InvalidName

指定した Name データ構造体に 0 名前コンポーネントがあるか、最初の名前コンポーネントのいずれか 1 つがネーミング・コンテキストに対して解決されなかったことを示します。

NotFound

Name またはそのコンポーネントのいずれか 1 つが見つからなかったことを示します。バインディングが既に存在しているか、またはバインディングが不正なタイプのものであるために例外が発生した場合、例外の rest_of_name メンバーには長さ 1 があります。

説明 rebind() メソッドにバインドされたネーミング・コンテキストは、複数の名前を解決のために渡すときに名前解決に参加しません。

戻り値 特にありません。

CosNaming::NamingContext::rebind_context()

概要 このメソッドは、bind_context() メソッドとほぼ同じです。違いは、rebind_context メソッドでは AlreadyBound 例外が発生しないという点です。指定した Name が既にほかのオブジェクトにバインドされている場合、そのバインディングは新しいバインディングに置き換えられます。

C++ マッピング

```
void rebind_context(in Name n, in NamingContext nc);
```

Java マッピング

```
void rebind_context(NameComponent [] n, NamingContext nc);
```

パラメータ

n
目的のオブジェクト名で初期化される Name データ構造体。

nc
再バインド対象の NamingContext オブジェクト。

例外

InvalidName
指定した Name データ構造体に 0 名前コンポーネントがあるか、最初の名前コンポーネントのいずれか 1 つがネーミング・コンテキストに対して解決されなかったことを示します。

NotFound
名前のコンポーネントがバインディングを識別しなかったか、バインディングのタイプが実行中のオペレーションに対して不正だったことを示します。バインディングが既に存在しているか、またはバインディングが不正なタイプのものであるために例外が発生した場合、例外の rest_of_name メンバーには長さ 1 があります。

説明 rebind_context メソッドにバインドされたネーミング・コンテキストは、複数の名前を解決のために渡すときに名前解決に参加しません。

戻り値 特にありません。

CosNaming::NamingContext::resolve()

概要 指定した Name を解決しようとします。

C++ マッピング

```
Object resolve(in Name n);
```

Java マッピング

```
Object resolve (NameComponent n);
```

パラメータ

n

目的のオブジェクト名で初期化される Name データ構造体。

例外

InvalidName

指定した Name データ構造体に 0 名前コンポーネントがあるか、最初の名前コンポーネントのいずれか 1 つがネーミング・コンテキストに対して解決されなかったことを示します。

NotFound

名前のコンポーネントがバインディングを識別しなかったか、バインディングのタイプが実行中のオペレーションに対して不正だったことを示します。

説明 指定した Name は、オブジェクトのバインドに使用する名前と同じものでなければなりません。CORBA ネーム・サービスは、オブジェクトのタイプを返しません。クライアント・アプリケーションでは、オブジェクトを適切なタイプにナロー変換します。

戻り値 指定した Name のオブジェクト・リファレンスを返します。

CosNaming::NamingContext::unbind()

概要 指定した Name に関連付けられたバインディングを削除し、bind() メソッドとは逆のオペレーションを実行します。

C++ マッピング

```
void unbind(in Name n);
```

Java マッピング

```
void unbind (NameComponent [] n);
```

パラメータ

n

目的のオブジェクト名で初期化される Name データ構造体。

例外

InvalidName

指定した Name データ構造体に 0 名前コンポーネントがあるか、最初の名前コンポーネントのいずれか 1 つがネーミング・コンテキストに対して解決されなかったことを示します。

NotFound

名前のコンポーネントがバインディングを識別しなかったか、バインディングのタイプが実行中のオペレーションに対して不正だったことを示します。

説明 このメソッドは、名前とオブジェクトの間のバインディングを削除します。オブジェクトは削除されません。オブジェクトを削除するには、CosNaming::NamingContext::unbind() メソッドを使用してから、CosNaming::NamingContext::destroy() メソッドを使用します。

戻り値 特にありません。

NamingContextExt オブジェクト

NamingContextExt オブジェクトは、CORBA ネーム・サービスで URL と文字列化された名前を使用する方法を提供します。NamingContextExt オブジェクトは、NamingContext オブジェクトから派生したものです。CORBA ネーム・サービスのルートは、NamingContextExt オブジェクトです (つまり、ルートは

NamingContext オブジェクトでもあります)。NamingContextExt オブジェクトのリファレンスを取得するのに特別なオペレーションは必要ありません。リスト 2-2 は、NamingContextExt オブジェクトの OMG IDL を示しています。

リスト 2-2　NamingContextExt オブジェクトの OMG IDL

```
module CosNaming {
       interface NamingContextExt : NamingContext {
              typedef string StringName;
              typedef string Address;
              typedef string URLString;

              StringName to_string(in Name n)
                     raises(InvalidName);
              Name to_name(in StringName sn)
                     raises(InvalidName);

              exception InvalidAddress {};

              URLString to_url(in Address addr, in StringName sn)
                     raises(InvalidAddress, InvalidName);
              Object resolve_str(in StringName n)
                     raises(NotFound,
                            CannotProceed,
                            InvalidName,
                            AlreadyBound);
       }
}
```

CosNaming::NamingContextExt::resolve_str()

概要 文字列化された名前を取り、それを Name に変換し、解決します。

構文
```
object resolve_str(in StringName n);
```

パラメータ *n*

解決対象の文字列化された名前。

例外 InvalidName

名前が無効であることを示します。長さ 0 の名前は無効です。

NotFound

名前のコンポーネントがバインディングを識別しなかったか、バインディングのタイプが実行中のオペレーションに対して不正だったことを示します。

説明 これは、CosNaming:NamingContext::resolve() メソッドと同じ方法で解決を実行する便利なメソッドです。
メソッドは、文字列化された名前を Name オブジェクトではなく引数として受け付けます。文字列化された名前が不正な場合、または文字列化された名前をバインドできなかった場合、メソッドはエラーを返します。

戻り値 バインドされた名前に対するリファレンス。

CosNaming::NamingContextExt::to_name()

概要　文字列化された名前を取り、Name オブジェクトを返します。

構文　`Name to_name(in StringName sn);`

パラメータ　*sn*

Name オブジェクトに対して解決する文字列化された名前。

例外　InvalidName

名前が無効であることを示します。長さ 0 の名前は無効です。

説明　このメソッドは、文字列化された名前を受け付け、Name オブジェクトを返します。名前が不正な場合、メソッドはエラーを返します。

戻り値　Name オブジェクトを返します。

CosNaming::NamingContextExt::to_string()

概要 Name オブジェクトを受け付け、文字列化された名前を返します。

構文
```
StringName to_string(in Name n);
```

パラメータ *n*

文字列化された名前に変換する Name オブジェクト。

例外 InvalidName

名前が無効であることを示します。長さ 0 の名前は無効です。

説明 このメソッドは、Name オブジェクトを受け付け、文字列化された名前を返します。名前が不正な場合、メソッドはエラーを返します。

戻り値 文字列化された名前を返します。

CosNaming::NamingContextExt::to_URL()

概要 URL と文字列化された名前を組み合わせ、URL 文字列を返します。

構文

```
CosNaming::NamingContextExt::to_URL()
URLString to_URL(in Address addr, in StringName sn);
```

パラメータ

addr

URL。このパラメータが定義されていない場合、IIOP プロトコルでローカル・ホスト名が使用されます。

sn

URL と組み合わせる文字列化された名前。

例外

`InvalidAddress`

URL が無効であることを示します。

`InvalidName`

名前が無効であることを示します。長さ 0 の名前は無効です。

戻り値 URL と文字列化された名前を組み合わせた URL 文字列を返します。

BindingIterator オブジェクト

`BindingIterator` オブジェクトを使用すると、クライアント・アプリケーションは、`NamingContext` オブジェクトの `list` メソッドで返されたバインディングのアンバウンディッド・コレクションをたどることができます。`BindingIterator` オブジェクトを使用すると、クライアント・アプリケーションは、各呼び出しで取得したバインディング数を制御できます。`BindingIterator` オブジェクトのメソッドの呼び出しの間にネーミング・コンテキストが変更された場合、以降の `next_one()` メソッドまたは `next_n()` メソッドの呼び出しの動作は、インプリメンテーションによって異なります。

