SYBASE®

プログラマーズ・ガイド補足

Open Client/Server™

12.5.1

Windows 版

ドキュメント ID：DC35454-01-1251-01
改訂：2003 年 11 月

マニュアルの注文

マニュアルの注文を承ります。ご希望の方は、サイベース株式会社営業部または代理店までご連絡ください。マニュアルの変更は、弊社の定期的なソフトウェア・リリース時にのみ提供されます。

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

目次

はじめに

このマニュアルでは、Open Client/Server™ リファレンス・マニュアルおよびプログラマーズ・ガイドを補足します。次の Microsoft Windows プラットフォーム用の Open Client/Server 製品を使用するアプリケーションの作成、設定、およびトラブルシューティングに必要な、プラットフォーム固有の情報を提供します。

- Windows NT
- Windows 2000
- Windows 2003
- Windows XP

このマニュアルでは、特に明記しないかぎり、すべてのプラットフォームを「Windows」と表記します。

対象読者

このマニュアルは、次の方を対象としています。

- Open Client/Server 製品を使用して、Sybase® およびサード・パーティのアプリケーションを作成するデスクトップ・アプリケーション開発者
- bcp、defncopy、isql、wbcp、wdefncopy、wdllvers、wisql ユーティリティについての情報を必要とする方
- cpre および cobpre プリコンパイラについての情報を必要としている方

関連マニュアル

各 Open Client/Server 製品には、ユーザ・マニュアルのセットがあります。次の表では製品とその関連マニュアルを示します。

表 1: 製品マニュアルのリスト

製品	関連マニュアル
Client-Library™	『Open Client Client-Library/C リファレンス・マニュアル』 『Open Client/Server Common Libraries リファレンス・マニュアル』 『Open Client Client-Library/C プログラマーズ・ガイド』
DB-Library™	『Open Client DB-Library/C リファレンス・マニュアル』
Server-Library	『Open Server Server-Library/C リファレンス・マニュアル』 『Open Client/Server Common Libraries リファレンス・マニュアル』 『Open Client Client-Library/C リファレンス・マニュアル』
Embedded SQL™	『Open Client Embedded SQL/C プログラマーズ・ガイド』 『Open Client Embedded SQL/COBOL プログラマーズ・ガイド』

次の情報については、『Open Client/Server 設定ガイド Windows 版』を参照してください。

- Open Client アプリケーションとサーバ間で通信するために環境を設定する方法
- Sybase アプリケーションをローカライズする方法

その他の情報ソース

Sybase Getting Started CD、Sybase Technical Library CD、Technical Library Product Manuals Web サイトを利用すると、製品について詳しく知ることができます。

- Getting Started CD には、PDF 形式のリリース・ノートとインストール・ガイドが収録されています。また、その他のマニュアルや、Technical Library CD には含まれない更新情報が収録されることもあります。この CD は製品のソフトウェアに同梱されています。Getting Started CD に収録されているマニュアルを参照または印刷するには、Adobe Acrobat Reader が必要です (CD 内のリンクを使用して Adobe の Web サイトから無料でダウンロードできます)。
- Technical Library CD には製品マニュアルが入っており、この CD は製品のソフトウェアに同梱されています。DynaText リーダー (Technical Library CD に収録) を使用すると、この製品に関する技術情報に簡単にアクセスできます。

  Technical Library のインストールと起動の方法については、マニュアル・パッケージに含まれている『Technical Library Installation Guide』を参照してください。
- Technical Library Product Manuals Web サイトは、Technical Library CD の HTML バージョンで、標準の Web ブラウザを使ってアクセスできます。また、製品マニュアルのほか、EBFs/Updates、Technical Documents、Case Management、Solved Cases、ニュース・グループ、Sybase Developer Network へのリンクもあります。

  Technical Library Product Manuals Web サイトにアクセスするには、Product Manuals (http://www.sybase.com/support/manuals/) にアクセスしてください。

Web 上の Sybase 製品の動作確認情報

Sybase Web サイトの技術的な資料は頻繁に更新されます。

❖ **製品認定の最新情報にアクセスする**

1 Web ブラウザで Technical Documents を指定します。(http://www.sybase.com/support/techdocs/)

2 左側のナビゲーション・バーから [Products] を選択します。

3 製品リストから製品名を選択し、[Go] をクリックします。

4 [Certification Report] フィルタを選択し、時間枠を指定して [Go] をクリックします。

5 [Certification Report] のタイトルをクリックして、レポートを表示します。

❖ **Sybase Web サイト (サポート・ページを含む) の自分専用のビューを作成する**

MySybase プロファイルを設定します。MySybase は無料サービスです。このサービスを使用すると、Sybase Web ページの表示方法を自分専用にカスタマイズできます。

1 Web ブラウザで Technical Documents を指定します。(http://www.sybase.com/support/techdocs/)

2 [MySybase] をクリックし、MySybase プロファイルを作成します。

Sybase EBF とソフトウェア・メンテナンス

❖ **EBF とソフトウェア・メンテナンスの最新情報にアクセスする**

1 Web ブラウザで Sybase Support Page (http://www.sybase.com/support) を指定します。

2 [EBFs/Maintenance] を選択します。すでに Web アカウントをお持ちの場合はユーザ名とパスワードを要求されますので、各情報を入力します。Web アカウントをお持ちでない場合は、新しいアカウントを作成します。サービスは無料です。

3 製品を選択します。

4 時間枠を指定して [Go] をクリックします。

5 EBF/Maintenance レポートを表示するには [Info] アイコンをクリックします。ソフトウェアをダウンロードするには製品の説明をクリックします。

表記の規則

構文の表記規則は、次のとおりです。

表 2: 構文の表記規則

構文要素	意味
command	太字で表記するものには、コマンド名、コマンドのオプション名、ユーティリティ名、ユーティリティのフラグ、キーワードがある。
variable	変数 (ユーザが入力する値を示す語句) は、斜体で表記する。
{ }	中カッコは、その中から必ず 1 つ以上のオプションを選択しなければならないことを意味する。コマンドには中カッコは入力しない。
[ ]	角カッコは、オプションを選択しても省略してもよいことを意味する。コマンドには角カッコは入力しない。
|	縦線は、中カッコまたは角カッコの中の縦線で区切られたオプションのうち 1 つだけを選択できることを意味する。
,	カンマは、中カッコまたは角カッコの中のカンマで区切られたオプションをいくつでも選択できることを意味する。オプションはコマンドの一部として入力する必要がある。

次に、上記の構文の表記規則の例を示します。

大カッコ内の縦線は、オプションを 1 つだけ選択することを意味します。

```
{red | yellow | blue}
```

大カッコ内のカンマは、オプションを 1 つ以上選択することを意味します。複数のオプションを選択する場合は、それぞれをカンマで区切ります。

```
{cash, check, credit}
```

角カッコは、オプションのパラメータを示します。角カッコ内の項目を 1 つも選択しなくてもかまいません。

角カッコに項目が 1 つだけある場合は、省略してもかまいません。

```
[anchovies]
```

角カッコ内の縦線は、オプションの選択が省略可能であること、または角カッコ内のオプションを 1 つだけ選択できることを示します。

```
[beans | rice | sweet_potatoes]
```

角カッコ内のカンマは、オプションを 1 つも選択しないか、または 1 つ以上のオプションを選択することを意味します。複数のオプションを選択する場合は、それぞれをカンマで区切ります。

```
[extra_cheese, avocados, sour_cream]
```

省略記号 (...) が続いている構文は、最後の単位を何度でも繰り返せることを意味します。次の構文の例では、1 つ以上のプログラム名と拡張子の組を角カッコで囲んでリストできます。

```
cpre -L program[.ext] [program[.ext]]...
```

不明な点があるときは

Sybase ソフトウェアがインストールされているサイトには、Sybase 製品の保守契約を結んでいるサポート・センタとの連絡担当の方 (コンタクト・パーソン) を決めてあります。マニュアルだけでは解決できない問題があった場合には、担当の方を通して Sybase のサポート・センタまでご連絡ください。

第 1 章

Open Client と Open Server のアプリケーションの構築

この章では、Windows プラットフォーム上で Open Client ライブラリと Open Server ライブラリを使用するアプリケーションを構築するために必要な情報について説明します。また、Sybase ライブラリを使用して Windows の実行プログラムを構築するための条件について説明します。

Open Client と Open Server の使用条件

Windows プラットフォームで Open Client と Open Server のアプリケーションをコンパイルしてリンクするには、次のような条件が必要です。

- ANSI 準拠の C コンパイラがインストールされている。
- INCLUDE 環境変数を定義する。
- LIB 環境変数を定義する。
- PATH 環境変数が *dll* サブディレクトリをインクルードするよう設定する。
- Windows 用の Open Client Net-Library ドライバが少なくとも 1 つサーバにネットワーク接続されている。

注意 Windows プラットフォーム対応の Sybase ライブラリは Win32 アプリケーション用に設計されています。

C コンパイラ

サンプル・プログラムを使用する場合やアプリケーションを構築する場合には、ANSI 準拠の C コンパイラがインストールされている必要があります。Sybase では、Microsoft Visual C++ バージョン 5.0 について動作を確認しています。ほかのコンパイラについても Sybase によって動作確認されている場合があります。動作確認されているコンパイラの最新のリストについては、Sybase の営業担当者に問い合わせてください。

Open Client および Open Server プログラムのコンパイルと実行の方法は、他の C 言語プログラムと同じです。アプリケーションをコンパイルしてリンクする方法については、使用するコンパイラのマニュアルを参照してください。

警告！ Sybase が動作確認をしていない ANSI 準拠の C コンパイラを使用して問題が発生した場合は、Sybase が動作確認しているコンパイラを使用して問題を再現できる場合にのみテクニカル・サポートを受けることができます。

Client-Library の互換性

Windows 上の Client-Library バージョン 12.5 は、表 1-1 に示されている Open Server™ と Sybase Adaptive Server® 製品で動作が保証されています。

表 1-1: Open Client の互換性

Open Client 12.5 プラットフォーム	Open Server 12.5.1	Open Server 12.5	Open Server 12.0	Adaptive Server 12.5.1	Adaptive Server 12.5	Adaptive Server 12.0
Windows NT 4.0 Service Pack 6 以降	x	x	x	x	x	x
Windows 2000、2003、XP	x	x	なし	x	x	なし

記号の説明：x = 互換性あり、なし = このプラットフォーム版の製品がない

このほかに、Open Client/C の互換性に関する次の問題に注意してください。

- Windows NT の場合、Sybase SQL Toolset™ バージョン 4.x および 5.0 (APT-Library™ アプリケーションおよび Report-Library アプリケーション) には、Open Client/C バージョン 12.0 のライブラリとの互換性がない。
- アプリケーションに含まれるヘッダ・ファイルは、アプリケーションがリンクしているライブラリと同じバージョン・レベルでなければならない。
- アプリケーションを構築するために使用するライブラリは、そのアプリケーションのコンパイルに使用するライブラリと同じバージョン・レベルでなければならない。

Open Server の互換性

Windows プラットフォームの Open Server バージョン 12.5 は、表 1-2 に示されている Client-Library/C と Adaptive Server の各製品との動作が保証されています。

表 1-2: Open Server の互換性

Open Server 12.5 のプラットフォーム	Client-Library 12.5.1	Client-Library 12.5	Client-Library 12.0	Adaptive Server 12.5.1	Adaptive Server 12.5	Adaptive Server 12.0
Windows NT 4.0 Service Pack 6 以降	x	x	x	x	x	x
Windows 2000、2003、XP	x	x		x	x	

記号の説明：**x** = 互換性あり、**なし** = このプラットフォーム版の製品がない

さらに、Open Server については、次の互換性の問題にも注意してください。

- アプリケーションに含まれるヘッダ・ファイルは、アプリケーションがリンクしているライブラリと同じバージョン・レベルでなければならない。
- Bulk-Library のルーチンは、Open Server バージョン 12.x のルーチンを呼び出すアプリケーションでは使用できない。
- DB-Library/C 11.x 以降は、Open Server 11.x 以降ではサポートされない。

環境変数とヘッダ・ファイル

アプリケーションを正しく機能させるには、いくつかの環境変数を設定しなければなりません。設定が必要な環境変数の説明については、表 1-3 を参照してください。

表 1-3: 環境変数の説明

変数	説明
INCLUDE	Sybase インストール・ディレクトリの *include* サブディレクトリを指すディレクトリ・パスを設定します。インストールが完了すると、このサブディレクトリにはヘッダ・ファイルが格納されます。
LIB	Sybase インストール・ディレクトリの *lib* サブディレクトリを指すディレクトリ・パスを設定します。インストールが完了すると、このサブディレクトリにはインポート・ライブラリ・ファイルが格納されます。
PATH	Sybase インストール・ディレクトリの *dll* サブディレクトリを指すディレクトリ・パスを設定します。インストールが完了すると、このサブディレクトリには Sybase DLL が格納されます。

ヘッダ・ファイル

表 1-4 は、Open Client と Open Server のアプリケーションをコンパイルするときにインクルードする必要のあるヘッダ・ファイルを示します。

表 1-4: Open Client と Open Server のヘッダ・ファイル

DB-Library ヘッダ・ファイル	Client-Library ヘッダ・ファイル	Server-Library ヘッダ・ファイル
sybdb.h ― DB-Library ルーチンで使用される定義と型定義を含んでいる。*sybdb.h* は、『Open Client DB-Library/C リファレンス・マニュアル』の説明どおりに使用する。*sybdb.h* ファイルには、ほかに必要なすべてのヘッダ・ファイルがインクルードされている。 *ssybfront.h* ― 『Open Client DB-Library/C リファレンス・マニュアル』で説明されている関数の戻り値と、STDEXIT および ERREXIT などの記号定数を定義している。また、*sybfront.h* には、プログラム変数の宣言に使用できるデータ型のための型定義も含まれている。 *ssyberror.h* ― エラー重大度の値を含んでいる。プログラムがこれらの値を参照する場合はインクルードする必要がある。	*ctpublic.h* ― すべての Client-Library アプリケーションで必須。このファイルには、ほかに必要なすべてのヘッダ・ファイルがインクルードされている。 *bkpublic.h* ― Bulk-Library を呼び出すアプリケーションの場合は必須。このファイルには、ほかに必要なすべてのヘッダ・ファイルがインクルードされている。	*ospublic.h* ― すべての Server-Library アプリケーションで必須。このファイルには、ほかに必要なすべてのヘッダ・ファイルがインクルードされている。 *bkpublic.h* ― Bulk-Library を呼び出すアプリケーションの場合は必須。このファイルには、ほかに必要なすべてのヘッダ・ファイルがインクルードされている。

インポート・ライブラリと DLL (ダイナミック・リンク・ライブラリ)

この項では、インポート・ライブラリと DLL (ダイナミック・リンク・ライブラリ) について説明します。

インポート・ライブラリ

Open Client と Open Server のインポート・ライブラリには、Open Client や Open Server のアプリケーションを構築するためにリンカで使用される情報が含まれています。表 1-5 は、アプリケーションをコンパイルしてリンクするときにインクルードする必要があるインポート・ライブラリを示します。

表 1-5: Open Client と Open Server のインポート・ライブラリ

DB-Library インポート・ライブラリ	Client-Library インポート・ライブラリ	Server-Library インポート・ライブラリ
libsybdb.dll － DB-Library *libcomn.lib* － 内部共有ユーティリティ・ライブラリ *libtcl.lib* － Net-Library™ *libintl.lib* － ローカライゼーション・サポート・ライブラリ	*libct.lib* － Client-Library *libcs.lib* － CS-Library *libblk.lib* － Bulk-Library Bulk-Library 呼び出しを使用する場合にだけ、Bulk-Library インポート・ライブラリ *libblk.lib* とリンクする必要がある。	*libsrv.lib* － Server-Library *libct.lib* － Client-Library *libcs.lib* － CS-Library *libblk.lib* － Bulk-Library Bulk-Library 呼び出しを使用する場合にだけ、Bulk-Library インポート・ライブラリ *libblk.lib* とリンクする必要がある。

DLL (ダイナミック・リンク・ライブラリ)

Windows の Open Client と Open Server ライブラリのアプリケーションは、実行時に Open Client DLL 内の関数を呼び出す必要があります。Sybase DLL がパスに含まれていることを確認してください。PATH 環境変数に Sybase インストール・ディレクトリの *dll* サブディレクトリを指定してください。表 1-6 は、Open Client と Open Server のライブラリに含まれる DLL を示します。

表 1-6: Open Client と Open Server の DLL

DB-Library の DLL	Client-Library の DLL	Server-Library の DLL
libsybdb.dll － DB-Library *libintl.dll* － ローカライゼーション・サポート・ライブラリ *libtcl.dll* － トランスポート制御レイヤ *libcomn.dll* － 内部共通ライブラリ	*libct.dll* － Client-Library *libcs.dll* － CS-Library *libintl.dll* － ローカライゼーション・サポート・ライブラリ *libtcl.dll* － トランスポート制御レイヤ *libcomn.dll* － 内部共通ライブラリ *libblk.dll* － Bulk-Library	*libct.dll* － Client-Library *libcs.dll* － CS-Library *libintl.dll* － ローカライゼーション・サポート・ライブラリ *libtcl.dll* － トランスポート制御レイヤ *libcomn.dll* － 内部共通ライブラリ *libsrv.dll* － Server-Library *libblk.dll* － Bulk-Library

設定条件

サンプル・プログラムや使用するアプリケーションが正しく動作するためには、次のような設定条件とシステム稼働条件が満たされている必要があります。

- SYBASE 環境変数が定義されている。
- *sql.ini* ファイルが、Open Client アプリケーションで使用されるサーバ名に対するクエリ・エントリを持っている。
- *sql.ini* ファイルが、Open Server アプリケーションで使用されるサーバ名に対するクエリ・エントリを持っている。
- Windows プラットフォームに最小限 64MB のメモリがある。

注意 SYBASE 環境変数の設定と *sql.ini* ファイルの設定の詳細については、『Open Client/Server 設定ガイド Windows 版』を参照してください。

プラットフォーム固有のデフォルト値

表 1-7 は、プラットフォーム固有のデフォルト値を持つ Open Client と Open Server のプロパティを示します。

表 1-7: Client-Library のプラットフォーム固有のプロパティ

ライブラリ	プロパティ名	説明	Windows でのデフォルト値
Client-Library Server-Library	CS_IFILE	*sql.ini* ファイルのパスと名前	SYBASE 環境変数で定義される *ini* サブディレクトリ内の *sql.ini* ファイル
	CS_MAX_CONNECT	このコンテキストの最大接続数	25
	CS_PACKETSIZE	TDS パケット・サイズ	512 バイト
DB-Library	DBSETFILE	*sql.ini* ファイルのパスと名前	SYBASE 環境変数で定義される *ini* サブディレクトリ内の *sql.ini* ファイル
	DBSETMAXPROS	このコンテキストの最大接続数	25

Client-Library のプログラミングについて

この項では、Windows プラットフォームでの特定の Client-Library ルーチンの動作と、『Open Client Client-Library/C リファレンス・マニュアル』および『Open Client Client-Library/C プログラマーズ・ガイド』でのそれらの説明との違いについて説明します。

ct_callback

ct_callback を使用して Client-Library に登録される Client-Library アプリケーション・ルーチンはすべて CS_PUBLIC として宣言し、*.def* ファイルにエクスポートしなければなりません。宣言の例については、サンプル・ディレクトリにある *exutils.c* 内のルーチン ex_clientmsg_cb を参照してください。

デバッグ DLL の使い方

インストール時に選択したオプションによっては、Client-Library のデバッグ・バージョンと非デバッグ・バージョン両方の *libct.dll* をインストールできます。デバッグ・バージョンの DLL は、Sybase *dll* ディレクトリの *debug* サブディレクトリに、非デバッグ・バージョンは *nondebug* サブディレクトリに格納されています。使用する方のバージョンを Sybase インストール・ディレクトリの *dll* サブディレクトリにコピーしてください。アプリケーションは Sybase インストール・ディレクトリの *dll* サブディレクトリ内の DLL を自動的に使用します。ct_debug ユーティリティは、デバッグ・バージョンの *libct.dll* を使用する場合にだけ機能します。

Client-Library のデバッグ・バージョンの詳細については、『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

オンライン・ヘルプ

Windows プラットフォーム用の Open Client では、C 言語版の Client-Library、CS-Library、Bulk-Library 用のオンライン・ヘルプが用意されています。このオンライン・ヘルプ機能の構成は次のとおりです。

- ルーチンの構文、ルーチンの戻り値、サンプル・コード、各ルーチンに関するコメントなどの C ルーチンの参照情報
- 3 つのライブラリのそれぞれについてのプログラミング情報

setup の実行時にオンライン・ヘルプ機能をインストールした場合、ヘルプの実行プログラムは Sybase インストール・ディレクトリ下の *¥help¥ctlib.hlp* に置かれます。Client-Library のヘルプには、次の方法のいずれかによってアクセスできます。

- [エクスプローラ] に表示される *ctlib.hlp* ファイルをダブルクリックします。
- [スタート] - [ファイル名を指定して実行] を選択し、ヘルプ・ファイルの名前を入力して [OK] をクリックします。

マルチスレッドのサポート

Client-Library バージョン 11.1 以降では、マルチスレッド・アプリケーションの開発に使用される Windows プラットフォームのスレッド・ライブラリがサポートされます。マルチスレッド・アプリケーションを作成する方法については、『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

