SYBASE®
An SAP® Company

Client-Library 移行ガイド

Open Client™

15.7

ドキュメント ID ： DC36064-01-1570-01
改訂： 2012 年 4 月

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™ DB-Library™ アプリケーションを Open Client Client-Library に移行する方法について説明します。

対象読者

このマニュアルでは、次の 2 つの役割のユーザを対象としています。

- 特定の DB-Library アプリケーションを Client-Library に移行することを決定する権限のあるマネージャ
- 移行を行う、経験のある DB-Library プログラマ

このマニュアルの内容

このマニュアルには、以下の章があります。

- 「第 1 章 Client-Library の概要」では、Client-Library の概要を紹介し、Client-Library 固有の機能を説明します。
- 「第 2 章移行のためのアプリケーション評価」では、DB-Library アプリケーションを Client-Library に移行すべきかどうかを判断する際のガイドラインを示します。
- 「第 3 章移行の計画」では、移行の計画に役立つ情報を提供します。
- 「第 4 章 DB-Library インフラストラクチャと Client-Library インフラストラクチャの比較」では、DB-Library と Client-Library のインフラストラクチャを比較します。
- 「第 5 章 DB-Library アプリケーション・コードの変換」では、Client-Library を使用した基本的な DB-Library のタスクの実行について説明します。
- 「第 6 章拡張機能」では、Client-Library の拡張機能についての情報を提供します。
- 「付録 A DB-Library ルーチンの Client-Library ルーチンへのマップ」では、DB-Library ルーチンを Client-Library にマップします。

関連マニュアル

詳細については、これらのマニュアルを参照できます。

- 『Open Server および SDK 新機能 Windows、Linux、UNIX 版』では、Open Server と Software Developer's Kit の新機能について説明しています。このマニュアルは、新機能の提供に伴って改訂されます。
- 使用しているプラットフォームの Open Server の『リリース・ノート』には、Open Server に関する重要な最新情報が記載されています。
- 使用しているプラットフォームの『Software Developer's Kit リリース・ノート』には、Open Client™ および SDK に関する重要な最新情報が記載されています。
- 『jConnect™ for JDBC™ リリース・ノート』には、jConnect に関する重要な最新情報が記載されています。
- 使用しているプラットフォームの『Open Client/Server 設定ガイド』では、システムを設定して Open Client/Server 製品を実行する方法について説明しています。
- 『Open Client Client-Library/C プログラマーズ・ガイド』では、Client-Library アプリケーションの設計方法および実装方法について説明しています。
- 『Open Client Client-Library/C リファレンス・マニュアル』では、Open Client Client-Library™ のリファレンス情報について説明しています。
- 『Open Server Server-Library/C リファレンス・マニュアル』では、Open Server Server-Library のリファレンス情報について説明しています。
- 『Open Client および Open Server Common Libraries リファレンス・マニュアル』では、CS-Library のリファレンス情報について説明しています。CS-Library は、Client-Library と Server-Library の両方のアプリケーションで役に立つユーティリティ・ルーチンの集まりです。
- 『Open Server DB-Library/C リファレンス・マニュアル』では、C バージョンの Open Client DB-Library™ のリファレンス情報について説明しています。
- 『Open Client/Server プログラマーズ・ガイド補足』では、Open Client/Server を使用するプログラマのために、プラットフォーム固有の情報について説明しています。このマニュアルには、次の情報が含まれています。

  - アプリケーションのコンパイルおよびリンク
  - Open Client/Server に含まれているサンプル・プログラム
  - プラットフォーム固有の動作をするルーチン
- 『Sybase® SDK DB-Library Kerberos 認証オプションのインストールおよびリリース・ノート』では、DB-Library で使用する MIT Kerberos セキュリティ メカニズムをインストールして有効化にする方法について説明しています。DB-Library でサポートされる Kerberos セキュリティ・メカニズムの機能は、ネットワーク認証サービスと相互認証サービスのみです。
- 『Open Client/Server 開発者用国際化ガイド』では、国際化されたアプリケーションとローカライズされたアプリケーションを作成する方法について説明しています。
- 『Open Client Embedded SQL™/C プログラマーズ・ガイド』では、C アプリケーションで Embedded SQL および Embedded SQL プリコンパイラを使用する方法について説明しています。
- 『Open Client Embedded SQL™/COBOL プログラマーズ・ガイド』では、COBOL アプリケーションで Embedded SQL および Embedded SQL プリコンパイラを使用する方法について説明しています。
- 『jConnect for JDBC プログラマーズ・リファレンス』では、jConnect for JDBC 製品について説明し、リレーショナル・データベース管理システムに保管されているデータにアクセスする方法について説明しています。
- 『Adaptive Server® Enterprise ADO.NET Data Provider ユーザーズ・ガイド』では、C#、Visual Basic .NET、マネージ拡張を備えた C++、J# など、.NET でサポートされる任意の言語を使用して Adaptive Server 内のデータにアクセスする方法について説明しています。
- Sybase® 製 Adaptive Server Enterprise *ODBC* ドライバの『ユーザーズ・ガイド』(Microsoft Windows および UNIX 版) では、Microsoft Windows および UNIX プラットフォームの Adaptive Server から、Open Database Connectivity (ODBC) ドライバを使用してデータにアクセスする方法について説明します。

- Sybase 製 *Adaptive Server Enterprise OLE DB* プロバイダの『ユーザーズ・ガイド』(Microsoft Windows 版) では、Microsoft Windows プラットフォームの Adaptive Server から、Adaptive Server OLE DB プロバイダを使用してデータにアクセスする方法について説明します。
- 『Perl 用 Adaptive Server Enterprise データベース・ドライバ・プログラマーズ・ガイド』では、Perl 開発者が Perl スクリプトを使用して Adaptive Server のデータベースに接続し、情報をクエリまたは変更する方法について説明しています。
- 『PHP 用 Adaptive Server Enterprise 拡張モジュール・プログラマーズ・ガイド』では、PHP 開発者が Adaptive Server データベースに対してクエリを実行する方法について説明しています。
- 『Python 用 Adaptive Server Enterprise 拡張モジュール・プログラマーズ・ガイド』では、Adaptive Server データベースに対してクエリを実行するときに使用できる Sybase 固有の Python インタフェースについて説明しています。

その他の情報

Sybase Getting Started CD および Sybase Product Documentation 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、リリース・ノートとインストール・ガイドが PDF 形式で含まれています。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)。
- Sybase Product Documentation Web サイトには、標準の Web ブラウザを使用してアクセスできます。また、製品ドキュメントのほか、EBFs/Maintenance、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。

  Sybase Product Documentation Web サイトは、Product Documentation (http://www.sybase.com/support/manuals/) にあります。

プログラム例でのリターン・コード・エラー・チェック

このマニュアルでは、Sybase が World Wide Web で提供する移行サンプル・プログラム (Migration Example Programs) から、多数のサンプル・プログラムを引用しています。

このマニュアルのサンプル・プログラムでは、次のような EXIT_ON_FAIL() サンプル・マクロを使用します。これと同様のマクロを使用することで、リターン・コード・エラー・チェックを簡易化できます。ただし、状況によりこのマクロが適切ではない場合もあります。

```
/*
** Define a macro that exits if a function return code indicates
** failure.Accepts a CS_CONTEXT pointer, a Client-Library
** or CS-Library return code, and an error string.If the
** return code is not CS_SUCCEED, the context will be
** cleaned up (if it is non-NULL), the error message is
** printed, and we exit to the operating system.
*/
#define EXIT_ON_FAIL(context, ret, str) {
   if (ret != CS_SUCCEED)
   {
      fprintf(stderr, "Fatal error:%s¥n", str);
      if (context != (CS_CONTEXT *) NULL) {
         (CS_VOID) ct_exit(context, CS_FORCE_EXIT);
         (CS_VOID) cs_ctx_drop(context);
      } ¥
      exit(ERROR_EXIT);
   }
}
```

World Wide Web のアクセス

移行のサンプル・プログラムは Sybase World Wide Web page (http://www.sybase.com/detail?id=1013159) を参照してください。サンプル・プログラムは、次の Open Server™ インストール・ディレクトリにもあります。

UNIX の場合：*$SYBASE/$SYBASE_OCS/sample/db2ct*

Windows の場合：*%SYBASE% ¥%SYBASE_OCS% ¥sample ¥db2ct*

Migration Example Programs とともに提供されている *README* ファイルには、ファイル例の記述リストが入っています。

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 (http://www.sybase.com/support/techdocs/) を指定します。
2. [Partner Certification Report] をクリックします。
3. [Partner Certification Report] フィルタで製品、プラットフォーム、時間枠を指定して [Go] をクリックします。
4. [Partner Certification Report] のタイトルをクリックして、レポートを表示します。

❖ **コンポーネント認定の最新情報にアクセスする**

1 Web ブラウザで Availability and Certification Reports (http://certification.sybase.com/) を指定します。

2 [Search By Base Product] で製品ファミリとベース製品を選択するか、[Search by Platform] でプラットフォームとベース製品を選択します。.

3 [Search] をクリックして、入手状況と認定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 (http://www.sybase.com/support/techdocs/) を指定します。

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 the Sybase Support Page (http://www.sybase.com/support) を指定します。

2 [EBFs/Maintenance] を選択します。MySybase のユーザ名とパスワードを入力します。

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。EBF/Maintenance リリースの一覧が表示されます。

鍵のアイコンは、「Technical Support Contact」として登録されていないため、一部の EBF/Maintenance リリースをダウンロードする権限がないことを示しています。未登録でも、Sybase 担当者またはサポート・コンタクトから有効な情報を得ている場合は、[Edit Roles] をクリックして、「Technical Support Contact」の役割を MySybase プロファイルに追加します。

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。

表記規則

表 1：構文の表記規則

凡例	定義
コマンド	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、キーワードは sans serif で示す。
変数	変数 (ユーザが入力する値を表す語) 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。コマンドには中カッコは入力しない。
( )	このカッコはコマンドの一部として入力する。
|	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。
,	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。複数のオプションを選択する場合には、オプションをカンマで区切る。

アクセシビリティ機能

このマニュアルには、アクセシビリティを重視した HTML 版もあります。この HTML 版マニュアルは、スクリーン・リーダーで読み上げる、または画面を拡大表示するなどの方法により、その内容を理解できるよう配慮されています。

Open Client および Open Server のマニュアルは、連邦リハビリテーション法第 508 条のアクセシビリティ規定に準拠していることがテストにより確認されています。第 508 条に準拠しているマニュアルは通常、World Wide Web Consortium (W3C) の Web サイト用ガイドラインなど、米国以外のアクセシビリティ・ガイドラインにも準拠しています。

注意 アクセシビリティ・ツールを効率的に使用するには、設定が必要な場合もあります。一部のスクリーン・リーダーは、テキストの大文字と小文字を区別して発音します。たとえば、すべて大文字のテキスト (ALL UPPERCASE TEXT など) はイニシャルで発音し、大文字と小文字の混在したテキスト (Mixed Case Text など) は単語として発音します。構文規則を発音するようにツールを設定すると便利かもしれません。詳細については、ツールのマニュアルを参照してください。

Sybase のアクセシビリティに対する取り組みについては、Sybase Accessibility (http://www.sybase.com/accessibility) を参照してください。Sybase Accessibility サイトには、第 508 条と W3C 標準に関する情報へのリンクもあります。

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

Client-Library の概要

この章では、Client-Library の概要を紹介し、Client-Library 固有の機能を説明します。

この章の内容は、次のとおりです。

Client-Library とは

Client-Library は、クライアント・アプリケーションを作成するときに使用するアプリケーション・プログラミング・インタフェース (API) です。Client-Library は、非データベース・アプリケーションを含む分散クライアント・アプリケーションを構成するための汎用構築ブロックを提供します。

Sybase は DB-Library、ODBC、Embedded SQL™ など、他のクライアント・インタフェースもいくつかサポートしていますが、Client-Library を使用するとアプリケーション・プログラマにとって次のような大きな利点があります。

- クエリ言語からもデータベースからも独立しているので、アプリケーション・プログラマは柔軟性に富んだ高処理のアプリケーションを幅広く作成できます。
- タイプ定義、各種定義、データ要素記述を Sybase の Open Server Server-Library インタフェースと共有するので、アプリケーション・プログラマはクライアント機能を Server-Library アプリケーション内に統合できます。
- 非同期インタフェースを提供するので、アプリケーション・プログラマは複数の作業要求を同時に実行するアプリケーションを作成できます。

- プログラマは、アプリケーション自体に変更を加えずにランタイム設定ファイルの設定プロパティを設定できます。

Client-Library は、新規 Sybase カスタマおよび新しいアプリケーションを作成するカスタマにおすすめできる API です。既存の DB-Library アプリケーションを使用しているカスタマの場合、Client-Library への移行は、既存アプリケーションが新しい Sybase 機能にアクセスする必要があるかどうか、移行にどのくらいの作業が必要かを考えて、決定してください。

クライアント・インタフェースの比較

表 1-1 では、Sybase のクライアント・インタフェースを比較します。

表 1-1：Sybase のクライアント・インタフェースの比較

	Client-Library	DB-Library	Embedded SQL	ODBC
使用可能なクライアント／サーバ機能	すべて	DB-Library では、次のものを除き、Client-Library バージョン 11.0 以降で追加された新しい機能をサポートしていない • サーバ接続情報を指定する dbsetconnect ルーチン • 実行時の DB-Library バージョン・レベルを外部から設定する SYBOCS_DBVERSION 環境変数 • Windows プラットフォームでの LDAP ディレクトリ・サービス・サポート • Linux、Solaris、Microsoft Windows プラットフォームでの MIT Kerberos ネットワーク認証サービスと相互認証サービス これらの機能については、『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。	データ・ストリーム・メッセージ機能、レジスタード・プロシージャ・ノーティフィケーションを除くすべての機能	DB-Library とほぼ同じ。実装によって提供される機能セットが異なったり、同じ機能が異なる方法で実装される場合がある。

	Client-Library	DB-Library	Embedded SQL	ODBC
クエリ言語は独立しているか	はい	いいえ	いいえ	いいえ
非データベース開発をサポートするか	はい	いいえ	はい	いいえ
インタフェース方式	同期または非同期	同期	同期	同期
主な長所	高処理、汎用、移植可能	簡単、移植可能	簡単、移植可能、国際標準規格	簡単、普及率が高い
主な短所	新しいインタフェースなので習得に多少時間がかかる	Sybase 固有のインタフェースであり、すべての汎用クライアント／サーバ・サービスをサポートするわけではない	CLI (Call level interface) より柔軟性が劣る	すべての実装に使用できる同一の準拠テストがないため、機能サポートのレベルが混合している

Client-Library 固有の機能

Sybase のクライアント・インタフェースの中でも、Client-Library は次の機能をサポートする唯一のインタフェースです。

- Open Server との緊密な統合
- サーバ側カーソルに対するクライアント・インタフェース
- 動的 SQL に対するクライアント・インタフェース
- 非同期モードのオペレーション
- マルチスレッド・アプリケーション・サポート
- ネットワーク・ベースのディレクトリ・サービスとセキュリティ・サービスのサポート
- ユーザ定義データ型と変換ルーチン
- ローカライゼーション・メカニズム
- 簡素化されたインタフェース

Open Server との緊密な統合

Client-Library と Server-Library は、パブリックのタイプ定義、マクロ、データ要素記述を共有します。さらに、Client-Library アプリケーションも Server-Library アプリケーションも、CS-Library ルーチンを使用して、共通のデータ構造体を割り付け、ローカライゼーションを処理し、データ値を変換します。

このような緊密な統合により、Server-Library とゲートウェイ・アプリケーションに、Client-Library ベースの機能を組み込むことができます。

サーバ側カーソルに対するクライアント・インタフェース

カーソルはきわめて有用なデータ管理ツールです。カーソルを使うことにより、クライアント・アプリケーションは、結果セットを処理しながら、結果ローを個別に更新できます。サーバ側カーソルは Adaptive Server Enterprise に存在するカーソルで、「ネイティブ・カーソル」と呼ばれることもあります。

Client-Library はサーバ側カーソルを完全にサポートし、クライアント・アプリケーションによるサーバ側カーソルの宣言、オープン、操作を可能にする CLI (Call-level Interface) を提供します。

DB-Library はサーバ側カーソルをサポートしません。代わりに、「クライアント側カーソル」と呼ばれる、ある種のカーソル・エミュレーションをサポートします。クライアント側カーソルは実際の Adaptive Server Enterprise カーソルとは対応していません。このため、DB-Library はローを内部でバッファして、キーセットの管理、ローの位置付け、同時制御など、カーソルを管理するのに必要なすべての操作を行います。

Client-Library のカーソル機能は DB-Library のロー・バッファリング機能に代わるものです。DB-Library のロー・バッファリングでは、バッファ内の各ローが個別に割り付けられたり解放されたりするので、メモリとパフォーマンスには悪影響があります。

動的 SQL に対するクライアント・インタフェース

動的 SQL によって、アプリケーションは、コンパイルされた SQL 文 (「準備文」とも呼ばれます) をサーバ上に作成し、自由に実行できます。このような文には、アプリケーションのエンド・ユーザが実行時に値を入力できるプレースホルダ変数を含めることができます。文に値が入力されている場合、クライアント・アプリケーションは、該当する文の入力値のフォーマットをサーバに問い合わせることができます。

Client-Library は動的 SQL を完全にサポートし、ANSI 標準の Embedded SQL の prepare、execute、execute immediate 文を組み込んだ CLI (Call-level Interface) を提供します。また、Client-Library は、アプリケーションによる準備文の入出力の記述の取得を可能にします。

クライアント・アプリケーションでは通常、動的 SQL を使用して、エンド・ユーザが実行時に SQL 文をカスタマイズできるようにします。たとえばあるアプリケーションが、特定のカスタマに関して知られている情報をすべて検索する SQL クエリを作成するとします。このクエリは、カスタマの名前をプレースホルダ変数とする動的 SQL 文として準備されます。実行時に、アプリケーションのエンド・ユーザがカスタマの名前を入力し、この準備文を実行します。

非同期モード

Client-Library の非同期モードによって、アプリケーションはある種のオペレーションが完了するまでの待ち時間を有効に利用できます。一般に、ネットワークまたは外部デバイスからの読み書きは、単純なプログラム実行よりもはるかに時間がかかります。

非同期動作が有効になっていると、プログラムの実行をブロックする可能性のある Client-Library ルーチンは、すべて非同期で動作します。つまり、次のどちらかの動作になります。

- 要求されたオペレーションを起動して、ただちに戻る。
- 非同期オペレーションがすでに保留になっているという情報とともに、ただちに戻る。

アプリケーションは、次の 2 つのモデルのどちらかを使用して、オペレーションの完了を認識します。

- 非ポーリング (割り込み駆動型)
- ポーリング

非ポーリング (割り込み駆動型)

非ポーリング・モデルは、割り込み駆動型 I/O またはマルチスレッド機能をサポートするプラットフォームで使用可能です。該当するプラットフォームは、すべての UNIX および Microsoft Windows プラットフォームです。

非同期オペレーションが完了すると、Client-Library はプログラマ・インストールの完了コールバック・ルーチンを自動的にトリガします。完了コールバック・ルーチンは通常、アプリケーションのメイン・コードに非同期ルーチンの完了を通知します。

ポーリング

ポーリング・モデルはすべてのプラットフォームで使用可能です。移植性が問題である場合には、ポーリング・モデルをおすすめします。

ポーリング・モデルの場合、アプリケーションは ct_poll を呼び出して、非同期オペレーションが完了しているかどうかを判断します。完了している場合、ct_poll はプログラマ・インストールの完了コールバック・ルーチンを自動的にトリガします。

マルチスレッド・アプリケーション・サポート

Client-Library ではリエントラント・ライブラリが提供され、これによってほとんどのプラットフォームでスレッドセーフ・アプリケーションがサポートされます。場合によっては、Client-Library 開発者はマルチスレッド設計を使用して、応答時間を短縮したり、処理能力を高めたりできます。次に例を示します。

- 対話型 Client-Library アプリケーションの場合、あるスレッドを使用してサーバに問い合わせを行う一方で、別のスレッドを使用してユーザ・インタフェースを管理することができます。このようなアプリケーションは、ユーザに対する応答能力がより優れているといえます。これは、クエリ・スレッドが結果を待っている間も、ユーザ・インタフェース・スレッドがユーザ・アクションに応答できるからです。

- 1 つまたは複数のサーバに対して複数の接続を使用するアプリケーションは、各接続を専用スレッド内で実行できます。1 つのスレッドがコマンド結果を待っている間に、別のスレッドは受け取った結果を処理したり、新しいコマンドを送信したりできます。このような処理方式では、アプリケーションが結果を待つアイドル時間が少なくなるので、処理能力が向上します。

どのシステム・スレッド・ライブラリが、使用しているプラットフォームの Client-Library とリンク可能であるかについては、『Open Client/Server プログラマーズ・ガイド補足』の「Client-Library」の章を参照してください。

マルチスレッド・アプリケーションに Client-Library 呼び出しをコーディング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』の「マルチスレッド・プログラミング」を参照してください。

ネットワーク・ベースのセキュリティ・サービスとディレクトリ・サービスのサポート

Client-Library と Server-Library では、アプリケーションは分散ネットワーク・セキュリティ・サービスとディレクトリ・サービスを利用できます。

セキュリティ・サービス

クライアント／サーバ・アプリケーションは、Sybase 提供のセキュリティ・ドライバを使用して、CyberSafe Kerberos、MIT Kerberos、Secure Sockets Layer (SSL)、Microsoft Windows LAN Manager などの分散ネットワーク・セキュリティ・ソフトウェアと統合できます。この統合によって、アプリケーションは次のようなネットワーク・ベースのセキュリティ機能を使用できます。

- 集中型ユーザ認証：アプリケーション・ユーザ名とパスワードは、各 Sybase サーバではなく、ネットワーク・セキュリティ・システムによって管理されます。ユーザは、ネットワーク・セキュリティ・システムにログインすることになり、サーバにログインするときはパスワードを提示する必要がありません。

- 安全が確保されていないネットワークでの安全な接続：Client-Library と Server-Library は、ネットワーク・セキュリティ・システムと対話して、暗号または整合性の確認などのパケット単位のセキュリティ・サービスを実行できます。これらのサービスによって、アプリケーションは、物理的に安全とはいえない通信媒体 (無線サービスや専用回線など) を介して機密データやコマンドを安全に送信できます。

ディレクトリ・サービス

Lightweight Directory Access Protocol (LDAP) などのネットワーク・ベースのディレクトリ・ソフトウェアは、複数の interfaces ファイルを管理する代替方法を提供します。アプリケーションは、Sybase 提供のディレクトリ・ドライバを使用して、ディレクトリ・プロバイダ・ソフトウェアと通信し、ネットワーク・アドレスの中から、指定された Sybase サーバを探します。

その他の情報

使用しているシステムで使用可能なディレクトリおよびセキュリティ・ドライバと、それらの設定方法については、『Open Client/Server 設定ガイド』を参照してください。

ネットワーク・ベース・ディレクトリとセキュリティ・サービスを使用する場合のアプリケーションのコーディング方法については、『Open Client Client-Library/C リファレンス・マニュアル』の次の項を参照してください。

- 「ディレクトリ・サービス」
- 「セキュリティ機能」

ユーザ定義データ型と変換ルーチン

アプリケーションはユーザ定義データ型を使用することがよくあります。Client-Library によって、アプリケーションがユーザ定義のデータ型を作成するのも変換するのも簡単になります。

- Client-Library アプリケーションでは、ユーザ定義データ型は C 言語型です。ユーザ定義データ型を作成するには、アプリケーションはただそのデータ型を宣言するだけです (Client-Library のユーザ定義型と Adaptive Server Enterprise のユーザ定義型を混同しないでください。Adaptive Server Enterprise のユーザ定義型はシステム・ストアド・プロシージャ sp_addtype で作成されるデータベース・カラム・データ型です)。
- ユーザ定義型間およびユーザ定義型と標準 Client-Library データ型間の変換では、カスタム変換ルーチンを書き、それらのルーチンを Client-Library にインストールするコードを追加することができます。変換ルーチンは一度インストールしてしまえば、あとは Client-Library がカスタム・ルーチンを呼び出して、すべての変換を処理してくれます。

ユーザ定義型に関する

CS-Library ルーチンには、次のようなルーチンがあります。

- cs_set_convert － 標準 Open Client データ型とユーザ定義データ型間の変換を行うカスタム変換ルーチンをインストールします。
- cs_will_convert － データ型の変換がサポートされるかどうかを示します。
- cs_setnull － ユーザ定義データ型の null 代入値を定義します。

ローカライゼーション・メカニズム

国際化アプリケーションは、変更を加えずに複数言語環境で実行できます。それぞれの環境で、外部設定ファイルや環境変数などの外部情報を使用して、アプリケーションがローカライズを行います。つまり、使用する言語、文字セット、日時フォーマット、通貨フォーマットを決定します。

Client-Library には、国際化アプリケーションの開発を容易にする強力なローカライゼーション・メカニズムが用意されています。

- ロケール・ファイルはロケール名を言語／文字セット／ソート順の組み合わせにマップします。

- アプリケーションは、実行時に環境変数の値を確認して、どのロケールを使用するかを決定できます。
- 1 つのアプリケーションで、部分ごとに異なるロケールを使用できます。たとえば、フランスではフランス語で、イタリアではイタリア語で動作する国際化販売アプリケーションても、ロンドン営業所では英語ロケールを使用してレポートを作成することができます。

簡素化されたインタフェース

Client-Library は簡素化されたインタフェースです。DB-Library が 200 以上のルーチンを持っているのに対して、Client-Library も CS-Library も合わせて 64 に満たないルーチンしか持っていません (どちらの場合もバルク・コピー・ルーチンは数えていません)。

また、Client-Library では、統一結果処理モデルが用意されています。このモデルではアプリケーションはすべてのタイプの結果を処理するのに同じルーチンを使用します。

Client-Library は、その規模と一貫性のある設計により、いっそう使いやすくなっています。

第 2 章

移行のためのアプリケーション評価

この章では、 DB-Library アプリケーションを Client-Library に移行すべきかどうかを判断する際のガイドラインを示します。

考慮すべき事項

DB-Library アプリケーションを Client-Library に移行すべきかどうかを判断するときに留意する必要のある事項は、主に次の 2 つです。

- 移行はアプリケーションにとって有益か
- 移行に要する作業はどのくらいか

これらの点を考慮した上で、移行の利点と必要な作業量を比較検討して、移行すべきかどうかを決定してください。

移行はアプリケーションにとって有益か

強化機能または新しい Sybase 機能へのアクセスを必要とするアプリケーションの場合、 Client-Library への移行は一般的に次のような利点があります。

- Client-Library は、現行のすべての Sybase サーバ機能をサポートするだけでなく、独自の優れた機能を数多く持っています (「 Client-Library 固有の機能」 (3 ページ) を参照) 。
- Client-Library は、リエントラント・ライブラリによってスレッドセーフ・アプリケーションをサポートします。 DB-Library はこれらをサポートしません。
- Client-Library はネットワーク・ベースのディレクトリ・サービスとセキュリティ・サービスをサポートします。一方、 DB-Library はこれらをサポートしません (詳細については、 「ネットワーク・ベースのセキュリティ・サービスとディレクトリ・サービスのサポート」 (7 ページ) を参照してください) 。

強化機能または新しい Sybase 機能へのアクセスを必要としないアプリケーションには、移行による利点はありません。

移行に要する作業はどのくらいか

DB-Library から Client-Library への移行にどのくらいの作業が必要かを知るには、 DB-Library アプリケーションがどのようなタスクを実行し、どのようなルーチンを使用するか、ということから調べる必要があります。

DB-Library のタスクの中には、 SQL コマンドのサーバへの送信など、どちらのライブラリでも簡単にできるものもあります。 Open Server レジスタード・プロシージャの使用など、 Client-Library ではより複雑になるタスクもあります。