`BindingIterator` オブジェクトを作成しても、`destroy` メソッドを呼び出さない場合、クライアント・アプリケーションはリソース不足になる場合があります。CORBA ネーム・サービスでは、クライアント・アプリケーションに通知せずに、いつでもバインディング・イテレータを破棄できます。`BindingIterator` オブジェクトに対する呼び出しで `OBJECT_NOT_EXIST` 例外を

予期し、この例外を適切に処理するようにクライアント・アプリケーションを記述する必要があります。

リスト 2-3 は、BindingIterator オブジェクトの OMG IDL を示しています。

リスト 2-3　BindingIterator オブジェクトの OMG IDL

```
module CosNaming {
       interface BindingIterator {
              boolean next_one(out Binding b);
              boolean next_n(in unsigned long how_many,
                             out BindingList b);
              void destroy();
       };
}
```

CosNaming::BindingIterator::destroy()

概要 BindingIterator オブジェクトを破棄し、オブジェクトに関連付けられているメモリを解放します。このメソッドの呼び出しに失敗すると、メモリの使用量が増加します。

C++ マッピング

```
void destroy();
```

Java マッピング

```
void destroy();
```

パラメータ 特にありません。

例外 特にありません。

説明 destroy メソッドを呼び出した後に、クライアント・アプリケーションが BindingIterator オブジェクトに対するオペレーションを呼び出すと、OBJECT_NOT_EXIST 例外が発生します。

戻り値 特にありません。

CosNaming::BindingIterator::next_n()

概要 リストから要求したバインディングの番号を格納する BindingList データ構造体を返します。返されたバインディングの番号は、要求された数がリストを超えている場合は、それよりも小さくなります。

C++ マッピング

```
boolean next_n(in unsigned_long how_many, out BindingList bl);
```

Java マッピング

```
boolean next_n(int how_many, BindingListHolder bl);
```

パラメータ

how_many
返すバインディングの最大数。

bl
要求した数を超えないバインディングを格納する BindingList データ構造体。

例外

BAD_PARAM
how_many パラメータに 0 の値がある場合に発生します。

戻り値 リストの上限に達している場合に、CORBA::FALSE が返されます。それ以外の場合、CORBA::TRUE が返されます。

CosNaming::BindingIterator::next_one()

概要 リスト内の次の Binding オブジェクトを返します。

C++ マッピング

```
boolean next_one(out Binding b);
```

Java マッピング

```
boolean next_one(BindingHolder b);
```

パラメータ

b

リスト内の次の Binding オブジェクト。

例外 特にありません。

戻り値 リストの上限に達している場合に、CORBA::FALSE が返されます。それ以外の場合、CORBA::TRUE が返されます。

CORBA ネーム・サービスで発生する例外

ここでは、CORBA ネーム・サービスで発生する例外について説明します。

AlreadyBound

構文 `exception AlreadyBound{};`

パラメータ 特にありません。

説明 この例外は、指定した名前にオブジェクトがバインドされている場合に発生します。1 つのコンテキストで名前にバインドできるオブジェクトは 1 つだけです。

CannotProceed

構文　`exception CannotProceed{};`

パラメータ　`NamingContext` *`cxt`*
オペレーションが再試行可能なコンテキスト。

`Name` *`rest_of_name`*
機能していないほかの名前。

説明　この例外は、予期しない例外のためにメソッドが適切に処理できなくなったときに発生します。

InvalidAddress

構文　`exception InvalidAddress{};`

パラメータ　特にありません。

説明　この例外は、URL が無効の場合に発生します。

InvalidName

構文　`exception InvalidName{};`

パラメータ　特にありません。

説明　この例外は、`Name` が無効の場合に発生します。長さ 0 の名前は無効です。

NotEmpty

構文 `exception NotEmpty{};`

パラメータ 特にありません。

説明 この例外は、バインディングを含む `NamingContext` オブジェクトに対して `destroy()` メソッドが使用されたときに発生します。`NamingContext` オブジェクトは、破棄する前に空になっている必要があります。

NotFound

構文　`exception NotFound{NotFoundReason why; Name rest_of_name;};`

パラメータ　*why*

オペレーションが再試行可能なコンテキスト。

rest_of_name

機能していないほかの名前。

説明　この例外は、名前のコンポーネントがバインディングを識別しなかったか、バインディングのタイプが実行中のオペレーションに対して不正だった場合に発生します。*why* パラメータは、エラーの理由を示します。*rest_of_name* パラメータは、エラーの原因を示します。以下の原因が表示されます。

- missing_node － *rest_of_name* パラメータの最初の名前コンポーネントが、親コンテキスト内のその名前の下でバインドされていないバインドであることを示します。
- not_context － *rest_of_name* パラメータの最初の名前コンポーネントが、ncontext のタイプが要求されているときに nobject のタイプのバインディングであることを示します。
- not_object － *rest_of_name* パラメータの最初の名前コンポーネントが、nobject のタイプが要求されているときに ncontext のタイプのバインディングであることを示します。

3　BEA Tuxedo 名前空間の管理

ここでは、次の内容について説明します。

- CORBA ネーム・サービスのインストール
- CORBA ネーム・サービスのサーバ・プロセスの起動
- 名前空間の永続化
- 永続ストレージ・ファイルの圧縮
- 関連付けのない NamingContext オブジェクトの削除
- 名前空間のフェデレーション
- バインディング・イテレータの管理

CORBA ネーム・サービスのインストール

BEA Tuxedo をインストールするときに、CORBA ネーム・サービスをインストールします。BEA Tuxedo のインストールの詳細については、『BEA Tuxedo システムのインストール』を参照してください。

CORBA ネーム・サービスのサーバ・プロセスの起動

CORBA ネーム・サービスのサーバ・プロセスを起動するには、使用している BEA Tuxedo CORBA アプリケーションの UBBCONFIG ファイルでサーバ・プロセスを定義する必要があります。CORBA ネーム・サービスのサーバ・プロセスを起動するには、cns コマンドを使用します。UBBCONFIG ファイルの CLOPT パラメータの後に、cns コマンド行オプションを入力します。BEA Tuxedo ドメインごとに実行される CORBA ネーム・サービスのサーバ・プロセスは 1 つだけです。リスト 3-1 は、CORBA ネーム・サービスのサーバ・プロセスに対する UBBCONFIG エントリの例です。

リスト 3-1　CORBA ネーム・サービスの UBBCONFIG ファイル・エントリ

```
...
#
#BEA Tuxedo CORBA ネーム・サービスのサーバ・プロセス
#
        cns
                SRVGRP = SYS_GRP
                SRVID = 6
                RESTART = N
        CLOPT = "-A -- -f C:\cnsroot.dat -M 0"
```

idl コマンドとオプションの詳細については、「第 2 章 CORBA ネーム・サービス・リファレンス」を参照してください。コンフィギュレーション・ファイル作成の詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo アプリケーションの設定』を参照してください。

CORBA ネーム・サービスのサーバ・プロセスを起動すると、名前空間の内容を表示し、名前空間内のオブジェクトを管理するために、表 3-1 のコマンドを使用できます。コマンドとオプションの詳細については、「第 2 章 CORBA ネーム・サービス・リファレンス」を参照してください。

表 3-1 BEA Tuxedo 名前空間を管理するためのコマンド

コマンド	説明
cns	BEA Tuxedo 名前空間のサーバ・プロセスを起動します。
cnsbind	アプリケーション・オブジェクトおよびネーミング・コンテキストを BEA Tuxedo 名前空間にバインドします。
cnsls	BEA Tuxedo 名前空間の内容を表示します。
cnsunbind	BEA Tuxedo 名前空間のバインディングを解除します。

名前空間の永続化

CORBA ネーム・サービスは、名前空間内に情報のコピーを 2 つ保持します。1 つはメモリ内に保持されます。このコピーにアクセスすると、高速化および名前解決の最適化が可能になります。もう 1 つのコピーは、必要に応じて永続ストレージに保存されます。このコピーでは、名前空間の状態と構造の保存および復元が可能になります。

名前空間の永続化する主な目的は、名前空間の現在実行中のインスタンスによって保持されるメモリ内のネーミング情報を現在の状態にしておくことです。名前空間の永続コピーを保持することで、CORBA ネーム・サービスは、サーバ・プロセスが終了した場合に現在のネーミング情報を再作成できます。永続ストレージ・ファイルを読み取り、最後に記録されたネーミング情報を再作成するように、CORBA ネーム・サービスのサーバ・プロセスの新しいインスタンスをコンフィギュレーションできます。