コンパイルとリンクの実行例

この項では、Microsoft Visual C/C++ コンパイラ (バージョン 4.0) で使用できる、Windows 版 Client-Library アプリケーション用の *makefile* の例を示します。

```
#####################################################################
# Microsoft makefile for sample programs
#
#####################################################################
MAKEFILE=MAKEFILE

!ifndef SYBASE
SYBASEHOME=c:¥sybase
!else
SYBASEHOME=$(SYBASE)
!endif

COMPILE_DEBUG = 1

# Compiler AND linker flags
!ifdef COMPILE_DEBUG
CFLAGS = /W3 /MD /nologo /Zi /DWIN32
LFLAGS= /MAP /SUBSYSTEM:CONSOLE /DEBUG /DEBUGTYPE:cv
!else
CFLAGS = /W3 /MD /nologo /Od /DWIN32
LFLAGS= /MAP /SUBSYSTEM:CONSOLE
!endif

ASYNCDEFS = -DUSE_SIG_HANDLER=0
```

```
HDRS = example.h exutils.h
MTHDRS = example.h thrdutil.h thrdfunc.h

#
# Where to get includes and libraries
#
# SYBASE is the environment variable for sybase home directory
#
SYBINCPATH =        $(SYBASEHOME)¥$(SYBASE_OCS)¥include
BLKLIB =        $(SYBASEHOME)¥$(SYBASE_OCS)¥lib¥libblk.lib
CTLIB =                 $(SYBASEHOME)¥$(SYBASE_OCS)¥lib¥libct.lib
CSLIB =                 $(SYBASEHOME)¥$(SYBASE_OCS)¥lib¥libcs.lib
SYSLIBS =       kernel32.lib advapi32.lib msvcrt.lib

# The generalized how to make an .obj rule
.c.obj:
    cl /I. /I$(SYBINCPATH) $(ASYNCDEFS) $(CFLAGS) -Fo$@ -c $<

all: exasync compute csr_disp getsend rpc blktxt i18n multthrd usedir firstapp exconfig
secct wide_rpc wide_dynamic wide_curupd wide_compute

uni: uni_firstapp uni_csr_disp uni_compute uni_blktxt uni_rpc

exasync compute csr_disp getsend rpc blktxt i18n multthrd usedir firstapp exconfig secct
twophase: $*.exe
            @echo Sample '$*' was built

wide_rpc wide_dynamic wide_curupd wide_compute: $*.exe
            @echo Sample '$*' was built

uni_firstapp uni_csr_disp uni_compute uni_blktxt uni_rpc: $*.exe
            @echo Sample '$*' was built

sample.exe: sample.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe sample.obj $(SYSLIBS)

exasync.exe: ex_alib.obj ex_amain.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe ex_alib.obj ex_amain.obj exutils.obj $(SYSLIBS) $(CTLIB)
$(CSLIB)

compute.exe: compute.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

wide_curupd.exe: wide_curupd.obj exutils.obj wide_util.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj wide_util.obj $(SYSLIBS) $(CTLIB)
$(CSLIB)

wide_dynamic.exe: wide_dynamic.obj exutils.obj wide_util.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj wide_util.obj $(SYSLIBS) $(CTLIB)
$(CSLIB)
```

```
wide_compute.exe: wide_compute.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj wide_compute.obj $(SYSLIBS) $(CTLIB)
$(CSLIB)

exconfig.exe: exconfig.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

firstapp.exe: firstapp.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

csr_disp.exe: csr_disp.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

getsend.exe: getsend.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

rpc.exe: rpc.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

wide_rpc.exe: wide_rpc.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

blktxt.exe: blktxt.obj exutils.obj $(MAKEFILE)
   link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB) $(BLKLIB)

i18n.exe: i18n.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

multthrd.exe: multthrd.obj thrdfunc.obj thrdutil.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj thrdfunc.obj thrdutil.obj $(SYSLIBS) $(CTLIB)
$(CSLIB)

usedir.exe: usedir.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

secct.exe: secct.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

uni_firstapp.exe: uni_firstapp.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

uni_csr_disp.exe: uni_csr_disp.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

uni_compute.exe: uni_compute.obj exutils.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

uni_blktxt.exe: uni_blktxt.obj exutils.obj $(MAKEFILE)
   link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB) $(BLKLIB)

uni_rpc.exe: uni_rpc.obj exutils.obj $(MAKEFILE)
```

```
    link $(LFLAGS) -out:$*.exe $*.obj exutils.obj $(SYSLIBS) $(CTLIB) $(CSLIB)

twophase.exe: twophase.obj ctpr.obj ctxact.obj $(MAKEFILE)
    link $(LFLAGS) -out:$*.exe $*.obj ctpr.obj ctxact.obj $(SYSLIBS) $(CTLIB) $(CSLIB)
clean:
            -del *.obj
            -del *.map
            -del *.exe
            -del *.err
            -del *.ilk
            -del *.pdb
```

この例では次の点に注意してください。

- INTEL ライブラリが使用されています。
- Sybase ライブラリは Win32 アプリケーション用に作成されています。
- SUBSYSTEM:CONSOLE はコンソール・アプリケーションを示しています。

詳細については、コンパイルとリンクに関する Microsoft の適切なマニュアルを参照してください。

DB-Library のプログラミングについて

この項では、Windows プラットフォームでの特定の DB-Library ルーチンの動作と、『Open Client DB-Library/C リファレンス・マニュアル』でのそれらの説明との違いについて説明します。

コンパイルとリンク行の例

DB-Library/C アプリケーションをコンパイルしてリンクするためのコマンドの一般的な形式は次のとおりです。

```
!ifdef COMPILE_DEBUG
 CFLAGS = /W3 /MD /nologo /Z7 /DWIN32
 LFLAGS= /MAP /SUBSYSTEM:CONSOLE /DEBUG /DEBUGTYPE:cv
 !else
 CFLAGS = /W3 /MD /nologo /Od
 LFLAGS= /MAP /SUBSYSTEM:CONSOLE
 !endif
```

Server-Library のプログラミングについて

この項では、Windows での特定の Server-Library ルーチンの動作と、『Open Server Server-Library/C リファレンス・マニュアル』でのそれらの説明との違いについて説明します。

srv_callback

srv_callback を使用して Server-Library に登録される Server-Library アプリケーション・ルーチンは、すべて CS_PUBLIC として宣言しなければなりません。宣言の例については、サンプル・ディレクトリにある *utils.c* 内の cs_err_handler ルーチンを参照してください。

スケジューリング・モード

Windows で実行される Server-Library アプリケーションは、コルーチン・スケジューリング・モードまたはプリエンプティブ・スケジューリング・モードのいずれかで実行できます。コルーチン・スケジューリング (デフォルト) は、プリエンプティブ・スケジューリングをサポートしないほかのプラットフォームと互換性があります。プリエンプティブ・スケジューリング・モードを選択するには、srv_config 関数を使用して SRV_PREEMPT オプションを “true” に設定します。

プリエンプティブ・モード・プログラミングの概要

プリエンプティブ・スケジューリング・モードでは、すべてのスレッドを同時に実行でき、Windows スケジューラによって処理できます。プリエンプティブ・スケジューリングでは、1 つのスレッドがサーバを占有することはありません。プリエンプティブ・モードで実行する場合は、アプリケーションはデバッガのスレッド機能を使用してスレッドを操作できます。この場合、サーバを停止させずにブロック処理オペレーションを実行することもできます。プリエンプティブ・モードでは、スレッドの同期化によるオーバヘッドがないので、Windows 上で稼働するアプリケーションのパフォーマンスが向上します。

注意 コルーチン・スケジューリングしか使用できないプラットフォームへの移植性を保証するには、Windows 固有のセマフォ API を使用するのではなく、常に Server-Library の mutex 機能を使用してグローバル・データを保護してください。

この項では、srv_sleep 呼び出しと srv_wakeup 呼び出しを使用して Windows 固有のプリエンプティブ・プログラミングを行う方法について説明します。

srv_sleep

次のコード例は、プリエンプティブ・モードでの srv_sleep の使い方を示します。

```
/*
 ** Request the mutex to prevent the logging service
 ** from calling srv_wakeup before srv_sleep is called.
 */
 if (WaitForSingleObject(Mutex,INFINITE) != WAIT_OBJECT_0)
         return(CS_FAIL);
 /*
 ** Send the log_request to the logging service.
 */
 if (srv_putmsgq(log_service,log_request, SRV_M_NOWAIT) == CS_FAIL)
         return(CS_FAIL);

 /*
 ** Sleep until the log service has processed the log request.
 */
 srv_sleep(log_request, LOGWAIT, NULL, NULL, (CS_VOID*)Mutex, (CS_VOID*)0);

 /*
 ** Return the log sequence number to the caller.
 */
 return(CS_SUCCEED);
```

srv_wakeup

mutex を使用してプリエンプティブ・モードで srv_sleep を使用する場合は、対応する srv_wakeup ルーチンの前に同じ mutex に対する要求がなければなりません。この処理によって、スリープしているスレッドは、srv_wakeup の実行に対する準備ができます。次のコード例は、プリエンプティブ・モードで使用される場合、srv_wakeup の前に mutex に対する要求がどのように置かれるかを示しています。

```
/*
 ** Loop forever, logging language text. srv_getmsg will cause
 ** this thread to be suspended until a message is available on
 ** the log_request message queue.
 */
 while((get_status = srv_getmsgq(msgqid, &log_request,
         SRV_M_WAIT, &info)) == CS_SUCCEED)
 {
      /*
      ** Do the logging here.
     */

     /*
     ** Request the mutex to make sure the sender
     ** has called srv_sleep.
```

```
    */
    if (WaitForSingleObject(Mutex,INFINITE) != WAIT_OBJECT_0)
            return(CS_FAIL);

        /*
        ** Wake up the thread that is waiting for the language
        ** text to be logged.
        */
        srv_wakeup(log_request, SRV_M_WAKE_FIRST, (CS_VOID*)0, (CS_VOID*)0);

    /*
    ** Release the mutex.
    */
    if (!ReleaseMutex(Mutex))
            return(CS_FAIL);
}
```

コンパイルとリンクの実行例

次の例は、Windows の 32 ビット・アプリケーションをコンパイルしてリンクするための *makefile* を示しています。

```
#############################################################################

#

# Microsoft makefile for building Sybase Open Server Samples for Windows NT

#

#############################################################################

MAKEFILE=MAKEFILE

!ifndef SYBASE
YBASEHOME=c:¥sybase
!else
YBASEHOME=$(SYBASE)¥$(SYBASE_OCS)
!endif

COMPILE_DEBUG = 1
# Compiler AND linker flags
!ifdef COMPILE_DEBUG
CFLAGS = /W3 /MD /nologo /Z7 /DWIN32
LFLAGS= /MAP /SUBSYSTEM:CONSOLE /DEBUG /DEBUGTYPE:cv
!else
CFLAGS = /W3 /MD /nologo /Od /DWIN32
LFLAGS= /MAP /SUBSYSTEM:CONSOLE
!endif
SYSLIBS = kernel32.lib advapi32.lib msvcrt.lib
SYBASELIBS = $(SYBASEHOME)¥lib¥libcs.lib $(SYBASEHOME)¥lib¥libct.lib
$(SYBASEHOME)¥lib¥libsrv.lib
```

```
BLKLIB = $(SYBASEHOME)¥lib¥libblk.lib
DBLIB = $(SYBASEHOME)¥lib¥libsybdb.lib
CTOSOBJ =       args.obj    attn.obj    bulk.obj ¥
                    connect.obj     ctos.obj    cursor.obj ¥
                    dynamic.obj     error.obj   events.obj ¥
                    language.obj    mempool.obj     options.obj ¥
                    params.obj ¥
                    rgproc.obj  results.obj     rpc.obj ¥
                    send.obj    shutdown.obj
all: lang fullpass ctos regproc ctwait version intlchar osintro multthrd secsrv
lang fullpass ctos regproc ctwait version intlchar osintro multthrd secsrv: $*.exe
            @echo Sample '$*' was built
# The generalized how to make an .obj rule
.c.obj:
    cl /I. /I$(SYBASEHOME)¥include $(CFLAGS) -Fo$@ -c $<
lang.exe: lang.obj utils.obj
    link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

fullpass.exe: fullpass.obj utils.obj
        link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

ctos.exe: $(CTOSOBJ)
        link $(LFLAGS) -out:$*.exe $(CTOSOBJ) $(SYSLIBS) $(SYBASELIBS) $(BLKLIB)

regproc.exe: regproc.obj utils.obj
        link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

ctwait.exe: ctwait.obj ctutils.obj
        link $(LFLAGS) -out:$*.exe $*.obj ctutils.obj $(SYSLIBS) $(SYBASELIBS)

version.exe: version.obj ctutils.obj
        link $(LFLAGS) -out:$*.exe $*.obj ctutils.obj $(SYSLIBS) $(SYBASELIBS)

intlchar.exe: intlchar.obj utils.obj
        link $(LFLAGS) -out:$*.exe $*.obj utils.obj
$(SYSLIBS) $(SYBASELIBS)

osintro.exe: osintro.obj utils.obj
        link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

multthrd.exe: multthrd.obj utils.obj
        link $(LFLAGS) -out:$*.exe $*.obj utils.obj $(SYSLIBS) $(SYBASELIBS)

secsrv.exe: secsrv.obj utils.obj secargs.obj
        link $(LFLAGS) -out:$*.exe $*.obj utils.obj secargs.obj $(SYSLIBS) $(SYBASELIBS)

clean:
            -del *.obj
            -del *.map
            -del *.exe
            -del *.err

/*
```

この例では次の点に注意してください。

- INTEL ライブラリが使用されています。
- Sybase ライブラリは Win32 アプリケーション用に作成されています。
- SUBSYSTEM:CONSOLE はコンソール・アプリケーションを示しています。

詳細については、コンパイルとリンクに関する Microsoft の適切なマニュアルを参照してください。

第 2 章

Client Library/C のサンプル・プログラム

Open Client Client-Library は、クライアント・アプリケーションの作成に使用するルーチンの集まりです。この章のサンプル・プログラムは、Client-Library 関数と CS-Library 関数の例を示します。この章の内容は、次のとおりです。

Client-Library には、サーバにコマンドを送信するルーチンとそれらのコマンドの結果を処理するルーチンが含まれています。アプリケーション・プロパティの設定、エラー条件の処理、サーバとのアプリケーションの対話に関するさまざまな情報の提供を行うルーチンもあります。

Open Client に含まれている CS-Library は、Open Client アプリケーションや Open Server アプリケーションを作成するために使用できるユーティリティ・ルーチンの集まりです。Client-Library ルーチンは CS-Library で割り付けられる構造体を使用するので、すべての Client-Library アプリケーションには、CS-Library に対する呼び出しが少なくとも 1 つ含まれています。

サンプル・プログラムの使い方

サンプル・プログラムは、Client-Library/C 固有の機能の例を示しています。これらのプログラムは Client-Library/C のトレーニング用としてではなく、アプリケーション・プログラマのためのガイドとして設計されています。サンプル・プログラムを使用する前に、各ソース・ファイルの先頭にある説明を読んで、ソース・コードの内容を確認してください。

これらの簡単なプログラムは、実際の運用環境で使用するために作成されているものではありません。運用環境で使用するプログラムには、エラー処理や特殊ケースの処理のためのコードを追加する必要があります。

作業を始める前に

Open Client サンプル・プログラムを実行する前に、次の手順に従います。

- SYBASE 環境変数に Sybase リリース・ディレクトリのパスを設定していない場合は、設定します。
- DSQUERY 環境変数に接続先のサーバの名前 (*server_name*) を設定します。
- *makefile* を使用して、*example_name* というサンプル実行プログラムを作成します。

使用する環境と変数の設定の詳細については、『Open Client/Server 設定ガイド Windows 版』を参照してください。

サンプル・プログラムのロケーション

Client-Library サンプル・プログラムは Sybase リリース・ディレクトリの *sample¥ctlib* サブディレクトリ内にあります。このディレクトリには次のようなファイルが含まれています。

- サンプル・プログラムのオンライン・ソース・コード
- サンプル・プログラムのためのデータ・ファイル
- サンプル・プログラムのためのヘッダ・ファイル *example.h*

注意 サンプル・プログラムが存在するサブディレクトリの内容のバックアップ・コピーを作成してください。これによって、もとのファイルの整合性に影響を与えずにサンプル・プログラムを使用することができます。

ヘッダ・ファイル

表 2-1 では、すべての Client-Library アプリケーションに必要なヘッダ・ファイルを説明しています。

表 2-1: Client-Library アプリケーションに必要なヘッダ・ファイル

ファイル	説明
ctpublic.h	Client-Library に対する呼び出しを含んでいるすべてのアプリケーション・ソース・ファイルで必要。ファイルの内容は次のとおり。 • Client-Library ルーチンで使用される記号定数の定義 • Client-Library ルーチンのための宣言
cspublic.h	CS-Library ヘッダ・ファイル。ファイルの内容は次のとおり。 • クライアント／サーバ共通の記号定数の定義 • クライアント／サーバ共通の構造体のための型定義 • CS-Library ルーチンのための宣言
bkpublic.h	バルク・コピー・ルーチンに対する呼び出しを行うすべてのアプリケーション・ソース・ファイルで必要。
cstypes.h	Client-Library のデータ型のための型定義を含む。
sqlca.h	SQLCA 定義構造体のための型定義を含む。

example.h ファイル

すべてのサンプル・プログラムは、サンプル・ヘッダ・ファイル example.h を参照します。example.h の内容は、次のとおりです。

```
** example.h
**
** This is the header file that goes with the Sybase
** Client-Library example programs.
**
**
*///* Sccsid %Z% %M% %I% %G% */
/*
** Define symbolic names, constants, and macros
*/
#define         EX_MAXSTRINGLEN        255
#define         EX_BUFSIZE         1024
#define         EX_CTLIB_VERSION CS_VERSION_125
#define         EX_BLK_VERSION         BLK_VERSION_125
#define         EX_ERROR_OUT           stderr
/*
```

```
** exit status values
*/
#ifdef vms
#include <stsdef.h>
#define EX_EXIT_SUCCEED        (STS$M_INHIB_MSG | STS$K_SUCCESS)
#define EX_EXIT_FAIL           (STS$M_INHIB_MSG | STS$K_ERROR)
#else
#define EX_EXIT_SUCCEED        0
#define EX_EXIT_FAIL           1
#endif /* vms */
/*
** Define global variables used in all sample programs
*/
#define EX_SERVER      NULL           /* use DSQUERY env var */
#define EX_USERNAME    "sa"
#define EX_PASSWORD    ""
/*
** Uncomment the following line to test the HA Failover feature.
*/
/* #define HAFAILOVER 1 */
/*
** For some platforms (e.g. windows 3.1), additional work needs to be done
** to insure that the output of some of the example programs can be displayed.
** QuickWin's standard output screen buffer to unlimited size.
*/
#if QWIN
#define EX_SCREEN_INIT() _wsetscreenbuf(_fileno(stdout), _WINBUFINF)

#else /* QWIN */

#define EX_SCREEN_INIT()

#endif /* QWIN */
** This macro will insure that any setup is done for the platform.
**

** For windows, _wsetscreenbuf(_fileno(stdout), _WINBUFINT) will set
```

次に EX_USERNAME と EX_PASSWOED に対する変更について説明します。

- EX_USERNAME は、*example.h* 内で “user” と定義されています。*example.h* を編集して “user” をサーバのログイン名に変更してから、サンプル・プログラムを使用してください。
- EX_PASSWORD は、*example.h* 内で “server_password” と定義されています。サンプル・プログラムを使用する前に、*example.h* を編集して “server_password” をサーバのパスワードに変更できます。

  EX_PASSWORD に関しては次のような 3 つのオプションがあります。必要に応じて適切なものを選択してください。

  - 方法 1 － サンプル・プログラムを実行している間だけサーバのパスワードを “server_ password” に変更します。ただし、この方法はセキュリティを侵害される可能性があります。パスワードがこのような公開された値に設定されていると、承認されていないユーザでもサーバにログインできるからです。これでは問題がある場合は、ほかの 2 つの方法のどちらかを選択してください。
  - 方法 2 － *example.h* 内の文字列 “server_password” を、使用するサーバのパスワードに変更します。オペレーティング・システムの保護メカニズムを使用して、使用中はほかのユーザがヘッダ・ファイルにアクセスできないようにします。サンプル・プログラムの使用を終了したら、変更した行を “server_password” に戻します。
  - 方法 3 － サンプル・プログラム内で、サーバのパスワードを設定する ct_con_props コードを削除して、ユーザにサーバのパスワードを要求するようにコードを変更します。このコードはプラットフォームに固有なので、Sybase からは提供されません。

Client-Library のサンプル・プログラム

サンプル・プログラムは、Client-Library ルーチンの一般的な使い方の例を示すために Client-Library にオンラインで提供されています。サンプル・プログラムには、Adaptive Server で提供されるサンプル・データベースを使用するものもあります。サンプル・データベースをインストールする方法の詳細については、インストール・ガイドを参照してください。

サンプル・プログラムは C ソース・ファイルです。Client-Library のサンプル・プログラムを使用する場合やアプリケーションを構築する場合は、使用するプラットフォーム上に適切なコンパイラをインストールする必要があります。

注意 Open Client 12.5 ワイド・テーブル機能と unichar 機能を示す新しいサンプル・プログラムが含まれています。これらのプログラムには、uni_ または wide_ プレフィクスが付いています。各プログラムについては、下記の各項で説明しています。一部のサンプル・プログラムには、unipubs データベースが必要です。

---

sybopts.sh スクリプトとアプリケーションの構築

sybopts.sh スクリプトはサンプル・プログラムに含まれています。このスクリプトによって、使用しているプラットフォームに対応する Open Client や Open Server アプリケーションを構築することができます。この場合、次のコマンドを実行して SYBPLATFORM 環境変数を読み込みます。

```
sybopts.sh <args>
```

args には次のいずれかの引数を指定します。

- compile － コンパイラのコマンドとプラットフォーム固有のコンパイル・フラグを返す。
- comlibs － アプリケーションにリンクさせなければならない必須の Sybase ライブラリのリストを返す。
- syslibs － アプリケーションにリンクさせなければならない Sybase 以外の必須ライブラリのリストを返す。