表 2-1 は、標準的な DB-Library アプリケーション・タスクを、 Client-Library で同じアプリケーション機能を行うのに必要な作業量に従って分類したものです。

表 2-1 ：必要な移行作業別に分類した DB-Library タスク

DB-Library タスク	関連ルーチンの部分リスト	移行に必要な作業量	注意
Transact-SQL 言語コマンドのサーバへの送信	dbcmd 、 dbfcmd 、 dbsqlexec	平均以下	言語コマンドの送信は Client-Library でも簡単に行える。
RPC コマンドのサーバへの送信	dbrpcinit 、 dbrpcparam 、 dbrpcsend	平均以下	RPC コマンドの送信は Client-Library でも簡単に行える。
サーバからの text および image データの挿入、検索	dbreadtext 、 dbwritetext 、 dbtxtptr 、 dbtxtimestamp	平均以下	Client-Library は text および image データを DB-Library よりもスムーズに処理する。
日時値の操作	dbdatename 、 dbdatepart 、 dbdatezero	平均	Client-Library には、これらに相当するルーチンは用意されていない。代わりに、 cs_dt_crack と cs_dt_info を使用する。

DB-Library タスク	関連ルーチンの部分リスト	移行に必要な作業量	注意
結果ローの自動フォーマット機能	dbprhead 、dbprrow 、dbspr1row 、dbsprhead	平均	Client-Library には、同等のルーチンは用意されていないが、*exutils.c* Client-Library サンプル・プログラムに示すようなアプリケーション・コードで簡単に置き換えることができる。 これらのルーチンをデバッグ目的で使用するアプリケーションは、代わりに ct_debug を使用できる。
バルク・コピー・オペレーション	バルク・コピー・ルーチン	平均	DB-Library の bcp_ ルーチンには、組み込みファイル I/O ルーチンが含まれている。このルーチンがホスト・データ・ファイルとフォーマット・ファイルの読み書き、およびエラー・ファイルの書き込みを行う。 Client-Library アプリケーションは Bulk-Library を使用するが、これにはファイル I/O ルーチンは含まれていない。
結果データをバインドする代わりに、結果データへのポインタを使用する	dbdata 、 dbadata	平均	現時点では、 Client-Library アプリケーションは、アプリケーションのデータ領域内のメモリに結果をバインドする必要がある。
ロー・バッファリング	DBCURROW 、DBFIRSTROW 、DBLASTROW 、dbsetrow	平均以上	Client-Library には、ロー・バッファリングの代わりに、カーソル機能と配列バインド機能が用意されている。ロー・バッファリングの代わりにカーソルを使用するには、多少のアプリケーション再設計作業が必要である。

DB-Library タスク	関連ルーチンの部分リスト	移行に必要な作業量	注意
レジスタード・プロシージャ	dbnpcreate 、dbnpdefine 、dbregdrop	平均以上	Client-Library には、同等のルーチンは用意されていない。 Client-Library アプリケーションは、 RPC コマンドを送信して Open Server システム・レジスタード・プロシージャ sp_regcreate および sp_regdrop を呼び出し、レジスタード・プロシージャの作成および削除を行う。
Adaptive Server Enterprise ページの読み書き	dbreadpage 、dbwritepage	変換なし	Client-Library はこの機能をサポートしない。
2 フェーズ・コミット	2 フェーズ・コミット・ルーチン	変換なし	Client-Library には、同等のルーチンは用意されていない。代わりに、 Client-Library はトランザクションを制御するトランザクション・モニタをサポートする。 詳細については、以下のディレクトリにある 2 フェーズ・コミットのサンプル・プログラムを参照。 • *$SYBASE/$SYBASE_OCS/sample/ctlibrary* (UNIX プラットフォーム) • *%SYBASE% ¥ %SYBASE_OCS% ¥ sample ¥ ctlib* (Windows)

概要

Client-Library は汎用インタフェースであるため、これを使用するアプリケーションは Sybase および業界の新技術を利用できるようになる、という利点があります。移行がそれに伴う作業量に値するかどうかを判断する際には、このことを考慮に入れてください。

アプリケーションがまだ発展途上にある、つまり、今後のニーズに応じて変化する可能性があるような場合には、移行は十分検討に値します。

第 3 章

移行の計画

この章では、移行の計画に役立つ情報を提供します。

ソフトウェアの入手

Client-Library も DB-Library も Software Developer's Kit の一部として提供されます。

このキットには、次のソフトウェア・コンポーネントが含まれています。

- 運用ライブラリ

  運用ライブラリは、DB-Library および Client-Library の運用アプリケーションの実行時ライブラリです。Microsoft システムの場合、このライブラリはインポート・ライブラリと DLLs です。UNIX システムの場合、これは静的な共有オブジェクト・ライブラリです。

- 開発ライブラリ

  開発ライブラリには、Client-Library ルーチン ct_debug のデバッグ記号とトレース・コードが含まれています。

- Bulk-Library、Embedded SQL/C (ESQL/C)、ESQL/COBOL
- インクルード・ファイル
- サンプル・プログラム

  Client-Library には、Client-Library の機能を例示する優れたサンプル・ファイルがいくつか含まれています。詳細については、使用しているプラットフォームの『Open Client/Server プログラマーズ・ガイド補足』を参照してください。
- Net-Library ドライバ

Client-Library の学習

コーディングを始める前に Client-Library についての理解を深めておけば、移行作業はスムーズに運びます。

Client-Library について学習する機会および資料としては、次のような講座とマニュアルが用意されています。

- サイベース株式会社プロフェッショナル・サービス部が定期的に開催しているトレーニング・セミナ (Client-Library コース)。詳細については、プロフェッショナル・サービス部にお問い合わせください。
- ソフトウェアには、サンプルの Client-Library プログラムが付属します。
- 『Open Client Client-Library/C プログラマーズ・ガイド』。このマニュアルには、Client-Library プログラムの構築方法に関する基本的な情報が含まれています。
- 以下の章には、Client-Library を使用して特定の DB-Library アプリケーション・タスクを実行する方法が説明されています。
  - 「第 4 章 DB-Library インフラストラクチャと Client-Library インフラストラクチャの比較」
  - 「第 5 章 DB-Library アプリケーション・コードの変換」
  - 「第 6 章 拡張機能」

  特に、「第 5 章 DB-Library アプリケーション・コードの変換」には、一般的なアプリケーション・タスクの DB-Library と Client-Library の呼び出しシーケンスが対比してリストされています。

サンプル・プログラムの利用

Sybase では、DB-Library コードを Client-Library に変換する方法を理解できるように、Sybase の Web サイト (Sybase Web site (http://www.sybase.com/detail?id=1013159)) に一連の Migration Example Programs (移行サンプル・プログラム) を用意しています。

DB-Library コードの切り離し

できれば、移行作業を開始する前に、DB-Library コードを他のアプリケーション・コードから切り離してください。DB-Library コードを別のルーチンまたはモジュールに置く方が、評価も置換も簡単です。また、変換されたコードの移行後のデバッグも簡単です。

DB-Library コードを切り離すためにコード変更を行う場合には、Client-Library 機能を導入する前に、アプリケーションをテストして、変更したコードが正しく機能するかどうかを確認してください。

アプリケーション再設計の検討

移行は、DB-Library ではサポートしない Client-Library 機能を利用できるようにアプリケーションを再設計するいい機会でもあります。この機会に、新しい Adaptive Server Enterprise 機能も利用できるように、アプリケーションを再設計できます。

以降では、どのような場合に再設計を検討すべきかを説明します。

統一結果の処理

DB-Library は統一結果処理モデルを使用しません。DB-Library では、アプリケーションは、呼び出すルーチンによって、異なるタイプの結果を取得します。

- 通常のローの結果カラムは dbbind でバインドされますが、計算ローの結果カラムは dbaltbind でバインドされます。

- 通常のローのデータおよび計算ローのデータは dbnextrow でフェッチされますが、ストアド・プロシージャ・リターン・パラメータは dbretdata で取得されます。

これに対して、Client-Library では次のようになります。

- すべてのタイプのフェッチ可能データが ct_bind でバインドされ、ct_fetch

  でフェッチされます。

- 統一結果処理モデルによって、アプリケーションによる結果処理コードの統合ができるようになります。

「結果を処理するコード」(61 ページ) を参照してください。

カーソル

Client-Library (サーバ側) カーソルは、以下のいくつかの DB-Library 機能の代わりになります。

- DB-Library カーソル

  Client-Library (サーバ側) カーソルは DB-Library カーソルより高速です。Client-Library にはスクロール可能カーソルがあり、結果セットの任意の場所にカーソルの位置を設定できます。結果セットへの絶対または相対ロー番号オフセットを使って、結果セット内で任意の現在位置から前後にナビゲーションできます。さらに、結果セット内で NEXT、FIRST、LAST、PREVIOUS のようなフェッチ方向を使って、今後の処理用に単一ローを選択することもできます。

- DB-Library ブラウズ・モード

  Client-Library はブラウズ・モードもサポートしますが、カーソルは同じ機能を、より移植性の高い、より柔軟な方法で提供します。

カーソルまたはブラウズ・モードを使用する DB-Library アプリケーションは、Client-Library (サーバ側) カーソルを使用するように再設計するのが得策です。

「Client-Library カーソル」(78 ページ) を参照してください。

配列バインド

Client-Library の配列バインド機能によって、アプリケーションは結果カラムをプログラム変数の配列にバインドできます。この結果、1 回の ct_fetch 呼び出しで、複数ローに相当するカラム値がフェッチされます。

配列バインドによって、アプリケーション・パフォーマンスは向上します。特に、結果セットが大きく (21 ロー以上) で、小さいカラム (合計ロー・サイズが 512 バイト未満) が少ししか含まれていない場合には、大きな効果が得られます。

配列サイズは 4 ～ 16 が最も効果的です。配列サイズをこれより大きくしても、処理能力が大幅に向上することはありません。

ロー・バッファリングを使用する DB-Library アプリケーションは、ロー・バッファリングの代わりに Client-Library 配列バインド機能を使用することがよくあります。

「Client-Library の配列バインド」(77 ページ) を参照してください。

非同期モード

Client-Library の非同期モードでは、アプリケーションはブロックする可能性のあるオペレーションを非同期で実行できます。これは、GUI インタフェースを使用するエンド・ユーザ・アプリケーションにとって非常に有用な機能です。アプリケーション・ユーザはブロックされたオペレーションの完了を待つ間に他の作業を進めることができるからです。

同期 DB-Library アプリケーションは、一般に、非同期 Client-Library アプリケーションとして再設計すると、機能が向上します。

「非同期プログラミング」(85 ページ) を参照してください。

マルチスレッド機能

マルチスレッド機能によって、対話型アプリケーションの応答時間が短縮でき、バッチ処理アプリケーションの処理能力が向上します。「マルチスレッド・アプリケーション・サポート」(6 ページ) を参照してください。

移行作業の見積もりの確認

Client-Library について理解し、アプリケーションの中にどのような種類の DB-Library コードがどのくらい含まれているかがわかり、アプリケーションのどの部分を再設計するかを決定したところで、前に行った移行作業見積もりを見直します。

再設計も移行時間に加算されますが、再設計はそれだけの作業をするに値します。

テストの計画

実際に移行を始める前に、テストの計画を立てて、テスト環境を構築します。テスト環境では、Client-Library アプリケーションのテスト結果と DB-Library アプリケーションのテスト結果を比較できるようにしてください。

スケジュールの作成

移行作業のスケジュールを作成する際には、まず難易度に応じて作業を分類し、それに従ってスケジュールを作成すると良いでしょう。

Sybase では、最初に最も簡単な移行作業、2 番目に最も困難な作業、そして 3 番目に中ぐらいのレベルの作業、という順序でスケジュールを立てることをおすすめします。

スケジュールに余裕がない場合は、最もむずかしい作業をあとまわしにしないでください。

環境の確認

移行環境が完成し、正しく設定されているかどうかを以下のように確認します。

- Client-Library がインストールされていますか。
- サーバのバージョンは正しいですか。
- サーバがアプリケーションをサポートするように設定されていますか。たとえば、暗黙カーソルを使用する予定がある場合は、バージョン 12.5 以降を使用します。正しい接続数の設定になっていますか。正しいデータベースがインストールされていますか。
- Client-Library サンプル・プログラムが正常に動作しますか。正常に動作しない場合は、問題点を修正してから作業を続行してください。
- テスト環境が設定されていますか。

この章で述べた計画作業が完了したら、いよいよコーディングに取りかかります。第 4 章、第 5 章、第 6 章には、コーディング段階で必要になる情報が記載されています。

- 「第 4 章 DB-Library インフラストラクチャと Client-Library インフラストラクチャの比較」では、ヘッダ・ファイル、ユーティリティ・ルーチン、データ構造体を比較します。
- 「第 5 章 DB-Library アプリケーション・コードの変換」では、基本的な DB-Library プログラミング・タスクを Client-Library で実現する方法を説明します。
- 「第 6 章 拡張機能」では、より上級のプログラミング・タスクを取り上げます。

第 4 章

DB-Library インフラストラクチャと Client-Library インフラストラクチャの比較

この章では、 DB-Library と Client-Library のインフラストラクチャを比較します。

ユーティリティ・ルーチン

DB-Library ユーティリティ・ルーチンは DB-Library の一部に含まれていますが、 Client-Library アプリケーションのユーティリティ・ルーチンは CS-Library から提供されます。

注意 dblib ベースの bcp 呼び出しは、 DOL テーブルまたは XNL テーブルに対してはサポートされません。開発者はこの点を考慮する必要があります。

CS-Library は共有 Open Client/Server ライブラリです。この中には、 Client-Library と Open Server Server-Library の両方のアプリケーションに使用できるルーチンが含まれています。

CS-Library には、次の機能をサポートするルーチンが含まれています。

- データ型変換 － cs_convert は dbconvert 呼び出しと置き換えることができます。
- 算術演算 － cs_calc は異なるいくつかの dbmny 呼び出しと置き換えることができます。

- 文字セット変換 ー cs_locale と cs_convert は dbload_xlate 呼び出しと dbxlate 呼び出しに置き換えることができます。
- 日時オペレーション ー cs_dt_crack は dbdtcrack 呼び出しと置き換えることができます。
- ソート順オペレーション ー cs_strcmp は dbstrsort 呼び出しと置き換えることができます。
- ローカライズしたエラー・メッセージ ー cs_strbuild は dbstrbuild 呼び出しと置き換えることができます。

CS-Library については、『 Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。

ヘッダ・ファイル

DB-Library は *sybfront.h* 、 *sybdb.h* 、 *syberror.h* ヘッダ・ファイルを使用します。

Client-Library は *ctpublic.h* ヘッダ・ファイルを使用します。

- *ctpublic.h* には、 CS-Library のヘッダ・ファイルである *cspublic.h* が含まれています。
- *cspublic.h* には、次のヘッダ・ファイルが含まれています。
  - Client-Library データ型の型定義が入っている *cstypes.h*
  - プラットフォーム依存のデータ型と定義が入っている *csconfig.h*
  - SQLCA 構造体の typedef が入っている *sqlca.h*

アプリケーションを移行するときに、 DB-Library ヘッダ・ファイル名を Client-Library ヘッダ・ファイル名 (*ctpublic.h*) に置き換えてください。

注意 必要な他のすべてのヘッダ・ファイルが含まれている *cspublic.h* が、 *ctpublic.h* に含まれているので、アプリケーション自体には、 *ctpublic.h* を含めるだけで済みます。

制御構造体

DB-Library は、 LOGINREC と DBPROCESS の 2 つの主制御構造体を使用します。

Client-Library は、 CS_CONTEXT 、 CS_CONNECTION 、 CS_COMMAND という 3 つの制御構造体を使用します。

- CS_CONTEXT 構造体は、アプリケーション・コンテキストまたはオペレーティング環境を定義します。
- CS_CONNECTION 構造体は、アプリケーション・コンテキスト内のクライアント／サーバ接続を定義します。 1 コンテキストに、複数の接続を定義できます。
- CS-COMMAND 構造体は、接続内のコマンド領域を定義します。 1 つの接続に、複数のコマンド構造体を定義できます。

CS_CONTEXT 構造体は、 DB-Library に同等のものはありませんが、 DB-Library 隠しグローバル変数に保管される情報と類似した情報を保管します。

CS_CONNECTION 構造体と CS_COMMAND 構造体は、 2 つ合わせて、 DBPROCESS 構造体にほぼ対応します。

DB-Library 構造体とは異なり、 Client-Library の制御構造体は、実際は隠しフィールドです。構造体名は Client-Library のパブリック・ヘッダ・ファイルに定義されますが、フィールドは定義されません。

注意　このマニュアルでは、 CS_CONTEXT 構造体を「コンテキスト構造体」、 CS_CONNECTION 構造体を「接続構造体」、 CS_COMMAND 構造体を「コマンド構造体」ともいいます。

制御構造体のプロパティ

Client-Library 制御構造体は「プロパティ」を持っています。プロパティの中には、 Client-Library がどのように動作するかを決定するプロパティ値もありますが、制御構造体に対応する単なる情報のプロパティ値もあります。

次に例を示します。

- CS_TIMEOUT は CS_CONTEXT 構造体のプロパティです。この値は、 Client-Library がサーバ応答をどのくらいの時間待ってからタイムアウト・エラーを提示するかを決定します。 DB-Library アプリケーションの場合、タイムアウト値を dbsettime で指定します。このタイムアウト値は DB-Library 隠しグローバル変数です。
- CS_NETIO は CS_CONNECTION 構造体のプロパティです。この値は、ネットワーク I/O が同期モードか、完全な非同期モードか、遅延非同期モードかを決定します。 DB-Library には、類似した概念はありません。 DB-Library アプリケーションは、同期モードか非同期モードかで異なるルーチンを呼び出します。
- CS_USERNAME は CS_CONNECTION 構造体のプロパティです。この値は、サーバにログインするためのユーザ名を指定します。 Client-Library アプリケーションは、 ct_connect で接続をオープンする前に、ユーザ名を設定します。接続がオープンしている場合、このプロパティは読み込み専用です。 DB-Library アプリケーションは、 LOGINREC 構造体の内容を変更するのに、 DBSETLUSER マクロを呼び出して、パケット・サイズを指定します。 dbopen が呼び出されると、 LOGINREC パスワードが DBPROCESS ユーザ名になります。
- CS_USERDATA は、 CS_CONNECTION 構造体のプロパティであると同時に、 CS_COMMAND 構造体のプロパティでもあります。この値は、特定の接続またはコマンド構造体に対応しているユーザ・データのアドレスです。 CS_USERDATA プロパティの用途は、 DB-Library アプリケーションの dbgetuserdata および dbsetuserdata の用途に似ています。

継承されるプロパティ値

どの CS_COMMAND 構造体にも、親 CS_CONNECTION 構造体があり、どの CS_CONNECTION 構造体にも、親 CS_CONTEXT 構造体があります。

構造体は、割り付けられるときに、適用できるすべてのプロパティ値を親から継承します。

たとえば、新しい CS_CONNECTION 構造体は親 CS_CONTEXT の CS_NETIO 値を継承します。親 CS_CONTEXT が同期ネットワーク I/O を使用するように設定されていれば、新しい CS_CONNECTION も同期モードになります。

継承されたプロパティ値は、構造体が割り付けられたあとで変更できます。

プロパティ値の設定

Client-Library 、 CS-Library 、 Server-Library のどれにも、プロパティ値を設定、取得するルーチンが含まれています。

CS_CONTEXT 構造体

CS_CONTEXT 構造体は、アプリケーション・コンテキストまたはオペレーティング環境を定義します。アプリケーションは複数の CS_CONTEXT 構造体を持つことができますが、通常は 1 アプリケーションに 1 CS_CONTEXT 構造体です。

アプリケーションは、最も高位の Client-Library 動作を定義するのに、以下のように CS_CONTEXT 構造体を使用します。

- CS_CONTEXT 構造体プロパティは、 DB-Library の隠しグローバル変数に代わるものです。たとえば、 DB-Library アプリケーションの dbsettime 呼び出しでは、グローバル・タイムアウト値が変更されます。これに対し、 Client-Library アプリケーションでは、CS_TIMEOUT プロパティを設定すると、その特定の CS_CONTEXT 構造体の子接続だけが影響を受けます。
- CS_CONTEXT 構造体にインストールされているメッセージ・ハンドラとエラー・ハンドラは、その CS_CONTEXT 内に割り付けられているすべての CS_CONNECTION によって継承されます。
- CS_CONTEXT には、ロケール名、言語、および日付順などのロケール情報を含めることができます。

CS_CONNECTION 構造体

CS_CONNECTION 構造体は、クライアント・アプリケーションからリモート・サーバへの接続を定義します。アプリケーションは、接続レベルの Client-Library 動作を定義し、接続に関する情報を保管、取得するのに、 CS_CONNECTION 構造体を使用します。

- CS_CONNECTION プロパティは、接続動作をカスタマイズします。たとえば、アプリケーションは、 CS_TDS_VERSION 接続プロパティを設定して、ある接続が特定の Tabular Data Stream™(TDS) プロトコル・バージョンを使用するように要求できます。
- CS_CONNECTION は親のコンテキストからメッセージ・ハンドラとエラー・ハンドラを継承しますが、アプリケーションは新しいハンドラをインストールすることによって、これらのデフォルト・ハンドラを無効にできます。

Client-Library の CS_CONNECTION 構造体は、 DB-Library の DBPROCESS に比べて次のような利点があります。

- メッセージ・ハンドラとエラー・ハンドラを接続単位でインストールできます。
- ログイン情報が接続にバインドされます。接続が確立したあと、ログイン・パラメータは読み込み専用プロパティになります。
- Client-Library 接続はアクティブなカーソルと別のコマンドを同時にサポートします。

CS_COMMAND 構造体

CS_COMMAND 構造体は、クライアント／サーバ接続内のコマンド領域を定義します。

アプリケーションは、コマンドをサーバに送信し、これらのコマンドの結果を処理するのに、 CS_COMMAND 構造体を使用します。

接続規則とコマンド規則

アプリケーションは、 Client-Library カーソルを使用する場合にかぎり、同一接続で複数のコマンド構造体をアクティブにすることができます。 Client-Library カーソルを使用する場合、アプリケーションは、カーソルから返されたローを処理しながら、新しいコマンドを送信することができます。

Client-Library カーソルの open コマンド以外のコマンドの結果を処理しているときは、元のコマンドの結果が完全に処理されるか、キャンセルされるまで、同一接続で追加コマンドを送信することはできません。

詳細については、『 Open Client Client-Library/C プログラマーズ・ガイド』の「第 7 章 Client-Library カーソルの使い方」を参照してください。

その他の構造体

Client-Library は前述の 3 つの基本制御構造体のほかに、次の構造体も使用します。

- CS_DATAFMT
- CS_IODESC
- CS_LOCALE
- CS_BLKDESC

CS_DATAFMT

アプリケーションは、データ値とプログラム変数を Client-Library ルーチンに記述するのに、 CS_DATAFMT 構造体を使用します。

次に例を示します。

- ct_bind には、送信先変数を記述する CS_DATAFMT 構造体が必要です。
- ct_describe は、結果データ項目を記述する CS_DATAFMT 構造体に値を設定します。
- ct_param には、入力パラメータを記述する CS_DATAFMT 構造体が必要です。
- cs_convert には、変換元と変換先データを記述している CS_DATAFMT 構造体が必要です。

ct_bind または ct_describe での CS_DATAFMT の使い方については、『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。 cs_convert での CS_DATAFMT の使い方については、『Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。

CS_IODESC

一般に、アプリケーションは、 text または image データを操作するときに、 CS_IODESC 構造体を使用します。 CS_IODESC 構造体は、結果セットの現在ロー内のカラムの I/O 記述子を定義します。この構造体には、そのカラムのテキスト・タイムスタンプなど、カラム・データに関する情報が含まれています。

「Client-Library の CS_IODESC 構造体」 (93 ページ) を参照してください。

CS_LOCALE

アプリケーションは、コンテキスト、接続、またはデータ要素レベルでカスタム・ローカライゼーション情報を指定するのに、CS_LOCALE 構造体を使用します。

「CS_LOCALE 構造体」 (99 ページ) を参照してください。

CS_BLKDESC

アプリケーションは、バルク・コピー・オペレーションを行うときに、CS_BLKDESC 構造体を使用します。

「Bulk-Library の初期化と終了処理」 (90 ページ) を参照してください。

第 5 章

DB-Library アプリケーション・コードの変換

この章では、DB-Library プログラムを Client-Library プログラムに適切に変換するために必要な情報を提供します。

この章の内容は、次のとおりです。

変換手順

DB-Library プログラムを Client-Library プログラムに変換するには、通常、次のような作業が必要です。

1 DB-Library ヘッダ・ファイル名を Client-Library ヘッダ・ファイル名に置き換えます (「ヘッダ・ファイル」(24 ページ) を参照してください)。

2 コード変換の計画を立てます。クライアント・アプリケーション・コードは、この章で説明する次の 5 つのカテゴリに分割できます。

- 初期化と終了処理コード
- 接続をオープンするコード
- エラー・ハンドラとメッセージ・ハンドラ
- コマンドを送信するコード
- 結果を処理するコード

各項では、DB-Library と Client-Library のプログラム論理を対比させて示します。変換を始める前に、この章を読んで Client-Library の基本機能をしっかり理解してください。拡張機能については、「第 6 章 拡張機能」で説明します。

3 変換を行います。

4 必要に応じて、DB-Library の宣言を置換または削除します。

5 DB-Library 関数呼び出しを Client-Library または CS-Library の対応する機能に置き換え、必要に応じてプログラム論理を変更します。表 A-1 (101 ページ) に、DB-Library ルーチンと Client-Library ルーチンを対比してリストしてあるので、参考にしてください。

注意 この章のコーディング例では、移行サンプル *dbtoctex.h* に指定されているように、EXIT_ON_FAIL サンプル・マクロを使用しています。このマクロについては、「プログラム例でのリターン・コード・エラー・チェック」(x ページ) を参照してください。

初期化と終了処理コード

初期化では、DB-Library または Client-Library プログラムのプログラミング環境を設定します。終了処理では、接続をクローズして、ライブラリ・データ構造体の割り付けを解除します。

呼び出しシーケンスの比較

表 5-1 では、初期化と終了処理に使用する DB-Library 呼び出しと、同等の Client-Library 機能を比較します。Client-Library については、10.x 以降でデフォルト・バージョン・レベルがすべての機能をサポートします。

各ルーチンの詳細については、そのルーチンのリファレンス・ページを参照してください。

表 5-1：DB-Library と Client-Library の比較 (初期化と終了処理)

DB-Library ルーチン	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
(なし)		cs_ctx_alloc(version, context)	CS_CONTEXT 構造体を割り付け、希望する CS-Library 動作のバージョン・レベルを指定する。*version* は、CS_VERSION_120、CS_VERSION_125、CS_VERSION_150、CS_VERSION_155、または CS_VERSION_157 のいずれか。
(なし)		cs_config(context, CS_SET, CS_MESSAGE_CB, handler, CS_UNUSED, NULL)	CS-Library エラー・ハンドラ・コールバック関数をインストールする。
dbinit()	DB-Library を初期化する。	ct_init(context, version)	Client-Library を初期化し、希望する動作のバージョン・レベルを指定する。
dbsetversion (dbproc, version)	(DB-Library 10.x 以降のアプリケーションのみ) 希望する動作のバージョン・レベルを指定する。*version* は、DBVERSION_46 または DBVERSION_100 のどちらか。Sybase では、更新されたバージョンに導入された機能やコード変更を使用できるように、DBVERSION_100 を推奨。	(なし)	
dberrhandle (handler)	DB-Library エラー・コールバック関数をインストールする。	ct_callback(context, NULL,CS_SET, CS_CLIENTMSG_CB, handler)	Client-Library エラー・コールバック関数をインストールする。 「エラー・ハンドラとメッセージ・ハンドラ」(48 ページ) を参照してください。