名前空間の永続コピーを作成し、そのコピーをファイルに保存するには、CORBA ネーム・サービスのサーバ・プロセスの起動時に `cns` コマンドの `-p` オプションを指定します。CORBA ネーム・サービスは、指定した場所と名前で永続ストレージ・ファイルを作成します。

-p オプションで指定した永続ストレージ・ファイルが既に存在する場合は、ファイルがオープンされ、処理されます。永続ストレージ・ファイルのバックアップは、CORBA ネーム・サービスのサーバ・プロセスが開始される前に作成されます。永続ストレージ・ファイルのバックアップ・コピーの名前は、*filename*.BAK です。永続ストレージ・ファイルの名前を再利用する場合、既存の永続ストレージ・ファイルを削除または移動し、CORBA ネーム・サービスのサーバ・プロセスを再起動する必要があります。

永続ストレージ・ファイルが正常に作成された場合、ファイルのエントリが ULOG ファイルに書き込まれます。エントリは、ディレクトリの場所とファイル名、ファイルが新しく作成されたかどうか、およびファイル名の決定に使用するメカニズム (指定、環境、デフォルトなど) を示します。永続ストレージ・ファイルの作成時にエラーが発生すると、発生したエラーのタイプを示すエントリが ULOG ファイルに書き込まれます。

CORBA ネーム・サービスのサーバ・プロセスは、起動時に永続ストレージ・ファイルから名前空間の構造を再作成するので、起動時間は、永続ストレージ・ファイルのサイズに比例します。数百メガバイト単位のサイズの大きな永続ストレージ・ファイルの場合、CORBA ネーム・サービスのサーバ・プロセスが起動時に名前空間を再作成するのに数秒または数分かかる場合もあります。

永続ストレージ・ファイルの圧縮

永続ストレージ・ファイルは、名前空間のメモリ内コピーに影響するすべてのオペレーションに関する情報を格納します。永続ストレージ・ファイルは、現在の名前空間の構造と状態の再作成に必要とする以上の情報を格納することがあります。名前空間の構造が同じサイズのままでも、永続ストレージ・ファイルのサイズは非常に大きくなることがあります。

CORBA ネーム・サービスでは、永続ストレージ・ファイルを圧縮して不要な情報を削除できます。cns コマンドの -c オプションは、永続ストレージ・ファイルの圧縮を制御します。圧縮オプションは、現在の情報を処理して、新しく圧縮された永続ストレージ・ファイルを作成します。

CORBA ネーム・サービスのサーバ・プロセスが起動されたときに、圧縮オペレーションは以下を実行します。

1. メモリ内の名前空間の構造を処理します。
2. 既存の永続ストレージ・ファイルを上書きします。

3. アンバインド、再バインド、または破棄オペレーションによって名前空間から削除されたすべてのバインドおよび再バインド・エントリを削除します。
4. バインドされていないバインディングをすべて削除します。バインドされていないバインディングは、バインディングが関連付けられているオブジェクトが名前空間から削除されても名前空間内に残っているバインディングです。バインドされていないバインディングは、ネーミング・コンテキストが親コンテキストからアンバウンドされた状態で `CosNaming::NamingContext::destroy()` メソッドを実行したときに発生します。

`-c` オプションは、`cns` コマンドの `-p` オプションが指定されている場合にのみ使用できます。`cns` コマンドの `-c` オプションの詳細については、「第 2 章 CORBA ネーム・サービス・リファレンス」を参照してください。

関連付けのない NamingContext オブジェクトの削除

関連付けのないコンテキストは、ほかのどのオブジェクトにもバインドされていないコンテキストです。このコンテキストは、一度もバインドされていない可能性もありますし、コンテキストにバインドされた後で明示的に、または再バインドによって破棄された可能性もあります。CORBA ネーム・サービスでは、関連付けのない `NamingContext` オブジェクトは、以下のいずれかの方法で作成されます。

- `CosNaming::NamingContext::new_context` メソッドを使用して、新しい `NamingContext` オブジェクトを作成する方法。ただし、新しい `NamingContext` オブジェクトを名前空間にはバインドしません。
- `CosNaming::NamingContext::rebind()` メソッドまたは `CosNaming::NamingContext::unbind()` メソッドを使用して、最新の親 `NamingContext` オブジェクトから `NamingContext` オブジェクトをアンバインドする方法。ただし、`NamingContext` オブジェクトは破棄しません。

NamingContext オブジェクトに連合されているクライアント・アプリケーションとほかの名前空間は、NamingContext オブジェクトへのオブジェクト・リファレンスが保持されている限り、関連付けのない NamingContext オブジェクトのオペレーションを実行できます。

名前空間の現在のインプリメンテーションは、関連付けのない NamingContext オブジェクトを、特殊な LostandFoundContext オブジェクトの形式で保持します。

cns コマンドの -d オプションを使用して、関連付けのない NamingContext オブジェクトを名前空間から削除します。-d オプションは、関連付けがないと認識されたすべての NamingContext オブジェクトをアンバインドして破棄します。

-d オプションは、cns コマンドの -p オプションが指定されている場合にのみ使用できます。cns コマンドの -d オプションの詳細については、「第 2 章 CORBA ネーム・サービス・リファレンス」を参照してください。

名前空間のフェデレーション

CORBA ネーム・サービスは、論理名前空間の要素が別の複数のリモート・マシンに存在することを意味する連合型名前空間の概念をサポートします。連合型名前空間では、ある名前空間で保持されている NamingContext オブジェクトへのオブジェクト・リファレンスを使用して、別の名前空間に NamingContext オブジェクトをバインドできます。CORBA ネーム・サービスは、サード・パーティのネーム・サービスに加えて、ほかのマシンで動作している CORBA ネーム・サービスのインプリメンテーションとのフェデレーション (連合) をサポートします。CORBA ネーム・サービスがサード・パーティのネーム・サービスと連合するには、サード・パーティのネーム・サービスは、Object Management Group (OMG) のインターオペラブル・ネーム・サービス (INS) 仕様で指定した命名形式をサポートしている必要があります。

以下のトピックでは、サード・パーティのネーム・サービスとのフェデレーション加えて、インバウンド・フェデレーションとアウトバウンド・フェデレーションをサポートする方法について説明します。

インバウンド・フェデレーション

インバウンド・フェデレーションは、ローカルの BEA Tuxedo 名前空間に存在する NamingContext オブジェクトをリモート・ネーム・サービスにバインドする機能です。名前空間が連合されると、リモート・ネーム・サービスは、BEA Tuxedo 名前空間内の NamingContext オブジェクトのオペレーションを実行できます。サード・パーティのネーム・サービスとのインバウンド・フェデレーションには、インターネット ORB 間プロトコル (IIOP) を使用します。したがって、非公式の IIOP 接続をサポートするように、CORBA ネーム・サービスの IIOP リスナ / ハンドラをコンフィギュレーションする必要があります。

IIOP リスナ / ハンドラで非公式の接続を有効化するには、ISL コマンドの -C パラメータを使用します。-C パラメータは、非公式の接続が確立されたときに IIOP リスナ / ハンドラがどのように動作するかを決定します。インバウンド・フェデレーションを使用するには、-C パラメータに対して warn または none 値を指定します。warn 値を指定すると、IIOP リスナ / ハンドラは、非公式の接続が検出されたときにメッセージをユーザ・ログ例外に記録します。例外は発生しません。none 値を指定すると、IIOP リスナ / ハンドラは非公式の接続を無視します。ISL コマンドの詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』を参照してください。

リスト 3-2 は、インバウンド・フェデレーションをサポートする IIOP リスナ / ハンドラに対する UBBCONFIG エントリの例です。

リスト 3-2 インバウンド・フェデレーションをサポートする IIOP リスナ / ハンドラに対する UBBCONFIG ファイルのエントリ

```
#
# インバウンド・フェデレーションをサポートする
# IIOP リスナ / ハンドラを開始するためのエントリ

ISL
                SRVGRP = SYS_GRP
                SRVID  = 5
                MIN    = 1
                MAX    = 1
                CLOPT  = "-A -- -n //blotto:2470 -C none"
```