サンプル・プログラムのためのユーティリティ・ルーチン

exutils.c ファイルには、ほかのすべての Client-Library サンプル・プログラムで使用されるユーティリティ・ルーチンが含まれています。この exutils.c は、アプリケーションが、より高いレベルのプログラムから Client-Library の実装の詳細部分を隠す方法を示しています。

これらのルーチンの詳細については、サンプル・ソース・ファイル内の先頭にあるコメントを参照してください。

firstapp.c サンプル・プログラム

サンプル・プログラム *firstapp.c* は、サーバに接続し、select クエリを送信して、ローを表示する初歩的な例です。このサンプル・プログラムについては、『Open Client Client-Library/C プログラマーズ・ガイド』の「第 1 章 Client-Library を使用する前に」を参照してください。

exconfig.c サンプル・プログラム

サンプル・プログラム *exconfig.c* は、Client-Library アプリケーションのプロパティを外部から設定する方法を示します。

このサンプル・プログラムを使用するには、Sybase インストール・ディレクトリ内にあるデフォルト・ランタイム設定ファイル *¥ini¥ocs.cfg* を編集する必要があります。このサンプル・プログラムは、Client-Library プロパティ CS_CONFIG_BY_SERVERNAME を設定し、*server_name* パラメータに “server1” を指定して ct_connect を呼び出します。これに対して、Client-Library は外部設定ファイルで [Server1] セクションを探します。このサンプル・プログラムを実行するには、必要に応じて Sybase インストール・ディレクトリに *¥ini¥ocs.cfg* を作成して、次のセクションを追加してください。

```
[server1]
 CS_SERVERNAME = real_server_name
```

real_server_name には、接続するサーバの名前を指定します。

Client-Library での外部設定ファイルの使用方法の詳細については、『Open Client Client-Library/C リファレンス・マニュアル』の「ランタイム設定ファイルの使い方」の項を参照してください。

バルク・コピーのサンプル・プログラム

サンプル・プログラム *blktxt.c* は、バルク・コピー・ルーチンを使用して静的データをサーバ・テーブルにコピーします。このプログラムでは、プログラム変数にバインドされている 3 つのローのデータが、1 つのバッチとしてサーバに送信されます。このローは、テキスト・データを送信するために blk_textxfer を使用してもう一度送信されます。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

注意　このサンプル・プログラムを実行するには、SQL Server® バージョン 11.1 以降が必要です。

計算ローのサンプル・プログラム

サンプル・プログラム *compute.c* は、計算結果の処理の例を示し、次の処理を実行します。

- このプログラムは、準備されたクエリを、言語コマンドを使用してサーバに送信します。
- 標準の ct_results の while ループを使用して結果を処理します。
- カラム値をプログラム変数にバインドします。
- そのあと、標準の ct_fetch の while ループでローをフェッチして表示します。

準備されたクエリは次のとおりです。

```
select type, price from titles
 where type like "%cook"
 order by type, price
 compute sum(price) by type
 compute sum(price)
```

このクエリは、通常のローと計算ローの両方を返します。計算ローは 2 つの compute 句によって生成されます。

- 最初の compute 句は、type の値が変化するたびに計算ローを生成します。

  ```
  compute sum(price) by type
  ```

- 2 つ目の compute 句は、最後に返される 1 つの計算ローを生成します。

  ```
  compute sum(price)
  ```

このプログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

注意 このサンプル・プログラムを実行するには、pubs2 データベースと titles テーブルが必要です。

読み込み専用カーソルのサンプル・プログラム

サンプル・プログラム *csr_disp.c* は、読み込み専用カーソルの使い方を示します。このプログラムは、まず準備されたクエリでカーソルをオープンします。次に、標準の ct_results の while ループを使用して結果を処理し、カラム値をプログラム変数にバインドします。最後に、標準の ct_fetch の while ループでローをフェッチして表示します。

準備されたクエリは次のとおりです。

```
select au_fname, au_lname, postalcode
 from authors
```

このプログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

注意　このサンプル・プログラムを実行するには、SQL Server バージョン 11.1 以降 (または Adaptive Server)、pubs2 データベース、authors テーブルが必要です。

非同期処理のサンプル・プログラム

このサンプル・プログラムには *ex_alib.c* と *ex_amain.c* の 2 つのファイルがあり、Client-Library で、非同期ライトを行う方法を示します。このプログラムは、Client-Library によって提供される仕組みを使用して連続的なポーリングと、Client-Library の完了コールバックの使用を可能にします。

このサンプル・プログラムは次の 2 つのファイルで構成されます。

- *ex_alib.c* － サンプル・プログラムのライブラリ部分のソース・コードを含んでいます。これは、非同期呼び出しをサポートするライブラリ・インタフェースの一部であることを意味します。このモジュールは、1 つの非同期オペレーション内でサーバにクエリを送信してサーバから結果を取り出す手段を提供します。
- *ex_amain.c* － *ex_alib.c* によって提供されるサービスを使用するメイン・プログラムのソース・コードが含まれています。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントと *EX_AREAD.ME* ファイルを参照してください。

注意　このサンプル・プログラムを実行するには、SQL Server バージョン 11.1 以降 (または Adaptive Server) が必要です。

text および *image* サンプル・プログラム

サンプル・プログラム *getsend.c* は、テキストとその他のデータ型を含んでいるテーブルから text データを取得して更新する方法の例を示すものです。この例に示す処理は、image データの検索と更新に使用することもできます。Open Server アプリケーションに接続する場合は、その Open Server アプリケーションは Adaptive Server 用の言語コマンドを処理できなければなりません。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

注意　このサンプル・プログラムを実行するには、SQL Server バージョン 11.1 以降 (または Adaptive Server)、pubs2 データベース、authors テーブルが必要です。

ローカライゼーションと国際化のサンプル・プログラム

i18n.c サンプル・プログラムは、Client-Library で使用できる次のような国際化機能の一部を示します。

- ローカライズされたエラー・メッセージ
- ユーザ定義のバインド・タイプ

このプログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

マルチスレッド・サンプル・プログラム

このサンプル・プログラムには、マルチスレッド Client-Library アプリケーションの例を示す *multthrd.c* と *thrdfunc.c* という 2 つのファイルが含まれています。

このサンプル・プログラムは次の 2 つのファイルで構成されます。

- *multthrd.c* − 5 つのスレッドを生成するソース・コードを含んでいます。各スレッドは 1 つのカーソルまたは 1 つの通常のクエリを処理します。メイン・スレッドはほかのスレッドがクエリ処理を完了するまで待ってから終了します。
- *thrdfunc.c* − サンプル・プログラムが実行に使用するスレッド・ルーチンと同期化ルーチンを決定するプラットフォーム固有の情報を含んでいます。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

注意 このサンプル・プログラムは、Client-Library によってサポートされるスレッド・パッケージが、使用しているプラットフォーム上に存在しない場合は実行できません。また、このサンプル・プログラムを実行するには、SQL Server バージョン 11.1 以降 (または Adaptive Server) が必要です。

RPC コマンドのサンプル・プログラム

RPC (リモート・プロシージャ・コール) コマンドのサンプル・プログラム *rpc.c* は、RPC コマンドをサーバに送信してその結果を処理します。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

注意 このサンプル・プログラムを実行するには、SQL Server バージョン 11.1 以降が必要です。

セキュリティ・サービスのサンプル・プログラム

サンプル・プログラム *secct.c* は、Client-Library アプリケーションでネットワークベース・セキュリティ機能を使用する方法を示します。

このサンプル・プログラムを実行するには、使用するマシンに DCE または CyberSafe Kerberos をインストールして稼働させる必要があります。また、Security Guardian や Open Server のサンプル・プログラム *secsrv.c* などのネットワークベース・セキュリティをサポートするサーバに接続することも必要です。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。ネットワーク・セキュリティ・サービスの詳細については、『Open Client/Server 設定ガイド Windows 版』を参照してください。

ディレクトリ・サービスのサンプル・プログラム

ディレクトリ・サービスのサンプル・プログラム *usedir.c* は、使用できるサーバのリストをディレクトリ・サービスに問い合わせます。

usedir.c は、ドライバ設定ファイル内の定義に従ってデフォルト・ディレクトリで Sybase サーバ・エントリを検索します。ネットワーク・ディレクトリ・サービスが使用されていない場合、*usedir.c* は *sql.ini* ファイルにサーバ・エントリがあるかどうかを調べます。そのあと、検索された各エントリの内容を表示して、接続するサーバをユーザが選択できるようにします。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。ディレクトリ・サービスの詳細については、『Open Client/Server 設定ガイド Windows 版』を参照してください。

uni_blktxt.c サンプル・プログラム

サンプル・プログラム *uni_blktxt.c* は、バルク・コピー・ルーチンを使用して静的データをサーバ・テーブルにコピーします。このプログラムは、unichar データ型と univarchar データ型を使用するために修正されています。プログラム変数にバインドされてサーバにまとめて送信される 3 つのローのデータがあります。このローは、テキスト・データを送信するために blk_textxfer を使用してもう一度送信されます。

uni_compute.c サンプル・プログラム

サンプル・プログラム *uni_compute.c* は、計算結果の処理の例を示すものです。このサンプル・プログラムは unichar データ型と univarchar データ型を使用するために *compute.c* サンプル・プログラムを修正したものです。実行するには unipubs データベースが必要です。このプログラムは、まず準備されたクエリを、言語コマンドを使用してサーバに送信します。標準の ct_results の while ループを使用して結果を処理します。次に、カラム値をプログラム変数にバインドします。最後に、標準の ct_fetch の while ループでローをフェッチして表示します。

uni_csr_disp.c サンプル・プログラム

サンプル・プログラム *uni_csr_disp.c* は、読み込み専用カーソルの使い方を示します。このサンプル・プログラムは *uni_csr_disp.c* サンプル・プログラムを修正したものです。実行するには unipubs データベースが必要です。このプログラムは、準備されたクエリでカーソルをオープンします。標準の ct_results の while ループを使用して結果を処理します。カラム値をプログラム変数にバインドします。そのあと、標準の ct_fetch の while ループでローをフェッチして表示します。

準備されたクエリは次のとおりです。

```
select au_fname, au_lname, postalcode
 from authors
```

uni_firstapp.c サンプル・プログラム

このサンプル・プログラムは、unichar データ型と univarchar データ型によって使用するために *firstapp.c* を修正したものです。サーバに接続する初歩的なサンプル・プログラムであり、select クエリを送信し、ローを出力します。*firstapp.c* プログラムについては、『Open Client Client-Library/C プログラマーズ・ガイド』を参照してください。

uni_rpc.c サンプル・プログラム

RPC コマンドのサンプル・プログラム *uni_rpc.c* は、RPC コマンドをサーバに送信してその結果を処理します。このサンプル・プログラムは unichar データ型と univarchar データ型を使用するために *rpc.c* サンプル・プログラムを修正したものです。実行するには unipubs データベースが必要です。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

usedir.c サンプル・プログラム

このサンプル・プログラムは、使用可能なサーバのリストをディレクトリ・サービスに問い合わせる Client-Library の機能を示します。

usedir.c は、ドライバ設定ファイル内の定義に従ってデフォルト・ディレクトリで Sybase サーバ・エントリを検索します。ネットワーク・ディレクトリ・サービスが使用されていない場合、*usedir.c* は *interfaces* ファイルにサーバ・エントリがあるかどうかを調べます。そのあと、検索された各エントリの内容を表示して、接続するサーバをユーザが選択できるようにします。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。ネットワーク・ディレクトリ・サービスの詳細については、『Open Client/Server 設定ガイド UNIX 版』を参照してください。

wide_compute.c サンプル・プログラム

サンプル・プログラム *wide_compute.c* は、ワイド・テーブルと大きなカラム・サイズによる計算結果の処理を示します。このサンプル・プログラムは Open Client と Open Server のバージョン 12.5 に実装されています。このプログラムは、まず準備されたクエリを、言語コマンドを使用してサーバに送信します。標準の ct_results の while ループを使用して結果を処理します。カラム値をプログラム変数にバインドします。そのあと、標準の ct_fetch の while ループでローをフェッチして表示します。

準備されたクエリは次のとおりです。

```
select type, price from titles
 where type like "%cook"
 order by type, price
 compute sum(price) by type
 compute sum(price)
```

このクエリは、通常のローと計算ローの両方を返します。計算ローは 2 つの compute 句によって生成されます。

- 最初の compute 句は、type の値が変化するたびに計算ローを生成します。

  ```
  compute sum(price) by type
  ```

- 2 つ目の compute 句は、最後に返される 1 つの計算ローを生成します。

  ```
  compute sum(price)
  ```

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

注意　このサンプル・プログラムを実行するには、pubs2 データベースが必要です。

---

wide_curupd.c サンプル・プログラム

このプログラムは、カーソルを使用して pubs2 データベース内の publishers テーブルからデータを取り出します。ローごとにデータを取得し、publishers テーブル内の “state” カラムに新しい値を入力するプロンプトを表示します。

更新のために、入力パラメータ用の値 (“publishers” テーブルの “state” カラム) を入力します。次に示すコマンドを実行して publishers3 テーブルを作成してから、サンプル・プログラムを実行してください。

```
use pubs2
go
drop table publishers3
go
create table publishers3 (pub_id char(4) not null,
pub_name varchar(400) null, city varchar(20) null,
state char(2) null)
go
select * into publishers3 from publishers
go
create unique index pubind on publishers3(pub_id)
```

wide_dynamic.c サンプル・プログラム

このプログラムは、カーソルを使用して *pubs2* データベース内の publishers テーブルからデータを取り出します。ローごとにデータを取得し、publishers テーブル内の “state” というカラムに新しい値を入力するプロンプトを表示します。

このプログラムは、動的 SQL を使用して tempdb データベース内の titles テーブルから値を取り出します。識別子の付いたプレースホルダを含む select 文が、サーバに送信されて部分的にコンパイルされ、保存されます。したがって、select を呼び出すたびに、取得されるローを決定するキー値の新しい値だけを渡します。動作は、ストアド・プロシージャに入力パラメータを渡す動作に似ています。また、このプログラムはカーソルを使用してローを 1 つずつ取得します。必要に応じて、この操作を実行できます。

wide_rpc.c サンプル・プログラム

RPC コマンドのサンプル・プログラム *wide_rpc.c* は、RPC コマンドをサーバに送信してその結果を処理します。この動作は *rpc.c* プログラムと同じですが、異なる点は、ワイド・テーブルと大きなカラム・サイズを使用することです。

このサンプル・プログラムの詳細については、サンプル・ソース・ファイルの先頭にあるコメントを参照してください。

第 3 章

Open Client DB-Library/C のサンプル・プログラム

Open Client DB-Library はクライアント・アプリケーションの作成に使用できるルーチンの集まりです。この章のサンプル・プログラムは DB-Library の機能の例を示します。この章の内容は、次のとおりです。

DB-Library には、サーバにコマンドを送信するルーチンとそれらのコマンドの結果を処理するルーチンが含まれています。アプリケーション・プロパティの設定、エラー条件の処理、サーバとのアプリケーションの対話に関するさまざまな情報の提供を行うルーチンもあります。

サンプル・プログラムの使い方

サンプル・プログラムは、DB-Library/C 固有の機能の例を示しています。これらのプログラムは DB-Library/C のトレーニング用としてではなく、アプリケーション・プログラマのためのガイドとして設計されています。サンプル・プログラムを使用する前に、各ソース・ファイルの先頭にある説明を読んで、ソース・コードの内容を確認してください。

注意 これらの簡単なプログラムは、実際の運用環境で使用するために作成されているものではありません。運用環境で使用するプログラムには、エラー処理や特殊ケースの処理のためのコードを追加する必要があります。

作業を始める前に

使用しているプラットフォームに対応してすべてのサンプル・プログラムを実行するには、特定の準備が必要です。必要な手順については、表 3-1 を参照してください。その他の条件については、個々のサンプル・プログラムと *README* ファイルを参照してください。

表 3-1: DB-Library アプリケーションの実行に必要な手順

プラットフォーム	手順
すべての Windows プラットフォーム	• DSQUERY 環境変数に接続先のサーバの名前 (*server_name*) を設定します。 • SYBASE 環境変数に Sybase インストール・ディレクトリのパスを設定していない場合は、設定します。 • *makefile* を使用して、*example name* というサンプル実行プログラムを作成します。

サンプル・プログラムのロケーション

サンプル・プログラムは Sybase インストール・ディレクトリの *¥sample¥dblibrary* ディレクトリ内に配置されます。

このディレクトリには次のようなファイルが含まれています。

- サンプル・プログラムのオンライン・ソース・コード
- サンプル・プログラムのためのデータ・ファイル
- *sybdbex.h* を含むヘッダ・ファイル

注意 サンプル・プログラムが存在するサブディレクトリの内容のバックアップ・コピーを作成してください。これによって、もとのファイルの整合性に影響を与えずにサンプル・プログラムを使用することができます。

ヘッダ・ファイル

次のヘッダ・ファイルは、すべての DB-Library/C アプリケーションで必要です。

- *sybfront.h* －関数の戻り値 (『Open Client DB-Library/C リファレンス・マニュアル』を参照) と、終了値 STDEXIT と ERREXIT などの記号定数を定義しています。*sybfront.h* には、プログラム変数の宣言に使用できるデータ型のための型定義も含まれています。
- *sybdb.h* － 追加の定義と型定義を含んでいます。これらの定義のほとんどは、DB-Library/C ルーチンだけで使用されます。*sybdb.h* の内容は、『Open Client DB-Library/C リファレンス・マニュアル』の説明に従って使用してください。
- *syberror.h* － エラー重大度の値を含んでいます。プログラムがこれらの値を参照する場合はインクルードする必要があります。

ヘッダ・ファイルの詳細については、『Open Client DB-Library/C リファレンス・マニュアル』を参照してください。

sybdbex.h ヘッダ・ファイル

すべてのサンプル・プログラムは、サンプル・ヘッダ・ファイル *sybdbex..h* を参照します。*sybdbex.h* の内容は、次のとおりです。

```
/*
 ** sybdbex.h
 **
 ** This is the header file that goes with the
 ** Sybase DB-Library example programs.
 **
 **
 */

 #define USER             "user"
 #define PASSWORD         "server_password"
 #define LANGUAGE         "us_english"
 #define SQLBUFLEN        255
 #define ERR_CH           stderr
 #define OUT_CH           stdout
 extern void              error();
 extern int               err_handler();
 extern int               msg_handler();
```

データ変換のサンプル・プログラム以外のすべてのサンプル・プログラムには、次の行が含まれています。

```
DBSETLUSER(login, USER);


DBSETLPWD(login, PASSWORD);
```

次に EX_USERNAME、EX_PASSWOED、および LANGUAGE 変数に対する変更について説明します。

- EX_USERNAME は、*sybdbex.h* 内で “user” と定義されています。*sybdbex.h* を編集して “user” をサーバのログイン名に変更してから、サンプル・プログラムを実行してください。
- EX_PASSWORD は、*sybdbex.h* 内で “server_password” と定義されています。サンプル・プログラムを実行する前に、*sybdbex.h* を編集して “server_password” をサーバのパスワードに変更することができます。

  EX_PASSWORD に関しては次のような 3 つのオプションがあります。必要に応じて適切なものを選択してください。

  - 方法 1 − サンプルを実行している間だけ、サーバのパスワードを “server_password” に変更します。ただし、この方法はセキュリティを侵害される可能性があります。パスワードがこのような公開された値に設定されていると、承認されていないユーザでもサーバにログインできてしまいます。これでは問題がある場合は、ほかの 2 つの方法のどちらかを選択してください。
  - 方法 2 − *sybdbex.h* 内の文字列 “server_password” を、使用するサーバのパスワードに変更します。オペレーティング・システムの保護メカニズムを使用して、使用中はほかのユーザがヘッダ・ファイルにアクセスできないようにします。サンプル・プログラムの使用を終了したら、変更した行を “server_password” に戻します。
  - 方法 3 − サンプル・プログラム内で、サーバのパスワードを設定する ct_con_props コードを削除して、ユーザにサーバのパスワードを要求するようにコードを変更します。このコードはプラットフォームに固有なので、Sybase からは提供されません。
- LANGUAGE は、*sybdbex.h* 内で “us_english” と定義されています。サーバの言語が “us_english” でない場合は、*sybdbex.h* を編集して “us_english” をサーバの言語に変更できます。国際言語ルーチンのサンプル・プログラム *exampl12.c* は、LANGUAGE を参照する唯一のサンプル・プログラムです。

DB-Library のサンプル・プログラム

サンプル・プログラムは、DB-Library ルーチンの一般的な使い方の例を示すためにオンラインで提供されています。サンプル・プログラムには、Adaptive Server で提供されるサンプル・データベースを使用するものもあります。サンプル・データベースをインストールする方法の詳細については、インストール・ガイドを参照してください。

サンプル・プログラムは C ソース・ファイルです。DB-Library のサンプル・プログラムを使用する場合やアプリケーションを構築する場合は、使用するプラットフォーム上に適切なコンパイラをインストールしてください。

クエリの送信、および結果のバインドと出力

example1.c は、1 つのコマンド・バッチで 2 つのクエリを Adaptive Server に送信し、結果をバインドして、返されたデータのローを出力します。

注意　このサンプル・プログラムを実行するには、Adaptive Server にアクセスする必要があります。

新しいテーブルへのデータ挿入

example2.c は、新しく作成されたテーブルにファイルからデータを挿入し、サーバのローを選択して、結果のバインドと出力を行います。

注意　このサンプル・プログラムを実行するには、Adaptive Server にアクセスする必要があります。また、このサンプル・プログラムには、提供されている *datafile* という名前のファイルと、ログイン・データベースに対する create database パーミッションが必要です。

集約結果と計算結果のバインド

example3.c は、pubs2 データベース内の titles テーブルから情報を選択して出力します。このプログラムは、集約結果と計算結果の両方のバインドの例を示すものです。

