SYBASE®
An SAP® Company

Common Libraries リファレンス・マニュアル

Open Client™/Open Server™

15.7

ドキュメント ID ： DC32841-01-1570-01
改訂： 2012 年 6 月

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

目次

はじめに

このマニュアルでは、次の内容について説明します。

- C 言語版の CS-Library は、Open Client™ Client-Library™ および Open Server™ Server-Library の両方のアプリケーションで使用できるユーティリティ・ルーチンを含みます。
- C 言語版の Bulk-Library は、Client-Library アプリケーションおよび Server-Library アプリケーションに対してバルク・コピー・ルーチンを提供します。バルク・コピーを使用すれば、データベース・テーブルとプログラム変数との間の高速データ転送が実現します。

注意　Open Client および Open Server の以前のリリースでは、Bulk-Library は「バルク・コピー・ルーチン」と呼ばれていました。

対象読者

このマニュアルは、Client-Library または Open Server のアプリケーションを作成するプログラマを対象としたリファレンス・マニュアルです。このマニュアルは、C プログラミング言語に精通したアプリケーション・プログラマを対象としています。

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 CS-Library の概要」では、CS-Library の概要について説明します。
- 「第 2 章 CS-Library ルーチン」では、指定できるパラメータや戻り値など、個々の CS-Library ルーチンに固有の情報について説明します。
- 「第 3 章 Bulk-Library」では、Bulk-Library の概要について説明します。
- 「第 4 章 Bulk-Library ルーチン」では、個々の Bulk-Library ルーチンに固有の情報について説明します。

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 および 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 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 Client-Library 移行ガイド』には、Open Client™ DB-Library™ アプリケーションを Open Client Client-Library に移行する方法に関する情報が記載されています。
- 『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 Product Documentation Web サイトを使用して製品について詳しく知ることができます。

- 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、Newsgroups、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。

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

CS-Library の概要

この章では、CS-Library の概要について説明します。この章の内容は、次のとおりです。

CS-Library の概要

CS-Library は、アプリケーション・プログラムの開発で使用されるユーティリティ・ルーチンの集まりで、以下の機能をサポートします。

- データ型変換
- 算術演算
- 文字セット変換
- 日時オペレーション
- ソート順オペレーション
- ローカライズされたエラー・メッセージ

CS-Library には、CS-Library 構造体の割り付けとその解除を行うルーチンもあります。

スタンドアロンの CS-Library アプリケーションを作成することはできますが、CS-Library の第一の役割は、Client-Library アプリケーションと Server-Library アプリケーションの両方に共通のユーティリティ・ルーチンを提供することです。

Client-Library および Server-Library のプログラムにはコンテキスト構造体が必要ですが、この構造体を割り付けるには CS-Library を使用する必要があるので、すべての Client-Library プログラムおよび Server-Library プログラムは、CS-Library を少なくとも 2 回呼び出します (CS_CONTEXT を割り付けるときと割り付けを解除するとき)。

コンテキスト構造体には、アプリケーションのランタイム環境、つまり「コンテキスト」に関する情報が格納されています。「構造体」(3 ページ) を参照してください。

CS-Library の使用

CS-Library のルーチンは、Client-Library アプリケーションまたは Server-Library アプリケーションから呼び出すことも、スタンドアロンの CS-Library アプリケーションから呼び出すこともできます。

Open Client アプリケーションと Open Server アプリケーション

一般に、CS-Library ルーチンは、Client-Library アプリケーションまたは Server-Library アプリケーションから呼び出されます。

Client-Library および Server-Library のヘッダ・ファイルである *ctpublic.h* と *ospublic.h* で CS-Library のヘッダ・ファイル *cspublic.h* がインクルードされているので、Client-Library および Server-Library アプリケーションでは、CS-Library を呼び出すための別のヘッダ・ファイルは必要ありません。

cs_ctx_alloc を呼び出して CS_CONTEXT を割り付けた後は、Client-Library または Server-Library のアプリケーションで他の CS-Library ルーチンを自由に呼び出すことができます。

スタンドアロン CS-Library アプリケーション

CS-Library の一般的な使用方法ではありませんが、スタンドアロンの CS-Library アプリケーションを開発することもできます。スタンドアロンのアプリケーションでは、たとえば、Open Client および Open Server のデータ型とデータ型変換ルーチンを呼び出すために CS-Library を呼び出します。

この種のアプリケーションは、CS-Library の標準ヘッダ・ファイル *cspublic.h* をインクルードする必要があります。

各プラットフォームでの CS-Library のコンパイルとリンクの手順については、『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

構造体

CS-Library は、CS_CONTEXT 制御構造体、CS_DATAFMT データ・フォーマット構造体、CS_LOCALE ロケール情報構造体などの構造体を使用します。