DB-Library ルーチン	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
dbmsghandle(handler)	DB-Library サーバ・メッセージ・コールバック関数をインストールする。	ct_callback(context, NULL, CS_SET, CS_SERVERMSG_CB, handler)	Client-Library サーバ・メッセージ・コールバック関数をインストールする。 「エラー・ハンドラとメッセージ・ハンドラ」(48 ページ) を参照してください。
(*表 5-2 : DB-Library と Client-Library の比較 (接続のオープン)* を参照)	接続をオープンする。	表 5-2 を参照してください。	接続をオープンする。接続をオープンする前に、必要なプロパティをコンテキスト／接続に設定すること。
dbexit()	すべての DBPROCESS 構造体をクローズして割り付けを解除し、dbinit で初期化した構造体の終了処理を行う。	ct_exit(context, option)	Client-Library を終了する。Client-Library を終了する前に、すべてのオープンされているコマンドとコンテキスト構造体の割り付けを解除する。 *option* は、通常 CS_UNUSED。エラーが原因で終了する場合には、CS_FORCE_EXIT が有用。
(なし)		cs_ctx_drop(context)	CS_CONTEXT 構造体の割り付けを解除する。

Client-Library アプリケーションは、CS_CONTEXT 構造体の割り付けおよびその解除を行う必要があります。CS_CONTEXT は、エラー・メッセージの言語と文字セットおよびアプリケーションのデフォルト・エラーとメッセージ・コールバックなど、基本的なアプリケーション・プロパティの「ハンドラ」として機能します。
「CS_CONTEXT 構造体」(27 ページ) を参照してください。

例：Client-Library の初期化と終了処理

次のコーディング例は、*ctfirst.c* 移行サンプル・プログラムの一部であり、Client-Library の初期化と終了処理を示しています。

この例では、CS-Library および Client-Library のエラー・ハンドラと、Client-Library サーバ・メッセージ・コールバックをインストールしています。Client-Library のエラー・ハンドラとサーバ・メッセージ・ハンドラの例については、『Open Client Client-Library/C リファレンス・マニュアル』の「コールバック」を参照してください。CS-Library のエラー・ハンドラの例については、『Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。

```
CS_CONTEXT    *context = (CS_CONTEXT *) NULL;
CS_CONNECTION *conn;
CS_RETCODE    ret;

/*
** Setup screen output.
*/

EX_SCREEN_INIT();

/*
** Step 1.
** Allocate a CS_CONTEXT structure and initialize Client-Libary.The
** EXIT_ON_FAIL() macro used for return code error checking is defined in
** dbtoctex.h. If the return code passed to EXIT_ON_FAIL() is not CS_SUCCEED,
** it:
- Cleans up the context structure if the pointer is not NULL.
- Exits to the operating system.
**
-- if (dbinit() == FAIL
-- exit(ERREXIT);
*/

ret = cs_ctx_alloc(CS_CURRENT_VERSION, &context);
EXIT_ON_FAIL(context, ret, "Could not allocate context.");

ret = ct_init(context, CS_CURRENT_VERSION);
EXIT_ON_FAIL(context, ret, "Client-Library initialization failed.");

/*
** Step 2.
** Install callback handlers for CS-Library errors, Client-Library errors, and
** Server-Library errors.The handlers are defined at the bottom of
** this source file.
**
-- dberrhandle(err_handler);
```

```
-- dbmsghandle(msg_handler);
*/

/*
** cs_config() installs a handler for CS-Library errors.
*/

ret = cs_config(context, CS_SET, CS_MESSAGE_CB, (CS_VOID *) cserror_cb,
      CS_UNUSED, NULL);
EXIT_ON_FAIL(context, ret, "Could not install CS-Library error handler.");

/*
** ct_callback() installs handlers for Client-Library errors and server
messages.
**
** ct_callback() lets you install handlers in the context or the connection.
** Here, we install them in the context so that they are inherited by the
** connections that are allocated using this context.
*/

ret = ct_callback(context, NULL, CS_SET, CS_CLIENTMSG_CB, (CS_VOID
      clientmsg_cb);
EXIT_ON_FAIL(context,ret,"Could not install Client-Library error handler.");
ret = ct_callback(context, NULL, CS_SET, CS_SERVERMSG_CB, (CS_VOID *)
      servermsg_cb);
EXIT_ON_FAIL(context,ret,"Could not install server message handler.");
... deleted code that connects and interacts with the server ...
/*
** Clean up Client-Library.
** ct_exit(context, CS_UNUSED) requests an "orderly" exit -- this
** call fails if we have open connections.If it fails, EXIT_ON_FAIL() calls
** ct_exit(context, CS_FORCE_EXIT) to force cleanup of Client-Library.
*/
ret = ct_exit(context, CS_UNUSED);
EXIT_ON_FAIL(context, ret, "ct_exit(CS_UNUSED) failed.");

/*
** Clean up CS-Library. cs_ctx_drop() always fails if ct_init()
** succeeded on the context but ct_exit() did not (or if ct_exit()
** was not called at all).
*/
(CS_VOID) cs_ctx_drop(context);
context = (CS_CONTEXT *) NULL;
exit(NORMAL_EXIT);
/*
** clientmsg_cb() -- Callback handler for Client-Library messages.
** Client-Library messages inform the application of errors or
** significant conditions.
** Parameters:
```

```
** context -- Pointer to the context structure where the error occurred.
** The handler can retrieve context properties and set the CS_USERDATA
** property.
** connection -- Pointer to the connection on which the error occurred.
** This parameter can be NULL if no connection was involved in the
** error.If connection is non-NULL, the handler can retrieve connection
** properties, set the CS_USERDATA property, and call
** ct_cancel(CS_CANCEL_ATTN) on the connection.
** errmsg -- Pointer to a CS_CLIENTMSG structure that describes the
** error.See the "CS_CLIENTMSG" topics page in the Client-Library
** reference manual for a description of the fields.
 ** Returns:CS_SUCCEED
** Side Effects:None.
*/
CS_RETCODE CS_PUBLIC
clientmsg_cb(context, connection, errmsg)
CS_CONTEXT     *context;
CS_CONNECTION  *connection;
CS_CLIENTMSG   *errmsg;

CS_RETCODE    ret;
CS_INT     timeout_val;
/*
** Composition of error messages.
** ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
** Client-Library message numbers encode values for severity,
** layer, origin, and number.The layer, origin, and number
** correspond to national language strings from the ctlib.loc
** locales file.Client-Library composes the text of the message
** (received in errmsg->msgstring) as follows:
** <routine name>:<layer string>:<origin string>:<description>
** where:
** <routine name> is the name of the Client-Library routine
** that was active when the exception occurred.
** <layer string> describes the layer where the exception occurred
** or was found.
** <origin string> indicates whether the error is internal or external
** to Client-Library.
** <description> is the error description.
*/
fprintf(ERR_CH, "Client-Library Message:");
fprintf(ERR_CH, "LAYER = (%ld) ORIGIN = (%ld) ",
(long)CS_LAYER(errmsg->msgnumber), (long)CS_ORIGIN(errmsg->msgnumber));

fprintf(ERR_CH, "SEVERITY = (%ld) NUMBER = (%ld)¥n",
long)CS_SEVERITY(errmsg->msgnumber), (long)CS_NUMBER(errmsg->msgnumber));
```

```
fprintf(ERR_CH, "Message String:%s¥n", errmsg->msgstring);
/*
** Operating system errors.
** ~~~~~~~~~~~~~~~~~~~~~~~
** Some exceptions reported by Client-Library are caused by exceptions
** in the underlying system software.When this occurs, Client-Library
** forwards the system error information to the application.
*/
if (errmsg->osstringlen > 0)
{
fprintf(ERR_CH, "Operating System Error:%s¥n",
errmsg->osstring);
}
/*
** Handler return values and their meaning.
** ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
** Client-Library error handlers must return CS_SUCCEED or CS_FAIL.


** Returning any other value "kills" the connection -- Client-


** Library responds by marking the connection "dead", which makes


** it unuseable.You can test for dead connections by retrieving


** the value of the CS_CON_STATUS connection property, which is


** a bit-masked value.The CS_CONSTAT_DEAD bit is set if the connection
** is dead.This functionality replaces DB-Library's DBDEAD() macro.
** Unlike the DB-Library error handler, there is no return code that

** causes Client-Library to exit to the operating system.The application
** must check return codes in the main-line code and abort from the
** main-line code.
*/
/*
** (Optional) Test for specific error conditions.
** ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
** The ERROR_SNOL() macro is defined at the top of this file.
```

```
** The component byte values of a message number (origin, layer, and
** number) are defined in the Client-Library locales file.
*/
/*
** Test for timeout errors.Timeout errors will be received when you:
** -- are using a synchronous mode connection,
** -- have set the CS_TIMEOUT context property to a non-zero positive value
** (representing a number of seconds).
** -- the server takes longer than the given time to respond to a command.
** For timeout errors, the command can be canceled with
** ct_cancel(CS_CANCEL_ATTN).Other ct_cancel() options are not
** to be used in an error handler.If we return CS_SUCCEED
** without canceling, then Client-Library will wait for another
** timeout period, then call this error handler again.If the
** we return CS_FAIL, then Client-Library kills the
** connection, making it unuseable.
*/
if (ERROR_SNOL(errmsg->msgnumber, CS_SV_RETRY_FAIL, 63, 2, 1))
{
/*
** Get the timeout period.This is not really neccessary, but
** demonstrated to show the correlation between timeout errors
** and the CS_TIMEOUT context property.
*/
ret = ct_config(context, CS_GET, CS_TIMEOUT, CS_VOID *)&timeout_val, CS_UNUSED,
(CS_INT *)NULL);
if (ret != CS_SUCCEED)
{
timeout_val = 0;
}
fprintf(ERR_CH, "¥nServer has not responded in at least %ld seconds.
Canceling.¥n",(long)timeout_val);
(CS_VOID)ct_cancel(connection, (CS_COMMAND *)NULL, CS_CANCEL_ATTN);
}
return CS_SUCCEED;
} /* clientmsg_cb() */
/*
** cserror_cb() -- Callback handler for CS-Library errors.
** Parameters:
** context -- Pointer to the context structure passed to the CS-Library
** call where the error occurred.The handler can retrieve any
** context property, and set the CS_USERDATA property.
** errmsg -- Pointer to a CS_CLIENTMSG structure that describes the
** error.See the "CS_CLIENTMSG" topics page in the Client-Library
** reference manual for a description of the fields.
 ** Returns:CS_SUCCEED
```

```
** Side Effects:None
*/
CS_RETCODE CS_PUBLIC
cserror_cb(context, errmsg)
CS_CONTEXT     *context;
CS_CLIENTMSG   *errmsg;
{
/*
** Composition of error messages.
** ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
** CS-Library message numbers are decoded the same way as Client-
** Library messages.See the comments in clientmsg_cb() for a
** description.
*/
fprintf(ERR_CH, "CS-Library error:");
fprintf(ERR_CH, "LAYER = (%ld) ORIGIN = (%ld) ",
(long)CS_LAYER(errmsg->msgnumber), (long)CS_ORIGIN(errmsg->msgnumber));
fprintf(ERR_CH, "SEVERITY = (%ld) NUMBER = (%ld)¥n",
(long)CS_SEVERITY(errmsg->msgnumber), (long)CS_NUMBER(errmsg->msgnumber));
fprintf(ERR_CH, "Message String:%s¥n", errmsg->msgstring);
/*
** Operating System Errors.
** ~~~~~~~~~~~~~~~~~~~~~~~~
** If an operating system error occurred and CS-Library was notified,
** then CS-Library forwards the error information to the application.
*/
if (errmsg->osstringlen > 0)
{
 fprintf(ERR_CH, "Operating System Error:%s¥n", errmsg->osstring);
}
/*
** Handler Return Values.
** ~~~~~~~~~~~~~~~~~~~~~
** CS-Library error handlers should return CS_SUCCEED.
*/
return CS_SUCCEED;
} /* cserror_cb */
/*
** servermsg_cb() -- Callback handler for server messages.The
** server sends messages to describe errors or significant
** events.Client-Library calls this function to forward
** server messages to the client program.
** Parameters:
** context -- Pointer to the context structure that is the parent of
** the connection.The handler can retrieve context properties
** and set the CS_USERDATA property.
```

```
** connection -- Pointer to the connection on which the message was
** received.The handler can retrieve any connection property, set
** the CS_USERDATA property, and call ct_cancel(CS_CANCEL_ATTN)
** on the connection.In addition, when the server sends
** extended error data with a message, the handler can retrieve
** the data.This handler ignores extended error data.
** srvmsg -- Pointer to a CS_SERVERMSG structure that contains the
** message info. See the "CS_SERVERMSG" topics page in the Client-
** Library reference manual for a description of the fields.All the
** information that the DB-Library message handler received as
** parameters is available in the CS_SERVERMSG structure.
 ** Returns:CS_SUCCEED
** Side Effects:None
*/
CS_RETCODE CS_PUBLIC
servermsg_cb(context, connection, srvmsg);
CS_CONTEXT     *context;
CS_CONNECTION  *connection;
CS_SERVERMSG   *srvmsg;
{
/*
** CS_SERVERMSG Fields.
** ~~~~~~~~~~~~~~~~~~~
** When connected to an Adaptive Server Enterprise, most of the CS_SERVERMSG
fields
** have corresponding columns in the sysmessages system table. 条件
** connected to an Open Server, it's up to the Open Server programmer
** to set the fields for the messages sent by the Open Server.
*/
fprintf(ERR_CH, "Server message:");
/*
** For Adaptive Server Enterprise connections, srvmsg->number and srvmsg-
>severity come
** from the sysmessages system table, columns 'error' and 'severity',
** respectively.
*/
fprintf(ERR_CH, "Number %ld, Severity %ld, ",
long)srvmsg->msgnumber, (long)srvmsg->severity);
/*
** For Adaptive Server Enterprise connections, srvmsg->line is the line number
** in a language batch, or, if srvmsg->proclen field is > 0, the
** line number within the stored procedure named in srvmsg->proc.
** srvmsg->state is the Adaptive Server Enterprise error state, which provides
** information to Sybase Technical Support about serious Adaptive
** Server errors.
*/
```

```
fprintf(ERR_CH, "State %ld, Line %ld¥n",
(long)srvmsg->state, (long)srvmsg->line);
/*
** For Adaptive Server Enterprise connections, srvmsg->srvname is the value of
** the @@servername global variable.See the Adaptive Server Enterprise
documentation
** for information on how to set or change @@servername.
*/
if (srvmsg->svrnlen > 0)
{
fprintf(ERR_CH, "Server '%s'¥n", srvmsg->svrname);
}
/*
** For Adaptive Server Enterprise connections, srvmsg->proclen is > 0 if the
message
** was raised while executing a stored procedure. srvmsg->proc is the
** procedure name in this case, and srvmsg->line is the line in the
** procedure's code where the error or condition was raised.
*/
if (srvmsg->proclen > 0)
{
 fprintf(ERR_CH, " Procedure '%s'¥n", srvmsg->proc);
}
/*
** Finally, for Adaptive Server Enterprise connections, srvmsg->text is the text
of the
** message from the 'description' column in sysmessages.
*/
 fprintf(ERR_CH, "Message String:%s¥n", srvmsg->text);
/*
** The Client-Library message handler must return CS_SUCCEED.
** Returning any other value "kills" the connection -- Client-
** Library responds by marking the connection "dead", which makes
** it unuseable.
*/
return CS_SUCCEED;
} /* servermsg_cb() */
```

接続をオープンするコード

DB-Library アプリケーションは、LOGINREC 構造体と DBPROCESS 構造体を使用してサーバへの接続をオープンします。Client-Library は、CS_CONNECTION 隠し構造体を使用します。
「CS_CONNECTION 構造体」(28 ページ) を参照してください。

呼び出しシーケンスの比較

表 5-2 では、接続のオープンに使用する DB-Library ルーチンを、Client-Library の同等の機能と比較します。

表 5-2：DB-Library と Client-Library の比較 (接続のオープン)

DB-Library ルーチン	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
dblogin()	dbopen で使用する LOGINREC を割り付ける。	ct_con_alloc(context, connection)	CS_CONNECTION 構造体を割り付ける。
DBSETLUSER(loginrec, username)	LOGINREC 構造体内のユーザ名を設定する。	ct_con_props(connection, CS_SET, CS_USERNAME, username, buflen, NULL)	接続構造体内のユーザ名プロパティを設定する。
DBSETLPWD(loginrec, password)	LOGINREC 構造体内のユーザのサーバ・パスワードを設定する。	ct_con_props(connection, CS_SET, CS_PASSWORD, password, buflen, NULL)	接続構造体内のユーザのサーバ・パスワードのプロパティを設定する。
DBSETLAPP(loginrec, application)	LOGINREC 構造体内のアプリケーション名を設定する。	ct_con_props(connection, CS_SET, CS_APPNAME, appname, buflen, NULL)	接続構造体内のアプリケーション名プロパティを設定する。
dbopen(loginrec, server)	サーバに接続する (DBPROCESS を割り付ける)。	ct_connect(connection, server_name, snamelen)	サーバに接続する (事前に割り付けられた接続構造体を使用する)。
dbloginfree(loginrec)	LOGINREC 構造体を解放する。	なし	
言語コマンド、RPC コマンド、TDS passthrough 呼び出し	DBPROCESS コマンド構造体を使用して、要求を送信して結果を処理する。	CS_COMMAND (「コマンドを送信するコード」(53 ページ) を参照)	コマンド構造体を使用して、要求を送信して結果を処理する。

DB-Library ルーチン	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
dbclose(dbproc)	DBPROCESS 構造体をクローズして、割り付けを解除する。	ct_close(connection, option)	サーバ接続をクローズする。 *option* は、通常 CS_UNUSED。CS_FORCE_CLOSE は、エラーが原因で接続をクローズする場合に有用。
(なし)		ct_con_drop(connection)	接続構造体の割り付けを解除する。

Client-Library の強化機能

Client-Library アプリケーションは、Windows NT Lan Manager (SSPI)、Kerberos などのネットワーク・ベース・セキュリティ・メカニズムによって提供されるネットワーク・ベースのユーザ認証機能を使用して接続を確立することもできます。この場合、Client-Library アプリケーションは、ct_con_props を呼び出してユーザ名とパスワードを設定する代わりに、次のようなタスクを実行します。

- (オプション) CS_SEC_MECHANISM 接続プロパティを設定して、接続のセキュリティ・メカニズムを指定します。ほとんどのアプリケーションは、Sybase セキュリティ・ドライバ設定に定義されているデフォルトを使用します。
- 接続の CS_USERNAME プロパティを設定して、ユーザのネットワーク名と対応させます。
- CS_SEC_NETWORKAUTH 接続プロパティを設定して、ネットワーク・ベースの認証ができるようにします。

ネットワーク・ベースの認証には、ネットワーク・セキュリティ・メカニズム用の Sybase セキュリティ・ドライバが必要です。サーバの中には、ネットワーク・ベースの認証をサポートしないものもあります。詳細については、『Open Client Client-Library/C リファレンス・マニュアル』の「セキュリティ機能」を参照してください。

LOGINREC コードの移行

DB-Library のアプリケーションでは、LOGINREC 構造体を使用して、オープン前に接続をカスタマイズします。Client-Library のアプリケーションでは、CS_CONNECTION プロパティを使用して、オープン前に接続をカスタマイズします。

複数の接続をオープンするのに同じ LOGINREC 構造体を使用する DB-Library コードを置き換えるには、ct_getloginfo および ct_setloginfo を次のように使用します。

1　ct_con_alloc で接続構造体を割り付けます。

2　ct_con_props を呼び出して、接続をカスタマイズします。

3　ct_connect で接続をオープンします。

4　各接続を同じログイン・プロパティでオープンするには、次のようにします。

- ct_getloginfo を呼び出して、CS_LOGINFO 構造体を割り付け、元の接続のログイン・プロパティをその構造体にコピーします。
- ct_con_alloc で新しい接続構造体を割り付けます。
- ct_setloginfo を呼び出して、CS_LOGINFO 構造体から新しい接続構造体にログイン・プロパティをコピーします。プロパティをコピーした後に、ct_setloginfo は CS_LOGINFO 構造体の割り付けを解除します。
- ct_con_props を呼び出して、新しい接続の非ログイン・プロパティをカスタマイズします。
- ct_connect で新しい接続をオープンします。

例：Client-Library 接続のオープン

次のコーディング例は、*ctfirst.c* 移行サンプル・プログラムの一部であり、Client-Library 接続のオープン例を示しています。

```
... deleted initialization code ...
/*
** Step 1.
** Allocate a CS_CONTEXT structure and initialize Client-Libary.The
** EXIT_ON_FAIL() macro used for return code error checking is defined in
** dbtoctex.h. If the return code passed to EXIT_ON_FAIL() is not CS_SUCCEED,
** it:
- Cleans up the context structure if the pointer is not NULL.
```

```
- Exits to the operating system.
**
-- if (dbinit() == FAIL
-- exit(ERREXIT);
*/

ret = cs_ctx_alloc(CS_CURRENT_VERSION, &context);
EXIT_ON_FAIL(context, ret, "Could not allocate context.");
ret = ct_init(context, CS_CURRENT_VERSION);
EXIT_ON_FAIL(context, ret, "Client-Library initialization failed.");
/*
... deleted code that defines callback handlers ...

/*
** Step 3.
** Connect to the server named by the DSQUERY environment
** variable using the credentials defined in dbtoctex.h
**
**  3a.Allocate a CS_CONNECTION structure.
**  3b.Insert the username, password, and other login parameters
** into the connection structure.
**  3c.Call ct_connect(), passing the CS_CONNECTION as an argument.
*/
/*
** Step 3a.
** Allocate a CS_CONNECTION structure.The CS_CONNECTION replaces
** DB-Library's LOGINREC and DBPROCESS structures.The LOGINREC
** fields are connection properties in Client-Library.
**
-- login = dblogin();
-- if (login == (LOGINREC *) NULL)
-- {
--   fprintf(ERR_CH, "dblogin() failed.Exiting.¥n");
--   dbexit();
-- exit(ERREXIT);
-- }
*/
ret = ct_con_alloc(context, &conn);
EXIT_ON_FAIL(context, ret, "Allocate connection structure failed.");
/*
** Step 3b.
** Put the username, password, and other login information into the
** connection structure.We do this with ct_con_props() calls.
** After the connection is open, Client-Library makes these properties
** read-only.
**
** USER and PASSWORD are defined in dbtoctex.h
```

```
**
-- DBSETLUSER(login, USER);
-- DBSETLPWD(login, PASSWORD);
-- DBSETLAPP(login, "dbfirst");
*/
ret = ct_con_props(conn, CS_SET, CS_USERNAME, USER, STRLEN(USER), NULL);
EXIT_ON_FAIL(context, ret, "Set connection username failed.");
ret = ct_con_props(conn, CS_SET, CS_PASSWORD, PASSWORD, STRLEN(PASSWORD),NULL);
EXIT_ON_FAIL(context, ret, "Set connection password failed.");
ret = ct_con_props(conn, CS_SET, CS_APPNAME, "ctfirst", STRLEN("ctfirst"),
      NULL);
EXIT_ON_FAIL(context, ret, "Set connection application name failed.");
/*
** Step 3c.
** Call ct_connect() to open the connection.Unlike dbopen(), ct_connect()
** uses a connection structure which is already allocated.
**
-- dbproc = dbopen(login, NULL);
-- if (dbproc == (DBPROCESS *) NULL)
-- {
--   fprintf(ERR_CH, "Connect attempt failed.Exiting.¥n");
--   dbexit();
--   exit(ERREXIT);
-- }
*/
ret = ct_connect(conn, NULL, STRLEN(NULL));
EXIT_ON_FAIL(context, ret, "Connection attempt failed.");
... deleted command code ...
/*
** Step 5.
** Close our connection.CS_UNUSED as the second ct_close() parameter
** requests an "orderly" close.This means that we expect the connection to
** be idle.If we had issued a command to the server, but had not
** read all the results sent by the server, then the connection would
** not be idle and this call would fail.
**
** If ct_close() were to fail here, then the code in EXIT_ON_FAIL() would
** ct_exit(CS_FORCE_EXIT) to force all connections closed before exiting.
**
-- dbclose(dbproc);
*/
ret = ct_close(conn, CS_UNUSED);
EXIT_ON_FAIL(context, ret, "Orderly connection-close failed.");
ret = ct_con_drop(conn);
EXIT_ON_FAIL(context, ret, "ct_con_drop() failed.");
/*
```

```
** Clean up Client-Library.
** ct_exit(context, CS_UNUSED) requests an "orderly" exit -- this
** call fails if we have open connections.If it fails, EXIT_ON_FAIL()
** calls ct_exit(context, CS_FORCE_EXIT) to force cleanup of Client-Library.
*/
ret = ct_exit(context, CS_UNUSED);
EXIT_ON_FAIL(context, ret, "ct_exit(CS_UNUSED) failed.");
/*
** Clean up CS-Library. cs_ctx_drop() always fails if ct_init()
** succeeded on the context but ct_exit() did not (or if ct_exit()
** was not called at all).
*/
(CS_VOID) cs_ctx_drop(context);
context = (CS_CONTEXT *) NULL;
exit(NORMAL_EXIT);
}
... deleted error callback routine code ...
```

エラー・ハンドラとメッセージ・ハンドラ

ほとんどのアプリケーションは、エラー・メッセージを処理するのに、コールバック・ルーチンを使用します。

Client-Library には、コールバック・メッセージ処理機能のほかにインライン・メッセージ処理機能が用意されています。インライン・メッセージ処理の場合、アプリケーションがメッセージを処理する時期を制御できます。ct_diag ルーチンが接続レベルでインライン・メッセージ処理機能を初期化します。

Client-Library と CS-Library は、次の構造体を使用して、エラー情報とメッセージ情報をメッセージ・コールバック・ルーチンに返します。

- CS_CLIENTMSG 構造体は、Client-Library と CS-Library のエラーを記述します。この構造体は、アプリケーションの Client-Library または CS-Library エラー・ハンドラに渡されます。この構造体のほとんどのフィールドは、DB-Library エラー・ハンドラ・パラメータに直接マップします。
- CS_SERVERMSG 構造体は、サーバ・メッセージを記述します。この構造体はアプリケーションのサーバ・メッセージ・ハンドラに渡されます。この構造体のほとんどのフィールドは、DB-Library メッセージ・ハンドラ・パラメータに直接マップします。

シーケンス・メッセージ

Client-Library は、コールバック・メッセージ・ハンドラ・ルーチンの連続呼び出しを使用して長いメッセージを処理します。メッセージ情報構造体のステータス・ビットマスクに、メッセージ・テキストが全文であるか、またはシーケンス・メッセージの最初、中間、最後のいずれかの部分であるかが示されます。ほとんどのサーバ・メッセージは 1 回のメッセージ・コールバック呼び出しで処理できる長さですが、Transact-SQL の raiserror または print コマンドから発信されるユーザ定義のメッセージは、例外です。これらのメッセージは、CS_SERVERMSG の 1024 バイトのテキスト・フィールドよりも長い場合があります。

DB-Library では、固定長のバッファにメッセージを置く Client-Library とは異なり、メッセージのポインタを提供します。

サーバ・メッセージ・ハンドラの置換

各 DB-Library サーバ・メッセージ・ハンドラ・パラメータは、CS_SERVERMSG 構造体の 1 フィールドにマップします。このほかに、CS_SERVERMSG には、DB-Library メッセージ・ハンドラ・パラメータにマップしないフィールドが 4 つ含まれています。これらのパラメータは、メッセージ・テキスト、サーバ名、プロシージャ名の長さ (バイト単位) とシーケンス・メッセージおよび拡張エラー・メッセージ情報に使用されるビットマスク・インジケータです。