注意　このサンプル・プログラムを実行するには、pubs2 データベースを格納している Adaptive Server にアクセスする必要があります。

ロー・バッファリング

example4.c はロー・バッファリングの例です。このプログラムは、Adaptive Server にクエリを送信し、返されたローをバッファに入れて、それらを対話的に調べることができるようにします。

注意 このサンプル・プログラムを実行するには、Adaptive Server にアクセスする必要があります。

データ変換

example5.c は、データ変換を処理する DB-Library/C ルーチン dbconvert の例を示します。

ブラウズ・モードでの更新

example6.c は、ブラウズ・モードについての例です。このプログラムはテーブルを作成し、そのテーブルにデータを挿入して、ブラウズ・モード・ルーチンを使用してそのテーブルを更新します。ブラウズ・モードはデータのローを一度に 1 つずつ更新するアプリケーションに便利です。

example6.c を使用するには、提供されている *datafile* という名前のファイルが必要です。このプログラムはデフォルト・データベースにテーブル alltypes を作成します。

注意 このサンプル・プログラムを実行するには、Adaptive Server にアクセスする必要があります。

ブラウズ・モードとアドホック・クエリ

example7.c は、ブラウズ・モードを使用してアドホック・クエリによる結果カラムのソースを調べます。

ブラウズ可能なテーブルから導出されたカラム、および SQL 式の結果ではないカラムを更新できるのはブラウズ・モードのアプリケーションだけなので、結果カラムのソースを調べることは重要です。

このサンプル・プログラムは、ブラウズ・モードを使用して更新できる、アドホック・クエリによる結果カラムはどれであるかをアプリケーションが調べる方法を示します。

このサンプル・プログラムは、アドホック・クエリの入力を要求します。select クエリがキーワード for browse を含んでいるかどうか、選択されるテーブルをブラウズできるかどうかによって、どのように結果が異なるかに注目してください。

注意　このサンプル・プログラムを実行するには、Adaptive Server にアクセスする必要があります。

RPC 呼び出しの実行

example8.c は、RPC (リモート・プロシージャ・コール) を送信し、その RPC による結果ローを表示して、リモート・プロシージャによって返されたパラメータとステータスを表示します。

このサンプル・プログラムを使用するには、デフォルト・データベース内にストアド・プロシージャ rpctest を作成しておかなければなりません。*example8.c* ソース・コードの先頭のコメントは、rpctest を作成するのに必要な create procedure 文を指定します。

注意　このサンプル・プログラムを実行するには、Adaptive Server にアクセスする必要があります。

text ルーチンと image ルーチン

example9.c では、ランダムなイメージを生成し、そのイメージをテーブルに挿入してから、挿入されたイメージを選択して元のイメージと比較します。このとき、次の手順に従います。

1　text 値または image 値を除くすべてのデータを、ローに挿入します。

2　ローを更新して text 値または image 値を NULL に設定します。これは必須の手順です。なぜなら、null である text 値または image 値に有効なテキスト・ポインタが割り当てられるのは、その null 値が update 文によって明示的に入力された場合に限られるからです。

3　ローを選択します。text 値または image 値を含むカラムを明確に選択しなければなりません。これは、アプリケーションの DBPROCESS に正しいテキスト・ポインタとテキスト・タイムスタンプ情報を提供するために必要な手順です。アプリケーションは、この select コマンドによって返されたデータを破棄します。

4 dbtxtptr を呼び出して、テキスト・ポインタを DBPROCESS から取り出します。dbtxtptr の *column* パラメータは整数で、手順 3 で実行された select オペレーションを参照します。たとえば、“text_column” が text カラムの名前である場合、select 文は次のような構文を読み込みます。

```
select date_column, integer_column, text_column
     from bigtable
```

dbtxtptr は、*column* が 3 として渡されるように要求します。

5 dbtxtimestamp を呼び出して、DBPROCESS からテキスト・タイムスタンプを取り出します。dbtxtimestamp の *column* パラメータは、手順 3 で実行した select オペレーションを参照します。

6 text 値または image 値を Adaptive Server に書き込みます。アプリケーションは次のどちらかを実行できます。

- 1 回の dbwritetext 呼び出しで値を書き込む。
- dbwritetext と dbmoretext を使用して、まとまりごとに値を書き込む。

注意 アプリケーションに、この text 値または image 値に対してさらに更新を実行させるときは、正常に実行された dbwritetext のオペレーションの終りに、Adaptive Server によって返される新しいテキスト・タイムスタンプを保存しなければならない場合があります。新しいテキスト・タイムスタンプには、dbtxtsnewval を使用してアクセスできます。また、あとで取り出せるように dbtxtsput を使用して保存できます。

また、このサンプル・プログラムを実行するには、pubs2 データベースを格納している Adaptive Server にアクセスする必要があります。

---

image の挿入

exampl10.c は、作家 ID とイメージを含んでいるファイルの名前を要求し、そのファイルからイメージを読み込んで、作家 ID とイメージを含んでいる新しいローを pubs2 データベースのテーブル au_pix に挿入します。text 値と image 値をデータベース・テーブルに挿入する方法については、「text ルーチンと image ルーチン」(39 ページ) を参照してください。

注意 *exampl10.c* を実行するには、Adaptive Server と pubs2 データベースにアクセスする必要があります。作家 ID は “*nnn-nn-nnnn*” のフォーマット (*n* は数値) でなければなりません。サンプル・コードとともに提供されている *imagefile* には image が入っています。

イメージの取り出し

exampl11.c は、pubs2 データベース内の *au_pix* テーブルからイメージを取り出します。入力する作家 ID によって、プログラムが選択するローが決まります。ローを取り出したあと、このサンプル・プログラムは *pic* カラムに含まれているイメージを指定のファイルにコピーします。

Adaptive Server から text または image 値を取得するには 2 つの方法があります。

- *exampl11.c* では、値を含んでいるローを選択し、dbnextrow を使用してそのローを処理します。dbnextrow が呼び出されると、dbdata を使用して、返されたイメージへのポインタを返すことができます。
- また、dbmoretext と一緒に dbreadtext を使用して、text または image 値をさらに小さないくつかのまとまりとして読み込むこともできます。dbreadtext の詳細については、『Open Client DB-Library/C リファレンス・マニュアル』を参照してください。

注意　このサンプル・プログラムを実行するには、pubs2 データベースを格納している Adaptive Server にアクセスする必要があります。

国際言語ルーチン

exampl12.c は、pubs2 データベースからデータを取り出して us_english フォーマットで出力します。

注意　このサンプル・プログラムを実行するには、Adaptive Server と pubs2 データベースにアクセスする必要があります。

バルク・コピーのサンプル・プログラム

バルク・コピーのサンプル・プログラム *bulkcopy.c* は、バルク・コピー・ルーチンを使用して、Adaptive Server の数種のデータ型を含んでいる新しく作成されたテーブルにホスト・ファイルからデータをコピーします。

注意　このサンプル・プログラムを実行するには、Adaptive Server にアクセスする必要があります。create database と create table パーミッションを持っていることも必要です。

2 フェーズ・コミットのサンプル・プログラム

サンプル・プログラム *twophase.c* は、2 つの異なるサーバに対して簡単な更新を実行します。実際の更新内容については、ソース・コードを参照してください。このサンプル・プログラムを実行すると、各サーバに対して isql を使用して、更新が実際に行われたかどうかを調べることができます。

このサンプル・プログラムでは、“SERVICE” と “PRACTICE” という 2 つの Adaptive Server が稼働していて、それぞれが pubs2 データベースを格納していることを前提としています。使用するサーバの名前がこれと異なる場合は、ソース・コードの “SERVICE” と “PRACTICE” をサーバの実際の名前で置き換えてください。

このサンプル・プログラムを実行する前に、*interfaces* ファイルが両方のサーバについて適切なエントリを持っていることを確認してください。*interfaces* ファイルについては、『Open Client DB-Library/C リファレンス・マニュアル』と『Open Client/Server 設定ガイド Windows 版』を参照してください。

“PRACTICE” サーバが “SERVICE” サーバとは異なるマシン上に存在する場合は、そのマシン上の *interfaces* ファイルも “SERVICE” クエリ・ポート用のエントリを持っている必要があります。詳細については、『Open Client DB-Library/C リファレンス・マニュアル』を参照してください。

第 4 章

Open Server Server-Library/C のサンプル・プログラム

Open Server Server-Library は、クライアント／サーバ・アーキテクチャの機能を利用するサーバを設計するために使用します。これらの機能には、RPC (リモート・プロシージャ・コール)、マルチスレッド処理、Adaptive Server へのリモート・アクセスなどがあります。この章で説明するサンプル・プログラムはこれらの機能の例を示すものです。

この章の内容は、次のとおりです。

Server-Library を使用すると、完全なスタンドアロン・サーバを作成できます。isql などの Open Client ベースのアプリケーションとユーティリティは、これらのサーバに組み込まれた機能を利用します。

サンプル・プログラムの使い方

サンプル・プログラムは、Server-Library/C 固有の機能の例を示しています。これらのプログラムは Server-Library/C のトレーニング用としてではなく、アプリケーション・プログラマのためのガイドとして設計されています。サンプル・プログラムを使用する前に、各ソース・ファイルの先頭にある説明を読んで、ソース・コードの内容を確認してください。

注意 これらの簡単なプログラムは、実際の運用環境で使用するために作成されているものではありません。運用環境で使用するプログラムには、エラー処理や特殊ケースの処理のためのコードを追加する必要があります。

作業を始める前に

Open Server サンプル・プログラムを実行する前に、次の手順に従います。

1 SYBASE 環境変数に Sybase リリース・ディレクトリのパスを設定していない場合は、設定します。

2 DSLISTEN 環境変数に、使用するサーバの名前を設定します。

3 *makefile* を使用して、*example_name* というサンプル実行プログラムを作成します。

使用する環境と変数の設定の詳細については、『Open Client/Server 設定ガイド Windows 版』を参照してください。

ロケーションと内容

Server-Library に付属しているサンプル・ファイルは Sybase インストール・ディレクトリ下の *¥sample¥srvlib* サブディレクトリ内に配置されています。

srvlib ディレクトリには、次のファイルが格納されています。

- サンプル・プログラムのオンライン・ソース・コード
- *README* － サンプル・プログラムの構築、実行、トラブルシューティングについて、プラットフォーム固有の一般的な注意が記述されたテキスト・ファイル
- *makefile* － サンプル・プログラムを構築するために使用するよう提供されています。Open Server アプリケーションの作成を開始するときにこの *makefile* を使用してください。
- SRV_CONNECT イベント・ハンドラ
- エラー・ハンドラ

注意 サンプル・プログラムが常駐するサブディレクトリの内容のバックアップ・コピーを作成してください。これによって、もとのファイルの整合性に影響を与えずにサンプル・プログラムを使用することができます。

トレース

トレース機能は、アプリケーションによって実行されるアクティビティに関して、選択したオプションに従って詳細な情報を提供します。Open Server のサンプル・プログラムは、トレース機能をサポートしており、トレース出力を Open Server ログ・ファイルに送ります。トレース機能を使用できるようにするには、サンプル・プログラムを実行するときにコマンド・ラインに次のようなオプションを指定します。

```
example_name [normal_sample_options]
[-h] [-d] [-i] [-a] [-m] [-t] [-e] [-q] [-n]
```

表 4-1 は、各オプションが提供するトレース情報のタイプの説明です。

表 4-1: トレース・オプション

オプション	説明
-h	TDS ヘッダ
-d	TDS データ
-i	I/O
-a	アテンション
-m	メッセージ・キュー
-t	TDS トークン
-e	イベント・トレース
-q	遅延イベント・キュー
-n	Net-Library トレース

注意　-e と -q を同時に指定することはできません。

ヘッダ・ファイル

次のヘッダ・ファイルは、すべての Open Server アプリケーションで必須です。

- *ospublic.h* − パブリックな Open Server の構造体、データ型定義、定義文、関数プロトタイプが含まれています。
- *oserror.h* − Open Server のエラー・メッセージの番号とテキストが含まれています。
- *oscompat.h* − 古いデータ型定義、データ型、ルーチン、定数、関数プロトタイプの、新しいバージョンへのマップが含まれています。

ヘッダ・ファイルの詳細については、『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。

Server-Library のサンプル・プログラム

この項では、Server-Library に付属しているサンプル・プログラムについて説明します。これらのサンプル・プログラムは、C 言語プログラムでの Server-Library ルーチンの一般的な使用法の例を示すものです。

サンプル・プログラムは C ソース・ファイルです。Server-Library のサンプル・プログラムを使用する場合やアプリケーションを構築する場合は、使用するプラットフォーム上に適切なコンパイラをインストールする必要があります。Sybase がサポートしているコンパイラの詳細については、「第 1 章 Open Client と Open Server のアプリケーションの構築」を参照してください。

サンプル・プログラムのテスト

サンプル・プログラムをテストするには、次の手順に従います。

1 isql を使用して *server_name* に接続します。

```
isql -Uusername -Ppassword -S server_name
```

2 次のクエリ (または任意の SQL コマンド) を入力します。

```
select * from sysservers
```

これで任意の SQL コマンドを実行できるようになります。

クライアント・プログラムは、Adaptive Server から stop_serv レジスタード・プロシージャを実行することによってサンプル・プログラムを停止できます。

Open Server 入門

osintro.c サンプル・プログラムは、Open Server アプリケーションを構築するための基本的なコンポーネントの例を示すものです。*osintro.c* には言語ハンドラは含まれていないので、isql から渡されるコマンドを読み込むことはできません。

ゲートウェイ Open Server

ctosdemo.c は、ゲートウェイ Open Server のアプリケーションです。このプログラムは、Server-Library 呼び出しと Client-Library 呼び出しを使用します。まず、クライアントからコマンドを受け取って、リモート Adaptive Server に渡し、次にリモート・サーバから結果を取り出して、クライアントに渡します。*ctosdemo.c* は、次のさまざまなクライアント・コマンドを処理します。

- バルク・コピー・コマンド
- カーソル・コマンド
- 動的 SQL コマンド

- 言語コマンド
- メッセージ・コマンド
- オプション・コマンド
- RPC (リモート・プロシージャ・コール)

ctosdemo.c はさらに、SRV_ATTENTION イベント・ハンドラを呼び出すことによってクライアントからのアテンション要求に応答します。このプログラムには、各タイプのクライアント・コマンドを処理するためにイベント・ハンドラ・ルーチンが含まれています。

注意 Server-Library に付属している他のサンプル・プログラムとは違って、*ctosdemo.c* は完成品に近いものです。このサンプル・プログラムは、運用環境で使用できるコーディング・テンプレートとして提供されています。*ctosdemo.c* プログラムを終了させるには、コマンド・ウィンドウから [CTRL+C] キーを押します。*README* ファイル内のコマンドは間違っています。

ゲートウェイの詳細については、『Open Server Server-Library/C リファレンス・マニュアル』を参照してください。

srv_language イベント・ハンドラ

lang.c は、SRV_LANGUAGE イベント・ハンドラの使用例を示すものです。このイベント・ハンドラは、情報メッセージを使用してクライアントの言語コマンドに応答します。このとき、イベント・ハンドラは srv_sendinfo ルーチンを使用して情報メッセージをクライアントに送信します。このプログラムには、SRV_CONNECT イベント・ハンドラとエラー・ハンドラも含まれています。

言語コマンドを処理する方法の詳細については、『Open Server Server-Library/C リファレンス・マニュアル』の「言語呼び出し」の項を参照してください。

TDS パススルー・モード

fullpass.c は、TDS (Tabular Data Stream™) passthrough モードを使用する Open Server ゲートウェイ・アプリケーションです。詳細については、『Open Server Server-Library/C リファレンス・マニュアル』の「パススルー・モード」の項を参照してください。

イベント・ハンドラ・ルーチンは srv_recvpassthru を使用してクライアント要求を受け取り、ct_sendpassthru ルーチンを使用してこの情報を Adaptive Server に転送します。クライアント・コマンド全体がリモート・サーバに転送されると、イベント・ハンドラは ct_recvpassthru を使用してリモート・サーバから結果を読み込み、srv_sendpassthru を使用してクライアントに渡します。

また、このアプリケーションには、SRV_CONNECT イベント・ハンドラも含まれています。このハンドラは、srv_getloginfo と ct_setloginfo を使用して、クライアント接続情報をリモート・サーバに転送します。次に ct_getloginfo と srv_setloginfo を使用して、接続確認情報をクライアントに渡します。TDS passthrough モードを使用するすべての Open Server アプリケーションは、その SRV_CONNECT イベント・ハンドラ内にこれらの呼び出しを含んでいる必要があります。

注意 このアプリケーションを実行するには、Adaptive Server にアクセスする必要があります。

レジスタード・プロシージャ

regproc.c は、Open Server 11.1 以降でのレジスタード・プロシージャの使用例を示すものです。このアプリケーションは起動時にいくつかのプロシージャを登録してからクライアントのコマンドを待ちます。Open Server イベント・ハンドラはインストールされません。

クライアントは RPC コマンドを送信して、*regproc.c* に定義されているレジスタード・プロシージャを実行します。

regproc.c で使用するためのクライアント・プログラムが、次のようにいくつか追加されています。

- *version.c* − Open Server のバージョン番号をクライアントに返すレジスタード・プロシージャ (rp_version) を実行します。
- *dbwait.c* − DB-Library で実装されており、レジスタード・プロシージャ rp_version が実行されるときにクライアントに通知するように Open Server に要求します。
- *ctwait.c* − Client-Library で実装されており、レジスタード・プロシージャ rp_version が実行されるときにクライアントに通知するように Open Server に要求します。

国際化言語と文字セット

intlchar.c サンプル・プログラムは、Open Server が国際言語と文字セットを処理する方法の例を示すものです。このサンプル・プログラムは、Open Server アプリケーションのネイティブ言語と文字セット用の値を初期設定して、クライアント要求に応答してこれらの値を変更します。

クライアント要求は、オプション・コマンドと言語コマンドのフォーマットで渡されます。*intlchar.c* は、SRV_OPTION イベント・ハンドラと SRV_LANGUAGE イベント・ハンドラ、および SRV_CONNECT ハンドラをインストールします。

マルチスレッド・プログラミング

multthrd.c プログラムは、次のようないくつかの Open Server マルチスレッド・プログラミング機能の例を示します。

- srv_spawn によるサービス・スレッドの作成
- クライアント接続スレッドとサービス・スレッド間でのメッセージ・キューによるスレッド間通信 (srv_getmsgq と srv_putmsgq を使用)
- スリープ・メカニズムとウェイクアップ・メカニズム (srv_sleep と srv_wakeup を使用)
- スケジューリング情報をレポートするためのコールバック・ルーチンの使用 (srv_callback を使用)

multthrd.c は、SRV_START ハンドラ、SRV_LANGUAGE ハンドラ、SRV_CONNECT ハンドラ、コールバック・ハンドラをインストールします。サービス・スレッドは、Open Server アプリケーションが受け取るすべての言語クエリのログを取ります。実行される内容は次のとおりです。

- アプリケーションの言語ハンドラでは、クライアント・スレッドはクライアントからクエリを読み込み、メッセージ・データとしてクエリを使用してメッセージを「ロガー」というサービス・スレッドに送信します。
- 送信後、クライアント・スレッドは待機します (srv_sleep)。サービス・スレッドは、メッセージを受け取るとクライアント・スレッドをウェイクアップします (srv_wakeup)。
- ロガーは、継続的にループしてメッセージを待ちます。メッセージを受け取ると、ロガーはクエリの内容をファイルに書き込んで送信側に通知します。
- ロガーとクライアント・スレッドは、SRV_C_RESUME、SRV_C_SUSPEND、SRV_C_TIMESLICE、SRV_C_EXIT コールバック・ハンドラをインストールして、スケジュール情報を出力します。

セキュリティ・サービス

secsrv.c サンプル・プログラムは、Open Server のネットワーク・ベースのセキュリティ・サービスの使用例を示します。

このサンプル・プログラム内の接続ハンドラは、クライアント・スレッドのセキュリティ・プロパティを取り出し、そのセッションでどのセキュリティ・サービスがアクティブになっているかを示すメッセージをクライアントに送信します。

セキュリティ・サービスの詳細については、『Open Client/Server 設定ガイド Windows 版』を参照してください。

第 5 章

Open Client Embedded SQL/C

Embedded SQL は、C などの言語で作成されたアプリケーション・プログラム内に Transact-SQL®文を埋め込むための Transact-SQL のスーパーセットです。Embedded SQL には、すべての Transact-SQL 文に加えて、アプリケーション・プログラムで Transact-SQL を使用するために必要な拡張機能が含まれています。

この章の内容は、次のとおりです。

cpre プリコンパイラでプリコンパイルされている Embedded SQL/C アプリケーションは、SQL Server データベースに格納されているデータの検索、挿入、変更を行うための簡単な方法を提供します。Embedded SQL/C のプログラミング技法については、『Open Client Embedded SQL/C プログラマーズ・ガイド』を参照してください。

Embedded SQL/C 実行プログラムの構築

Embedded SQL アプリケーションから実行プログラムを構築する基本手順は次のとおりです。

1 アプリケーションをプリコンパイルします。

2 プリコンパイラによって生成された C ソース・コードをコンパイルします。

3 アプリケーションを必要に応じてオブジェクトやライブラリとリンクします。

4 プリコンパイラによって生成されたストアド・プロシージャをロードします。

次の項では各手順について説明します。

アプリケーションのプリコンパイル

次のように、Embedded SQL/C コードをプリコンパイルしてから、アプリケーションをコンパイルしてください。

```
cpre    [/a] [/b] [/c] [/d] [/e] [/f] [/l] [/m]      [/p] [/r] [/s] [/v] [/w] [/x] [/y]
    [/Ccompiler]
    [/Ddatabase_name]
    /Ffips_level]
    [/G[isql_file_name]]
    [/Iinclude_directory]...
    [/Jlocale_for_charset]
    [/Ksyntax_level]
    [/L[listing_file_name]]
    [/Mlabelname=labelvalue]
    [/Nsql.ini]
    [/Otarget_file_name]
    [/P[password]]
    [/Sserver_name]
    [/Ttag_id]
    [/Uuser_id]
    [/Vversion_number]
    [/Zlocale_for_messages]
 program[.ext] [program[.ext]]...
```