CS_CONTEXT 構造体は、その内部をアプリケーションで使用することのできない隠し構造体です。CS_CONTEXT については、次の項で簡単に説明します。

CS_CONTEXT 構造体は、Client-Library アプリケーションおよび Server-Library アプリケーションでも必要です。

- Client-Library が CS_CONTEXT 構造体を使用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』または『Open Client Client-Library/C プログラマーズ・ガイド』を参照してください。
- Server-Library が CS_CONTEXT 構造体を使用する方法については、『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。

CS_DATAFMT 構造体および CS_LOCALE 構造体については、『Open Client Client-Library/C リファレンス・マニュアル』の「第 2 章 トピックス」を参照してください。

CS_CONTEXT 構造体

CS-Library で定義される制御構造体は、CS_CONTEXT の 1 つだけです。

CS_CONTEXT 構造体は、特定のプログラミング・コンテキストを記述する構成情報を格納します。アプリケーションでは、CS_CONTEXT 構造体を割り付けてから、他の Client-Library、Server-Library、または CS-Library のルーチンを呼び出すようにする必要があります。

アプリケーションで CS_CONTEXT 構造体を割り付けるには、cs_ctx_alloc または cs_ctx_global を呼び出します。

アプリケーションで CS_CONTEXT をカスタマイズするには、コンテキスト・プロパティの値を変更します。次のルーチンは、コンテキスト・プロパティの値を変更します。

- CS-Library ルーチン cs_config (コンテキストを割り付けた後)
- Client-Library ルーチン ct_config (Client-Library ルーチン ct_init をコンテキストに対して呼び出した後)
- Server-Library ルーチン srv_props (Server-Library ルーチン srv_version をコンテキストに対して呼び出した後)

既存のすべてのコンテキスト構造体の割り付けを解除してから、アプリケーションを終了してください。アプリケーションで CS_CONTEXT 構造体の割り付けを解除するには、cs_ctx_drop を呼び出します。

データ型、定数、規則

CS-Library は、Client-Library および Server-Library と同じデータ型、定数、および規則を使用します。詳細については、次のマニュアル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「Open Client および Open Server のデータ型の使い方」の章
- 『Open Client Client-Library/C リファレンス・マニュアル』の「データ型」の項
- 『Open Server Server-Library/C リファレンス・マニュアル』の「データ型」の項

エラー処理

すべての CS-Library ルーチンは、成功または失敗のステータスを返します。アプリケーションで、これらのリターン・コードをチェックすることをおすすめします。

さらに、CS-Library ルーチンは、CS-Library メッセージを生成できます。このメッセージには、情報メッセージから致命的なエラーまでの重大度レベルがあります。アプリケーションでこれらのメッセージを受信して処理してください。ほとんどの場合、CS-Library ルーチンが失敗すると、失敗の理由を説明するメッセージが CS-Library によって生成されます。

メッセージ処理の 2 つの方法

アプリケーションで CS-Library メッセージを処理するには、次の 2 つの方法があります。

- メッセージを処理するためのコールバック・ルーチンをインストールする方法
- CS-Library ルーチン cs_diag を使用してインラインで処理する方法

コールバック方式には、次の利点があります。

- 予期しないエラーを整然と処理する方法の提供

  メッセージが生成されるたびに、CS-Library によって適切なメッセージ・コールバック・ルーチンが自動的に呼び出されるので、アプリケーションは予期しないエラーをトラップできます。インライン・エラー処理ロジックのみを使用したアプリケーションは、予期しないエラーを正常にトラップできないことがあります。

- メッセージ処理コードの集約

  すべてのエラーはコールバック内で処理されるので、各 CS-Library 呼び出しの後にインラインのメッセージ処理コードを追加する必要はありません。

インライン・メッセージ処理方式には、アプリケーションが特定の時点でメッセージの有無をチェックできるという利点があります。たとえば、接続を確立するために連続した呼び出しを行うアプリケーションは、接続関連の呼び出しがすべて完了してからメッセージの有無をチェックします。

ほとんどのアプリケーションは、コールバック方式を使用してメッセージを処理します。

アプリケーションは、特定のコンテキストに対してどの方法を使用するかを指定するために、cs_config を呼び出してメッセージ・コールバック・ルーチンをインストールするか、cs_diag を呼び出してインライン・メッセージ処理を初期化します。

アプリケーションは、インライン方式とコールバック方式を切り替えることができます。

- メッセージ・コールバック・ルーチンをインストールすると、インライン・メッセージ処理がオフになります。保存されたメッセージはすべて破棄されます。
- 同様に、cs_diag を呼び出してインライン・メッセージ処理を初期化すると、アプリケーションの CS-Library メッセージ・コールバックが「削除」されます。その結果、アプリケーションが初めて CS_GET を指定して cs_diag を呼び出すときに、この影響を示す警告メッセージが返されます。