アウトバウンド・フェデレーション

アウトバウンド・フェデレーションは、リモート・ネーム・サービスに存在する NamingContext オブジェクトを CORBA ネーム・サービスの名前空間にバインドする機能です。アウトバウンド・フェデレーションを使用すると、CORBA ネーム・サービスを使用し、この NamingContext オブジェクトに対してオペレーションを実行できます。サード・パーティのネーム・サービスとのアウトバウンド・フェデレーションには、IIOP を使用します。したがって、アウトバウンド・フェデレーションをサポートするように、CORBA ネーム・サービスの IIOP リスナ / ハンドラをコンフィギュレーションする必要があります。

IIOP リスナ / ハンドラでアウトバウンド・フェデレーションを有効化するには、ISL コマンドの -o パラメータを使用します。-o パラメータを使用すると、IIOP ハンドラに接続されていないサーバ・アプリケーション内のオブジェクトに対するアウトバウンド IIOP 呼び出しが IIOP リスナで可能になります。ISL コマンドの詳細については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo コマンド・リファレンス』を参照してください。

リスト 3-3 は、アウトバウンド・フェデレーションをサポートする IIOP リスナ / ハンドラに対する UBBCONFIG エントリの例です。

リスト 3-3 アウトバウンド・フェデレーションをサポートする IIOP リスナ / ハンドラに対する UBBCONFIG ファイルのエントリ

```
#
# アウトバウンド・フェデレーションをサポートする
# IIOP リスナ / ハンドラのエントリ
#

ISL
                SRVGRP = SYS_GRP
                SRVID  = 5
                MIN    = 1
                MAX    = 1
                CLOPT  = "-A -- -n //blotto:2470 -O"
```

複数の BEA Tuxedo ドメイン間のフェデレーション

異なる BEA Tuxedo ドメインで実行されている複数の CORBA ネーム・サービスのサーバ・プロセス間のフェデレーションでは、ドメイン間接続を可能にするためにドメイン・ゲートウェイを使用する必要があります。ドメイン・ゲートウェイのコンフィギュレーションの詳細については、管理トピックの『BEA Tuxedo システム入門』の「マルチ・ドメイン・コンフィギュレーション」を参照してください。

バインディング・イテレータの管理

OMG INS 仕様では、未処理のバインディング・イテレータのコレクションが使用可能です。バインディング・イテレータではクライアント・アプリケーションによる明示的な破棄が必要なので、バインディング・イテレータを正しく削除できない場合があります。

未使用のバインディング・イテレータの数が増えたために CORBA ネーム・サービスがリソース不足になる可能性をできる限り抑えるには、`cns` コマンドの `-M` オプションを使用して、ネーム・サービスのバインディング・イテレータの最大数を設定します。この最大数に達すると、新しいバインディング・イテレータへの要求によって、未使用のバインディング・イテレータを破棄できるようになります。CORBA ネーム・サービスは、使用履歴の最も古いアルゴリズムを基に、削除するバインディング・イテレータを選択します。

CORBA ネーム・サービスのサーバ・プロセスが `-M` オプションで開始された場合、CORBA ネーム・サービスは、BEA Tuxedo CORBA アプリケーションで使用中のバインディング・イテレータを破棄することがあるので、すべての BEA Tuxedo アプリケーションは、バインディング・イテレータに対する呼び出しに際して `CORBA::OBJECT_NOT_EXIST` システム例外を処理できるようにする必要があります。

セキュリティで保護された BEA Tuxedo アプリケーションでの CORBA ネーム・サービスの使用

セキュリティで保護された BEA Tuxedo CORBA アプリケーションで cnsls、cnsbind、および cnsunbind コマンドを使用する場合、BEA Tuxedo ドメインの PrincipalAuthenticator オブジェクトを取得し、適切なセキュリティ情報でそのドメインにログオンする必要があります。

セキュリティで保護されたログオンを有効にするには、TOBJ_SYSAUTH または TOBJ_APPAUTH のセキュリティ・レベルで BEA Tuxedo ドメインをコンフィギュレーションする必要があります。cnsls、cnsbind、および cnsunbind コマンドのユーザ名は cnsutils です。cnsutils という名前のユーザを作成するには、tpusradd コマンドを使用する必要があります。BEA Tuxedo ドメインに対して指定したセキュリティ・レベルに従って、UBBCONFIG ファイルの USER_AUTH および APP_PW 環境変数でユーザ・パスワードやドメイン・パスワードを定義できます。これらの環境変数が設定されてなく、BEA Tuxedo ドメインに TOBJ_SYSAUTH または TOBJ_APPAUTH セキュリティ・レベルがある場合、cnsls、cnsbind、および cnsunbind コマンドでは、パスワードを入力するように求められます。

セキュリティ・レベルの詳細、および BEA Tuxedo セキュリティ環境でのユーザの定義については、BEA Tuxedo オンライン・マニュアルの『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』を参照してください。

4 CORBA ネーム・サービスを使用するアプリケーションの開発

ここでは、次の内容について説明します。

開発手順

表 4-1 は、CORBA ネーム・サービスを使用する BEA Tuxedo CORBA アプリケーションの開発プロセスの概略です。

表 4-1 開発プロセス

手順	説明
1	CosNaming インターフェイスの OMG IDL を取得します。
2	CosNaming インターフェイスの宣言およびプロトタイプをインクルードします。
3	BEA Tuxedo 名前空間に接続します。
4	BEA Tuxedo 名前空間にオブジェクトをバインドします。
5	名前を使用して BEA Tuxedo 名前空間でオブジェクトをロケートします。

このトピックの手順を実行する前に、CORBA ネーム・サービスのサーバ・プロセスを開始する必要があります。詳細については、第 3 章の 2 ページ「CORBA ネーム・サービスのサーバ・プロセスの起動」を参照してください。

このトピックの開発手順が完了したら、`buildobjclient` コマンドと `buildobjserver` コマンドを使用して、CORBA ネーム・サービスを使用するサーバ・アプリケーションとクライアント・アプリケーションをコンパイルします。`buildobjclient` コマンドと `buildobjserver` コマンドの詳細については、『BEA Tuxedo コマンド・リファレンス』を参照してください。

ステップ 1: CosNaming インターフェイスの OMG IDL を取得する

BEA Tuxedo CORBA アプリケーションは、CosNaming.idl で定義したインターフェイスを使用して CORBA ネーム・サービスにアクセスします。この Object Management Group (OMG) インターフェイス定義言語 (IDL) ファイルは、CORBA ネーム・サービスで使用するインターフェイス、COSnaming データ構造体、例外を定義します。CosNaming.idl ファイルは、以下のディレクトリの場所にあります。

Windows

drive:\%TUXDIR%\include\CosNaming.idl

UNIX

/*usr*/*local*/$TUXDIR/include/CosNaming.idl

リスト 4-1 は、CosNaming.idl の OMG IDL を示しています。CORBA C++ および Java アプリケーションの両方で同じ OMG IDL ファイルが使用されます。