表 5-3 : DB-Library メッセージ・ハンドラ・パラメータと CS_SERVERMSG フィールドの対応

DB-Library メッセージ・ハンドラ・パラメータ	パラメータまたはフィールドの説明	Client-Library CS_SERVERMSG 構造体フィールド
severity	エラー・メッセージの重大度	severity
msgno	エラー・メッセージの識別番号	msgnumber
msgstate	サーバ・メッセージに対応するサーバ・エラー・ステータス	state
msgtxt	サーバ・メッセージのテキスト	text
(なし)	*text* の長さ (バイト単位)	textlen
srvname	メッセージを生成したサーバの名前	svrname
(なし)	*svrname* の長さ (バイト単位)	svrnlen
procname	メッセージがある場合、その原因となったストアド・プロシージャの名前	proc
(なし)	*proc* の長さ (バイト単位)	proclen

DB-Library メッセージ・ハンドラ・パラメータ	パラメータまたはフィールドの説明	Client-Library CS_SERVERMSG 構造体フィールド
line	メッセージを生成したコマンド・バッチまたはストアド・プロシージャ行がある場合、その番号	line
(なし)	msgstring がメッセージ全体であるか、シーケンス・メッセージのどの部分であるかを示すビットマスク・インジケータ	status
(なし)	エラーに対応する SQL ステータス値がある場合、それが入っているバイト文字列	sqlstate

接続が成功すると、DB-Library アプリケーションのサーバ・メッセージ・ハンドラは 0 を返し、Client-Library アプリケーションのサーバ・メッセージ・ハンドラは CS_SUCCEED を返します。Client-Library サーバ・メッセージ・ハンドラが CS_SUCCEED 以外の値を返した場合、Client-Library は接続を “dead”とマーク付けし、その接続は使用不能になります。CS_SUCCEED 以外のコードが返されると、サーバおよびクライアントのメッセージ・コールバックからの接続は dead とマーク付けされます。

サーバ・メッセージ・コールバック例については、『Open Client Client-Library/C リファレンス・マニュアル』の「コールバック」を参照してください。

DB-Library エラー・ハンドラの置換

DB-Library エラー・ハンドラは、CS-Library エラー・ハンドラおよび Client-Library クライアント・メッセージ・ハンドラで置き換える必要があります。DB-Library エラー・ハンドラは dberrhandle でインストールします。CS-Library エラー・ハンドラは ct_config で、Client-Library クライアント・メッセージ・ハンドラは ct_callback でインストールします。CS-Library 呼び出しでエラーが発生すると、CS-Library ハンドラ が呼び出され、Client-Library 呼び出しでエラーが発生すると、Client-Library ハンドラが呼び出されます。

CS-Library ハンドラも Client-Library ハンドラも CS_CLIENTMSG 構造体を使用します。各 DB-Library エラー・ハンドラ・パラメータは、CS_CLIENTMSG 構造体のフィールドにマップします。

このほかに、CS_CLIENTMSG には、DB-Library エラー・ハンドラ・パラメータにマップしない 3 つのフィールドが含まれています。たとえば、CS_CLIENTMSG には、メッセージ・テキストとオペレーティング・システム・メッセージ・テキストの長さをバイト単位で指定する整数フィールドが用意されています。これらのフィールドでは、null ターミネータをサポートしない文字セットを使用できます。

表 5-4 は、DB-Library エラー・ハンドラ・パラメータと CS_CLIENTMSG フィールドの対応を示します。

表 5-4：DB-Library エラー・ハンドラ・パラメータと CS_CLIENTMSG フィールドの対応

DB-Library エラー・ハンドラ・パラメータ	パラメータまたはフィールドの説明	Client-Library CS_CLIENTMSG 構造体フィールド
severity	エラーの重大度	severity
dberr	エラーの識別番号	msgnumber
dberrstr	出力可能なメッセージ記述文字列	msgstring
(なし)	msgstring の長さ (バイト単位)	msgstringlen
oserr	オペレーティング・システム固有のエラー番号	osnumber
oserrstr	出力可能なオペレーティング・システム・メッセージ記述文字列	osstring
(なし)	osstring の長さ (バイト単位)	osstringlen
(なし)	msgstring がメッセージ全体であるか、シーケンス・メッセージのどの部分であるかを示すビットマスク・インジケータ	status
(なし)	エラーに対応する SQL ステータス値がある場合、それが入っているバイト文字列	sqlstate

エラー・ハンドラ戻り値

Client-Library と DB-Library では、異なるエラー・ハンドラ戻り値が必要です。

- DB-Library エラー・ハンドラは、次の値を返します。
  - INT_EXIT － DB-Library は、エラー・メッセージを出力してプログラムをアボートし、エラー表示をオペレーティング・システムに返します。

    - INT_CANCEL － DB-Library は、エラーの原因となった DB-Library ルーチンから FAIL を返します。
    - INT_TIMEOUT － タイムアウト・エラーの場合、DB-Library は、タイムアウトになったサーバ・コマンド・バッチをキャンセルします。それ以外のエラーでは、INT_TIMEOUT は INT_EXIT として処理されます。
    - INT_CONTINUE － タイムアウト・エラーの場合、DB-Library は 1 タイムアウト期間だけ待機し、再度エラー・ハンドラを呼び出します。それ以外のエラーでは、INT_CONTINUE は INT_EXIT として処理されます。
- Client-Library メッセージ・ハンドラは、次の値を返します。
    - CS_SUCCEED － Client-Library は、この接続での現在の処理を続行します。タイムアウト・エラーの場合、1 タイムアウト期間だけ待機して再度エラー・ハンドラを呼び出します。CS_SUCCEED が返された場合、アプリケーションはエラー後も処理を続行できます。DB-Library には、これに相当するリターン・コードはありません。
    - CS_FAIL － Client-Library はこの接続の現在の処理を中断し、接続を dead とマーク付けします。接続を再使用するには、アプリケーションをいったんクローズして再オープンする必要があります。

エラー・ハンドラの戻り値が直接の原因で Client-Library がプログラムをアボートすることはありません。

INT_CONTINUE の動作は CS_SUCCEED に組み込まれています。

Client-Library アプリケーションが INT_TIMEOUT と同じ動作を行うには、コールバック・ルーチンから ct_cancel(CS_CANCEL_ATTN) を呼び出す必要があります。

DB-Library エラーのエラーおよび重大度コードは、Client-Library と CS-Library のエラーおよび重大度コードに直接マップしません。

詳細については、以下を参照してください。

- CS-Library エラー・ハンドラのコーディング方法については、『Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。

- Client-Library メッセージ・ハンドラのコーディング方法については、『Open Client Client-Library/C リファレンス・マニュアル』の「コールバック」を参照してください。
- Client-Library エラー番号の情報については、『Open Client Client-Library/C リファレンス・マニュアル』の「CS_CLIENTMSG 構造体」を参照してください。

コマンドを送信するコード

Client-Library の CS_COMMAND は、コマンドをサーバに送信し、結果を処理するための制御構造体です。1 つの接続構造体から、複数のコマンド構造体を割り付けることができます。

DB-Library アプリケーションは、次のタイプのコマンドを送信できます。

- 言語コマンド ― 1 つまたは複数の SQL 文のバッチを定義し、そのバッチをサーバに送信してコンパイルおよび実行します。「言語コマンドの送信」(54 ページ) を参照してください。
- RPC (リモート・プロシージャ・コール) コマンド ― Adaptive Server Enterprise ストアド・プロシージャまたは Open Server レジスタード・プロシージャを呼び出して、宣言されたデータ型でパラメータを渡します。「RPC コマンドの送信」(56 ページ) を参照してください。
- TDS passthrough 呼び出し ― TDS パケットを読み書きするために、Open Server ゲートウェイによって使用されます。「TDS パススルー」(61 ページ) を参照してください。

このほかにも、DB-Library に同等の機能がない Client-Library コマンド・タイプがあります。Client-Library コマンド・タイプについては、『Open Client Client-Library/C プログラマーズ・ガイド』の「第 5 章 コマンド・タイプの選択」で説明されています。

言語コマンドの送信

言語コマンドは、1 つまたは複数の SQL 文のバッチを定義し、そのバッチをサーバに送信してコンパイルおよび実行します。

表 5-5 では、言語コマンドを送信するのに使用する DB-Library ルーチンと、それと同等の Client-Library の機能を比較します。

表 5-5 : DB-Library と Client-Library の比較 (言語コマンドの送信)

DB-Library ルーチン	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
(なし)	(なし)	ct_cmd_alloc(connection, cmd_pointer)	CS_COMMAND 構造体を割り付ける。
dbfcmd(dbproc, string, args...)	テキストをフォーマットして DBPROCESS コマンド・バッファに追加する。DB-Library には 1K のバッファ制限がある。	sprintf(cmd_string, control_string, args...)	テキストをフォーマットし、sprintf、strcpy、または他のシステム呼び出しを使用して言語コマンド文字列を初期化する。
dbcmd(dbproc, string)	テキストを DBPROCESS コマンド・バッファに追加する。	ct_command(cmd, CS_LANG_CMD, cmd_string, string_len, CS_MORE)	後にコマンド・テキストが続く形で *cmd_string* を使用して、言語コマンドを開始する。
(なし)		ct_command(cmd, CS_LANG_CMD, cmd_string, string_len, CS_END)	このコマンドの最後のコマンド・テキストとして *cmd_string* を追加する。
dbsqlexec(dbproc)	コマンド・バッチを実行するためにサーバに送信する。	ct_send(cmd)	コマンド・バッチを実行するためにサーバに送信する。

Client-Library の強化機能

Client-Library では、次のように言語コマンド機能が強化されています。

- 言語コマンドに、(コマンド・テキストの “@param” のような宣言されていない変数で識別される) ホスト言語パラメータを指定できます。最新の ct_command 呼び出しと ct_send 呼び出しの間で、アプリケーションは、ct_param または ct_setparam を呼び出して、各ホスト言語パラメータの値を指定します。

- Client-Library では、言語コマンドを再送信することができます。前の実行の結果を処理したらすぐに、アプリケーションは、ct_send を呼び出して同じコマンドを再送信できます。言語コマンドとそのパラメータの定義は、アプリケーションが ct_command、ct_cursor、ct_dynamic、または ct_sendpassthru を呼び出して同じコマンド構造体で新しいコマンドを起動するまで、コマンド構造体と対応したままになります。

例：Client-Library 言語コマンドの送信

次のコード例は、Client-Library 言語コマンドを送信する例です。この例は *ex01ct.c* 移行サンプル・プログラムの一部です。

```
CS_CONNECTION *conn;
CS_COMMAND *cmd;

... connection has been opened ...
/*
** Allocate a command structure.
*/
ret = ct_cmd_alloc(conn, &cmd);
EXIT_ON_FAIL(context, ret, "Could not allocate command structure.");  /*
-- dbcmd(dbproc, "select name, type, id, crdate from sysobjects");
-- dbcmd(dbproc, " where type = 'S'");
-- dbcmd(dbproc, "select name, type, id, crdate from sysobjects");
-- dbcmd(dbproc, " where type = 'P'");
*/

/*
** Build up a language command. ct_command() constructs language,
** RPC, and some other server commands.
**
** Note that the application manages the language buffer:You
** must format the language string with stdlib calls before
** passing it to ct_command().
*/
strcpy(sql_string, "select name, type, id, crdate from sysobjects");
strcat(sql_string, " where type = 'S'");
strcat(sql_string, "select name, type, id, crdate from sysobjects");
strcat(sql_string, " where type = 'P'");
ret = ct_command(cmd, CS_LANG_CMD, (CS_VOID *) sql_string,
                 CS_NULLTERM,CS_UNUSED);
EXIT_ON_FAIL(context, ret, "Init language command failed.");  /*
--  * Send the commands to Adaptive Server Enterprise and start execution.*
-- dbsqlexec(dbproc);
```

```
*/
/*
** Send the command.Unlike dbsqlexec(), ct_send() returns as
** soon as the command has been sent.It does not wait for
** the results from the first statement to arrive.
*/
ret = ct_send(cmd);
EXIT_ON_FAIL(context, ret, "Send language command failed.");
... deleted results processing code ...
```

RPC コマンドの送信

RPC コマンドは、Adaptive Server Enterprise ストアド・プロシージャか Open Server レジスタード・プロシージャを呼び出して、宣言されたデータ型でパラメータを渡します。

表 5-6 では、RPC コマンドを定義および送信するための Client-Library と DB-Library の呼び出しシーケンスを比較します。

表 5-6：DB-Library と Client-Library の比較 (RPC コマンドの送信)

DB-Library ルーチン	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
(なし)	(なし)	ct_cmd_alloc(connection, cmd_pointer)	CS_COMMAND 構造体を割り付ける。
dbrpcinit(dbproc, rpc_name, option)	RPC を初期化する。 *option* は DBRPCRECOMPILE または 0。	ct_command(cmd, CS_RPC_CMD, rpc_name, buflen, option)	RPC コマンドを開始する。 *option* は CS_RECOMPILE、CS_NO_RECOMPILE、または CS_UNUSED。DB-Library プログラムにある 0 の値は、CS_UNUSED または CS_NO_RECOMPILE にマップする。
dbrpcparam(dbproc, paramname, status, type, maxlen, datalen, data)	RPC にパラメータを追加する。	ct_param または ct_setparam(cmd, datafmt, data, datalen, indicator)	RPC パラメータを定義する。
dbrpcsend(dbproc)	RPC 呼び出しを実行するためにサーバに送信する。	ct_send(cmd)	コマンドを実行するためにサーバに送信する。

RPC コマンドの ct_param の使い方は、dbrpcparam の使い方とよく似ています。dbrpcparam のパラメータのほとんどは、ct_param の *datafmt* パラメータとして渡される CS_DATAFMT 構造体のフィールドにマップされます。

- dbrpcparam の *paramname*、*status*、*type*、*maxlen* パラメータは、ct_param の *datafmt* パラメータとして受け取る CS_DATAFMT 構造体のフィールドにマップされます。
- dbrpcparam 呼び出しでは、*datalen* を 0 として渡すことで null 値を指定します。
  ct_param 呼び出しでは、*indicator* を -1 として渡すことで null 値を指定します。

Client-Library の強化機能

DB-Library とは異なり、Client-Library では、アプリケーションは RPC コマンドを再送信できます。アプリケーションは、前の実行の結果を処理したあとで ct_send を呼び出すだけで、RPC コマンドを再送信できます。RPC コマンドとそのパラメータの定義は、アプリケーションが ct_command、ct_cursor、ct_dynamic、または ct_sendpassthru を呼び出して同じコマンド構造体で新しいコマンドを起動するまで、コマンド構造体と対応したままになります。

例：RPC コマンドの送信

次のコード例は、Client-Library で RPC コマンドを送信する例です。この例では、次のように定義されている Adaptive Server Enterprise ストアド・プロシージャ rpctest を呼び出します。

```
create procedure rpctest
         (@param1 int out,
          @param2 int out,
          @param3 int out,
          @param4 int)
as
     begin
         select "rpctest is running."
         select @param1 = 11
         select @param2 = 22
         select @param3 = 33
         select @param1
         return 123
     end
```

次のコードは、Client-Library クライアントから rpctest を呼び出します。この例は *ex08ct.c* 移行サンプル・プログラムの一部です。

```
CS_CONNECTION *conn;
  CS_COMMAND *cmd;

  ... connection has been opened ...

  /*
  ** Allocate a command structure.
  */
  ret = ct_cmd_alloc(conn, &cmd);
  EXIT_ON_FAIL(context, ret, "Could not allocate command structure.");  /*
  -- * Make the rpc.*
  -- if (dbrpcinit(dbproc, "rpctest", (DBSMALLINT)0) == FAIL)
  -- {
  --   printf("dbrpcinit failed.¥n");
--   dbexit();
-- exit(ERREXIT);
  -- }
  */  /*
  ** Initiate an RPC command.In Client-Library ct_command is used for
  ** language commands (dbsqlexec or dbsqlsend commands in DB-Library),
  ** RPC commands (dbrpcinit), and text/image "send-data" commands
  ** (dbwritetext).
  */

  ret = ct_command(cmd, CS_RPC_CMD, "rpctest", CS_NULLTERM, CS_UNUSED);
  EXIT_ON_FAIL(context, ret, "Could not initiate RPC command.");  /*
  ** Pass a value for each RPC parameter with ct_param. In this case,
  ** the required RPC parameters are the parameters in the definition of
  ** the rpctest stored procedure.
  **
  ** The parameter's name, datatype, and status (input-only or output)
  ** are passed within a CS_DATAFMT structure.
  */  /*
  -- if (dbrpcparam
  --       (dbproc, "@param1", (BYTE)DBRPCRETURN,
  --        SYBINT4, -1, -1, &param1)
  --   == FAIL)
  -- {
  --   printf("dbrpcparam failed.¥n");
--   dbexit();
-- exit(ERREXIT);
  -- }
  */  /*
  ** @param1 is integer (CS_INT) and is a return parameter.
```

```
  ** The datafmt.status field must be set to indicate whether
  ** each parameter is ’for output’(CS_RETURN) or not
  ** (CS_INPUTVALUE)
  */  datafmt.datatype = CS_INT_TYPE;
  datafmt.maxlength = CS_UNUSED;
  datafmt.status = CS_RETURN;
  strcpy(datafmt.name, "@param1");
  datafmt.namelen = strlen(datafmt.name);  ret = ct_param(cmd, &datafmt,
  (CS_VOID *) (paramvals+1),
                CS_UNUSED, 0);
  EXIT_ON_FAIL(context, ret, "ct_param() for @param1 failed.");  /*
  -- if (dbrpcparam(dbproc, "@param2", (BYTE)0, SYBINT4,
  --                -1, -1, &param2)
  --   == FAIL)
  -- {
  --   printf("dbrpcparam failed.¥n");
--   dbexit();
-- exit(ERREXIT);
  -- }
  */  /*
  ** @param2 is integer (CS_INT) and is not a return parameter.
  */
  datafmt.datatype = CS_INT_TYPE;
  datafmt.maxlength = CS_UNUSED;
  datafmt.status = CS_INPUTVALUE;
  strcpy(datafmt.name, "@param2");
  datafmt.namelen = strlen(datafmt.name);  ret = ct_param(cmd, &datafmt,
  (CS_VOID *) (paramvals+2),
                CS_UNUSED, 0);
  EXIT_ON_FAIL(context, ret, "ct_param() for @param2 failed.");  /*
  -- if (dbrpcparam


  --        (dbproc, "@param3", (BYTE)DBRPCRETURN, SYBINT4,
  --         -1, -1, &param3)
  --   == FAIL)
  -- {
  --   printf("dbrpcparam failed.¥n");
--   dbexit();
-- exit(ERREXIT);
  -- }
  */  /*
  ** @param3 is integer (CS_INT) and is a return parameter.
  */

  datafmt.datatype = CS_INT_TYPE;
  datafmt.maxlength = CS_UNUSED;
```

```
  datafmt.status = CS_RETURN;
  strcpy(datafmt.name, "@param3");
  datafmt.namelen = strlen(datafmt.name);  ret = ct_param(cmd, &datafmt,
  (CS_VOID *) (paramvals+3),
                CS_UNUSED, 0);
  EXIT_ON_FAIL(context, ret, "ct_param() for @param3 failed.");  /*
  -- if (dbrpcparam(dbproc, "@param4", (BYTE)0, SYBINT4,
  --                -1, -1, &param4)
  --   == FAIL)
  -- {
  --   printf("dbrpcparam failed.¥n");
--   dbexit();
-- exit(ERREXIT);
  -- }
  */  /*
  ** @param4 is integer (CS_INT) and is not a return parameter.
  */
  datafmt.datatype = CS_INT_TYPE;
  datafmt.maxlength = CS_UNUSED;
  datafmt.status = CS_INPUTVALUE;
  strcpy(datafmt.name, "@param4");
  datafmt.namelen = strlen(datafmt.name);  ret = ct_param(cmd, &datafmt,
  (CS_VOID *) (paramvals+4),
                CS_UNUSED, 0);
  EXIT_ON_FAIL(context, ret, "ct_param() for @param4 failed.");  /*
  -- if (dbrpcsend(dbproc) == FAIL)
  -- {
  --   printf("dbrpcsend failed.¥n");
--   dbexit();
-- exit(ERREXIT);
  -- }
  */  /*


  ** Send the command to the server.The ct_send routine sends
  ** any kind of command, not just RPC commands.
  */
  ret = ct_send(cmd);
  EXIT_ON_FAIL(context, ret, "ct_send() failed.");

  ... deleted results processing code ...
```

TDS パススルー

TDS (Tabular Data Stream) 転送ルーチンは、ゲートウェイ・アプリケーションで有用です。DB-Library ルーチンの dbrecvpassthru と dbsendpassthru は、Client-Library ルーチンの ct_recvpassthru と ct_sendpassthru に直接マップします。Client-Library ルーチンは CS_COMMAND 構造体を使用するのに対して、DB-Library ルーチンは DBPROCESS 構造体を使用します。

結果を処理するコード

ここでは、DB-Library 結果処理が Client-Library 結果処理にどのようにマップするかを説明します。

結果処理のプログラム構造体

表 5-7 は、DB-Library プログラムに見られる結果のタイプを処理するためのループ構造体を示します。また、表 5-8 (62 ページ) は、これに対応する Client-Library プログラム論理を示します。

表 5-7 : DB-Library の結果ループ構造体

ループ制御	`while ((results_ret = dbresults(dbproc)) != NO_MORE_RESULTS)`
	`{`
	`  if (results_ret == SUCCEED)`
	`  {`

通常ローと計算ローの取得	*Bind regular rows.*
	Bind compute rows.
	`  while (dbnextrow(dbproc)`
	`  != NO_MORE_ROWS)`
	`  {`
	* 通常ローと計算ローの取得*
	`  } /* while */`

リターン・パラメータ値の取得	`  if (dbnumrets(dbproc) > 0)`
	`  {`
	* Retrieve output parameter values.*
	`  }`

リターン・ステータス値の取得	`  if (dbhasretstatus(dbproc))`
	`  {`
	* Retrieve stored procedure return status.*
	`  }`

(オプション) 情報の取得	`if (DBROWS(dbproc) != -1)`
	`{`
	`  Find out number of rows affected.`
	`}`

コマンド・エラー・チェック (サーバ側またはクライアント側)	`} /* if results_ret == SUCCEED */`
	
	`else if (results_ret == FAIL)`
	`{`
	`  printf( "Command failed");`
	`}`
	`} /* while */`

表 5-8 は、標準的な Client-Library プログラムの結果ループ構造体を示します。

表 5-8 : Client-Library の結果ループ構造体

ループ制御	`while ((results_ret = ct_results(cmd, &result_type))`
	`== CS_SUCCEED)`
	`{`
	`  switch(result_type)`
	`  {`

通常ローと計算ローの取得	`  case CS_ROW_RESULT:`
	`  Bind regular rows.`
	`  Fetch regular rows.`
	`break;`
	
	`  case CS_COMPUTE_RESULT:`
	`  Bind compute rows.`
	`  Fetch compute rows.`
	`break;`

リターン・パラメータ値の取得	`  case CS_PARAM_RESULT:`
	`  Bind output parameter values.`
	`  Fetch output parameter values.`
	`break;`

リターン・ステータス値の取得	`  case CS_STATUS_RESULT:`
	`  Bind stored procedure return status.`
	`  Fetch stored procedure return status.`
	`break;`

(オプション) 情報の取得	`  case CS_CMD_DONE:`
	`  Find out number of rows affected.`
	`break;`

コマンド・エラー・チェック (サーバ側)

```
            case CS_CMD_FAIL:
                printf("Command failed on server.")
break;

            case CS_CMD_SUCCEED:
break;
```

コマンド・エラー・チェック (クライアント側)

```
            default:/* case */
                printf("Unexpected result type");
break;

        } /* end switch */
    } /* end while */
    if (results_ret != CS_END_RESULTS
        && results_ret != CS_CANCELED)
        printf("ERROR:ct_results failed!");
```

dbresults と ct_results のリターン・コードの比較

DB-Library の dbresults は、SUCCEED、FAIL、または NO_MORE_RESULTS を返します。

- SUCCEED は、コマンドが正常に実行されて、アプリケーションが取得するデータが存在する可能性があることを示します。
- FAIL は、通常は、コマンドがサーバで実行できなかったことを示しますが、ネットワークまたは内部 DB-Library エラーも示します。また、コマンドがサーバで実行に失敗して、dbresults が FAIL を返しても、後続コマンドからのデータが使用可能な場合もあります。
- NO_MORE_RESULTS は、処理する結果がないことを示します。標準的なアプリケーションは、NO_MORE_RESULTS が返されるまで、dbresults をループで呼び出します。アプリケーションは、ループ内で dbresults からのリターン・コード SUCCEED または FAIL を探します。

Client-Library の場合、同期モードの ct_results 呼び出しは、CS_SUCCEED、CS_FAIL、CS_CANCELED、または CS_END_RESULTS を返します (非同期呼び出しの場合、完了ステータスはこれらの値のいずれかです)。

- CS_SUCCEED は、ct_results ルーチンが正常に実行されたことを示します。コマンドの結果については、何も表示しません。

- CS_FAIL は、ct_results ルーチンが正常に実行できなかったことを示します。このコードが返された場合は、必ず、重大なネットワーク・エラーかクライアント側エラーが発生しています。ct_results が CS_FAIL を返したあとは、結果データは使用できません。
- CS_END_RESULTS は、dbresults の NO_MORE_RESULTS とまったく同じ意味です。
- CS_CANCELED は、ct_cancel(CS_CANCEL_ATTN) または ct_cancel(CS_CANCEL_ALL) によって結果がキャンセルされたことを示します。

ct_results は、*result_type* 出力パラメータを使用して、サーバ側のエラーまたは成功を示します。

- 結果タイプの CS_CMD_FAIL は、コマンドがサーバで実行できなかったことを示します。DB-Library の場合は、dbsqlexec、dbsqlok、または dbresults のうち、サーバがエラーを通知したときにアクティブであったものから FAIL を返して、コマンドの失敗を通知します。
- 結果タイプの CS_CMD_SUCCEED は、データ変更コマンド (create、update、insert など) または exec コマンドが正常に実行されたことを示します。たとえば、delete 言語コマンドが正常に実行されたあと、アプリケーションは *result_type* 値として CS_CMD_SUCCEED を受け取ります。

コマンド処理エラーの処理

ここでは、DB-Library と Client-Library で、コマンド処理エラーの処理方法がどのように異なるかを、2 つの例で示します。

- アプリケーションが、構文エラーを含む言語コマンドを送信する場合

  DB-Library では、dbsqlexec または dbsqlok のうち、どちらか呼び出されている方がアプリケーションのサーバ・メッセージ・ハンドラを呼び出して、サーバから通知されたエラーを転送します。dbsqlexec または dbsqlok は FAIL を返します。データは返されず、dbresults 呼び出しは NO_MORE_RESULTS を返します。

Client-Library では、ct_results が、アプリケーションのサーバ・メッセージ・ハンドラを呼び出して、サーバから通知されたエラーを転送します。ct_results は CS_SUCCEED を返しますが、*result_type* は CS_CMD_FAIL に設定されます。アプリケーションは、残りの結果を ct_results で処理するか、ct_cancel で処理する必要があります。

- 4 つの文からなる言語バッチの 2 番目の文でオブジェクトを選択するが、ユーザには対象となるオブジェクトの select パーミッションがない場合

  DB-Library では、dbresults がアプリケーションのサーバ・メッセージ・ハンドラを呼び出して、サーバから通知されたパーミッション違反を転送します。dbresults は FAIL を返します。このバッチ内の残りのコマンドからの結果は使用可能であり、アプリケーションは、これらの結果を dbresults で取得するか、dbcancel でキャンセルする必要があります。

  Client-Library では、ct_results がアプリケーションのサーバ・メッセージ・ハンドラを呼び出して、サーバから通知されたパーミッション違反を転送します。ct_results は CS_SUCCEED を返しますが、*result_type* は CS_CMD_FAIL に設定されます。アプリケーションは、残りの結果を ct_results で処理するか、ct_cancel で処理する必要があります。

ct_results の *result_type* と DB-Library プログラム論理の比較

Client-Library では、ct_results は *result_type* インジケータへのポインタ引数を取ります。*result_type* は、コマンド・ステータス (CS_CMD_SUCCEED と CS_CMD_FAIL) を示すほかに、結果が使用可能かどうか、それらがどのようなタイプの結果であるかも示します。