メッセージ・コールバックがインストールされておらず、インライン・メッセージ処理も不可能な場合、CS-Library はメッセージ情報を破棄します。

メッセージを処理するコールバックの使用

アプリケーションでコールバック関数を使用して CS-Library のエラーを処理するには、次の操作を行う必要があります。

- 「CS-Library メッセージ・コールバックの定義」(6 ページ) の説明に従ってコールバック関数を宣言すること。
- cs_config を呼び出して CS_MESSAGE_CB プロパティを設定することで、コールバック・エラー・ハンドラをインストールすること。詳細については 「CS-Library メッセージ・コールバック・プロパティ」(20 ページ) を参照してください。

CS-Library メッセージ・コールバックの定義

CS-Library メッセージ・コールバックは、次のように定義されます。

```
CS_RETCODE CS_PUBLIC cslibmsg_cb(context, message)
CS_CONTEXT     *context;
CS_CLIENTMSG   *message;
```

各パラメータの意味は次のとおりです。

- *context* は、メッセージが発生した CS_CONTEXT 構造体を指すポインタです。
- *message* は、メッセージ情報が格納されている CS_CLIENTMSG 構造体を指すポインタです。CS_CLIENTMSG 構造体に関する詳細については、『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。Client-Library との類似点を次に示します。
  - CS-Library エラーのエラー重大度の意味は、Client-Library のエラーと同じです。
  - *message->msgnumber* フィールドは、ビットパックされた CS_INT です。この番号をアンパックするには、マクロ CS_LAYER、CS_ORIGIN、CS_NUMBER、CS_SEVERITY を使用します。この方法は Client-Library メッセージの場合と同じです。

  メッセージ・コールバックが呼び出されるたびに *message* が新しい値に設定されていることに注意してください。

CS-Library メッセージ・コールバックは、次のいずれかを返します。

- このコンテキストで現在行っている処理を継続するように CS-Library に指示するための CS_SUCCEED
- このコンテキストで現在行っている処理を終了するように CS-Library に指示するための CS_FAIL

CS-Library メッセージ・コールバックの例

```
/*
 **  cslib_err_handler() - CS-Library error handler.
 **
 **   This routine is the CS-Library error handler used by this
 **   application.It is called by CS-Library whenever an error
 **   occurs.Here, we simply print the error and return.
 **
 ** Parameters:
 **    context
 **      A pointer to the context handle for context
 **      on which the error occurred.
 **    error_msg
 **      The structure containing information about the
 **      error.
```

```
**
** Returns:
**      CS_SUCCEED
*/
CS_RETCODE CS_PUBLIC cslib_err_handler(context, errmsg)
CS_CONTEXT       *context;
CS_CLIENTMSG      *errmsg;
{
  /*
  ** Print the error details.
  */
  fprintf(stdout, "CS-Library error:");
  fprintf(stdout, "LAYER = (%ld) ORIGIN = (%ld) ",
          CS_LAYER(errmsg->msgnumber),
          CS_ORIGIN(errmsg->msgnumber));
  fprintf(stdout, "SEVERITY = (%ld) NUMBER = (%ld)¥n",
          CS_SEVERITY(errmsg->msgnumber),
          CS_NUMBER(errmsg->msgnumber));
  fprintf(stdout, "¥t%s¥n", errmsg->msgstring);
  /*
  ** Print any operating system error information.
  */
  if (errmsg->osstringlen > 0)
  {
    fprintf(stdout, "CS-Library OS error %ld - %s.¥n",
            errmsg->osnumber, errmsg->osstring);
  }
   /*
  ** All done.
  */
  return (CS_SUCCEED);
}
```

インライン・メッセージ処理

アプリケーションは、cs_diag を呼び出して、コンテキストのインライン CS-Library メッセージ処理を初期化します。

取得したメッセージを SQLCA、SQLCODE、または SQLSTATE に格納するアプリケーションでは、CS-Library の CS_EXTRA_INF プロパティを CS_TRUE に設定する必要があります。