注意 オプションは、スラッシュ (/) またはハイフン (-) のどちらかを使ってフラグできます。
したがって、cpre -I と cpre /I は同じことを表します。

正しくないオプションを指定した場合は、プリコンパイラは使用可能なオプションのリストを表示します。

program には、Embedded SQL/C ソース・ファイルの名前を指定します。*program* のデフォルトの拡張子は *.pc* です。cpre 文を使用すると、拡張子が *.c* の出力ファイルが生成されます。

オプションのなかには、プリコンパイラの機能 (たとえばストアド・プロシージャの生成) をアクティブにするスイッチとして動作するものもあります。デフォルトでは、これらの機能は無効になっています。これらの機能を有効にするには、cpre コマンド・ラインでオプションを指定します。このほかの文修飾子は、パスワードなど、プリプロセッサに対する値を指定します。値はオプションのあとに入力します (間にスペースを入れても入れなくてもかまいません)。

cpre オプションの詳細については、「付録 B プリコンパイラ・リファレンス」を参照してください。

アプリケーションのコンパイルとリンク

Embedded SQL バージョン 11.1 以降は、Windows プラットフォームの Microsoft C コンパイラと、Windows NT の Borland C コンパイラを使用して動作確認されています。コンパイルとリンク用の実際の構文については *makefile* を参照してください。*makefile* は、Sybase リリース・ディレクトリ下の *¥sample¥esqlc* サブディレクトリ内にあります。

リンク・ライブラリ

リンク行には次のライブラリを指定する必要があります。

- *libct* − Client-Library DLL
- *libcs* − CS-Library DLL
- *libtcl* − トランスポート制御層 DLL
- *libcomn* − 内部共有ライブラリ DLL
- *libintl* − ローカライゼーション・サポート・ライブラリ DLL

ストアド・プロシージャのロード

Embedded SQL/C プログラムを実行する前に、プリコンパイラで生成されたストアド・プロシージャを Adaptive Server にロードしておかなければなりません。このためには、-G オプションを使用してプログラムをプリコンパイルする必要があります。-G オプションは isql スクリプト・ファイルを作成します。次に isql -i オプションを使用して、作成されたファイルをロードします。

isql の詳細については、「付録 A ユーティリティ」を参照してください。

Embedded SQL/C のサンプル・プログラム

Embedded SQL には、一般的な Embedded SQL アプリケーションの例を示す 2 つのオンライン・サンプル・プログラムが含まれています。サンプル・プログラムは Sybase リリース・ディレクトリ下の *¥sample¥esqlc* サブディレクトリ内にあります。このサブディレクトリには、*README* ファイルと *makefile* も含まれています。次の項では、これらのサンプル・プログラムの概要について説明します。

サンプル・プログラムは、Embedded SQL/C 固有の機能の例を示しています。これらのプログラムは、Embedded SQL/C のトレーニング用ではなく、アプリケーション・プログラマのためのガイドとして設計されています。サンプル・プログラムを使用する前に、各ソース・ファイルの先頭にある説明を読んで、ソース・コードの内容を確認してください。これらの簡単なプログラムは、実際の運用環境で使用するために作成されているものではありません。運用環境で使用するプログラムには、エラー処理のためのコードを追加する必要があります。

作業を始める前に

Embedded SQL/C のサンプル・プログラムを実行するには、SYBASE 環境変数に Sybase インストール・ディレクトリのパスを設定します (このパスがまだ設定されていない場合)。

sql.ini ファイルを調べて、使用されるサーバ名のエントリが存在することを確認します。*sql.ini* ファイルを調べるには、dsedit を使用します。*sql.ini* ファイルにサーバを追加した場合は、『Open Client/Server 設定ガイド Windows 版』で説明しているように、ocscfg を使用すると各サーバへの接続テストを実行できます。

サンプル・プログラムを実行するには、pubs2 データベースを格納している Adaptive Server にアクセスする必要があります。pubs2 データベースをインストールする方法については、『ASE インストール・ガイド』を参照してください。

サンプル・プログラムをプリコンパイルする前に、次に示すようにサンプル・ヘッダ・ファイルを編集し、ユーザ名とパスワードを、使用している SQL Server に有効な値に置き換えておく必要があります。プログラム内のコメントに、変更が必要な箇所が示されています。

ヘッダ・ファイル

すべてのサンプル・プログラムは、サンプル・ヘッダ・ファイル *sybsqlex.h* を参照します。*sybsqlex.h* の内容は、次のとおりです。

```
/**********************************************
 *                                            *
 *  sybsqlex.h - header file for Embedded SQL/C    *
 *               examples                     *
 *                                            *
 **********************************************/


 #define USER            "sa"

 #define PASSWORD""


#define ERREXIT     -1
 #define STDEXIT     0
```

すべてのサンプル・プログラムには、次の行が含まれています。

```
#include "sybsqlex.h"
```

sybsqlex.h 内では、USER は “username” として定義され、PASSWORD は “password” として定義されています。サンプル・プログラムを実行する前に *sybsqlex.h* を編集します。“username” を Adaptive Server のログイン名に、“password” を Adaptive Server のパスワードに変更してください。

データベース・クエリのためのカーソルの使い方

example1 は、対話型クエリ・プログラムでのカーソルの使用例を示すものです。このプログラムは次のように動作します。

- 本のタイプのリストを表示します。ユーザはタイプを 1 つ選択します。
- 選択されたタイプの本のすべてのタイトルを表示して、タイトル ID を要求します。
- 選択されたタイトルについての詳細情報を表示し、さらにタイトル ID を要求します。
- プロンプト画面で [Return] キーが押されると終了します。

テーブルのローの表示と編集

example2 は、カーソルを使用してローを更新する方法を示しています。このプログラムは次のように動作します。

- 作家テーブル内のカラムをローごとに表示します。
- ユーザは au_id カラムを除くすべてのカラム内の作家情報を更新できます。ユーザがカラム情報に対して [Return] キーを押した場合は、そのカラムのデータは変更されないでもとのままになります。
- ユーザが更新を確認すると、データを Adaptive Server に送信します。

第 6 章

Open Client Embedded SQL/COBOL

Embedded SQL は Transact-SQL のスーパーセットであり、COBOL 言語などで作成されるアプリケーション・プログラムに Transact-SQL 文を埋め込むことができます。Embedded SQL には、すべての Transact-SQL 文に加えて、アプリケーション・プログラムで Transact-SQL を使用するために必要な拡張機能が含まれています。

この章の内容は、次のとおりです。

cobpre プリコンパイラでプリコンパイルされている Embedded SQL/COBOL プログラムを使用すると、SQL Server/Adaptive Server データベースに格納されているデータを簡単に取得、挿入、変更することができます。

Embedded SQL/COBOL 実行プログラムの構築

Embedded SQL アプリケーションから実行プログラムを構築するには、次の手順に従います。

1 アプリケーションをプリコンパイルします。

2 プリコンパイラによって生成された COBOL ソース・コードをコンパイルします。

3 アプリケーションを必要に応じてファイルやライブラリとリンクします。

4 プリコンパイラによって生成されたストアド・プロシージャをロードします。

次の項ではこれらの手順について説明します。

アプリケーションのプリコンパイル

Embedded SQL ソース・プログラムをプリコンパイルするための構文は、次のとおりです。

```
cobpre    [/a] [/b] [/c] [/d] [/e] [/f] [/l]
     [/m] [/r] [/s] [/v] [/w] [/x] [/y]
    [/Ccompiler]
    [/Ddatabase_name]
    [/Ffips_level]
    [/G[isql_file_name]]
    [/Iinclude_directory]...
    [/Jlocale_for_charset]
    [/Ksyntax_level]
    [/L[listing_file_name]]
    [/Mlabelname=labelvalue]
    [/Nsql.ini]
    [/Otarget_file_name]
    [/P[password]]
    [/Sserver_name]
    [/Ttag_id]
    [/Uuser_id]
    [/Vversion_number]
    [/Zlocale_for_messages]
  program[.ext] [program[.ext]]...
```

注意 オプションは、スラッシュ (/) またはハイフン (-) のどちらかを使ってフラグできます。
したがって、cobpre -l と cobpre /l は同じことを表します。

正しくないオプションを指定した場合は、プリコンパイラは使用可能なオプションのリストを表示します。

program は、Embedded SQL/COBOL ソース・ファイルの名前です。ソース・ファイル名の指定は必須です。*program* のデフォルトの拡張子は *.pco* です。cobpre を実行すると、拡張子が *.cbl* の出力ファイルが生成されます。

オプションの一部には、ストアド・プロシージャの生成などの、プリコンパイラの機能を有効にするためのスイッチもあります。デフォルトでは、これらの機能は「オフ」になっています。これらの機能を「オン」にするには、cobpre 文の行にオプションを指定します。このほかの文修飾子は、パスワードなど、プリプロセッサに対する値を指定します。値はオプションのあとに入力します (間にスペースを入れても入れなくてもかまいません)。

cobpre オプションの詳細については、「付録 B プリコンパイラ・リファレンス」を参照してください。

アプリケーションのコンパイルとリンク

Embedded SQL バージョン 11.1 以降は、MicroFocus net Express 3.1 で動作確認されています。コンパイルとリンク用の実際の構文については、Sybase インストール・ディレクトリ下の *¥sample¥esqlcob* ディレクトリ内にある *makefile* を参照してください。

リンク・ライブラリ

リンク・コマンド行には次に示すライブラリの一部または全部を指定する必要があります。

- *libcobct* － Client-Library への COBOL インタフェース
- *libct* － Client-Library DLL
- *libcs* － CS-Library DLL
- *libtcl* － トランスポート制御層 DLL
- *libcomn* － 内部共有ライブラリ DLL
- *libintl* － ローカライゼーション・サポート・ライブラリ DLL

ストアド・プロシージャのロード

プリコンパイル時に -G オプションを使用した場合は、プリコンパイラで生成されたストアド・プロシージャを Adaptive Server にロードする必要があります。この作業は、isql -i オプションを使用して実行できます。

isql の詳細については、「付録 A ユーティリティ」を参照してください。

Embedded SQL/COBOL のサンプル・プログラム

Embedded SQL には、一般的な Embedded SQL アプリケーションの例を示す 2 つのサンプル・プログラムが含まれています。サンプル・プログラムは Sybase インストール・ディレクトリ下の *¥sample¥esqlcob* サブディレクトリ内にあります。

同じディレクトリ内の *README* ファイルには、サンプル・プログラムを構築して実行するための手順とそれらを使用するときの注意が記載されています。*COBOL.pco* ファイルは、SQL Server/Adaptive Server のログイン名とパスワードを定義します。サンプル・プログラムをコンパイルする前に、このファイルの中にあるログイン情報を更新してください。

System 11 のサンプル・プログラムは、Embedded SQL/COBOL の特定の機能についての例を示すものです。これらのプログラムは、Embedded SQL/COBOL のトレーニング用ではなく、アプリケーション・プログラマのためのガイドとして設計されています。サンプル・プログラムを使用する前に、各ソース・ファイルの先頭にある説明を読んで、ソース・コードの内容を確認してください。これらの簡単なプログラムは、実際の運用環境で使用するために作成されているものではありません。運用環境で使用するプログラムには、エラー処理のためのコードを追加する必要があります。

一般的な条件

Embedded SQL/COBOL のサンプル・プログラムを実行するには、SYBASE 環境変数に Sybase インストール・ディレクトリのパスを設定します (このパスがまだ設定されていない場合)。

サンプル・プログラムを実行するには、pubs2 サンプル・データベースがインストールされている SQL Server/Adaptive Server にアクセスする必要があります。pubs2 データベースをインストールする方法については、『ASE インストール・ガイド』を参照してください。

プログラムをプリコンパイルする前に、ユーザ名とパスワードを SQL Server/Adaptive Server で有効な値に置き換えてください。変更箇所についてはプログラム内のコメントを参照してください。

注意 サンプル・プログラムの結果を表示するには、[Return] キーを押す必要があります。

MicroFocus COBOL 用環境変数の追加

UNIX プラットフォームでは、COBOL のホーム・ディレクトリ *$COBDIR/coblib* をライブラリ・パス環境変数に追加してから、Embedded SQL/COBOL サンプル・プログラムを実行します。

データベース・クエリのためのカーソルの使い方

example1 は、対話型クエリ・プログラムでのカーソルの使用例を示すものです。このプログラムは次のように動作します。

- 本のタイプのリストを表示します。ユーザはタイプを 1 つ選択します。
- 選択されたタイプの本のすべてのタイトルを表示して、タイトル ID を要求します。
- 選択されたタイトルについての詳細情報を表示し、さらにタイトル ID を要求します。
- プロンプト画面で [Return] キーが押されると、プログラムは終了します。

テーブルのローの表示と編集

example2 は、カーソルを使用してローを更新する方法を示しています。このプログラムは次のように動作します。

- 作家テーブル内のカラムをローごとに表示します。
- ユーザは au_id カラムを除くすべてのカラム内の作家情報を更新できます。ユーザがカラム情報に対して [Return] キーを押した場合は、そのカラムのデータは変更されないでもとのままになります。
- ユーザが更新を確認すると、データを SQL Server/Adaptive Server に送信します。

付 録　A

ユーティリティ

この付録では、次のユーティリティについて説明します。

ユーティリティ	説明	ページ
bcp	ユーザ指定のフォーマットで、データベース・テーブルをオペレーティング・システム・ファイルに、またはオペレーティング・システム・ファイルをデータベース・テーブルにコピーする、バルク・コピー・ユーティリティです。	63
defncopy	指定されたビュー、ルール、デフォルト、トリガ、プロシージャ、レポートの定義をデータベースからオペレーティング・システム・ファイルに、またはオペレーティング・システム・ファイルからデータベースにコピーする、定義コピー・ユーティリティです。	77
isql	Adaptive Server または Open Server に接続してクエリを実行する対話型 SQL パーサです。	82

bcp、defncopy、isql ユーティリティの代わりとして、Open Client は Windows NT のユーザに SQL Advantage を提供します。詳細については、『SQL Advantage User's Guide』を参照してください。

bcp

説明

ユーザが指定したフォーマットで、データベース・テーブルをオペレーティング・システム・ファイルに、またはオペレーティング・システム・ファイルからデータベース・テーブルにコピーします。

構文

```
bcp [[database_name.]owner.]table_name {in | out}
    datafile
[-c] [-E] [-n] [-v] [-X]
[-a display_charset]
[-A size]
[-b batchsize]
[-e errfile]
[-f formatfile]
[-F firstrow]
[-I interfaces_file]
[-J client_charset]
[-K keytab_file]
[-L lastrow]
[-m maxerrors]
[-P password]
[-q datafile_charset]
[-r row_terminator]
```

[-R *remote_server_principal*]
[-S *server*]
[-t *field_terminator*]
[-T *text_or_image_size*]
[-U *username*]
[-V [*security_options*
[-Y]
[-z *language*]
[-Z *security_mechanism*]

パラメータ

database_name

コピーするテーブルがデフォルト・データベース内または *master* 内にある場合は、このパラメータはオプションとして使用できます。そうでない場合は、データベース名を指定しなければなりません。

owner

ユーザまたはデータベース所有者がコピーするテーブルを所有している場合は、このパラメータはオプションとして使用できます。所有者を指定しないと、bcp はまず、ユーザが所有するこの名前のテーブルを探します。次にデータベース所有者が所有するテーブルを探します。それ以外のユーザがテーブルを所有している場合は、所有者の名前を指定しなければなりません。指定しないと、コマンドは失敗します。

table_name

コピーするデータベース・テーブルまたはデータベース・ビューの名前です。

in | out

コピーの方向を示します。in は、ファイルからデータベース・テーブルへのコピーであることを示し、out は、データベース・テーブルからファイルへのコピーであることを示します。

datafile

オペレーティング・システム・ファイルのフル・パス名です。パス名は、1 ～ 255 文字で指定します。

-a *display_charset*

bcp を実行しているマシンの文字セットと異なる文字セットを使用する端末から、bcp を実行できます。-a と -J をともに使用すると、変換に必要な文字セット変換ファイル (*.xlt* ファイル) が指定できます。-a を使用するときに -J を省略できるのは、クライアントの文字セットがデフォルトの文字セットと同じ場合だけです。

-A size

この bcp セッションで使用するネットワーク・パケット・サイズを指定します。次に例を示します。

```
bcp -A 2048
```

この例では、bcp セッションのパケット・サイズを 2048 バイトに設定します。*size* には、default network packet size 設定変数と maximum network packet size 設定変数の間の値であり、512 の倍数である必要があります。

大量のバルク・コピー・オペレーションのパフォーマンスを向上させるには、デフォルトよりも大きなネットワーク・パケット・サイズを使用します。

-b *batchsize*

バッチごとにコピーされるデータのロー数です (デフォルトでは、1 つのバッチ処理ですべてのローがコピーされます)。バッチ処理は、バルク・コピー・インの場合にだけ適用されます。バルク・コピー・アウトには適用されません。

-c

char データ型をデフォルトとして使用してコピー操作を実行します。このオプションは各フィールドの入力を要求しません。デフォルトの記憶タイプとして *char* データ型を使用し、プレフィクスなしで、デフォルトのフィールド・ターミネータとして ¥t (タブ)、デフォルトのロー・ターミネータとして ¥n (復帰改行文字) を使用します。

-e *errfile*

bcp がファイルからデータベースに転送できなかったローを保管する、エラー・ファイルのフル・パス名です。bcp プログラムからのエラー・メッセージは、使用している端末に表示されます。bcp がエラー・ファイルを作成するのは、このオプションが指定されたときだけです。

-E

テーブルの IDENTITY カラムの値を明示的に指定します。

デフォルトでは、IDENTITY カラムを持つテーブルにデータをバルク・コピーすると、bcp は IDENTITY カラムの一時的な値として 0 を各ローに割り当てます。bcp が各ローをテーブルに挿入するときに、サーバは 1 から始まる連続した、ユニークな IDENTITY カラムの値をローに割り当てます。データをテーブルにコピーするときに -E フラグを指定すると、bcp は各ローに対する IDENTITY カラム値を明示的に入力するように要求します。挿入されるローの数が IDENTITY カラム値の最大値を超える場合、Adaptive Server はエラーを返します。

デフォルトでは、IDENTITY カラムを持つテーブルからデータをバルク・コピーすると、bcp は、カラムに関するすべての情報を出力ファイルから取り除きます。-E フラグを指定すると、bcp は既存の IDENTITY カラム値を出力ファイルにコピーします。

-f *formatfile*

同一テーブル内で前回の bcp 実行時の応答が保管してあるファイルのフル・パス名です。bcp に対して使用するフォーマットを入力すると、bcp はフォーマット・ファイルとしてその形式を保存するかどうかを尋ねてきます。フォーマット・ファイルの作成はオプションです。デフォルトのファイル名は、*bcp.fmt* です。bcp プログラムは、データのコピー時にフォーマット・ファイルを参照することができます。したがって、前回と同じフォーマット応答を再度対話的に繰り返す必要がなくなります。このオプションは、コピー・インまたはコピー・アウトに対して以前に作成したフォーマット・ファイルを、今回も使用する必要があるときにだけ使用します。このオプションを指定していない場合、bcp はフォーマット情報を対話的に問い合わせてきます。

-F *firstrow*

コピーを開始する最初のローの番号を指定します (デフォルトは 1 番目のローです)。

-I *interfaces_file*

Adaptive Server に接続するときに検索する interfaces ファイルの名前とロケーションを指定します。-I を指定していなければ、bcp は、SYBASE 環境変数によって指定されたディレクトリ下の *ini* ディレクトリ内にある interfaces ファイル (Windows プラットフォームでは *sql.ini*) を探します。

-J *client_charset*

クライアントで使用する文字セットを指定します。bcp は、フィルタを使用して *client_charset* と Adaptive Server 文字セット間で入力を変換します。

-J *client_charset* は、クライアントで使用する文字セットである *client_charset* とサーバの文字セット間の変換を Adaptive Server に要求します。

-J

引数を指定しないと、文字セット変換が無効になります。この場合、変換は行われません。クライアントとサーバが同じ文字セットを使用する場合に、このパラメータを使用してください。

-J の指定を省略すると、文字セットはプラットフォームのデフォルトに設定されます。

デフォルトの文字セットは、クライアントが使用する文字セットと同じであるとはかぎりません。文字セットおよび関連するフラグの詳細については、『システム管理ガイド』を参照してください。

-K *keytab_file*

DCE セキュリティにだけ使用できます。-U オプションによって指定されるユーザ名のセキュリティ・キーを含んでいる DCE *keytab* ファイルを指定します。*keytab* ファイルは、DCE の dcecp ユーティリティを使用して作成できます。詳細については、DCE のマニュアルを参照してください。

-K オプションを指定していない場合、bcp ユーザは -U オプションで指定するユーザ名と同じユーザ名を使用して DCE にログインしなければなりません。

-L *lastrow*

コピーを終了する最後のローの番号を指定します (デフォルトは最後のローです)。

-m *maxerrors*

bcp がコピーをアボートするまでに許容されるエラーの最大数を指定します。bcp は、構築できない各ローを 1 つのエラーとしてカウントして、除外します。このオプションを指定していない場合、bcp はデフォルト値の 10 を使用します。

-n

ネイティブの (オペレーティング・システムの) フォーマットを使用して、コピー・オペレーションを実行します。このオプションは各フィールドの入力を要求しません。ネイティブ・データ・フォーマットのファイルは、人間の目で判読できるフォーマットではありません。

