BEA Tuxedo®

サンプルを使用した BEA Tuxedo アプリケーションの開発方法

BEA Tuxedo 8.1
2003 年 1 月

Copyright

Restricted Rights Legend

Trademarks or Service Marks

目次

3. bankapp (複雑な C 言語アプリケーション) のチュートリアル

6. XMLSTOCKAPP のチュートリアル

このマニュアルについて

このマニュアルでは、BEA Tuxedo® ATMI アプリケーションの実行方法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「BEA Tuxedo アプリケーションの開発」では、型付きバッファとメッセージング・パラダイムを使用して ATMI のクライアント / サーバ・アプリケーションを開発する方法を説明します。
- 第 2 章「simpapp (簡単な C 言語アプリケーション) のチュートリアル」では、`simpapp` サンプル・アプリケーションの実行方法を説明します。
- 第 3 章「bankapp (複雑な C 言語アプリケーション) のチュートリアル」では、`bankapp` サンプル・アプリケーションの実行方法を説明します。
- 第 4 章「CSIMPAPP (簡単な COBOL アプリケーション) のチュートリアル」では、`csimpapp` サンプル・アプリケーションの実行方法を説明します。
- 第 5 章「STOCKAPP (複雑な COBOL アプリケーション) のチュートリアル」では、`stockapp` サンプル・アプリケーションの実行方法を説明します。

対象読者

このマニュアルは、主にアプリケーション開発者を対象にしています。ATMI のクライアント・アプリケーションとサーバ・アプリケーションの開発方法、および BEA Tuxedo ソフトウェアに付属のサンプル・アプリケーションを実行する方法について説明します。

e-docs Web サイト

BEA 製品のマニュアルは BEA 社の Web サイト上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、e-docs Web サイトの BEA Tuxedo マニュアル・ページから入手できます。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 Documents ページの [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 をお持ちではない場合は、Adobe Web サイト (http://www.adobe.co.jp/) から無償でダウンロードできます。

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。
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文字も示します。
	例 :
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

等幅太字	コード内の重要な単語を示します。 例 : `void **commit** ( )`
等幅イタリック体	コード内の変数を示します。 例 : `String *expr*`

規則	**項目**
大文字	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
{ }	構文の行で選択肢を示します。かっこは入力しません。
[ ]	構文の行で省略可能な項目を示します。かっこは入力しません。 例 : `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	構文の行で、相互に排他的な選択肢を分離します。記号は入力しません。
...	コマンド行で次のいずれかを意味します。 ■ コマンド行で同じ引数を繰り返し指定できること ■ 省略可能な引数が文で省略されていること ■ 追加のパラメータ、値、その他の情報を入力できること 省略符号は入力しません。 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。省略符号は入力しません。

章

1 BEA Tuxedo 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。

- BEA Tuxedo アプリケーションを開発する前に
- BEA Tuxedo ATMI クライアントの作成
- BEA Tuxedo ATMI サーバの作成
- アプリケーションの型付きバッファ
- アプリケーションでの BEA Tuxedo メッセージング・パラダイム

BEA Tuxedo アプリケーションを開発する前に

BEA Tuxedo のアプリケーション・トランザクション・モニタ・インターフェイス (ATMI) を開発する場合、まず設計に関連する各種の概念や利用できるツールを確認しておきます。たとえば、クライアント (外部からの入力データが収集されて処理される方法)、およびサーバ (入力データを処理するビジネス・ロジックが定義されたプログラム) について理解しておきます。また、型付きバッファ (クライアント・プログラムがメモリ領域を割り当て、別のプログラムにデータを送る方法) についても理解しておきます。

BEA Tuxedo のメッセージング・パラダイムも重要な概念です。ATMI クライアント・プログラムは、ATMI ライブラリを呼び出すことによって、BEA Tuxedo システムにアクセスします。ATMI ライブラリのほとんどの呼び出しでは、要求 / 応答型や会話型などの通信方法がサポートされています。これらの通信は、BEA Tuxedo アプリケーションの基本部分です。

アプリケーション・キュー、イベント・ベースの通信、ATMI の使用方法の概念、および利用できるツールの詳細については、『BEA Tuxedo システム入門』の 2-2 ページの「BEA Tuxedo ATMI 環境の基本アーキテクチャ」を参照してください。アプリケーションのプログラミングの詳細については、『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』、および COBOL を使用した BEA Tuxedo アプリケーションのプログラミングを参照してください。

BEA Tuxedo ATMI クライアントの作成

BEA Tuxedo のクライアントは、C または C++ などのプログラミング言語でプログラムを作成するのと同じように作成できます。BEA Tuxedo システムには、BEA Tuxedo ATMI と呼ばれる C 言語に基づいたプログラミング・インターフェイスが用意されています。ATMI は、BEA Tuxedo のクライアントとサーバの開発をサポートする使いやすいインターフェイスです。

注記　BEA Tuxedo ATMI では、COBOL インターフェイスもサポートされています。図 1-1 は、C/C++ API の例です。

クライアントのタスク

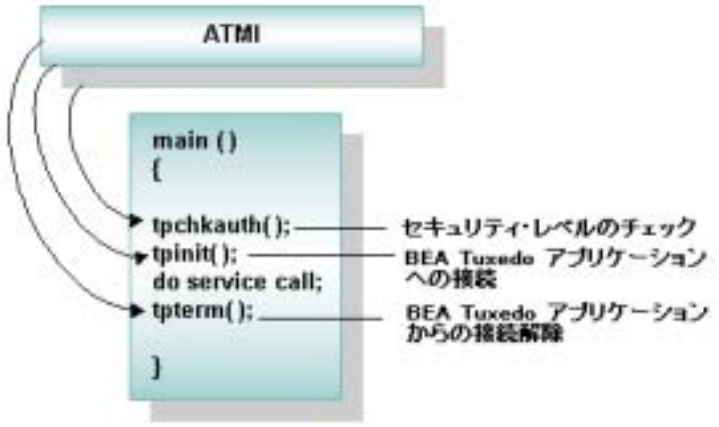
クライアントでは、次のような基本タスクが行われます。

- `tpchkauth()` を呼び出して、アプリケーションに参加するために必要なセキュリティ・レベルを決定します。戻り値として返されるセキュリティ・レベルには、セキュリティなし、アプリケーション・パスワード、アプリケーション認証、アクセス制御リスト、リンク・レベル暗号化、

公開鍵の暗号化、監査などがあります。これらの値は、セキュリティ・レベルを設定している場合にのみ返されます。

- `tpinit()` を呼び出して、BEA Tuxedo アプリケーションに接続します。必要なセキュリティ情報は、`tpinit()` の引数としてアプリケーションに渡されます。
- サービスを要求します。
- `tpterm()` を呼び出して、BEA Tuxedo アプリケーションとの接続を解除します。

図 1-1 クライアントが実行するタスク



関連項目

- 『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』の 4-1 ページの「クライアントのコーディング」
- 『BEA Tuxedo CORBA アプリケーションのセキュリティ機能』の 2-1 ページの「セキュリティの管理」
- 1-7 ページの「アプリケーションでの BEA Tuxedo メッセージング・パラダイム」

- 『BEA Tuxedo システム入門』の 2-27 ページの「型付きバッファ」
- 『BEA Tuxedo システム入門』の 2-5 ページの「ATMI の使用」

BEA Tuxedo ATMI サーバの作成

BEA Tuxedo のクライアントとサーバは、ATMI プログラミング・インターフェイスを使用して作成できます。ただし、BEA Tuxedo サーバは完全なプログラム、つまり標準的な `main` 関数を使ったプログラムとして開発されたものではありません。代わりに、アプリケーション開発者は、サービスと呼ばれる特定のビジネス関数を使用します。この関数は、BEA Tuxedo のバイナリ形式でコンパイルされて、実行可能サーバが生成されます。

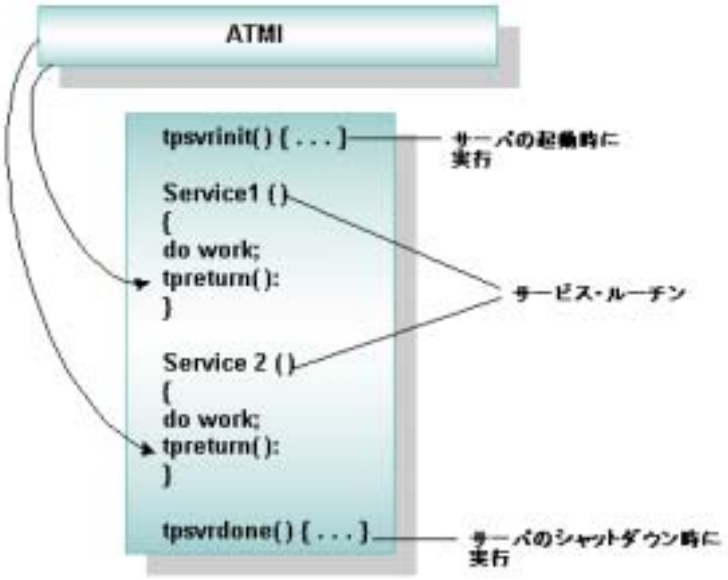
BEA Tuxedo サーバが起動すると、シャットダウン・メッセージを受信するまで稼動し続けます。通常、BEA Tuxedo サーバはシャットダウンして再起動するまでに、何千ものサービス呼び出しを行います。

サーバのタスク

- アプリケーション開発者が記述したコードにより、BEA Tuxedo ATMI のサーバは起動時に `tpsvrinit()` 関数を呼び出します。プログラマは、この関数を使用して、後で使用するアプリケーション・リソース (データベースなど) を開きます。
- アプリケーション開発者が記述したコードにより、BEA Tuxedo ATMI のサーバは、シャットダウン時に `tpsvrdone()` 関数を呼び出します。プログラマは、この関数を使用して、`tpsvrinit()` によって開かれたアプリケーション・リソースを閉じます。
- アプリケーション開発者が記述したコードにより、BEA Tuxedo ATMI のサーバは、クライアントの要求を処理するアプリケーション・サービスを要求します。BEA Tuxedo ATMI のクライアントは、*サーバ*を名前で呼び出すのではなく、*サービス*を呼び出します。BEA Tuxedo ATMI のクライアントは、要求を処理するサーバの場所を認識しません。

- ATMI サーバは、tpreturn() 関数を呼び出してサービス要求を終了させ、必要に応じて呼び出し元のクライアントにバッファを返します。

図 1-2 サーバが実行するタスク



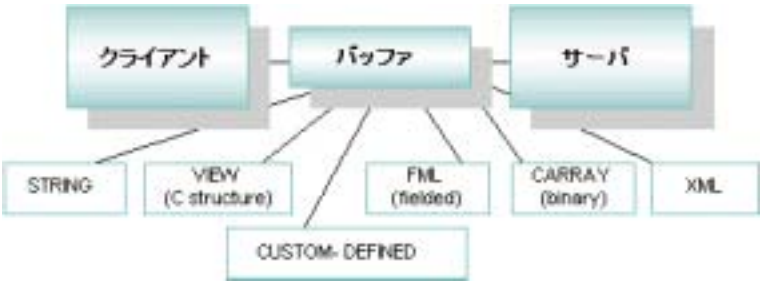
関連項目

- 『C 言語を使用した BEA Tuxedo アプリケーションのプログラミング』の 5-1 ページの「サーバのコーディング」
- 1-7 ページの「アプリケーションでの BEA Tuxedo メッセージング・パラダイム」
- 『BEA Tuxedo システム入門』の 2-27 ページの「型付きバッファ」
- 『BEA Tuxedo システム入門』の 2-5 ページの「ATMI の使用」

アプリケーションの型付きバッファ

BEA Tuxedo システムでは、すべての通信が型付きバッファを使用して行われます。BEA Tuxedo システムには通信を行うための各種のバッファ型があり、アプリケーション開発者がその型を選択することができます。BEA Tuxedo システムで渡されるすべてのバッファには特殊なヘッダがあり、BEA Tuxedo ATMI (`tpalloc()`、`tprealloc()`、および `tpfree()`) を介してバッファの割り当てや解放が行われます。

図 1-3 各種のバッファ型



型付きバッファ機能によって同じバッファ型を共有できるようになり、どの種類のネットワークやプロトコルでも、BEA Tuxedo システムでサポートされるどの種類の CPU アーキテクチャやオペレーティング・システムにも、データを送ることができるようになります。分散型の環境で型付きバッファを使用すると、各種の通信ネットワークを介してリンクされた異種コンピュータ間でデータを転送する場合に、クライアントおよびサーバでデータを準備するための処理が簡略化されます。そのため、プログラマはこのような機能をプログラムに組み込む必要がなくなり、本来のビジネス・ロジックの開発に集中することができます。

関連項目

- 『BEA Tuxedo システム入門』の 2-27 ページの「型付きバッファ」

アプリケーションでの BEA Tuxedo メッセージング・パラダイム

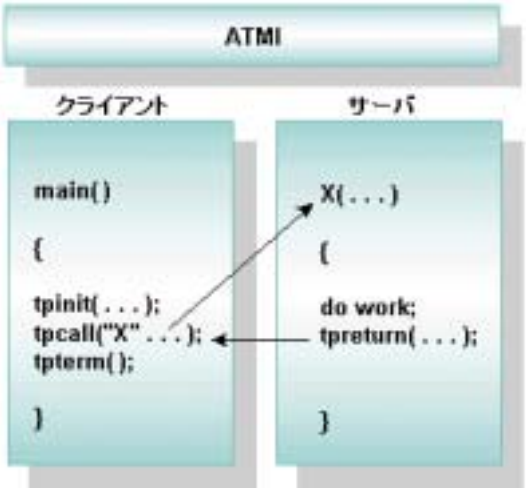
BEA Tuxedo ATMI には、アプリケーションで使用できる次のような通信モデルがあります。

- 要求 / 応答型モデル (同期呼び出し)
- 要求 / 応答型モデル (非同期呼び出し)
- 入れ子になった呼び出し
- 転送された呼び出し
- 会話型通信
- 任意通知型通知
- イベント・ベースの通信
- キュー・ベースの通信
- トランザクション

要求 / 応答型モデル (同期呼び出し)

同期呼び出しを行う場合、BEA Tuxedo ATMI クライアントは ATMI 関数 `tpcall()` を使用して、BEA Tuxedo ATMI サーバに要求を送ります。この関数は、BEA Tuxedo サーバを名前で呼び出すのではなく、指定されたサービスを呼び出します。指定されたサービスは、そのサービスを提供できる任意のサーバによって提供されます。クライアントは、送信したサービス要求が処理されるまで待機します。この要求に対する応答を受け取るまで、クライアントはほかの操作を行うことができません。つまり、クライアントでは、応答を受信するまで操作が「ブロック」されます。

図 1-4 同期要求 / 応答型モデル



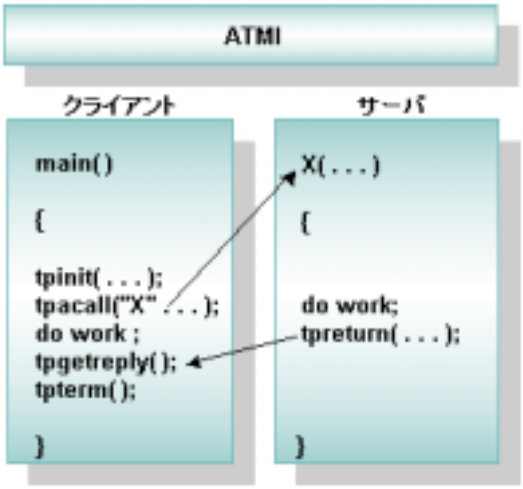
関連項目

- 『BEA Tuxedo システム入門』の 2-13 ページの「要求 / 応答型通信」

要求 / 応答型モデル (非同期呼び出し)

非同期呼び出しを行う場合、クライアントは 2 つの ATMI 関数を呼び出します。つまり、`tpacall(3c)` 関数を使ってサービスを要求し、`tpgetrply(3c)` 関数を使って応答を取得します。この方法は、クライアントが要求を送り、その応答を受け取るまでに別の操作を行う場合に使用します。

図 1-5 非同期呼び出し



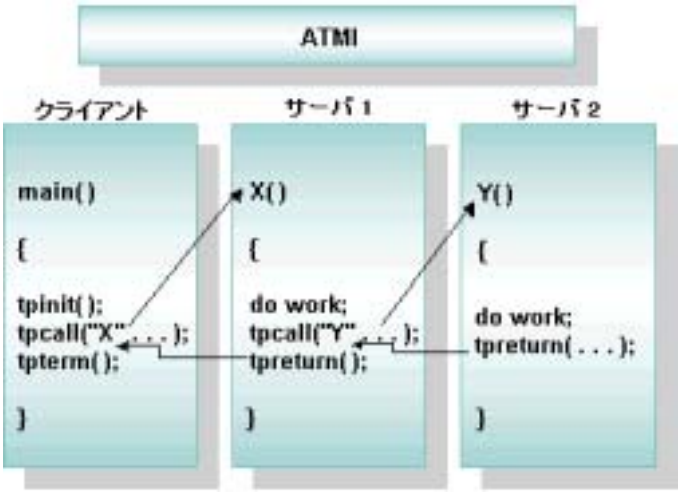
関連項目

- 『BEA Tuxedo システム入門』の 2-13 ページの「要求 / 応答型通信」

入れ子になった呼び出し

サービスは、BEA Tuxedo ATMI クライアントとして動作して、別の BEA Tuxedo サービスを呼び出すことができます。つまり、サービスを要求して、そのサービスが別のサービスを要求することができます。たとえば、BEA Tuxedo クライアントがサービス X を呼び出し、その応答を待ちます。その後、サービス X はサービス Y を呼び出し、その応答を待ちます。サービス X が応答を受け取ると、サービス X は呼び出し元のクライアントにその応答を返します。この方法では、サービス X がサービス Y からの応答を取得し、その応答になんらかの処理を行い、返されたバッファを変更してから最終的な応答をクライアントに返すことができます。

図 1-6 入れ子になった呼び出し



関連項目

- 『BEA Tuxedo システム入門』の 2-21 ページの「入れ子になった要求」

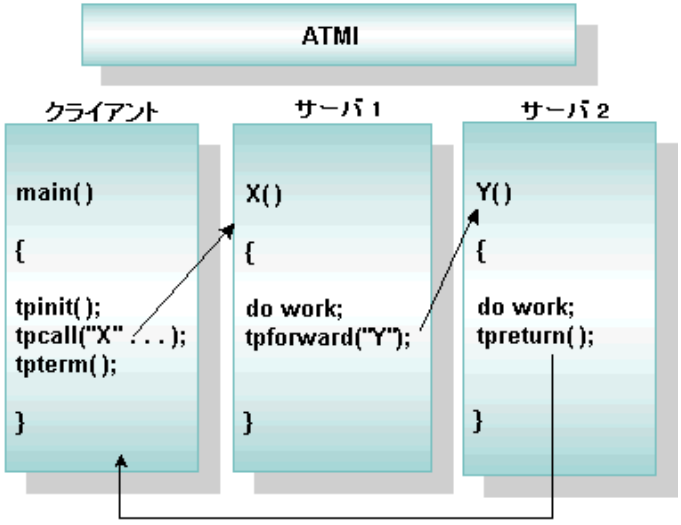
転送された呼び出し

呼び出しを転送すると、入れ子になったサービスが、最初に呼び出されたサービスを経由せずに直接 ATMI クライアントに応答を返すことができます。このため、最初のサービスはほかの要求を処理できるようになります。この機能は、最初に呼び出されたサービスが配信エージェントとしてのみ機能し、入れ子になったサービスから返された応答にデータを追加しない場合に使用します。

呼び出しを転送する場合、クライアントによって呼び出されたサービスは tpforward(3c) 関数を使用して、別のサーバ Y に要求を渡します。このような処理でのみ、BEA Tuxedo サービスは tpreturn(3c) を呼び出さずに、サービス呼び出しを終了することができます。

呼び出しの転送は、クライアントに透過的に行われます。つまり、同じクライアント・コードを使用して、1 つのサービスによって処理されるサービス要求も、複数のサービスによって処理されるサービス要求も扱うことができます。

図 1-7 転送された呼び出し



関連項目

- 『BEA Tuxedo システム入門』の 2-23 ページの「要求の転送」

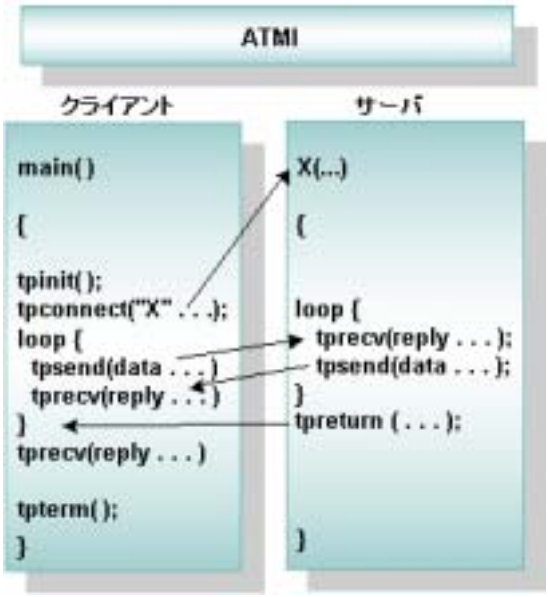
会話型通信

BEA Tuxedo ATMI クライアントとサービス間で、状態情報を保持して複数のバッファを送信する必要がある場合は、BEA Tuxedo の会話型通信を使用します。

サーバが一度会話に参加すると、会話が終了するまでそのサーバを利用できなくなるため、BEA Tuxedo 会話型通信は慎重に使用します。会話型通信をインプリメントするには、次の手順をコードに記述します。

1. BEA Tuxedo クライアントは、`tpconnect()` 関数によって会話を開始します。
2. BEA Tuxedo クライアントおよび会話型サーバは、`tpsend()` と `tprecv()` 関数を使用してバッファを交換します。サービス呼び出しには特定のフラグが設定され、クライアントとサーバのどちらが会話を制御しているのかが示されます。
3. サーバが `tpreturn()` 関数または `tpdiscon()` 関数を呼び出すと、会話は正常に終了します。

図 1-8 会話型通信



関連項目

- 『BEA Tuxedo システム入門』の 2-14 ページの「会話型通信」

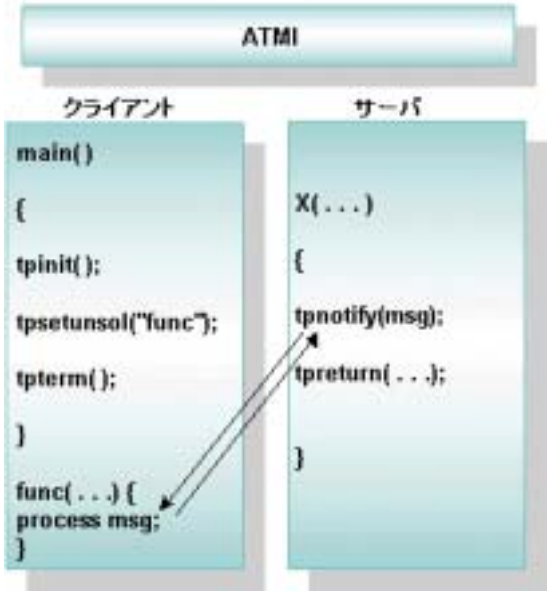
任意通知型通知

任意通知型通知を有効にするには、BEA Tuxedo ATMI クライアントで `tpsetunsol()` 関数を使用して任意通知型メッセージ・ハンドルを作成します。任意通知型メッセージを送信する場合、BEA Tuxedo クライアントまたはサーバは、単一のクライアントにメッセージを送信するときは `tpnotify()` 関数、複数のクライアントに同時にメッセージを送信するとき

は tpbroadcast() 関数を使用します。クライアントがメッセージを受信すると、BEA Tuxedo システムはクライアントの任意通知型ハンドラ関数を呼び出します。

シグナル・ベースのシステムでは、クライアントは任意通知型メッセージをポーリングする必要はありません。ただし、非シグナル・ベースのシステムでは、クライアントは tpchkunsol() 関数を使用して、任意通知型メッセージを確認する必要があります。クライアントがサービス要求を作成すると、暗黙的に tpchkunsol() が呼び出されます。

図 1-9 任意通知型通知による処理



注記　tpack フラグ・ビットを設定して tpnotify() を呼び出すと、要求に対する肯定応答が送信されます。

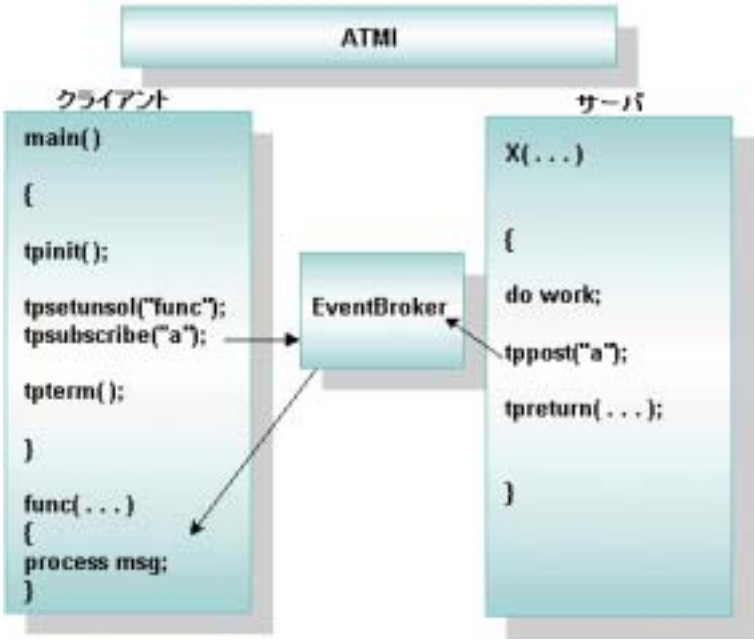
関連項目

- 『BEA Tuxedo システム入門』の 2-19 ページの「任意通知型通信」

イベント・ベースの通信

イベント・ベースの通信では、イベントをアプリケーション・キュー、ログ・ファイル、およびシステム・コマンドにポストできます。BEA Tuxedo ATMI クライアントは、`tpsubscribe()` 関数を使用してユーザ定義のイベントをサブスクライブし、BEA Tuxedo サービスまたはクライアントが `tppost()` 関数を呼び出したときに任意通知型メッセージを受信することができます。また、ATMI クライアントは、BEA Tuxedo システムがイベントを検出したときに呼び出されるシステム定義のイベントをサブスクライブすることもできます。たとえば、サーバが異常終了すると、`.SysServerDied` イベントがポストされます。このイベントのポストは BEA Tuxedo システムによって行われるため、アプリケーション・サーバがポストする必要がありません。

図 1-10 イベント・ベースの通信



関連項目

- 『BEA Tuxedo システム入門』の 2-18 ページの「イベントの通知」

キュー・ベースの通信

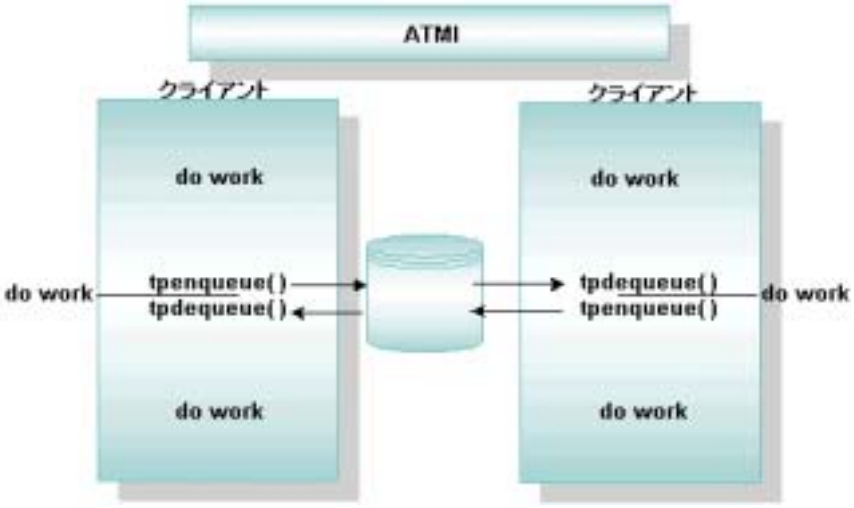
/Q システムとのインターフェイスとして、BEA Tuxedo クライアントは 2 つの ATMI 関数を使用します。つまり、キュー・スペースにメッセージを格納する `tpenqueue()` 関数と、キュー・スペースからメッセージを取得する `tpdequeue()` 関数を使用します。

次の図は、ピア・ツー・ピアの非同期メッセージングを表しています。この例では、クライアントは `tpenqueue()` を使用してサービスにメッセージを登録します。オプションで、`tpenqueue()` の呼び出しに、応答キューおよび異常終了キューの名前を指定できます。クライアントは、メッセージに付ける相関識別子の値を指定することもできます。この値は、キュー間で同じです。そのため、キューのメッセージに対応付けられたどの応答メッセージまたは異常終了メッセージも、応答キューまたは異常終了キューから読み取られるときに識別できます。

クライアントは、デフォルトのキューの順序付けを使用するか (たとえば、メッセージがキューから出されるまでの時間)、またはデフォルトのキューの順序付けの無効化を指定すること (たとえば、メッセージをキューの先頭に置くこと、またはキューの別のメッセージの前に置くこと) ができます。`tpenqueue()` は、`TMQUEUE` サーバにメッセージを送信し、そのメッセージは安定記憶領域のキューに登録され、肯定応答がクライアントに送信されます。肯定応答は、クライアントが直接確認することはできません。ただし、クライアントが正常な戻り値を取得すると、肯定応答が送られたと見なすことができます。異常終了の戻り値には、異常終了の内容についての情報が含まれています。キュー・マネージャによって割り当てられたメッセージ識別子は、アプリケーションに返されます。この識別子は、特定のメッセージをキューから取り出す場合に使用します。また、この識別子は、別の `tpenqueue()` 内で使用して、キューに次に登録されるメッセージの前にあるメッセージを識別することができます。

キューに登録されたメッセージをキューから取り出すには、そのメッセージをキューに登録しているトランザクションが正常にコミットされる必要があります。クライアントは tpdequeue() を使用して、キューからメッセージを取り出します。