表 5-9 では、*result_type* の取り得る値をリストし、対応する DB-Library プログラム論理と比較します。『Open Client Client-Library/C リファレンス・マニュアル』の「ct_results」のリファレンス・ページを参照してください。

表 5-9 : ct_results の result_type パラメータと DB-Library プログラム論理の比較

Client-Library *result_type*	意味	DB-Library プログラム論理
CS_CMD_DONE	論理コマンドの結果の処理が完了した。	なし Client-Library プログラムによる CS_CMD_DONE の受信は、DB-Library dbresults ループ 1 反復の終了に相当する。

Client-Library *result_type*	意味	DB-Library プログラム論理
CS_CMD_FAIL	コマンド実行中にサーバがエラーを検出した。	アクティブ・ルーチン (dbsqlexec、dbsqlok、または dbresults) が FAIL を返す。
CS_CMD_SUCCEED	Transact-SQL insert 文を含む言語コマンドなどの、データを返さないコマンドが正常に終了した。	dbresults が SUCCEED を返す。DBCMDROW は FAIL を返し、コマンドがローを返せなかったことを示す。
CS_COMPUTE_RESULT	計算ローの結果。	DBROWS を呼び出して、ローが返されるかどうかを判断する。Client-Library には、DBROWS に相当する呼び出しまたはマクロは存在しない。 dbnumcompute を呼び出して、計算ローが返されるかどうかを判断する。 dbnextrow ループでは、計算ローが取得されると、dbnextrow が > 0 を返す。
CS_PARAM_RESULT	リターン・パラメータ結果。	dbnextrow が NO_MORE_ROWS を返したあと、dbnumrets が > 0 を返すかどうかを確認する。
CS_ROW_RESULT	通常のローの結果。	現在のコマンドがローを返せる場合は、DBCMDROW が TRUE を返す。 dbnextrow は、通常のローがそれぞれ取得されたあと、REG_ROW を返す。
CS_STATUS_RESULT	ストアド・プロシージャ・リターン・ステータス結果。	dbnextrow が NO_MORE_ROWS を返したあと、dbhasretstat が TRUE を返すかどうかを確認する。
CS_CURSOR_RESULT	カーソル・ロー結果。	なし DB-Library はサーバ・ベースのカーソルをサポートしない。
CS_COMPUTEFMT_RESULT	• 計算ロー・フォーマット情報。 • フォーマット結果は、CS_EXPOSE_FORMATS プロパティが有効になっている場合にだけ表示される。	なし。
CS_ROWFMT_RESULT	• 通常のローのフォーマット情報。 • フォーマット結果は、CS_EXPOSE_FORMATS プロパティが有効になっている場合にだけ表示される。	なし。

Client-Library *result_type*	意味	DB-Library プログラム論理
CS_MSG_RESULT	Client-Library メッセージ結果セットの到着。	なし。DB-Library はメッセージ・コマンドと結果をサポートしない。
CS_DESCRIBE_RESULT	動的 SQL 記述情報。	なし。DB-Library は動的 SQL をサポートしない。

データ値の検索

Client-Library アプリケーションは、DB-Library の dbbind/dbnextrow モデルによく似たバインド／フェッチ・モデルを使用して、データを取得します。両者の主な違いは、Client-Library の方が多くのタイプの結果データをフェッチ可能である点です。次にリストするすべての結果タイプのデータ値が、ct_bind と ct_fetch を使用して取得できます。

- 通常のロー (DB-Library でもフェッチ可能)
- 計算ロー (DB-Library でもフェッチ可能)
- 出力パラメータ値
- ストアド・プロシージャ・リターン・ステータス値

注意　DB-Library では、出力パラメータ値とリターン・ステータス値の検索はオプションです。Client-Library アプリケーションは、出力パラメータ値とリターン・ステータス値を含めて、サーバから送信されるフェッチ可能な結果をすべて取得またはキャンセルする必要があります。

ct_bind と *dbbind* の比較

DB-Library には、次の 4 つのバインド・ルーチンが用意されています。

- dbbind － 通常のローのカラムをバインドします。
- dbbind_ps (バージョン 10.0 以降) － dbbind と同じですが、decimal データ型と numeric データ型の精度と位取りをサポートします。
- dbaltbind － 計算ロー・カラムをバインドします。
- dbaltbind_ps (バージョン 10.0 以降) － dbaltbind_ps と同じですが、decimal データ型と numeric データ型の精度と位取りをサポートします。

dbbind_ps の用途が ct_bind の用途にどのようにマップするかがわかっていれば、ほかのどの DB-Library バインド・ルーチン呼び出しでも、対応する ct_bind 呼び出しに変換できます。dbbind_ps は、dbbind の機能を強化したルーチンです。. このルーチンは、追加パラメータとして DBTYPEINFO 構造体を取り、numeric データ型と decimal データ型の精度情報と位取り情報を伝達します。numeric、decimal 以外のデータ型については、追加パラメータは無視され、dbbind_ps の機能は dbbind とまったく同じです。

表 5-10 では、dbbind_ps パラメータと ct_bind パラメータを比較します。

表 5-10 : dbbind_ps パラメータと ct_bind パラメータの比較

dbbind_ps パラメータ	パラメータの説明	ct_bind パラメータ	パラメータの説明
dbproc	この接続の DBPROCESS 構造体へのポインタ。	*cmd*	CS_COMMAND 構造体へのポインタ。
column	バインドするカラムの番号を表す整数。	*item*	バインドするカラムの番号を表す整数。
		datafmt	送信先変数を記述する CS_DATAFMT へのポインタ。
vartype	DBPROCESS からデータのコピーを受け取るプログラム変数のデータ型に対応する記号値。	*datafmt–>datatype*	*detatype* は送信先変数のデータ型を表す記号 (CS_xxx_TYPE)。
		datafmt–>format	*format* は文字またはバイナリ・データの送信先フォーマットを記述する記号。
varlen	プログラム変数の長さ (バイト単位)。	*datafmt–>maxlength*	*buffer* 送信先変数の長さ (バイト単位)。
typeinfo–>precision *typeinfo–>scale*	*typeinfo* は DBTYPEINFO 構造体へのポインタ。この構造体には、decimal または numeric データの精度と位取りに関する情報が含まれている。 NULL の *typeinfo* は dbbind の呼び出しと同じ。	*datafmt–>precision* *datafmt–>scale*	送信先変数に使用する精度と位取り。送信元データが送信先と同じデータ型である場合、位取りと精度を CS_SRC_VALUE に設定して、送信元データから値を取り出すことができる。
(なし)		*datafmt–>count*	1 回の ct_fetch 呼び出しでプログラム変数にコピーするローの数 (配列にバインドしない場合は、1 に設定する)。
varaddr	データのコピー先であるプログラム変数のアドレス。	*バッファ (buffer)*	*datafmt–>count* 変数の配列のアドレス。各変数のサイズは *datafmt–>maxlength*。

dbbind_ps パラメータ	パラメータの説明	ct_bind パラメータ	パラメータの説明
(なし)		*copied*	*datafmt–>count* 整数変数の配列のアドレス。各変数には、フェッチ時に、コピーしたデータの長さが入る (オプション)。
(なし ― dbnullbind ルーチンと dbanullbind ルーチンはインジケータ変数をバインドする)		*indicator*	*datafmt–>count* CS_SMALLINT 変数の配列のアドレス。各変数はフェッチ時に充てんされ、フェッチしたデータに関する特定条件を示す。

DB-Library *vartype* 値を Client-Library CS_DATAFMT datatype および *format* 値にマップする作業は、すべての固定長データ型について簡単に行えます。

character 型と binary 型のマップを 表 5-11 に示します。

表 5-11 : DB-Library の vartype と CS_DATAFMT データ型および format フィールド

プログラム変数タイプ	DB-Library *vartype*	CS_DATAFMTA*datatype*	CS_DATAFMTA*format*
DBCHAR	CHARBIND	CS_CHAR_TYPE	CS_FMT_PADBLANK
DBCHAR	STRINGBIND	CS_CHAR_TYPE	CS_FMT_NULLTERM
DBCHAR	NTBSTRINGBIND	CS_CHAR_TYPE	CS_FMT_NULLTERM **注意**　Client-Library は後続ブランクをトリムしない。
DBVARYCHAR	VARYCHARBIND	CS_VARCHAR_TYPE	CS_FMT_UNUSED
DBBINARY	BINARYBIND	CS_BINARY_TYPE	CS_FMT_PADNULL
DBVARYBIN	VARYBINBIND	CS_VARBINARY_TYPE	CS_FMT_UNUSED

dbbind では、*vartype* の NTBSTRINGBIND を渡すと、DB-Library が後続ブランクを送信先文字列からトリムします。Client-Library には、後続ブランクを除去するフォーマット・オプションはありません。

Adaptive Server Enterprise カラム・データの場合、初めから後続ブランクがあるのは、固定長 char カラムとして作成された値だけです。これは、入力時に Adaptive Server Enterprise がエントリの varchar カラムから後続ブランクをトリムするためです。

NTBSTRINGBIND 動作に依存している DB-Library アプリケーションの場合、そのアプリケーションの Client-Library バージョンでは、後続ブランクそのものをトリムする必要があります。

ct_get_data と dbdata の比較

Client-Library には、DB-Library の dbdata ルーチンまたはこれに類する dbadata、dbretdata、dbretstatus ルーチンとまったく同等の機能は用意されていません。これらのルーチンはすべて、データ値が入るバッファへのポインタを返します。

Client-Library では、バインドの代替として、ct_get_data でデータ値を取得できます。アプリケーションは一般に、大きな text または image カラムを取得するのに ct_get_data を使用しますが、このルーチンはどのデータ型のデータにも使用できます。

ct_get_data は、データ値の全部または一部を、呼び出し元が用意したバッファにコピーします。dbdata、dbadata、dbretdata、または dbretstatus に対する呼び出しは、ct_get_data に対する呼び出しに置き換えることができます。ただし、ct_get_data には、次のような制限があります。

- ct_get_data では、アプリケーションはデータ用のバッファを事前に割り付けておく必要があります。
- アプリケーションは、ct_get_data を、ct_bind で最後にバインドした項目より後の結果項目にしか使用できません。たとえば、結果項目番号 1、3、4 がバインドされている場合、項目番号 1 から 4 に ct_get_data 呼び出しを行うと、エラーになります。
- dbretdata と dbretstatus では、アプリケーションはパラメータ値またはリターン・ステータス値をフェッチする必要がありませんでした。Client-Library では、前もって ct_fetch を呼び出さなければ、リターン・パラメータ値またはリターン・ステータス値を ct_get_data で取得することはできません。
- CS_SUCCEED を返す ct_fetch に対する各呼び出しでは、アプリケーションは 1 回の ct_get_data で 1 データ項目しか取得できません。

次のコード例は、CS_INT データ項目を取得する ct_get_data 呼び出しの例です。

```
CS_INT status;
... after ct_fetch() has returned CS_SUCCEED ...
ret = ct_get_data(cmd, 1, (CS_VOID *)status,
                  CS_SIZEOF(CS_INT), (CS_INT *) NULL);
if (ret != CS_END_ITEM && ret != CS_END_DATA)
{
```

```
   printf("Error:ct_get_data failed.¥n");
}
else
{
   printf("Status is %ld.¥n", (long) status);
}
```

dbdata の場合と同様に、ct_get_data で取得されたデータは、希望するデータ型で値が表されていない場合には、変換が必要です。Client-Library アプリケーションは CS-Library ルーチンの cs_convert を呼び出してデータを変換します。

結果データの記述の取得

アプリケーションは、項目をバインドし、ローをフェッチする前に、結果セット内の項目の数と各項目のフォーマットを決定する必要があります。

既知のクエリの結果を処理するアプリケーションはすでにこの情報を持っていますが、アドホック・クエリの結果を処理するアプリケーションには、この情報がありません。

アドホック・クエリの結果を処理するには、アプリケーションは次の項目を決定する必要があります。

- 結果カラムの数
- 各カラムの名前、データ型、長さなど

結果セット内の項目数の取得

DB-Library では、アプリケーションは、取り出す結果のタイプによって異なるルーチンを呼び出して、結果セット内の項目の数を取得します。

Client-Library では、ct_results の *result_type* パラメータがフェッチ可能なデータを表示するたびに、アプリケーションは ct_res_info(CS_NUMDATA) を呼び出してデータ項目の数を取り出すことができます。

表 5-12 は、ct_res_info(CS_NUMDATA) に置き換えられる DB-Library ルーチンを示します。

表 5-12：ct_res_info(CS_NUMDATA) に変換される DB-Library ルーチン

ルーチン	説明
dbnumalts	計算ローのカラム数を返す。
dbnumcols	現在の結果セットの通常カラム数を決定する。
dbnumrets	ストアド・プロシージャによって生成されるリターン・パラメータ値の数を決定する。

個別項目のフォーマット記述の取得

DB-Library アプリケーションは、1 データ項目の記述を得るのに、複数のルーチンを呼び出します。

Client-Library アプリケーションは、ct_describe に対する 1 回の呼び出しで、データ値を完全に記述している CS_DATAFMT 構造体を初期化します。

表 5-13 は、ct_describe に置き換えられる DB-Library ルーチンを示します。

表 5-13：DB-Library データ記述ルーチンと CS_DATAFMT フィールドの対応

DB-Library ルーチン	返される値	CS_DATAFMT フィールド (ct_describe によって設定される)
dbaltlen	特定の計算カラムの最大データ長	maxlength
dbcollen	特定の通常の結果カラムの最大データ長	maxlength
dbretlen	ストアド・プロシージャ・リターン・パラメータ値の長さ	maxlength
dbalttype	計算カラムのデータ型	datatype
dbcoltype	通常の結果カラムのデータ型	datatype
dbrettype	ストアド・プロシージャ・リターン・パラメータ値のデータ型	datatype
dbaltutype	計算カラムのユーザ定義データ型	usertype
dbcolutype	通常の結果カラムのユーザ定義データ型	usertype
dbcolname	通常の結果カラムの名前	name
dbretname	特定のリターン・パラメータ値のストアド・プロシージャ・パラメータの名前	name
dbdatlen	通常の結果カラム値の実際の長さ	なしこの情報は ct_bind の *copied* パラメータまたは ct_get_data の *outlen* パラメータを使用して返される。
dbadlen	計算カラム値の実際の長さ	
dbretlen	リターン・パラメータ値の実際の長さ	

結果情報の取得

DB-Library には、DBCURCMD、DBCOUNT など、アプリケーションが結果情報を取得できるようにするルーチンが用意されています。

これらの DB-Library ルーチンのほとんどは Client-Library の ct_res_info ルーチンに直接マップします。

コマンド番号の取得 (DBCURCMD)

DB-Library の DBCURCMD は現在の論理コマンドの番号を返します。

Client-Library では、ct_res_info(CS_CMD_NUMBER) が現在の論理コマンドの番号を返します。

次の Client-Library コード例は、ct_res_info を使用して、現在のコマンド番号を得る方法を示します。

```
CS_INT cur_cmdnum;
...
ret = ct_res_info(cmd, CS_CMD_NUMBER, &cur_cmdnum,
CS_UNUSED, NULL);
EXIT_ON_FAIL(context, ret,
            "ct_res_info(CMD_NUMBER) failed.");
```

影響を受けたロー数の取得

DB-Library の DBCOUNT は、現在のサーバ・コマンドによって影響を受けたローの数を返します。DBCOUNT は、すべてのローを取得したあとで、dbresults ループ内で呼び出されます。

Client-Library では、ct_res_info(CS_ROW_COUNT) が現在のサーバ・コマンドによって影響を受けたローの数を返します。どのローにも影響を与えないコマンドの場合には、ct_res_info も DBCOUNT と同様、-1 のロー・カウントを返します。

次のコード例は、ct_res_info を使用して、ロー・カウントを得る方法を示します。この例は、*result_type* が CS_CMD_DONE である場合には、ct_results ループで実行されます。

```
CS_INT rowcount;
...
ret = ct_res_info(cmd, CS_ROW_COUNT, (CS_VOID *)&rowcount,
CS_UNUSED, NULL);
EXIT_ON_FAIL(context, ret, "ct_res_info(CS_ROW_COUNT) failed.");
if (rowcount != -1)
  printf("(%ld rows affected)¥n", rowcount);
```

DBCOUNT と ct_res_info(CS_ROW_COUNT) はほとんど同じように機能します。両者とも、現在のコマンドによって影響を受けたローの数を返します。ただし、現在のコマンドがストアド・プロシージャを実行するコマンドである場合は、両者の動作に次のような大きな違いがあります。

- DBCOUNT はストアド・プロシージャが最後に実行した select 文によって影響を受けたローの数を返します。

  たとえば、ストアド・プロシージャが実行した最後の 2 つの文が select 文と update 文である場合、DBCOUNT は update ではなく select によって影響を受けたローの数を返します。

- ct_res_info(CS_ROW_COUNT) は、ストアド・プロシージャによって実行された、ローに影響を与える最後の文によって影響を受けたローの数を返します。

  たとえば、プロシージャによって実行された最後の 2 つの文が select 文と update 文である場合、ct_res_info(CS_ROW_COUNT) は update によって影響を受けたローの数を返します。

DB-Library アプリケーションがストアド・プロシージャ実行後の DBCOUNT の動作に論理的に依存する場合、アプリケーションを Client-Library に変換するときにプログラム論理を変更する必要があります。

現在のローの番号の取得

DB-Library の DBCURROW マクロは通常のローの結果セットにある現在のローを返します。アプリケーションはローの処理中に DBCURROW を呼び出して、中間のロー・カウントを得ることができます。

Client-Library には、DBCURROW 呼び出しに代わるルーチンはありません。ただし、各ローをフェッチするたびにカウンタ値を増加するアプリケーション・コードを追加できます。詳細については、表 A-1 (101 ページ) の DBCURROW を参照してください。

結果のキャンセル

DB-Library プログラムは dbcanquery および dbcancel でクエリをキャンセルし、結果を廃棄します。

Client-Library では、ct_cancel は *type* パラメータを取り、3 つのタイプのキャンセル・オペレーションを行うことができます。

表 5-14 では、DB-Library と Client-Library のキャンセル・オペレーションを比較します。

表 5-14：DB-Library と Client-Library の比較 (結果のキャンセル)

DB-Library ルーチン	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
dbcancel(dbproc)	現在のコマンド・バッチをキャンセルして、コマンド・バッチによって生成された結果を廃棄する。	ct_cancel(connection, cmd, CS_CANCEL_ALL) *または* ct_cancel(connection, cmd, CS_CANCEL_ATTN)	現在のコマンドをキャンセルして、そのコマンドによって生成された結果をすべて廃棄する。 現在のコマンドをキャンセルして、アプリケーションが次にサーバから読み込みを行ったときに、結果をすべて廃棄する (内部のコールバック機能を使用)。
dbcanquery(dbproc)	最後に実行されたクエリから保留中のローを廃棄する。dbcancel は dbproc のすべてのコマンドをキャンセルし、dbcanquery は処理中のコマンドだけをキャンセルする。	ct_cancel(connection, cmd, CS_CANCEL_CURRENT)	現在の結果セットを廃棄する。

dbcancel と ct_cancel には、次のような大きな違いがあります。

- dbcancel は 1 つの DBPROCESS の現在のコマンド・バッチに影響を与えます。
- ct_cancel (CS_CANCEL_ALL または CS_CANCEL_ATTN) は、コマンド・レベルまたは接続レベルで呼び出すことができます。接続レベルで使用する場合、キャンセル・オペレーションはその接続内のすべてのコマンド構造体に適用されます。

CS_CANCEL_ATTN

結果を廃棄するには、Client-Library は結果ストリームから読み込まなければなりませんが、結果ストリームからの読み込みは必ずしも安全ではありません。CS_CANCEL_ATTN の場合、Client-Library は、アプリケーションがサーバから読み込もうとするまで待ってから、結果を廃棄します。

CS_CANCEL_ATTN は、コールバックまたは割り込みハンドラ内で使用してください。非同期モード・アプリケーションの場合は、非同期呼び出しの完了の保留中に、CS_CANCEL_ATTN を使用してください。

CS_CANCEL_ALL

すべてのメインライン・コードには、CS_CANCEL_ALL を使用してください。非同期モード・アプリケーションの場合は、非同期呼び出しの完了の保留中には、CS_CANCEL_ALL を使用しないでください。

CS_CANCEL_CURRENT

CS_CANCEL_CURRENT は、dbcanquery に直接マップします。CS_CANCEL_CURRENT は、CS_END_DATA を返すまで、ct_fetch の呼び出しとまったく同じです。

CS_CANCEL_CURRENT の機能は次のとおりです。

- 現在の結果セットを廃棄する。
- 結果項目とプログラム変数の間のすべてのバインドを解除する。
- 次の結果セットがある場合はそれを使用可能にし、現在のコマンドには影響を与えない。

注意 CS_CANCEL_ALL または CS_CANCEL_ATTN を使用すると、接続のオープン・カーソルは未定義のステータスに入ります。カーソルの open コマンドをキャンセルするよりも、カーソルをクローズすることをおすすめします。CS_CANCEL_CURRENT は、オープン・カーソルのある接続で使用して差し支えありません。

第 6 章

拡張機能

この章では、Client-Library の拡張機能について説明します。

この章の内容は、次のとおりです。

Client-Library の配列バインド

配列バインドとは、プログラム変数の配列に結果カラムをバインドすることです。カラム値に相当する複数ローが、フェッチ時に、1 回の ct_fetch 呼び出しで変数の配列にコピーされます。

配列バインドの使い方

アプリケーションは、ct_bind を呼び出すときに、CS_DATAFMT 構造体パラメータの *count* フィールドを 1 より大きい値に設定することによって、配列バインドを示します。

count は結果セット内のすべてのカラムで同じでなければなりません (例外 : 0 と 1 の *count* 値は等価であると見なされます。どちらの値の場合も、ct_fetch は 1 つのローをフェッチします)。

配列バインドは通常のローとカーソル・ローの結果セットだけに有用です。このタイプの結果セットだけが複数ローを持つことができるからです。

配列バインドの例

ex04ct.c 移行サンプル・プログラムで、配列バインドの例を示します。*ex04ct.c* は DB-Library の *example4.c* を Client-Library に変換したプログラムであり、DB-Library ロー・バッファリング・コードを Client-Library 配列バインド・コードに変換する例を示しています。*ex04ct.c* は実際には *ctrowbuf.c* 移行サンプル・プログラム内でルーチンを呼び出して、配列バインドを実行しています。*ctrowbuf.c* は簡単な配列バインド・ユーティリティ・ライブラリです。サンプル・プログラムは次のディレクトリに格納されています。

- UNIX では *$SYBASE/$SYBASE_OCS/sample/dblibrary*
- Microsoft Windows では *%SYBASE% ¥%SYBASE_OCS% ¥sample ¥dblib*

詳細については、使用しているプラットフォームの『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

Client-Library カーソル

アプリケーションは、次のタイプの DB-Library 機能の代わりに、Client-Library カーソルを使用できます。

- DB-Library カーソル
- DB-Library ブラウズ・モード

DB-Library カーソルと Client-Library カーソルの比較

DB-Library はクライアント側のカーソルをサポートしますが、Client-Library はサーバ側のカーソルをサポートします。

- クライアント側カーソルは Adaptive Server Enterprise カーソルに対応していません。このため、DB-Library はローを内部でバッファして、キーセットの管理、ローの位置付け、同時制御など、カーソルを管理するのに必要なすべての操作を行います。
- サーバ側カーソルは「ネイティブ・カーソル」とも呼ばれる、実際の Adaptive Server Enterprise カーソルです。アプリケーションがサーバ側カーソルを宣言し、オープンし、操作するためのインタフェースを提供するのは Client-Library ですが、実際にカーソルを管理するのは Adaptive Server Enterprise です。

表 6-1 は、DB-Library カーソルと Client-Library カーソルの主な違いを示します。

表 6-1 : DB-Library カーソルと Client-Library カーソルの違い

DB-Library カーソル	Client-Library カーソル
カーソル・ローの位置はクライアントによって定義される。	カーソル・ローの位置はサーバによって定義される。
オプティミスティック同時制御を定義できる。	オプティミスティック同時制御を定義できない。
逆方向のフェッチが可能 (dbcursoropen 呼び出しの *scrollopt* が CUR_KEYSET または CUR_DYNAMIC である場合)。	スクロール可能カーソルでは、次のどの fetch 方向でもデータのフェッチが可能。 • ABSOLUTE • RELATIVE • FIRST • LAST • PREVIOUS
メモリ要件は、dbcursoropen 時に指定されたフェッチ・バッファのサイズによって異なる。	メモリ要件は、カーソル・ロー設定と、カーソルのオープン中にアプリケーションが接続の新しいコマンドを送信するかどうかによって異なる。
Open Server アプリケーションにアクセスするには、そのアプリケーションが必要な DB-Library ストアド・プロシージャをインストールしていなければならない。	カーソルをサポートするようにコーディングされている Open Server アプリケーションにアクセスできる。
パフォーマンスが低い。	パフォーマンスが高い。
1 つの DBPROCESS で複数のカーソルを使用できる。	1 つの CS_CONNECTION で複数のカーソルを使用できる。 1 つの CS_COMMAND 構造体では 1 つのカーソルしか使用できない。

カーソル結果の処理規則

一般に Client-Library アプリケーションは、サーバにコマンドを送信するとき、ct_results が CS_END_RESULTS、CS_CANCELED、または CS_FAIL を返すまでは、同じ接続で別のコマンドを送信することはできません。

この規則に対する唯一の例外は、ct_results がカーソル結果を返す場合です。この場合には、アプリケーションは次のことが可能です。

- カーソル結果を処理しているのと同じコマンド構造体のカーソル・コマンドを送信できます。通常、アプリケーションはこの方法を使用して、カーソルの更新と削除を行います。
- 他のコマンド構造体上の、関連のないコマンドを送信できます。

カーソル・ルーチンの比較

表 6-2 では、DB-Library カーソル・ルーチンと Client-Library カーソル・ルーチンを比較します。これらのルーチン呼び出しの詳細については、次の項を参照してください。

- 『Open Client Client-Library/C プログラマーズ・ガイド』の「第 7 章 Client-Library カーソルの使い方」
- 『Open Client DB-Library/C リファレンス・マニュアル』の「付録 A カーソル」

表 6-2：DB-Library と Client-Library の比較 (カーソル・コマンド)

DB-Library の同等の機能	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
dbcursoropen(dbproc, stmt, scrollopt, concuropt, nrows, pstatus)	カーソルをオープンし、カーソル、スクロール・オプション、同時実行オプション、フェッチ・バッファのローの数、およびロー・ステータス・インジケータの配列へのポインタを定義する SQL 文を指定する。	ct_cursor(cmd, CS_CURSOR_DECLARE, name, namelen, text, textlen, option)	カーソルを宣言するコマンドを開始し、カーソルの本体である SQL テキストを指定する。 *option* は CS_UNUSED、または次の値のビットごとの論理和。 • CS_MORE • CS_END • CS_FOR_UPDATE • CS_READ_ONLY • CS_UNUSED • CS_IMPLICIT_CURSORS • CS_SCROLL_INSENSITIVE • CS_SCROLL_SEMISENSITIVE • CS_SCROLL_CURSOR • CS_NOSCROLL_INSENSITIVE