警告！ データのリカバリ、サルベージ、または非常時の解析を実行するために bcp をネイティブ・フォーマットで使用しないでください。異なるハードウェア・プラットフォーム間、異なるオペレーティング・システム間、または異なるメジャー・リリースの Adaptive Server 間で、ネイティブ・フォーマットの bcp を使用してデータを転送しないでください。ネイティブ・フォーマットを使用して bcp を実行した場合、Adaptive Server に再ロードできないフラット・ファイルが作成され、データをリカバリできなくなることがあります。bcp を文字フォーマットで再実行できなければ、たとえば、テーブルのトランケートや削除、ハードウェアの損傷、データベースの削除などの場合に、データをリカバリできなくなります。

-P *password*

Adaptive Server のパスワードを指定します。-P *password* を指定しないと、bcp はパスワードを要求します。パスワードが NULL のときは、コマンド・ラインの最後に -P フラグを単独で指定します。

-q *datafile_charset*

bcp を実行して、クライアントの文字セットと異なる文字セットを使用するファイル・システムとの間で、文字データを双方向にコピーできるようになります。

日本語環境では、-q フラグは半角カタカナを全角カタカナに変換します。

注意 ascii_7 文字セットは、すべての文字セットと互換性があります。Adaptive Server またはクライアントの文字セットのどちらかが ascii_7 に設定されている場合、すべての 7 ビット ASCII 文字はクライアントとサーバ間でそのまま渡すことができます。その他の文字セットを使用している場合は、変換エラーが発生します。文字セットの変換の詳細については、『システム管理ガイド』を参照してください。

-r *row_terminator*

デフォルトのロー・ターミネータを指定します。

-R *remote_server_principal*

リモート・サーバのプリンシパル名を指定します。デフォルトでは、サーバのプリンシパル名はサーバのネットワーク名 (-S オプションまたは DSQUERY 環境変数で指定) と一致します。サーバのプリンシパル名とネットワーク名が同じでない場合は、-R オプションを使用する必要があります。

-S *server*
接続先の Adaptive Server の名前を指定します。-S を引数なしで指定すると、bcp は DSQUERY 環境変数で指定されるサーバを使用します。

-t *field_terminator*
デフォルトのフィールド・ターミネータを指定します。

-T *text_or_image_size*
Adaptive Server が送信する text または image データの最大長をバイト単位で指定します。デフォルトは 32K です。text または image フィールドが -T の値またはデフォルト値より大きい場合、bcp はオーバフロー部分を送信しません。

-U *username*
Adaptive Server ログイン名を指定します。*username* を指定していない場合、bcp は現在のユーザのオペレーティング・システム・ログイン名を使用します。

-v
bcp プログラムの現在のバージョンと版権メッセージを表示します。

-V *security_options*
ネットワーク・ベースのユーザ認証を指定します。このオプションを使用する場合、ユーザはユーティリティを実行する前にネットワークのセキュリティ・システムにログインする必要があります。この場合、ユーザは -U オプションにネットワーク・ユーザ名を指定する必要があり、-P オプションに指定されたパスワードは無視されます。

-V の後にキー文字オプションの *security_options* 文字列を続けると、他のセキュリティ・サービスを有効にできます。指定できるキー文字は、以下のとおりです。

c − データ機密性サービスを有効にする

i − データ整合性サービスを有効にする

m − 接続の確立に相互認証を有効にする

o − データ・オリジン・スタンピング・サービスを有効にする

q − 順序不整合の検出を有効にする

r − データ・リプレイの検出を有効にする

-X
サーバへの現在の接続で、アプリケーションがクライアント側のパスワード暗号化を使用してログインを開始するように指定します。bcp (クライアント) は、サーバに対してパスワードの暗号化が要求されていることを通知します。サーバは、bcp がパスワードを暗号化するために使う暗号化キーを返送し、パスワードを受け取ると、そのキーを使用してそのパスワードを確認します。

-Y

bcp IN の使用時に、サーバでの文字セット変換を無効にし、クライアント側で bcp によって文字セット変換を実行することを指定します。クライアント側での変換が有効になります。

注意 bcp OUT 使用時には、すべての文字セット変換はサーバで行われます。

-z *language*

サーバが bcp のプロンプトとメッセージの表示に使用する代替言語の公式名です。-z フラグを指定しないと、bcp はサーバのデフォルト言語を使用します。インストール時、またはインストールしたあとで Adaptive Server に言語を追加するには、langinstall ユーティリティまたは sp_addlanguage ストアド・プロシージャを使用します。

-Z *security_mechanism*

接続で使用するセキュリティ・メカニズムの名前を指定します。

セキュリティ・メカニズムの名前は、Sybase インストール・ディレクトリ下の *ini* サブディレクトリ内にある *libtcl.cfg* 設定ファイルに定義されています。*security_mechanism* で名前を指定しない場合は、デフォルトのメカニズムが使用されます。

セキュリティ・メカニズム名の詳細については、『Open Client/Server 設定ガイド Windows 版』の *libtcl.cfg* ファイルの説明を参照してください。

例

例 1 次の例は、-c オプションによって publishers テーブルからデータを文字フォーマットでコピーします (すべてのフィールドには char を使用します)。-t field_terminator オプションは各フィールドをカンマで終了し、-r row_terminator オプションは各ラインを改行文字で終了します。bcp はパスワードだけを要求します。

```
bcp pubs2..publishers out pub_out -c -t ,
 -r ¥r
```

例 2 次の例は、あとでデータを Adaptive Server に再ロードするために、bcp を使用して publishers テーブルから *pub_out* という名前のファイルにデータをコピーします。[Return] キーを押すと、プロンプト画面に表示されたデフォルトが使用されます。publishers テーブルにデータをコピーするときも、同じプロンプトが表示されます。

```
bcp pubs2..publishers out pub_out
 パスワード：

 フィールド pub_id のファイル記憶タイプを入力してください [char]：
 フィールド pub_id のプレフィクス長を入力してください [0]：
 フィールド pub_id の長さを入力してください [4]：
 フィールド・ターミネータを入力してください [none]：
 フィールド pub_name のファイル記憶タイプを入力してください [char]：
 フィールド pub_name のプレフィクス長を入力してください [1]：
 フィールド pub_name の長さを入力してください [40]：
 フィールド・ターミネータを入力してください [none]：
```

```
フィールド city のファイル記憶タイプを入力してください [char]：
フィールド city のプレフィクス長を入力してください [1]：
フィールド city の長さを入力してください [20]：
フィールド・ターミネータを入力してください [none]：

フィールド state のファイル記憶タイプを入力してください [char]：
フィールド state のプレフィクス長を入力してください [1]：
フィールド state の長さを入力してください [2]：
フィールド・ターミネータを入力してください [none]：
```

例 3 次の例は、-c オプションによって publishers テーブルからデータを文字フォーマットでコピーします (すべてのフィールドには char を使用します)。-t field_terminator オプションは各フィールドをカンマで終了し、-r row_terminator オプションは各ラインを改行文字で終了します。bcp はパスワードだけを要求します。

```
bcp pubs2..publishers out pub_out -c -t ,
 -r ¥r
```

例 4 次の例は、あとでデータを Adaptive Server に再ロードするために、bcp を使用して publishers テーブルから *pub_out* という名前のファイルにデータをコピーします。[Return] キーを押すと、プロンプト画面に表示されたデフォルトが使用されます。publishers テーブルにデータをコピーするときも、同じプロンプトが表示されます。

```
bcp pubs2..publishers out pub_out
 パスワード：

 フィールド pub_id のファイル記憶タイプを入力してください [char]：
 フィールド pub_id のプレフィクス長を入力してください [0]：
 フィールド pub_id の長さを入力してください [4]：
 フィールド・ターミネータを入力してください [none]：

 フィールド pub_name のファイル記憶タイプを入力してください [char]：
 フィールド pub_name のプレフィクス長を入力してください [1]：
 フィールド pub_name の長さを入力してください [40]：
 フィールド・ターミネータを入力してください [none]：

 フィールド city のファイル記憶タイプを入力してください [char]：
 フィールド city のプレフィクス長を入力してください [1]：
 フィールド city の長さを入力してください [20]：
 フィールド・ターミネータを入力してください [none]：

 フィールド state のファイル記憶タイプを入力してください [char]：
 フィールド state のプレフィクス長を入力してください [1]：
 フィールド state の長さを入力してください [2]：
 フィールド・ターミネータを入力してください [none]：
```

使用法

- bcp は、データベース・テーブルとオペレーティング・システム・ファイル間でデータを転送するための便利で高速な方法を提供します。bcp は、さまざまなフォーマットのファイルの読み込みと書き込みを行うことができます。ファイルからコピー・インする場合、bcp はデータを既存のデータベース・テーブルに挿入します。ファイルにコピー・アウトする場合は、bcp はファイルの以前の内容を上書きします。
- 処理を完了すると、bcp は、正常にコピーしたデータのロー数、コピーに要した合計時間、1 つのローのコピーに要した平均時間 (ミリ秒単位)、コピーしたロー数 (毎秒) を表示します。

System 11 用の bcp は、Client-Library を使用して構築されています。bcp のユーザ・インタフェースは、次の点以外は変更されていません。

- 接続でのネットワーク・ベースのセキュリティ・サービスを使用可能にする、次のようなコマンド・ライン・オプションが新しく追加されました。

  -K*keytab_file*
  -R*remote_server_principal*
  -V*security_options*
  -Z *security_mechanism*

- -y *sybase_directory* オプションは無視されます。
- エラー・メッセージのフォーマットが、以前のバージョンの bcp とは異なります。以前のメッセージの値に基づいてルーチンを実行するスクリプトがある場合は、それらの書き換えが必要な場合があります。

  たとえば、転送されたローの数を示す表示メッセージが変更されています。セッションの間、このバージョンの bcp は転送されたローの合計を定期的にレポートします。このメッセージは、以前のバージョンの bcp で出力されていたメッセージ “1000 rows transferred” に置き換わるものです。

注意　以前のバージョンの bcp を使用するには、*ocs.cfg* ファイルの [bcp] セクションの CS_BEHAVIOR プロパティを、次のように設定してください。

[bcp]

CS_BEHAVIOR = CS_BEHAVIOR_100

CS_BEHAVIOR が CS_BEHAVIOR_100 に設定されていない場合、bcp 11.1 以降の機能を使用することができます。

インデックスまたはトリガのあるテーブルのコピー

- bcp プログラムは、データと関連のあるインデックスやトリガを持たないテーブルにデータをロードするために最適化されています。bcp は、インデックスやトリガを使用せずに、ロギングを最小限にすることで、データをテーブルに最大限の速度でロードします。ページの割り付けはログを取られますが、ローの挿入はログを取られません。

  1 つまたは複数のインデックスやトリガを持っているテーブルにデータをコピーする場合は、bcp の低速バージョンが自動的に使用されて、ローの挿入をログに取ります。これには、create table 文の一意性制約を使用して暗黙的に作成されたインデックスも含まれます。しかし、bcp は、テーブルに定義された他の整合性制約は適用しません。

  bcp の高速バージョンはログを取らずにデータを挿入します。したがって、システム管理者やデータベース所有者は、最初にシステム・プロシージャ sp_dboption, “DB”, true を設定する必要があります。オプションが true でないときに、インデックスやトリガを持っていないテーブルにデータをコピーしようとすると、Adaptive Server はエラー・メッセージを表示します。データをファイルにコピー・アウトする場合や、インデックスやトリガを持っているテーブルにデータをコピー・インする場合は、このオプションを設定する必要はありません。

  注意 bcp は、インデックスまたはトリガを持っているテーブルに対する挿入をログに取るので、ログが非常に大きくなる可能性があります。dump transaction によってログをトランケートできるのは、バルク・コピーが完了し、dump database によるデータベースのバックアップが完了したあとです。

- select into/bulkcopy オプションがオンのときは、トランザクション・ログをダンプすることはできません。dump transaction を発行すると、エラー・メッセージが表示され、代わりに dump database を使用するように指示されます。

  警告！ select into/bulkcopy フラグをオフにする前にデータベースをダンプしてください。ログが取られていないデータをデータベースに挿入し、dump database を実行する前に dump transaction を実行した場合は、そのデータをリカバリすることはできません。

- ログが取られていない bcp の実行速度は、dump database が実行されている間は遅くなります。
- 表 A-1 では、コピー・インのときに bcp がどのバージョンを使用するのかを示し、select into/bulkcopy オプションに必要な設定を示しています。また、トランザクション・ログが保持されるかどうか、またダンプできるかどうかも示しています。

表 A-1: 高速 bcp と低速 bcp の比較

	select into/ bulkcopy が on の場合	select into/ bulkcopy が off の場合
高速 bcp	実行可能	bcp
(対象のテーブルにインデックスやトリガがない場合)	dump transaction の実行は不可	dump transaction の実行は不可
低速 bcp	実行可能	実行可能
(1 つまたは複数のインデックスやトリガがある場合)	dump transaction の実行は不可	dump transaction は実行可能

- デフォルトでは、新しく作成されたデータベースの select into/bulkcopy オプションはオフです。デフォルトの設定を変更するには、model データベースでこのオプションをオンにします。

注意 インデックスまたはトリガがあるテーブルにデータをコピーする場合には、パフォーマンスが大幅に低下します。膨大な数のローをコピー・インする場合は、まず drop index (または一意性制約として作成されたインデックスには alter table) と drop trigger を使用してすべてのインデックスとトリガを削除し、データベース・オプションの設定、テーブルへのデータのコピー、インデックスとトリガの再生成を行ってからデータベースをダンプすると、処理速度が上がることがあります。ただし、インデックスとトリガの構成用として、データに必要な格納領域の約 2.2 倍のディスク領域を割り付ける必要があります。

---

bcp プロンプトに対する応答

-n (ネイティブ・フォーマット) または -c (文字フォーマット) オプションを使用してデータをコピー・インまたはコピー・アウトする場合、bcp はパスワード入力のプロンプトだけを表示します。ただし、-P オプションによってパスワードが指定されている場合、プロンプトは表示されません。-n、-c、または -f *formatfile* オプションのどれも指定していない場合、bcp は、テーブル内の各フィールドについて情報を入力するようプロンプトを表示します。

- 各プロンプトでは、デフォルト値は角カッコで表示されます。[Return] キーを押すと、この値を選択できます。プロンプトには、次のものがあります。
  - ファイル記憶タイプ。character データ型または Adaptive Server で有効な任意のデータ型。
  - プレフィクス長 (後続のデータの長さをバイトで示す整数)。
  - ファイル内の非 NULL フィールドのデータの記憶長。
  - フィールド・ターミネータ (任意の文字列)。
  - numeric データ型と decimal データ型の位取りと精度。

  ロー・ターミネータは、テーブルまたはファイルの最後のフィールドのフィールド・ターミネータです。

- 角カッコ内のデフォルト値は、該当するフィールドのデータ型として適切な値を表しています。ファイルへコピー・アウトする場合の空き領域の最適な使用方法は、次のとおりです。
  - デフォルトのプロンプトを使用する。
  - すべてのデータをそのテーブルのデータ型でコピーする。
  - 表示どおりのプレフィクスを使用する。
  - ターミネータを使用しない。
  - デフォルトの長さを使用する。

  表 A-2 に、デフォルトおよび代替可能な応答を示します。

表 A-2: bcp プロンプトのデフォルトと応答

プロンプト	デフォルト設定	可能な応答
ファイル記憶タイプ	次のフィールド以外のほとんどのフィールドに対してデータベースの記憶タイプを使用する。 *varchar* には *char* *varbinary* には *binary*	人間が判読できるフォーマットのファイルの作成や読み込みには char データ型。暗黙の変換がサポートされる場合は任意の Adaptive Server データ型。
プレフィクス長	• 0 － *char* データ型とすべての固定長データ型で定義されるフィールド (記憶タイプではない) の場合 • 1 － その他のほとんどのデータ型の場合 • 2 － char として保存される *binary* と *varbinary* の場合 • 4 － text と image の場合	0 － プレフィクスが不要な場合。ほかのすべての場合にはデフォルトの使用を推奨。
記憶長	*char* と *varchar* の場合は、定義されている長さを使用する。 *char* として保存される *binary* と *varbinary* の場合は、デフォルトを使用する。 その他のすべてのデータ型の場合は、トランケーションやデータのオーバフローを避けるために、必要な最大長を使用する。	デフォルト値以上の値を推奨。
フィールド・ターミネータまたはロー・ターミネータ	なし。	最大 30 文字、または次のいずれか。 ¥t タブ ¥n 改行 ¥r 復帰改行 ¥0 null ターミネータ ¥ 円記号

- bcp は、そのネイティブ (データベース) データ型、または暗黙の変換が当該データ型に対してサポートされる任意のデータ型のどちらかとして、データをファイルにコピー・アウトできます。bcp は、ユーザ定義のデータ型をそのベース・データ型または暗黙の変換がサポートされる任意のデータ型としてコピーします。データ型の変換の詳細については、『Open Client DB-Library/C リファレンス・マニュアル』の dbconvert を参照してください。

  注意 すべてのリリースに同じデータ型があるとはかぎらないので、異なるリリースの Adaptive Server からデータをコピーする場合は注意が必要です。

- プレフィクス長は、各データ値の長さをバイト単位で表現する 1 バイト、2 バイト、または 4 バイトの整数で、ホスト・ファイル内のデータ値の直前に位置します。
- データベース内で *char*、*nchar*、*binary* として定義されるフィールドは、データベース内で定義された全長に達するまで、常にスペース (binary の場合は null バイト) が埋め込まれます。*timestamp* データは *binary(8)* として扱われます。

  varchar と *varbinary* フィールド内のデータがコピー・アウトに指定する長さよりも長いと、bcp は、ファイル内のデータを指定の長さに暗黙のうちにトランケートします。
- フィールド・ターミネータ文字列は、30 文字まで指定できます。特に一般的なターミネータは、タブ (“¥t” と入力し、最後のカラム以外のすべてのカラムに使用される) と改行 (“¥n” と入力し、ローの最終フィールドに使用される) です。その他、“¥0” (null ターミネータ)、“¥” (円記号)、“¥r” (復帰改行) があります。ターミネータを選択するときは、それと同じ文字がほかの文字データ部分に存在しないものにします。たとえば、タブを含んでいる文字列に対してタブをターミネータとして使用すると、bcp は、どのタブが文字列の最後を表すのかを区別できません。bcp は常に最初に見つけたタブをターミネータと見なすので、この場合は間違ったものを見つけることになります。

  ターミネータまたはプレフィクスが存在する場合は、実際に転送されるデータの長さに影響します。ファイルにコピー・アウトされるエントリの長さが記憶長に足りないときは、その直後にターミネータが続くか、次のフィールドのためのプレフィクスが続きます。この場合、エントリに記憶長分の埋め込みは行われません (*char*、*nchar*、*binary* データは、Adaptive Server から返されるときすでに、いっぱいの長さまで埋め込みが行われています)。

  ファイルからコピー・インする場合、データは「長さ」プロンプトで指定されたバイト数がコピーされるか、ターミネータが検出されるまで転送されます。指定された長さのバイト数の転送が終了すると、残りのデータはターミネータが検出されるまでフラッシュされます。ターミネータがない場合、テーブルの記憶長が使用されます。

- 次の表は、ファイル内のデータに対するプレフィクス長、ターミネータ、フィールドの長さの関係を示します。“P” は格納テーブル内のプレフィックスを、“T” はターミネータを、ダッシュ (“--”) は付加されるスペースを示します。“...” は各フィールドに対してパターンを繰り返すことを示します。各カラムのフィールド長は 8 で、“string” はそれぞれ 6 文字のフィールドを表します。

表 A-3: Adaptive Server の char データ

	プレフィクス長 = 0	プレフィクス長 1、2、または 4
ターミネータなし	string--string--	Pstring--Pstring--
ターミネータ	string--Tstring--T	Pstring--TPstring--T

表 A-4: char 記憶タイプに変換されるその他のデータ型

	プレフィクス長 = 0	プレフィクス長 1、2、または 4
ターミネータなし	string--string--	PstringPstring
ターミネータ	stringTstringT	PstringTPstringT

- ファイル内のカラムの記憶タイプと長さは、データベース・テーブル内のカラムの記憶タイプと長さと同じである必要はありません (コピー・インされるタイプとフォーマットがデータベース・テーブルの構造と互換性がないと、コピーは失敗します)。
- ファイル記憶長は、通常はターミネータやプレフィクスを除く、カラムに転送されるデータの最大サイズを示します。
- テーブルにデータをコピーする場合は、bcp はカラムに対して定義されているデフォルトとユーザ定義のデータ型を調べます。しかし、bcp は可能なかぎり迅速にデータをロードするためにルールを無視します。
- bcp は null 値を含むことができるすべてのデータ・カラムを可変長であると見なすので、データの各ローの長さを示すプレフィクス長かターミネータを使用してください。
- ネイティブ・フォーマットでホスト・ファイルに書き込まれたデータは、その精度をすべて保持します。*datetime* 値および *float* 値は、文字フォーマットに変換される際も精度を保持します。Adaptive Server は、*money* 値を通貨単位の 1 万分の 1 の精度まで保持します。しかし、*money* 値が文字フォーマットに変換される場合は、文字フォーマット値は、近似値 2 桁しか記録されません。
- 文字フォーマットのデータをファイルからデータベース・テーブルにコピーする前に、『ASE リファレンス・マニュアル』のデータ型に関する項で説明されているデータ型の入力規則をチェックしてください。bcp を使用してデータベースにコピーされる文字データは、これらの規則に従わなければなりません。特に、*(yy)yymmdd* のような区切り文字のない日付形式の場合は、年が最初に指定されていないと、オーバフロー・エラーになる可能性があります。

- 使用している端末と異なる端末を使用するサイトにホスト・データ・ファイルを送るには、送り先に、そのファイルを作成するために使用した *datafile_charset* を通知する必要があります。