リスト 4-1 CosNaming.idl

```
#ifndef _COSNAMING_IDL_
#define _COSNAMING_IDL_

module CosNaming {

#pragma prefix "omg.org/CosNaming"
       typedef string Istring;

       struct NameComponent {
              Istring id;
              Istring kind;
       };

       typedef sequence<NameComponent> Name;

       enum BindingType { nobject, ncontext };

       struct Binding {
              Name        binding_name;
              BindingType binding_type;
       };
```

```
typedef sequence <Binding> BindingList;

interface BindingIterator;

interface NamingContext {
       enum NotFoundReason { missing_node,
                             not_context,
                             not_object };

       exception NotFound {
              NotFoundReason  why;
              Name            rest_of_name;
       };

       exception CannotProceed {
              NamingContext   cxt;
              Name            rest_of_name;
       };

       exception InvalidName{};

       exception AlreadyBound {};

       exception NotEmpty{};

       void   bind(in Name n, in Object obj)
              raises(NotFound,
                     CannotProceed,
                     InvalidName,
                     AlreadyBound);

       void   rebind(in Name n, in Object obj)
              raises(NotFound,
                     CannotProceed,
                     InvalidName);

       void   bind_context(in Name n, in NamingContext nc)
              raises(NotFound,
                     CannotProceed,
                     InvalidName,
                     AlreadyBound);

       void   rebind_context(in Name n, in NamingContext nc)
              raises(NotFound,
                     CannotProceed,
                     InvalidName);

       Object resolve (in Name n)
              raises(NotFound,
                     CannotProceed,
                     InvalidName);

       void   unbind(in Name n)
              raises(NotFound,
                     CannotProceed,
                     InvalidName);
```

```
                NamingContext   new_context();
                NamingContext   bind_new_context(in Name n)
                        raises(NotFound,
                                AlreadyBound,
                                CannotProceed,
                                InvalidName);

                void    destroy() raises(NotEmpty);
                void    list(in unsigned long      how_many,
                                out BindingList     bl,
                                out BindingIterator bi);

        };

        interface BindingIterator {
                boolean next_one(out Binding b);
                boolean next_n(in unsigned long how_many,
                                out BindingList bl);
                void    destroy();
        };

        interface NamingContextExt:NamingContext {
                typedef string StringName;
                typedef string Address;
                typedef string URLString;

        StringName  to_string(in Name n) raises(InvalidName);
        Name        to_name(in StringName sn)
                                raises(InvalidName);
        exception InvalidAddress {};

        URLString   to_url(in Address addr, in StringName sn)
                                raises(InvalidAddress, InvalidName);

        Object      resolve_str(in StringName n)
                                raises(NotFound,
                                        CannotProceed,
                                        InvalidName,
                                        AlreadyBound
                                );
                };
};

#pragma ID CosNaming "IDL:omg.org/CosNaming:1.0"
#endif // _COSNAMING_IDL_
```

ステップ 2: CosNaming インターフェイスの宣言およびプロトタイプをインクルードする

CosNaming インターフェイスの宣言とプロトタイプは、CORBA ネーム・サービスのソフトウェア・キットの一部として提供されます。

- CORBA C++ クライアント・アプリケーションでは、次の文を BEA Tuxedo CORBA クライアント・アプリケーションに追加して、ネーミング・インターフェイスの宣言とプロトタイプをインクルードします。

  ```
  #include "CosNaming_c.h"
  ```

  BEA Tuxedo CORBA C++ クライアント・アプリケーションのインクルード・ファイルは、`$TUXDIR/include` ディレクトリ (UNIX システム) または `%TUXDIR%\include` ディレクトリ (Windows システム) にあります。

- CORBA Java クライアント・アプリケーションでは、次の文を BEA Tuxedo CORBA クライアント・アプリケーションに追加して、ネーミング・インターフェイスの宣言とプロトタイプをインクルードします。

  ```
  import org.omg.CosNaming.*;
  ```

  CORBA ネーム・サービスのインターフェイスは、`org.omg.CosNaming` パッケージにあります。

  BEA Tuxedo CORBA Java クライアント・アプリケーションの Java パッケージは、`$TUXDIR/udataobj/java/jdk/m3envobj.jar` ファイル (UNIX システム) および `%TUXDIR%\udataobj\java\jdk\m3envobj.jar` ファイル (Windows システム) にあります。

- サード・パーティのオブジェクト・リクエスト・ブローカ (ORB) を使用する場合、コンパイルする前に、クライアント・ソース・スタブ・プログラムに CosNaming インターフェイスをインクルードまたはインポートします。

ステップ 3: BEA Tuxedo 名前空間に接続する

Bootstrap オブジェクトは、名前空間のルートに接続するために、NameService 環境オブジェクトをサポートしています。NameService 環境オブジェクトを使用する場合、オブジェクト・リクエスト・ブローカ (ORB) が名前空間のルートを特定します。次に、オブジェクト・リファレンスを CosNaming::NamingContext または CosNamingContextExt にナロー変換できます。オブジェクトを名前空間にバインドし、名前空間の名前を解決する前に、BEA Tuxedo 名前空間に接続する必要があります。

Bootstrap オブジェクトまたは CORBA インターオペラブル・ネーミング・サービス (INS) のブートストラップ処理メカニズムを使用すると、NameService 環境オブジェクトへの初期リファレンスを取得できます。BEA クライアント ORB を使用する場合は、BEA 社固有のメカニズムを使用します。別のベンダのクライアント ORB を使用する場合は、CORBA INS メカニズムを使用します。BEA Tuxedo ドメインのブートストラップ処理の詳細については、BEA Tuxedo オンライン・マニュアルの『CORBA プログラミング・リファレンス』の「第 4 章 CORBA ブートストラップ処理のプログラミング・リファレンス」を参照してください。

リスト 4-2 およびリスト 4-3 では、BEA Tuxedo 名前空間との通信を確立する C++ コードと Java コードを示します。

リスト 4-2 C++ による名前空間への接続例

```
...
Tobj_Bootstrap * bootstrap = new Tobj_Bootstrap (v_orb.in(), "");
CORBA::Object_var var_nameservice_oref=
       bootstrap.resolve_initial_references("NameService");
root = CosNaming::NamingContext::_narrow (obj);
...
```

リスト 4-3 Java による名前空間への接続例

```
...
Tobj_Bootstrap bootstrap = new Tobj_Bootstrap(orb, "");
```

```
org.omg.CORBA.Object NameServiceobj =
       gBootstrapObjRef.resolve_initial_references("NameService");
CosNaming.NamingCOntextExt ns_root =
       CosNaming.NamingContextExtHelper.narrow (ns_obj);
...
```

名前空間のルートの文字列化されたオブジェクト・リファレンスを使用して、BEA Tuxedo ドメインで名前空間に接続することもできます。文字列化されたオブジェクト・リファレンスを使用するには、CORBA ネーム・サービスのサーバ・プロセスを開始するときに -f コマンド行オプションを指定する必要があります。-f コマンド行オプションは、文字列化されたオブジェクト・リファレンスを CNS_ROOT_FILE 環境変数、または以下の場所のいずれかに書き込みます。

Windows

```
%APPDIR%\cnsroot.dat
```

UNIX

```
$APPDIR/cnsroot.dat
```

名前空間のルート用の文字列化されたオブジェクト・リファレンスは、CORBA ネーム・サービスのサーバ・プロセスが開始または終了しても変更されません。文字列化されたオブジェクト・リファレンスは、特定のサーバ・プロセスではなく、特定のホスト・マシンに関連付けられているためです。BEA Tuxedo 名前空間と通信するために取得されている文字列化されたオブジェクト・リファレンスを、ほかの BEA Tuxedo と通信するために使用することはできません。

リスト 4-4 およびリスト 4-5 では、文字列化されたオブジェクト・リファレンスを使用して BEA Tuxedo 名前空間との通信を確立する C++ コードと Java コードを示します。

リスト 4-4　文字列化されたオブジェクト・リファレンスを使用する C++ の例

```
...
Tobj_Bootstrap * bootstrap;
bootstrap = new Tobj_Bootstrap (v_orb.in(), "");
CORBA::Object_var obj = GetRefFromFile ("cnsroot.dat", v_orb);
root = CosNaming::NamingContext::_narrow (obj);
...
```

リスト 4-5 文字列化されたオブジェクト・リファレンスを使用する Java の例

```
...
Tobj_Bootstrap bootstrap = new Tobj_Bootstrap(orb, "");
BufferedReader inFile =
       newBufferedReader(new FileReader ("cnsroot.dat"));
String root_ior_string = inFile.readLine ();
org.omg.CORBA.Object ns_obj =
       orb.string_to_objecet (root_ior_string);
CosNaming.NamingContextExt ns_root =
       CosNaming.NamingContextExtHelper.narrow (ns_obj);
...
```

セキュリティとトランザクションも利用する BEA Tuxedo CORBA アプリケーションで、文字列化されたオブジェクト・リファレンスを使用する場合は、以下の制限事項に注意してください。

1. BEA Tuxedo CORBA アプリケーションは、文字列化されたオブジェクト・リファレンスを使用して BEA Tuxedo 名前空間に接続する前に、Bootstrap オブジェクトを作成して、IIOP リスナ / ハンドラに接続する必要があります。Bootstrap オブジェクトを最初に呼び出すことで、BEA Tuxedo アプリケーションは、IIOP リスナ / ハンドラへの正式な接続を確立します。

   BEA Tuxedo アプリケーションが最初に Bootstrap オブジェクトを作成しなかった場合、名前空間から取り出したオブジェクトでトランザクションとセキュリティを使用することはできません。トランザクションとセキュリティは、正式な接続を使用する必要があります。