図 1-11 ピア・ツー・ピアの非同期メッセージング・モデル

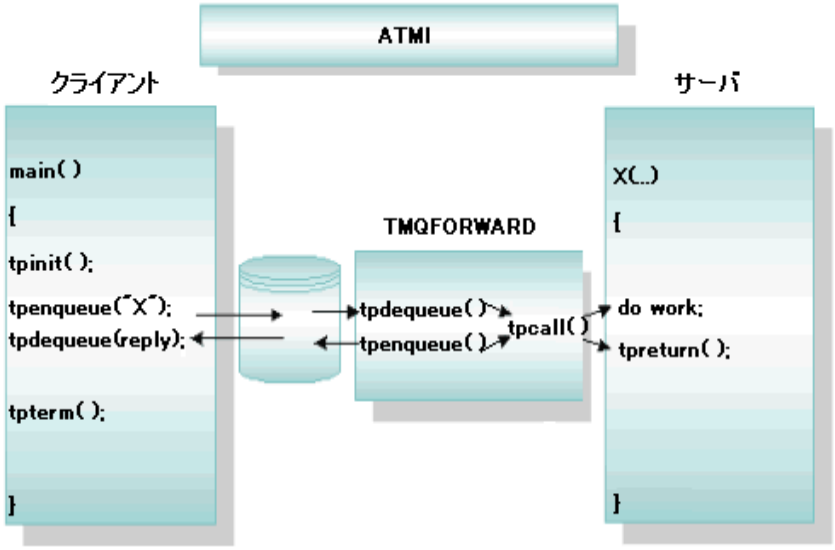


次の図は、別のサーバへのメッセージ転送を示しています。

クライアントはサーバ上のサービス X へのメッセージをキューに登録します。サービスは、メッセージがアクティブになった場合、およびメッセージの処理条件が満たされた場合に、メッセージを受信します。たとえば、メッセージが金曜日の午後 6 時にアクティブになるように符号化されている場合などに、メッセージを受け取ります。サービスが完了すると、キュー・スペースに応答が返されます。クライアントは、そこから応答を取得します。

この方法は、サービスに透過的です。つまり、同じアプリケーション・コードを使用して、キューを経由してサービスを呼び出すことも、tp(a)call 関数を使って直接サービスを呼び出すこともできます。

図 1-12 キュー転送を使用したキュー・ベースのサービス呼び出し



関連項目

- 2-15 ページの「メッセージ・キューイング通信」『BEA Tuxedo システム入門』

トランザクション

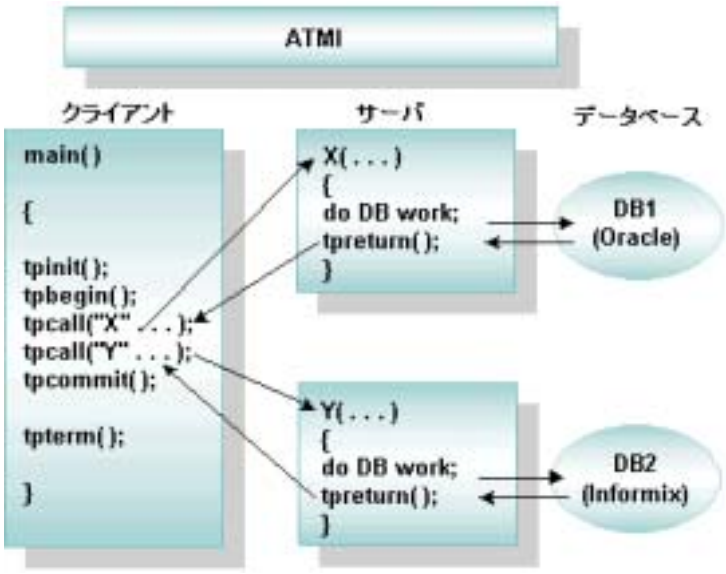
トランザクションをインプリメントする場合、アプリケーション・プログラマは次の 3 つの ATMI 関数を使用します。

- トランザクションを開始する `tpbegin()`
- 2 フェーズ・コミット・プロセスを開始する `tpcommit()`
- 即座にトランザクションを取り消す `tpabort()`

トランザクションに含まれるコードは、開始、コミット、およびアボートのみです。

次の例では、クライアントがトランザクションを開始し、2 つのサービスを要求し、トランザクションをコミットしています。サービス要求はトランザクションが開始されてからコミットされるまでの間に作成されるため、どちらのサービスも同じトランザクションに参加します。

図 1-13 トランザクション



関連項目

- 3-1 ページの「bankapp (複雑な C 言語アプリケーション) のチュートリアル」
- 4-1 ページの「CSIMPAPP (簡単な COBOL アプリケーション) のチュートリアル」
- 2-1 ページの「simpapp (簡単な C 言語アプリケーション) のチュートリアル」

- 5-1 ページの「STOCKAPP (複雑な COBOL アプリケーション) のチュートリアル」

2 simpapp (簡単な C 言語アプリケーション) のチュートリアル

ここでは、次の内容について説明します。

- simpapp とは
- simpapp のファイルおよびリソースの準備
  - ステップ 1: simpapp ファイルのコピー
  - ステップ 2: クライアント・プログラムの検証およびコンパイル
  - ステップ 3: サーバ・プログラムの検証およびコンパイル
  - ステップ 4: コンフィギュレーション・ファイルの編集とロード
  - ステップ 5: アプリケーションの起動
  - ステップ 6: ランタイム・アプリケーションの実行
  - ステップ 7: ランタイム・アプリケーションの監視
  - ステップ 8: アプリケーションのシャットダウン

simpapp とは

simpapp は、1 クライアントと 1 サーバで構成される ATMI サンプル・アプリケーションです。このアプリケーションは、BEA Tuxedo ソフトウェアに同梱されています。サーバでは、1 つのサービスだけが実行されます。つまり、クライアントから小文字の英字文字列を受け取り、その文字列を大文字で返します。

simpapp のファイルおよびリソースの準備

ここでは、BEA Tuxedo の ATMI アプリケーションを開発して実行するための手順を順に示します。次の図は、それらの作業をまとめたものです。各作業をクリックすると、その作業を行う手順が表示されます。

図 2-1simpapp の開発プロセス

はじめに

このチュートリアルのサンプル・アプリケーションを実行するには、BEA Tuxedo ATMI のクライアント / サーバ・ソフトウェアがインストールされ、ここで説明するファイルやコマンドを使用できることが必要です。既にインストールされている場合は、ソフトウェアのインストール先のディレクトリのパス名 (`TUXDIR`) を確認する必要があります。また、BEA Tuxedo ディレクトリ構造内のディレクトリとファイルに読み取りパーミッションと実行パーミッションを設定し、`simpapp` の各ファイルをコピーしたり、BEA Tuxedo の各コマンドを実行できるようにします。

このチュートリアルについて

ここで説明する `simpapp` の手順は、UNIX システムに基づいています。UNIX オペレーティング・システム環境でのプラットフォーム固有の手順はそのまま適用できますが、`simpapp` ファイルのコピーや環境変数の設定など

の手順は Windows 2000 など UNIX 以外のプラットフォームでは異なる場合があります。このため、チュートリアルで示す例は、お使いのプラットフォームに適さない場合があります。

このチュートリアルの目的

ここで説明する作業を行うと、ATMI クライアントおよびサーバが実行できるタスクについて理解し、環境に応じてコンフィギュレーション・ファイルを変更でき、`tmadmin` を呼び出してアプリケーションの動作を確認できるようになります。つまり、BEA Tuxedo アプリケーションの基本的な要素 (クライアント・プロセス、サーバ・プロセス、コンフィギュレーション・ファイルなど) を理解し、BEA Tuxedo システムの各コマンドを使用してアプリケーションを管理できるようになります。

ステップ 1: simpapp ファイルのコピー

注記　以下に示す手順は、UNIX システムに基づいています。Windows 2000 など UNIX 以外のプラットフォームでは、手順が異なる場合があります。サンプル・アプリケーションで使用されているコード例は、プラットフォームによって内容が異なります。

1. `simpapp` 用にディレクトリを作成し、そのディレクトリに移動 (`cd`) します。

   ```
   mkdir simpdir
   cd simpdir
   ```

   注記　この作業は省略せずに行ってください。この作業を行うと、最初からあった `simpapp` のファイルと、手順に従って作成したファイルを確認できるようになります。`csh` を使用せずに、標準シェル (`/bin/sh`) または Korn シェルを使用してください。

2. 環境変数を設定し、エクスポートします。

   `TUXDIR=`*`BEA Tuxedo システムのルート・ディレクトリのパス名`*
   `TUXCONFIG=`*`現在の作業ディレクトリのパス名`*

```
PATH=$PATH:$TUXDIR/bin
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$TUXDIR/lib
  export TUXDIR TUXCONFIG PATH LD_LIBRARY_PATH
```

TUXDIR および PATH を設定して、BEA Tuxedo ディレクトリ構造内のファイルにアクセスでき、BEA Tuxedo コマンドを実行できるようにします。Sun Solaris では、PATH の先頭に /usr/5bin を指定する必要があります。AIX (RS/6000) では、LD_LIBRARY_PATH ではなく LIBPATH を使用します。HP-UX (HP 9000) では、LD_LIBRARY_PATH ではなく SHLIB_PATH を使用します。

コンフィギュレーション・ファイルをロードするには、TUXCONFIG を設定する必要があります。詳細については、2-13 ページの「ステップ 4: コンフィギュレーション・ファイルの編集とロード」*を参照してください。*

3. simpapp ファイルをコピーします。

```
cp $TUXDIR/samples/atmi/simpapp/* .
```

注記　一部のファイルは、後で編集して実行可能ファイルを作成します。そのため、ソフトウェアに同梱のオリジナルのファイルではなく、そのコピーを使用することをお勧めします。

4. ファイルを一覧表示します。

```
$ ls
   README       env        simpapp.nt  ubbmp     wsimpcl
   README.as400 setenv.cmd simpcl.c    ubbsimple
   README.nt    simpapp.mk simpserv.c  ubbws
$
```

注記　README ファイルのほかに、UNIX システム以外のプラットフォーム用に simp*.* ファイルと ubb* ファイルがあります。README ファイルには、ほかのファイルの説明が記述されています。

次は、アプリケーションの核となる 3 つのファイルです。

- simpcl.c—クライアント・プログラムのソース・コード
- simpserv.c—サーバ・プログラムのソース・コード
- ubbsimple—このアプリケーション用のテキスト形式のコンフィギュレーション・ファイル

関連項目

- 2-2 ページの「simpapp とは」

ステップ 2: クライアント・プログラムの検証およびコンパイル

クライアント・プログラムの検証

ATMI クライアント・プログラムのソース・コードの内容を確認します。

```
$ more simpcl.c
```

以下に出力結果を示します。

コード リスト 2-1 simpcl.c **のソース・コード**

```
1     #include <stdio.h>
2     #include "atmi.h"            /* TUXEDO */
3
4
5
6
7     #ifdef __STDC__
8     main(int argc, char *argv[])
9
10    #else
11
12    main(argc, argv)
13    int argc;
14    char *argv[];
15    #endif
16
17    {
18
```

```
19              char *sendbuf, *rcvbuf;
20              int sendlen, rcvlen;
21              int ret;
22
23              if(argc != 2) {
24                      fprintf(stderr, "Usage: simpcl string\n");
25                      exit(1);
26              }
27              /* クライアント・プロセスとして BEA TUXEDO にアタッチします。*/
28              if (tpinit((TPINIT *) NULL) == -1) {
29                      fprintf(stderr, "Tpinit failed\n");
30                      exit(1);
31              }
32              sendlen = strlen(argv[1]);
33             if((sendbuf = (char *)tpalloc("STRING", NULL, sendlen+1))== NULL){
34                      fprintf(stderr,"Error allocating send buffer\n");
35                      tpterm();
36                      exit(1);
37              }
38              if((rcvbuf = (char *)tpalloc("STRING", NULL, sendlen+1))== NULL){
39                      fprintf(stderr,"Error allocating receive buffer\n");
40                      tpfree(sendbuf);
41                      tpterm();
42                      exit(1);
43              }
44              strcpy(sendbuf, argv[1]);
45              ret = tpcall("TOUPPER", sendbuf, NULL, &rcvbuf, &rcvlen, 0);
46              if(ret == -1) {
47                     fprintf(stderr, "Can't send request to service TOUPPER\n");
48                      fprintf(stderr, "Tperrno = %d, %s\n", tperrno,
49                                tmemsgs[tperrno]);
50                      tpfree(sendbuf);
51                      tpfree(rcvbuf);
52                      tpterm();
53                      exit(1);
54              }
55              printf("Returned string is: %s\n", rcvbuf);
56
57              /* バッファを解放し、BEA TUXEDO からのアタッチを解除します。*/
58              tpfree(sendbuf);
59              tpfree(rcvbuf);


60              tpterm();
61      }
```

表 2-1simpcl.c ソース・コードでの重要なコード行

行数	ファイル / 関数	目的
2	atmi.h	BEA Tuxedo の ATMI 関数を使用する場合に、常に必要となるヘッダ・ファイル。
28	tpinit()	クライアント・プログラムがアプリケーションに参加する際に使用される ATMI 関数。
33	tpalloc()	型付きバッファの割り当てに使用される ATMI 関数です。STRING は、BEA Tuxedo の 5 つの基本的なバッファ型のひとつです。NULL は、サブタイプの引数がないことを示します。最後の引数 sendlen + 1 は、バッファの長さを指定します。文字列の終わりを示す NULL 文字用に 1 が加えられています。
38	tpalloc()	応答メッセージにほかのバッファを割り当てます。
45	tpcall()	最初の引数に指定された TOUPPER サービスにメッセージ・バッファを送信します。また、応答バッファのアドレスも指定します。tpcall() はメッセージが返されるまで待機します。
35、41、52、60	tpterm()	アプリケーションを終了するための ATMI 関数。tpterm() は、エラー条件の発生により終了する (36、42、53 行目) 前に、アプリケーションを終了する場合に使用します。tpterm() への最終呼び出し (60 行目) は、メッセージが出力された後に行われます。
40、50、51、58、59	tpfree()	バッファの割り当てを解放します。tpfree() は、tpalloc() とは反対の処理を行います。
55	printf()	プログラムの正常終了を示します。サーバから返されたメッセージを出力します。

クライアント・プログラムのコンパイル

1. buildclient を実行して、クライアント・プログラムをコンパイルします。

   ```
   buildclient -o simpcl -f simpcl.c
   ```

   simpcl は出力ファイル、simpcl.c は入力ソース・ファイルです。

2. 結果を確認します。

   ```
   $ ls -l
   total 97
   -rwxr-x--x 1 usrid   grpid  313091 May 28 15:41 simpcl
   -rw-r----- 1 usrid   grpid    1064 May 28 07:51 simpcl.c
   -rw-r----- 1 usrid   grpid     275 May 28 08:57 simpserv.c
   -rw-r----- 1 usrid   grpid     392 May 28 07:51 ubbsimple
   ```

   simpcl という実行可能モジュールが作成されました。simpcl ファイルのサイズは一定ではありません。

関連項目

- 2-2 ページの「simpapp とは」
- 『BEA Tuxedo コマンド・リファレンス』の buildclient(1)
- *『BEA Tuxedo C リファレンス』*

ステップ 3: サーバ・プログラムの検証およびコンパイル

サーバ・プログラムの検証

ATMI サーバ・プログラムのソース・コードの内容を確認します。

```
$ more simpserv.c
```

コード リスト 2-2 simpserv.c **のソース・コード**

```
   */

/* #ident"@(#) apps/simpapp/simpserv.c$Revision: 1.1 $" */
1  #include <stdio.h>
2  #include <ctype.h>
3  #include <atmi.h>/* TUXEDO ヘッダ・ファイル */
4  #include <userlog.h>/* TUXEDO ヘッダ・ファイル */
5  /* tpsvrinit は、サーバの起動時
   ( 要求の処理を行う前 ) に実行されます。この関数は必須ではありません。
   この例では使用されていませんが、
   サーバのシャットダウン時に呼び出される tpsvrdone も使用できます。

9  */
10 #if defined(__STDC__) || defined(__cplusplus)

12 tpsvrinit(int argc, char *argv[])
13 #else
14 tpsvrinit(argc, argv)
15 int argc;
16 char **argv;
17 #endif
18     {
19         /* argc と argv が使用されないと、コンパイラによる警告メッセージが出力されます。
20          */
21        argc = argc;
22        argv = argv;
23        /* userlog は TUXEDO 中央メッセージ・ログに書き込みます。*/
```

```
24          userlog("Welcome to the simple server");
25         return(0);
26 }
27 /* この関数は、クライアントから要求された実際のサービスを実行します。
      引数には、
       データ・バッファへのポインタ、データ・バッファの長さなどを含む構造体を指定します。
30 */
31 #ifdef __cplusplus
32 extern "C"
33 #endif
34 void
35 #if defined(__STDC__) || defined(__cplusplus)
36 TOUPPER(TPSVCINFO *rqst)
37 #else
38 TOUPPER(rqst)
39 TPSVCINFO *rqst;
40 #endif
41  {
42        int i;
43
44       for(i = 0; i < rqst->len-1; i++)
45                rqst->data[i] = toupper(rqst->data[i]);
46        /* 転送されたバッファを要求の発信元に返します。*/
47       tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);
48 }
```

表 2-2simpserv.c ソース・コードでの重要なコード行

行数	ファイル / 関数	目的
ファイル全体		BEA Tuxedo のサーバには、main() 関数が含まれていません。main() 関数は、サーバのビルド時に BEA Tuxedo システムによって提供されます。

表 2-2simpserv.c ソース・コードでの重要なコード行

行数	ファイル / 関数	目的
12	tpsvrinit()	このサブルーチンは、サーバの初期化時 (サーバがサービス要求を処理する前) に呼び出されます。BEA Tuxedo システムで提供されるデフォルトのサブルーチンでは、サーバが起動したことを示すメッセージが USERLOG に書き込まれます。userlog(3c) は、BEA Tuxedo システムで使用されるログで、アプリケーションでも使用できます。
38	TOUPPER()	サービス (simpserv で唯一実行されるサービス) の宣言です。このサービスに必要な唯一の引数は、TPSVCINFO 構造体へのポインタです。この構造体には、大文字に変換する文字列データが格納されます。
45	for loop	TOUPPER を繰り返し呼び出して、入力を大文字に変換します。
49	tpreturn()	TPSUCCESS フラグに設定された値と、変換された文字列をクライアントに返します。

サーバ・プログラムのコンパイル

1. buildserver を実行して、ATMI サーバ・プログラムをコンパイルします。

```
buildserver -o simpserv -f simpserv.c -s TOUPPER
```

simpserv は作成される実行ファイル、simpserv.c は入力ソース・ファイルです。-s TOUPPER オプションは、サーバの起動時に宣言されるサービスを指定します。

2. 結果を確認します。

```
$ ls -l
total 97
-rwxr-x--x 1  usrid grpid 313091    May 28 15:41 simpcl
-rw-r----- 1  usrid grpid   1064    May 28 07:51 simpcl.c
-rwxr-x--x 1  usrid grpid 358369    May 29 09:00 simpserv
-rw-r----- 1  usrid grpid    275    May 28 08:57 simpserv.c
-rw-r----- 1  usrid grpid    392    May 28 07:51 ubbsimple
```

simpserv という実行可能モジュールが作成されました。

関連項目

- 2-2 ページの「simpapp とは」
- 『BEA Tuxedo コマンド・リファレンス』の buildserver(1)
- 『BEA Tuxedo C リファレンス』

ステップ 4: コンフィギュレーション・ファイルの編集とロード

コンフィギュレーション・ファイルの編集

1. テキスト・エディタで、simpapp のコンフィギュレーション・ファイル ubbsimple の内容を確認します。

コード リスト 2-3　simpapp **コンフィギュレーション・ファイル**

```
1$
2
3 #BEA Tuxedo サンプル・アプリケーションのスケルトン・ファイルです。
```

```
4 #< 山かっこで囲まれた文字列 > を適切な値で置き換えます。
5 RESOURCES
6 IPCKEY            <32,768 よりも大きい IPC キーで置き換えます。>
7
8 # 例 :
9
10 #IPCKEY         62345
11
12 MASTER          simple
13 MAXACCESSERS    5
14 MAXSERVERS      5
15 MAXSERVICES     10
16 MODEL           SHM
17 LDBAL           N
18
19 *MACHINES
20
21 DEFAULT:
22
23         APPDIR="< 現在のパス名で置き換えます。>"
24         TUXCONFIG="<TUXCONFIG のパス名で置き換えます。>"
25         TUXDIR="<Tuxedo のルート・ディレクトリ (/ 以外 ) を指定します。>"
26 # 例 :
27 #       APPDIR="/usr/me/simpdir"
28 #       TUXCONFIG="/usr/me/simpdir/tuxconfig"
29 #       TUXDIR="/usr/tuxedo"
30
31 <Machine-name>  LMID=simple
32 # 例 :
33 #tuxmach        LMID=simple
34 *GROUPS
35 GROUP1
36              LMID=simple   GRPNO=1  OPENINFO=NONE
37
38 *SERVERS
39  DEFAULT:
40              CLOPT="-A"
41  simpserv    SRVGRP=GROUP1 SRVID=1
42 *SERVICES
43  TOUPPER
```

2. 山かっこで囲まれた各 < 文字列 > を適切な値に置き換えます。

コンフィギュレーション・ファイルのロード

1. tmloadcf を実行して、コンフィギュレーション・ファイルをロードします。

```
$ tmloadcf ubbsimple
Initialize TUXCONFIG file: /usr/me/simpdir/tuxconfig [y, q] ? y
$
```

2. 結果を確認します。

```
$ ls -l
total 216
-rwxr-x--x 1 usrid grpid 313091    May 28 15:41 simpcl
-rw-r----- 1 usrid grpid   1064    May 28 07:51 simpcl.c
-rwxr-x--x 1 usrid grpid 358369    May 29 09:00 simpserv
-rw-r----- 1 usrid grpid    275    May 28 08:57 simpserv.c
-rw-r----- 1 usrid grpid 106496    May 29 09:27 tuxconfig
-rw-r----- 1 usrid grpid    382    May 29 09:26 ubbsimple
```

TUXCONFIG というファイルが作成されました。TUXCONFIG ファイルは、BEA Tuxedo システムで制御される新しいファイルです。

関連項目

- 2-2 ページの「simpapp とは」
- 『BEA Tuxedo コマンド・リファレンス』の tmloadcf(1)
- 『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5)

ステップ 5: アプリケーションの起動

1. tmboot を実行して、アプリケーションを起動します。

```
$ tmboot
Boot all admin and server processes?(y/n): y
Booting all admin and server processes in
```

```
/usr/me/simpdir/tuxconfig
Booting all admin processes ...
   exec BBL -A:
   process id=24223 ...Started.

Booting server processes ...

   exec simpserv -A :
   process id=24257 ... Started.
2 processes started.
$
```

BBL は、アプリケーションの共用メモリを監視する管理プロセスです。simpserv は、要求を受け取るために継続的に実行している simpapp サーバです。

関連項目

- 2-2 ページの「simpapp とは」
- 『BEA Tuxedo コマンド・リファレンス』の tmboot(1)
- 『BEA Tuxedo アプリケーション実行時の管理』の 1-10 ページの「アプリケーションの起動」

ステップ 6: ランタイム・アプリケーションの実行

simpapp を実行するには、クライアントから要求を送信します。

```
$ simpcl “hello, world”
Returned string is:HELLO, WORLD
```

関連項目

- 2-2 ページの「simpapp とは」

ステップ 7: ランタイム・アプリケーションの監視

システム管理者は tmadmin コマンド・インタプリタを使用して、アプリケーションを調べ、動的に変更を加えることができます。tmadmin を実行するには、TUXCONFIG 変数を設定する必要があります。

tmadmin を使用すると、50 個以上のコマンドを解釈したり実行することができます。各コマンドの説明については、tmadmin(1) を参照してください。この例では、2 つの tmadmin コマンドを使用します。

1. 次のコマンドを入力します。

   ```
   $ tmadmin
   ```

   次が出力されます。

   ```
   tmadmin - Copyright (c) 1999 BEA Systems, Inc. All rights
   reserved.
   >
   ```

   注記　大なり記号 (>) は、tmadmin のプロンプトです。

2. printserver(psr) コマンドを入力して、サーバに関する情報を出力します。

```
> psr
a.out Name Queue Name  Grp Name ID RqDone Load Done Current Service
---------- ----------  -------- -- ------ --------- ---------------
BBL        531993      simple    0      0         0 ( IDLE )
simpserv   00001.00001 GROUP1    1      0         0 ( IDLE )
>
```

3. printservice(psc) コマンドを入力して、サービスに関する情報を出力します。

```
> psc
Service Name Routine Name a.out Name Grp Name ID Machine # Done Status
------------ ------------ ---------- -------- -- ------- ------ ------
TOUPPER      TOUPPER      simpserv   GROUP1    1 simple       - AVAIL
>
```

関連項目

- 2-2 ページの「simpapp とは」
- 『BEA Tuxedo コマンド・リファレンス』の tmadmin(1)

ステップ 8: アプリケーションのシャットダウン

1. tmshutdown を実行して、アプリケーションをシャットダウンします。

```
$ tmshutdown
Shutdown all admin and server processes?(y/n): y
Shutting down all admin and server processes in
/usr/me/simpdir/tuxconfig
Shutting down server processes ...

Server Id = 1 Group Id = GROUP1 Machine = simple:    shutdown
succeeded.
Shutting down admin processes ...

Server Id = 0 Group Id = simple Machine = simple:    shutdown
succeeded.
2 processes stopped.
$
```

2. ULOG の内容を確認します。

```
$ cat ULOG*
$
```

```
113837.tuxmach!tmloadcf.10261:CMDTUX_CAT:879:A new file system
has been created. (size = 32 4096-byte blocks)
113842.tuxmach!tmloadcf.10261:CMDTUX_CAT:871: TUXCONFIG file
/usr/me/simpdir/tuxconfig has been created
113908.tuxmach!BBL.10768:LIBTUX_CAT:262: std main starting
113913.tuxmach!simpserv.10925:LIBTUX_CAT:262: std main starting
113913.tuxmach!simpserv.10925: Welcome to the simple server
114009.tuxmach!simpserv.10925:LIBTUX_CAT:522: Default
tpsvrdone() function used.
114012.tuxmach!BBL.10768:CMDTUX_CAT:26:Exiting system
```

関連項目

- 2-2 ページの「simpapp とは」
- 『BEA Tuxedo コマンド・リファレンス』の `tmshutdown(1)`
- 『BEA Tuxedo C リファレンス』の `userlog(3c)`
- 『BEA Tuxedo アプリケーション実行時の管理』の 1-14 ページの「アプリケーションのシャットダウン」
- 『BEA Tuxedo アプリケーション実行時の管理』の 2-20 ページの「ユーザ・ログ (ULOG) とは」

3 bankapp (複雑な C 言語アプリケーション) のチュートリアル

ここでは、次の内容について説明します。

- bankapp とは
- bankapp について
- bankapp のファイルおよびリソースの準備
- bankapp の起動

bankapp とは

bankapp は、BEA Tuxedo ソフトウェアに同梱されている、銀行業務の ATMI サンプル・アプリケーションです。このアプリケーションでは、口座の開設と解約、残高照会、預け入れまたは引き出し、口座振り替えなどの銀行業務を行うことができます。

このチュートリアルについて

このチュートリアルでは、bankapp アプリケーションを開発して実行するための手順を順に示します。このチュートリアルで bankapp の「開発」を経験すると、独自のアプリケーションを開発できるようになります。

bankapp チュートリアルは、次の 3 つの節から構成されています。

- bankapp について
- bankapp のファイルおよびリソースの準備
- bankapp の起動

注記　ここで説明する内容は、アプリケーションの開発、管理、プログラミングに経験のある UNIX および Windows 2000 システムのユーザを対象としています。また、BEA Tuxedo ソフトウェアについて理解していることを前提としています。

bankapp について

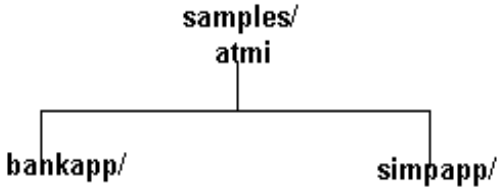
このサンプル・アプリケーションの手順は、UNIX または Windows 2000 環境で動作するシェル・スクリプト RUNME.sh と RUNME.cmd によって自動化されています。対応する readme ファイルに、これらのファイルの実行方法が説明されています。これらのファイルに目をとおし実行の手順を十分に理解した上で、ここで説明する内容に従って操作をすれば、分散アプリケーションの設定と管理をよりスムーズに行うことができます。

bankapp サンプル・アプリケーションでは、ソフトウェアに同梱のデモ版リレーショナル・データベースが使用されています。サンプル・アプリケーションに用意されたデモ用の各種コマンドと SQL コードを使用して、データベースにアクセスします。

このマニュアルでは、bankapp アプリケーションを構成するファイル、クライアント、およびサービスについて説明します。各作業をクリックすると、その作業を行う手順が表示されます。

bankapp について

bankapp アプリケーションを構成するファイルは、bankapp ディレクトリに置かれています。このディレクトリは、次の図のように構成されています。



銀行業務アプリケーションのファイルについて

bankapp ディレクトリには、次のファイルが置かれています。

- 埋め込み型の SQL 文を使用したサービス・サブルーチン用の 5 つのソース・ファイル
- 8 つの C 言語のソース・ファイル

- 要求 / 応答型クライアント・プログラム (audit)
- 会話型サーバ (AUDIT)
- 会話型クライアント (auditcon)
- 3 つのサーバ (またはサーバに対応付けられたファイル)
- アプリケーション用のデータまたはトランザクションを生成する 2 つのファイル
- その他のファイル
- BEA Tuxedo アプリケーションで共通のファイル (BEA Tuxedo アプリケーションの必須ファイル)
- 各種アドオンをコンパイルするための Makefile
- bankapp をサンプル・アプリケーションとして実行するためのファイル

次の表は、銀行業務アプリケーションを構成するファイルを示しています。BEA Tuxedo ソフトウェアに同梱のソース・ファイル、bankapp.mk の実行時に生成されるファイル、および各ファイルの簡単な説明がまとめられています。