DB-Library の同等の機能	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
		ct_cursor(cmd, CS_CURSOR_ROWS, NULL, CS_UNUSED, NULL, CS_UNUSED, nrows)	Client-Library に返されるローの数を内部フェッチごとに指定する。デフォルトは 1 です。
		ct_cursor(cmd, CS_CURSOR_OPTION, NULL, CS_UNUSED, NULL, CS_UNUSED, option)	カーソル・オプション設定コマンドを開始する。 *option* の値は次のどちらか。 • CS_FOR_UPDATE • CS_READ_ONLY • CS_UNUSED • CS_SCROLL_INSENSITIVE • CS_SCROLL_SEMISENSITIVE • CS_SCROLL_CURSOR • CS_NOSCROLL_INSENSITIVE
		ct_cursor(cmd, CS_CURSOR_UPDATE, name, namelen, text, textlen, option)	カーソル更新コマンドを開始する。 *option* の値は次のどちらか。 • CS_UNUSED • CS_MORE • CS_END
		ct_cursor(cmd, CS_CURSOR_DELETE, name, namelen, NULL, CS_UNUSED, CS_UNUSED)	カーソルを削除するコマンドを開始する。
		ct_cursor(cmd, CS_CURSOR_DEALLOC, NULL, CS_UNUSED, NULL, CS_UNUSED, CS_UNUSED)	カーソルを割り付け解除するコマンドを開始する。
		ct_cursor(cmd, CS_CURSOR_OPEN, NULL, CS_UNUSED, NULL, CS_UNUSED, option)	コマンドを開始してカーソルをオープンする。 *option* の値は次のどちらか。 • CS_RESTORE_OPEN • CS_UNUSED

DB-Library の同等の機能	DB-Library の機能	Client-Library ルーチン	Client-Library の機能
(なし)		ct_send, ct_results	ct_cursor コマンドの結果を送信して処理する。 カーソルの宣言、カーソル・オプション、およびカーソル・ローのコマンドはバッチ処理ができ、1 つのコマンドとして送信できる。他の ct_cursor コマンドはバッチにはできない。
dbcursorbind (hc, col, vartype, varlen, poutlen, pvaraddr)	カーソル・カラムのバインド情報を登録する。	ct_bind(cmd, item, datafmt, buffer, copied, indicator)	カーソル結果をプログラム変数にバインドする。
dbcursorfetch (hc, fetchtype, rownum)	ローのブロックを、dbcursorbind の呼び出しで指定したプログラム変数にフェッチする。 DB-Library はスクロール可能カーソルをサポートしない。	ct_fetch(cmd, CS_UNUSED, CS_UNUSED, CS_UNUSED, rows_read)	カーソル結果データをフェッチする。
		ct_scroll_fetch(cmd, type, CS_UNUSED, CS_TRUE, rows_read)	結果セットからカーソルをフェッチする。 結果セット内をナビゲーションして、今後の処理用の単一ローを選択するためのブラウズ機能を提供する。
none		ct_keydata(cmd, action, colnum, buffer, buflen, outlen)	キー・カラムの内容の設定 (*action*=CS_SET) または取得 (*action*=CS_GET) を行う。
dbcursorclose (hc)	指定されたハンドル (*hc*) に対応しているカーソルをクローズする。 カーソル・ハンドルは再使用しない。	ct_cursor(cmd, CS_CURSOR_CLOSE, NULL, CS_UNUSED, NULL, CS_UNUSED, option)	カーソルをクローズする。 *option* は CS_DEALLOC または CS_UNUSED。 カーソルの割り付けを解除していない場合、同じコマンド構造体で ct_cursor を呼び出すことによって同じカーソルを後で再オープンできる。

DB-Library フェッチ・タイプと Client-Library カーソル

dbcursorfetch はさまざまなフェッチ・タイプをサポートします。表 6-3 は、dbcursorfetch のフェッチ・タイプと対応する Client-Library 機能を示します。

表 6-3 : dbcursorfetch フェッチ・タイプと対応する Client-Library 機能

dbcursor フェッチ・タイプ	Client-Library の同等の機能
FETCH_FORWARD	フェッチ方向 (またはフェッチ・タイプ) を CS_NEXT に設定した ct_fetch または ct_scroll_fetch。
FETCH_FIRST	フェッチ方向 (またはフェッチ・タイプ) を CS_FIRST に設定した ct_scroll_fetch。
FETCH_PREVIOUS	フェッチ方向 (またはフェッチ・タイプ) を CS_PREV に設定した ct_scroll_fetch。
FETCH_RANDOM	フェッチ方向 (またはフェッチ・タイプ) を CS_ABSOLUTE に設定した ct_scroll_fetch。
FETCH_RELATIVE	フェッチ方向 (またはフェッチ・タイプ) を CS_RELATIVE に設定した ct_scroll_fetch。
FETCH_LAST	フェッチ方向 (またはフェッチ・タイプ) を CS_LAST に設定した ct_scroll_fetch。

ct_keydata の使い方

配列バインドを使用してカーソル・ローを取得するアプリケーションには、一般に、ct_keydata が有用です。このルーチンに対する呼び出しで、Client-Library カーソルの更新または削除は、フェッチしたばかりの最新ロー以外のローに影響を与えるように再位置付けされます。

配列バインドを使用する場合、バインドされたカラム配列内の最後のローを除くローに対する更新は、ct_keydata を呼び出して再位置付けする必要があります。

DB-Library には、ct_keydata に直接対応する機能はありません。

Client-Library カーソルとブラウズ・モード更新の比較

Client-Library カーソルとブラウズ・モード更新の間には、次のような違いがあります。

- Client-Library カーソルには接続が 1 つだけ必要です。ブラウズ・モードには更新用に別の接続が必要です。この接続のために、追加のクライアントとサーバ・リソースを使用します。
- ブラウズ・モードにはタイムスタンプが必要ですが、Client-Library カーソルにはこれは必要ありません。
- 機密カーソルは、基本データ・テーブルを直接指すので、現在のカーソル・ローが入っているページを他のユーザが更新することはできません。非機密カーソルは、サーバ上のワーク・テーブルにあるデータのコピーを指します。

  基本テーブルにはロックは適用されないので、ブラウズ・モード更新は常に非機密です。Client-Library カーソルは、機密、非機密のどちらにもなります。

  非機密の Client-Library カーソルが更新可能であることがあります。この場合、基本データの同時更新は、「バージョン・キー」によって管理されます。カーソルを通して更新する場合、サーバは、値を比較して、クライアントがコピーを受信してからローが変更されたかどうかを判断します。

  一般に、“order by” 句で宣言された Client-Library カーソルは非機密です。

カーソルによる配列バインドの使い方

DB-Library ルーチン dbcursorbind は、カーソル結果カラムをプログラム変数の配列にバインドします。配列には、アプリケーションの dbcursoropen 呼び出しに指定したフェッチ・バッファのサイズに等しい数のローがあります。

Client-Library ルーチンの ct_bind は、カーソル結果カラムを 1 つのプログラム変数またはプログラム変数の配列のどちらかにバインドできます。配列のサイズは *datafmt->count* の値によって決定されます。

DB-Library も Client-Library も配列のサイズは結果セットのすべてのカラムについて同じでなければなりません。

更新可能な Client-Library カーソルで配列バインドを使用する場合には、次の注意事項が適用されます。

- Client-Library カーソルがオープンする前に、アプリケーションは ct_cmd_props を呼び出して、CS_HIDDEN_KEYS プロパティを可能にする必要があります。
- 結果配列内の中間ローの更新の前に、ct_keydata を呼び出して、中間ローのキー値で更新を位置付ける必要があります。更新がこのように位置付けられていない場合は、中間ローではなく、最後にフェッチされたローが更新の対象となります。

Client-Library カーソルの例

移行サンプル・プログラム *ex06ct.c* では、DB-Library ブラウズ・モード・コードを Client-Library カーソル・コードに変換する例を示します。*ex06ct.c* は *example6.c* DB-Library サンプル・プログラムを変換したプログラムです。*ex06ct.c* は簡単なテーブルを作成し、カーソルを使用して、テーブル・ローを移動しながら、各カラムを更新します。

ex06ct.c には、Client-Library カーソルが複数のコマンドを 1 接続でどのようにアクティブにするかを示すコード例も含まれています。

非同期プログラミング

非同期プログラミングでは、クライアント・アプリケーションは、サーバがコマンドを処理して結果を返すのを待つ間に他の作業を実行できます。

DB-Library の限定非同期サポート

すべてのプラットフォームでは、DB-Library は、dbrpcsend、dbsqlsend、dbpoll、dbsqlok 呼び出しを使用して、「非ブロック化読み込み」を限定的にサポートします。標準的な呼び出しシーケンスを次に示します。

- dbrpcsend または dbsqlsend － RPC または言語コマンドを送信して、ただちに返します。
- dbpoll － *return_reason* パラメータが DBRESULT に設定されるまでループで呼び出されます (Windows DB-Library 4.2 アプリケーションは dbpoll の代わりに dbdataready ルーチンを使用します)。

- dbsqlok ― コマンドから初期結果を取得します。

DB-Library では、コマンドの結果の初期読み込みだけが非同期です。アプリケーションは初期結果の到着をポーリングする必要があります。dbsqlok が呼び出されたときに初期結果が使用可能でなければ、dbsqlok はブロックされます。dbsqlok のあとの dbresults と dbnextrow に対する呼び出しは同期です。

Client-Library 非同期サポート

Client-Library では、ネットワークから読み書きするすべてのルーチンが非同期に動作します。これらのルーチンは次のとおりです。

- ct_connect、ct_close、ct_options
- ct_send、ct_cancel、ct_results、ct_fetch
- ct_get_data、ct_send_data
- ct_recvpassthru、ct_sendpassthru
- blk_init、blk_done
- blk_sendrow、blk_sendtxt
- blk_rowxfer、blk_textxfer

Client-Library には、完全非同期とポーリングの 2 つのモデルの非同期プログラミングが用意されています。

デフォルトでは、接続は同期モードで動作します。非同期プログラミング・モデルを要求する場合は、CS_NETIO プロパティを CS_ASYNC_IO (完全非同期動作の場合) または CS_DEFER_IO (ポーリング・モデルの場合) に設定してください。この設定は、コンテキスト・レベルで設定すると、あとで割り付けられるすべての接続に影響を与えます。接続ごとに個別にプロパティを設定することもできます。

完全非同期モデル

完全非同期モデルの場合、アプリケーションは完了コールバックをインストールし、Client-Library は、非同期ルーチンが完了するたびに、このコールバックを呼び出します。完全非同期モデルは、割り込み駆動型ネットワーク I/O 機能を持っているプラットフォームまたは Client-Library がオペレーティング・システム・スレッドを使用してネットワーク I/O を実行するプラットフォームでのみ、サポートされます。

ポーリング・モデル

ポーリング・モデルの場合、アプリケーションは、CS_PENDING を返す非同期ルーチンに対する各呼び出しのあと、ループ内で ct_poll を呼び出します。ポーリング・モデルはすべてのプラットフォームでサポートされます。移植性を懸念しているのであれば、非同期アプリケーションを作成するときにはポーリング・モデルを使用してください。

これらのプログラミング・モデ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』の「非同期プログラミング」を参照してください。

ct_poll の使い方

ct_poll は、非同期オペレーション完了とレジスタード・プロシージャ・ノーティフィケーションの接続をポーリングします。その機能は、dbpoll とほぼ同じです。

ct_poll と dbpoll の主な違いは次のとおりです。

- ct_poll は CS_CONTEXT パラメータか CS_CONNECTION パラメータのどちらかを取ることができますが、dbpoll は DBPROCESS パラメータを取ります。
- ct_poll はより多様な完了タイプ (*compid*) をサポートします。
- ct_poll では、完了したオペレーションの最終リターン・コードが使用可能ですが、dbpoll ではこれは使用できません。

これらの違いの詳細については、*表 6-4 : dbpoll と ct_poll の比較*を参照してください。

コールバック機能を使用できるプラットフォームの場合、ct_poll は、完了したオペレーションまたはノーティフィケーションを検出すると、適切なコールバック・ルーチンがインストールされていれば、それを自動的に呼び出します。

ct_poll には、次のような制限があります。

- CS_DISABLE_POLL プロパティが CS_TRUE に設定されている場合、ct_poll は非同期オペレーションの完了をチェックしません。

- CS_ASYNC_NOTIFS が CS_FALSE である場合、ct_poll はレジスタード・プロシージャ・ノーティフィケーションを探すのに、ネットワークから読み込みません。コマンド結果の読み込み中にすでに検出されているノーティフィケーションは報告されます。言い換えれば、CS_ASYNC_NOTIFS が CS_FALSE であるときに ct_poll がレジスタード・プロシージャ・ノーティフィケーションを報告するには、アプリケーションはアクティブにコマンドを送信し、結果を読み込んでいる必要があります。

表 6-4 : dbpoll と ct_poll の比較

dbpoll パラメータ	パラメータの説明	*ct_poll* パラメータ	パラメータの説明
dbproc	dbprocess 構造体へのポインタ。*dbproc* が NULL である場合、dbpoll はオープンされているすべての DBPROCESS 接続について応答の到着を確認する。	*context* *connection* (*context* か *connection* のどちらかは NULL にする)	CS_CONTEXT 構造体と CS_CONNECTION 構造体へのポインタ。*context* が NULL の場合、ct_poll は接続を 1 つだけ確認する。*connection* が NULL である場合、ct_poll はそのコンテキスト内でオープンされているすべての接続を確認する。
milliseconds	dbpoll が返す前に保留オペレーションの完了を待つ時間 (ミリ秒単位)。*milliseconds* が 0 である場合、dbpoll はただちに返す。*milliseconds* が -1 である場合、dbpoll は、サーバ応答が到着するかシステム割り込みが発生するまで、返さない。	*milliseconds*	ct_poll が返す前に保留オペレーションの完了を待つ時間 (ミリ秒単位)。*milliseconds* が 0 である場合、ct_poll はただちに返す。*milliseconds* が CS_NO_LIMIT である場合、ct_poll は、サーバ応答が到着するかシステム割り込みが発生するまで、返さない。

dbpoll パラメータ	パラメータの説明	*ct_poll* パラメータ	パラメータの説明
ready_dbproc	DBPROCESS 構造体へのポインタ。dbpoll はこの値を、サーバ応答が到着している DBPROCESS、または応答が到着していない場合には NULL に設定する。	*compconn*	*compconn* はポインタ変数のアドレス。*connection* が NULL である場合、すべての接続がポーリングされ、ct_poll は *compconn* を、最初に検出した完了オペレーションを所有している CS_CONNECTION 構造体を指すように設定する。オペレーションが完了していない場合、または *connection* が NULL でない場合、ct_poll は *compconn* を NULL に設定する。
		compcmd	*compcmd* はポインタ変数のアドレス。ct_poll は *compcmd* を、最初に検出した完了オペレーションを所有している CS_COMMAND 構造体を指すように設定する。オペレーションが完了していない場合、ct_poll は *compcmd* を NULL に設定する。
return_reason	dbpoll が返した理由を示す記号値へのポインタ。	*compid*	どのルーチンが完了したかを示す記号値 (CS_SEND、CT_FETCH) へのポインタ。
(なし)		*compstatus*	CS_RETCODE タイプの変数へのポインタ。ct_poll はこのポインタを、「完了ステータス」と呼ばれる完了したオペレーションの最後のリターン・コードを示すように設定する。完了ステータスは、CS_PENDING を除いて、そのルーチンにリストされているリターン・コードのいずれか。

ct_wakeup の使い方

ct_wakeup は、アプリケーションによって呼び出されると、接続の完了コールバックを呼び出します。この ct_wakeup ルーチンは、Client-Library の上位に実装される、高レベルの非同期レイヤを提供するアプリケーションに有用です。『Open Client Client-Library/C リファレンス・マニュアル』の「非同期プログラミング」を参照してください。

バルク・コピー・インタフェース

Bulk-Library は、Client-Library と Server-Library のバルク・コピー・ルーチンからなる API です。Bulk-Library ルーチンの中には、Client-Library または Server-Library のどちらかに固有なルーチンと、どちらにも共通のルーチンがあります。

Bulk-Library ルーチンの名前には、“blk” というプレフィクスが付き、CT-Library バルク・コピー・ルーチンの名前には、“bcp” というプレフィクスが付きます。

CT-Library バルク・コピーと Bulk-Library の大きな違いは、ファイル I/O サポート機能が CT-Library だけに組み込まれている点です。

Adaptive Server Enterprise が暗号化カラムをサポートする場合は、CT-Library バルク・コピーと Bulk-Library はどちらも暗号化カラムをサポートします。

『Open Client/Server Common Libraries リファレンス・マニュアル』を参照してください。

Bulk-Library の初期化と終了処理

Bulk-Library オペレーションには、CS_BLKDESC 構造体が必要です。アプリケーションは、blk_alloc を呼び出して、CS_BLKDESC を割り付けることができます。バルク・オペレーションが完了すると、アプリケーションは blk_drop を呼び出して、その CS_BLKDESC を削除できます。

blk_init はバルク・コピー・オペレーションを開始します。

Bulk-Library ルーチンの blk_init には、CT-Library の bcp_init パラメータに相当する、構造体、テーブル名、方向値のパラメータがあります。ただし、blk_init はホスト・ファイル名パラメータまたはエラー・ファイル名パラメータを処理しません。

転送ルーチン

Bulk-Library アプリケーションは、Client-Library の ct_bind、ct_recvpassthru、および ct_sendpassthru ルーチンに類似したルーチンを使用して、データを転送します。

Bulk-Library アプリケーションも Client-Library アプリケーションも、CS_DATAFMT 構造体を使用して、バインドするプログラム変数を記述し、両者とも配列バインドをサポートします。

blk_describe は CS_DATAFMT 構造体のフィールドを設定します。アプリケーションは、カラムをプログラム変数にバインドする blk_bind 呼び出しの中でこの CS_DATAFMT 構造体を使用できます。

CT-Library の bcp_bind パラメータの中には CS_DATAFMT 構造体のフィールドにマップするものもありますが、それ以外のパラメータに相当するものはありません。特に、bcp_bind の長さプレフィクス、ターミネータ、ターミネータ長パラメータは、Bulk-Library には対応するフィールドがありません。アプリケーションは、blk_bind の *datalen* パラメータを使用して、プログラム変数からコピーするバイト数を指定したり、プログラム変数に書き込むバイト数を決定したりします。

DB-Library バルク・コピーとのその他の違い

カラムのデフォルト値を取得する blk_default は Client-Library 固有の機能です。

Bulk-Library には、次の CT-Library ルーチンに相当する機能はありません。これらはホスト・ファイルまたはフォーマット・ファイルをサポートする機能です。

- bcp_colfmt、bcp_colfmt_ps、bcp_columns
- bcp_exec
- bcp_readfmt、bcp_writefmt

テキスト／イメージ・インタフェース

ここでは、Client-Library と DB-Library のテキスト／イメージ・インタフェースを比較します。

text または *image* データの検索

一般に、Client-Library アプリケーションは、結果セットのローを処理しているフェッチ・ループの中で ct_get_data を呼び出して、大きな text または image 値を取得します。

ct_get_data は、dbreadtext と類似した機能を持っていますが、これよりも処理性が高く、柔軟性もあります。これには次のような特性があります。

- 変換を行わずに、サーバから送信されたとおりにデータを取得します。
- ストアド・プロシージャのリターン・パラメータとリターン・ステータス値だけでなく、通常カラム、計算カラムからのデータを取得するのに使用できます (「ct_get_data と dbdata の比較」(70 ページ) を参照)。
- 任意のデータ型の複数カラムを取得するのに使用できます ((dbreadtext は、1 つの text または image カラムを返す Transact-SQL クエリだけに使用されます)。
- 大きな text または image 値の取得に最もよく使用されます。

ct_get_data の使用には、次の制限が適用されます。

- ct_bind と ct_get_data の両方を使用する場合、ct_get_data を使用して最初に取得したカラムを、ct_bind で最後にバインドしたカラムの後ろに入れる必要があります。

  たとえば、アプリケーションが 4 つのカラムを選択して、最初と 3 番目のカラムをプログラム変数にバインドすると、アプリケーションは 2 番目のカラムに入っているデータを ct_get_data を使用して取得することはできません。ただし、ct_get_data を使用して、4 番目のカラム内のデータを取得することはできます。

  この制限があるために、ct_get_data で取得する text または image カラムは必ず選択リストの末尾に置くようにします。
- 配列バインドがさらに前の ct_bind 呼び出しに示されている場合、アプリケーションは結果セット内のどのカラムにも ct_get_data を使用できません。

DB-Library のテキスト・タイムスタンプ

DB-Library では、text カラムの select は、テキスト・タイムスタンプ値を現在のローから DBPROCESS 構造体にコピーします。DB-Library アプリケーションはこのテキスト・タイムスタンプ値を dbtxtimestamp で取得できます。

Client-Library では、カラムのテキスト・タイムスタンプを保管するのに CS_IODESC 構造体を使用します。

Client-Library の CS_IODESC 構造体

CS_IODESC 構造体には、text または image データを記述します。

更新するカラムから text または image データを検索する場合、Client-Library アプリケーションは ct_data_info を呼び出して、text または image カラムを記述する CS_IODESC 構造体を入手します。

アプリケーションは通常、ct_get_info を呼び出す前に、検索するカラムのために ct_get_data を呼び出す必要があります。ただし、ct_get_data を Server-Library API srv_send_data とともに使用する場合、ゲートウェイ Open Server アプリケーションで text、image、XML カラムをまとまりとして転送するには、ct_get_data を呼び出す前に ct_data_info を呼び出す必要があります。

カラムのデータを取得する必要がない場合には、ct_get_data の *buflen* に 0 を割り当てます。この方法は、取得前に text または image 値の長さを決定する際に便利です。

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

カラムを更新するときに、アプリケーションは ct_data_info を再度呼び出して、更新オペレーションに CS_IODESC フィールドを適用します。

DB-Library には、カラムまたは値のテキスト・タイムスタンプを操作する特別なルーチンがあります。Client-Library では、アプリケーションは ct_data_info を呼び出し、結果の CS_IODESC 構造体を直接変更して、これらのタスクを処理します。

一般的なアプリケーションは、CS_IODESC 構造体を更新オペレーションで使用する前に、この構造体内の次の 3 つのフィールドだけを変更します。

- *total_txtlen*

  このフィールドは、新しい値の合計の長さをバイト単位で指定します。このフィールドは dbwritetext. の *size* パラメータと同等です。

- *log_on_update*

  このフィールドは、サーバが更新をログに記録するかどうかを指定します。このフィールドは dbwritetext の *log* パラメータと同等です。

- *locale*

  このフィールドは、この値のローカライゼーション情報が入っている CS_LOCALE 構造体がある場合は、それを指します。DB-Library には、これに相当する機能はありません。

CS_IODESC の *timestamp* フィールドでは、text または image カラムの最終変更時間がマーク付けされます。

表 6-5 では、DB-Library と Client-Library のテキスト・タイムスタンプ機能を比較します。

表 6-5 : DB-Library と Client-Library の比較 (テキスト・タイムスタンプ)

DB-Library ルーチン	DB-Library の機能	Client-Library の同等の機能
dbtxtimestamp(dbproc, column)	現在のローのカラムに対するテキスト・タイムスタンプの値を返す。	現在のロー内の I/O 記述子を取得し、それを CS_IODESC に置く。 ct_data_info(cmd, CS_GET, colnum, iodesc). テキスト・タイムスタンプは CS_IODESC –>timestamp 内にある。
dbtxptr(dbproc,column)	現在のローのカラムに対するテキスト・ポインタの値を返す。	テキスト・ポインタは CS_IODESC –>textptr 内にある。
dbtxtsnewval(dbproc)	dbwritetext 呼び出しのあと、テキスト・タイムスタンプの新しい値を返す。	ct_send_data 呼び出しのあと、テキスト・タイムスタンプの新しい値を含む、リターン・パラメータ結果セット (ct_results は CS_PARAM_RESULT の *result_type* で返す) を処理する。

text または *image* データの送信

単一のまとまりを更新する場合、ct_send_data の機能は dbwritetext と同じです。

複数のまとまりを更新する場合、ct_send_data の機能は dbwritetext と dbmoretext と同じです。

- DB-Library アプリケーションはまず text を null として dbwritetext を呼び出し、次に dbmoretext をループで呼び出してデータを送信します。
- Client-Library アプリケーションは ct_send_data をループで呼び出すだけでデータを送信します。

Client-Library アプリケーションは、一般に、更新オペレーションを行うときに、次のような呼び出しシーケンスを使用します。

1　ct_fetch を呼び出して、対象となるローをフェッチします。

2　ct_get_data を呼び出してカラムの値を取得し、そのカラムの I/O 記述子をリフレッシュします。

3　ct_data_info を呼び出して、I/O 記述子を CS_IODESC 構造体に取得します。

現在の I/O 記述子を使用して、次の手順で更新を行います。

1　*type* を CS_SEND_DATA_CMD に設定した ct_command を呼び出して、コマンドを起動します。

2　必要に応じて、*locale*、*total_txtlen*、または *log_on_update* を変えて、CS_IODESC を変更し、ct_data_info を呼び出して、そのカラム値の I/O 記述子を設定します。

3　ct_send_data をループで呼び出して、値全体を書き込みます。

4　ct_send を呼び出して、そのコマンドを送信します。ct_send_data はデータをバッファするので、ct_send で、すべてのデータをサーバにフラッシュするようにします。

5　ct_results を呼び出して、コマンドの結果を処理します。text または image 値を更新すると、単一パラメータが入っているパラメータ結果セットが生成されます。これがこの値の新しいテキスト・タイムスタンプです。カラムを再度更新する場合、アプリケーションは、ct_data_info を呼び出して次の更新の I/O 記述子を設定する前に、新しいタイムスタンプを保存して、その値を CS_IODESC にコピーする必要があります。

更新オペレーション

更新オペレーションでは、Open Client アプリケーションによって取得されたテキスト・タイムスタンプ値が、データベースのテキスト・タイムスタンプ値と比較されます。これによって、競合するアプリケーションが互いの変更に影響を及ぼさないようにします。DB-Library ルーチンの dbwritetext は、null *timestamp* ポインタで呼び出すことができます。この結果、更新はデータベース・テキスト・タイムスタンプ値に関係なく行われます。

Client-Library ルーチンの ct_send_data は、CS_IODESC の *timestamp* が現在のデータベース・テキスト・タイムスタンプと一致しない場合には、必ずエラーになります。

表 6-6 では、DB-Library と Client-Library のテキスト更新機能を比較します。

表 6-6：テキスト更新機能の比較

DB-Library ルーチン (パラメータ)	DB-Library の機能	Client-Library の同等の機能
dbwritetext(objname)	ピリオドで区切られた、対象のテーブルとカラムの名前 (たとえば *table.column*)。	CD_IODESC–>name ct_data_info によって設定。
dbwritetext (textptr)	変更される text または image 値のテキスト・ポインタを指すポインタ。	CS_IODESC–>textptr ct_data_info によって設定。
dbwritetext(textptrlen)	dbwritetext では DBTXPLEN。	CS_IODESC–>textptrlen ct_data_info によって設定。
dbwritetext(timestamp)	変更される text または image 値のタイムスタンプを指すポインタ。	CS_IODESC–>timestamp ct_data_info によって設定、またはカラムを更新したあとにパラメータ結果として取得。
dbwritetext(log)	サーバがこの text または image の変更をログに記録するかどうかを示すブール値。	CS_IODESC–>log_on_update アプリケーションによって設定。
dbwritetext(size)	送信する値の合計サイズ (バイト単位)。	CS_IODESC–>total_txtlen アプリケーションによって設定。
dbmoretext(size)	送信されている値のこの部分のサイズ (バイト単位)。	ct_send_data(buflen)
dbmoretext(text)	データの書き込まれる部分を指すポインタ。	ct_send_data(buffer)

text および image の例

次の移行サンプル・プログラムは、DB-Library の text と image コードの変換例を示します。

- *ex09ct.c* － DB-Library の *example9.c* を Client-Library に変換したプログラムです。1 回の dbwritetext 呼び出しで text/image カラムを更新するコードの変換例です。