CS-Library メッセージのインライン処理方式の詳細については、「第 2 章 CS-Library ルーチン」の cs_diag を参照してください。

第 2 章

CS-Library ルーチン

この章では、CS-Library の各ルーチンについて説明します。

cs_calc

説明

二項算術演算を実行します。

構文

```
CS_RETCODE cs_calc(context, op, datatype, var1,
        var2, dest)

CS_CONTEXT    *context;
CS_INT        op;
CS_INT        datatype;
CS_VOID       *var1;
CS_VOID       *var2;
CS_VOID       *dest;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

op

次の記号値のいずれかです。

op の値	算術演算	* 戻り時の dest 値
CS_ADD	加算	var1 + var2
CS_SUB	減算	var1 - var2
CS_MULT	乗算	var1 * var2
CS_DIV	除算	var1 /var2

datatype
var1、var2、dest のデータ型を示す次の記号値のいずれかです。

datatype の値	値が示すデータ型
CS_DECIMAL_TYPE	CS_DECIMAL
CS_MONEY_TYPE	CS_MONEY
CS_MONEY4_TYPE	CS_MONEY4
CS_NUMERIC_TYPE	CS_NUMERIC

*var1、*var2、*dest には、*datatype* の値が示すデータ型と同じデータ型を指定します。

var1
算術演算の 1 番目のオペランドを指すポインタです。

var2
算術演算の 2 番目のオペランドを指すポインタです。

dest
変換先バッファを指すポインタです。cs_calc が CS_FAIL を返した場合、**dest* は変更されません。

戻り値

cs_calc は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_calc の失敗の主な理由は、次のとおりです。

- 無効なパラメータの設定
- 0 による除算
- 変換先バッファのオーバフロー

cs_calc は、失敗の状態に関する CS-Library エラー・メッセージを生成します。「エラー処理」(5 ページ) を参照してください。

使用法

- *var1*、*var2*、*dest* には、datatype パラメータと同じデータ型の値を指定してください。
- エラーが発生すると、**dest* は変更されません。

参照

cs_convert

cs_cmp

説明

2 つのデータ値を比較します。

構文

```
CS_RETCODE cs_cmp(context, datatype, var1, var2,
        result)
CS_CONTEXT    *context;
CS_INT        datatype;
CS_VOID       *var1;
CS_VOID       *var2;
CS_INT        *result;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

datatype

var1 および var2 のデータ型を示す次の次の記号値のいずれかです。

datatype の値	値が示すデータ型
CS_DATE_TYPE	CS_DATE
CS_TIME_TYPE	CS_TIME
CS_DATETIME_TYPE	CS_DATETIME
CS_DATETIME4_TYPE	CS_DATETIME4
CS_DECIMAL_TYPE	CS_DECIMAL
CS_MONEY_TYPE	CS_MONEY
CS_MONEY4_TYPE	CS_MONEY4
CS_NUMERIC_TYPE	CS_NUMERIC
CS_BIGDATETIME_TYPE	CS_BIGDATETIME
CS_BIGTIME_TYPE	CS_BIGTIME

var1

比較の 1 番目のオペランドを指すポインタです。

var2

比較の 2 番目のオペランドを指すポインタです。

result

正常に処理された場合、比較結果を示すため ***result** に次の値のいずれかが設定されます。

*result の値	意味
-1	*var1* は *var2* よりも小さい。
0	*var1* と *var2* は等しい。
1	*var1* は *var2* よりも大きい。

戻り値

cs_cmp は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。cs_cmp が CS_FAIL を返すときには、**result* は設定されない。

cs_cmp の失敗の主な理由は、無効なパラメータ設定です。

cs_cmp は、失敗の状態に関する CS-Library エラー・メッセージを生成します。「エラー処理」(5 ページ) を参照してください。

使用法

- cs_cmp は 2 つの数値を比較して、比較結果を **result* に設定します。
- *var1* および *var2* には、*datatype* パラメータが示すデータ型の値を指定する必要があります。
- 文字列値を比較するには、cs_strcmp を呼び出します。