表 3-1 銀行業務アプリケーションを構成するファイル

ソース・ファイル	生成されるファイル	フィールド値
ACCT.ec	ACCT.c、ACCT.o、ACCT	口座の開設と解約を行う OPEN_ACCT サービスと CLOSE_ACCT サービスが定義されたファイル。
ACCTMGR.c	ACCTMGR	イベント通知およびログ通知をサブスクライブするサーバ。WATCHDOG サービスおよび Q_OPENACCT_LOG サービスが定義されています。
AUDITC.c	AUDITC	auditcon クライアントからのサービス要求を処理する会話型サーバ。

ソース・ファイル	生成されるファイル	フィールド値
BAL.ec	BAL.c、BAL.o、BAL	audit クライアント・プログラムが銀行単位または支店単位で口座残高と窓口残高を取得するための ABAL サービス、TBAL サービス、ABAL_BID サービス、および TBAL_BID サービスが定義されたファイル。
BALC.ec	BALC.c、BALC.o、BALC	ABALC_BID サービスおよび TBALC_BID サービス。この 2 つのサービスは、前述の TBAL_BID および ABAL_BID サービスと同じです。ただし、支店番号が見つからない場合、TPSUCCESS が返されて auditcon の処理が続行します。
bankmgr.c	bankmgr	特に関心のあるイベントをサブスクライブするクライアント・プログラム。
BTADD.ec	BTADD.c、BTADD.o、BTADD	データベースに支店および窓口を追加するための BR_ADD サービスと TLR_ADD サービスが定義されたファイル。
cracl.sh	-	アクセス制御のセキュリティ・レベルを示すアクセス制御リスト (ACL) を生成するシェル・スクリプト。
crqueue.sh	-	イベント通知に使用するアプリケーション・キューを生成するシェル・スクリプト。
crusers.sh	-	認証セキュリティ・レベルで定義されるグループおよびユーザを生成するシェル・スクリプト。
event.flds	-	イベント機能で使用されるフィールド・テーブル・ファイル。
FILES	-	bankapp のすべてのファイルを説明したファイル・リスト。
README	-	Windows 2000 を除くすべてのプラットフォームでのインストールと起動の手順について説明したファイル。
README.nt	-	Windows 2000 プラットフォームでのインストールと起動の手順について説明したファイル。

ソース・ファイル	生成されるファイル	フィールド値
README2	-	新機能を実現するための bankapp の拡張部分について説明したファイル。このファイルは、samples/atmi/bankapp ディレクトリにあります。
README2.nt	-	Windows 2000 用の新機能を実現するための bankapp の拡張部分について説明したファイル。このファイルは、samples\atmi\bankapp ディレクトリにあります。
RUNME.cmd		Windows 2000 でアプリケーションをビルド、設定、起動、およびシャットダウンするための対話型スクリプト。
RUNME.sh	-	UNIX でアプリケーションをビルド、設定、起動、およびシャットダウンするための対話型スクリプト。
showq.sh!	-	メッセージ・キューの状態と内容を出力するシェル・スクリプト。
TLR.ec	TLR.c、TLR.o、TLR	WITHDRAWAL、DEPOSIT、および INQUIRY の 3 つのサービスがあります。
usrevtf.sh	-	BEA Tuxedo サーバの TMUSREVT に ENVFILE を生成するシェル・スクリプト。
XFER.c	XFER.o、XFER	TRANSFER サービスが定義されたファイル。
aud.v	aud.V、aud.h	audit クライアントと BAL サーバの間でやり取りされる構造体を定義する FML VIEW。
appinit.c	appinit.o	TLR を除くすべてのサーバを対象とする tpsvrinit() および tpsvrdone() のカスタマイズ・バージョン。
audit.c	audit.o、audit	ABAL、TBAL、ABAL_BID、および TBAL_BID サービスを使用して、銀行単位または支店単位で口座残高と窓口残高を取得するクライアント。
auditcon.c	auditcon	会話型モードで動作し、:ABAL サービス、TBAL サービス、ABALC_BID サービス、および TBALC_BID サービスを使用する対話型 audit プログラム。

ソース・ファイル	生成されるファイル	フィールド値
bankapp.mk	-	UNIX 用のアプリケーションの makefile。
bankapp.nt	-	Windows 2000 用のアプリケーションの makefile。
bank.flds	bank.flds.h	サーバに使用される銀行データベース・フィールドと FML 補助フィールドを含むフィールド・テーブル・ファイル。
bank.h	-	アプリケーション内の複数の C 言語プログラムに共通するデータ定義。
bankvar	-	bankapp の一部の環境変数が定義されたファイル。ほかの変数は ENVFILE で定義されています。ただし、ENVFILE は bankvar を参照するので、アプリケーションのすべての環境変数は bankvar で定義できます。
crbank.sh	crbank	bankapp を SHM モードで実行した場合に、全銀行のデータベースを生成するシェル・スクリプト。
crbankdb.sh	crbankdb	1 つのサーバ・グループで使用されるデータベースを生成するシェル・スクリプト。
crtlog.sh	crtlog、TLOG	マスタ・サイトで UDL と TLOG を生成し、非マスタ・サイトで UDL を生成するシェル・スクリプト。
driver.sh	driver	ud(1) を使用して、FML バッファを介してトランザクション要求を送り、アプリケーションの処理を行うシェル・スクリプト。
envfile.sh	envfile、ENVFILE	tmloadcf で使用する ENVFILE を生成するシェル・スクリプト。
gendata.c	gendata	ud で読み取り可能な要求を生成して、銀行を 10、窓口を 30、口座を 200 追加するシェル・プログラム。
gentran.c	gentran	ud で読み取り可能なトランザクション要求を DEPOSIT、WITHDRAWAL、TRANSFER、および INQUIRY サービスから生成するプログラム。

ソース・ファイル	生成されるファイル	フィールド値
populate.sh	populate	ud(1) を使用して、FML バッファに支店、窓口、および口座を追加する要求を格納し、それをデータベースに追加するシェル・スクリプト。
ubbmp	TUXCONFIG	MP モードのコンフィギュレーションで使用する UBBCONFIG のサンプル・ファイル。
ubbshm	TUXCONFIG	SHM モードのコンフィギュレーションで使用する UBBCONFIG のサンプル・ファイル。
util.c	util.o	サービスで共通して使用される getstrl() などの関数。
bankclt.c	bankclt	bankapp のクライアント。

関連項目

- 3-3 ページの「bankapp について」

bankapp クライアントの検証

bankclt.c ファイルとは

bankclt ファイルには、bankapp アプリケーションから BEA Tuxedo サービスを要求するクライアント・プログラムが定義されています。このクライアント・プログラムはテキスト形式で、次のオプションが提供されています。

- 残高照会
- 引き出し
- 預け入れ

- 振り替え
- 口座の開設
- 口座の解約
- アプリケーションの終了

アプリケーションの終了を除く各オプションでは、次のタスクを実行するサブルーチンが呼び出されます。

1. `get_account()`、`get_amount()`、`get_socsec()`、`get_phone()`、および `get_val()` 関数を使用して、ユーザによるキーボードからの入力を取得します。
2. グローバル FML バッファ (`*fbfr`) に値を追加します。関数によっては、ほかのフィールドを追加する場合もあります。これは、サーバで必要となる情報によって異なります。
3. `do_tpcall()` 関数を使用して BEA Tuxedo システムに要求を送信するルーチンが、必要なサービスを呼び出せるようにします。次の表は、関数とその関数によって呼び出されるサービスを示しています。

表 3-2 関数によって呼び出されるサービス

関数名	FML 入力フィールド	FML 出力フィールド	Service Name
`BALANCE()`	`ACCOUNT_ID`	`SBALANCE`	`INQUIRY`
`WITHDRAWAL()`	`ACCOUNT_ID` `SAMOUNT`	`SBALANCE`	`WITHDRAWAL`
`DEPOSIT()`	`ACCOUNT_ID` `SAMOUNT`	`SBALANCE`	`DEPOSIT`

`TRANSFER()`	`ACCOUNT_ID (0)`[1]
	`ACCOUNT_ID (1)`
	`SAMOUNT`
		`SBALANCE (0)`
		`SBALANCE (1)`
			`TRANSFER`

関数名	FML 入力フィールド	FML 出力フィールド	Service Name
OPEN_ACCT()	LAST_NAME FIRST_NAME MID_INIT SSN ADDRESS PHONE ACCT_TYPE BRANCH_ID SAMOUNT	ACCOUNT_ID SBALANCE	OPEN_ACCT
CLOSE_ACCT()	ACCOUNT_ID	SBALANCE	CLOSE_ACCT

[1] かっこ内の数値は、そのフィールドでの FML オカレンスの数です。

4. 処理が正常に完了すると、各関数は返された FML バッファから必要なフィールドを取得し、その結果を出力します。

以下は、do_tpcall() 関数 (bankclt.c の 447 行目以降) のコード例です。

コード リスト 3-1 bankclt.c **での** **do_tpcall() 関数のコード例**

```
/*
 * この関数は Tuxedo への tpcall を実行します。
 */
static int
do_tpcall(char *service)

{

        long len;
        char *server_status;
        /* グローバル・トランザクションの開始 */
        if (tpbegin(30, 0) == -1) {
                (void)fprintf(stderr, "ERROR: tpbegin failed (%s)\n",
                        tpstrerror(tperrno));
                return(-1);
        }
        /* Request the service with the user data */
        if (tpcall(service, (char *)fbfr, 0, (char **)&fbfr, &len,
                        0) == -1) {
```

```
                if(tperrno== TPESVCFAIL && fbfr != NULL &&
                    (server_status=Ffind(fbfr,STATLIN,0,0)) != 0) {
                       /* サーバの返信が失敗 */
                       (void)fprintf(stderr, "%s returns failure
                       (%s)\n",
                              service,server_status);
                }
                else {
                       (void)fprintf(stderr,
                              "ERROR: %s failed (%s)\n", service,
                              tpstrerror(tperrno));
                }
                /* トランザクションのアボート */
                (void) tpabort(0);
                return(-1);
        }
        /* トランザクションのコミット */
        if(tpcommit(0) < 0) {
                (void)fprintf(stderr, "ERROR: tpcommit failed
                       (%s)\n",
                       tpstrerror(tperrno));
                return(-1);
        }
        return(0);
}
```

do_tpcall() 関数は、次のタスクを実行します。

- tpbegin() を呼び出して、グローバル・トランザクションを開始します。これにより、すべての処理が 1 つの単位として実行されます。
- 要求されたサービス名 (char *service) と割り当てられた FML バッファ (グローバルな *fbfr ポインタ) を渡して、tpcall() を呼び出します。
- tpcall() がサーバ・エラー (TPSVCERR) が原因で失敗すると、サーバからのメッセージを STATLIN FML フィールドに出力します。tpabort() を使用してトランザクションをロールバックし、-1 を返します。
- tpcall() がほかのエラーが原因で失敗すると、そのエラーメッセージを出力します。tpabort() を使用してトランザクションをロールバックし、-1 を返します。

- tpcall() が正常に終了すると、tpcommit() を使用してトランザクションをコミットし、0 を返します。

注記　unsolfcn() 関数は、クライアントへの任意通知型メッセージがある場合に呼び出されます。この関数では、STRING 型バッファだけを使用でき、メッセージが出力されます。

bankapp での ud(1) の使い方

bankapp では、BEA Tuxedo プログラム ud(1) が使用されます。ud(1) は、標準入力からフィールド化バッファを読み取り、それをサービスに送信します。サンプル・アプリケーションでは、ud は populate および driver プログラムで使用されています。

- populate では、gendata と呼ばれるプログラムが、サービス要求と bankapp データベースに格納される顧客の口座情報を ud に送ります。
- driver プログラムでもデータの流れは同じです。ただし、使用されるプログラムは gentran で、アプリケーションにトランザクションを渡してアクティブなシステムをシミュレートします。

要求 / 応答クライアント :audit.c

audit は、ABAL、TBAL、ABAL_BID、および TBAL_BID サービスを使用して、銀行単位または支店単位で残高照会を行う要求 / 応答型のクライアント・プログラムです。このプログラムは、次の 2 とおりの方法で実行できます。

- audit [-a | -t]―銀行単位で全口座の総額、または銀行単位で全窓口の現金供給額を出力します。オプション -a または -t を使用して、口座残高または窓口残高のどちらを計算するのかを指定します。
- audit [-a | -t] branch_ID―支店単位の全口座の総額、または branch_ID で識別される支店の全窓口の現金供給額を出力します。オプション -a または -t を使用して、口座残高または窓口残高のどちらを計算するのかを指定します。

audit のソース・コードは、main() および sum_bal() サブルーチンから構成されます。BEA Tuxedo の ATMI 関数は、この両方で使用されます。このプログラムでは、VIEW 型バッファと aud.h ヘッダ・ファイルで定義された構造体が使用されます。構造体のソース・コードは、VIEW 記述ファイル aud.v に記述されています。

次の擬似コードは、このプログラムのアルゴリズムを示しています。

コード リスト 3-2 audit **の擬似コード**

```
 main()
{
     Parse command-line options with getopt();
     Join application with tpinit();
     Begin global transaction with tpbegin();
     If (branch_ID specified) {
               Allocate buffer for service requests with tpalloc();
               Place branch_ID into the aud structure;
               Do tpcall() to "ABAL_BID" or "TBAL_BID";
               Print balance for branch_ID;
               Free buffer with tpfree();
     }
     else /* branch_ID が指定されていない場合 */
               call subroutine sum_bal();
     Commit global transaction with tpcommit();
     Leave application with tpterm();
}
sum_bal()
}
     Allocate buffer for service requests with tpalloc();
     For (each of several representative branch_ID's,
               one for each site)
               Do tpacall() to "ABAL" or "TBAL";
     For (each representative branch_ID) {
               Do tpgetrply() wtith TPGETANY flag set
                        to retrieve replies;
               Add balance to total;
               Print total balance;
     }
     Free buffer with tpfree();
}
```

以下は、audit ソース・コードの構成要素である main() および sum_bal() で行われる処理をまとめたものです。

main() 関数では、次の処理が行われます。

1. /* アプリケーションへの参加 */
2. /* グローバル・トランザクションの開始 */
3. /* バッファの作成、およびデータ・ポインタの設定 */
4. /* tpcall の実行 */
5. /* グローバル・トランザクションのコミット */
6. /* アプリケーションからの分離 */

sum_bal サブルーチンでは、次の処理が行われます。

1. /* バッファの作成、およびデータ・ポインタの設定 */
2. /* tpacall の実行 */
3. /* tpgetrply を使用した応答の検索 */

会話型クライアント :auditcon.c

auditcon は、audit プログラムの会話型バージョンです。auditcon のソース・コードでは、会話型通信のための ATMI 関数が使用されます。たとえば、クライアントとサーバ間の接続を確立する tpconnect()、メッセージを送信する tpsend()、メッセージを受信する tprecv() が使用されます。

次の擬似コードは、このプログラムのアルゴリズムを示しています。

コード リスト 3-3 auditcon **の擬似コード**

```
main()
{
        Join the application
        Begin a transaction
        Open a connection to conversational service AUDITC
        Do until user says to quit: {
                 Query user for input
                 Send service request
                 Receive response
                 Print response on user's terminal
                 Prompt for further input
        }

        Commit transaction
        Leave the application
}
```

イベントを監視するクライアント bankmgr.c

bankmgr は、bankapp の継続的に稼働するクライアントです。このプログラムは、新規口座の開設や 10,000 ドル以上の引き出しなど、アプリケーション定義のイベントで特に関心のあるものをサブスクライブします。bankmgr.c クライアントの詳細については、bankapp の README2 ファイル、または bankmgr.c コードを参照してください。

関連項目

- 3-3 ページの「bankapp について」
- 『BEA Tuxedo システム入門』の 2-5 ページの「ATMI の使用」
- 『BEA Tuxedo システム入門』の 2-11 ページの「BEA Tuxedo のメッセージング・パラダイム」
- 3-1 ページの「bankapp とは」
- 『BEA Tuxedo システム入門』の 2-27 ページの「型付きバッファ」
- 1-15 ページの「イベント・ベースの通信」
- 『BEA Tuxedo コマンド・リファレンス』
- 『BEA Tuxedo C リファレンス』

bankapp サーバと bankapp サービスの検証

ここでは、次の内容について説明します。

- `bankapp` で提供されるサーバおよびサービス
- サービスがサーバにリンクされ、編集されるしくみ
- BEA Tuxedo の `bankclt.c` またはアプリケーション・クライアント `audit.c` がアクセスする各サービスの擬似コード
- `bankapp` サービスとサーバの関係
- BEA Tuxedo システムで定義された `main()` 関数で各サーバをコンパイルおよびビルドする場合の `buildserver(1)` コマンド・オプション
- サーバを構築する別の方法

サーバとは、1 つ以上のサービスを提供する実行可能プロセスです。BEA Tuxedo システムでは、サーバはクライアントとして動作するプロセスから継続的に要求を受け取り、それを適切なサービスに転送します。サービスとは、アプリケーションの処理を行うために記述された C 言語のサブルーチンです。BEA Tuxedo アプリケーションは、サービスを提供し、リソース・マネージャにアクセスできるように作成されています。サービス・ルーチンは、BEA Tuxedo アプリケーション・プログラマが作成します。

データベースと直接やり取りは行わない TRANSFER サービスを除くすべての bankapp サービスのコードは、C 言語に埋め込み型 SQL 文が使用されています。TRANSFER サービスは、XFER サーバによって実行される C 言語プログラムです。つまり、ソース・ファイルは .ec ファイルではなく .c ファイルです。

bankapp のすべての bankapp サービスでは、ATMI で提供される関数を使用して、次の処理が行われます。

- 型付きバッファの管理
- ほかのサービスとの同期通信または非同期通信
- グローバル・トランザクションの定義
- リソース・マネージャへのアクセス
- クライアントへの応答

bankapp の要求 / 応答型サーバ

bankapp の 5 つのサーバは、応答 / 要求モードで動作します。このうちの 4 つは、埋め込み型 SQL 文を使用して、リソース・マネージャにアクセスします。これらのサーバのソース・ファイル (bankapp サンプル・アプリケーションのサブディレクトリにあります) には、ファイル名に拡張子として .ec が付いています。

5 番目のサーバ XFER は振り替えに使用されるサーバで、リソース・マネージャ自体への呼び出しは行いません。このサーバは、TLR サーバによって提供される WITHDRAWAL サービスおよび DEPOSIT サービスを呼び出し、口座間

の振り替えを行います。XFER では、リソース・ネージャへの呼び出しは行われず、埋め込み型 SQL 文が使用されていないので、XFER のソース・ファイルは .c ファイルです。

サーバ名	提供される機能
BTADD.ec	支店および窓口の記録を任意のサイトから適切なデータベースに追加します。
ACCT.ec	主な顧客サービスである口座の開設および解約 (OPEN_ACCT および CLOSE_ACCT) を行います。
TLR.ec	窓口サービスである WITHDRAWAL、DEPOSIT、および INQUIRY を実行します。TLR プロセスでは、サーバのコマンド行に指定された -T オプションによって、そのプロセスが TELLER ファイル内の実際の窓口であることが認識されます。
XFER.c	データベースの任意の口座間で振り替えを行います。
BAL.ec	データベースのすべての支店または特定の支店の口座残高を計算します。

bankapp 会話型サーバ

AUDITC は、会話型サーバの一例です。このサーバでは、AUDITC と呼ばれるサービスが提供されます。会話型クライアント auditcon は、AUDITC への接続を確立し、そこに監査情報を要求します。

AUDITC は要求を評価し、必要な情報を得るために適切なサービス (ABAL、TBAL、ABAL_BID、または TBAL_BID) を呼び出します。呼び出されたサービスから応答を受信すると、AUDITC はその応答を auditcon. に返します。会話型サーバのサービスは、要求 / 応答サービスを呼び出すことができます。別の会話型サーバとの接続を確立することもできますが、この機能は AUDITC では提供されていません。

bankapp サービス

bankapp では、12 種類の要求 / 応答サービスが提供されます。bankapp の各サービス名は、サーバのソース・コード内にある C 言語の関数名と一致します。

bankapp サービス	サービスを提供するサーバ	入力	処理内容
BR_ADD	BTADD	FML バッファ	■ 新規支店レコードを追加します。
TLR_ADD	BTADD	FML バッファ	■ 新規窓口レコードを追加します。
OPEN_ACCT	ACCT	FML バッファ	■ ACCOUNT ファイルにレコードを挿入し、DEPOSIT を呼び出して開設時の預け入れ額を加算します。 ■ 開設を行った窓口の BRANCH_ID に基づき、新しい口座用の ACCOUNT_ID を選択します。
CLOSE_ACCT	ACCT	FML バッファ	■ ACCOUNT レコードを削除します。 ■ ACCOUNT_ID を確認します。 ■ WITHDRAWAL を呼び出し、解約時の残高を削除します。

bankapp サービス	サービスを提供するサーバ	入力	処理内容
WITHDRAWAL	TLR	FML バッファ	■ 指定された支店、窓口、口座の残高から引き出し金額を差し引きます。 ■ ACCOUNT_ID および SAMOUNT フィールドを確認します。 ■ 口座および窓口から預金が引き出し可能であることを確認します。
DEPOSIT	TLR	FML バッファ	■ 指定された支店、窓口、口座の残高に預け入れ額を加算します。 ■ ACCOUNT_ID および SAMOUNT フィールドを確認します。
INQUIRY	TLR	FML バッファ	■ 口座残高を取得します。 ■ ACCOUNT_ID を確認します。
TRANSFER	XFER	FML バッファ	■ tpcall() を呼び出して WITHDRAWAL サービスを要求し、再度 tpcall() を呼び出して DEPOSIT サービスを要求します。
ABAL	BAL	aud.v の VIEW バッファ	■ 特定のサイトにおける全支店の口座残高を計算します。
TBAL	BAL	入力としての aud.v の VIEW バッファ	■ 特定のサイトにおける全支店の窓口残高を計算します。
ABAL_BID	BAL	入力としての aud.v の VIEW バッファ	■ 特定の BRANCH_ID の窓口残高を計算します。

bankapp サービス	サービスを提供するサーバ	入力	処理内容
TBAL_BID	BAL	入力としての aud.v の VIEW バッファ	■ 特定の BRANCH_ID の窓口残高を計算します。

bankapp サービスのアルゴリズム

次の擬似コードは、bankapp の各サービスで使用されるアルゴリズムを示しています。サービスには BR_ADD、TLR_ADD、OPEN_ACCT、CLOSE_ACCT、WITHDRAWAL、DEPOSIT、INQUIRY、TRANSFER、ABAL、TBAL、ABAL_BID、および TBAL_BID があります。このコード例を参考にすると、bankapp サーバのソース・コードを理解しやすくなります。

コード リスト 3-4　BR_ADD **の擬似コード**

```
void BR_ADD (TPSVCINFO *transb)
{
        TPSVCINFO データ・バッファへのポインタを設定 ;
        フィールド化バッファからサービス要求に関連するすべての値を取得 ;
        BRANCH にレコードを挿入 ;
        tpreturn() で成功を示す値を返す ;
}
```

コード リスト 3-5　TLR_ADD **の擬似コード**

```
void TLR_ADD (TPSVCINFO *transb)
{
        TPSVCINFO データ・バッファへのポインタを設定 ;
        フィールド化バッファからサービス要求に関連するすべての値を取得 ;
        支店の LAST_ACCT を読み取り TELLER_ID を取得 ;
        窓口レコードの挿入 ;
        LAST_TELLER で BRANCH を更新 ;
        tpreturn() で成功を示す値を返す ;
}
```

コード リスト 3-6　OPEN_ACCT **の擬似コード**

```
void OPEN_ACCT(TPSVCINFO *transb)
{
    Fget() および Fvall() で、フィールド化バッファからサービス要求に関連する
    すべての値を取得 ;
    預け入れ金額が正の値であるか確認し、正でない場合は tpreturn() で
   エラーを示す値を返す ;
    口座番号が妥当であるか確認し、妥当でなければ tpreturn() で
   エラーを示す値を返す ;
   トランザクション整合性レベルを、読み取り／書き込みに設定 ;
    支店の LAST_ACCT フィールドに基づき、BRANCH レコードを検索し、新たな
    口座番号を選択 ;
    ACCOUNT ファイルに新たな口座レコードを挿入 ;
    LAST_ACCT で BRANCH レコードを更新 ;
    tpalloc() で預け入れ要求バッファを割り当て、Finit() で初期化し
    FML タイプとして使用 ;
    DEPOSIT サービス要求への値を預け入れバッファに格納 ;
      サービスからの呼び出しにより DEPOSIT 要求の優先順位を上げる ;
    tpcall() で DEPOSIT サービスを呼び出し、初期残高を加算 ;
   必要な情報を含む応答バッファを用意 ;
    tpfree() で預け入れ要求バッファを解放 ;
   tpreturn() で成功を示す値を返す ;
}
```

コード リスト 3-7　CLOSE_ACCT **の擬似コード**

```
void CLOSE_ACCT(TPSVCINFO *transb)
{
   Fvall() でフィールド化バッファから口座番号を取得 ;
   口座番号が妥当であるか確認し、妥当でなければ tpreturn() で
   エラーを示す値を返す ;
   トランザクション整合性レベルを、読み取り／書き込みに設定 ;
   最終的な引き出し金額の確認のため、ACCOUNT レコードを検索 ;
   tpalloc() を使用して引き出し要求バッファを割り当て、Finit() で初期化し
   再度初期化 ;
   WITHDRAWAL サービス要求への値を引き出しバッファに格納 ;
    サービスからの呼び出しにより DEPOSIT 要求の
    優先順位を上げる ;
   tpcall() で WITHDRAWAL サービスを呼び出し、口座残高を引き出す ;
   ACCOUNT レコードを削除 ;
   必要な情報を含む応答バッファを用意 ;
   tpfree() で引き出し要求バッファを解放 ;
    tpreturn() で、成功を示す値を返す ;
}
```

コード リスト 3-8　WITHDRAWAL **の擬似コード**

```
void WITHDRAWAL(TPSVCINFO *transb)
{
    Fvall() および Fget() で、フィールド化バッファから口座番号と金額を取得 ;
   口座番号が妥当であるか確認し、妥当でなければ tpreturn() でエラーを示す値を返す ;
   引き出し金額が正の値であるか確認し、正でなければ tpreturn() で
   エラーを示す値を返す ;
   トランザクション整合性レベルを、読み取り／書き込みに設定 ;
   口座残高を取得するため、ACCOUNT レコードを検索 ;
   引き出し金額が ACCOUNT で示される残高を超えないか確認 ;
   窓口残高と支店番号を取得するため、TELLER レコードを検索 ;
   引き出し金額が TELLER の残高を超えないか確認 ;
   支店残高を取得するため、BRANCH レコードを検索 ;
   引き出し金額が BRANCH の残高を超えないか確認 ;
   引き出し金額を差し引き、新たな口座残高を決定 ;
   新たな口座残高で ACCOUNT レコードを更新 ;
   引き出し金額を差し引き、新たな窓口残高を決定 ;
   新たな窓口残高で TELLER レコードを更新 ;
   引き出し金額を差し引き、新たな支店残高を決定 ;
```

```
    支店残高で BRANCH レコードを更新 ;
    トランザクション情報と共に、新たな HISTORY レコードを挿入 ;
    必要な情報を含む応答バッファを用意 ;
     tpreturn() で、成功を示す値を返す ;
}
```

コード リスト 3-9　DEPOSIT **の擬似コード**

```
void DEPOSIT(TPSVCINFO *transb)
{
     Fvall() および Fget() で、フィールド化バッファから口座番号と金額を取得 ;
     口座番号が妥当であるか確認し、妥当でなければ tpreturn() でエラーを示す値を返す ;
     預け入れ金額が正の値であるか確認し、正でなければ tpreturn() で
     エラーを示す値を返す ;
    トランザクション整合性レベルを、読み取り／書き込みに設定 ;
    口座残高を取得するため、ACCOUNT レコードを検索 ;
    窓口残高と支店番号を取得するため、TELLER レコードを検索 ;
    支店残高を取得するため、BRANCH レコードを検索 ;
    預け入れ金額を加算し、新たな口座残高を決定 ;
    新たな口座残高で ACCOUNT レコードを更新 ;
    預け入れ金額を加算し、新たな窓口残高を決定 ;
    新たな窓口残高で TELLER レコードを更新 ;
    預け入れ金額を加算し、新たな支店残高を決定 ;
    支店残高で BRANCH レコードを更新 ;
    トランザクション情報と共に、新たな HISTORY レコードを挿入 ;
    必要な情報を含む応答バッファを用意 ;
    tpreturn() で成功を示す値を返す ;
}
```

コード リスト 3-10　INQUIRY **の擬似コード**

```
void INQUIRY(TPSVCINFO *transb)
{
   Fvall() でフィールド化バッファから口座番号を取得 ;
    口座番号が妥当であるか確認し、妥当でなければ tpreturn() でエラーを示す値を返す ;
   トランザクション整合性レベルを、読み取りのみに設定 ;
   口座残高を取得するため、ACCOUNT レコードを検索 ;
   必要な情報を含む応答バッファを用意 ;
   tpreturn() で成功を示す値を返す ;
}
```

コード リスト 3-11　TRANSFER **の擬似コード**

```
void TRANSFER(TPSVCINFO *transb)
{
   Fvall() および Fget() で、フィールド化バッファから、口座番号と
   金額を取得 ;
   口座番号が両方とも妥当であるか確認し、妥当でない場合は、tpreturn() で
   エラーを示す値を返す ;
   振り替え金額が正の値であるか確認し、正でない場合は tpreturn() で
   エラーを示す値を返す ;
   tpalloc() を使用して引き出し要求バッファを割り当て、
    Finit() で初期化し
    FML タイプとして使用 ;
     WITHDRAWAL サービス要求への値を引き出し要求バッファに格納 ;
    サービスからの呼び出しにより DEPOSIT 要求の
    優先順位を上げる ;
   tpcall() で WITHDRAWAL サービスを呼び出す ;
   応答要求バッファから情報を取得 ;
   預け入れ要求バッファとして使用するため、引き出し要求バッファを Finit() により
   再度初期化 ;
   DEPOSIT サービス要求への値を預け入れ要求のバッファに格納 ;
   DEPOSIT 要求の優先順位を上げる ;
   tpcall() で DEPOSIT サービスを呼び出す ;
   必要な情報を含む応答バッファを用意 ;
   tpfree() で引き出し／預け入れ要求バッファの解放 ;
   tpreturn() で成功を示す値を返す ;
}
```