- *ex10ct.c* － DB-Library の *example10.c* を Client-Library に変換したプログラムです。dbwritetext と dbmoretext を使用して、大きな text/image カラムをまとまりで更新するコードの変換例です。
- *ex11ct.c* － DB-Library の *example11.c* を Client-Library に変換したプログラムです。大きな text/image カラムを取得してオペレーティング・システム・ファイルに保存するコードの変換例です。

サンプル・プログラムは次のディレクトリに格納されています。

- UNIX

  では *$SYBASE/$SYBASE_OCS/sample/dblibrary*

- Microsoft Windows では *%SYBASE% ¥%SYBASE_OCS% ¥sample ¥dblib*

詳細については、使用しているプラットフォームの『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

ローカライゼーション

アプリケーションのローカライゼーションでは、次のことを決定します。

- Client-Library と Adaptive Server Enterprise のメッセージの言語
- 日時値のフォーマット
- 文字列を変換および比較するときに使用する文字セットとソート順

Client-Library は、ほとんどのプラットフォームで、環境変数を使用して、アプリケーションが使用するデフォルト・ローカライゼーション値を決定します。

ロケール・ファイル *locales.dat* は、言語、文字セット名、およびソート順をロケール名に関連付けます。Open Client/Server 製品は、ローカライゼーション情報をロードするときにこのロケール・ファイルを使用します。ロケール・ファイル内のエントリは、アプリケーションの稼働条件に従って、追加または変更できます。

ある環境のデフォルト・ローカライゼーション値がアプリケーションの稼働条件を満たしている場合、ローカライゼーションは必要ありません。デフォルト値がアプリケーションの稼働条件を満たしていない場合には、CS_LOCALE 構造体を使用して、カスタム・ローカライゼーション値を設定できます。アプリケーションは、コンテキスト、接続、データ要素の各レベルでローカライゼーション値を設定できます。

CS_LOCALE 構造体

Client-Library アプリケーションは、CS_LOCALE 構造体を使用して、カスタム・ローカライゼーション値を設定できます。これを行うために、アプリケーションは次のような処理を行います。

1　cs_loc_alloc で CS_LOCALE 構造体を割り付けます。

2　cs_locale を呼び出して、CS_LOCALE 構造体にローカライゼーション値をロードします。

3　希望するレベルでロケールを設定します。アプリケーションは次のことを実行できます。

- cs_config でローカライゼーション値をコンテキスト構造体にコピーする。
- 接続をオープンする前に、ct_con_props でローカライゼーション値を接続構造体にコピーする。
- カスタム・ローカライゼーション値を受け付けるルーチンにパラメータとして CS_LOCALE 構造体を渡す (cs_convert、cs_time)。
- 送信先プログラム変数を記述している CS_DATAFMT 構造体に CS_LOCALE 構造体へのポインタを含める (cs_convert、ct_bind)。

ローカライゼーション優先度

どのローカライゼーション値を使用するかを決定するとき、Client-Library は次の優先度に従います。

1　データ要素ローカライゼーション値

- データ要素を記述している CS_DATAFMT 構造体に対応している CS_LOCALE
- パラメータとしてルーチンに渡される CS_LOCALE

2　接続構造体ローカライゼーション値

3　コンテキスト構造体ローカライゼーション値

コンテキスト構造体ローカライゼーション値は常に定義されています。これは、新しく割り付けられたコンテキスト構造体は、その時点で有効なデフォルト・ローカライゼーション値を割り当てられるためです。

付 録　A

DB-Library ルーチンの Client-Library ルーチンへのマップ

この章では、DB-Library ルーチンをリストし、それぞれに対応する Client-Library 呼び出しと CS-Library 呼び出しを示します。

DB-Library ルーチンの Client-Library ルーチンへのマップ

表 A-1 は、DB-Library ルーチンとそれに対応する同等の Client-Library と CS-Library のルーチンを示します。

表 A-1：DB-Library ルーチンの Client-Library ルーチンへのマップ

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
db12hour	指定された言語が、12 時間制と 24 時間制のどちらを使用するかを決定する。	cs_dt_info(CS_12HOUR)
dbadata	計算カラムのデータを指すポインタを返す。	直接の同等機能はない。アプリケーションはバインドするか、または ct_get_data を使用して、データ値を取得する。「データ値の検索」(67 ページ) を参照。
dbadlen	計算カラムのデータの実際の長さを返す。	直接の同等機能はない。 • データの最大長 (CS_DATAFMT の *maxlength* フィールド) を決定するには、ct_describe を使用する。 • バインドされた変数に置かれるデータ値の長さを決定するには、ct_bind *copied* パラメータを使用する。 • ct_get_data で取得されるデータ値の長さを決定するには、ct_get_data *outlen* パラメータを使用する。
dbaltbind	計算カラムをプログラム変数にバインドする。	ct_bind

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbaltbind_ps	numeric または decimal データの精度および位取りをサポートして、計算カラムをプログラム変数にバインドする。	ct_bind
dbaltcolid	計算カラムのカラム ID を返す。	ct_compute_info(CS_COMP_COLID)
dbaltlen	特定の計算カラムのデータの最大長を返す。	ct_describe (CS_DATAFMT の *maxlength* フィールド)
dbaltop	特定の計算カラムの集合演算子のタイプを返す。	ct_compute_info(CS_COMP_OP)
dbalttype	計算カラムのデータ型を返す。	ct_describe (CS_DATAFMT の *datatype* フィールド)
dbaltutype	計算カラムのユーザ定義データ型を返す。	ct_describe (CS_DATAFMT の *usertype* フィールド)
dbanullbind	インジケータ変数を計算ロー・カラムに関連付ける。	ct_bind
dbbind	通常の結果カラムをプログラム変数にバインドする。	ct_bind
dbbind_ps	numeric または decimal データの精度および位取りをサポートして、通常の結果カラムをプログラム変数にバインドする。	ct_bind
dbbufsize	DBPROCESS ロー・バッファのサイズを返す。	なし。Client-Library はロー・バッファリングの組み込みサポートを提供しない。
dbbylist	計算ローの bylist を返す。	次の呼び出しシーケンスと置き換える。 • bylist の長さを決定するには、ct_compute_info(CS_BYLIST_LEN)。 • bylist を保持するには、CS_SMALLINT 配列を割り付ける (または既存の配列が十分であることを確認する)。 • bylist を配列にコピーするには、ct_compute_info(CS_COMP_BYLIST)。
dbcancel	現在のコマンド・バッチを取り消す。	次のいずれか 1 つ。 • メインライン・コードからの ct_cancel(CS_CANCEL_ALL)(CS_CANCEL_ALL)、または • クライアント・メッセージ・ハンドラからの ct_cancel(CS_CANCEL_ATTN)
dbcanquery	最後に実行されたクエリから保留中のローを取り消す。	ct_cancel(CS_CANCEL_CURRENT)

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbchange	コマンド・バッチが現在のデータベースを変更したかどうかを調べる。	なし。 この機能を必要とするアプリケーションは、サーバのメッセージ・ハンドラでサーバ・メッセージ番号 5701 をトラップするようにコードできる。メッセージ 5701 のテキストにはデータベース名が含まれている。
dbcharsetconv	サーバが文字セット変換を実行しているかどうかを示す。	ct_con_props(CS_CHARSETCNV)
dbclose	単一の DBPROCESS 構造体をクローズして、割り付けを解除する。	次のいずれか 1 つ。 • 接続をクローズするには ct_close • 構造体の割り付けを解除するには ct_con_drop
dbclrbuf	ロー・バッファからローを削除する。	なし。Client-Library はロー・バッファリングの組み込みサポートを提供しない。
dbclropt	dbsetopt によって設定されたオプションをクリアする。	ct_options(CS_CLEAR)
dbcmd	テキストを DBPROCESS 言語コマンド・バッファに追加する。	ct_command(CS_LANG_CMD) がテキストを言語バッファに入れる。 言語バッファにさらにテキストを追加する場合、*option* を CS_MORE として渡す。そうでない場合は CS_END として渡す。
DBCMDROW	現在のコマンドがローを返せるかどうかを調べる。	直接の同等機能はない。ct_results は *result_type* を CS_CMD_SUCCEED に設定し、データを返さないコマンドの正常な終了を示す。 ct_results *result_type* 値の DB-Library プログラム論理との比較については、「結果を処理するコード」(61 ページ) を参照。
dbcolbrowse	通常の結果カラムのソースが、ブラウズ・モードの更新によって更新可能かどうかを決定する。	ct_br_column (CS_BROWSEDESC の *isbrowse* フィールド)
dbcollen	通常の結果カラム内のデータの最大長を返す。	ct_describe (CS_DATAFMT の *maxlength* フィールド)
dbcolname	通常の結果カラムの名前を返す。	ct_describe (CS_DATAFMT の *name* フィールド)
dbcolsource	指定された通常の結果カラムの導出元であるデータベース・カラムの名前を指すポインタを返す。	ct_br_column (CS_BROWSEDESC の *orignam*e フィールド)

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbcoltype	通常の結果カラムのデータ型を返す。	ct_describe (CS_DATAFMT の *datatype* フィールド)
dbcoltypeinfo	numeric カラム値の精度と位取りの値が格納されている構造体を返す。	ct_describe (CS_DATAFMT の *precision* と *scale* フィールド)
dbcolutype	通常の結果カラムのユーザ定義データ型を返す。	ct_describe (CS_DATAFMT の *usertype* フィールド)
dbconvert	データを別のデータ型に変換する。	cs_convert
dbconvert_ps	numeric または decimal データの精度および位取りをサポートして、データを別のデータ型に変換する。	cs_convert
DBCOUNT	Transact-SQL コマンドによる影響を受けたローの数を返す。	ct_res_info(CS_ROW_COUNT) ct_results が CS_CMD_DONE の *result_type* 値を返すときに呼び出す。 **注意** ストアド・プロシージャの実行後、DBCOUNT と ct_res_info で返されるローの数が異なる場合がある。詳細については、「影響を受けたロー数の取得」(73 ページ) を参照。
DBCURCMD	現在のコマンドの番号を返す。	ct_res_info(CS_CMD_NUMBER)

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
DBCURROW	現在読み込み中のローの番号を返す。	直接の同等機能はない。 アプリケーションは、通常の結果ローと計算結果ローをフェッチすると増加されるカウンタ変数を使用できる。DBCURROW と同等にカウントを維持するには、次の手順を実行する。 • ct_results が *result_type* パラメータを CS_ROW_RESULT または CS_COMPUTE_RESULT に設定するときに、CS_SUCCEED または CS_ROW_FAIL を返す ct_fetch 呼び出しが行われるたびにカウントを増分する。配列バインドを使用する場合は、ct_fetch *rows_read* パラメータで返される値の分だけ増加する。それ以外は、1 増加する。 • ct_results のループの前にカウンタを 0 に設定し、ct_results が CS_CMD_DONE *result_type* 値を返すたびに 0 にリセットする。
dbcursor	フェッチ・バッファ内の特定のローを挿入、更新、削除、ロック、またはリフレッシュする。	ct_cursor ct_cursor コマンドは ct_send を使って送信され、その結果は ct_results で処理される必要がある。 **注意**　DB-Library カーソルと ct_cursor カーソルの機能セットは同じではない。「Client-Library カーソル」(78 ページ) を参照。
dbcursorbind	カーソル・カラムのバインド情報を登録する。	ct_results が CS_CURSOR_RESULT の *result_type* で返す場合は、ct_bind。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbcursorclose	指定のハンドルに対応しているカーソルをクローズして、そのカーソルに属するすべてのデータを解放する。	• ct_cursor(CS_CURSOR_CLOSE) はカーソルをクローズするコマンドを開始する。 • ct_cursor(CS_CURSOR_DEALLOC) はカーソルに関連するサーバ・リソースの割り付けを解除するコマンドを開始する。 *option* をカーソル・クローズ・コマンドを開始する ct_cursor 呼び出し内の CS_DEALLOC として渡すことで、1 つのコマンドでカーソルをクローズして割り付け解除できる。 ct_cursor コマンドは ct_send を使って送信され、その結果は ct_results で処理される必要がある。
dbcursorcolinfo	オープン・カーソル内の指定カラム番号のカラム情報を返す。	ct_results が CS_CURSOR_RESULT の *result_type* で返す場合、ct_describe 。
dbcursorfetch	ローのブロックを、dbcursorbind でユーザが宣言したプログラム変数にフェッチする。	ct_results が CS_CURSOR_RESULT の *result_type* で返す場合、ct_fetch 。
dbcursorinfo	キーセットが結果セットの末尾になった場合、キーセット内のカラム数およびロー数を返す。	直接の同等機能はない。Client-Library カーソルはサーバによって管理され、キーセットに相当する概念はない。 カーソル結果セット・カラムがキーかどうかを明らかにするには、ct_describe を呼び出して CS_DATAFMT 構造体の *status* フィールドを確認する。
dbcursoropen	スクロール・オプション、同時実行オプション、およびフェッチ・バッファのサイズ (1 回のフェッチで取得されるローの数) を指定して、カーソルをオープンする。	ct_cursor **注意** DB-Library カーソルと ct_cursor カーソルの機能セットは同じではない。「Client-Library カーソル」(78 ページ) を参照。
dbdata	通常の結果カラム内のデータを指すポインタを返す。	直接の同等機能はない。 アプリケーションはバインドするか、または ct_get_data を使用して、データ値を取得する。「ct_get_data と dbdata の比較」(70 ページ) を参照。
dbdate4cmp	2 つの DBDATETIME4 値を比較する。	cs_cmp

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbdate4zero	DBDATETIME4 変数を 1900 年 1 月 1 日午前 12 に初期化する。	直接の同等機能はない。 アプリケーションは、cs_convert を呼び出して、文字列の表記を同等の CS_DATETIME 値に変換する。 CS_DATETIME4 構造体のバイトを 0 にするには、アプリケーションは memset (または同等のプラットフォーム固有の機能) を使用できる。これは日付値を 1900 年 1 月 1 日午前 12 時 00 分に効果的に設定する。 memset はより優れたパフォーマンスを提供する。
dbdatechar	DBDATETIME 値の要素を表す整数を文字フォーマットに変換する。	直接の同等機能はない。 ネイティブ言語での月日の名前を取得する dbdatechar 呼び出しを置き換えるには、cs_dt_info を使用する。 他の dbdatechar 呼び出しは、整数を 10 進数の文字列に変換だけを行う。これらは sprintf (または同等の変換ルーチン) の呼び出しで置き換えられる。
dbdatecmp	2 つの DBDATETIME 値を比較する。	cs_cmp
dbdatecrack	マシンが読み込み可能な DBDATETIME 値をユーザがアクセス可能なフォーマットに変換する。	cs_dt_crack DBDATEREC 構造体と CS_DATEREC 構造体は同一。
dbdatename	DBDATETIME 構造体の指定コンポーネントを対応する文字列に変換する。	直接の同等機能はない。 ネイティブ言語での月日の名前を取得する dbdatename 呼び出しを置き換えるには、cs_dt_crack と cs_dt_info を使用する。他の呼び出しは、次の呼び出しシーケンスで置き換えられる。 • 日付を CS_DATEREC 構造体に拡張するには、cs_dt_crack を呼び出す。 • CS_DATEREC フィールドで簡単な計算を実行する。 • 結果を文字列に変換するには sprintf (または同等の変換ルーチン) を呼び出す。
dbdateorder	指定した言語の日付コンポーネントの順序を返す。	cs_dt_info(CS_DATEORDER)

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbdatepart	DBDATETIME 値の指定部分を整数値で返す。	直接の同等機能はない。 dbdatepart 呼び出しは cs_dt_crack の呼び出しと適切な CS_DATEREC フィールドの参照で置き換えられる。DBDATE_QQ と DBDATE_WK を計算する呼び出しを置き換えるには、アプリケーションは適切な CS_DATEREC フィールドで簡単な演算を行う必要がある。
dbdatezero	DBDATETIME 値を 1900 年 1 月 1 日午前 12:00:00:000 に初期化する。	直接の同等機能はない。アプリケーションは、cs_convert を呼び出して、文字列の表記を同等の CS_DATETIME 値に変換する。 CS_DATETIME 構造体のバイトを 0 にするには、アプリケーションは memset (または同等のプラットフォーム固有の機能) を使用できる。これは日付値を 1900 年 1 月 1 日午前 12:00:00:000 に効果的に設定する。 memset はより優れたパフォーマンスを提供する。
dbdatlen	通常の結果カラム内のデータの長さを返す。	直接の同等機能はない。 • データ (CS_DATAFMT の *maxlength* フィールド内のデータ) の最大長を取得するには、ct_describe を使用する。 • バインドされた変数に置かれるデータ値の長さを調べるには、ct_bind *copied* パラメータを使用する。 • ct_get_data で取得されるデータ値の長さを得るには、ct_get_data *outlen* パラメータを使用する。
dbdayname	指定された言語の指定された曜日の名前を調べる。	cs_dt_info(CS_DAYNAME)
DBDEAD	特定の DBPROCESS が dead であるかどうかを調べる。	ct_con_props(CS_GET, CS_CON_STATUS) 返された値の CS_CONSTAT_DEAD ビットを確認する。
dberrhandle	DB-Library エラーを処理するユーザ関数をインストールする。	• ct_callback(CS_SET, CS_CLIENTMSG_CB) • cs_config(CS_SET, CS_MESSAGE_CB) 「エラー・ハンドラとメッセージ・ハンドラ」(48 ページ) を参照。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbexit	すべての DBPROCESS 構造体をクローズして割り付けを解除し、dbinit で初期化されたすべての構造体を解除する。	• ct_exit • cs_ctx_drop
dbfcmd	C ランタイム・ライブラリ sprintf 型のフォーマットを使用して、テキストを DBPROCESS コマンド・バッファに追加する。	直接の同等機能はない。 ct_command を呼び出す前に sprintf (または使用するシステムの同等の機能) を使用して、言語コマンド文字列をフォーマットする。 言語バッファにさらにテキストを追加する場合、*option* を CS_MORE として渡す。そうでない場合は CS_END として渡す。 TDS 5.0 以降を使用した接続の場合、Client-Library は言語コマンドにパラメータを使用できる。テキスト内の “@” 変数でパラメータを識別し、ct_param または ct_setparam を使用して値を渡す。
DBFIRSTROW	ロー・バッファ内の最初のローの番号を返す。	なし Client-Library はロー・バッファリングの組み込みサポートを提供しない。
dbfree_xlate	1 組の文字セット変換テーブルを解放する。	直接の同等機能はない。 文字セットは、隠し CS_LOCALE 構造体の一部として格納される。CS_LOCALE 構造体を割り付けるには cs_loc_alloc を使用し、構造体のメモリを解放するには cs_loc_drop を使用する。
dbfreebuf	コマンド・バッファをクリアする。	直接の同等機能はない。 System 10 Client-Library は、ct_send を呼び出すたびにコマンド・バッファをクリアする。 コマンドが開始されたが送信されていない場合、ct_cancel を使用してコマンド・バッファをクリアする。
dbfreequal	dbqual で割り付けられたメモリを解放する。	直接の同等機能はない。Client-Library は where 句を構築するための組み込み関数を提供しない。 この表の dbqual の項目を参照。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbfreesort	dbloadsort で割り付けられたソート順序構造体を解放する。	直接の同等機能はない。 ソート順は、隠し CS_LOCALE 構造体の一部として格納される。CS_LOCALE 構造体を割り付けるには cs_loc_alloc を使用し、構造体のメモリを解放するには cs_loc_drop を使用する。
dbgetchar	コマンド・バッファ内の文字を指すポインタを返す。	直接の同等機能はない。 言語コマンドを、ct_command に渡す前にフォーマットする。内部言語バッファはアプリケーションにはアクセスできない。
dbgetcharset	クライアント文字セットの名前を DBPROCESS 構造体から取得する。	次の呼び出しシーケンスと置き換える。 • CS_LOCALE 構造体を割り付けるには、cs_loc_alloc。 • 接続のロケールをアプリケーションの CS_LOCALE 構造体にコピーするには、ct_con_props(CS_LOC_PROP)。 • 文字セット名を取得するには、cs_locale(CS_GET, CS_SYB_CHARSET)。 • CS_LOCALE を削除するには、cs_loc_drop。
dbgetloginfo	TDS ログイン応答情報を、DBPROCESS 構造体から、新しく割り付けられた DBLOGINFO 構造体に転送する。	ct_getloginfo
dbgetlusername	LOGINREC 構造体からユーザ名を返す。	ct_con_props(CS_GET, CS_USERNAME)
dbgetmaxprocs	同時にオープンする DBPROCESS の現在の最大数を調べる。	ct_config(CS_GET, CS_MAX_CONNECT)
dbgetnatlang	ネイティブ言語を DBPROCESS 構造体から取得する。	次の呼び出しシーケンスと置き換える。 • CS_LOCALE 構造体を割り付けるには、cs_loc_alloc。 • 接続のロケールをアプリケーションの CS_LOCALE 構造体にコピーするには、ct_con_props(CS_LOC_PROP)。 • 言語名を取得するには、cs_locale(CS_GET, CS_SYB_LANG)。 • CS_LOCALE を削除するには、cs_loc_drop。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbgetoff	コマンド・バッファ内に Transact-SQL 構成体があるかどうかを確認する。	なし。
dbgetpacket	現在使用している TDS パケット・サイズを返す。	ct_con_props(CS_GET, CS_PACKETSIZE)
dbgetrow	ロー・バッファ内の指定のローを読み込む。	なし。Client-Library はロー・バッファリングの組み込みサポートを提供しない。
DBGETTIME	DB-Library が、SQL コマンドに対するサーバ応答を待つ秒数を返す。	ct_config(CS_GET, CS_TIMEOUT)
dbgetuserdata	DBPROCESS 構造体からユーザ割り付けのデータを指すポインタを返す。	ユーザ・データはコンテキスト、接続、またはコマンドのレベルでインストールされる。 • cs_config(CS_USERDATA) はコンテキスト・レベルのユーザ・データを設定または取得する。 • ct_con_props(CS_USERDATA) は接続レベルのユーザ・データを設定または取得する。 • ct_cmd_props(CS_USERDATA) はコマンド・レベルのユーザ・データを設定または取得する。 子構造体は CS_USERDATA 値を引き継がない。
dbhasretstat	現在のコマンドまたは RPC がリターン・ステータス番号を生成したかどうかを調べる。	ct_results は、ストアド・プロシージャのリターン・ステータスが到着すると CS_STATUS_RESULT の *result_type* 値を返す。 「結果を処理するコード」(61 ページ) を参照。
dbinit	DB-Library を初期化する。	• cs_ctx_alloc • ct_init
DBIORDESC (UNIX と AOS/VS のみ)	サーバから送られてくるデータを読み込むために DB-Library が使用する、UNIX ファイル記述子に対するアクセスをプログラムに提供する。	ct_con_props(CS_ENDPOINT) この機能をサポートしないプラットフォームでは、取得されるプロパティ値は -1。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
DBIOWDESC (UNIX と AOS/VS のみ)	サーバから送られてくるデータを読み込むために DB-Library が使用する、UNIX ファイル記述子に対するアクセスをプログラムに提供する。	ct_con_props(CS_ENDPOINT) この機能をサポートしないプラットフォームでは、取得されるプロパティ値は -1。
DBISAVAIL	DBPROCESS が、汎用的に使用可能かどうかを調べる。	直接の同等機能はない。プログラム論理が DBISAVAIL と DBSETAVAIL に依存している場合には、Client-Library の接続レベルまたはコマンド・レベル CS_USER_DATA プロパティを使用して、これらの呼び出しを置き換える。
dbisopt	サーバ・オプションまたは DB-Library オプションのステータスを確認する。	ct_options(CS_GET)
DBLASTROW	ロー・バッファ内の最後のローの番号を返す。	なし。Client-Library はロー・バッファリングの組み込みサポートを提供しない。
dbload_xlate	1 組の文字セット変換テーブルをロードする。	直接の同等機能はない。 文字セットは、隠し CS_LOCALE 構造体の一部として格納される。CS_LOCALE 構造体を割り付けるには cs_loc_alloc を使用し、構造体のメモリを解放するには cs_loc_drop を使用する。CS_LOCALE 構造体の文字セットを変更するには cs_locale を使用する。
dbloadsort	サーバのソート順をロードする。	直接の同等機能はない。 ソート順は、隠し CS_LOCALE 構造体の一部として格納される。CS_LOCALE 構造体を割り付けるには cs_loc_alloc を使用し、構造体のメモリを解放するには cs_loc_drop を使用する。 CS_LOCALE のソート順を変更するには、cs_locale を使用する。
dblogin	dbopen で使用するログイン・レコードを割り付ける。	ct_con_alloc 使用方法については、「接続をオープンするコード」(43 ページ) を参照。
dbloginfree	ログイン・レコードを解放する。	ct_con_drop
dbmny4add	2 つの DBMONEY4 値を加算する。	cs_calc
dbmny4cmp	2 つの DBMONEY4 値を比較する。	cs_cmp

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン

dbmny4copy	DBMONEY4 の値をコピーする。	同等の機能は組み込まれていない。
		C の標準ライブラリ memcpy (または同等の機能) を使用する。
		`CS_MONEY4 dest_mny4;`
		`CS_MONEY4 src_mny4;`
		`memcpy(&dest_mny4, &src_mny4,`
		`sizeof(CS_MONEY4));`

dbmny4divide	ある DBMONEY4 値を別の DBMONEY 値で除算する。	cs_calc
dbmny4minus	DBMONEY4 値の符号を反転する。	直接の同等機能はない。 cs_calc を使用して、ゼロ値の CS_MONEY4 変数から値を減算する。
dbmny4mul	2 つの DBMONEY4 値を乗算する。	cs_calc
dbmny4sub	ある DBMONEY4 値から別の DBMONEY 値を減算する。	cs_calc
dbmny4zero	DBMONEY4 変数を $0.0000 に初期化する。	CS_MONEY4 構造体のフィールドを 0 にするには、memset (または同等の機能) を使用する。
dbmnyadd	2 つの DBMONEY 値を加算する。	cs_calc
dbmnycmp	2 つの DBMONEY 値を比較する。	cs_cmp

dbmnycopy	DBMONEY の値をコピーする。	同等の機能は組み込まれていない。
		C の標準ライブラリ memcpy (または同等の機能) を使用する。
		`CS_MONEY dest_mny;`
		`CS_MONEY src_mny;`
		`memcpy(&dest_mny,`
		`&src_mny,`
		`sizeof(CS_MONEY));`

dbmnydec	DBMONEY 値を 10,000 分の 1 ドルだけ減らす。	直接の同等機能はない。 cs_convert を使用して、10,000 分の 1 の CS_FLOAT 値を CS_MONEY に変換し、cs_calc を使用する。
dbmnydivide	ある DBMONEY 値を別の DBMONEY 値で除算する。	cs_calc

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbmnydown	DBMONEY 値を正の整数で除算する。	直接の同等機能はない。 cs_convert を使用して、整数値を CS_MONEY に変換し、cs_calc を呼び出して、変換した値を除算する。
dbmnyinc	DBMONEY 値を 10,000 分の 1 ドルだけ増分する。	直接の同等機能はない。 cs_convert を使用して、10,000 分の 1 の CS_FLOAT 値を CS_MONEY に変換し、cs_calc を使用する。
dbmnyinit	dbmnyndigit を呼び出しのための DBMONEY 値を用意する。	dbmnyinit と dbmnyndigit については、直接の同等機能はない。 この表の dbmnydigit の項目を参照。
dbmnymaxneg	サポートされる負の最大 DBMONEY の値を返す。	なし。
dbmnymaxpos	サポートされる正の最大 DBMONEY の値を返す。	なし。
dbmnyminus	DBMONEY 値の符号を反転する。	直接の同等機能はない。 cs_calc を使用して、ゼロ値の CS_MONEY4 変数から値を減算する。
dbmnymul	2 つの DBMONEY 値を乗算する。	cs_calc
dbmnyndigit	DBMONEY 値の右端の桁を DBCHAR で返す。	直接の同等機能はない。 cs_convert を使用して、CS_MONEY 値を文字列に変換し、必要に応じて文字列を再度フォーマットする。CS_CHAR への変換で精度を失うのを避けるには、CS_MONEY、CS_NUMERIC、CS_CHAR の順に変換する。
dbmnyscale	DBMONEY 値に正の整数 (*multiplier*) を乗算し、指定の金額 (*addend* では 10,000) を加算する。	直接の同等機能はない。 cs_convert を使用して、*multiplier* および *addend* 値を同等の CS_MONEY に変換し、cs_calc を使用して、乗算と加算を実行する。
dbmnysub	ある DBMONEY 値から別の DBMONEY 値を減算する。	cs_calc
dbmnyzero	DBMONEY 値を $0.0000 に初期化する。	CS_MONEY 構造体のフィールドを 0 にするには、memset (または同等の機能) を使用する。
dbmonthname	指定された月の名前を指定された言語で返す。	• cs_dt_info(CS_MONTH)、または • cs_dt_info(CS_SHORTMONTH)