参照

cs_calc, cs_convert, cs_strcmp

cs_config

説明

CS-Library プロパティを設定および取得します。

構文

```
CS_RETCODE cs_config(context, action, property,
        buffer, buflen, outlen)

CS_CONTEXT    *context;
CS_INT        action;
CS_INT        property;
CS_VOID       *buffer;
CS_INT buflen;
CS_INT        *outlen;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

action

次の記号値のいずれかです。

action	cs_config
CS_SET	プロパティの値を設定する。
CS_GET	プロパティの値を取得する。
CS_CLEAR	プロパティの値をデフォルト値にリセットすることによってクリアする。

property

次の表に示すように、値を設定または取得するプロパティです。

表 2-1 : cs_config の property パラメータの値

property の値	内容	対処法	*buffer の値
CS_APPNAME	アプリケーション名。	設定、取得、クリア	CS_CHAR 文字列。 デフォルトは NULL。
CS_CONFIG_FILE	Open Client/Server ランタイム設定ファイルの名前とパス。CS_EXTERNAL_CONFIG の設定により外部設定が可能な場合のみ有効である。	設定、取得、クリア	CS_CHAR 文字列。 デフォルトは NULL で、プラットフォーム固有のデフォルトを使用する。「ランタイム設定ファイル・プロパティ」(17 ページ) を参照してください。
CS_DEFAULT_IFILE	代替のデフォルト interfaces ファイルの名前とパス。	設定、取得、クリア	新しい interfaces ファイルへの CS_CHAR 文字列。
CS_EXTERNAL_CONFIG	Client-Library ルーチン ct_init が外部設定ファイルを読み込み、デフォルトのプロパティ値を設定するかどうかを指定する。	設定、取得、クリア	CS_TRUE または CS_FALSE。 デフォルト (CS_TRUE) は外部設定ファイルが存在するかどうかによって異なる。「外部設定プロパティ」(18 ページ) を参照してください。
CS_EXTRA_INF	SQLCA、SQLCODE、または SQLSTATE を使用してメッセージをインラインで処理するときに必要な追加情報を返すかどうかを指定する。	設定、取得、クリア	CS_TRUE または CS_FALSE。 デフォルトは CS_FALSE。
CS_LIBTCL_CFG	代替の *libtcl.cfg* ファイルの名前とパス。	設定、取得、クリア	新しい *libtcl.cfg* 設定ファイルへの CS_CHAR 文字列。
CS_LOC_PROP	このコンテキストに対するローカライゼーション情報を定義する CS_LOCALE 構造体。	設定、取得、クリア	アプリケーションによって以前に割り付けられた CS_LOCALE 構造体。

property の値	内容	対処法	*buffer の値
CS_MESSAGE_CB	CS-Library メッセージ・コールバック・ルーチン。アプリケーションが提供する、CS-Library のエラーおよび情報メッセージ用ハンドラである。	設定、取得、クリア	*action* が CS_SET の場合、**buffer* はメッセージ・コールバック・ルーチンである。 *action* が CS_GET の場合、**buffer* には、現在インストールされているメッセージ・コールバック・ルーチンのアドレスが設定される。 デフォルトは NULL で、ハンドラがインストールされていないことを示す。
CS_NOAPI_CHK	ライブラリ関数が呼び出されるときに CS-Library が関数の引数を検証するかどうかを指定する。	設定、取得、クリア	CS_TRUE または CS_FALSE。 CS_FALSE はデフォルトであり、引数のチェックが行われることを示す。
CS_SYBASE_HOME	代替の Sybase ホーム・ディレクトリのロケーション。	設定、取得、クリア	新しい Sybase ホーム・ディレクトリへの CS_CHAR 文字列。
CS_USERDATA	ユーザ割り付けデータ。	設定、取得、クリア	ユーザ割り付けデータ。 デフォルトは適用されない。
CS_VERSION	CS-Library のバージョン。	取得のみ可能	ライブラリのバージョンを示す記号コード • CS_VERSION_100 － バージョン 10.0 の動作を示す。 • CS_VERSION_110 － バージョン 11.1 の動作を示す。 • CS_VERSION_120 － バージョン 12.0 の動作を示す。 • CS_VERSION_125 － バージョン 12.5 の動作を示す。 • CS_VERSION_150 － バージョン 15.0 の動作を示す。 • CS_VERSION_155 － バージョン 15.5 の動作を示す。 • CS_VERSION_157 － バージョン 15.7 の動作を示す。

buffer

プロパティ値を設定する場合、*buffer* はプロパティの設定に使用する値を指します。

プロパティ値を取得する場合、*buffer* は cs_config によりプロパティの値が格納される領域を指します。

プロパティ値をクリアする場合、*buffer* を NULL に、*buflen* を CS_UNUSED にして渡します。

buflen

一般に、*buflen* は **buffer* のバイト単位の長さです。

プロパティ値を設定する場合、**buffer* の値が NULL で終了している場合には、*buflen* を CS_NULLTERM として渡します。

**buffer* が固定長または記号値の場合、*buflen* を CS_UNUSED として渡します。

outlen

整数変数を指すポインタです。

プロパティ値を設定する場合には、*outlen* を使用しません。

プロパティ値を取得する場合、cs_config は、**outlen* を要求した情報のバイト単位の長さに設定します。

情報の大きさが *buflen* バイトを超える場合は、**outlen* の値を使用すると、その情報を保持するために必要なバイト数を判断できます。

アプリケーションがプロパティ値を設定する場合、または取得した値の出力長を必要としない場合に、*outlen* を NULL として渡すことができます。

戻り値

cs_config は次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

使用法

- コンテキスト・プロパティは 3 種類あります。
  - CS-Library 固有のコンテキスト・プロパティ
  - Client-Library 固有のコンテキスト・プロパティ
  - Server-Library 固有のコンテキスト・プロパティ

cs_config は、CS-Library コンテキスト・プロパティの値を設定および取得します。CS_LOC_PROP を除く cs_config によって設定されるプロパティは、CS-Library にだけ反映されます。

ct_config は、Client-Library 固有のコンテキスト・プロパティの値を設定および取得します。ct_config によって設定されるプロパティは、Client-Library にだけ反映されます。

srv_props は、Server-Library 固有のコンテキスト・プロパティの値を設定および取得します。srv_props によって設定されるプロパティは Server-Library にだけ反映されます。

- Client-Library プロパティの情報については、『Open Client Client-Library/C リファレンス・マニュアル』の「プロパティ」を参照してください。

アプリケーション名プロパティ

- CS_APPNAME は、アプリケーションが自身を呼び出すときの名前を指定します。
- cs_config は、CS_CONTEXT 構造体のアプリケーション名を設定します。Client-Library では、割り付けられた接続は、親の CS_CONTEXT 構造体のアプリケーション名を継承します。
- アプリケーション名は、Open Client/Server ランタイム設定ファイル内のセクション名を指定します。「ランタイム設定ファイル・プロパティ」(17 ページ) を参照してください。
- CS_CONTEXT 構造体の割り付け時に CS_VERSION_110 以降を指定していない場合、CS_APPNAME の設定、取得、クリアはできません。

ランタイム設定ファイル・プロパティ

- CS_CONFIG_FILE は、Open Client/Server ランタイム設定ファイルの名前とパスを指定します。
- デフォルト値は NULL です。NULL が指定されていると、プラットフォーム固有のデフォルト・ファイルが使用されます。
  - UNIX プラットフォームでは、デフォルト・ファイルは *$SYBASE/$SYBASE_OCS/config/ocs.cfg* です。この *$SYBASE* は Sybase インストール・ディレクトリのパスです。このパスは SYBASE 環境変数の値として指定されます。

- Windows プラットフォームでは、デフォルト・ファイルは *%SYBASE% ¥%SYBASE_OCS% ¥ini ¥ocs.cfg* です。この *%SYBASE%* は SYBASE インストール・ディレクトリのパスです。このパスは SYBASE 環境変数の値として指定されます。

注意 デフォルト値は、環境変数 SYBOCS_CFG によって上書きされることがあります。

- SYBOCS_CFG 環境変数を設定すると、CS_EXTERNAL_CONFIG のデフォルト値が上書きされます。これは、CSCONFIG_FILE の値を cs_config によって設定しないアプリケーションにのみ影響します。
- デフォルトの外部設定ファイルが存在しており、アプリケーションによって CS_EXTERNAL_CONFIG プロパティが明示的に CS_FALSE に設定されていなければ、Client-Library はデフォルトの外部設定ファイルから設定値を読み込みます。「外部設定プロパティ」(18 ページ) を参照してください。
- CS_CONTEXT 構造体の割り付け時に CS_VERSION_110 以降を指定していない場合、CS_CONFIG_FILE の設定、取得、クリアはできません。

外部設定プロパティ

- CS_EXTERNAL_CONFIG は、デフォルトの Client-Library プロパティ値を CS_CONTEXT 構造体に設定するために Client-Library ルーチン ct_init が Open Client/Server ランタイム設定ファイルを読み込むかどうかを制御します。
- Open Client/Server ランタイム設定ファイルの名前は、CS_CONFIG_FILE プロパティによって指定されます。「ランタイム設定ファイル・プロパティ」(17 ページ) を参照してください。
- CS_EXTERNAL_CONFIG、CS_TRUE のデフォルトは、デフォルトの外部設定ファイルが存在するかどうかによって異なります (「ランタイム設定ファイル・プロパティ」(17 ページ) を参照)。外部設定ファイルが存在する場合は、CS_EXTERNAL_CONFIG のデフォルト値が CS_TRUE になります。それ以外の場合は、CS_EXTERNAL_CONFIG のデフォルト値が CS_FALSE になります。
- 設定情報は、次のラベルが付けられたファイルの項から読み込まれます。

  [*appname*]

appname は、CS_APPNAME プロパティの値です (「アプリケーション名プロパティ」(17 ページ) を参照)。アプリケーションによって CS_APPNAME プロパティが設定されていない場合は、次のラベルの項から読み込まれます。