コード リスト 3-12　ABAL **の擬似コード**

```
void ABAL(TPSVCINFO *transb)
{
   トランザクション整合性レベルを、読み取りのみに設定 ;
   ABAL サーバ・グループのデータベースで、ACCOUNT ファイルの
   すべての BALANCE 値の合計を検索
   ( 単一の ESQL 文で十分 );
   合計を応答バッファ・データ構造体へ格納 ;
   tpreturn() で成功を示す値を返す ;
}
```

コード リスト 3-13　TBAL **の擬似コード**

```
void TBAL(TPSVCINFO *transb)
{
   トランザクション整合性レベルを、読み取りのみに設定 ;
   TBAL サーバ・グループのデータベースで、TELLER ファイルの
   すべての BALANCE 値の合計を検索
   ( 単一の ESQL 文で十分 );
   合計を応答バッファ・データ構造体へ格納 ;
   tpreturn() で成功を示す値を返す ;
}
```

コード リスト 3-14　ABAL_BID **の擬似コード**

```
void ABAL_BID(TPSVCINFO *transb)
{
   トランザクション整合性レベルを、読み取りのみに設定 ;
   transb バッファに基づき、branch_id を設定 ;
   BRANCH_ID = branch_id に該当するレコードの中で、ACCOUNT ファイルの
   すべての BALANCE 値の合計を検索
   ( 単一の ESQL 文で十分 );
   合計を応答バッファ・データ構造体へ格納 ;
   tpreturn() で成功を示す値を返す ;
}
```

コード リスト 3-15　TBAL_BID **の擬似コード**

```
void TBAL_BID(TPSVCINFO *transb)
{
   トランザクション整合性レベルを、読み取りのみに設定 ;
   transb バッファに基づき、branch_id を設定 ;
   BRANCH_ID = branch_ID に該当するレコードの中で、TELLER ファイルの
   すべての BALANCE 値の合計を検索
   ( 単一の ESQL 文で十分 );
   合計を応答バッファ・データ構造体へ格納 ;
   tpreturn() で成功を示す値を返す ;
}
```

サーバに組み込みのユーティリティ

bankapp のソース・ファイルには、appinit.c と util.c という 2 つの C 言語サブルーチン・ファイルがあります。

- appinit.c には、アプリケーション固有の tpsvrinit() と tpsvrdone() サブルーチンが記述されています。tpsvrinit() と tpsvrdone() は、BEA Tuxedo ATMI の標準 main() に含まれているサブルーチンです。tpsvrinit() は、デフォルトで 2 つの関数を呼び出します。リソース・マネージャを開く tpopen() と、サーバが起動したことを示すメッセージを記録する userlog() です。また、tpsvrdone() も、デフォルトで 2 つの関数を呼び出します。リソース・マネージャを閉じる tpclose() と、サーバがシャットダウンされることを示すメッセージを記録する userlog() です。これらのデフォルトのサブルーチンの代わりに、アプリケーション固有の tpsvrinit() と tpsvrdone() サブルーチンを使用できます。そのため、アプリケーション固有の初期化処理とシャットダウンの前処理を行うことができます。
- util.c には getstr() というサブルーチンが記述されています。これは bankapp で SQL のエラー・メッセージを処理する場合に使用されます。

サービスをコーディングする別の方法

bankapp のソース・ファイルでは、すべてのサービスがサーバのソース・コードとして参照されるファイルに組み込まれています。これらのファイルは bankapp サーバと名前が同じですが、main() 関数が含まれていないので実際にはサーバではありません。標準 main() は、buildserver コマンドの実行時に BEA Tuxedo ATMI によって提供されます。

BEA Tuxedo システム・アプリケーションを作成する別の方法として、各サービス・サブルーチンを別のファイルに記述する方法があります。この方法を使用して、TLR サーバを作成するとします。TLR.ec ファイルには 3 つのサービスが定義されています。次の手順に従って、これらのサービスを INQUIRY.ec、WITHDRAW.ec、および DEPOSIT.ec の 3 つの .ec ファイルに分けます。

1. 各 .ec ファイルを .o ファイルにコンパイルします。
2. 各 .o ファイルに -f オプションを指定して、buildserver コマンドを実行します。

```
buildserver -r TUXEDO/SQL \
            -s DEPOSIT -s WITHDRAWAL -s INQUIRY \
            -o TLR \
            -f DEPOSIT.o -f WITHDRAW.o -f INQUIRY.o \
            -f util.o -f -lm
```

注記　上記のコマンド行のバックスラッシュは、改行を明示的に示すために表記されているだけです。コマンドとオプションは 1 行に入力できます。

この例に示したように、すべてのサービス関数を 1 つのソース・ファイルに記述する必要はありません。つまり、サーバはソース・プログラム・ファイルとして存在する必要はありません。サーバは、各種のソース・ファイルを使用して、buildserver コマンド行に指定されたファイルに基づいて 1 つの実行可能サーバとしてビルドできます。この方法により、サーバを柔軟にビルドできるようになります。

関連項目

- 3-3 ページの「bankapp について」
- 『BEA Tuxedo システム入門』の 2-5 ページの「ATMI の使用」
- 『BEA Tuxedo コマンド・リファレンス』の `buildserver(1)`
- 『BEA Tuxedo コマンド・リファレンス』

bankapp のファイルおよびリソースの準備

ここでは、bankapp の実行に必要なファイルとリソースを作成するための手順を順に示します。

各作業をクリックすると、その作業を行う手順が表示されます。

ステップ 1: 環境変数の設定

環境変数は、bankvar ファイルに定義されています。このファイルは数多くのコメントが記述された大きな (約 185 行から構成される) ファイルです。

1. テキスト・エディタで、bankvar ファイルの内容を確認します。
   - コメントではない最初のコード行では、TUXDIR が設定されているかどうかが確認されます。設定されていない場合、ファイルの実行が失敗して、次のメッセージが表示されます。

     ```
     TUXDIR:parameter null or not set
     ```

2. TUXDIR パラメータに BEA Tuxedo システムのディレクトリ構造でのルート・ディレクトリを設定し、エクスポートします。
3. bankvar の別のコード行で、APPDIR に ${TUXDIR}/samples/atmi/bankapp が設定されています。これは、bankapp ソース・ファイルが置かれたディレクトリです。APPDIR は、BEA Tuxedo システムによって、アプリケーション固有のファイルが検索されるディレクトリです。オリジナルのソース・ファイルを上書きしないように、bankapp ファイルを別のディレクトリにコピーします。その場合は、そのディレクトリを APPDIR に指定します。TUXDIR の下位ディレクトリである必要はありません。
4. DIPCKEY に値を設定します。これは、BEA Tuxedo システム・データベースの IPCKEY です。DIPCKEY の値は、UBBCONFIG ファイルで指定された BEA Tuxedo システムの IPCKEY とは異なる値を指定する必要があります。

   注記　bankvar で指定されるほかの変数は、サンプル・アプリケーションで各種の働きをします。独自のアプリケーションを開発する場合は、それらの働きについて認識しておくことが必要です。bankvar にはすべての変数が定義されているので、後で実際のアプリケーションのテンプレートとして使用できます。

5. bankvar に必要な変更を加えたら、次のように bankvar を実行します。

   ```
   . ./bankvar
   ```

コード リスト 3-16　bankvar:bankapp **の環境変数**

```
# Copyright (c) 1997, 1996 BEA Systems, Inc.
# Copyright (c) 1995, 1994 Novell, Inc.
# Copyright (c) 1993, 1992, 1991, 1990 Unix System Laboratories, Inc.
# All rights reserved
#
# このファイルには、bankapp を実行するために
# Tuxedo システムで必要なすべての環境変数が設定されています。
#
# このディレクトリには、すべての Tuxedo ソフトウェアが置かれています。
# システム管理者はこの変数を設定する必要があります。
#
if [ -z "${TUXDIR}" ] ; then
 if [ ! -z "${ROOTDIR}" ] ; then
  TUXDIR=$ROOTDIR
  export TUXDIR
 fi
 fi
TUXDIR=${TUXDIR:?}
#
# 必要に応じて LANG を設定し直します。
#
if [ ! -d ${TUXDIR}/locale/C -a -d ${TUXDIR}/locale/english_us ] ; then
 export LANG
 LANG=english_us.ascii
fi
#
# このディレクトリには、ユーザ作成の全コードが置かれています。
#
# アプリケーション・ジェネレータによって生成されたファイルが
# 置かれるディレクトリの絶対パス名です。
#
APPDIR=${TUXDIR}/apps/bankapp
#
# SVR4 など特定の環境での実行時に
# 動的にリンクされる共用オブジェクトのパスです。
#
LD_LIBRARY_PATH=${TUXDIR}/lib:${LD_LIBRARY_PATH}
#
# HP-UX の場合に、共用オブジェクトのパスを設定します。
#
SHLIB_PATH=${TUXDIR}/lib:${SHLIB_PATH}
#
# AIX の場合に、共用オブジェクトのパスを設定します。
#
```

```
LIBPATH=${TUXDIR}/lib:/usr/lib:${LIBPATH}
#
# 論理ブロック・サイズです。データベース管理者はこの変数を設定する必要があります。
#
BLKSIZE=512
#
# データベース・ユーティリティおよびデータベース作成スクリプトで使用する
# データベースのデフォルト名を設定します。
#
DBNAME=bankdb
#
# データベースを共用モードで開くか、またはプライベート・モードで開くかを指定します。
#
DBPRIVATE=no
#
# データベースの IPC キーを設定します。これは、UBBCONFIG の IPC とは異なる値を指定する必要があります。
# *RESOURCES IPCKEY パラメータ
#
DIPCKEY=80953
#
# tmloadcf で使用される環境ファイルです。
#
ENVFILE=${APPDIR}/ENVFILE
#
# mc、viewc、tmloadcf などで使用されるフィールド・テーブル・ファイルのリストです。
#
FIELDTBLS=Usysflds,bankflds,creditflds,eventflds
#
FIELDTBLS32=Usysfl32,evt_mib,tpadm
#
# フィールド・テーブル・ファイルを検索するディレクトリのリストです。
#
FLDTBLDIR=${TUXDIR}/udataobj:${APPDIR}
#
FLDTBLDIR32=${TUXDIR}/udataobj:${APPDIR}
#
# データベースの汎用デバイス・リストです。
#
FSCONFIG=${APPDIR}/bankdl1
#
# MENU スクリプトで使用されるネットワーク・アドレスです。
#
NADDR=
#
# ネットワーク・デバイス名です。
#
```

```
NDEVICE=
#
# MENU スクリプトで使用されるネットワーク・リスナ・アドレスです。
#
NLSADDR=


#
# TERM が設定されていることを確認します。
#
TERM=${TERM:?}
#
# トランザクション・ログのデバイスを設定します。これは、
# UBBCBSHM ファイルの *MACHINES セクションにあるこのサイトの LMID の
TLOGDEVICE パラメータと一致する必要があります。
#
TLOGDEVICE=${APPDIR}/TLOG
#
# BEA Tuxedo システムにすべての情報を提供するバイナリ・ファイルのデバイスです。
#
TUXCONFIG=${APPDIR}/tuxconfig
#
# 中央ユーザ・ログを記録するファイルの接頭語を設定します。
# これは、UBBCONFIG ファイルの *MACHINES セクションにある
# このサイトの LMID の ULOGPFX パラメータと一致する必要があります。
#
ULOGPFX=${APPDIR}/ULOG
#
# RUNME.sh で使用されるシステム名です。
#
UNAME=
#
# viewc、tmloadcf などで使用される VIEW ファイルのリストです。
#
VIEWFILES=aud.V
#
VIEWFILES32=mib_views,tmib_views
#
# VIEW ファイルを検索するディレクトリのリストです。
#
VIEWDIR=${TUXDIR}/udataobj:${APPDIR}
#
VIEWDIR32=${TUXDIR}/udataobj:${APPDIR}
#
# Q デバイスを指定します ( デモにイベントが含まれている場合 )。
#
QMCONFIG=${APPDIR}/qdevice
```

```
#
# 設定したすべての変数をエクスポートします。
#
export TUXDIR APPDIR BLKSIZE DBNAME DBPRIVATE DIPCKEY ENVFILE
export LD_LIBRARY_PATH SHLIB_PATH LIBPATH
export FIELDTBLS FLDTBLDIR FSCONFIG MASKPATH OKXACTS TERM
export FIELDTBLS32 FLDTBLDIR32
export TLOGDEVICE TUXCONFIG ULOGPFX
export VIEWDIR VIEWFILES


export VIEWDIR32 VIEWFILES32
export QMCONFIG
#
# 設定されていない場合は、TUXDIR/bin を PATH に追加します。
#
a="`echo $PATH | grep ${TUXDIR}/bin`"
if [ x"$a" = x ]
then
 PATH=${TUXDIR}/bin:${PATH}
 export PATH
fi
#
# 設定されていない場合は、APPDIR を PATH に追加します。
#
a="`echo $PATH | grep ${APPDIR}`"
if [ x"$a" = x ]
then
 PATH=${PATH}:${APPDIR}
 export PATH
fi

#
# ほかのマシン・タイプの bin ディレクトリを確認します。
#
for DIR in /usr/5bin /usr/ccs/bin /opt/SUNWspro/bin
do
if [ -d ${DIR} ] ; then
 PATH="${DIR}:${PATH}"
fi
done
```

注記　Sun Solaris を使用している場合は、シェルとして csh ではなく、/bin/sh を使用します。また、次のように、PATH の先頭に /usr/5bin を指定します。

```
PATH=/usr/5bin:$PATH;export PATH
```

関連項目

- 3-31 ページの「bankapp のファイルおよびリソースの準備」

ステップ 2: bankapp でのサーバのビルド

buildserver(1) コマンドを使用して、BEA Tuxedo システムの ATMI main() 関数で実行可能 ATMI サーバを作成します。このコマンドではオプションを使用して、出力ファイル、アプリケーションで提供される入力ファイル、各種の方法で BEA Tuxedo システム・アプリケーションを実行するためのライブラリを指定します。

buildserver は、cc コマンドを呼び出します。環境変数の CC を設定すると別のコンパイル・コマンドを指定でき、CFLAGS を設定するとコンパイル時と編集時にフラグを設定できます。buildserver コマンドは bankapp.mk で使用され、銀行業務アプリケーションの各サーバをコンパイルしてビルドします。以下に、bankapp の 6 種類のサーバについて説明します。

- ACCT サーバのビルド
- BAL サーバのビルド
- BTADD サーバのビルド
- TLR サーバのビルド
- XFER サーバのビルド
- ステップ 3: bankapp Makefile の編集

ACCT サーバのビルド

ACCT サーバは、OPEN_ACCT および CLOSE_ACCT 関数のコードが記述された ACCT.ec ファイルから生成されます。このサーバをビルドするには、ACCT.ec をコンパイルして ACCT.o ファイルを生成し、それを buildserver コマンドに渡します。この方法では、コンパイル・エラーを特定でき、サーバをビルドする前に修正することができます。

1. ACCT.o ファイルを作成します (bankapp.mk で作成されます)。
   - esql ACCT.ec と指定して、.c ファイルを生成します。
   - cc -I $TUXDIR/include -c ACCT.c と指定して、.o ファイルを生成します。
   - 次の buildserver コマンドを指定して、ACCT サーバを作成します。

```
buildserver -r TUXEDO/SQL \
            -s OPEN_ACCT -s CLOSE_ACCT \
            -o ACCT \
            -f ACCT.o -f appinit.o -f util.o
```

   注記　上記のコマンド行のバックスラッシュは、改行を明示的に示すために表記されているだけです。コマンドとオプションは 1 行に入力できます。

以下は、buildserver コマンド行で指定されている各オプションの説明です。

- -r オプションは、実行可能サーバにリンクされるリソース・マネージャのアクセス・ライブラリを指定します。指定する値は、文字列 TUXEDO/SQL で開始します。
- -s オプションは、サーバの起動時に宣言されるサーバのサービス名を指定します。サービスを実行する関数名がサービス名と異なる場合、関数名が -s オプションの引数の一部になります。
  bankapp では、関数名はサービス名と同じなので、サービス名だけを指定します。サービス名は、すべての文字列を大文字で指定します。たとえば、OPEN_ACCT サービスは、OPEN_ACCT() 関数で処理されます。ただし、buildserver の -s オプションでは、サーバ内のサービスを処理する関数には任意の名前を付けることができます。詳細については、buildserver(1) のリファレンス・ページを参照してください。システム

管理者は、buildserver コマンドでサーバを作成した際に使用されたサービスのサブセットだけをサーバの起動時に利用できるように設定することもできます。

- -o オプションは、実行可能出力ファイルに名前を指定する場合に使用します。名前が指定されていない場合は、SERVER という名前が付きます。
- -f オプションは、リンク時と編集時に使用されるファイルを指定します。関連情報については、buildserver(1) リファレンス・ページの -l を参照してください。ファイルがリストされる順序は、関数の参照、およびその参照がどのライブラリで解決されるかによって決定されます。ソース・モジュールは、関数の参照が解決されるライブラリの前にリストされます。.c ファイルが存在する場合、それが最初にコンパイルされます。上記の例では、appinit.o と util.o は既にコンパイルされています。オブジェクト・ファイルは、別個の .o ファイル、またはアーカイブ (.a) ファイルにあるファイル・グループです。-f の引数として 1 つ以上のファイル名を指定する場合は、二重引用符で各ファイル名を囲む必要があります。-f の引数としては、1 つのファイルまたは二重引用符で囲まれた 1 つのファイル・リストしか指定できませんが、-f オプションは 1 つのコマンド行で必要に応じて何度でも指定できます。

以下に、ACCT サーバを作成するために、buildserver コマンド行に指定されたオプションで行われる操作を簡単にまとめます。

- -r オプションは、BEA Tuxedo システム SQL リソース・マネージャを指定します。
- -s オプションは、ACCT サーバを構成するサービスとして、OPEN_ACCT サービスと CLOSE_ACCT サービス (ACCT.ec ファイルにある同名の関数で定義されるサービス) を指定します。
- -o オプションは、実行可能出力ファイルに ACCT という名前を指定します。
- -f オプションは、ACCT.o、appinit.o、および util.o ファイルがビルドでのリンク時と編集時に使用されることを指定します。

  注記　appinit.c ファイルには、システムによって提供される tpsvrinit() および tpsvrdone() 関数が含まれています。これら

のルーチンの使用方法については、tpservice(3c) のリファレンス・ページを参照してください。

BAL サーバのビルド

BAL サーバは、ABAL、TBAL、ABAL_BID、および TBAL_BID 関数のコードが記述された BAL.ec ファイルから生成されます。ACCT.ec ファイルと同様に、BAL.ec をコンパイルして BAL.o ファイルを生成し、それを buildserver コマンドに渡します。そのため、コンパイル・エラーを特定でき、サーバをビルドする前に修正することができます。

1. 次のように、buildserver コマンドで BAL サーバをビルドします。

```
buildserver -r TUXEDO/SQL \
            -s ABAL -s TBAL -s ABAL_BID -s TBAL_BID\
            -o BAL \
            -f BAL.o -f appinit.o
```

注記　上記のコマンド行のバックスラッシュは、改行を明示的に示すために表記されているだけです。コマンドとオプションは 1 行に入力できます。

- -r オプションは、BEA Tuxedo システム SQL リソース・マネージャを指定します。
- -s オプションは、BAL サーバを構成するサービスとして、ABAL、TBAL、ABAL_BID、TBAL_BID を指定します。これらのサービス名は、そのサービスを定義する BAL.ec ファイルの関数名と同じです。
- -o オプションは、実行可能サーバに BAL という名前を指定します。
- -f オプションは、BAL.o と appinit.o ファイルがリンク時と編集時に使用されることを指定します。

BTADD サーバのビルド

BTADD サーバは、BR_ADD および TLR_ADD 関数のコードが記述された BTADD.ec ファイルから生成されます。BTADD.ec をコンパイルして BTADD.o ファイルを生成し、それを buildserver コマンドに渡します。

1. 次のように、buildserver コマンドで BTADD サーバをビルドします。

```
buildserver -r TUXEDO/SQL \
            -s BR_ADD -s TLR_ADD \
            -o BTADD \
            -f BTADD.o -f appinit.o
```

注記　上記のコマンド行のバックスラッシュは、改行を明示的に示すために表記されているだけです。コマンドとオプションは 1 行に入力できます。

- -r オプションは、BEA Tuxedo システム SQL リソース・マネージャを指定します。
- -s オプションは、BTADD を構成するサービスとして、BR_ADD と TLR_ADD を指定します。これらのサービス名は、そのサービスを定義する BTADD.ec ファイルの関数名と同じです。
- -o オプションは、実行可能サーバに BTADD という名前を指定します。
- -f オプションは、BTADD.o と appinit.o ファイルがリンク時と編集時に使用されることを指定します。

TLR サーバのビルド

TLR サーバは、DEPOSIT、WITHDRAWAL、および INQUIRY 関数のコードが記述された TLR.ec ファイルから生成されます。TLR.ec をコンパイルして TLR.o ファイルを生成し、それを buildserver コマンドに渡します。

1. 次のように、buildserver コマンドで TLR サーバをビルドします。

```
buildserver -r TUXEDO/SQL \
            -s DEPOSIT -s WITHDRAWAL -s INQUIRY \
            -o TLR \
            -f TLR.o -f util.o -f -lm
```

注記　上記のコマンド行のバックスラッシュは、改行を明示的に示すために表記されているだけです。コマンドとオプションは 1 行に入力できます。

- -r オプションは、BEA Tuxedo システム SQL リソース・マネージャを指定します。
- -s オプションは、TLR サーバを構成するサービスとして、DEPOSIT、WITHDRAWAL、および INQUIRY を指定します。これらのサービス名は、そのサービスを定義する TLR.ec ファイルの関数名と同じです。
- -o オプションは、実行可能サーバに TLR という名前を指定します。
- -f オプションは、TLR.o と util.o ファイルがリンク時と編集時に使用されることを指定します。

注記　上記のコード例では、-f を使用して、オプション (-lm) を cc コマンドに渡しています。このコマンドは、buildserver によって呼び出されます。-f に -lm の引数を指定すると、コンパイル時に math ライブラリがリンクされます。

コンパイル時に使用できるオプションについては、『UNIX System V User's Reference Manual』の cc(1) を参照してください。

XFER サーバのビルド

XFER サーバは、TRANSFER 関数のコードが記述された XFER.c ファイルから生成されます。XFER.c をコンパイルして XFER.o ファイルを生成し、それを buildserver コマンドに渡します。

1. 次のように、buildserver コマンドで XFER サーバをビルドします。

```
buildserver -r TUXEDO/SQL \
            -s TRANSFER \
```

```
-o XFER \
-f XFER.o -f appinit.o
```

注記　上記のコマンド行のバックスラッシュは、改行を明示的に示すために表記されているだけです。コマンドとオプションは 1 行に入力できます。

- -r オプションは、BEA Tuxedo システム SQL リソース・マネージャを指定します。
- -s オプションを使用して、XFER サーバを構成するサービスとして、BR_ADD と TLR_ADD を指定します。これらのサービス名は、そのサービスを定義する TRANSFER ファイルの関数名と同じです。
- -o オプションを使用して、実行可能サーバに XFER という名前を指定します。
- -f オプションを使用して、XFER.o と appinit.o ファイルがリンク時と編集時に使用されることを指定します。

bankapp.mk ファイルでビルドされるサーバ

bankapp サーバをビルドする場合、buildserver コマンドの指定方法を理解していることが大切です。ただし、実際にビルドする場合、makefile にビルドの定義を記述することがよくあります。bankapp でもその方法が採用されています。

ステップ 3: bankapp Makefile の編集

bankapp には、すべてのスクリプトを実行可能にし、VIEW 記述ファイルをバイナリ形式に変換し、アプリケーション・サーバの作成に必要なすべてのプリコンパイル、コンパイル、およびビルドを行う makefile が提供されています。また、最初からやり直す場合にもこのファイルを利用できます。

提供されている bankapp.mk をそのまま使わずにフィールドを編集した方がよい場合があります。また、説明が必要なフィールドもあります。以下にそれらのフィールドについて説明します。

TUXDIR パラメータの編集

1. bankapp.mk ファイルの 40 行目前後に、次のコメントと TUXDIR パラメータが記述されています。

```
    #
    # Tuxedo システムのルート・ディレクトリです。このファイルは、編集してこの値を正しく設定するか、
# または "make -f STOCKAPP.mk TUXDIR=/correct/rootdir" を使用して正しい値を渡します。
# そのようにしないと、bankapp のビルドは失敗します。
#
    TUXDIR=../..
```

2. TUXDIR パラメータに、BEA Tuxedo システムがインストールされたルート・ディレクトリの絶対パス名を指定します。

APPDIR パラメータの編集

1. APPDIR パラメータに設定された値を確認します。bankapp の場合、APPDIR には bankapp ファイルが置かれたディレクトリ (TUXDIR の相対パス) が指定されています。次に示す bankapp.mk のセクションには、APPDIR の設定についての説明と定義が記述されています。

```
    #
    # bankapp アプリケーションのソース・コードと実行可能ファイルが置かれたディレクトリです。
    # このファイルは、編集してこの値を正しく設定するか、
    # または "make -f bankapp.mk APPDIR=/correct/appdir" を使用して正しい値を渡します。
    # そのようにしないと、bankapp のビルドは失敗します。
    #
    APPDIR=$(TUXDIR)/samples/atmi/bankapp
    #
```

2. README ファイルに従って、別のディレクトリにファイルをコピーした場合、APPDIR にはファイルのコピー先のディレクトリを指定します。makefile を実行すると、そのディレクトリにアプリケーションがビルドされます。

リソース・マネージャのパラメータの設定

デフォルトでは、bankapp はデータベース・リソース・マネージャとして BEA Tuxedo/SQL を使用するように設定されています。その場合、ご使用のシステムに BEA Tuxedo システム・データベースがインストールされていることが必要です。インストールされていない場合は、RM パラメータに TUXDIR/udataobj/RM にリストされているリソース・マネージャの名前を設定します。

```
#
# リソース・マネージャ
#
RM=TUXEDO/SQL
#
```

注記　BEA Tuxedo SQL リソース・マネージャは、デモ用のプログラムです。

bankapp.mk ファイルの実行

1. bankapp.mk に必要な変更を加えたら、次のコマンドを入力して実行します。

```
nohup make -f bankapp.mk &
```

2. nohup.out ファイルを調べて、処理が正しく行われたことを確認します。

注記　bankvar には、bankapp.mk の実行時に参照されるパラメータが設定されています。

関連項目

- 3-31 ページの「bankapp のファイルおよびリソースの準備」

ステップ 4: bankapp データベースの作成

ここでは、bankapp とリソース・マネージャ (通常はデータベース管理システム) 間のインターフェイス、および bankapp データベースを作成する方法について説明します。bankapp では、BEA Tuxedo システム・データベース、つまり XA 準拠のリソース・マネージャの BEA Tuxedo/SQL の機能が使用されます。

注記　BEA Tuxedo SQL リソース・マネージャは、デモ用のプログラムです。

bankapp データベースを作成する方法は、単一プロセッサでアプリケーションを起動するか (SHM モード)、複数のプロセッサをネットワーク上で起動するか (MP モード) によって異なります。

SHM モードでのデータベースの作成

1. 次のコマンドを入力して、環境変数を設定します。

   ```
   . ./bankvar
   ```

2. crbank を実行します。crbank は crbankdb を 3 回呼び出し、そのたびに環境変数が変更されます。そのため、1 つのマシンで 3 つの異なるデータベース・ファイルが作成されます。つまり、ネットワーク接続されていない場合でも、複数のマシンから構成される BEA Tuxedo システム環境をシミュレートできます。

MP モードでのデータベースの作成

1. 次のコマンドを入力して、環境変数を設定します。

   ```
   . ./bankvar
   ```

   注記　既に環境変数が設定されている場合もあります。詳細については、「環境変数の設定」を参照してください。

2. crbankdb を実行して、このサイトのデータベースを作成します。

3. BEA Tuxedo システムのネットワークに追加した各マシンで、bankvar を編集して FSCONFIG 変数のパス名を指定します。このパス名は、そのサイトで使用され、コンフィギュレーション・ファイル ubbmp で参照される変数です。手順 1 と 2 を繰り返します。

関連項目

- 3-31 ページの「bankapp のファイルおよびリソースの準備」

ステップ 5: XA 準拠のリソース・マネージャの準備

XA 準拠の代替リソース・マネージャで bankapp を実行するには、各種のファイルを変更する必要があります。ここでは、次の内容について説明します。

- bankvar ファイルの変更
- bankapp サービスの変更
- bankapp.mk ファイルの変更
- crbank および crbankdb の変更
- コンフィギュレーション・ファイルの変更

bankvar ファイルの変更

1. BEA Tuxedo システム・データベースを作成する環境変数を確認します。これらの環境変数には、次のデフォルト値が設定されています。

```
BLKSIZE=512
DBNAME=bankdb
DBPRIVATE=no
DIPCKEY=80953
FSCONFIG=${APPDIR}/bankdl1
```

   注記　これらの環境変数は、BEA Tuxedo システムだけを対象としています。ほかのデータベース管理システムを使用する場合は、必要に応じてほかの環境変数を設定してください。

2. 必要に応じてこれらの変数を変更して、リソース・マネージャのデータベースを作成します。

bankapp サービスの変更

bankapp では、データベースへのすべてのアクセスは埋め込み型 SQL 文で実行されます。そのため、新しいリソース・マネージャで SQL がサポートされている場合は問題がありません。appinit.c ユーティリティには、tpopen() と tpclose() への呼び出しが定義されています。

bankapp.mk ファイルの変更

1. bankapp.mk の RM パラメータを編集し、新しいリソース・マネージャの名前を指定します。
2. RM ファイルに次のエントリがあることを確認します。

   ```
   $TUXDIR/udataobj/RM
   ```

3. 必要に応じて、SQL コンパイラ名とそのオプションを変更します。ソース・ファイル名に拡張子 .ec が使用されていない場合があります。生成される .c ファイルをコンパイルするには、デフォルトではない名前を設定することが必要な場合もあります。

crbank および crbankdb の変更

1. crbank が代替リソース・マネージャで処理されないことがあります。crbank は、変数をリセットして、crbankdb を 3 回実行するだけです。
2. crbankdb を使用する場合は注意します。次のコード例は、crbankdb スクリプトの最初の部分です。このコード例の後に、BEA Tuxedo システム以外のリソース・マネージャでは機能しないコード部分について説明してあります。