メッセージ

変換テーブルのロード中にエラーが発生しました。

-q パラメータで指定した文字変換ファイルが存在しないか、入力した名前が正しくありません。

defncopy

説明

指定されたビュー、ルール、デフォルト、トリガ、プロシージャの定義を、データベースからオペレーティング・システム・ファイルに、またはオペレーティング・システム・ファイルからデータベースにコピーします。

注意　defncopy ユーティリティは、Report Workbench™ を使用して作成されたレポートやテーブル定義をコピーすることはできません。

構文

defncopy {in *filename dbname* | out *filename dbname*
[*owner*.]*objectname* [[*owner*.]*objectname*]...
[-v] [-X]
[-a display_charset]
[-I *interfaces_file*]
[-J [*client_charset*]]
[-K *keytab_file*]
[-P *password*]
[-R *remote_server_principal*]
[-S [*server*]]
[-U *username*]
[-V [*security_options*]]
[-z *language*]
[-Z *security_mechanism*]
{in *filename dbname* | out *filename dbname*
[*owner*.]*objectname* [[*owner*.]*objectname*...] }

パラメータ

in | out

定義をコピーする方向を指定します。

filename

定義コピーの送信元または送信先であるオペレーティング・システム・ファイルの名前を指定します。コピー・アウトを行うと、既存のファイルはすべて上書きされます。

dbname

定義コピーの送信元または送信先であるデータベースの名前を指定します。

objectname

defncopy がコピー・アウトするデータベース・オブジェクトの名前を指定します。定義をコピー・インするときは、objectname を使用しないでください。

-a *display_charset*

defncopy が実行されるマシンの文字セットと異なる文字セットの端末から defncopy を実行できるようにします。-a と -J をともに使用すると、変換に必要な文字セット変換ファイル (*.xlt* ファイル) が指定できます。-a を使用するときに -J を省略できるのは、クライアントの文字セットがデフォルトの文字セットと同じ場合だけです。

-I *interfaces_file*

Adaptive Server に接続するときに検索する *interfaces* ファイルの名前とロケーションを指定できるようにします。-I を指定していない場合、defncopy は、SYBASE 環境変数によって指定されたディレクトリ下の *ini* ディレクトリ内にある *interfaces* ファイル (Windows プラットフォームでは *sql.ini*) を探します。

-J *client_charset*

クライアントで使用する文字セットを指定します。フィルタによって、*client_charset* と Adaptive Server の文字セットとの間で入力が変換されます。

-J *client_charset* は、クライアントで使用する文字セットである *client_charset* とサーバの文字セット間の変換を Adaptive Server に要求します。

-J に引数を指定しないと、文字セットの変換は NULL に設定されます。この場合、変換は行われません。クライアントとサーバが同じ文字セットを使用する場合に、このパラメータを使用してください。

-J の指定を省略すると、文字セットはプラットフォームのデフォルトに設定されます。文字セットおよび関連するフラグの詳細については、『システム管理ガイド』を参照してください。

注意 ascii_7 文字セットは、すべての文字セットと互換性があります。Adaptive Server またはクライアントの文字セットのどちらかが ascii_7 に設定されている場合、すべての 7 ビット ASCII 文字はクライアントとサーバ間でそのまま渡すことができます。その他の文字セットを使用している場合は、変換エラーが発生します。文字セット変換の詳細については、『システム管理ガイド』を参照してください。

-K *keytab_file*

DCE セキュリティにだけ使用できます。-U オプションによって指定されるユーザ名のセキュリティ・キーを含んでいる DCE *keytab* ファイルを指定します。*keytab* ファイルは、DCE の dcecp ユーティリティを使用して作成できます。詳細については、DCE のマニュアルを参照してください。

-K オプションを指定していない場合、defncopy ユーザは -U オプションで指定したユーザ名と同じユーザ名を使用して DCE にログインしなければなりません。

-P *password*

パスワードを指定できるようにします。-V を指定すると、このオプションは無視されます。

-R *remote_server_principal*

リモート・サーバのプリンシパル名を指定します。デフォルトでは、サーバのプリンシパル名はサーバのネットワーク名 (-S オプションまたは DSQUERY 環境変数で指定) と一致します。サーバのプリンシパル名とネットワーク名が同じでない場合は、-R オプションを使用する必要があります。

-S *server*

接続先の Adaptive Server の名前を指定します。-S の指定がない場合、defncopy は DSQUERY 環境変数で指定されたサーバを探します。

-U *username*

ログイン名を指定できるようにします。ログイン名では、大文字と小文字を区別します。*username* を指定しない場合、defncopy は現在のユーザのオペレーティング・システム・ログイン名を使用します。

-V *security_options*

ネットワーク・ベースのユーザ認証を指定します。このオプションを使用する場合、ユーザはユーティリティを実行する前にネットワークのセキュリティ・システムにログインする必要があります。この場合、ユーザは -U オプションにネットワーク・ユーザ名を指定する必要があり、-P オプションに指定されたパスワードは無視されます。

-V の後にキー文字オプションの *security_options* 文字列を続けると、他のセキュリティ・サービスを有効にできます。指定できるキー文字は、以下のとおりです。

c − データ機密性サービスを有効にする

i − データ整合性サービスを有効にする

m − 接続の確立に相互認証を有効にする

o − データ・オリジン・スタンピング・サービスを有効にする

q − 順序不整合の検出を有効にする

r − データ・リプレイの検出を有効にする

-v

defncopy のバージョン番号と版権メッセージを表示して、オペレーティング・システムに戻ります。

-X

サーバへの現在の接続で、アプリケーションがクライアント側のパスワード暗号化を使用してログインを開始するように指定します。defncopy (クライアント) は、サーバに対してパスワードの暗号化が要求されていることを通知します。サーバは暗号化キーを送り返し、defncopy はそれを使用してパスワードを暗号化します。サーバはパスワードを受け取ると、そのキーを使用してパスワードの認証を行います。

-z *language*

サーバが defncopy のプロンプトとメッセージの表示に使用する代替言語の公式名です。-z フラグが指定されていない場合、defncopy はサーバのデフォルト言語を使用します。インストール時、またはインストールしたあとで Adaptive Server に言語を追加するには、langinstall ユーティリティまたは sp_addlanguage ストアド・プロシージャを使用します。

-Z *security_mechanism*

接続で使用するセキュリティ・メカニズムの名前を指定します。

セキュリティ・メカニズム名は Sybase インストール・ディレクトリの中の *ini* サブディレクトリにある *libtcl.cfg* 設定ファイルで定義されます。*security_mechanism* で名前を指定しない場合は、デフォルトのメカニズムが使用されます。セキュリティ・メカニズム名の詳細については、『Open Client/Server 設定ガイド Windows 版』の *libtcl.cfg* ファイルの説明を参照してください。

例

例 1 定義を *new_proc* ファイルから “MERCURY” サーバ上のデータベース stagedb へコピーします。“MERCURY” への接続は、ユーザ名 “sa”、パスワード NULL で確立されます。

```
defncopy -Usa -P -SMERCURY in new_proc stagedb
```

例 2 sp_calccomp オブジェクトと sp_vacation オブジェクトの定義を Sybase サーバ上の employees データベースから *dc.out* ファイルへコピーします。メッセージとプロンプトは「フランス語」で表示されます。ユーザには、パスワード入力のプロンプトが表示されます。

```
defncopy -S -z french out dc.out employees sp_calccomp sp_vacation
```

使用法

- defncopy プログラムは、オペレーティング・システムから直接起動します。defncopy を使用すると、ビュー、ルール、デフォルト、トリガ、プロシージャ用の定義 (create 文) をデータベースからオペレーティング・システム・ファイルにコピー・アウトするときに、非対話型操作でコピーを実行できます。または、指定されたファイルからすべての定義をコピーできます。

  定義をコピー・アウトするには、sysobjects テーブルおよび syscomments テーブルに対する select パーミッションが必要です。オブジェクト自体のパーミッションは必要ありません。

- コピー・インするオブジェクトのタイプについては、適切な create パーミッションを持つ必要があります。コピー・インされたオブジェクトはコピーするユーザが所有します。ユーザの代わりに定義をコピーするシステム管理者は、再構築されるデータベース・オブジェクトに対してそのユーザが正しくアクセスできるように、そのユーザとしてログインしなければなりません。

- in *filename* または out *filename*、およびデータベース名は必須であり、明確に指定される必要があります。コピー・アウトする場合は、オブジェクト名とその所有者の両方を表すファイル名を使用してください。

- defncopy は、コピー・アウトする各定義を次のようなコメントで終了します。

  ```
  /* ### DEFNCOPY: END OF DEFINITION  */
  ```

  defncopy を使用してデータベースにコピーするオペレーティング・システム・ファイル内の定義をアセンブルする場合、各定義はこの “END OF DEFINITION” という文字列を使用して終了しなければなりません。

- defncopy に対して指定する値がシェルにとって意味を持つ文字を含んでいる場合は、それらを引用符で囲んでください。

警告！ 100 文字を超える長いコメントが create 文の前にあると、defncopy が失敗することがあります。

System 11 の defncopy は、Client-Library を使用して構築されています。defncopy のユーザ・インタフェースは、次の点以外は変更されていません。

- 接続でのネットワーク・ベースのセキュリティ・サービスを使用可能にする、次のような新しいコマンド・ライン・オプションが追加されました。

  -K *keytab_file*
  -R *remote_server_principal*
  -V *security_options*
  -Z *security_mechanism*

- -y *sybase_directory* オプションは削除されました。

isql

説明

Adaptive Server に対する対話型の SQL パーサです。

構文

isql [-b] [-e] [-F] [-n] [-p] [-v] [-X] [-Y]
[-a *display_charset*]
[-A *size*]
[-c *cmdend*]
[-D *database*]
[-h *headers*]
[-H *hostname*]
[-i *inputfilename*] [-I *interfaces_file*]
[-J *client_charset*]
[-K *keytab_file*]
[-l *login_timeout*]
[-m *errorlevel*]
[-o *outputfilename*]
[-P *password*]
[-Q *option*]|
[-R *remote_server_principal*]
[-s *colseparator*]
[-S *server*]
[-t *timeout*]
[-U *username*]
[-V [*security_options*]]
[-w *columnwidth*]
[-z *language*]
[-Z *security_mechanism*]

パラメータ

-a *display_charset*

isql が実行されるマシンの文字セットと異なる文字セットを使用する端末から isql を実行できるようにします。-a と -J をともに使用すると、変換に必要な文字セット変換ファイル (*.xlt* ファイル) が指定できます。-a を使用するときに -J を省略できるのは、クライアントの文字セットがデフォルトの文字セットと同じ場合だけです。

-A *size*

この isql セッションに使用するネットワーク・パケットのサイズを指定します。次に例を示します。

```
isql -A 2048
```

この例は、この isql セッションのパケット・サイズを 2048 バイトに設定します。確認するには、次のように入力します。

```
select * from sysprocesses
```

値は *network_pktsz* という見出しの下に表示されます。

size は、default network packet size 設定変数と maximum network packet size 設定変数の間の値で、512 の倍数を指定してください。

readtext や writetext オペレーションのように I/O 集約型のオペレーションを実行する場合は、デフォルトよりも大きいパケット・サイズを使用してください。

Adaptive Server のパケット・サイズの設定や変更は、リモート・プロシージャ・コールのパケット・サイズには影響しません。

-b － テーブル・ヘッダを出力しないようにします。

-c *cmdend*

コマンド・ターミネータを再設定します。デフォルトでは、各コマンドを終了させて、“go” を 1 行に単独で入力することによって、Adaptive Server に各コマンドを送信します。コマンド・ターミネータを再設定する場合は、SQL の予約語や制御文字は使用してはいけません。? ( ) [ ] $ などのシェル・メタ文字は、必ずエスケープしてください。

-D *database*

isql のセッションを開始するデータベースを選択します。

-e

入力内容をエコーします。

-E *editor*

デフォルト・エディタ以外のエディタを指定します。

-F

FIPS フラガを使用可能にします。このオプションを使用すると、Adaptive Server は送信される標準外の SQL コマンドすべてにフラグを付けます。

-h *headers*

カラム見出しから次のカラム見出しまでの間に出力されるローの数を指定します。デフォルトでは、クエリ結果のセットごとに 1 回だけ見出しが出力されます。

-H *hostname*

クライアント・ホスト名を設定します。

-i *inputfilename*

isql への入力に使用するオペレーティング・システム・ファイルの名前を指定します。このファイルには、コマンド・ターミネータを含むようにしてください (デフォルトでは “go”)。

-I *interfaces_file*

Adaptive Server に接続するときに検索する interfaces ファイルの名前とロケーションを指定します。-I を指定していない場合、isql は、SYBASE 環境変数によって指定されたディレクトリ下の *ini* ディレクトリ内にある *interfaces* ファイル (Windows プラットフォームでは *sql.ini*) を探します。

-J *client_charset*

クライアントで使用する文字セットを指定します。-J *client_charset* は、クライアントで使用する文字セットである *client_charset* とサーバの文字セット間の変換を Adaptive Server に要求します。フィルタによって、*client_charset* と Adaptive Server 文字セット間で入力を変換します。

-J に引数を指定しないと、文字セットの変換は NULL に設定されます。この場合、変換は行われません。クライアントとサーバが同じ文字セットを使用する場合に、このパラメータを使用してください。

-J の指定を省略すると、文字セットはプラットフォームのデフォルトに設定されます。デフォルトの文字セットは、クライアントが使用する文字セットと同じであるとはかぎりません。

-K *keytab_file*

DCE セキュリティにだけ使用できます。-U オプションによって指定されるユーザ名のセキュリティ・キーを含んでいる DCE *keytab* ファイルを指定します。keytab ファイルは、DCE の dcecp ユーティリティを使用して作成できます。詳細については、DCE のマニュアルを参照してください。

-K オプションを指定しない場合、isql のユーザは -U オプションで指定するユーザ名と同じユーザ名を使用して DCE にログインしなければなりません。

-l *login_timeout*

Adaptive Server に接続する場合の最大タイムアウト値を指定します。

-m *errorlevel*

エラー・メッセージの表示をカスタマイズします。指定の重大度レベル以上のエラーの場合には、メッセージ番号、ステータス、エラー・レベルを表示し、エラー・テキストは表示しません。指定した重大度より低いレベルのエラーでは、何も表示されません。

-n

入力行から行番号とプロンプト記号 (>) を削除します。

-o *outputfilename*

isql からの出力を保管するオペレーティング・システム・ファイルの名前を指定します。

-p

パフォーマンスの統計値を出力します。

-P *password*

現在の Adaptive Server パスワードを指定します。-V を指定すると、このオプションは無視されます。パスワードは大文字と小文字が区別され、6 ～ 30 文字の範囲で指定できます。

-Q *option*

HA フェールオーバを有効にします。

-R *remote_server_principal*

リモート・サーバのプリンシパル名を指定します。デフォルトでは、サーバのプリンシパル名はサーバのネットワーク名 (-S オプションまたは DSQUERY 環境変数で指定) と一致します。サーバのプリンシパル名とネットワーク名が同じでない場合は、-R オプションを使用する必要があります。

-s *colseparator*

カラム・セパレータ文字をリセットします。デフォルト・カラム・セパレータ文字はブランクです。オペレーティング・システムに対して特別な意味を持つ文字 (| ; & <> など) を使用するには、それらを引用符で囲むか、前に円記号を付けます。

-S *server*

接続先の Adaptive Server の名前を指定します。-S を指定しない場合、isql は DSQUERY 環境変数で指定されたサーバを探します。

-t *timeout*

コマンドがタイムアウトになるまでの秒数を指定します。タイムアウト値の指定がないと、コマンドは永久に実行を続けます。これは、isql 内から発行されたコマンドに影響するもので、接続時間には影響しません。isql にログインするためのデフォルトのタイムアウトは、60 秒です。

-U *username*

ログイン名を指定します。ログイン名は、大文字と小文字を区別します。

-V *security_options*

ネットワーク・ベースのユーザ認証を指定します。このオプションを使用する場合、ユーザはユーティリティを実行する前にネットワークのセキュリティ・システムにログインする必要があります。この場合、ユーザは -U オプションにネットワーク・ユーザ名を指定する必要があり、-P オプションに指定されたパスワードは無視されます。

-V の後にキー文字オプションの *security_options* 文字列を続けると、他のセキュリティ・サービスを有効にできます。指定できるキー文字は、以下のとおりです。

c − データ機密性サービスを有効にする

i − データ整合性サービスを有効にする

m − 接続の確立に相互認証を有効にする

o − データ・オリジン・スタンピング・サービスを有効にする

q − 順序不整合の検出を有効にする

r − データ・リプレイの検出を有効にする

-v

使用している isql ソフトウェアのバージョンと版権メッセージを表示します。

-w *columnwidth*

出力画面の幅を設定します。デフォルトでは、80 文字です。出力行が画面幅いっぱいになった場合は、複数の行に分割されます。

-X
サーバへのログイン接続をクライアント側のパスワード暗号化を使用して開始します。isql (クライアント) は、サーバに対してパスワードの暗号化が必要であることを通知します。サーバは、isql がパスワードを暗号化するために使う暗号化キーを返送し、パスワードを受け取ると、そのキーを使用してそのパスワードを確認します。

-Y
連鎖トランザクションを使用するように Adaptive Server に指示します。

-z *language*
isql のプロンプトとメッセージの表示に使用する代替言語の公式名です。-z フラグを指定しないと、isql はサーバのデフォルト言語を使用します。インストール時、またはインストールしたあとで Adaptive Server に言語を追加するには、langinstall ユーティリティまたは sp_addlanguage ストアド・プロシージャを使用します。

-Z *security_mechanism*
接続で使用するセキュリティ・メカニズムの名前を指定します。

セキュリティ・メカニズム名は Sybase インストール・ディレクトリの中の *ini* サブディレクトリにある *libtcl.cfg* 設定ファイルで定義されます。*security_mechanism* で名前を指定しない場合は、デフォルトのメカニズムが使用されます。セキュリティ・メカニズム名の詳細については、『Open Client/Server 設定ガイド Windows 版』の *libtcl.cfg* ファイルの説明を参照してください。

例

例 1 コマンドを実行します。

```
isql
 Password:

 1>select *
 2>from authors
 3>where city = "Oakland"
 4>go
```

例 2 クエリを編集できるようになります。ファイルに書き込みを行って保存すると、isql に戻ります。クエリが表示されます。クエリを実行するには go を入力します。

```
isql -Ujoe -Pabracadabra
1>select *
2>from authors
3>where city = "Oakland"
4>ed
```

例 3 reset によってクエリ・バッファがクリアされます。quit を入力すると、オペレーティング・システムに戻ります。

```
isql -U alma Password:
1>select *
2>from authors
3>where city = "Oakland"
4>reset
5>quit
```

使用法

- isql を対話型で使用するには、オペレーティング・システムのプロンプト画面で isql コマンド (および任意のオプション・フラグ) を入力します。isql プログラムは、SQL コマンドを受け取り、それを Adaptive Server に送信します。結果は、フォーマットされ、標準出力に出力されます。isql を終了するには、quit または exit を使用します。
- デフォルトのコマンド・ターミネータ go で始まる行を入力するか、または -c オプションを使用する場合は、ほかのコマンド・ターミネータで始まる行を入力してコマンドを終了します。コマンド・ターミネータのあとに、コマンドを実行する回数を指定する整数を指定できます。たとえば、あるコマンドを 100 回実行する場合は、次のように入力します。

  ```
  select x = 1
   go 100
  ```

  結果は、実行の終了時に 1 回表示されます。
- コマンド・ラインにオプションを複数回入力した場合、isql は最後の値を使用します。たとえば、次のコマンドを入力したとします。

  ```
  isql -c. -csend
  ```

  -c の 2 番目の値 “send” は、最初の値 “.” を上書きします。これによって、設定したすべてのエイリアスを無効にすることができます。
- 既存のクエリ・バッファをクリアするには、行に reset とだけ入力します。isql は、未処理の入力内容をすべて破棄します。入力行の任意の場所で [Ctrl + C] を押すことによって、現在のクエリをキャンセルして isql のプロンプト画面に戻ることもできます。
- isql のフラグを使用する場合には、大文字と小文字を区別してください。
- isql は float または real データを丸めて、小数点以下 6 桁までを表示します。
- isql を対話型で使用するときは、次のコマンドを使用して、オペレーティング・システム・ファイルをコマンド・バッファに読み込みます。

  ```
  :r filename
  ```

  ファイル内にコマンド・ターミネータを含めないで、編集を終わったあとにターミネータを対話的に入力してください。

- isql によって Adaptive Server に渡される Transact-SQL 文には、コメントを入れることができます。コメントは、次の例に示すように “/*” と “*/” で囲みます。

```
select au_lname, au_fname
 /*retrieve authors' last and first names*/
 from authors, titles, titleauthor
 where authors.au_id = titleauthor.au_id
 and titles.title_id = titleauthor.title_id
 /*this is a three-way join that links authors
**to the books they have written.*/
```

  go コマンドをコメントにする場合は、コマンドが行の先頭にならないようにします。たとえば、go コマンドをコメントにする場合は次のようにしてください。

```
/*
**go
*/
```

  下記のようにはしないでください。

```
/*
go
*/
```

System 11 の isql は、Client-Library を使用して構築されています。isql のユーザ・インタフェースは、次の点以外は変更されていません。

- 5701 サーバ・メッセージ (「データベースが変更されました」) は、ログイン後または use database コマンドの発行後には表示されません。
- 次の 2 つのオプション・フラグが新しく追加されました。

  -b − カラム・ヘッダを出力しないようにします。
  -D *database* − isql が使用する起動時のデータベースを選択します。
- 接続でのネットワーク・ベースのセキュリティ・サービスを使用可能にする、次のようなコマンド・ライン・オプションが追加されました。