2. 複数の IIOP リスナ / ハンドラが UBBCONFIG ファイルで定義されている場合、BEA Tuxedo CORBA アプリケーションは、UBBCONFIG ファイル内で TOBJADDR 環境変数が定義している最初の IIOP リスナ / ハンドラを使用する必要があります。

   CORBA ネーム・サービスは、IIOP リスナ / ハンドラのデフォルトのホストとポートを使用して、名前空間のルートに対して文字列化されたオブジェクト・リファレンスを作成します。UBBCONFIG ファイルで定義されている最初の IIOP リスナ / ハンドラが、デフォルト IIOP リスナ / ハンドラと見なされます。デフォルト IIOP リスナ / ハンドラを使用すると、CORBA ネーム・サービスによって取得されたすべてのオブジェクト・リファレンスが正式な接続になります。トランザクションとセキュリティは、正式な接続を使用する必要があります。

ステップ 4: BEA Tuxedo 名前空間にオブジェクトをバインドする

BEA Tuxedo 名前空間にオブジェクトをバインドする方法には、以下の 2 種類があります。

- cnsbind コマンド
- CosNaming::NamingContext オブジェクトの bind() メソッド

cnsbind コマンドを使用すると、アプリケーション・オブジェクトまたはネーミング・コンテキスト・オブジェクトを BEA Tuxedo 名前空間にバインドできます。cnsbind コマンドを使用する前に、CORBA ネーム・サービスのサーバ・プロセスを開始する必要があります。cnsbind コマンドの詳細については、「第 2 章 CORBA ネーム・サービス・リファレンス」を参照してください。

リスト 4-6 とリスト 4-7 では、CosNaming::NamingContext オブジェクトの bind() メソッドの C++ および Java コード・インプリメンテーションを示します。このコード例では、Name の id フィールドと kind フィールドを表す 2 つのパラメータを受け付けます。この 2 つのパラメータは、SimpleFactory オブジェクトの Name を初期化し、SimpleFactory オブジェクトを名前空間にバインドします。

リスト 4-6　C++ による BEA Tuxedo 名前空間への名前のバインディング例

```
...
// 名前空間で SimpleFactory オブジェクトの識別に使用する
//Name の確立

CosNaming::Name_var        factory_name = new CosNaming::Name(1);
       factory_name->length(1);
       factory_name[(CORBA::ULong) 0].id =
                                   (const char * "simple_factory";
       factory_name[(CORBA::ULong) 0].kind =
                                   (const char *) "";
//SimpleFactory のオブジェクト・リファレンスを作成

s_v_factory_refer = TP::create_object_reference(
                                   _tc_SimpleFactory->id(),
                                   "simple_factory",
                                   CORBA::NVList::_nil()
```

```
);
//NameService オブジェクト・リファレンスを取得。リスト 4-2 を参照
//SimpleFactory のオブジェクト・リファレンスを名前空間に配置

root->bind(factory_name, s_v_fact_ref);
...
```

リスト 4-7 Java による BEA Tuxedo 名前空間への名前のバインディング例

```
...
//SimpleFactory のオブジェクト・リファレンスを作成
org.omg.CORBA.object fact_ref =
       TP.create_object_reference(
              SimpleFactoryHelper.id()
              "simple_factory",
              null
       );

...
//NameService オブジェクト・リファレンスを取得。リスト 4-3 を参照
//SimpleFactory のオブジェクト・リファレンスを名前空間に配置
       CosNaming.NameComponent[] factName =
              ns_root.to_name("simple_factory");
       ns_root.bind(factName, fact_ref);

...
```

ステップ 5: 名前を使用して BEA Tuxedo 名前空間でオブジェクトをロケートする

`CosNaming::NamingContext` オブジェクトの `resolve()` メソッドを使用して、BEA Tuxedo ドメインの名前空間でオブジェクトをロケートします。リスト 4-8 とリスト 4-9 では、`Name` の `id` フィールドと `kind` フィールドを表す 2 つのパラメータを受け付ける C++ および Java コードを示します。コード例では、ネーミング・コンテキストにバインドし、名前を解決し、指定したオブジェクトのオブジェクト・リファレンスを取得します。

リスト 4-8 C++ による BEA Tuxedo 名前空間での名前のロケート例

```
...
// 名前空間で SimpleFactory オブジェクトの識別に使用する
//Name の確立
CosNaming::Name_var        factory_name = new CosNaming::Name(1);
       factory_name->length(1);
       factory_name[(CORBA::ULong) 0].id =
                                   (const char * "simple_factory";
       factory_name[(CORBA::ULong) 0].kind =
                                   (const char *) "";

//SimpleFactory オブジェクトを名前空間でロケート
CORBA::Object_var v_simple_factory_oref =
             root->resolve( *factory_name);
SimpleFactory_var v_simple_factory_ref =
             SimpleFactory::_narrow(v_simple_factory_oref.in());

//BEA Tuxedo CORBA ネーム・サービスから取得したリファレンスを使用して
//Simple オブジェクトを検索
Simple_var v_simple = v_simple_factory_ref->find_simple();
...
```

リスト 4-9 Java による BEA Tuxedo 名前空間での名前のロケート例

```
...
// 文字列名を基に名前空間で SimpleFactory オブジェクトを検索
org.omg.CORBA.Object simple_fact_oref =
       ns_root.resolve_str("simple_factory");
SimpleFactory simple_factory_ref =
       SimpleFactoryHelper.narrow(simple_fact_oref);

//Simple オブジェクトを検索
Simple simple = simple_factory_ref.find_simple();
...
```

5 CORBA Name Service サンプル・アプリケーションの使用

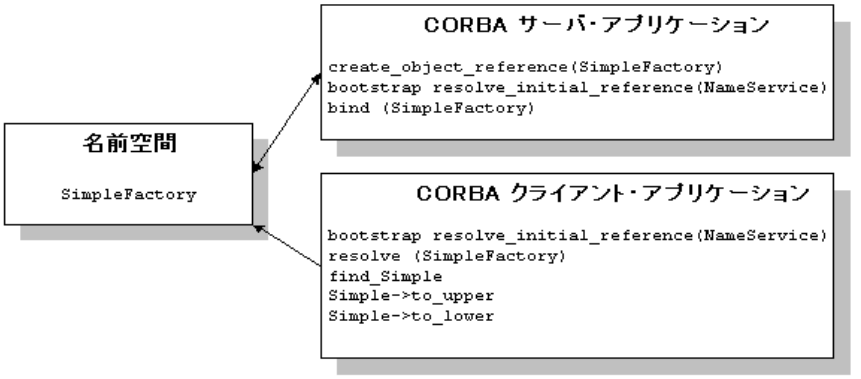
ここでは、次の内容について説明します。

- Name Service サンプル・アプリケーションのしくみ
- Name Service サンプル・アプリケーションのビルドと実行

Name Service サンプル・アプリケーションのしくみ

CORBA Name Service サンプル・アプリケーションは、Simpapp サンプル・アプリケーションを変更したものです。このサンプル・アプリケーションでは、CORBA C++ のクライアントとサーバ、および CORBA Java のクライアントを提供します。Name Service サンプル・アプリケーションは、名前空間を使用して `SimpleFactory` オブジェクトを格納します。サーバ・アプリケーションは、`SimpleFactory` オブジェクトを作成し、それを名前空間にバインドします。クライアント・アプリケーションは、名前空間に接続し、`SimpleFactory` オブジェクトの名前を解決して、`SimpleFactory` のメソッドを呼び出します。図 5-1 は、Name Service サンプル・アプリケーションのしくみを示しています。

図 5-1 Name Service サンプル・アプリケーション



Name Service サンプル・アプリケーションは、表 5-1 のリストにある CORBA インターフェイスをインプリメントします。

表 5-1 Name Service サンプル・アプリケーションの CORBA インターフェイス

インターフェイス	説明	オペレーション
SimpleFactory	Simple オブジェクトのオブジェクト・リファレンスを作成します。	find_simple()
Simple	文字列の大文字と小文字を変換します。	to_upper() to_lower()

リスト 5-1 は、Name Service サンプル・アプリケーションの CORBA インターフェイスを定義する simple.idl ファイルを示しています。