コード リスト 3-17 crbankdb **スクリプト**

```
    #Copyright (c) BEA Systems, Inc.
    #All rights reserved
    #
    # デバイス・リストを作成します。
    #
    dbadmin<<!
    echo
    crdl
    # 次の行をデバイス 0 のエントリに置き換えます。
    ${FSCONFIG}        0         2560
    !
#
    # データベース・ファイル、フィールド、および二次インデックスを作成します。
    #
    sql<<!
    echo
    create database ${DBNAME} with (DEVNAME='${FSCONFIG}',
          IPCKEY=${DIPCKEY}, LOGBLOCKING=0, MAXDEV=1,
          NBLKTBL=200,       NBLOCKS=2048,  NBUF=70,       NFIELDS=80,
          NFILES=20,         NFLDNAMES=60,  NFREEPART=40,  NLCKTBL=200,
          NLINKS=80,         NPREDS=10,     NPROCTBL=20,   NSKEYS=20,
          NSWAP=50,          NTABLES=20,    NTRANTBL=20,   PERM='0666',
          STATISTICS='n'
    )


    create table BRANCH (
          BRANCH_ID             integer not null,
          BALANCE               real,
          LAST_ACCT             integer,
          LAST_TELLER           integer,
          PHONE                 char(14),
          ADDRESS               char(60),
          primary key(BRANCH_ID)
    ) with (
          FILETYPE='hash',      ICF='PI',     FIELDED='FML',
          BLOCKLEN=${BLKSIZE},  DBLKS=8,      OVBLKS=2
    )
    !
```

最初の約 40 行を参照すると、変更が必要な部分と不要な部分を判断できます。コードからわかるように、crbankdb は dbadmin への入力と sql シェルコマンドへの入力から構成されています。最初の here ファイルは、BEA Tuxedo システムのコマンド dbadmin に渡されて、データベースのデバイス・リストが作成されます。

このコマンドは、BEA Tuxedo リソース・マネージャだけで機能します。テーブル・スペースを作成したり、適切な権限を認めるために、ほかのコマンドが必要になる場合があります。

コンフィギュレーション・ファイルの変更

GROUPS セクションで、TMSNAME および OPENINFO パラメータに適切な値 (つまり、リソース・マネージャで認識される値) を指定します。

Windows 2000 プラットフォーム上での bankapp と Oracle 8 (XA 準拠の RM) の統合

1. nt\bankvar.cmd を編集し、次の環境変数に適切な値を指定します。

   TUXDIR :BEA TUXEDO システムのインストール先のルート・ディレクトリ

   APPDIR :bankapp ファイルが置かれたアプリケーション・ディレクトリ

   ORACLE_HOME :Oracle8 のインストール先のルート・ディレクトリ

   ORACLE_SID :Oracle システム ID

   BLK_SIZE: 論理ブロック・サイズ

   DBNAME: データベース・ユーティリティおよびデータベース作成スクリプトによって使用されるデータベースのデフォルト名

DBPRIVATE: データベースを共用モードで開くか、またはプライベート・モードで開くかを指定 (yes または no)

FSCONFIG: データベースの汎用デバイス・リスト

```
PATH=%TUXDIR%\bin;%TUXDIR%\include;%TUXDIR%\lib;%ORACLE_HOME
%\bin;%PATH%

INCLUDE=%ORACLE_HOME%\rdbms80\xa;
%ORACLE_HOME%\pro80\c\include;%include%

NLSPATH=%TUXDIR%\locale\C

PLATFORM=inwnt40

LIB=%TUXDIR%\lib; %ORACLE_HOME%\pro80\lib\msvc;
%ORACLE_HOME%\rdbms80\xa; %lib%;
```

2. スクリプトを実行して、環境変数を設定します。

   ```
   >bankvar
   ```

3. 次のように、TUXDIR\udataobj\RM ファイルを編集します。

   - $TUXDIR\udataobj\RM ファイルに次の行を追加します。

     ```
     Oracle_XA;xaosw;%ORACLE_HOME%\pro80\lib\msvc\sqllib80.lib
     %ORACLE_HOME%\RDBMS80\XA\xa80.lib
     ```

   Oracle がネットワーク上にある場合は、次のように編集します。

   - マシンをドライブ (たとえば F ドライブ) にマップします。
   - $TUXDIR\udataobj\RM ファイルに次の行を追加します。

     ```
     Oracle_XA;xaosw;f:\orant\pro80\lib\msvc\sqllib80.lib
     f:\orant\RDBMS80\XA\xa80.lib
     ```

   - RM ファイルで、Oracle_XA の以前のエントリを削除します。

4. Oracle8 のトランザクション・マネージャ・サーバをビルドします。

   ```
   cd $APPDIR
   buildtms -r Oracle_XA -o TMS_ORA
   ```

5. 次の表に従って、nt\bankapp.mak ファイルを編集します。

作業	値
次の環境変数を指定します。	TUXDIR=BEA Tuxedo システムのインストール先のルート・ディレクトリ
	APPDIR=bankapp ファイルが置かれたアプリケーション・ディレクトリ
	RM=Oracle_XA
	ORACLE_LIBS=$(ORACLE_HOME) \PRO80\LIB
	RMNAME=Oracle_XA
	SQLPUBLIC=$(ORACLE_HOME)\PRO80\C\INCLUDE
	CFLAGS=$(HOST) -DNOWHAT=1 $(CGFLAGS) $(DFML32)
	CGFLAGS=-DWIN32 -W3 -MD -nologo

作業	値
	ORACLE_DIR=$(ORACLE_HOME)\bin
	INCDIR=$(TUXDIR)\include
	CC=cl
.ec.c セクションで、埋め込み型の SQL プログラムから C プログラムを作成する場合の規則を編集し、proc コンパイラを使用して次の値を設定します。	set TUXDIR=$(TUXDIR) & $(ORACLE_DIR)\proc80 mode=ansi release_cursor=yes include=$(SQLPUBLIC) include=$(INCDIR) $(SQL_PLATFORM_INC) -c iname=$*.ec
.c.obj セクションで、C プログラムからオブジェクト・ファイルを作成する場合の規則を編集し、次の値を設定します。	$(CC) -c $(CFLAGS) $(SQLPUBLIC) $(INCLUDE) $*.c

6. *.ec ファイルを更新します。Oracle SQL コマンドを使用します。
7. makefile を実行します。

```
copy nt\bankapp.mak to %APPDIR%
nmake -f bankapp.mak
```

8. 次のように、nt\ubbshm を編集します。

```
USER_ID=0
GROUP_ID=0
UNAME_SITE1= ホスト名で返されるノード名
TUXDIR=bankvar で指定された値
APPDIR=bankvar で指定された値
```

9. 次のように、コンフィギュレーション・ファイルの GROUPS セクションを変更します。

```
TMSNAME=TMS_ORA
BANKB1 GRPNO=1
OPENINFO="Oracle_XA:Oracle_XA+Acc=P/user1/PaSsWd1+SesTm=0+LogDi
r=."
[
Oracle_XA +
 required fields:
 Acc=P/oracle_user_id/oracle_password +
```

```
 SesTm=Session_time_limit (maximum time a transaction can be
inactive) +
 optional fields:
 LogDir=logdir (where XA library trace file is located) +
 MaxCur=maximum_#_of_open cursors +
 SqlNet=connect_string (if Oracle exists over the network)
 (eg. SqlNet=hqfin@NEWDB indicates the database with sid=NEWDB
accessed at host hqfin by TCP/IP)
]
BANKB2 GRPNO=2

OPENINFO="Oracle_XA:Oracle_XA+Acc=P/user2/PaSsWd2+SesTm=0+LogDi
r=."
BANKB3 GRPNO=3
OPENINFO="Oracle_XA:Oracle_XA+Acc=P/user3/PaSsWd3+SesTm=0+LogDi
r=."
```

10. バイナリ形式の BEA Tuxedo コンフィギュレーション・ファイルを作成します。

    ```
    tmloadcf -y nt/ubbshm
    ```

11. マスタ・マシン上にデバイス・リストと TLOG デバイスを作成します。

    ```
    crtlog -m
    ```

12. Oracle データベース・インスタンスが起動されていない場合は起動します。

13. BEA Tuxedo システム・サーバを起動します。

    ```
    tmboot -y
    ```

14. v$XATRANS$ ビューがデータベースにあることを確認します。V$XATRANS$ ビューは、XA ライブラリのインストール時に作成されます。

15. v$XATRANS$ ビューが作成されていない場合は、次のように作成します。

    - 環境変数 ORACLE_HOME および ORACLE_SID が設定されていることを確認します。
    - SYS としてデータベースにログインします。

      ```
      Execute the sql script
      ${ORACLE_HOME}/RDBMS80/ADMIN/XAVIEW.sql
      ```

    - XA ライブラリを使用するすべての Oracle アカウント・アプリケーションのビューに対する SELECT 権限を認めます。

16. bankapp データベースと Oracle RM のデータベース・オブジェクトを作成します。

    - Oracle ユーティリティ SQL*plus または SQL*DBA に、任意の Oracle ユーザとしてログインします。
    - notepad crbank-ora8.sql
    - Oracle 8 のインストール時にサンプル・データベースが作成されます。bankapp アプリケーションでこのデータベースを使用できます。sql スクリプトによってデータベース内に新しいテーブル・スペースが作成され、bankapp のすべてのデータベース・オブジェクトを格納できます。新しいテーブル・スペースを使用するには、Oracle システム・ユーザ・パスワードとファイルの絶対パス名を入力する必要があります。
    - 次のように、crbank-ora8.sql を編集します。

```
WHENEVER OSERROR EXIT ;
/* システムのユーザのパスワードを取得します。*/
PROMPT
PROMPT
PROMPT -- Some of the operations require "system" user privileges
PROMPT -- Please specify the Oracle "system" user password
PROMPT
ACCEPT syspw CHAR PROMPT 'system passwd:' HIDE ;
CONNECT system/&syspw ;
SHOW user ;
PROMPT
/* デフォルトのデータベースに、「bankapp」で使用する新しいテーブル・スペースを作成します。*/
DROP TABLESPACE bank1
    INCLUDING CONTENTS
CASCADE CONSTRAINTS;
PROMPT
PROMPT
PROMPT -- Will create a 3MB tablespace for bankapp ;
PROMPT -------- Please specify full pathname below for Datafile ;
PROMPT -------- Ex: %ORACLE_HOME%/dbs/bankapp.dbf
PROMPT
ACCEPT datafile CHAR PROMPT 'Datafile:' ;

CREATE TABLESPACE bank1
    DATAFILE '&datafile'  SIZE 3M REUSE
    DEFAULT STORAGE (INITIAL 10K   NEXT 50K
            MINEXTENTS 1           MAXEXTENTS 120
            PCTINCREASE 5)
    ONLINE;
```

```
/***************** 「user1」というユーザを作成します。***************/

DROP USER user1 CASCADE;

PROMPT Creating user "user1"

CREATE USER user1 IDENTIFIED by PaSsWd1
    DEFAULT TABLESPACE bank1
    QUOTA UNLIMITED ON bank1 ;

GRANT CREATE SESSION TO user1 ;
GRANT CREATE TABLE TO user1 ;

CONNECT user1/PaSsWd1 ;
SHOW user ;

PROMPT Creating database objects for user "user1" ;
PROMPT Creating table "branch" ;

    CREATE TABLE branch (
          branch_id       NUMBER NOT NULL PRIMARY KEY,
          balance         NUMBER,
          last_acct       NUMBER,
          last_teller     NUMBER,
          phoneCHAR(14),
          address         CHAR(60)

 )
          STORAGE         (INITIAL 5K   NEXT 2K
                          MINEXTENTS 1 MAXEXTENTS 5  PCTINCREASE 5) ;

PROMPT Creating table "account" ;

    CREATE TABLE account (
          account_id      NUMBER NOT NULL PRIMARY KEY,
          branch_id       NUMBER NOT NULL,
          ssn             CHAR(12) NOT NULL,
          balance         NUMBER,
          acct_type       CHAR,
          last_name       CHAR(20),
          first_name      CHAR(20),
          mid_init        CHAR,
          phoneCHAR(14),
          address         CHAR(60)

 )
          STORAGE         (INITIAL 50K NEXT 25K
                          MINEXTENTS 1 MAXEXTENTS 50   PCTINCREASE 5) ;

PROMPT Creating table "teller" ;
     CREATE TABLE teller (
```

```
          teller_id       NUMBER NOT NULL PRIMARY KEY,
          branch_id       NUMBER NOT NULL,
          balance         NUMBER,
          last_name       CHAR(20),
          first_name      CHAR(20),
          mid_init        CHAR
 )
          STORAGE         (INITIAL 5K  NEXT 2K
                          MINEXTENTS 1  MAXEXTENTS 5    PCTINCREASE 5) ;
PROMPT Creating table "history" ;
    CREATE TABLE history (
          account_id      NUMBER NOT NULL,
          teller_id       NUMBER NOT NULL,
          branch_id       NUMBER NOT NULL,
          amount          NUMBER

 )
          STORAGE         (INITIAL 400K   NEXT 200K
                          MINEXTENTS 1   MAXEXTENTS 5     PCTINCREASE 5) ;
```

17. 前述の手順に従って、パスワードが PaSsWd2 である user2、およびパスワードが PaSsWd3 である user3 を作成します。

    ```
    SQL*plus> start $APPDIR/ crbank-ora8.sql
    ```

18. データベースにデータを追加します。

    ```
    nt\populate
    ```

19. データベースに対してトランザクションを生成します。

    ```
    driver
    ```

20. bankapp クライアントを実行します。

    ```
    run
    ```

21. アプリケーションをシャットダウンします。

    ```
    tmshutdown -y
    ```

関連項目

- 3-31 ページの「bankapp のファイルおよびリソースの準備」

ステップ 6: コンフィギュレーション・ファイルの編集

コンフィギュレーション・ファイルには、アプリケーションの実行方法が定義されています。bankapp には、テキスト形式のコンフィギュレーション・ファイル (UBBCONFIG(5) を参照) が 2 つ提供されています。単一のコンピュータ上のアプリケーションを定義する ubbshm と、ネットワーク上のアプリケーションを定義する ubbmp です。

初期化スクリプトは、サンプル・アプリケーションに提供されています。また、ご使用のコンフィギュレーションおよびマシンに合わせて、.sh を使用して 10 個までの完全なコンフィギュレーション・ファイルを生成できます。

1. テキスト・エディタで、bankapp の ubbshm および ubbmp コンフィギュレーション・ファイルの内容を確認します。

コード リスト 3-18 ubbmp **コンフィギュレーション・ファイル**

```
     #Copyright (c) 1999 BEA Systems, Inc.
     #All rights reserved

     *RESOURCES
     IPCKEY                80952
001  UID                   <id(1) からのユーザ ID>
002  GID                   <id(1) からのグループ ID>
     PERM                  0660
     MAXACCESSERS          40
     MAXSERVERS            35
     MAXSERVICES           75
     MAXCONV               10
     MAXGTT                20
     MASTER                SITE1,SITE2
     SCANUNIT              10
     SANITYSCAN            12
     BBLQUERY              180
     BLOCKTIME             30
     DBBLWAIT          6
```

```
     OPTIONS           LAN,MIGRATE
     MODEL             MP
     LDBAL             Y
     ##SECURITY        ACL
     #
     *MACHINES
003  <SITE1 の名前 >     LMID=SITE1
004                    TUXDIR="<TUXDIR>"
005                    APPDIR="<APPDIR>"
                       ENVFILE="<APPDIR>/ENVFILE"
                       TLOGDEVICE="<APPDIR>/TLOG"
                       TLOGNAME=TLOG
                       TUXCONFIG="<APPDIR>/tuxconfig"
006                    TYPE="< マシン・タイプ >"
                       ULOGPFX="<APPDIR>/ULOG"
007  <SITE2 の名前 >  LMID=SITE2
                       TUXDIR="<TUXDIR>"
                       APPDIR="<APPDIR>"
                       ENVFILE="<APPDIR>/ENVFILE"
                       TLOGDEVICE="<APPDIR>/TLOG"
                       TLOGNAME=TLOG
                       TUXCONFIG="<APPDIR>/tuxconfig"
                       TYPE="< マシン・タイプ >"
                       ULOGPFX="<APPDIR>/ULOG"
     #
     *GROUPS
     #
     # 認証サーバのグループ
     #
     Group for Application Queue (/Q) Servers
     #
     ##QGRP1      LMID=SITE1    GRP=102
     ##       TMSNAME=TMS_QM    TMSCOUNT=2
     ##       OPENINFO=”TUXEDO/QM:<APPDIR>/qdevice:QSP_BANKAPP”
     #
     # イベント・ブローカ・サーバのグループ
     #
     ##EVBGRP1     LMID=SITE1    GRPNO=104


     DEFAULT: TMSNAME=TMS_SQL   TMSCOUNT=2
     BANKB1   LMID=SITE1        GRPNO=1


008  OPENINFO="TUXEDO/SQL:<APPDIR>/bankdl1:bankdb:readwrite"
     BANKB2   LMID=SITE2         GRPNO=2
     OPENINFO="TUXEDO/SQL:<APPDIR>/bankdl2:bankdb:readwrite"
     *NETWORK
```

```
009  SITE1      NADDR="<SITE1 のネットワーク・アドレス >"
010             NLSADDR="<SITE1 のネットワーク・リスナ・アドレス >"
011  SITE2      NADDR="<SITE2 のネットワーク・アドレス >"
012             NLSADDR="<SITE2 のネットワーク・リスナ・アドレス >"
```

2. アプリケーションのパスワード機能を有効にするために、次の行を ubbshm または ubbmp の RESOURCES セクションに追加します。

```
SECURITY        APP_PW
```

3. 両方のコンフィギュレーション・ファイルで、一部のパラメータが山かっこ (< >) で囲まれています。山かっこで囲まれた文字列は、実際のインストールに合わせた値に置き換えます。これらのフィールドは、RESOURCES、MACHINES、および GROUPS セクションにあります。ubbmp では、NETWORK セクションにも置き換えが必要な値があります。次の表は、ubbmp の NETWORK セクションについて説明しています。また、単一マシン用のアプリケーションを作成する際、RESOURCES、MACHINES、および GROUPS の各セクションに必要となる変更もすべて示しています。

表 3-3 値の説明

行番号	置き換える文字列	説明
001	UID	掲示板の IPC 構造体の所有者を示す有効なユーザ ID (UID)。マルチプロセッサ・コンフィギュレーションでは、この値はすべてのマシンで同じにする必要があります。この値を BEA Tuxedo システム・ソフトウェアの所有者と同じにすると、簡単に統一できます。
002	GID	掲示板の IPC 構造体の所有者を示す有効なグループ ID (GID)。マルチプロセッサ・コンフィギュレーションでは、この値はすべてのマシンで同じにする必要があります。アプリケーションの各ユーザは、このグループ ID を共有する必要があります。
003	SITE1 の名前	マシン名。UNIX プラットフォームでは、UNIX コマンド uname -n で出力される値を使用します。

表 3-3 値の説明 (続き)

行番号	置き換える文字列	説明
004	TUXDIR	BEA Tuxedo ソフトウェアのルート・ディレクトリの絶対パス名。ファイルに出現するすべての <TUXDIR> を特定のパス名に置き換えます。
005	APPDIR	アプリケーションを実行するディレクトリの絶対パス名。ファイルに出現するすべての <APPDIR> を特定のパス名に置き換えます。
006	マシン・タイプ	異なる種類のマシンが存在するネットワーク・アプリケーションで使用される識別文字列。BEA Tuxedo システムでは、通信を行う各マシンのマシン・タイプの値が確認されます。通信を行う 2 つのマシンのマシン・タイプが異なる場合は、メッセージの符号化 / 復号化ルーチンが呼び出されて、両方のマシンで認識される形式にデータが変換されます。
007	SITE2 の名前	2 番目のマシン名。UNIX プラットフォームでは、UNIX コマンド uname -n で出力される値を使用します。
008	OPENINFO	この文とそれに続くエントリは、BEA Tuxedo システムのリソース・マネージャで認識される形式になっています。ほかのリソース・マネージャで認識されるように変更 (または削除) する必要があります。
009	SITE1 のネットワーク・アドレス	このマシンの BRIDGE プロセスに対するネットワーク・リスナのアドレス。
010	SITE1 のネットワーク・リスナ・アドレス	このマシンの tlisten プロセスに対するネットワーク・リスナのアドレス。
011	SITE2 のネットワーク・アドレス	このマシンの BRIDGE プロセスに対するネットワーク・リスナのアドレス。マシンごとに異なる値を指定する必要があります。

表 3-3 値の説明 (続き)

行番号	置き換える文字列	説明
012	SITE2 のネットワーク・リスナ・アドレス	このマシンの tlisten プロセスに対するネットワーク・リスナのアドレス。

関連項目

- 3-31 ページの「bankapp のファイルおよびリソースの準備」
- 『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5)
- 『BEA Tuxedo アプリケーションの設定』の 2-1 ページの「コンフィギュレーション・ファイルとは」

ステップ 7 とステップ 8: バイナリ形式のコンフィギュレーション・ファイルとトランザクション・ログ・ファイルの作成

バイナリ形式のコンフィギュレーション・ファイルを作成する前に

バイナリ形式のコンフィギュレーション・ファイルを作成する場合、bankapp ファイルが置かれたディレクトリに移動し、環境変数を設定することが必要です。その場合、次の手順に従います。

1. bankapp ファイルが置かれたディレクトリに移動します。
2. 次のコマンドを入力して、環境変数を設定します。

```
. ./bankvar
```

注記　SHM モードで bankapp を実行する場合は、tlisten プロセスを作成したり、別のマシンにトランザクション・ログを作成する必要はありません。

コンフィギュレーション・ファイルのロード

コンフィギュレーション・ファイルを編集したら、それを MASTER マシン上にバイナリ・ファイルとしてロードする必要があります。バイナリ形式のコンフィギュレーション・ファイルの名前は TUXCONFIG、そのパス名は TUXCONFIG に定義されています。このファイルは、BEA Tuxedo のシステム管理者の有効なユーザ ID およびグループ ID を持つユーザが作成します。この 2 つの ID は、ご使用のコンフィギュレーション・ファイルの UID および GID の値と同じであることが必要です。同じではない場合、bankapp の実行時にパーミッションの問題が発生します。

1. 次のコマンドを入力して、TUXCONFIG を作成します。

```
tmloadcf ubbmp
```

   コンフィギュレーションのロード時には、このコンフィギュレーションをインストールするかどうか、およびインストールする場合は既存のコンフィギュレーション・ファイルを上書きすることを確認するメッセージが何回か表示されます。このような確認を省くには、コマンド行で -y オプションを指定します。

2. アプリケーションで必要な IPC 資源を BEA Tuxedo システムで計算する場合は、コマンド行で -c オプションを指定します。

TUXCONFIG は、MASTER マシン上だけにインストールできます。アプリケーションの起動時に tmboot によってほかのマシンに複製転送されます。

コンフィギュレーションのオプションとして SECURITY が指定されている場合、tmloadcf の実行時にアプリケーション・パスワードの入力が求められます。30 文字までのパスワードを指定できます。アプリケーションに参加するクライアント・プロセスでは、パスワードを入力する必要があります。

ロードする前に tmloadcf によってテキスト形式のコンフィギュレーション・ファイル (UBBCONFIG) が解析されます。構文エラーが検出された場合、ファイルのロードは失敗します。

トランザクション・ログ (TLOG) ファイルの作成

TLOG は、BEA Tuxedo システムがグローバル・トランザクションを管理するために使用するトランザクション・ログです。アプリケーションを起動する前に、アプリケーションのすべてのマシン上のすべてのファイルに TLOG のエントリが作成されることが必要です。また、ログ自体のファイルは、MASTER マシン上に作成されることが必要です。

bankapp では、デバイス・リストと TLOG を作成する crtlog と呼ばれるスクリプトが提供されています。デバイス・リストは、bankvar の TLOGDEVICE 変数を使用して作成されます。

1. MASTER マシンで次のコマンドを入力して、TLOG およびデバイス・リストを作成します。

   ```
   crtlog -m
   ```

   注記　開発環境では、デバイス・リストはデータベースで使用されているものと同じでもかまいません。

2. ほかのすべてのマシンでは、-m オプションを指定する必要はありません。システムの起動時に、MASTER 以外のマシン上の BBL によってログが作成されます。

XA 準拠ではないリソース・マネージャを使用している場合、トランザクション・ログは不要です。

関連項目

- 3-31 ページの「bankapp のファイルおよびリソースの準備」

ステップ 9: 各マシン上でのリモート・サービス接続の作成

tlisten はリスナ・プロセスで、BEA Tuxedo アプリケーションのマシン間で tmboot などのプロセスにリモート・サービス接続を提供します。コンフィギュレーション・ファイルの NETWORK セクションで定義されたネットワーク上のすべてのマシンにインストールされていることが必要です。

tlisten の起動については、『BEA Tuxedo システムのインストール』の 6-18 ページの「tlisten プロセスの開始」を参照してください。

1. bankapp 用に新たに tlisten プロセスを起動することをお勧めします。その場合、次のコマンドを入力します。

   ```
   tlisten -l nlsaddr
   ```

   nlsaddr の値は、コンフィギュレーション・ファイルでこのマシンの NLSADDR パラメータに指定された値と同じにする必要があります。この値はマシンによって異なるため、tlisten の引数がコンフィギュレーション・ファイルで指定された値と一致していることが重要です。

   注記　この指定に関するエラーは、簡単には検出できません。tmloadcf では、コンフィギュレーション・ファイルの値と tlisten コマンドの引数が一致しているかどうかは確認されません。この 2 つのアドレスが異なる場合、nlsaddr の値が一致しないマシンで起動に失敗します。また、tlisten プロセスが開始されていないマシンでも起動に失敗します。

tlisten で使用されるログ・ファイルは、BEA Tuxedo システムのほかのすべてのログ・ファイルとは別のログ・ファイルです。ただし、複数の tlisten プロセスで 1 つのログ・ファイルを使用できます。デフォルトのファイル名は、TUXDIR/udataobj/tlog です。

リスナ・プロセス (tlisten) の停止

tlisten は、デーモン・プロセスとして実行されます。tlisten を起動スクリプトに組み込んだり、cron ジョブとして実行する方法については、『BEA Tuxedo コマンド・リファレンス』の tlisten(1) を参照してください。

bankapp では、tlisten に対して開始または終了のどちらかの操作しかできません。tlisten を終了するには、次のように SIGTERM シグナルを送信します。

kill -15 *pid*

注記　Windows 2000 環境でリスナ・プロセスを開始または停止するには、コマンド行で tlisten を実行するか、またはコントロール・パネルを使用します。

tlisten のエラー・メッセージの例

tlisten がリモートで実行されていない場合、次のように画面上にブート処理に関するメッセージが表示されます。

```
Booting admin processes
exec DBBL -A :
      on MASTER -> process id=17160Started.
exec BBL -A :
      on MASTER -> process id=17161Started.
exec BBL -A :
      on NONMAST2 -> CMDTUX_CAT:814: cannot propagate TUXCONFIG file
tmboot:WARNING: No BBL available on site NONMAST2.
      Will not attempt to boot server processes on that site.
exec BBL -A :
      on NONMAST1 -> CMDTUX_CAT:814: cannot propagate TUXCONFIG file
tmboot:WARNING: No BBL available on site NONMAST1.
      Will not attempt to boot server processes on that site.
2 processes started.
```

```
    and messages such as these will be in the ULOG:
    133757.mach1!DBBL.17160:LIBTUX_CAT:262: std main starting
    133800.mach1!BBL.17161:LIBTUX_CAT:262: std main starting
    133804.mach1!BRIDGE.17162:LIBTUX_CAT:262: std main starting
    133805.mach1!tmboot.17159:LIBTUX_CAT:278: Could not contact NLS on NONMAST2
    133805.mach1!tmboot.17159:LIBTUX_CAT:276: No NLS available for remote
      machine NONMAST2
    133806.mach1!tmboot.17159:LIBTUX_CAT:276: No NLS available for remote
      machine NONMAST2
    133806.mach1!tmboot.17159:CMDTUX_CAT:850: Error sending TUXCONFIG
      propagation request to TAGENT on NONMAST2
    133806.mach1!tmboot.17159:WARNING: No BBL available on site NONMAST2.
           Will not attempt to boot server processes on that site.
    133806.mach1!tmboot.17159:LIBTUX_CAT:278: Could not contact NLS on NONMAST1
    133806.mach1!tmboot.17159:LIBTUX_CAT:276: No NLS available for
      remote machine NONMAST1
    133806.mach1!tmboot.17159:LIBTUX_CAT:276: No NLS available for
     remote machine NONMAST1
    133806.mach1!tmboot.17159:CMDTUX_CAT:850: Error sending TUXCONFIG
      propagation request to TAGENT on NONMAST1
    133806.mach1!tmboot.17159:WARNING: No BBL available on site NONMAST1.
            Will not attempt to boot server processes on that site.
If tlisten is started with the wrong machine address, the following messages
appear in the tlisten log.

    Mon Aug 26 10:51:56 1991; 14240; BEA TUXEDO System Listener Process Started
    Mon Aug 26 10:51:56 1991; 14240; Could not establish listening endpoint
    Mon Aug 26 10:51:56 1991; 14240; Terminating listener process, SIGTERM
```

関連項目

- 3-31 ページの「bankapp のファイルおよびリソースの準備」
- `tlisten(1)`
- `tmadmin(1)`
- `tmloadcf(1)`

bankapp の起動

ここでは、bankapp を起動し、各種のクライアント・プログラムとトランザクションを行ってテストし、終了する手順について順に説明します。各作業をクリックすると、その作業を行う手順が表示されます。

ステップ 1: 起動する前に行う準備作業

1. bankapp を起動する前に、アプリケーションをサポートするのに十分な IPC 資源がマシンにあることを確認します。IPC 資源に関するレポートを出力するには、tmboot コマンドに -c オプションを指定します。

   注記　IPC 資源が不足していると起動が失敗する場合があるので、コンフィギュレーションに対して適切な値が指定されていることを確認してください。

コード リスト 3-19　IPC **レポート**

```
Ipc sizing (minimum /T values only)
               Fixed Minimums Per Processor
SHMMIN:1
```

```
SHMALL:1
SEMMAP: SEMMNI
                Variable Minimums Per Processor
                SEMUME,           A                              SHMMAX
                SEMMNU,           *                                *
Node            SEMMNS  SEMMSL  SEMMSL  SEMMNI  MSGMNI  MSGMAP  SHMSEG
------          ------  ------  ------  ------  ------  ------  ------
sfpup               60       1      60   A + 1      10      20     76K
sfsup               63       5      63   A + 1      11      22     76K
where 1 <= A <= 8.
```