  -K *keytab_file*
  -R *remote_server_principal*
  -V *security_options*
  -Z *security_mechanism*
- エラー・メッセージのフォーマットが、以前のバージョンの isql とは異なります。これらのメッセージの内容に基づいたルーチンを実行するスクリプトは、修正が必要な場合があります。
- -y *sybase_directory* オプションは削除されました。

isql 内で使用できる追加コマンド

- isql を終了する場合：quit または exit
- クエリ・バッファを消去する場合：reset
- オペレーティング・システム・コマンドを実行する場合：!! *command*

参照　『ASE リファレンス・マニュアル』の sp_addlanguage、sp_addlogin、sp_configure、sp_defaultlanguage、sp_droplanguage、sp_helplanguage

付 録 B

プリコンパイラ・リファレンス

Embedded SQL プリコンパイラ cpre と cobpre は同じ構文とフラグ情報を使用します。この項では、両方に適用されるリファレンスをまとめて説明します。

cobpre と cpre

説明

cpre は C ソース・プログラムをプリコンパイルして、ターゲット・ファイル、リスティング・ファイル、isql ファイルを生成します。

cobpre は COBOL ソース・プログラムをプリコンパイルして、ターゲット・ファイル、リスティング・ファイル、isql ファイルを生成します。

構文

```
cpre|cobpre [/a] [/b] [/c] [/d] [/e] [/f] [/l] [/m]      [/p(cpre only)] [/s] [/r] [/v] [/w]
      [/x] [/y]
    [/Ccompiler]
    [/Ddatabase_name]
    [/Ffips_level]
    [/G[isql_file_name]]
    [/Iinclude_directory]...
    [/Jlocale_for_charset]
    [/Ksyntax_level]
    [/L[listing_file_name]]
    [/Mlabelname=labelvalue]
    [/Nsql.ini]
    [/Otarget_file_name]
    [/P[password]]
    [/Sserver_name]
    [/Ttag_id]
    [/Uuser_id]
    [/Vversion_number]
    [/Zlocale_for_messages]
 program[.ext] [program[.ext]]...
```

注意 オプションは、スラッシュ (/) またはハイフン (-) のどちらかを使ってフラグできます。したがって、cpre -l と cpre /l は同じことを表します。

パラメータ

/a

トランザクション間で、カーソルをオープンしたままにできるようにします。カーソルとトランザクションの詳細については、『ASE リファレンス・マニュアル 』を参照してください。このオプションを使用しない場合、カーソルは set close on endtran on が有効であるかのように動作します。これは ANSI 準拠の動作です。

/b

fetch 文で一般的に使用されるホスト変数アドレスの再バインドを無効にします。このオプションを使用しない場合、Embedded SQL/C または Embedded SQL/COBOL プログラム内で特に指定のないかぎり、fetch 文ごとに再バインドが行われます。

/b オプションは、Embedded SQL プリコンパイラの 11.1 バージョンと 10.0.x バージョンで次のような相違点があります。

- 11.1 以降のバージョンの cpre では、norebind 属性が適用されるのはカーソルのすべての fetch 文です。ただし、カーソルの宣言が /b オプションによってプリコンパイルされていた場合です。
- 10.0.x バージョンの cpre では、norebind 属性が適用されるのは、/b オプションによってプリコンパイルされた各 Embedded SQL ソース・ファイル内にあるすべての fetch 文です。この場合、カーソルがこのオプションによって宣言されたかどうかは関係ありません。

/c

ct_debug に対する呼び出しを生成することによって Client-Library のデバッグ機能をオンにします。

このオプションは、アプリケーションの開発のときには役立ちますが、最終的にアプリケーションを配布するときにはオフにしてください。このオプションを正しく機能させるには、Sybase リリース・ディレクトリの */devlib* ディレクトリ内にあるライブラリおよび DLL とアプリケーションをリンクして実行する必要があります。

/d

区切り識別子 (二重引用符で囲まれた識別子) をオフにして、SQL 文中の引用符で囲まれた文字列を文字リテラルとして扱えるようにします。

/e

exec sql connect 文を処理するときに、外部設定ファイルを使用して接続の設定を行うように、Client-Library に指示します。詳細については、/x オプションと『Open Client Client-Library/C リファレンス・マニュアル』の CS_CONFIG_BY_SERVERNAME プロパティを参照してください。

このオプションの指定がなければ、プリコンパイラは Client-Library の関数呼び出しを生成して接続の設定を行います。外部設定ファイルの詳細については、『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

/f

ANSI FIPS 準拠の検査を行うための FIPS フラガをオンにします。

/l

#line ディレクティブを生成しないようにします。このオプションは ESQL/C でのみ使用できます。

/m

アプリケーションを Sybase のオートコミット・モードで実行します。このモードではトランザクションが連鎖しません。明示的な開始トランザクションと終了トランザクションが必要であり、なければそれぞれの文は即座にコミットされます。このオプションを指定しないと、アプリケーションは ANSI 形式の連鎖トランザクション・モードで実行されます。

/p (cpre のみ)

入力ホスト変数を持つモジュール内の SQL 文ごとに別のコマンド・ハンドルを生成し、各コマンド・ハンドルに対して継続バインドを有効にします。このオプションを使用すると、入力パラメータの付いたコマンドを繰り返し実行するときのパフォーマンスが改善されます。ただし、格納領域の使用量が増加して、各コマンドの初回の応答時間が長くなります。

ホスト文字列変数が空のときに NULL 文字列の代わりに空の文字列が挿入されないと動作しないアプリケーションは、/p オプションがオンになっていると正しく機能しません。継続バインドを実装しているので、Embedded SQL は Client-Library プロトコル (NULL 文字列を挿入する) を回避することができません。

/r

繰り返し読み出しを無効にします。このオプションを使用していない場合、connect 文の最中に実行される、set transaction isolation level 3 文が生成されます。デフォルトの独立性レベルは 1 です。

/s

静的関数宣言をインクルードします。

/v

プリコンパイラのバージョン情報だけを表示します。プリコンパイルは実行しません。

/w

警告メッセージの表示をオフにします。

/x

外部設定ファイルを使用します。『Open Client Client-Library/C リファレンス・マニュアル』にある CS_EXTERNAL_CONFIG プロパティの説明、および『Open Client Embedded SQL/C リファレンス・マニュアル』と『Open Client Embedded SQL/COBOL リファレンス・マニュアル』にある INITIALIZE_APPLICATION 文の説明を参照してください。

/y

S_TEXT データ型と CS_IMAGE データ型を入力ホスト変数として使用できるようにします。実行時に、データはサーバに送信される文字列に直接挿入されます。サポートされるのは静的 SQL 文だけです。動的 SQL に対する入力パラメータとして、text および image を使用することはできません。この引数のコマンド文字列への置換は、-y コマンド・ライン・オプションが使用されたときだけ実行されます。

/C *compiler*
対象のホスト言語コンパイラを指定します。

- COBOL の場合、使用可能な値は次のとおりです。

  “mf_byte” − バイト整列データを使用する Micro Focus COBOL (/C NOIBMCOMP)

  “mf_word” − ワード整列データを使用する Micro Focus COBOL (/C IBMCOMP)

- C の場合、使用可能な値は次のとおりです。

  “ANSI_C” − ANSI C コンパイラ

  “MSVC” − Microsoft Visual C コンパイラ。“MSVC” プリコンパイラの出力は、1K バイトより長い文字列を生成しません。

/D*database_name*
解析対象のデータベースの名前を指定します。このオプションは、プリコンパイル時に SQL のセマンティックを検査する場合に使用します。/G を指定すると、use *database* コマンドが *filename.sql* ファイルの先頭に追加されます。このオプションを指定しない場合、プリコンパイラは Adaptive Server のデフォルト・データベースを使用します。

/Ffips_level
指定の準拠レベルを調べます。現在のところ、プリコンパイラは SQL89 または SQL92E を調べます。

/G[isql_file_name] (引数は省略可能)
該当する SQL 文に使用するストアド・プロシージャを生成し、isql によってデータベースへの入力用としてストアド・プロシージャをファイルに保存します。複数の入力ファイルがある場合は、/G を使用できますが、引数を指定することはできません。

複数の入力ファイルがある場合、または引数を指定しない場合、デフォルトのターゲット・ファイル名 (1 つまたは複数) は、拡張子 *isql* を付加した (または入力ファイルの拡張子に置き換えた) 入力ファイル名になります。

また、ストアド・プロシージャにタグ ID を指定するときは、-Ttag_id オプションも参照してください。

/G オプションを使用しない場合、ストアド・プロシージャは生成されません。

/Iinclude_directory
Embedded SQL がインクルード・ファイルを探すディレクトリを指定します。このオプションは何回でも指定できます。Embedded SQL はコマンド・ラインに指定された順に複数のディレクトリを探します。このオプションを指定しないときのデフォルトは、Sybase リリース・ディレクトリの */include* ディレクトリと現在の作業ディレクトリです。

/J locale_for_charset

プリコンパイルするソース・ファイルの文字セットを指定します。このオプションの値は、ロケール・ファイル内のエントリに対応するロケール名でなければなりません。/J を指定しない場合、プリコンパイラはソース・ファイルがプリコンパイラのデフォルト文字セットを使用していると解釈します。

デフォルトとして使用する文字セットを決定するために、プリコンパイラはロケール名を調べます。CS-Library は次の順序で情報を検索します。

- LC_ALL
- LANG

  LC_ALL が定義されていると、CS-Library はこの値をロケール名として使用します。LC_ALL が定義されておらず、LANG が定義されている場合は、CS-Library はその値をロケール名として使用します。これらのロケール値がどれも定義されていない場合、CS-Library は「デフォルト」のロケール名を使用します。

プリコンパイラは *locales* ファイル内からロケール名を探して、そのロケール名に対応する文字セットをデフォルトの文字セットとして使用します。

/K syntax_level

実行する構文検査のレベルを指定します。選択肢は次のとおりです。

- NONE
- SYNTAX
- SEMANTIC

デフォルト値は NONE です。SYNTAX または SEMANTIC のどちらかを使用する場合は、/U、/P、/S、/D オプションも指定し、使用している Adaptive Server に Embedded SQL を接続できるようにする必要があります。

このオプションを使用しない場合は、プリコンパイラはサーバに接続しないか、ターゲット・ファイルを生成するために必要とされる入力ファイルの SQL 構文検査だけを実行します。

/Llisting_file_name (引数は省略可能)

1 つまたは複数のリスティング・ファイルを生成します。リスティング・ファイルは、行番号が付けられた各行のあとに、エラーがある場合は該当するエラー・メッセージが続く入力ファイル・フォーマットのファイルです。複数の入力ファイルがある場合は、/L を使用できますが、引数を指定することはできません。

複数の入力ファイルがある場合、または引数を指定しない場合、デフォルトのリスティング・ファイル名 (1 つまたは複数) は、*.lis* 拡張子を付加した (または、入力ファイルの拡張子を置き換えた) 入力ファイル名になります。

このオプションを指定しない場合は、リスティング・ファイルは生成されません。

/M
セキュリティ機能をオンにして、セキュリティ・ラベルを B1 に設定します。

/N sql.ini
プリコンパイラに設定ファイルの名前を指定します。

/O target_file_name
ターゲット・ファイルとなる出力ファイルの名前を指定します。複数の入力ファイルがあるときは、このオプションは使用できません (デフォルトのターゲット・ファイル名が割り当てられます)。このオプションを使用しなければ、デフォルトのターゲット・ファイル名は、*.cbl* 拡張子を付加した (または入力ファイルの拡張子に置き換えた) 入力ファイル名になります。

/P password](/Uuser_id; オプションとともに使用。引数は省略可能)
プリコンパイル時に SQL 構文を検査するための Adaptive Server パスワードを指定します。引数を付けずに /P を指定するか、引数としてキーワード NULL を使用すると、null (““) パスワードが指定されます。/P を使用しないで /Uuser_id オプションを使用すると、プリコンパイラはパスワード入力のプロンプトを表示します。このオプションは /G フラグとともに使用してください。

/S server_name
プリコンパイル時に SQL 構文を検査する Adaptive Server の名前を指定します。このオプションを使用しない場合は、DSQUERY 環境変数の値がデフォルト Adaptive Server 名として使用されます。DSQUERY が設定されていない場合には、SYBASE がサーバの名前として使用されます。

/T tag_id (/G オプションとともに使用)
生成されるストアド・プロシージャ・グループ名の最後に付加するタグ ID (最大 3 文字) を指定します。

たとえば、コマンドの一部として /Tdbg を入力すると、生成されるストアド・プロシージャは、入力ファイルの名前にタグ ID として *dbg* が付加された名前 (*program_dbg;1*、*program_dbg;2* など) になります。

プログラマはタグ ID を使用することによって、使用中の可能性がある既存の生成されたストアド・プロシージャに影響を与えずに、既存のアプリケーションに対する変更をテストできます。

このオプションを使用していない場合は、ストアド・プロシージャ名にタグ ID は付加されません。

/U user_id

Adaptive Server のユーザ ID を指定します。

このオプションを使用すると、プリコンパイル時に SQL 構文を検査できます。このオプションは、SQL 文を解析のためだけにサーバに渡すようにプリコンパイラに指示します。サーバが構文エラーを検出すると、エラーがレポートされてコードは生成されません。/P[password] オプションを使用しない場合、このオプションを選択すると、パスワード入力のプロンプトが表示されます。

/K、/P、/S、/D オプションも参照してください。

/V version_number

Client-Library のバージョン番号を指定します。COBOL の場合、バージョン番号は cobpub.cbl からの値の 1 つと一致しなければなりません。このオプションを指定しないときのデフォルトは、プリコンパイラで使用できる Client-Library の最新のバージョン (Open Client/Open Server バージョン 12.5 では CS_VERSION_125) になります。

/Z locale_for_messages

プリコンパイラがメッセージ用に使用する言語と文字セットを指定します。/Z を指定していない場合、プリコンパイラはそのデフォルト言語と文字セットをメッセージ用に使用します。

プリコンパイラは、メッセージ用のデフォルトとして使用する言語と文字セットを次の順序に従って決定します。

1　ロケール名を探します。CS-Library は次の順序で情報を検索します。

- LC_ALL
- LANG

LC_ALL が定義されている場合、CS-Library はこの値をロケール名として使用します。LC_ALL が定義されておらず、LANG が定義されている場合は、CS-Library はその値をロケール名として使用します。これらのロケール値がどれも定義されていない場合、CS-Library は「デフォルト」のロケール名を使用します。

2　*locales* ファイル内でロケール名を探して、そのロケール名に対応する言語と文字セットを調べます。

3　手順 2 で調べた言語と文字セットに対応する、ローカライズされたメッセージと文字セットの情報をロードします。

program[.ext] [program[.ext]] . . .

ESQL/COBOL ソース・プログラムの入力ファイル名 (1 つまたは複数) を指定します。入力ファイル名は必要な数だけいくつでも入力できます。ファイル名の形式と長さは、規則に違反しないかぎり、どのようなものでもかまいません。ターゲット・ファイルと複数の入力ファイルについては、「コメント」の項を参照してください。

@file_name

上記のコマンド・ライン引数のいずれかを含んでいるファイルを指定するために使用します。プリコンパイラは、すでに指定されている引数に加えてこのファイルに含まれている引数を読み込みます。@file_name で指定するファイル内にプリコンパイルするファイルの名前が含まれている場合は、この引数はコマンド・ラインの最後に置いてください。

例

1 プリコンパイラを実行します (ANSI 準拠)。

```
cobpre|cpre program.pco
```

2 生成されたストアド・プロシージャと FIPS フラグを使用して、プリコンパイラを実行します (ANSI 準拠)。

```
cobpre|cpre /G /f program1.pco program2.pco
```

3 トランザクション間で開かれたままのカーソルを持つ 2 つの入力ファイルに対して、プリコンパイラを実行します (ANSI 非準拠)。

```
cobpre|cpre /a program1.pco program2.pco
```

4 プリコンパイラのバージョン情報だけを表示します。

```
cobpre|cpre /v
```

5 SQL チェックの最高のレベルでプリコンパイラを実行します。

```
cobpre|cpre /K SEMANTIC /Uuser_id /Ppassword /Sserver_name ¥
     /Dpubs2 example1.[pc|pco]
```

使用法

- cobpre|cpre コマンドのデフォルトは、ANSI 規格の動作にセットアップされます。
- /a、/c、/f、/m、/r、/V オプションは connect 文だけに影響します。ソース・ファイルが connect 文を含んでいない場合、または /e か /x を使用する場合は、これらのオプションは効果がありません。
- ターゲット・ファイル
  デフォルトのターゲット・ファイル名は、*.cbl* 拡張子 (Micro Focus COBOL の場合) を付加した (または、入力ファイル名の拡張子を置き換えた) 入力ファイル名になります。入力ファイルが 1 つだけの場合は、ターゲット・ファイル名の指定に /O target_file_name オプションを使用できます。複数の入力ファイルがある場合は、デフォルトのターゲット・ファイルは、*first_input_file.cbl*、*second_input_file.cbl* などになります。
- オプションのフォーマット
  オプションは、引数の前にスペースがあってもなくてもかまいません。たとえば、次のどちらのフォーマットでも使用できます。

  /Tdbg
  または
  /T dbg

- プリコンパイラは複数の入力ファイルを処理できます。ただし、/O *target_file_name* オプションを使用することはできません。デフォルトのターゲット・ファイル名を使用する必要があります (上記の「ターゲット・ファイル」の説明を参照)。/G[isql_file_name] オプションを使用するときは、引数を指定することはできず、デフォルトの isql ファイル名は、*first_input_file.sql*、*second_input_file.sql* などになります。
  /L[listing_file_name] オプションを使用するときも、引数を指定することはできません。デフォルトのリスティング・ファイル名は、*first_input_file.lis*、*second_input_file.lis* などになります。

アプリケーションの開発

この項では、Embedded SQL アプリケーションを開発するための、一般的に使用される手順について説明します。この手順は、稼働条件に合うように適応させることが必要な場合もあります。これらの手順は、DOS コマンド・プロンプト画面で実行してください。

1　構文検査とデバッグのために、/c、/Ddatabase_name、/P[password]、/Sserver_name、/K[SYNTAX| SEMANTIC]、/Uuser_id オプションを使用してプリコンパイラを実行します。/G[isql_file_name] は使用しないでください。プログラムのコンパイルとリンクを実行して、構文が正しいかどうか確認します。

2　必要な修正をすべて行います。オプション /Ddatabase_name、/G[isql_file_name]、/Ttag_id を使用してプリコンパイラを実行し、テスト・プログラム用のタグ ID を持つストアド・プロシージャを生成します。テスト・プログラムをコンパイルしてリンクします。次のコマンドを使用して、ストアド・プロシージャをロードします。

```
isql /P[password] /Sserver_name /Uuser_id ¥
      /iisql_file_name
```

プログラム上でテストを実行します。

3　修正したプログラム上で /Ddatabase_name オプションと /G[isql_file_name] オプションを使用してプリコンパイラを実行します (/T は使用しません)。プログラムをコンパイルしてリンクします。次のコマンドを使用して、ストアド・プロシージャをロードします。

```
isql /P[password] /Sserver_name /Uuser_id ¥
/iisql_file_name
```

これで、最終的な配布用プログラムを実行する準備が完了しました。

プリコンパイラがサーバの名前を調べる方法

プリコンパイル時に Adaptive Server に接続することによって、プリコンパイル時に追加で構文チェックを実行できます。プリコンパイラは、次の 3 つの方法のいずれかを使用してサーバの名前を調べます。

- cpre または cobpre コマンド・ラインで -S オプションを使用する
- DSQUERY 変数を設定する
- デフォルト値 “SYBASE” を使用する

-S オプションは、DSQUERY によって設定される値を上書きします。

プリコンパイラのコマンド・ライン上でサーバを選択する場合の構文は、次のとおりです。

```
cobpre|cpre -Usa -P -Sserver_name
```

または、接続呼び出しか接続文からのサーバ名をそのまま残すと、*server_name* はその値を DSQUERY 環境変数のランタイム値から取得します。アプリケーション・ユーザが DSQUERY を設定していない場合、サーバ名のランタイム値はデフォルトの “SYBASE” になります。DSQUERY の詳細については、『Open Client/Server 設定ガイド Windows 版』を参照してください。

cobpre | cpre のデフォルト

次の表は、cobpre | cpre ユーティリティのオプションとデフォルトを示します。

表 B-1: cobpre | cpre のデフォルト

オプション	オプションを使用しない場合のデフォルト
/C*compiler*	COBOL の場合は mf_byte コンパイラ、C の場合は ANSI-C。
/D*database_name*	Adaptive Server のデフォルト・データベース。
/F*fips_level*	(FIPS フラグは使用不可)
/G[*isql_file_name*]	ストアド・プロシージャは生成されない。
/I*include_directory*	デフォルト・ディレクトリは Sybase リリース・ディレクトリの */include* ディレクトリ。
locale_for_charset	[プラットフォームによって異なる]
/K[syntax | semantic | none]	syntax も semantic も選択しない場合、デフォルト設定は “None”。
/L[*listing_file_name*]	リスティング・ファイルは生成されない。
/M	(現在は使用不可)
/Nsql.ini	Sybase リリース・ディレクトリの */ini* 内にある *sql.ini* ファイル。
/Otarget_file_name	デフォルトのターゲット・ファイル名は、拡張子 *.cbl* または *.c* が付加された (または、入力ファイル名の拡張子に置き換えられた) 入力ファイル名。
/P[password]	/U*user_id* を使用しないかぎり、パスワード入力のプロンプトは表示されない。
/Sserver_name	デフォルトの Adaptive Server 名は DSQUERY 環境変数から取得される。
/Ttag_id	/G を使用して生成されるストアド・プロシージャにはタグ ID は付加されない。
/Uuser_id	なし。
/Vversion_number	バージョン 11.1.x 以降は CS_VERSION_110、バージョン 12.5.x 以降は CS_VERSION_125。
/Z*locale_for_messages*	[プラットフォームまたは環境によって異なる]

索引

E

F

G

I

L

M

O

P

R

S

T

U

W

い

お

か

こ

さ

ま

も

ゆ

ら