```
[DEFAULT]
```

- 『Open Client Client-Library/C リファレンス・マニュアル』の「ランタイム設定ファイルの使い方」では、設定ファイル・エントリの構文とキーワードについて説明しています。
- CS_CONTEXT 構造体の割り付け時に CS_VERSION_110 以降を使用していない場合、CS_EXTERNAL_CONFIG の設定、取得、クリアはできません (詳細については、cs_ctx_alloc を参照してください)。

追加情報プロパティ

- CS_EXTRA_INF は、SQLCA、SQLCODE、または SQLSTATE 構造体に入れる必要がある追加情報を CS-Library が返すかどうかを決定します。
- アプリケーションがメッセージを取得して SQLCA、SQLCODE、または SQLSTATE 構造体に入れない場合、追加情報は通常の CS-Library メッセージとして返されます。

ロケール情報プロパティ

- CS_LOC_PROP プロパティは、コンテキストのローカライゼーション情報を含む CS_LOCALE 構造体を定義します。ローカライゼーション情報には、言語、文字セット、日時、通貨、数値フォーマット、照合順が含まれます。
- 新しい接続は、親コンテキストからデフォルトのローカライゼーション情報を取得するので、CS_LOC_PROP は CS-Library と Client-Library に反映されます。
- アプリケーションが、コンテキストのローカライゼーション情報を定義するための cs_config を呼び出さないと、コンテキストは、オペレーティング・システム環境のデフォルトのローカライゼーション情報を使用します。オペレーティング・システム環境内のローカライゼーション情報が使用できない場合、コンテキストは、プラットフォーム特有のデフォルトのローカライゼーション値を使用します。
- cs_loc_alloc ルーチンは、CS_LOCALE 構造体を割り付けます。

CS-Library メッセージ・コールバック・プロパティ

- CS_MESSAGE_CB プロパティは、CS-Library メッセージ・コールバック・ルーチンを指すポインタです。このルーチンは、ユーザが用意します。アプリケーションはこのプロパティを使用して、CS-Library からエラー・メッセージまたは情報メッセージのハンドラをインストールします。
  - デフォルト値は NULL です。ハンドラがインストールされていないことを意味します。
  - アプリケーション関数は CS-Library エラーのハンドラとしてインストールできます。
  - ハンドラをインストールすると、CS-Library ルーチンでエラーや例外が発生したときに、CS-Library によってそのハンドラが呼び出されます。
- CS-Library エラー・ハンドラのコーディングの説明と例については、「CS-Library メッセージ・コールバックの定義」(6 ページ) を参照してください。
- 次のコード例は、CS-Library エラーを処理するためのハンドラ関数のインストール方法を示します。