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
DBMORECMDS	処理する結果がさらにあるかどうかを示す。	直接の同等機能はない。 すべての結果が処理されると、ct_results は CS_END_RESULTS を返す。サーバから送信される結果すべてを処理するか、期待しない結果をキャンセルするように、結果ループをコーディングする。 結果処理コードの変換については、「結果を処理するコード」(61 ページ) を参照。 コマンドのキャンセルについては、「結果のキャンセル」(74 ページ) を参照。
dbmoretext	text 値または image 値の一部をサーバに送信する。	ct_send_data 使用方法については、表 6-6 (97 ページ) を参照。
dbmsghandle	サーバ・メッセージを処理するユーザ関数をインストールする。	ct_callback(CS_SERVERMSG_CB) 「エラー・ハンドラとメッセージ・ハンドラ」(48 ページ) を参照。
dbname	現在のデータベースの名前を返す。	直接の同等機能はない。 次の言語コマンドを送信して、Adaptive Server Enterprise から情報を受信する。 `select db_name()`
dbnextrow	次の結果ローを読み込む。	ct_fetch (クエリが計算ローを返す場合は ct_results)。 通常のローおよび計算ローの処理方法の図は、「結果を処理するコード」(61 ページ) を参照。 dbnextrow によって返される計算 ID を取得するには、ct_compute_info(CS_COMP_ID) を使用する。
dbnpcreate	ノーティフィケーション・プロシージャを作成する。	直接の同等機能はない。 Client-Library RPC コマンドを使用して、Open Server システム・ストアド・プロシージャ sp_regcreate を呼び出す。sp_regcreate については、『Open Server Server-Library/C リファレンス・マニュアル』を参照。
dbnpdefine	ノーティフィケーション・プロシージャを定義する。	直接の同等機能はない。 Client-Library RPC コマンドを使用して、Open Server システム・ストアド・プロシージャ sp_regcreate を呼び出す。sp_regcreate については、『Open Server Server-Library/C リファレンス・マニュアル』を参照。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbnullbind	インジケータ変数を通常の結果ローのカラムと対応させる。	ct_bind
dbnumalts	計算ローのカラム数を返す。	ct_results が CS_COMPUTE_RESULT の *result_type* で返るときは、ct_res_info(CS_NUMDATA)。
dbnumcols	現在の結果セットの通常カラムの数を返す。	ct_results が CS_ROW_RESULT の *result_type* で返るときは、ct_res_info(CS_NUMDATA)。
dbnumcompute	現在の結果セットの COMPUTE 句の数を返す。	ct_results が CS_COMPUTE_RESULT の *result_type* で返るときは、ct_res_info(CS_NUM_COMPUTES)。
DBNUMORDERS	Transact-SQL の select 文の order by 句に指定されたカラムの数を返す。	ct_res_info(CS_NUMORDERCOLS) が CS_ROW_RESULT の *result_type* で返る。
dbnumrets	ストアド・プロシージャによって生成されたリターン・パラメータ値の数を調べる。	ct_res_info(CS_NUMDATA) ct_results は、リターン・パラメータ値が到着すると CS_PARAM_RESULT の *result_type* を返す。
dbopen	DBPROCESS 構造体を作成し、初期化する。	ct_connect 使用方法については、「接続をオープンするコード」(43 ページ) を参照。
dbordercol	最後に実行されたクエリの order by 句にあるカラムの ID を返す。	次の呼び出しシーケンスと置き換える。 • order-by リストの長さを取得するには ct_res_info(CS_NUMORDERCOLS)。 • order-by リストを保持するには、CS_INT 配列を割り付ける (または既存の配列が十分大きいことを確認する)。 • order-by リストを select リスト識別子の CS_INT 配列にコピーするには、ct_res_info(CS_ORDERBY_COLS)。
dbpoll	サーバの応答が DBPROCESS に到着したかどうかを確認する。	ct_poll **注意** 使用方法が異なる。『Open Client Client-Library/C リファレンス・マニュアル』の「非同期プログラミング」を参照。
dbprhead	サーバから返されるローのカラム見出しを出力する。	直接の同等機能はない。アプリケーションのコードで置き換える。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbprrow	サーバから返されたすべてのローを出力する。	直接の同等機能はない。アプリケーションのコードで置き換える。 Client-Library のサンプル・プログラム *exutils.c* にある ex_fetch_data 関数の例では、同じような機能を提供する。このサンプル・プログラムの詳細については、使用しているプラットフォーム用の『Open Client/Server プログラマーズ・ガイド補足』を参照。
dbprtype	トークン値を読み込み可能な文字列に変換する。	直接の同等機能はない。アプリケーションのコードで置き換える。

dbqual	ブラウズ可能なテーブル内で現在のローを更新するのに使用する where 句へのポインタを返す。	直接の同等機能はない。where 句を構築するためのカラムとテーブル名を取得する ct_br_column と ct_br_table を呼び出すアプリケーションのコードで置き換える。
		ブラウズ・モードをクエリ送信する前に、アプリケーションは CS_HIDDEN_KEYS コマンド・プロパティを使用可能にする。また、アプリケーションはテーブルのタイムスタンプ・カラムにバインドし、where 句でタイムスタンプを使用する。
		次に where 句のフォーマットを示す。
		`where key1 = value_1`
		`and key2 = value_2 ...`
		`and tsequal(timestamp, ts_value)`
		各パラメータの意味は次のとおり。
		• *key1*、*value_1*、*key2*、*value_2*、およびそれに続くものは、キー・カラムとその値。
		• *ts_value* は、文字列に変換されるバイナリ・タイムスタンプ値。

DBRBUF (UNIX と AOS/VS のみ)	DB-Library ネットワーク・バッファに読み込んでいないバイトがあるかどうかを調べる。	直接の同等機能はない。非同期接続を使用する。 『Open Client Client-Library/C リファレンス・マニュアル』の「非同期プログラミング」を参照。
dbreadpage	サーバから 1 ページ分のバイナリ・データを読み込む。	なし。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbreadtext	サーバから text 値または image 値の一部を読み込む。	ct_get_data 使用方法については、「text または image データの検索」(92 ページ) を参照。
dbrecftos	アプリケーションからサーバへ送信されたすべての SQL を記録する。	なし。 アプリケーションの問題を診断するには、ct_debug を使用する。
dbrecvpassthru	サーバから TDS パケットを受信する。	ct_recvpassthru
dbregdrop	レジスタード・プロシージャを削除する。	直接の同等機能はない。 Client-Library RPC コマンドを使用して、Open Server システム・ストアド・プロシージャ sp_regdrop を呼び出す。sp_regdrop については、『Open Server Server-Library/C リファレンス・マニュアル』を参照。
dbregexec	レジスタード・プロシージャを実行する。	ct_send
dbreghandle	レジスタード・プロシージャ・ノーティフィケーションのハンドラ・ルーチンをインストールする。	ct_callback (CS_NOTIF_CB)
dbreginit	レジスタード・プロシージャの実行を開始する。	ct_command (CS_RPC_CMD)
dbreglist	Open Server で現在定義されているレジスタード・プロシージャのリストを返す。	直接の同等機能はない。 Client-Library RPC コマンドを使用して、Open Server システム・ストアド・プロシージャ sp_reglist を呼び出す。sp_reglist については、『Open Server Server-Library/C リファレンス・マニュアル』を参照。
dbregnowatch	レジスタード・プロシージャが実行するときに通知される要求を取り消す。	直接の同等機能はない。 Client-Library RPC コマンドを使用して、Open Server システム・ストアド・プロシージャ sp_regnowatch を呼び出す。sp_regnowatch については、『Open Server Server-Library/C リファレンス・マニュアル』を参照。
dbregparam	レジスタード・プロシージャ・パラメータを定義または記述する。	ct_param または ct_setparam

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbregwatch	レジスタード・プロシージャが実行するときにノーティフィケーションを要求する。	直接の同等機能はない。 Client-Library RPC コマンドを使用して、Open Server システム・ストアド・プロシージャ sp_regwatch を呼び出す。sp_regwatch については、『Open Server Server-Library/C リファレンス・マニュアル』を参照。
dbregwatchlist	DBPROCESS が監視しているレジスタード・プロシージャのリストを返す。	直接の同等機能はない。 Client-Library RPC コマンドを使用して、Open Server システム・ストアド・プロシージャ sp_regwatchlist を呼び出す。sp_regwatchlist については、『Open Server Server-Library/C リファレンス・マニュアル』を参照。
dbresults	次のクエリの結果を設定する。	ct_results 「結果を処理するコード」(61 ページ) を参照。
dbretdata	ストアド・プロシージャによって生成されるリターン (出力) パラメータ値へのポインタを返す。	直接の同等機能はない。リターン・パラメータ値をバインドおよびフェッチするか、または ct_get_data を使用する。 「データ値の検索」(67 ページ) を参照。
dbretlen	ストアド・プロシージャによって生成されるリターン・パラメータ値の長さを調べる。	直接の同等機能はない。 • データ (CS_DATAFMT の *maxlength* フィールド内のデータ) の最大長を取得するには、ct_describe を使用する。 • バインドされた変数に置かれたデータ値の長さを取得するには、ct_bind *copied* パラメータを使用する。 • ct_get_data で取得されるデータ値の長さを得るには、ct_get_data *outlen* パラメータを使用する。
dbretname	特定のリターン・パラメータ値に関連したストアド・プロシージャ・パラメータの名前を調べる。	ct_describe (CS_DATAFMT の *name* フィールド)
dbretstatus	現在のコマンドまたは RPC によって返されるストアド・プロシージャ・ステータス番号を調べる。	直接の同等機能はない。 リターン・ステータス値をバインドおよびフェッチするか、または ct_get_data を使用する。 「データ値の検索」(67 ページ) を参照。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbrettype	ストアド・プロシージャによって生成されるリターン・パラメータ値のデータ型を調べる。	ct_describe (CS_DATAFMT の *datatype* フィールド)
DBROWS	現在のコマンドが実際にローを返したかどうかを示す。	直接の同等機能はない。 コマンドがローを返すと、ct_results は CS_ROW_RESULT の *result_type* 値を返す。 「結果を処理するコード」(61 ページ) を参照。
DBROWTYPE	現在のローの型を返す。	ct_results は現在の結果セットの型を示す。 「結果を処理するコード」(61 ページ) を参照。
dbrpcinit	RPC を初期化する。	ct_command(CS_RPC_COMMAND)
dbrpcparam	RPC にパラメータを追加する。	ct_param または ct_setparam
dbrpcsend	RPC の終了を知らせる。	ct_send
dbrpwclr	LOGINREC 構造体からすべてのリモート・パスワードをクリアする。	ct_remote_pwd(CS_CLEAR)
dbrpwset	LOGINREC 構造体にリモート・パスワードを追加する。	ct_remote_pwd(CS_SET)
dbsafestr	文字列中の引用符を二重にする。	なし。アプリケーションのコードで置き換える。
dbsechandle	セキュア・ログインを処理するユーザ関数をインストールする。	• dbsechandle(DBENCRYPT) を置き換えるには、ct_callback(CS_ENCRYPT_CB) • dbsechandle(DBLABELS) を置き換えるには、ct_callback(CS_CHALLENGE_CB)
dbsendpassthru	サーバに TDS パケットを送信する。	ct_sendpassthru
dbservcharset	サーバの文字セット名を取得する。	直接の同等機能はない。 Adaptive Server Enterprise または Open Server の接続の場合、RPC コマンドを送信して sp_serverinfo Adaptive Server Enterprise カタログ・ストアド・プロシージャ (または同名の Open Server システム・レジスタード・プロシージャ) を呼び出す。“server_csname” を、名前付けしていない CS_CHAR パラメータとして渡す。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbsetavail	DBPROCESS に通常の使用可能というマーク (being available) を付ける。	直接の同等機能はない。 プログラム論理が DBISAVAIL と DBSETAVAIL に依存している場合には、ct_con_props(CS_USER_DATA) または ct_cmd_props(CS_USER_DATA) を使用して、これらの呼び出しを置き換える。
dbsetbusy	DB-Library がサーバから読み込みを行っているときに、ユーザ提供の関数を呼び出す。	直接の同等機能はない。代わりに非同期接続を使用する。 『Open Client Client-Library/C リファレンス・マニュアル』の「非同期プログラミング」を参照。
dbsetconnect	サーバ接続情報を設定する。	ct_con_props(CS_SERVERADDR)
dbsetdefcharset	アプリケーションのデフォルト文字セット名を設定する。	ロケール・ファイルの “default” エントリが、CS_CONTEXT 構造体のデフォルト文字セットを決定する。アプリケーションは、cs_loc_alloc、cs_locale、cs_config(CS_LOC_PROP) を使用してコンテキストの文字セットを変更できる。
dbsetdeflang	アプリケーションのデフォルト言語名を設定する。	ロケール・ファイルの “default” エントリが、CS_CONTEXT 構造体のデフォルト言語を決定する。アプリケーションは、cs_loc_alloc、cs_locale、cs_config(CS_LOC_PROP) を使用してコンテキストの言語を変更できる。
dbsetidle	DB-Library がサーバからの読み込みを終了したときに、ユーザ提供する関数を呼び出す。	直接の同等機能はない。 非同期接続を使用する。Client-Library は、非同期ルーチンの動作が完了するたびに接続の完了コールバックを呼び出す。 『Open Client Client-Library/C リファレンス・マニュアル』の「非同期プログラミング」を参照。
dbsetifile	Sybase interfaces ファイルの名前とロケーションを指定する。	ct_config(CS_IFILE) cs_config(CS_DEFAULT_IFILE) は、代替の Sybase interfaces ファイルの名前とロケーションを指定する。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbsetinterrupt	サーバからの読み込みを待っている間、割り込みを処理するために、ユーザが提供する関数を呼び出す。	直接の同等機能はない。 Client-Library がシグナル駆動型 I/O を使用するプラットフォームでは、システム割り込みハンドラをインストールするには ct_callback(CS_SIGNAL_CB) を使用する。 アプリケーションが、Client-Library が完了を呼び出す前に保留中のクエリをキャンセルする機能を必要とする場合は、非同期接続を使用する。Client-Library 呼び出しの完了が保留中の場合、コマンドをキャンセルするには ct_cancel(CS_CANCEL_ATTN) を使用する。
DBSETLAPP	LOGINREC 構造体内のアプリケーション名を設定する。	ct_con_props(CS_APPNAME)
DBSETLCHARSET	LOGINREC 構造体内の文字セットを設定する。	次の呼び出しシーケンスと置き換える。 • CS_LOCALE 構造体を割り付けるには、cs_loc_alloc。 • 接続の内部 CS_LOCALE 構造体をコピーするには、ct_con_props(CS_GET, CS_LOC_PROP)。 • 文字セット名を変更するには、cs_locale(CS_SET, CS_SYB_CHARSET)。 • 変更した CS_LOCALE 構造体を元の接続にコピーするには、ct_con_props(CS_SET, CS_LOC_PROP)。 • CS_LOCALE を削除するには、cs_loc_drop。 近接の DBSETLCHARSET および DBSETLNATLANG 呼び出しが置き換えられている場合は、3 番目の手順で言語と文字セットの両方を変更する。
DBSETLENCRYPT	Adaptive Server Enterprise にログインするときにパスワードの暗号化を使用するかどうかを指定する。	ct_con_props(CS_SET, CS_SEC_ENCRYPTION)
DBSETLHOST	LOGINREC 構造体の中のホスト名を設定する。	ct_con_props(CS_SET, CS_HOSTNAME)

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
DBSETLNATLANG	LOGINREC 構造体の中の言語名を設定する。	次の呼び出しシーケンスと置き換える。 • CS_LOCALE 構造体を割り付けるには、cs_loc_alloc。 • 接続の内部 CS_LOCALE 構造体をコピーするには、ct_con_props(CS_GET, CS_LOC_PROP)。 • 言語名を設定するには、cs_locale(CS_SET, CS_SYB_LANG)。 • 変更した CS_LOCALE 構造体を元の接続にコピーするには、ct_con_props(CS_SET, CS_LOC_PROP)。 • CS_LOCALE を削除するには、cs_loc_drop。 近接の DBSETLCHARSET および DBSETLNATLANG 呼び出しが置き換えられている場合は、3 番目の手順で言語と文字セットの両方を変更する。
dbsetloginfo	TDS ログイン情報を DBLOGINFO 構造体から LOGINREC 構造体へ転送する。	ct_setloginfo
dbsetlogintime	DB-Library が DBPROCESS 接続の要求に対するサーバの応答を待つ時間を秒数で設定する。	ct_config(CS_SET, CS_LOGIN_TIMEOUT)
DBSETLPACKET	アプリケーションの LOGINREC 構造体内の TDS パケット・サイズを設定する。	ct_con_props(CS_SET, CS_PACKETSIZE)
DBSETLPWD	LOGINREC 構造体の中のユーザのサーバ・パスワードを設定する。	ct_con_props(CS_SET, CS_PASSWORD)
DBSETLUSER	LOGINREC 構造体の中のユーザ名を設定する。	ct_con_props(CS_SET, CS_USERNAME)
dbsetmaxprocs	同時にオープンできる DBPROCESSes の最大数を設定する。	ct_config(CS_SET, CS_MAX_CONNECT)
dbsetnull	null 値をバインドするときに使用される代入値を定義する。	cs_setnull
dbsetopt	サーバおよび DB-Library のオプションを設定する。	ct_options がサーバのオプションを設定する。ct_config、ct_con_props、および ct_cmd_props が Client-Library プロパティを設定する。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbsetrow	バッファされたローの 1 つを「現在のロー」に設定する。	なし Client-Library はロー・バッファリングの組み込みサポートを提供しない。
dbsettime	DB-Library が SQL コマンドへのサーバの応答を待つ時間を、秒単位で設定する。	ct_config(CS_SET, CS_TIMEOUT) タイムアウトが発生したときにキャンセルするには、ct_cancel(CS_CANCEL_ATTN) をクライアント・メッセージ・ハンドラで呼び出す。次にタイムアウト・エラーのフォーマットを示す。 • 重大度 = CS_SV_RETRY_FAIL • 番号 = 63 • オリジン = 2 • レイヤ = 1
dbsetuserdata	DBPROCESS 構造体を使用して、ユーザ割り付けのデータを指すポインタを保存する。	ユーザ・データはコンテキスト、接続、またはコマンドのレベルでインストールされる。 • cs_config(CS_USERDATA) はコンテキスト・レベルのユーザ・データを設定または取得する。 • ct_con_props(CS_USERDATA) は接続レベルのユーザ・データを設定または取得する。 • ct_cmd_props(CS_USERDATA) はコマンド・レベルのユーザ・データを設定または取得する。 子構造体は CS_USERDATA 値を引き継がない。
dbsetversion	DB-Library のバージョン・レベルを指定する。	cs_ctx_alloc および ct_init は、両方ともバージョン番号をパラメータとして取る。
dbspid	指定された DBPROCESS のサーバ・プロセス ID を取得する。	直接の同等機能はない。Adaptive Server Enterprise では、以下のように言語コマンドを使用する。 `select @@spid`
dbspr1row	サーバのクエリ結果のローを 1 つバッファに入れる。	直接の同等機能はない。アプリケーションのコードで置き換える。
dbspr1rowlen	dbsprhead、dbsprline、dbspr1row が返す結果を保持するために割り付けるバッファの大きさを決定する。	直接の同等機能はない。アプリケーションのコードで置き換える。
dbsprhead	サーバのクエリ結果のヘッダをバッファに入れる。	直接の同等機能はない。アプリケーションのコードで置き換える。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbsprline	dbsprhead が生成したカラム名に下線を引く文字を選択する。	直接の同等機能はない。アプリケーションのコードで置き換える。
dbsqlexec	コマンド・バッチをサーバに送信する。	ct_send はバッチを送信する。ct_results はサーバの初期応答を取得する。 dbsqlexec リターン・コード論理の変換については、「結果を処理するコード」(61 ページ) を参照。
dbsqlok	サーバからの結果を待ち、サーバが応答している命令の正当性を検証する。	ct_results dbsqlok リターン・コード論理の変換については、「結果を処理するコード」(61 ページ) を参照。
dbsqlsend	コマンド・バッチをサーバに送信する。応答を待つことはしない。	ct_send DB-Library アプリケーションが dbsqlsend のあとに dbpoll を使用する場合は、変換したアプリケーションで非同期接続を使用する。『Open Client Client-Library/C リファレンス・マニュアル』の「非同期プログラミング」を参照。
dbstrbuild	変数のプレースホルダを含むテキストから、出力可能な文字列を構築する。	cs_strbuild
dbstrcmp	指定されたソート順を使用して、2 つの文字列を比較する。	cs_strcmp(CS_COMPARE)
dbstrcpy	コマンド・バッファの一部をコピーする。	直接の同等機能はない。 言語コマンドを、ct_command に渡す前にフォーマットする。内部言語バッファはアプリケーションにはアクセスできない。
dbstrlen	コマンド・バッファの長さを、文字数で返す。	直接の同等機能はない。 言語コマンドを、ct_command に渡す前にフォーマットする。内部言語バッファはアプリケーションにはアクセスできない。
dbstrsort	2 つの文字列のどちらが先に、ソートされたリストに現れるかを決定する。	cs_strcmp(CS_SORT)
dbtabbrowse	指定されたテーブルが、ブラウズ・モードの更新によって更新可能かどうかを決定する。	ct_br_table(CS_ISBROWSE)
dbtabcount	現在の select クエリに含まれるテーブルの数を返す。	ct_br_table(CS_TABNUM)

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbtabname	番号に基づいて、テーブル名を返す。	ct_br_table(CS_TABNAME)
dbtabsource	特定の結果カラムが取り出された、テーブル名とテーブル番号を返す。	ct_br_column (CS_BROWSEDESC の *tablename* および *tablenum* フィールド)
DBTDS	使用されている TDS (Tabular Data Stream プロトコル) のバージョンを調べる。	ct_con_props(CS_TDS_VERSION)
dbtextsize	現在のローについてまだ読み込まれていない text/image のバイト数を返す。	ct_data_info(CS_GET) は、CS_IODESC 構造体を初期化する。構造体は、*total_txtlen* フィールドに text/image カラムの全長を提供する。 「Client-Library の CS_IODESC 構造体」(93 ページ) を参照。
dbtsnewlen	ブラウズ・モードの更新後に、新しいタイムスタンプ・カラムの値の長さを返す。	直接の同等機能はない。 この表の dbtsnewval の項目を参照。
dbtsnewval	ブラウズ・モード更新後に、新しいタイムスタンプ・カラムの値を返す。	直接の同等機能はない。 ブラウズ・モード更新のあとで、サーバはパラメータ (CS_PARAM_RESULT) 結果セットとして新しいタイムスタンプを送信する。アプリケーションは新しいタイムスタンプをバインドおよびフェッチする。新しいタイムスタンプは、同じローを再度更新する where 句を構築するのに使用できる。
dbtsput	タイムスタンプ・カラムの新しい値を DBPROCESS 内の指定のテーブルの現在のローに入れる。	なし DB-Library では、dbtsput は dbtsnewval とともに使用する。どちらのルーチンも Client-Library に同等のルーチンはない。 連続したブラウズ・モードの更新がどのように Client-Library に実装されるかについては、この表の dbtsnewval の項目を参照。
dbtxptr	現在のローのカラムに対するテキスト・ポインタの値を返す。	ct_data_info(CS_GET) (CS_IODESC の *textptr* フィールド) 使用方法については、「Client-Library の CS_IODESC 構造体」(93 ページ) を参照。
dbtxtimestamp	現在のローのカラムに対するテキスト・タイムスタンプの値を返す。	ct_data_info(CS_GET) (CS_IODESC の *timestamp* フィールド) 「Client-Library の CS_IODESC 構造体」(93 ページ) を参照。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbtxtsnewval	dbwritetext の呼び出しのあと、新しいテキスト・タイムスタンプの値を返す。	アプリケーションが正常な text/image 更新を ct_send_data で送信したあとで、サーバはパラメータ (CS_PARAM_RESULT) 結果セットとして新しいタイムスタンプを送信する。 アプリケーションは返されたタイムスタンプを、text/image 更新オペレーションを制御するのに使用する CS_IODESC 構造体の *timestamp* フィールドにバインドする。 「text または image データの送信」(95 ページ) を参照。
dbtxtsput	DBPROCESS の現在のローの指定されたカラムに、新しいテキスト・タイムスタンプの値を入れる。	ct_data_info(CS_SET) タイムスタンプは CS_IODESC 構造体の *timestamp* フィールドによって表記される。 新しいテキスト・タイムスタンプがどのように取得されるかについては、この表の dbtxtsnewval の項目を参照。
dbuse	特定のデータベースを使用する。	直接の同等機能はない。 Transact-SQL use database コマンドを含む言語コマンドを送信し、結果を処理する。
dbvarylen	指定された通常の結果カラムのデータ長が、可変かどうかを決定する。	なし。
dbversion	使用中の DB-Library のバージョンを決定する。	Client-Library のバージョン文字列を取得するには、ct_config(CS_GET, CS_VER_STRING)。((dbversion は DB-Library バージョン文字列を返す。) このコンテキスト用の Client-Library を初期化するのに呼び出された ct_init のバージョンと一致する CS_INT を取得するには、ct_config(CS_GET, CS_VERSION)。
dbwillconvert	特定のデータ型変換が、DB-Library で可能かどうかを決定する。	cs_willconvert
dbwritepage	1 ページ分のバイナリ・データをサーバに書き込む。	なし。
dbwritetext	text 値または image 値をサーバに送信する。	ct_send_data 使用方法については、表 6-6 (97 ページ) を参照。

DB-Library ルーチン	DB-Library の機能	同等の Client-Library または CS-Library ルーチン
dbxlate	ある文字セットから別の文字セットに、文字列を変換する。	直接の同等機能はない。 文字列を別の文字セットに変換するには、次の呼び出しシーケンスを使用する。 • *loc1* と *loc2* の 2 つのロケールを割り付けるには、cs_loc_alloc を呼び出す。*srcfmt* と *destfmt* の 2 つの CS_DATAFMT 構造体を宣言または割り付ける。 • *loc1* と *loc2* に文字セットを設定するには、cs_locale を呼び出す。 • CS_DATAFMT 構造体の、*srcfmt* と *destfmt* の *locale* フィールドに、それぞれ *loc1* と *loc2* を割り当てる。文字データを記述するために、*srcfmt* と *destfmt* の残りのフィールドを初期化する。 • 文字列を *loc1* 文字セットから *loc2* 文字セットに変換するには、cs_convert を呼び出す。それぞれの呼び出しの前に、*srcfmt.maxlength* を、バイト単位でソース文字列の長さに設定する。 • CS_LOCALE 構造体を cs_loc_drop で解放する。

索引

B

C

D

E

I

L

R

S

T

あ

い

え

か

き

け

こ

さ

し

す

せ

そ

た

つ

て

と

ね

は

ひ

ふ

へ

ほ

ら

り

る

ろ

わ