2. プロセッサごとに使用されるアプリケーション・クライアントの数を各 MSGMNI 値に追加します。MSGMAP は MSGMNI の 2 倍にします。
3. IPC の最低条件とご使用のマシンに対して設定されたパラメータとを比較します。これらのパラメータを定義する場所は、プラットフォームによって異なります。
   - ほとんどの UNIX システム・プラットフォームの場合、マシンのパラメータは /etc/conf/cf.d/mtune に定義されています。
   - Windows 2000 プラットフォームの場合、マシンのパラメータはコントロール・パネルで設定したり表示します。

関連項目

- 3-69 ページの「bankapp の起動」

ステップ 2: bankapp の起動

1. 環境変数を設定します。

   ```
   . ./bankvar
   ```

2. 次のコマンドを入力して、アプリケーションを起動します。

   ```
   tmboot
   ```

   次のプロンプトが表示されます。

   ```
   Boot all admin and server processes?(y/n):y
   ```

   次のようなレポートが画面に出力されます

   ```
   Booting all admin and server processes in
   /usr/me/appdir/tuxconfig
   Booting all admin processes
   exec BBL -A:
              process id=24223 Started.
   ```

このレポートは、コンフィギュレーション内のすべてのサーバが起動するまで出力されます。起動したサーバの合計数が出力された時点で終了します。

必要であれば、コンフィギュレーションの一部のサーバだけを起動することもできます。たとえば、管理サーバだけを起動するには、`-A` オプションを指定します。オプションが指定されていない場合は、アプリケーション全体が起動します。

`tmboot` では、起動したサーバ数がレポートされるほかに、`ULOG` にメッセージが送信されます。

関連項目

- 3-69 ページの「bankapp の起動」

ステップ 3: データベースへのデータの追加

populate.sh は、データベースにレコードを追加するシェル・スクリプトです。このスクリプトを使用して、bankapp を実行して、その機能をテストします。populate は、gendata と呼ばれるプログラムからシステム・サーバ ud にレコードを渡す 1 行のスクリプトです。gendata プログラムは、10 支店、30 窓口、200 口座のレコードを作成します。作成されたファイル内のレコードは、pop.out に記録されるので、サービス要求の作成時にデータベースの値を使用することができます。

スクリプトを実行するには、「populate」と入力します。

注記　populate スクリプトで出力された pop.out では、口座番号や支店番号などのフィールドを指定して取得できるので、自分のサービス要求に対する出力を生成できます。

関連項目

- 3-69 ページの「bankapp の起動」
- 『BEA Tuxedo コマンド・リファレンス』の tmboot(1)
- 『BEA Tuxedo コマンド・リファレンス』の ud、wud(1)
- 『BEA Tuxedo C リファレンス』の userlog(3c)
- 『BEA Tuxedo アプリケーション実行時の管理』の 2-20 ページの「ユーザ・ログ (ULOG) とは」
- 『BEA Tuxedo アプリケーション実行時の管理』の 1-10 ページの「アプリケーションの起動」
- 『BEA Tuxedo アプリケーション実行時の管理』の 1-14 ページの「アプリケーションのシャットダウン」

ステップ 4: bankapp サービスのテスト

1. 実行中のシステムにログインする場合は、bankapp に環境変数を設定する必要があります。その場合、次のコマンドを入力します。

   ```
   . ./bankvar
   ```

2. audit クライアント・プログラムを実行します。その場合、次のコマンドを入力します。

   ```
   audit {-a | -t} [branch_id]
   ```

   口座残高を取得する場合は -a、窓口残高を取得する場合は -t を指定します。branch_id が指定されている場合、指定された支店だけがレポートされます。指定されていない場合、すべての支店データがレポートされます。口座番号、支店番号など audit に入力する値には、populate プログラムの出力である pop.out にリストされている値を使用できます。

3. auditcon を実行します。audit プログラムの会話型バージョンを起動するには、次のコマンドを入力します。

   ```
   auditcon
   ```

   画面に次のメッセージが表示されます。

   ```
   to request a TELLER or ACCOUNT balance for a branch,
   type the letter t or a, followed by the branch id,
   followed by <return>
   for ALL TELLER or ACCOUNT balances, type t or a <return>
   q <return> quits the program
   ```

   要求を入力して Enter キーを押すと、指定された情報と次のメッセージが表示されます。

   ```
   another balance request ??
   ```

4. このメッセージは、q を入力するまで繰り返し表示されます。

5. driver プログラムを使用します。デフォルトでは、driver プログラムは 300 のトランザクションを生成します。トランザクションの数を変更する場合は、次のように -n オプションを使用します。

   ```
   driver -n1000
   ```

このコマンドは、プログラムが 1000 回ループして実行されることを指定しています。

driver は、システム上での処理をシミュレートするための一連のトランザクションを生成するスクリプトです。このスクリプトは、bankapp の一部として含まれているので、tmadmin コマンドを実行して実際的な統計を取得できます。

関連項目

- 3-69 ページの「bankapp の起動」

ステップ 5: bankapp のシャットダウン

bankapp を終了するには、次のように、引数を指定せずに tmshutdown(1) コマンドを MASTER マシンで入力します。

```
$ tmshutdown
Shutdown all server processes?(y/n): y
Shutting down all server processes in /usr/me/BANKAPP/TUXCONFIG
Shutting down server processes ...

Server Id = 1 Group Id = BANKB1 Machine = Site1:shutdown succeeded.
```

このコマンド (または tmadmin のシャットダウン・コマンド) を実行すると、次のタスクが行われます。

- すべてのアプリケーション・サーバ、ゲートウェイ・サーバ、TMS サーバ、および管理サーバがシャットダウンします。
- 割り当てられていたすべての IPC 資源が削除されます。

関連項目

- 3-69 ページの「bankapp の起動」

- 『BEA Tuxedo コマンド・リファレンス』の `tmadmin(1)`
- 『BEA Tuxedo コマンド・リファレンス』の `tmshutdown(1)`

4 CSIMPAPP (簡単な COBOL アプリケーション) のチュートリアル

ここでは、次の内容について説明します。

- CSIMPAPP とは
- CSIMPAPP のファイルおよびリソースの準備
  - ステップ 1: CSIMPAPP ファイルのコピー
  - ステップ 2: クライアントの検証およびコンパイル
  - ステップ 3: サーバの検証およびコンパイル
  - ステップ 4: コンフィギュレーション・ファイルの編集およびロード
  - ステップ 5: アプリケーションの起動
  - ステップ 6: ランタイム・アプリケーションのテスト
  - ステップ 7: ランタイム・アプリケーションの監視
  - ステップ 8: アプリケーションのシャットダウン

CSIMPAPP とは

CSIMPAPP は、BEA Tuxedo システムに同梱の ATMI サンプル・アプリケーションです。ここで説明する内容は Microfocus COBOL コンパイラを前提としています。そのため、ご使用のコンパイラによって多少手順が異なる場合があります。BEA Tuxedo システムでサポートされている COBOL プラットフォームを確認するには、『BEA Tuxedo システムのインストール』の付録 A 「BEA Tuxedo 8.1 プラットフォーム・データ・シート」を参照してください。

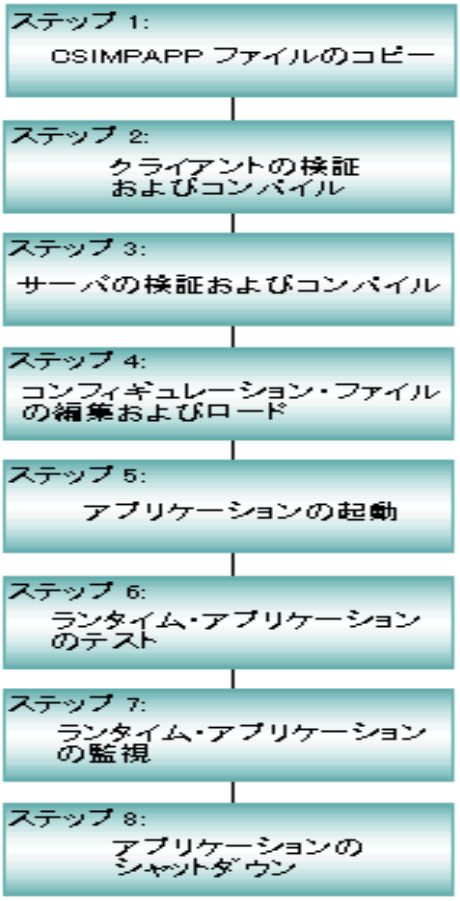
CSIMPAPP では、クライアントとサーバがそれぞれ 1 つずつ使用されています。サーバでは、1 つのサービスだけが実行されます。つまり、クライアントから文字列を受け取り、その文字列を大文字で返します。

CSIMPAPP のファイルおよびリソースの準備

ここでは、CSIMPAPP の開発を始めるための準備作業について説明します。次の図は、それらの準備作業をまとめたものです。

各作業をクリックすると、その作業を行う手順が表示されます。

図 4-1CSIMPAPP の開発プロセス



はじめに

このチュートリアルのサンプル・アプリケーションを実行するには、BEA Tuxedo ATMI のクライアント / サーバ・ソフトウェアがインストールされ、ここで説明するファイルやコマンドを使用できることが必要です。BEA Tuxedo システム・ソフトウェアのインストールを行う場合は、『BEA Tuxedo システムのインストール』を参照してください。既にインストールされている場合は、ソフトウェアのインストール先ディレクトリのパス名 (TUXDIR) を確認する必要があります。また、BEA Tuxedo システム・ディレクトリ構造内のディレクトリとファイルに読み取りパーミッションと実行パーミッションを設定し、CSIMPAPP の各ファイルをコピーしたり、BEA Tuxedo システムの各コマンドを実行できるようにします。

このチュートリアルの目的

ここで説明する作業を行うと、クライアントおよびサーバが実行できるタスクについて理解し、環境に応じてコンフィギュレーション・ファイルを変更でき、tmadmin を呼び出してアプリケーションの動作を確認できるようになります。つまり、BEA Tuxedo のすべてのアプリケーションの基本的な要素 (クライアント・プロセス、サーバ・プロセス、コンフィギュレーション・ファイルなど) を理解し、BEA Tuxedo システムの各コマンドを使用してアプリケーションを管理できるようになります。

ステップ 1: CSIMPAPP ファイルのコピー

1. CSIMPAPP 用にディレクトリを作成し、そのディレクトリに移動します。

```
mkdir CSIMPDIR
cd CSIMPDIR
```

注記　この作業は省略せずに行ってください。この作業を行うと、最初からあった CSIMPAPP のファイルと、手順に従って作成したファイルを確認できるようになります。csh を使用せずに、標準シェル (/bin/sh) または Korn シェルを使用してください。

2. 環境変数を設定し、エクスポートします。

```
TUXDIR=<BEA Tuxedo システム・ルート・ディレクトリのパス名>
APPDIR=< 現在の作業ディレクトリのパス名>
TUXCONFIG=$APPDIR/TUXCONFIG
COBDIR=<COBOL コンパイラ・ディレクトリのパス名>
COBCPY=$TUXDIR/cobinclude
COBOPT="-C ANS85 -C ALIGN=8 -C NOIBMCOMP -C TRUNC=ANSI -C OSEXT=cbl"
CFLAGS="-I$TUXDIR/include"
PATH=$TUXDIR/bin:$APPDIR: $PATH
LD_LIBRARY_PATH=$COBDIR/coblib:${LD_LIBRARY_PATH}
export TUXDIR APPDIR TUXCONFIG UBBCONFIG COBDIR COBCPY
export COBOPT CFLAGS PATH LD_LIBRARY_PATH
```

TUXDIR および PATH を設定して、BEA Tuxedo ディレクトリ構造内のファイルにアクセスでき、BEA Tuxedo コマンドを実行できるようにします。

- Sun Solaris では、PATH の最上位ディレクトリとして /usr/5bin を指定する必要があります。
- AIX プラットフォーム (RS/6000) では、LD_LIBRARY_PATH ではなく LIBPATH を使用します。
- HP-UX プラットフォーム (HP 9000) では、LD_LIBRARY_PATH の代わりに SHLIB_PATH を使用します。ステップ 4 で示すように、TUXCONFIG を設定して、コンフィギュレーション・ファイルをロードできるようにする必要があります。

3. CSIMPAPP ファイルをコピーします。

```
cp TUXDIR/samples/atmi/CSIMPAPP/* .
```

注記　一部のファイルは、後で編集して実行可能ファイルを作成します。そのため、ソフトウェアに同梱のオリジナルのファイルではなく、そのコピーを使用することをお勧めします。

4. ファイルを一覧表示します。

```
$ ls
CSIMPCL.cbl
```

```
CSIMPSRV.cbl
README
TPSVRINIT.cbl
UBBCSIMPLE
WUBBCSIMPLE
envfile
ws
$
```

このアプリケーションを構成するファイルは、次のとおりです。

- CSIMPCL.cbl—クライアント・プログラムのソース・コード
- CSIMPSRV.cbl—サーバ・プログラムのソース・コード
- TPSVRINIT.cbl—サーバ初期化プログラムのソース・コード
- UBBCSIMPLE—このアプリケーション用のテキスト形式のコンフィギュレーション・ファイル
- WUBBCSIMPLE—ワークステーションのサンプル用コンフィギュレーション・ファイル
- ws—3 種類のワークステーション・プラットフォームに対するクライアント・プログラムの .MAK ファイルが置かれたディレクトリ

ステップ 2: クライアントの検証およびコンパイル

クライアント・プログラムの検証

クライアント・プログラムのソース・コードの内容を確認します。

```
$ more CSIMPCL.cbl
```

以下に出力結果を示します。

コード リスト 4-1 CSIMPCL.cbl **のソース・コード**

```
1       IDENTIFICATION DIVISION.
2       PROGRAM-ID. CSIMPCL.
```

```
3       AUTHOR. Tuxedo DEVELOPMENT.
4       ENVIRONMENT DIVISION.
5       CONFIGURATION SECTION.
6       WORKING-STORAGE SECTION.
7   ******************************************************
8   * Tuxedo definitions
9   ******************************************************
10      01 TPTYPE-REC.
11      COPY TPTYPE.
12   *
13      01 TPSTATUS-REC.
14      COPY TPSTATUS.
15   *
16      01 TPSVCDEF-REC.
17      COPY TPSVCDEF.
18   *
19      01 TPINFDEF-REC VALUE LOW-VALUES.
20      COPY TPINFDEF.
21   ******************************************************
22   * Log messages definitions
23   ******************************************************
24      01 LOGMSG.
25             05 FILLER   PIC X(8) VALUE "CSIMPCL:".
26             05 LOGMSG-TEXT PIC X(50).
27      01 LOGMSG-LEN      PIC S9(9) COMP-5.
28   *
29      01 USER-DATA-REC   PIC X(75).
30      01 SEND-STRING     PIC X(100) VALUE SPACES.
31      01 RECV-STRING     PIC X(100) VALUE SPACES.
32   ******************************************************
33   * Command line arguments
34   ******************************************************
35   * Start program with command line args
36   ******************************************************
37
38   PROCEDURE
39   START-CSIMPCL.
40      MOVE LENGTH OF LOGMSG TO LOGMSG-LEN.
41      ACCEPT SEND-STRING FROM COMMAND-LINE.
42      DISPLAY “SEND-STRING:” SEND-STRING.
43
44      MOVE “Started” TO LOGMSG-TEXT.
45      PERFORM DO-TPINIT.
46      PERFORM DO-TPCALL.
47       DISPLAY “RECV-STRING:” RECV-STRING.
48       PERFORM DO-TPTERM.
49       PERFORM EXIT-PROGRAM.
50   ******************************************************
51   * Now register the client with the system.
```

```
52  ***************************************************
53  DO-TPINIT.
54     MOVE SPACES TO USRNAME.
55     MOVE SPACES TO CLTNAME.
56     MOVE SPACES TO PASSWD.
57     MOVE SPACES TO GRPNAME.
58     MOVE ZERO TO DATALEN.
59     SET TPU-DIP TO TRUE.
60
61    CALL "TPINITIALIZE" USING TPINFDEF-REC
62              USER-DATA-REC
63              TPSTATUS-REC.
64
65     IF NOT TPOK
66              MOVE "TPINITIALIZE Failed" TO LOGMSG-TEXT
67              PERFORM DO-USERLOG
68              PERFORM EXIT-PROGRAM
69     END-IF.
70


71  ***************************************************
72  * Issue a TPCALL
73  ***************************************************
74  DO-TPCALL.
75     MOVE 100 to LEN.
76     MOVE "STRING" TO REC-TYPE.
77     MOVE "CSIMPSRV" TO SERVICE-NAME.
78     SET TPBLOCK TO TRUE.
79     SET TPNOTRAN TO TRUE.
80     SET TPNOTIME TO TRUE.
81     SET TPSIGRSTRT TO TRUE.
82     SET TPCHANGE TO TRUE.
83
84     CALL "TPCALL" USING TPSVCDEF-REC
85                  TPTYPE-REC
86                  SEND-STRING
87                  TPTYPE-REC
88                  RECV-STRING
89                  TPSTATUS-REC.
90
91     IF NOT TPOK
92           MOVE "TPCALL Failed" TO LOGMSG-TEXT
93           PERFORM DO-USERLOG
94     END-IF.
95
96    ***************************************************
97    * Leave Tuxedo
98    ***************************************************
```

```
99   DO-TPTERM.
100    CALL "TPTERM" USING TPSTATUS-REC.
101    IF NOT TPOK
102           MOVE "TPTERM Failed" TO LOGMSG-TEXT
103           PERFORM DO-USERLOG
104    END-IF.
105
106  ************************************************
107  * Log messages to the userlog
108  ************************************************
109  DO-USERLOG.
110    CALL "USERLOG" USING LOGMSG
111           LOGMSG-LEN
112           TPSTATUS-REC.
113
114  ************************************************
115  *Leave Application
116  ************************************************
117  EXIT-PROGRAM.
118    MOVE "Ended" TO LOGMSG-TEXT.


119    PERFORM DO-USERLOG.
120    STOP RUN.
```

表 4-1CSIMPCL.cbl **ソース・コードでの重要なコード行**

行数	ファイル / 関数	目的
11, 14, 17, 20	COPY	BEA Tuxedo ATMI 関数を使用する場合に、常に必要となるファイルを複製するためのコマンド。
61	TPINITIALIZE	クライアント・プログラムがアプリケーションに参加する際に使用される ATMI 関数。

表 4-1CSIMPCL.cbl ソース・コードでの重要なコード行 (続き)

行数	ファイル / 関数	目的
84	TPCALL	SERVICE-NAME で指定されたサービスにメッセージ・レコードを送信する際に使用される ATMI 関数。TPCALL は、メッセージが返されるまで待機します。STRING は、BEA Tuxedo の 3 つの基本的なレコード・タイプのいずれかになります。LEN IN TPTYPE-REC 引数は、USER-DATA-REC のレコードの長さを指定します。
100	TPTERM	アプリケーションを終了するための ATMI 関数。TPTERM は、STOP RUN を実行する前にアプリケーションを終了する場合に呼び出します。
110	USERLOG	tpcall が正常に実行された場合に、サーバから返されるメッセージを表示する関数。

クライアント・プログラムのコンパイル

1. buildclient を実行して、クライアント・プログラムをコンパイルします。

   ```
   buildclient -C -o CSIMPCL -f CSIMPCL.cbl
   ```

   CSIMPCL は出力ファイル、CSIMPCL.cbl は入力ソース・ファイルです。

2. 結果を確認します。

   ```
   $ ls CSIMPCL*
   CSIMPCL   CSIMPCL.cbl   CSIMPCL.idy   CSIMPCL.int   CSIMPCL.o
   ```

   CSIMPCL という実行可能モジュールが作成されました。

関連項目

- 『BEA Tuxedo コマンド・リファレンス』の buildclient(1)

- 『BEA Tuxedo COBOL リファレンス』の TPINITIALIZE(3cbl)
- 『BEA Tuxedo COBOL リファレンス』の TPTERM(3cbl)
- 『BEA Tuxedo COBOL リファレンス』の TPCALL(3cbl)
- 『BEA Tuxedo COBOL リファレンス』の USERLOG(3cbl)

ステップ 3: サーバの検証およびコンパイル

サーバ・プログラムの検証

1. CSIMPSRV ATMI サーバ・プログラムのソース・コードの内容を確認します。

   ```
   $ more CSIMPSRV.cbl
   ```

コード リスト 4-2 CSIMPSRV.cbl **のソース・コード**

```
1       IDENTIFICATION DIVISION.
2       PROGRAM-ID. CSIMPSRV.
3       AUTHOR. BEA Tuxedo DEVELOPMENT.
4       ENVIRONMENT DIVISION.
5       CONFIGURATION SECTION.
6       WORKING-STORAGE SECTION.
7   ******************************************************
8   * Tuxedo definitions
9   ******************************************************
10      01 TPSVCRET-REC.
11      COPY TPSVCRET.
12   *
13      01 TPTYPE-REC.
14      COPY TPTYPE.
15   *
16      01 TPSTATUS-REC.
17      COPY TPSTATUS.
18   *
```

```
19     01 TPSVCDEF-REC.
20     COPY TPSVCDEF.
21   ******************************************************
22   * Log message definitions
23   ******************************************************
24     01 LOGMSG.
25              05 FILLER        PIC X(10) VALUE
26                     "CSIMPSRV :".
27              05 LOGMSG-TEXT   PIC X(50).
28     01 LOGMSG-LEN             PIC S9(9) COMP-5.
29   ******************************************************
31   * User defined data records
32   ******************************************************
33     01 RECV-STRING            PIC X(100).
34     01 SEND-STRING            PIC X(100).
35   *
36     LINKAGE SECTION.
37   *
38     PROCEDURE DIVISION.
39   *
40    START-FUNDUPSR.
41     MOVE LENGTH OF LOGMSG TO LOGMSG-LEN.
42     MOVE "Started" TO LOGMSG-TEXT.
43     PERFORM DO-USERLOG.
44
45   ******************************************************
46   * Get the data that was sent by the client
47   ******************************************************
48     MOVE LENGTH OF RECV-STRING TO LEN.
49     CALL "TPSVCSTART" USING TPSVCDEF-REC
50                     TPTYPE-REC
51                     RECV-STRING
52                     TPSTATUS-REC.
53
54     IF NOT TPOK
55            MOVE "TPSVCSTART Failed" TO LOGMSG-TEXT
56            PERFORM DO-USERLOG
57            PERFORM EXIT-PROGRAM
58     END-IF.
59
60     IF TPTRUNCATE
61            MOVE "Data was truncated" TO LOGMSG-TEXT
62            PERFORM DO-USERLOG
63            PERFORM EXIT-PROGRAM
64     END-IF.


65
66     INSPECT RECV-STRING CONVERTING
```

```
67      "abcdefghijklmnopqrstuvwxyz" TO
68      "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
69      MOVE "Success" TO LOGMSG-TEXT.
70      PERFORM DO-USERLOG.
71      SET TPSUCCESS TO TRUE.
72      COPY TPRETURN REPLACING
73             DATA-REC BY RECV-STRING.
74
75   ****************************************************
76   * Write out a log err messages
77   ****************************************************
78    DO-USERLOG.
79      CALL "USERLOG" USING LOGMSG
80             LOGMSG-LEN
81             TPSTATUS-REC.
82   ****************************************************
83   * EXIT PROGRAM
84   ****************************************************
85    EXIT-PROGRAM.
86      MOVE "Failed" TO LOGMSG-TEXT.
87      PERFORM DO-USERLOG.
88      SET TPFAIL TO TRUE.
89      COPY TPRETURN REPLACING
90                DATA-REC BY RECV-STRING.
```

表 4-2CSIMPSRV.cbl **ソース・コードでの重要なコード行**

行数	ルーチン	目的
49	TPSVCSTART	このサービスを開始し、サービスのパラメータとデータを取得するためのルーチン。呼び出しが成功すると、クライアントによって送信されたデータが RECV-STRING に格納されます。
66-68	INSPECT 文	入力を大文字に変換する文 (Microfocus 固有)。
72	COPY TPRETURN	TPSUCCESS に設定された値と、変換された文字列をクライアントに返すコマンド行。
79	USERLOG	BEA Tuxedo システムとアプリケーションで使用されるメッセージを記録するルーチン。

2. サーバの初期化時 (つまり、サービス要求の処理がサーバで開始される前) に、BEA Tuxedo システムによって TPSVRINIT サブルーチンが呼び出されます。TPSVRINIT を理解するには、そのソース・コードの内容を参照します。

   ```
   $ more TPSVRINIT.cbl
   ```

コード リスト 4-3 TPSVRINIT.cbl **のソース・コード**

```
1   IDENTIFICATION DIVISION.
2   PROGRAM-ID. TPSVRINIT.
3   ENVIRONMENT DIVISION.
4   CONFIGURATION SECTION.
5  *
6   DATA DIVISION.
7   WORKING-STORAGE SECTION.
8  *
9   01 LOGMSG.
10      05 FILLER               PIC X(11) VALUE "TPSVRINIT :".
11      05 LOGMSG-TEXT          PIC X(50).
12  01 LOGMSG-LEN               PIC S9(9) COMP-5.
13  *
14   01 TPSTATUS-REC.
15   COPY TPSTATUS.
16  ********************************************************
17   LINKAGE SECTION.
18   01 CMD-LINE.
19    05 ARGC PIC 9(4) COMP-5.
20    05 ARG.
21      10 ARGS PIC X OCCURS 0 TO 9999 DEPENDING ON ARGC.
22  *
23   01 SERVER-INIT-STATUS.
24   COPY TPSTATUS.
25  ********************************************************
26   PROCEDURE DIVISION USING CMD-LINE SERVER-INIT-STATUS.
27   A-000.
28  MOVE LENGTH OF LOGMSG TO LOGMSG-LEN.
29  ********************************************************
30  * There are no command line parameters in this TPSVRINIT
31  ********************************************************
32     IF ARG NOT EQUAL TO SPACES
33            MOVE "TPSVRINIT failed" TO LOGMSG-TEXT
34            CALL "USERLOG" USING LOGMSG
35                   LOGMSG-LEN
36                   TPSTATUS-REC
37     ELSE
38            MOVE "Welcome to the simple service" TO LOGMSG-TEXT
```

```
39            CALL "USERLOG" USING LOGMSG
40                   LOGMSG-LEN
41                   TPSTATUS-REC
42     END-IF.
43   *
44    SET TPOK IN SERVER-INIT-STATUS TO TRUE.
45   *
46    EXIT PROGRAM.
```

BEA Tuxedo システムでは、デフォルトでサーバが起動したことを示すメッセージが USERLOG に書き込まれます。

サーバ・プログラムのコンパイル

1. 次に示すように buildserver を実行して、ATMI サーバ・プログラムをコンパイルします。

```
buildserver -C -o CSIMPSRV -f CSIMPSRV.cbl -f TPSVRINIT.cbl -s CSIMPSRV
```

   CSIMPSRV は作成される実行可能ファイル、CSIMPSRV.cbl および TPSVRINIT.cbl は入力ソース・ファイルです。-s CSIMPSRV は、CSIMPSRV サーバによって提供されているサービスです。

2. カレント・ディレクトリにあるファイルのリストを出力し、結果を確認します。

```
$ ls
CSIMPCL      CSIMPCL.int   CSIMPSRV.cbl    CSIMPSRV.o     TPSVRINIT.int
CSIMPCL.cbl  CSIMPCL.o     CSIMPSRV.idy    TPSVRINIT.cbl  TPSVRINIT.o
CSIMPCL.idy  CSIMPSRV      CSIMPSRV.int    TPSVRINIT.idy  UBBCSIMPLE
```

   CSIMPSRV 実行可能モジュールが作成されました。

関連項目

- 『BEA Tuxedo コマンド・リファレンス』の buildserver(1)
- 『BEA Tuxedo COBOL リファレンス』の TPSVCSTART(3cbl)

- 『BEA Tuxedo COBOL リファレンス』の TPSVRINIT(3cbl)
- 『BEA Tuxedo COBOL リファレンス』の TPRETURN(3cbl)
- 『BEA Tuxedo COBOL リファレンス』の USERLOG(3cbl)

ステップ 4: コンフィギュレーション・ファイルの編集およびロード

コンフィギュレーション・ファイルの編集

1. テキスト・エディタで、CSIMPAPP のコンフィギュレーション・ファイルの内容を確認します。

コード リスト 4-4 CSIMPAPP **コンフィギュレーション・ファイル**

```
# 簡単な BEA Tuxedo COBOL アプリケーション用の UBBCONFIG スケルトン・ファイルです。
#< 山かっこで囲まれた文字列 > を適切な値に置き換えます。

*RESOURCES
IPCKEY              < 有効な IPC キーで置き換えます。>

# 例 :
#IPCKEY             123456

DOMAINID            UBBCSIMPLE
MASTER              simple
MAXACCESSERS        5
MAXSERVERS          5
MAXSERVICES         10
MODEL               SHM
LDBAL               N

*MACHINES
DEFAULT:
                    APPDIR="< 現在のパス名で置き換えます。>"
                    TUXCONFIG="<TUXCONFIG のパス名で置き換えます。>"
                    TUXDIR="<BEA Tuxedo のルート・ディレクトリ (/ 以外 ) を指定します。>"
```

```
                    ENVFILE="< 環境変数ファイルのパス名を指定します。>"
# 例：
#                   APPDIR="/home/me/simpapp"
#                   TUXCONFIG="/home/me/simpapp/TUXCONFIG"
#                   TUXDIR="/usr/tuxedo"
#                   ENVFILE=”/home/me/simpapp/envfile”
<Machine-name>      LMID=simple

# 例：
#usltux             LMID=simple

*GROUPS
GROUP1
                    LMID=simple     GRPNO=1     OPENINFO=NONE

*SERVERS
DEFAULT:
CLOPT="-A"

CSIMPSRV            SRVGRP=GROUP1   SRVID=1

*SERVICES
CSIMPSRV
```

2. 山かっこで囲まれた各*文字列*を適切な値に置き換えます。
   - IPCKEY—ほかのユーザと競合しない値を指定します。
   - TUXCONFIG—TUXCONFIG バイナリ・ファイルの絶対パス名を指定します。
   - TUXDIR—BEA Tuxedo システム・ルート・ディレクトリの絶対パス名を指定します。
   - APPDIR—アプリケーションを起動するディレクトリ (この例ではカレント・ディレクトリ) の絶対パス名を指定します。
   - ENVFILE—mc、viewc、tmloadcf などで使用される環境ファイルの絶対パス名を指定します。
   - *machine-name*—UNIX プラットフォームの uname -n コマンドで返されるマシン名を指定します。