```
/*
** Install the function cslib_err_handler as the
** handler for CS-Library errors.
*/
if (cs_config(context, CS_SET, CS_MESSAGE_CB,
              (CS_VOID *)cslib_err_handler,
              CS_UNUSED, NULL)
    != CS_SUCCEED)
{
  /* Release the context structure.*/
  (void)cs_ctx_drop(context);
  fprintf(stdout,
    "Can't install CS-Lib error handler.¥
    Exiting.¥n");
  exit(1);
}
```

- cs_ctx_alloc および cs_ctx_drop 以外の CS-Library ルーチンを呼び出す Client-Library アプリケーションには、専用の CS-Library エラー処理が必要です。アプリケーションは CS-Library エラー・ハンドラをインストールするか、cs_diag を使用して CS-Library エラーのインライン処理を行います。

  注意　CS-Library エラー・メッセージは Client-Library エラー・ハンドラには送信されません。

- Client-Library 用と CS-Library 用のコールバック・エラー・ハンドラは別々にインストールします。
  - アプリケーションは、Client-Library コールバック・ルーチンをインストールするために ct_callback を呼び出します。
  - CS-Library メッセージ・コールバック・ルーチンをインストールするには、cs_config を呼び出して CS_MESSAGE_CB プロパティの値を設定します。

  上記の違いを除けば、CS-Library メッセージ・コールバックは、Client-Library クライアント・メッセージ・コールバックとよく似ています。コールバック・ルーチンの一般情報については、『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。

API 引数をチェックするプロパティ

- CS_NOAPI_CHK プロパティは、ライブラリ関数が呼び出されたときに CS-Library が関数引数を検証するかどうかを決定します。
- CS_NOAPI_CHK の値が CS_FALSE (デフォルト) の場合、API 関数が呼び出されたときに CS-Library が引数をチェックします。CS_NOAPI_CHK を CS_TRUE に設定する場合、API のチェックはできません。
- 引数をチェックするために、CS-Library はそれぞれの関数呼び出しで渡されたパラメータを検証します。CS_LOCALE などの CS-Library 隠し構造体を指すポインタがチェックされます。また、構造体のフィールド値の不正な組み合わせもチェックされます。CS-Library が不正な引数を検出した場合に API のチェックが有効化されていると、CS-Library はエラー・メッセージを生成して関数は失敗します。CS-Library コールバック・エラー・ハンドラによってこれらのエラー・メッセージがトラップされ、表示されます。

- CS_NOAPI_CHK の値が CS_TRUE の場合、引数が使用されないと引数は検証されません。アプリケーションが CS-Library に不正な引数を渡すと、アプリケーションは正しく動作しません。これは、メモリの破壊、メモリのアクセス違反 (UNIX の「コア・ダンプ」)、および不正な結果が発生する原因となります。アプリケーションのこの状態について警告するエラー・メッセージは生成されません。API 引数チェックを有効にしてアプリケーションのデバックを完了するまでは、API 引数のチェックを無効にしないでください。

  警告！デフォルト設定 (CS_FALSE) でアプリケーションのデバックを完了していない限り、CS_NOAPI_CHK を CS_TRUE に設定しないで下さい。

ユーザ割り付けデータ・プロパティ

- CS_USERDATA プロパティは、ユーザ割り付けデータを定義します。このプロパティによって、アプリケーションは、ユーザ・データを特定のコンテキスト構造体に関連付けることができます。
- CS-Library は、ユーザ・データを内部データ領域にコピーします。その後アプリケーションは cs_config を呼び出して、データを取得できます。
- 入力段階で文字列の長さを計算するときに、文字列の null ターミネータを含めない場合、cs_config (CS_GET) を呼び出すと (その文字列の null ターミネータを引いた) 文字列だけが返されます。たとえば、null ターミネータのある 2 バイトの文字列を入力して、文字列の長さに 2 バイトを指定すると、cs_config (CS_GET) は文字列だけを返します。これに対して、null ターミネータのある 2 バイトの文字列を入力して、文字列の長さに 3 バイトを指定すると、cs_config (CS_GET) は文字列と null ターミネータを返します。
- Client-Library にも CS_USERDATA プロパティがありますが、Client-Library の CS_USERDATA は、接続レベルおよびコマンド・レベルでだけ設定されます。

Sybase ホーム・プロパティ

- CS_SYBASE_HOME プロパティは、代替の Sybase ホーム・ディレクトリの名前とパスを指定し、環境変数 *$SYBASE* (Windows の場合は *%SYBASE%*) を上書きします。

- CS_SYBASE_HOME は、CS-Library コンテキストを割り付ける前に設定する必要があります。これは、コンテキストの割り付けには、その設定元となる有効な Sybase ホーム・ディレクトリが必要となるためです。つまり、CS_SYBASE_HOME は、cs_ctx_alloc() または cs_ctx_global() を呼び出す前に設定する必要があります。CS_SYBASE_HOME を設定するには、cs_config() を NULL コンテキストで呼び出す必要があります。次に例を示します。