リスト 5-1 Name Service サンプル・アプリケーションの OMG IDL コード

```
#pragma prefix "beasys.com"

interface Simple
{
     // 文字列を小文字に変換 （新しい文字列を返す）
     string to_lower(in     string val);

     // 文字列を大文字に変換 （置換）
     void to_upper(inout string val);
};

interface SimpleFactory
{
     Simple find_simple();
};
```

Name Service サンプル・アプリケーションのビルドと実行

Name Service サンプル・アプリケーションをビルドおよび実行するには、次の手順に従います。

1. Name Service サンプル・アプリケーションのファイルを作業ディレクトリにコピーします。
2. 作業ディレクトリのファイルに対する権限を変更します。
3. 環境変数で定義した場所を確認します。
4. `runme` コマンドを実行します。

ステップ 1: Name Service サンプル・アプリケーションのファイルを作業ディレクトリにコピーする

Name Service サンプル・アプリケーションのファイルを、ローカル・マシンの作業ディレクトリにコピーします。作業ディレクトリでサンプル・アプリケーションを実行すると、サンプル実行時に作成されたファイルを識別できるようになります。以下のセクションでは、Name Service サンプル・アプリケーションのディレクトリの場所とソース・ファイルの詳細を説明します。

CORBA C++ クライアントとサーバ、および CORBA Java クライアント・バージョンの Name Service サンプル・アプリケーション

Name Service サンプル・アプリケーションのファイルは、次のディレクトリにあります。

Windows

drive:*tuxdir*\samples\corba\cnssimpapp

UNIX

/usr/local/tuxdir/samples/corba/cnssimpapp

表 5-2 にリストされているファイルを使用して、Name Service サンプル・アプリケーションをビルドおよび実行します。

表 5-2 Name Service サンプル・アプリケーションに含まれるファイル

ファイル	説明
simple.idl	Simple インターフェイスと SimpleFactory インターフェイスを宣言する OMG IDL コード。
simples.cpp	Name Service サンプル・アプリケーションの CORBA サーバ・アプリケーションの C++ ソース・コード。

表 5-2 Name Service サンプル・アプリケーションに含まれるファイル (続き)

ファイル	説明
simplec.cpp	Name Service サンプル・アプリケーションの CORBA クライアント・アプリケーションの C++ ソース・コード。
simple_i.cpp	Simple メソッドと SimpleFactory メソッドをインプリメントする C++ ソース・コード。
simple_i.h	Simple メソッドと SimpleFactory メソッドのインプリメンテーションを定義する C++ ヘッダ・ファイル。
SimpleClient.java	Name Service サンプル・アプリケーションの CORBA クライアント・アプリケーションの Java ソース・コード。
Readme.txt	C++ クライアントとサーバ、および Java クライアントの Name Service サンプル・アプリケーションのビルドと実行に関する情報を提供します。
runme.cmd	Name Service サンプル・アプリケーションをビルドおよび実行する Windows コマンド。
runme.ksh	Name Service サンプル・アプリケーションをビルドおよび実行する UNIX Korn シェル・スクリプト。
makefile.mk	UNIX オペレーティング・システムの Name Service サンプル・アプリケーションの makefile。このファイルは、Name Service サンプル・アプリケーションを手動でビルドするのに使用します。詳細については、Readme.txt を参照してください。実行可能な UNIX make コマンドの場所は、PATH 環境変数で定義する必要があります。

表 5-2 Name Service サンプル・アプリケーションに含まれるファイル (続き)

ファイル	説明
makefile.nt	Windows オペレーティング・システムの Name Service サンプル・アプリケーションの makefile。この makefile は、Visual C++ の nmake コマンドで直接使用できます。このファイルは、Name Service サンプル・アプリケーションを手動でビルドするのに使用します。詳細については、Readme.txt を参照してください。実行可能な Windows nmake コマンドの場所は、PATH 環境変数で定義する必要があります。

ステップ 2: Name Service サンプル・アプリケーションのファイルに対する保護属性を変更する

サンプル・アプリケーションのファイルは、読み取り専用のパーミッション・レベルでインストールされます。Name Service サンプル・アプリケーションのファイルを編集またはビルドする前に、作業ディレクトリにコピーするファイルの保護属性を次のように変更する必要があります。

Windows

```
prompt> attrib -r drive:\workdirectory\*.*
```

UNIX

```
1. prompt> /bin/ksh
2. ksh prompt> chmod u+w /workdirectory/*.*
```

また、UNIX プラットフォームでは、ファイルに実行権限を与えるために runme.ksh のパーミッションを次のように変更する必要もあります。

```
ksh prompt> chmod +x runme.ksh
```

ステップ 3: 環境変数の設定を確認する

Name Service サンプル・アプリケーションを実行する前に、一部の環境変数が正しい場所で定義されていることを確認する必要があります。ほとんどの場合、環境変数はインストール手順の一部として設定されます。一部の環境変数は、runme コマンドの実行時に設定されます。環境変数をチェックして、正しい情報を反映していることを確認する必要があります。

表 5-3 では、Name Service サンプル・アプリケーションの実行に必要な環境変数を示します。

表 5-3 Name Service サンプル・アプリケーションで必須の環境変数

環境変数	説明
APPDIR	runme コマンドを実行すると、この環境変数がカレント・ディレクトリの絶対パス名に設定されます。サンプル・アプリケーション・ファイルのコピー先ディレクトリから runme コマンドを実行します。次に例を示します。 **Windows** APPDIR=C:\workdirectory\cnssimpapp **UNIX** APPDIR=/usr/workdirectory/cnssimpapp
JAVA_HOME	Java 2 Software Development Kit (SDK) をインストールしたディレクトリ・パス。次に例を示します。 **Windows** JAVA_HOME=C:\JDK1.3 **UNIX** JAVA_HOME=/usr/local/JDK1.3 **注記** Name Service サンプル・アプリケーションで Java クライアントを実行する場合に、この環境変数を定義します。この環境変数が設定されていない場合、runme コマンドは、Java クライアント・アプリケーションを実行しません。 **注記** Java クライアント・アプリケーションでは、wstring データ型および wchar データ型をサポートするために Java 2 Software Development Kit (SDK) バージョン 1.3 が必要です。

表 5-3 Name Service サンプル・アプリケーションで必須の環境変数（続き）

環境変数	説明
RESULTSDIR	runme コマンドを実行すると、この環境変数は、APPDIR 環境変数で定義された場所の下位ディレクトリ results に設定されます。 **Windows** RESULTSDIR=%APPDIR%\results **UNIX** RESULTSDIR=$APPDIR\results
TUXCONFIG	runme コマンドを実行すると、この環境変数は、コンフィギュレーション・ファイルのディレクトリ・パスとファイル名に設定されます。 **Windows** TUXCONFIG=%RESULTSDIR%\tuxconfig **UNIX** TUXCONFIG=$RESULTSDIR/tuxconfig

インストール時に定義された環境変数の情報が正しいかどうかを確認するには、以下の手順を実行します。

Windows

1. [スタート] メニューの [設定] を選択します。
2. [設定] メニューの [コントロール パネル] を選択します。

   [コントロール パネル] が表示されます。
3. [システム] アイコンをクリックします。

   [システムのプロパティ] ウィンドウが表示されます。
4. [環境] タブをクリックします。

   [環境] ページが表示されます。
5. 環境変数の設定をチェックします。

UNIX

```
ksh prompt> printenv TUXDIR
ksh prompt> printenv JAVA_HOME
```

設定を変更するには、以下の手順を実行します。

Windows

1. [システムのプロパティ] ウィンドウの [環境] ページで、変更する環境変数をクリックするか、[変数] フィールドに環境変数の名前を入力します。
2. [値] フィールドに、環境変数の正しい情報を入力します。
3. [OK] をクリックして変更内容を保存します。

UNIX

```
ksh prompt> export TUXDIR=directorypath
ksh prompt> export JAVA_HOME=directorypath
```

ステップ 4: runme コマンドを実行する

runme コマンドは、以下の手順を最初から最後まで実行します。

1. システム環境変数の設定します。
2. UBBCONFIG ファイルをロードします。
3. クライアント・アプリケーションのコードをコンパイルします。
4. サーバ・アプリケーションのコードをコンパイルします。
5. tmboot コマンドを使用してサーバ・アプリケーションを起動します。
6. クライアント・アプリケーションを起動します。
7. tmshutdown コマンドを使用してサーバ・アプリケーションを終了します。