注記　TUXCONFIG および TUXDIR のパス名は、既に設定してエクスポートした TUXCONFIG および TUXDIR のパス名と一致する必要があります。実際のパス名を指定してください。TUXCONFIG などの環境変数で参照されるパス名は使用できません。山かっこは必ず削除してください。

コンフィギュレーション・ファイルのロード

1. tmloadcf を実行して、コンフィギュレーション・ファイルをロードします。

```
$ tmloadcf UBBCSIMPLE
Initialize TUXCONFIG file: /usr/me/CSIMPDIR/TUXCONFIG [y, q] ? y
$
```

2. カレント・ディレクトリにあるファイルのリストを出力し、結果を確認します。

```
$ ls
CSIMPCL       CSIMPCL.o     CSIMPSRV.int    TPSVRINIT.int
CSIMPCL.cbl   CSIMPSRV      CSIMPSRV.o      TPSVRINIT.o
CSIMPCL.idy   CSIMPSRV.cbl  TPSVRINIT.cbl   TUXCONFIG
CSIMPCL.int   CSIMPSRV.idy  TPSVRINIT.idy   UBBCSIMPLE
```

BEA Tuxedo システムによって制御される新しいファイル TUXCONFIG がロードされました。

関連項目

- 『BEA Tuxedo コマンド・リファレンス』の tmloadcf(1)
- 『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5)

ステップ 5: アプリケーションの起動

tmboot を実行して、アプリケーションを起動します。

```
$ tmboot
Boot all admin and server processes?(y/n): y
Booting all admin and server processes in /usr/me/CSIMPDIR/TUXCONFIG

Booting all admin processes ...

exec BBL -A:
       process id=24223 ...Started.

Booting server processes ...

exec CSIMPSRV -A :
      process id=24257 ... Started.
2 processes started.
$
```

BBL (Bulletin Board Liaison) は、アプリケーションの共用メモリを監視する管理プロセスです。CSIMPSRV は、要求を受け取るために継続的に実行している CSIMPAPP サーバです。

関連項目

- 『BEA Tuxedo コマンド・リファレンス』の tmboot(1)

ステップ 6: ランタイム・アプリケーションのテスト

CSIMPAPP をテストするには、クライアントから要求を送信します。

```
$ CSIMPCL “hello world”
HELLO WORLD
```

ステップ 7: ランタイム・アプリケーションの監視

システム管理者は tmadmin コマンド・インタプリタを使用して、アプリケーションを調べ、動的に変更を加えることができます。tmadmin を実行するには、TUXCONFIG 変数を設定する必要があります。

tmadmin を使用すると、50 個以上のコマンドを解釈したり実行することができます。各コマンドの説明については、『BEA Tuxedo コマンド・リファレンス』の tmadmin(1) を参照してください。この例では、2 つの tmadmin コマンドを使用します。

1. 次のコマンドを入力します。

   ```
   tmadmin
   ```

   次が出力されます。

   ```
   tmadmin - Copyright (c) 1999 BEA Systems Inc.; 1991 USL.All
   rights reserved.
   >
   ```

   注記　大なり記号 (>) は、tmadmin のプロンプトです。

2. printserver(psr) コマンドを入力して、サーバに関する情報を出力します。

```
> psr
a.out Name  Queue Name   Grp Name  ID  RqDone   Load Done  Current Service
----------  ----------   --------  --  ------   ---------  ---------------
BBL         531993       simple     0      0           0   (IDLE)
CSIMPSRV    00001.00001  GROUP1     1      0           0   (IDLE)
>
```

3. printservice(psc) コマンドを入力して、サービスに関する情報を出力します。

```
> psc
Service Name  Routine Name  a.out Name  Grp Name  ID  Machine #  Done  Status
------------  ------------  ----------  --------  --  -------    ----  -------
CSIMPSRV      CSIMPSRV      CSIMPSRV    GROUP1    1   simple       -   AVAIL
>
```

4. プロンプトで「q」と入力して、tmadmin を終了します。アプリケーションの起動とシャットダウンは、tmadmin から行うことができます。

関連項目

- 『BEA Tuxedo コマンド・リファレンス』の tmadmin(1)

ステップ 8: アプリケーションのシャットダウン

1. tmshutdown を実行して、アプリケーションをシャットダウンします。

```
$ tmshutdown
Shutdown all admin and server processes?(y/n): y
Shutting down all admin and server processes in /usr/me/CSIMPDIR/TUXCONFIG

Shutting down server processes ...

Server Id = 1 Group Id = GROUP1 Machine = simple:shutdown succeeded.

Shutting down admin processes ...

Server Id = 0 Group Id = simple Machine = simple:  shutdown succeeded.
2 processes stopped.
$
```

2. ULOG の内容を確認します。

```
$ cat ULOG*
$
140533.usltux!BBL.22964:LIBTUX_CAT:262: std main starting
140540.usltux!CSIMPSRV.22965:COBAPI_CAT:1067:INFO: std main starting
140542.usltux!CSIMPSRV.22965: TPSVRINIT :Welcome to the simple service
140610.usltux!?proc.22966: CSIMPCL:Started
140614.usltux!CSIMPSRV.22965: CSIMPSRV :Started
140614.usltux!CSIMPSRV.22965: CSIMPSRV :Success
140614.usltux!?proc.22966: switch to new log file
/home/usr_nm/CSIMPDIR/ULOG.112592
140614.usltux!?proc.22966:CSIMPCL:Ended
```

このセッションでは、ULOG の各行に意味があります。まず、ULOG 行の形式を確認してください。

```
time (hhmmss).machine_uname!process_name.process_id:log message
```

次に、実際のコード行を確認してください。

```
140542. Message from TPSVRINIT in CSIMPSRV
```

関連項目

- 『BEA Tuxedo コマンド・リファレンス』の tmshutdown(1)
- 『BEA Tuxedo COBOL リファレンス』の USERLOG(3cbl)

5 STOCKAPP (複雑な COBOL アプリケーション) のチュートリアル

ここでは、次の内容について説明します。

- STOCKAPP とは
- STOCKAPP について
- STOCKAPP のファイルおよびリソースの準備
  - ステップ 1: 環境変数の設定
  - ステップ 2: STOCKAPP でのサーバのビルド
  - ステップ 3: STOCKAPP.mk ファイルの編集
  - ステップ 4: コンフィギュレーション・ファイルの編集
  - ステップ 5: バイナリ形式のコンフィギュレーション・ファイルの作成
- STOCKAPP の実行

STOCKAPP とは

STOCKAPP は、BEA Tuxedo システム・ソフトウェアに同梱されている株式売買の ATMI サンプル・アプリケーションです。このアプリケーションでは、顧客のアカウント情報の検証と更新、株式や株式投資信託の売買注文の執行など、株式売買関連の操作を実行できます。

ここでは、STOCKAPP アプリケーションを開発する方法について、手順を順に示して説明します。このチュートリアルに従って STOCKAPP の「開発」を経験すると、独自のアプリケーションを開発できるようになります。

STOCKAPP チュートリアルは、次の 3 つの節から構成されています。

- 5-2 ページの「STOCKAPP について」
- 5-11 ページの「STOCKAPP のファイルおよびリソースの準備」
- 5-25 ページの「STOCKAPP の実行」

注記　ここで説明する内容は、アプリケーションの開発、管理、プログラミングに経験のあるシステム・ユーザを対象としています。また、BEA Tuxedo システム・ソフトウェアについて理解していることを前提としています。BEA Tuxedo アプリケーションをビルドするには、開発ラインセンスが必要です。

STOCKAPP について

ここでは、STOCKAPP アプリケーションを構成するファイル、クライアント、およびサービスについて説明します。各作業をクリックすると、その作業を行う手順が表示されます。

STOCKAPP のファイルについて

STOCKAPP アプリケーションを構成するファイルは、STOCKAPP ディレクトリに置かれています。このディレクトリは、次のように構成されています。

株式アプリケーション (STOCKAPP) のファイルについて

STOCKAPP ディレクトリには、次のファイルが置かれています。

- 8 つの .cbl ファイル
- 4 つのクライアント :BUY.cbl、SELL.cbl、FUNDPR.cbl、FUNDUP.cbl

- 1 つの会話型サーバ :FUNDUPSR.cbl
- サーバまたはサーバに対応付けられた 3 つのファイル
- アプリケーションのトランザクションまたはデータ生成を行う 2 つのサーバ
- STOCKAPP をサンプル・アプリケーションとして実行するためのファイル

次の表は、STOCKAPP を構成するファイルを示しています。この表には、BEA Tuxedo システム・ソフトウェアで提供されるソース・ファイル、株式アプリケーションのビルド時に生成されるファイル、および各ファイルの簡単な説明がまとめられています。

表 5-1 株式アプリケーションを構成するファイル

ソース・ファイル	生成されるファイル	フィールド値
BUY.cbl	BUY.o BUY	クライアント
BUYSR.cbl	BUYSR.o BUYSR	BUY サービス
ENVFILE		tmloadcf で使用される ENVFILE
FILES		STOCKAPP のすべてのファイルを説明したファイル・リスト
FUNDPR.cbl	FUNDPR.o FUNDPR	クライアント
FUNDPRSR.cbl	FUNDPRSR.o FUNDPRSR	PRICE QUOTE サービス
FUNDUP.cbl	FUNDUP.o FUNDUP	クライアント
FUNDUPSR.cbl	FUNDUPSR.o FUNDUPSR	FUND UPDATE サービス
README		インストールと起動についてのオンラインの説明

表 5-1 株式アプリケーションを構成するファイル (続き)

ソース・ファイル	生成されるファイル	フィールド値
SELL.cbl	SELL.o SELL	クライアント
SELLSR.cbl	SELLSR.o SELLSR	SELL サービス
STKVAR		変数の設定内容 (ENVFILE の変数を除く)
STOCKAPP.mk		アプリケーション makefile
UBBCBSHM	TUXCONFIG	SHM モードのコンフィギュレーションで使用する UBBCONFIG サンプル・ファイル
cust	CUST.cbl cust.V cust.h	BUY クライアントと SELL クライアント、および BUYSR サーバと SELLSR サーバでやり取りされる構造体を定義するための VIEW ファイル
quote	QUOTE.cbl quote.V quote.h	FUNDPR クライアントと FUNDUP クライアント、およびすべてのサーバでやり取りされる構造体を定義するための VIEW ファイル

関連項目

- 5-2 ページの「STOCKAPP について」

STOCKAPP クライアントの検証

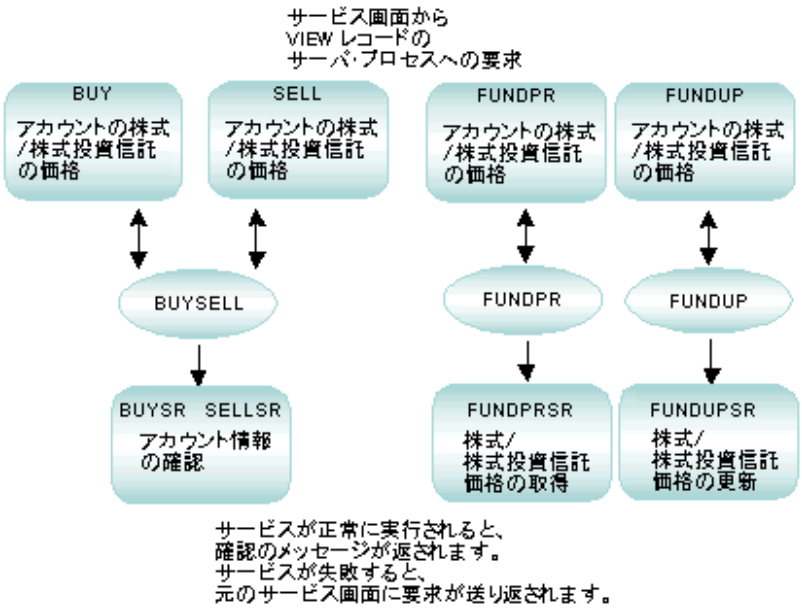
BEA Tuxedo システムの ATMI クライアント / サーバ・アーキテクチャでは、通信に次の 2 つのモードがあります。

- 要求 / 応答モード。サーバが実行する 1 つのサービス要求を送り、1 つの応答を受け取ります。
- 会話モード。クライアント (またはクライアントとして動作するサーバ) とサーバ間に専用の接続が確立されます。終了するまで、この接続はアクティブになったままです。接続がアクティブである限り、参加している 2 つのプロセス間で、サービスに対する要求と応答をそれぞれ含むメッセージをやり取りできます。

システム・クライアント・プログラム

次の図は、STOCKAPP の構成を示しています 4 つのサービス要求のいずれかを選択します。楕円は、アプリケーション・サービスを表しています。

図 5-1STOCKAPP の要求



型付きバッファ

型付きバッファは、BEA Tuxedo システムの基本部分です。BEA Tuxedo システムでは、型付きバッファに特定のデータ型が格納されます。定義されている型は、VIEW、STRING、CARRAY、X_OCTET、X_COMMON、および XML の 6 つです。アプリケーションで別の型を定義することもできます。

要求 / 応答クライアント :BUY.cbl

BUY は、クライアント・プログラムの 1 つです。このプログラムでは、アカウントの照会が作成されて、BUYSR サービスが呼び出されます。このプログラムは、次のように実行可能プログラムとして呼び出します。

```
BUY
```

BUY.cbl のソース・コード

BUY.cbl プログラムの次のセクションを確認します。

```
* Now register the client with the system
* Issue a TPCALL
* Clean up
```

これらのセクションは、BUY.cbl で BEA Tuxedo ATMI 関数が使用される場所です。csimpl.cbl と同様に、BUY.cbl では、TPINITIALIZE を呼び出してアプリケーションに参加し、TPCALL を呼び出してサービスの RPC 要求を行い、TPTERM を呼び出してアプリケーションを終了します。また、BUY.cbl では、cust ファイルで定義される構造体と VIEW 型のレコードが使用されます。構造体のソース・コードは、VIEW 記述ファイル cust.V に記述されています。

クライアントのビルド

cust などの VIEW 記述ファイルは、VIEW コンパイラ viewc(1) でコンパイルします。次に示すように、view(c) を実行して、コンパイルします。

```
viewc-C-n

  cust.v
```

viewc には、COBOL ファイル (CUST.cbl)、バイナリ形式の VIEW 記述ファイル (cust.V)、およびヘッダ・ファイル (cust.h) の 3 つの出力ファイルがあります。

クライアント・プログラムの BUY.cbl、FUNDPR.cbl、FUNDUP.cbl、および SELL.cbl は、buildclient(1) でコンパイルされ、必要な BEA Tuxedo ライブラリとリンクされます。

必要に応じて、これらのコマンドは個別に使用できます。その手順に関する規則は、STOCKAPP.mk に定義されています。

関連項目

- 『BEA Tuxedo システム入門』の 2-5 ページの「ATMI の使用」
- 『BEA Tuxedo システム入門』の 2-27 ページの「型付きバッファ」
- 『BEA Tuxedo コマンド・リファレンス』および『BEA Tuxedo C リファレンス』の ATMI コマンドおよび関数
- 5-2 ページの「STOCKAPP について」

STOCKAPP サーバの検証

ここでは、次の内容について説明します。

- 株式アプリケーションを構成するサービス
- STOCKAPP サービスおよびサーバの関係
- 各サーバをコンパイルしてビルドするための buildserver コマンドのオプション

ATMI サーバとは、1 つ以上のサービスを提供する実行可能プロセスです。BEA Tuxedo システムでは、サーバはクライアントとして動作するプロセスから継続的に要求を受け取り、それを適切なサービスに転送します。サービスは、アプリケーション用に記述された COBOL 言語コードのサブルーチンです。BEA Tuxedo システムのトランザクション処理アプリケーションの機能性は、これらのサービスがリソース・マネージャにアクセスすることに

よって実現されます。サービス・ルーチンは、BEA Tuxedo システム・プログラマによって作成されなければならないアプリケーションの一部です。ユーザ定義のクライアントは、アプリケーションの別の部分に相当します。

すべての STOCKAPP サービスでは、ATMI (Application to Transaction Monitor Interface) で提供される関数が使用されて、次のタスクが行われます。

- ほかのサービスとの同期通信または非同期通信
- グローバル・トランザクションの定義
- クライアントへの応答

STOCKAPP サービス

STOCKAPP には、4 つのサービスがあります。STOCKAPP の各サービスは、次に示すように、サーバのソース・コードにある COBOL 関数名と一致します。

BUYSR
: 株式 / 株式投資信託レコードを買います。BUYSELL サーバで提供されます。入力として VIEW レコードを受け取ります。CUSTFILE レコードを挿入します。

SELLSR
: 株式 / 株式投資信託レコードを売ります。BUYSELL サーバで提供されます。入力として VIEW レコードを受け取ります。CUSTFILE レコードを挿入します。

FUNDPRSR
: 株式の値を照会します。PRICEQUOTE サーバで提供されます。入力として VIEW レコードを受け取ります。

FUNDUPSR
: 株式投資信託を更新します。会話型のサービスです。FUNDUPDATE サーバで提供されます。入力として VIEW レコードを受け取ります。

STOCKAPP のファイルおよびリソースの準備

ここでは、STOCKAPP を実行するために必要なファイルやリソースを作成するための手順を順に示します。

各作業をクリックすると、その作業を行う手順が表示されます。

ステップ 1: 環境変数の設定

STOCKAPP に必要な環境変数は、STKVAR ファイルに定義されています。このファイルは数多くのコメントが記述された大きな (約 100 行から構成される) ファイルです。

1. テキスト・エディタで、STKVAR ファイルの内容を確認します。9 行目のコードで、TUXDIR が設定されていることがわかります。設定されていない場合、ファイルの実行が失敗して、次のメッセージが表示されます。

   ```
   TUXDIR:parameter null or not set
   ```

2. TUXDIR パラメータに BEA Tuxedo システムのディレクトリ構造でのルート・ディレクトリを設定し、エクスポートします。

3. STKVAR の別のコード行で APPDIR にディレクトリ {TUXDIR}/samples/atmi/STOCKAPP が設定されています。これは、STOCKAPP のソース・ファイルが置かれたディレクトリです。APPDIR は、BEA Tuxedo システムによって、アプリケーション固有のファイルが検索されるディレクトリです。オリジナルのソース・ファイルを上書きしないように、STOCKAPP ファイルを別のディレクトリにコピーします。その場合、そのディレクトリを APPDIR に指定します。TUXDIR の下位ディレクトリである必要はありません。

   注記　STKVAR で指定されるほかの変数は、サンプル・アプリケーションで各種の働きをします。独自のアプリケーションを開発する場合は、それらの働きについて認識しておくことが必要です。STKVAR には各種の変数が定義されているので、後で実際のアプリケーションのテンプレートとして使用できます。

4. STKVAR に必要な変更を加えたら、次のように STKVAR を実行します。

   ```
   . ./STKVAR
   ```

コード リスト 5-1　STKVAR:STOCKAPP **の環境変数**

```
#ident    "@(#)samples/atmi:STOCKAPP/STKVAR
#
# このファイルには、bankapp を実行するために
# Tuxedo システムで必要なすべての環境変数が設定されています。
#
# このディレクトリには、すべての Tuxedo ソフトウェアが置かれています。
# システム管理者はこの変数を設定する必要があります。
#
TUXDIR=${TUXDIR:?}
#
# このディレクトリには、ユーザ作成の全コードが置かれています。
#
```

```
# アプリケーション・ジェネレータによって生成されたファイルが
# 置かれるディレクトリの絶対パス名です。
#
APPDIR=${HOME}/STOCKAPP
#
# tmloadcf で使用される環境ファイルです。
#
COBDIR=${COBDIR:?}
#
# このディレクトリには、コンパイルとリンクに必要な
# COBOL ファイルが置かれています。
#
LD_LIBRARY_PATH=$COBDIR/coblib:${LD_LIBRARY_PATH}
#
# coblib を LD_LIBRARY_PATH に追加します。
#
ENVFILE=${APPDIR}/ENVFILE
#
# CBLVIEWC、tmloadcf などで使用されるフィールド・テーブル・ファイルのリストです。
#
FIELDTBLS=fields,Usysflds
#
# フィールド・テーブル・ファイルを検索するディレクトリのリストです。
#
FLDTBLDIR=${TUXDIR}/udataobj:${APPDIR}
#
# トランザクション・ログのデバイスを設定します。これは、
# UBBCBSHM ファイルの *MACHINES セクションにあるこのサイトの LMID の
# TLOGDEVICE パラメータと一致する必要があります。
#
TLOGDEVICE=${APPDIR}/TLOG
#
# コンフィギュレーション・ファイルのデバイスです。
#
UBBCBSHM=$APPDIR/UBBCBSHM
#
# /T にすべての情報を提供するバイナリ・ファイルのデバイスです。
#
TUXCONFIG=${APPDIR}/TUXCONFIG
#
# 中央ユーザ・ログを記録するファイルの接頭語を設定します。
# これは、UBBCONFIG ファイルの *MACHINES セクションにある
# このサイトの LMID の ULOGPFX パラメータと一致する必要があります。
#
ULOGPFX=${APPDIR}/ULOG
#
# VIEW ファイルを検索するディレクトリのリストです。
```

```
#
VIEWDIR=${APPDIR}
#
# CBLVIEWC、tmloadcf などで使用される VIEW ファイルのリストです。
#
VIEWFILES=quote.V,cust.V
#
# COBCPY を設定します。
#
COBCPY=$TUXDIR/cobinclude


#
# COBOPT を設定します。
#
COBOPT="-C ANS85 -C ALIGN=8 -C NOIBMCOMP -C TRUNC=ANSI -C OSEXT=cbl"
#
# CFLAGS を設定します。
#
CFLAGS="-I$TUXDIR/include -I$TUXDIR/sysinclude"
#
# 設定したすべての変数をエクスポートします。
#
export TUXDIR APPDIR ENVFILE
export FIELDTBLS FLDTBLDIR TLOGDEVICE
export UBBCBSHM TUXCONFIG ULOGPFX LD_LIBRARY_PATH
export VIEWDIR VIEWFILES COBDIR COBCPY COBOPT CFLAGS
#
# 設定されていない場合は、TUXDIR/bin を PATH に追加します。
#
a="`echo $PATH | grep ${TUXDIR}/bin`"
if [ x"$a" = x ]
then
PATH=${TUXDIR}/bin:${PATH}
export PATH
fi
#
# 設定されていない場合は、APPDIR を PATH に追加します。
#
a="`echo $PATH | grep ${APPDIR}`"
if [ x"$a" = x ]
then
PATH=${PATH}:${APPDIR}
export PATH
fi
#
# 設定されていない場合は、COBDIR を PATH に追加します。
#
```

```
a="`echo $PATH | grep ${COBDIR}`"
if [ x"$a" = x ]
then
PATH=${PATH}:${COBDIR}
export PATH
fi
```

そのほかの要件

- AIX の場合、LD_LIBRARY_PATH ではなく LIBPATH を設定します。
- HP-UX の場合、LD_LIBRARY_PATH ではなく SHLIB_PATH を設定します。
- オペレーティング・システムが Sun Solaris の場合、PATH の先頭で /usr/5bin を指定する必要があります。次のコマンドを使用します。

  ```
  PATH=/usr/5bin:$PATH;export PATH
  ```

  シェルとしては、csh ではなく /bin/sh を使用します。

関連項目

- 5-11 ページの「STOCKAPP のファイルおよびリソースの準備」

ステップ 2: STOCKAPP でのサーバのビルド

実行可能な ATMI サーバをビルドするには、buildserver を使用します。このコマンドではオプションを使用して、出力ファイル、アプリケーションで提供される入力ファイル、各種の方法で BEA Tuxedo システム・アプリケーションを実行するためのライブラリを指定します。

-C オプションを指定して buildserver を実行すると、cobcc コマンドが呼び出されます。環境変数の ALTCC を設定すると別のコンパイル・コマンドを指定でき、ALTCFLAGS を設定するとコンパイル時と編集時にフラグを設定できます。buildserver のコマンド行で指定できる主なオプションを以下の例に示します。

buildserver コマンドは STOCKAPP.mk で使用され、株式アプリケーションの各サーバをコンパイルしてビルドします。詳細については、『BEA Tuxedo コマンド・リファレンス』の buildserver(1) を参照してください。

BUYSELL サーバのビルド

BUYSELL ATMI サーバは、BUYSR および SELLSR 関数のコードが記述されたファイルから作成されています。BUYSELL サーバは、まず BUYSELL.o ファイルにコンパイルされ、次に buildserver コマンドに渡されます。そのため、コンパイル・エラーを特定でき、サーバをビルドする前に修正することができます。

1. BUYSELL.o ファイルを作成します (STOCKAPP.mk で作成されます)。次のように、buildserver コマンドで BUYSELL サーバをビルドします。

```
buildserver -C -v -o BUYSELL -s SELLSR -f SELLSR.cbl -s BUYSR -f BUYSR.cbl
```

以下は、コマンド行で指定されている各オプションの説明です。

- -C オプションは、COBOL モジュールでサーバをビルドする場合に指定します。
- -v オプションは、冗長モードを使用する場合に指定します。その標準出力に cc コマンドが出力されます。
- -o オプションは、実行可能出力ファイルに名前を指定する場合に使用します。名前が指定されていない場合は、SERVER という名前が付きます。
- -s オプションは、サーバの起動時に宣言されるサーバのサービス名を指定します。サービスを実行する関数名がサービス名と異なる場合、関数名が -s オプションの引数の一部になります。STOCKAPP では、関数名はサービス名と同じなので、サービス名だけを指定しま

す。サービス名では、すべての文字列を大文字で指定します。ただし、buildserver の -s オプションでは、サーバ内のサービスを処理する関数には任意の名前を付けることができます。詳細については、『BEA Tuxedo コマンド・リファレンス』の buildserver(1) を参照してください。システム管理者は、buildserver コマンドでサーバを作成した際に使用されたサービスのサブセットだけをサーバの起動時に利用できるように設定することもできます。詳細については、『BEA Tuxedo アプリケーション実行時の管理』、および『BEA Tuxedo アプリケーションの設定』を参照してください。

- -f オプションは、リンク時と編集時に使用されるファイルを指定します。buildserver リファレンス・ページの -l オプションも参照してください。この 2 つのオプションの詳細については、COBOL を使用した BEA Tuxedo アプリケーションのプログラミングの 5-37 ページの「サーバのビルド」を参照してください。ファイルがリストされる順序には意味があります。この順序は、関数の参照、およびその参照がどのライブラリで解決されるかによって決定されます。ソース・モジュールは、関数の参照が解決されるライブラリの前にリストされます。.cbl ファイルが存在する場合、それが最初にコンパイルされます。オブジェクト・ファイルは、別個の .o ファイル、またはアーカイブ (.a) ファイルにあるファイル・グループです。-f の引数として 1 つ以上のファイル名を指定する場合は、二重引用符で各ファイル名を囲む必要があります。-f オプションは、必要な数だけ使用できます。
- -s オプションは、BUYSELL サーバを構成するサービスとして、SELLSR および BUYSR を指定します。-o オプションは実行可能出力ファイルに BUYSELL という名前を付け、-f オプションは SELLSR.cbl および BUYSR.cbl ファイルがビルドでのリンク時と編集時に使用されることを指定します。

STOCKAPP.mk でビルドされるサーバ

STOCKAPP サーバをビルドする場合、buildserver コマンドの指定方法を理解していることが大切です。ただし、実際にビルドする場合、makefile にビルドの定義を記述することがよくあります。STOCKAPP でもその方法が採用されています。

関連項目

- 5-2 ページの「STOCKAPP について」
- buildserver(1)

ステップ 3: STOCKAPP.mk ファイルの編集

STOCKAPP には、すべてのスクリプトを実行可能にし、VIEW 記述ファイルをバイナリ形式に変換し、アプリケーション・サーバの作成に必要なすべてのプリコンパイル、コンパイル、およびビルドを行う makefile が提供されています。また、最初からやり直す場合にもこのファイルを利用できます。

提供されている STOCKAPP.mk をそのまま使わずにフィールドを編集した方がよい場合があります。また、説明が必要なフィールドもあります。以下にそれらのフィールドについて説明します。

TUXDIR パラメータの編集

STOCKAPP.mk にある次のコメントと TUXDIR パラメータを確認します。

```
#
    # Tuxedo システムのルート・ディレクトリです。このファイルは、編集してこの値を正しく設定するか、
# または "make -f STOCKAPP.mk TUXDIR=/correct/rootdir" を使用して正しい値を渡すことが必要となります。
# これを行わないと、STOCKAPP のビルドは失敗します。
#
TUXDIR=../..
```

TUXDIR パラメータには、BEA Tuxedo システムがインストールされたルート・ディレクトリの絶対パス名を指定します。

APPDIR パラメータの編集

APPDIR パラメータに設定されている値を変更する場合があります。提供された STOCKAPP では、APPDIR には STOCKAPP ファイルが置かれたディレクトリ (TUXDIR の相対パス) が指定されています。次に示す STOCKAPP.mk のセクションには、APPDIR の設定についての説明と定義が記述されています。

```
#
# STOCKAPP アプリケーションのソース・コードと実行可能ファイルが置かれたディレクトリです。
    # このファイルは、編集してこの値を正しく設定するか、
# または "make -f STOCKAPP.mk APPDIR=/correct/appdir" を使用して正しい値を渡すことが必要となります。
# これを行わないと、STOCKAPP のビルドは失敗します。
#
APPDIR=$(TUXDIR)/samples/atmi/STOCKAPP
#
```

README ファイルに従って、別のディレクトリにファイルをコピーした場合、APPDIR にはファイルのコピー先のディレクトリを指定します。makefile を実行すると、そのディレクトリにアプリケーションがビルドされます。

STOCKAPP.mk ファイルの実行

1. STOCKAPP.mk に必要な変更を加えたら、次のコマンドを使用してこのファイルを実行します。

   ```
   nohup make -f STOCKAPP.mk install &
   ```

2. nohup.out ファイルを調べて、処理が正しく行われたことを確認します。

関連項目

- 5-11 ページの「STOCKAPP のファイルおよびリソースの準備」

ステップ 4: コンフィギュレーション・ファイルの編集

STOCKAPP コンフィギュレーション・ファイルには、複数のコンピュータ上でアプリケーションを実行する方法が定義されています。STOCKAPP には、UBBCONFIG(5) で説明されているテキスト形式のコンフィギュレーション・ファイルが提供されています。UBBCBSHM には、単一のコンピュータ上のアプリケーションが定義されています。