注記 Name Service サンプル・アプリケーションを手動で実行することもできます。Name Service サンプル・アプリケーションを手動で実行する手順については、Readme.txt ファイルで説明しています。

Name Service サンプル・アプリケーションをビルドおよび実行するには、次のように runme コマンドを入力します。

Windows

```
prompt> cd workdirectory

prompt> runme
```

UNIX

```
ksh prompt> cd workdirectory

ksh prompt> ./runme.ksh
```

Name Service サンプル・アプリケーションが最初から最後まで正常に実行されると、次の一連のメッセージが出力されます。

```
Testing NameService simpapp
    cleaned up
    prepared
    built
    loaded ubb
    booted
    ran
    shutdown
    saved results
  PASSED
```

表 5-4 では、runme コマンドで作業ディレクトリ内に生成されたファイルを示しています。

表 5-4 runme コマンドで生成される C++ ファイル

ファイル	説明
simple_c.cpp	idl コマンドによって生成されます。このファイルは、SimpleFactory インターフェイスと Simple インターフェイスのクライアント・スタブを格納します。
simple_c.h	idl コマンドによって生成されます。このファイルは、SimpleFactory インターフェイスと Simple インターフェイスのクライアント定義を格納します。

表 5-4 runme コマンドで生成される C++ ファイル

ファイル	説明
simple_s.cpp	idl コマンドによって生成されます。このファイルは、SimpleFactory インターフェイスと Simple インターフェイスのサーバ・スケルトンを格納します。
simple_s.h	idl コマンドによって生成されます。このファイルは、SimpleFactory インターフェイスと Simple インターフェイスのサーバ定義を格納します。
.adm/.keybd	セキュリティ暗号鍵データベースを格納するファイルです。サブディレクトリは、runme コマンドの tmloadcf コマンドによって生成されます。
results	runme コマンドで作成されたディレクトリ。APPDIR 環境変数によって定義された場所の下位にあります。

表 5-5 では、runme コマンドで作業ディレクトリ内に生成された Java ファイルを示しています。

表 5-5 runme コマンドで生成される Java ファイル

ファイル	説明
SimpleClient.class	javac コマンドを実行することで製品として構築される Java クラス・ファイル。
SimpleClient.jar	javac コマンドを実行することで製品として構築される Java アーカイブ・ファイル。
SimpleFactory.java	SimpleFactory インターフェイス用に idltojava コマンドによって生成されます。SimpleFactory インターフェイスは、Java バージョンの OMG IDL インターフェイスを含みます。これは、org.omg.CORBA.Object の拡張です。

表 5-5 runme コマンドで生成される Java ファイル (続き)

ファイル	説明
SimpleFactoryHelper.java	SimpleFactory インターフェイス用に idltojava コマンドによって生成されます。このクラスは、補助機能、特に narrow メソッドを提供します。
SimpleFactoryHolder.java	SimpleFactory インターフェイス用に idltojava コマンドによって生成されます。このクラスは、SimpleFactory 型のパブリック・インスタンス・メンバを保持します。また、CORBA に含まれているものの、Java に正確にマップされない out 引数と inout 引数に対するオペレーションを提供します。
_SimpleFactoryStub.java	SimpleFactory インターフェイス用に idltojava コマンドによって生成されます。このクラスは、SimpleFactory.java インターフェイスをインプリメントするクライアント・スタブです。
_SimpleFactoryImplBase.java	SimpleFactory インターフェイス用に idltojava コマンドによって生成されます。この抽象クラスは、サーバ・スケルトンです。SimpleFactory.java インターフェイスをインプリメントします。ユーザ記述のサーバ・クラス SimpleFactoryImpl は、_SimpleFactoryImplBase の拡張です。
Simple.java	Simple インターフェイス用に idltojava コマンドによって生成されます。Simple インターフェイスは、Java バージョンの OMG IDL インターフェイスを含みます。これは、org.omg.CORBA.Object の拡張です。
SimpleHelper.java	Simple インターフェイス用に idltojava コマンドによって生成されます。このクラスは、補助機能、特に narrow メソッドを提供します。

表 5-5 runme コマンドで生成される Java ファイル (続き)

ファイル	説明
SimpleHolder.java	Simple インターフェイス用に idltojava コマンドによって生成されます。このクラスは、Simple 型のパブリック・インスタンス・メンバを保持します。また、CORBA に含まれているものの、Java に正確にマップされない out 引数と inout 引数に対するオペレーションを提供します。
_SimpleStub.java	Simple インターフェイス用に idltojava コマンドによって生成されます。このクラスは、Simple.java インターフェイスをインプリメントするクライアント・スタブです。
.adm/.keybd	セキュリティ暗号鍵データベースを格納するファイルです。サブディレクトリは、runme コマンドの tmloadcf コマンドによって生成されます。

表 5-6 では、runme コマンドで results ディレクトリ内に生成されるファイルを示しています。

表 5-6 runme コマンドで results ディレクトリ内に生成されるファイル

ファイル	説明
input	runme コマンドが Java クライアント・アプリケーションに提供する入力を格納します。
output	runme コマンドが Java クライアント・アプリケーションを実行するときに生成される出力を格納します。

表 5-6 runme コマンドで results ディレクトリ内に生成されるファイル (続き)

ファイル	説明
expected_output	runme コマンドが Java クライアント・アプリケーションを実行するときに予測される出力を格納します。テストに成功したか失敗したかを判別するために、output ファイルのデータは expected_output ファイルのデータと比較されます。
log	runme コマンドで生成される出力を格納します。runme コマンドが失敗した場合は、このファイルでエラーをチェックします。
setenv.cmd	Windows オペレーティング・システム・プラットフォームの Name Service サンプル・アプリケーションのビルドと実行に必要な環境変数を設定するためのコマンドを格納します。
setenv.ksh	UNIX オペレーティング・システム・プラットフォームの Name Service サンプル・アプリケーションのビルドと実行に必要な環境変数を設定するためのコマンドを格納します。
stderr	tmboot コマンドによって生成されるコマンドからの出力です。このコマンドは、runme コマンドによって実行されます。-noredirect JavaServer オプションが UBBCONFIG ファイルで指定されている場合、System.err.println メソッドは、ULOG ファイルではなく、stderr ファイルに出力を送信します。

表 5-6 runme コマンドで results ディレクトリ内に生成されるファイル (続き)

ファイル	説明
stdout	tmboot コマンドによって生成される出力です。このコマンドは、runme コマンドによって実行されます。-noredirect JavaServer オプションが UBBCONFIG ファイルで指定されている場合、System.out.println メソッドは、ULOG ファイルではなく、stdout ファイルに出力を送信します。
tmsysevt.dat	TMSYSEVT (システム・イベント・リポート) プロセスで使用するフィルタ規則および通知規則を格納します。このファイルは、runme コマンドの tmboot コマンドによって生成されます。
tuxconfig	バイナリ形式の UBBCONFIG ファイル。
ubb	Java Name Service サンプル・アプリケーション用の UBBCONFIG ファイル。
ULOG.*date*	tmboot コマンドによって生成されるメッセージを含んだログ・ファイル。

Name Service サンプル・アプリケーションの使用

Name Service サンプル・アプリケーションのサーバ・アプリケーションを次のように実行します。

Windows

```
prompt> tmboot
```

UNIX

```
ksh prompt> tmboot
```

Name Service サンプル・アプリケーションのクライアント・アプリケーションを次のように実行します。

Windows

```
prompt> java -classpath %CLIENTCLASSPATH%
-DTOBJADDR=%TOBJADDR% SimpleClient

String?
Hello World
HELLO WORLD
hello world
```

UNIX

```
ksh prompt> java -classpath $CLIENTCLASSPATH
/m3envobj.jar -DTOBJADDR=$TOBJADDR SimpleClient

String?
Hello World
HELLO WORLD
hello world
```

別のサンプル・アプリケーションを使用する前に、以下のコマンドを入力して Name Service サンプル・アプリケーションを終了し、不要なファイルを作業ディレクトリから削除します。

Windows

```
prompt> tmshutdown -y

prompt> nmake -f makefile.nt clean
```

UNIX

```
ksh prompt> tmshutdown -y

ksh prompt> make -f makefile.mk clean
```

索引

I

J

N

O

R

T

U

い

え

か

こ

さ

せ

そ

な

は

ふ

ま

も

れ