1. テキスト・エディタで、STOCKAPP のコンフィギュレーション・ファイルの内容を確認します。

コード リスト 5-2 **値を変更する** UBBCBSHM **コンフィギュレーション・ファイルのフィールド**

```
      #Copyright (c) 1992 Unix System Laboratories, Inc.
      #All rights reserved
      #Tuxedo COBOL サンプル・アプリケーション用の UBBCONFIG スケルトン・ファイルです。
      *RESOURCES
      IPCKEY                  5226164
      DOMAINID            STOCKAPP
001   UID                 <id(1) からのユーザ ID>
002   GID                 <id(1) からのグループ ID>
      MASTER                  SITE1
      PERM                0660
      MAXACCESSERS            20
      MAXSERVERS          15
      MAXSERVICES         30
      MODEL               SHM
      LDBAL               Y
      MAXGTT                  100
      MAXBUFTYPE          16
      MAXBUFSTYPE         32
      SCANUNIT                10
      SANITYSCAN          12
      DBBLWAIT                6
      BBLQUERY                180
      BLOCKTIME           10
      TAGENT                  “TAGENT"
```

```
      #
      *MACHINES
003   <SITE1 の名前 >     LMID=SITE1
004                        TUXDIR="<TUXDIR1>"
005                        APPDIR="<APPDIR1>"
                           ENVFILE="<APPDIR1>/ENVFILE"
                           TUXCONFIG="<APPDIR1>/TUXCONFIG"
                           TUXOFFSET=0
006                        TYPE="< マシン・タイプ >"
                           ULOGPFX="<APPDIR>/ULOG"
                           MAXWSCLIENTS=5
      #
      *GROUPS
      COBAPI               LMID=SITE1         GRPNO=1
      #
      #
      *SERVERS
      FUNDUPSR SRVGRP=COBAPI      SRVID=1 CONV=Y ENVFILE="<APPDIR1>/ENVFILE"
      FUNDPRSR SRVGRP=COBAPI      SRVID=2 ENVFILE="<APPDIR1>/ENVFILE"
      BUYSELL  SRVGRP=COBAPI      SRVID=3 ENVFILE="<APPDIR1>/ENVFILE"
      #
      #
      *SERVICES
```

2. アプリケーションのパスワード機能を有効にするために、次の行を UBBCBSHM の RESOURCES セクションに追加します。

   ```
   SECURITY       APP_PW
   ```

3. 一部のパラメータ値は山かっこ (<>) で囲まれています。山かっこで囲まれた文字列は、実際のインストールに合わせた値に置き換えます。これらのフィールドは、RESOURCES、MACHINES、および GROUPS セクションにあります。次の表は、山かっこで囲まれた文字列を置き換える値について説明しています。各文字列を適切な値に置き換えます。

表 5-2 値の説明

行番号	置き換える文字列	目的
001	UID	掲示板の IPC 構造体の所有者を示す有効なユーザ ID。マルチプロセッサ・コンフィギュレーションでは、この値はすべてのマシンで同じにする必要があります。この値を BEA Tuxedo ソフトウェアの所有者と同じにすると、簡単に統一できます。
002	GID	掲示板の IPC 構造体の所有者を示す有効なグループ ID。マルチプロセッサ・コンフィギュレーションでは、この値はすべてのマシンで同じにする必要があります。アプリケーションの各ユーザは、このグループ ID を共有する必要があります。
003	SITE1 の名前	マシンのノード名。UNIX コマンド uname -n で出力される値を使用します。
004	TUXDIR	BEA Tuxedo システム・ソフトウェアのルート・ディレクトリの絶対パス名。ファイルに出現するすべての <TUXDIR1> に値を設定して、ファイル全体で値を置き換えます。
005	APPDIR	アプリケーションを実行するディレクトリの絶対パス名。ファイルに出現するすべての <APPDIR1> に値を設定して、ファイル全体で値を置き換えます。
006	*マシン・タイプ*	このパラメータは、異なる種類のマシンが存在するネットワーク環境のアプリケーションで重要です。BEA Tuxedo システムでは、マシン間で行われるすべての通信に対してこの値が確認されます。値が異なる場合にだけ encode/decode ルーチンが呼び出されてデータを変換します。

関連項目

- 5-11 ページの「STOCKAPP のファイルおよびリソースの準備」
- 『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5)

ステップ 5: バイナリ形式のコンフィギュレーション・ファイルの作成

バイナリ形式のコンフィギュレーション・ファイルを作成する前に

バイナリ形式のコンフィギュレーション・ファイルを作成する場合、STOCKAPP ファイルが置かれたディレクトリに移動し、環境変数を設定することが必要です。その場合、次の手順に従います。

1. STOCKAPP ファイルが置かれたディレクトリに移動します。
2. 次のコマンドを入力して、環境変数を設定します。

```
. ./STKVAR
```

コンフィギュレーション・ファイルのロード

コンフィギュレーション・ファイルを編集したら、それを MASTER マシン上にバイナリ・ファイルとしてロードする必要があります。バイナリ形式のコンフィギュレーション・ファイルの名前は TUXCONFIG、そのパス名は TUXCONFIG に定義されています。このファイルは、BEA Tuxedo のシステム管理者の有効なユーザ ID およびグループ ID を持つユーザが作成します。こ

の 2 つの ID は、ご使用のコンフィギュレーション・ファイルの UID および GID の値と同じであることが必要です。同じではない場合、STOCKAPP の実行時にパーミッションの問題が発生します。

1. 次のコマンドを入力して、TUXCONFIG を作成します。

   ```
   tmloadcf UBBCBSHM
   ```

   コンフィギュレーションのロード時には、このコンフィギュレーションをインストールするかどうか、およびインストールする場合は既存のコンフィギュレーション・ファイルを上書きすることを確認するメッセージが何回か表示されます。このような確認を省くには、コマンド行で -y オプションを指定します。

2. アプリケーションで必要な IPC 資源を BEA Tuxedo システムで計算する場合は、コマンド行で -c オプションを指定します。

   TUXCONFIG は、MASTER マシン上だけにインストールできます。アプリケーションの起動時に tmboot によってほかのマシンに複製転送されます。

   コンフィギュレーションのオプションとして SECURITY が指定されている場合、tmloadcf の実行時にアプリケーション・パスワードの入力が求められます。30 文字までのパスワードを指定できます。アプリケーションに参加するクライアント・プロセスでは、パスワードを入力する必要があります。

   ロードする前に tmloadcf によってテキスト形式のコンフィギュレーション・ファイル (UBBCONFIG) が解析されます。構文エラーが検出された場合、ファイルのロードは失敗します。

関連項目

- 5-11 ページの「STOCKAPP のファイルおよびリソースの準備」
- 『BEA Tuxedo コマンド・リファレンス』の tmloadcf(1)

STOCKAPP の実行

ここでは、STOCKAPP を起動し、各種のクライアント・プログラムとトランザクションを行ってテストし、終了する手順について順に説明します。

各作業をクリックすると、その作業を行う手順が表示されます。

ステップ 1: 起動する前に行う準備作業

1. STOCKAPP を起動する前に、アプリケーションをサポートできるだけの IPC 資源がマシンにあることを確認します。IPC 資源に関するレポートを出力するには、tmboot コマンドに -c オプションを指定します。

コード リスト 5-3 IPC **レポート**

```
Ipc sizing (minimum /T values only)
                Fixed Minimums Per Processor
SHMMIN:1
SHMALL:1
SEMMAP: SEMMNI
                Variable Minimums Per Processor
                SEMUME,           A                              SHMMAX
                SEMMNU,           *                                *
Node            SEMMNS  SEMMSL  SEMMSL  SEMMNI  MSGMNI  MSGMAP  SHMSEG
------          ------  ------  ------  ------  ------  ------  ------
machine 1               60       1      60   A + 1      10      20     76K
machine 2               63       5      63   A + 1      11      22     76K
where 1 <= A <= 8.
```

2. プロセッサごとに使用されるアプリケーション・クライアントの数を各 MSGMNI 値に追加します。MSGMAP は MSGMNI の 2 倍にします。
3. IPC の最低条件とご使用のマシンに対して設定されたパラメータとを比較します。これらのパラメータを定義する場所は、プラットフォームによって異なります。
   - ほとんどの UNIX システム・プラットフォームの場合、マシンのパラメータは /etc/conf/cf.d/mtune に定義されています。
   - Windows 2000 プラットフォームの場合、マシンのパラメータはコントロール・パネルで設定したり表示します。

関連項目

- 5-25 ページの「STOCKAPP の実行」

ステップ 2: STOCKAPP の起動

1. 環境変数を設定します。

   ```
   ../STKVAR
   ```

2. 次のコマンドを入力して、アプリケーションを起動します。

   ```
   tmboot
   ```

   次のプロンプトが表示されます。

   ```
   Boot all admin and server processes?(y/n):y
   ```

   プロンプトで「y」と入力すると、次のようなレポートが画面に出力されます。

   ```
   Booting all admin and server processes in
   /usr/me/appdir/tuxconfig
   Booting all admin processes
   exec BBL -A:
              process id=24223 Started.
   ```

このレポートは、コンフィギュレーション内のすべてのサーバが起動するまで出力されます。起動したサーバの合計数が出力された時点でレポートは終了します。

必要に応じて、コンフィギュレーションの一部のサーバだけを起動することもできます。たとえば、管理サーバだけを起動するには、-A オプションを指定します。オプションが指定されていない場合は、アプリケーション全体が起動します。

tmboot では、起動したサーバ数がレポートされるほかに、ULOG にメッセージが送信されます。

関連項目

- 5-25 ページの「STOCKAPP の実行」
- 『BEA Tuxedo コマンド・リファレンス』の tmboot(1)

- 『BEA Tuxedo COBOL リファレンス』の USERLOG(3cbl)

ステップ 3: STOCKAPP サービスのテスト

1. 実行中のシステムにログインする場合は、STOCKAPP に環境変数を設定する必要があります。その場合、次のコマンドを入力します。

   ```
   ../STKVAR
   ```

2. BUY クライアント・プログラムを実行します。その場合、次のコマンドを入力します。

   ```
   BUY
   ```

3. STOCKAPP を監視します。STOCKAPP の実行中に、tmadmin のサブコマンドなど各種のコマンドを実行し、出力される状態情報の種類を確認します。

関連項目

- 5-25 ページの「STOCKAPP の実行」

ステップ 4: STOCKAPP のシャットダウン

STOCKAPP を終了するには、次のように、引数を指定せずに tmshutdown(1) コマンドを MASTER マシンで入力します。

```
tmshutdown
```

このコマンド (または tmadmin のシャットダウン・コマンド) を実行すると、次のタスクが行われます。

- すべてのアプリケーション・サーバ、ゲートウェイ・サーバ、TMS サーバ、および管理サーバがシャットダウンします。
- 対応付けられたすべての IPC 資源が削除されます。

関連項目

- 5-25 ページの「STOCKAPP の実行」
- 『BEA Tuxedo コマンド・リファレンス』の tmadmin(1)
- 『BEA Tuxedo コマンド・リファレンス』の tmshutdown(1)

6 XMLSTOCKAPP のチュートリアル

ここでは、次の内容について説明します。

- XMLSTOCKAPP とは
- XMLSTOCKAPP について
- XMLSTOCKAPP のファイルおよびリソースの準備
  - ステップ 1: 新しいディレクトリへの XMLSTOCKAPP ファイルのコピー
  - ステップ 2: 環境変数の設定
  - ステップ 3: クライアントのビルド
  - ステップ 4: XMLSTOCKAPP でのサーバのビルド
  - ステップ 5: コンフィギュレーション・ファイルの編集
  - ステップ 6: バイナリ形式のコンフィギュレーション・ファイルの作成
- XMLSTOCKAPP の実行

XMLSTOCKAPP とは

XMLSTOCKAPP は、BEA Tuxedo システム・ソフトウェアに同梱されている株式売買の ATMI サンプル・アプリケーションです。このアプリケーションでは、単一のマシン上にある 2 つのサーバを実行し、C および C++ Tuxedo サーバからのパーサの呼び出しと XML バッファのルーティングを示します。片方のサーバは C++ で記述された Tuxedo サーバです (stockxml)。もう一方のサーバは C で記述されています (stockxml_c)。2 つのサーバは同じ STOCKQUOTE サービスを提供します。クライアントはサービスを呼び出し、株価を返します。次に XML バッファを出力します。

ここでは、XMLSTOCKAPP アプリケーションを開発する方法について、手順を順に示して説明します。このチュートリアルに従って XMLSTOCKAPP の「開発」を経験すると、独自のアプリケーションを開発できるようになります。

XMLSTOCKAPP チュートリアルは、次の 3 つの節から構成されています。

- 6-2 ページの「XMLSTOCKAPP について」
- 6-5 ページの「XMLSTOCKAPP のファイルおよびリソースの準備」
- 6-13 ページの「XMLSTOCKAPP の実行」

注記　ここで説明する内容は、アプリケーションの開発、管理、プログラミングに経験のあるシステム・ユーザを対象としています。また、BEA Tuxedo システム・ソフトウェアについて理解していることを前提としています。BEA Tuxedo アプリケーションをビルドするには、開発ラインセンスが必要です。

XMLSTOCKAPP について

ここでは、XMLSTOCKAPP アプリケーションを構成するファイル、クライアント、およびサービスについて説明します。それぞれの詳細については以下を参照してください。

XMLSTOCKAPP のファイルについて

XMLSTOCKAPP アプリケーションを構成するファイルは、samples/atmi/xmlstockapp ディレクトリに置かれています。このサンプルに用意されているファイルは、次のとおりです。

XMLSTOCKAPP ディレクトリには、次のファイルが置かれています。

- クライアントへの 2 つの .xml 入力ファイル : stock_quote_beas.xml および stock_quote_msft.xml
- 1 つのクライアント : Client.cpp
- 2 つのサーバ・ファイル : stockxml および stockxml_c
- STOCKAPP をサンプル・アプリケーションとして実行するためのファイル
  - SAXPrint.cpp
  - SAXPrintHandler.cpp
  - DOMTreeErrorReporter.cpp
  - xmlWrapper.cpp

XMLSTOCKAPP クライアントの検証

BEA Tuxedo システムの ATMI クライアント / サーバ・アーキテクチャでは、通信に次の 2 つのモードがあります。

- 要求 / 応答モード。サーバが実行する 1 つのサービス要求を送り、1 つの応答を受け取ります。

- 会話モード。クライアント (またはクライアントとして動作するサーバ) とサーバ間に専用の接続が確立されます。終了するまで、この接続はアクティブになったままです。接続がアクティブである限り、参加している 2 つのプロセス間で、サービスに対する要求と応答をそれぞれ含むメッセージをやり取りできます。

XMLSTOCKAPP では要求 / 応答モードをインプリメントし、STOCKQUOTE サービスを使用して株価を要求します。

1. BEAS または MSFT の株価の要求。
2. XML ファイル内の単一の引数で実行されるクライアントが、STOCKQUOTE サービスを呼び出します。
3. サービスは XML バッファを株価で更新します。
4. クライアントは XML バッファを出力します。

要求 / 応答クライアント : stock_quote_beas.xml

Client.cpp は XML ファイル stock_quote_beas.xml または stock_quote_msft.xml からの入力を使用するクライアント・プログラムです。STOCKQUOTE サービスを呼び出す照会を行い、BEAS または MSFT の株価を返します。このプログラムは、次のように実行可能プログラムとして呼び出します。

```
Client stock_quote_beas.xml
```

または

```
Client stock_quote_msft.xml
```

関連項目

- 『BEA Tuxedo システム入門』の 2-5 ページの「ATMI の使用」
- 『BEA Tuxedo システム入門』の 2-27 ページの「型付きバッファ」
- 『BEA Tuxedo コマンド・リファレンス』および『BEA Tuxedo C リファレンス』の ATMI コマンドおよび関数

XMLSTOCKAPP サーバの検証

ATMI サーバとは、1 つ以上のサービスを提供する実行可能プロセスです。BEA Tuxedo システムでは、サーバはクライアントとして動作するプロセスから継続的に要求を受け取り、それを適切なサービスに転送します。BEA Tuxedo システムのトランザクション処理アプリケーションの機能性は、これらのサービスがリソース・マネージャにアクセスすることによって実現されます。サービス・ルーチンは、BEA Tuxedo システム・プログラマによって作成されなければならないアプリケーションの一部です。ユーザ定義のクライアントは、アプリケーションの別の部分に相当します。

XMLSTOCKAPP プログラムの STOCKQUOTE サービスは、アプリケーション・トランザクション・モニタ・インターフェイス (ATMI) で提供される機能を使用して、クライアントに XML バッファとして株価を返します。

XMLSTOCKAPP のファイルおよびリソースの準備

ここでは、XMLSTOCKAPP を実行するために必要なファイルやリソースを作成するための手順を順に示します。

- ステップ 1: 新しいディレクトリへの XMLSTOCKAPP ファイルのコピー
- ステップ 2: 環境変数の設定
- ステップ 3: クライアントのビルド
- ステップ 4: XMLSTOCKAPP でのサーバのビルド
- ステップ 5: コンフィギュレーション・ファイルの編集
- ステップ 6: バイナリ形式のコンフィギュレーション・ファイルの作成

ステップ 1: 新しいディレクトリへの XMLSTOCKAPP ファイルのコピー

ファイルを編集したりサンプルを実行したりする前に、XMLSTOCKAPP ファイルを独自のディレクトリにコピーしておくことをお勧めします。

ステップ 2: 環境変数の設定

環境変数ファイルを編集する必要があります。

1. TUXDIR が設定されていることを確認します。設定されていない場合、ファイルの実行が失敗して、次のメッセージが表示されます。

   ```
   TUXDIR:parameter null or not set
   ```

2. TUXDIR パラメータに BEA Tuxedo システムのディレクトリ構造でのルート・ディレクトリを設定し、エクスポートします。

3. APPDIR を {TUXDIR}/samples/atmi/XMLSTOCKAPP ディレクトリに設定します。これは、XMLSTOCKAPP のソース・ファイルが置かれたディレクトリです。APPDIR は、BEA Tuxedo システムによって、アプリケーション固有のファイルが検索されるディレクトリです。オリジナルのソース・ファイルを上書きしないように XMLSTOCKAPP ファイルを別のディレクトリにコピーした場合は、そのディレクトリを入力します。TUXDIR の下位ディレクトリである必要はありません。

4. 環境変数ファイルに必要な変更を加えたら、次のように実行します。

   ```
   . ./<VARFILE>
   ```

   <VARFILE> は環境変数ファイルの名前です。

そのほかの要件

HP-UX と AIX の場合を除いて、LD_LIBRARY_PATH には、共用ライブラリを使用するシステムの $TUXDIR/lib が含まれている必要があります。

- AIX の場合、LD_LIBRARY_PATH ではなく LIBPATH を設定します。

- HP-UX の場合、LD_LIBRARY_PATH ではなく SHLIB_PATH を設定します。
- オペレーティング・システムが Sun Solaris の場合、PATH の先頭で /usr/5bin を指定する必要があります。次のコマンドを使用します。

  ```
  PATH=/usr/5bin:$PATH;export PATH
  ```

  シェルとしては、csh ではなく /bin/sh を使用します。

ステップ 3: クライアントのビルド

クライアントをビルドするには、以下のコマンドを使用します。

```
export CFLAGS=-I
```

特定のオペレーティング・システムの場合は、以下のコマンドを使用します。

- Solaris の場合

  ```
  export CC=CC
  ```

- HP-UX の場合

  ```
  export CC=aCC
  ```

- Digital Unix の場合

  ```
  export CC=cxx
  ```

- AIX の場合

  ```
  export CC=xlC_r
  ```

- Linux の場合

  ```
  export CC=g++
  ```

クライアントをビルドするには、以下のコマンドを使用します。

```
buildclient -o Client -f Client.cpp -f SAXPrint.cpp -f
SAXPrintHandlers.cpp -f -ltxml
```

ステップ 4: XMLSTOCKAPP でのサーバのビルド

XMLSTOCKAPP サンプルには 2 つのサーバが用意されています。ただし、このサンプルのサーバをビルドするには、README ファイルの指示に従う必要があります。

実行可能な ATMI サーバをビルドするには、buildserver を使用します。このコマンドではオプションを使用して、出力ファイル、アプリケーションで提供される入力ファイル、各種の方法で BEA Tuxedo システム・アプリケーションを実行するためのライブラリを指定します。

buildserver のコマンド行で指定できる主なオプションを以下の例に示します。

buildserver コマンドは .mk で使用され、株式アプリケーションの各サーバをコンパイルしてビルドします。詳細については、『BEA Tuxedo コマンド・リファレンス』の buildserver(1) を参照してください。

stockxml および stockxml_c サーバのビルド

次のように、buildserver コマンドで stockxml サーバと stockxml_c サーバをビルドします。

```
buildserver -s STOCKQUOTE -o stockxml -f stockxml.cpp -f DOMTreeErrorReporter.cpp
-f -ltxml
buildserver -s STOCKQUOTE -f stockxml_c.c -o stockxml_c -f xmlWrapper.cpp -f
DOMTreeErrorReporter.cpp -f -ltxml
```

以下は、コマンド行で指定されている各オプションの説明です。

- -o オプションは、実行可能出力ファイルに名前を指定する場合に使用します。名前が指定されていない場合は、SERVER という名前が付きます。
- -s オプションは、サーバの起動時に宣言されるサーバのサービス名を指定します。サービスを実行する関数名がサービス名と異なる場合、関数名が -s オプションの引数の一部になります。XMLSTOCKAPP では、関数名はサービス名と同じなので、サービス名だけを指定します。サービス名では、すべての文字列を大文字で指定します。ただし、buildserver の -s オプションでは、サーバ内のサービスを処理

する関数には任意の名前を付けることができます。詳細については、『BEA Tuxedo コマンド・リファレンス』の buildserver(1) を参照してください。システム管理者は、buildserver コマンドでサーバを作成した際に使用されたサービスのサブセットだけをサーバの起動時に利用できるように設定することもできます。詳細については、『BEA Tuxedo アプリケーション実行時の管理』および『BEA Tuxedo アプリケーションの設定』を参照してください。

- -f オプションは、リンク時と編集時に使用されるファイルを指定します。buildserver リファレンス・ページの -l オプションも参照してください。ファイルがリストされる順序には意味があります。この順序は、関数の参照、およびその参照がどのライブラリで解決されるかによって決定されます。ソース・モジュールは、関数の参照が解決されるライブラリの前にリストされます。オブジェクト・ファイルは、別個の .o ファイル、またはアーカイブ (.a) ファイルにあるファイル・グループです。-f の引数として 1 つ以上のファイル名を指定する場合は、二重引用符で各ファイル名を囲む必要があります。-f オプションは、必要な数だけ使用できます。
- -s オプションは、stockxml および stockxml_c サーバを構成するサービスとして STOCKQUOTE サービスを指定します。-o オプションは実行可能出力ファイルに stockxml および stockxml_c という名前を付け、-f オプションは stockxml.cpp、DOMTreeErrorReporter.cpp、および xmlWrapper.cpp ファイルがビルドでのリンク時と編集時に使用されることを指定します。

関連項目

- 6-2 ページの「XMLSTOCKAPP について」
- buildserver(1)

ステップ 5: コンフィギュレーション・ファイルの編集

ubbsimple サンプル・コンフィギュレーション・ファイルを編集して、山かっこで囲まれた項目を実際のインストールに適した値に置き換える必要があります。TUXDIR および TUXCONFIG 環境変数はコンフィギュレーション・ファイルの値と一致していなければなりません。

コード リスト 6-1 ubbsimple **コンフィギュレーション・ファイル**

```
1$
2
3 #BEA Tuxedo サンプル・アプリケーションのスケルトン・ファイルです。
4 #< 山かっこで囲まれた文字列 > を適切な値で置き換えます。
5 RESOURCES
6 IPCKEY            <32,768 よりも大きい IPC キーで置き換えます。>
7
8 # 例 :
9
10 #IPCKEY         62345
11
12 MASTER          simple
13 MAXACCESSERS    5
14 MAXSERVERS      5
15 MAXSERVICES     10
16 MODEL           SHM
17 LDBAL           N
18
19 *MACHINES
20
21 DEFAULT:
22
23         APPDIR="< 現在のパス名で置き換えます。>"
24         TUXCONFIG="<TUXCONFIG のパス名で置き換えます。>"
25         TUXDIR="<Tuxedo のルート・ディレクトリ (/ 以外 ) を指定します。>"
26 # 例 :
27 #       APPDIR="/usr/me/simpdir"
28 #       TUXCONFIG="/usr/me/simpdir/tuxconfig"
29 #       TUXDIR="/usr/tuxedo"
30
31 <Machine-name>  LMID=simple
```

```
32 # 例：
33 #tuxmach        LMID=simple
34 *GROUPS
35 GROUP1
36              LMID=simple   GRPNO=1  OPENINFO=NONE
37
38 *SERVERS
39  DEFAULT:
40              CLOPT="-A"
41  stockxml    SRVGRP=GROUP1 SRVID=1
42  stockxml_c  SRVGRP=GROUP1 SRVID=1
43 *SERVICES
44  STOCKQUOTE
```

5. 山かっこで囲まれた各 < 文字列 > を適切な値に置き換えます。

関連項目

- 6-5 ページの「XMLSTOCKAPP のファイルおよびリソースの準備」
- 『BEA Tuxedo のファイル形式とデータ記述方法』の UBBCONFIG(5)

ステップ 6: バイナリ形式のコンフィギュレーション・ファイルの作成

バイナリ形式のコンフィギュレーション・ファイルを作成する場合、XMLSTOCKAPP ファイルが置かれたディレクトリに移動し、環境変数を設定することが必要です。その場合、次の手順に従います。

1. XMLSTOCKAPP ファイルが置かれたディレクトリに移動します。
2. 次のコマンドを入力して、環境変数を設定します。

   ```
   . ./<variable_file>
   ```

   <variable_file> は変数ファイルの名前です。

コンフィギュレーション・ファイルのロード

コンフィギュレーション・ファイルを編集したら、それを MASTER マシン上にバイナリ・ファイルとしてロードする必要があります。バイナリ形式のコンフィギュレーション・ファイルの名前は TUXCONFIG、そのパス名は TUXCONFIG に定義されています。このファイルは、BEA Tuxedo のシステム管理者の有効なユーザ ID およびグループ ID を持つユーザが作成します。この 2 つの ID は、ご使用のコンフィギュレーション・ファイルの UID および GID の値と同じであることが必要です。同じではない場合、XMLSTOCKAPP の実行時にパーミッションの問題が発生します。

1. 次のコマンドを入力して、TUXCONFIG を作成します。

   ```
   tmloadcf ubbsimple
   ```

   コンフィギュレーションのロード時には、このコンフィギュレーションをインストールするかどうか、およびインストールする場合は既存のコンフィギュレーション・ファイルを上書きすることを確認するメッセージが何回か表示されます。このような確認を省くには、コマンド行で -y オプションを指定します。

2. アプリケーションで必要な IPC 資源を BEA Tuxedo システムで計算する場合は、コマンド行で -c オプションを指定します。

   TUXCONFIG は、MASTER マシン上だけにインストールできます。アプリケーションの起動時に tmboot によってほかのマシンに複製転送されます。

   ロードする前に tmloadcf によってテキスト形式のコンフィギュレーション・ファイル (UBBCONFIG) が解析されます。構文エラーが検出された場合、ファイルのロードは失敗します。

関連項目

- 6-5 ページの「XMLSTOCKAPP のファイルおよびリソースの準備」
- 『BEA Tuxedo コマンド・リファレンス』の tmloadcf(1)

XMLSTOCKAPP の実行

ここでは、XMLSTOCKAPP を起動し、各種のクライアント・プログラムとトランザクションを行ってテストし、終了する手順について順に説明します。

ステップ 1: 起動する前に行う準備作業

XMLSTOCKAPP を起動する前に、アプリケーションをサポートするのに十分な IPC 資源がマシンにあることを確認します。IPC 資源に関するレポートを出力するには、tmboot コマンドに -c オプションを指定します。

ステップ 2: XMLSTOCKAPP の起動

1. 環境変数を設定します。

   ```
   ../<variable_file>
   ```

2. 次のコマンドを入力して、アプリケーションを起動します。

   ```
   tmboot -y
   ```

   「-y」と入力すると、次のようなレポートが画面に出力されます。

   ```
   Booting all admin and server processes in
   /usr/me/appdir/tuxconfig
   Booting all admin processes
   exec BBL -A:
              process id=24223 Started.
   ```

このレポートは、コンフィギュレーション内のすべてのサーバが起動するまで出力されます。起動したサーバの合計数が出力された時点でレポートは終了します。

必要であれば、コンフィギュレーションの一部のサーバだけを起動することもできます。たとえば、管理サーバだけを起動するには、-A オプションを指定します。オプションが指定されていない場合は、アプリケーション全体が起動します。

tmboot では、起動したサーバ数がレポートされるほかに、ULOG にメッセージが送信されます。

関連項目

- 『BEA Tuxedo コマンド・リファレンス』の tmboot(1)
- 『BEA Tuxedo COBOL リファレンス』の USERLOG(3cbl)

ステップ 3: XMLSTOCKAPP サービスのテスト

1. 実行中のシステムにログインする場合は、XMLSTOCKAPP に環境変数を設定する必要があります。その場合、次のコマンドを入力します。

   ```
   ../<variable_file>
   ```

2. クライアント・プログラムを実行します。クライアント・プログラムを実行するには、以下のコマンドを入力します。

   ```
   Client stock_quote_beas.xml
   ```

ステップ 4: XMLSTOCKAPP のシャットダウン

XMLSTOCKAPP を終了するには、次のように、引数を指定せずに tmshutdown(1) コマンドを MASTER マシンで入力します。

```
tmshutdown -y
```

このコマンド (または tmadmin のシャットダウン・コマンド) を実行すると、次のタスクが行われます。

- すべてのアプリケーション・サーバ、ゲートウェイ・サーバ、TMS サーバ、および管理サーバがシャットダウンします。

- 割り当てられていたすべての IPC 資源が削除されます。

関連項目

- 『BEA Tuxedo コマンド・リファレンス』の `tmadmin(1)`
- 『BEA Tuxedo コマンド・リファレンス』の `tmshutdown(1)`