  ```
  ret = cs_config(NULL, CS_SET, CS_SYBASE_HOME,
  "/work/NewSybase", CS_NULLTERM, NULL);
  ```

  isql の -y オプション、および bcp ユーティリティを使用して、代替の Sybase ホーム・ディレクトリを指定することもできます。

libtcl.cfg ファイル・プロパティ

- CS_LIBTCL_CFG プロパティは、代替の *libtcl.cfg* ファイルの名前とパスを指定します。CS_SYBASE_HOME プロパティの場合と同様、CS_LIBTCL_CFG は cs_config() で NULL コンテキストを使用して設定します。また、CS-Library コンテキストを割り付ける前に設定する必要があります。次に例を示します。

  ```
  ret = cs_config(NULL, CS_SET, CS_LIBTCL_CFG,
  "/work/Sybase/OCS-15_5/config/libtcl.cfg",
  CS_NULLTERM, NULL);
  ```

デフォルトの interfaces ファイル・プロパティ

- CS_DEFAULT_IFILE プロパティは、代替の *interfaces* ファイルの名前とパスを指定します。CT-Library のプロパティ CS_IFILE とは異なり、CS_DEFAULT_IFILE は、*libtcl.cfg* ファイルで事前に指定されている代替のディレクトリ・サービスの使用を上書きしません。CS_DEFAULT_IFILE の主な目的は、*interfaces* ファイルをディレクトリ・サービスとして使用する場合に、*interfaces* ファイルの新しいデフォルト・ロケーションを設定することです。
- CS-Library コンテキストは、cs_config() を呼び出す前に割り付け、CS_DEFAULT_IFILE プロパティを設定する際に cs_config() に渡す必要があります。次に例を示します。

  ```
  ret = cs_config(ctx, CS_SET, CS_DEFAULT_IFILE,
  "/work/NewSybase/interfaces", CS_NULLTERM, NULL);
  ```

バージョン・レベル・プロパティ

- CS_VERSION プロパティは、アプリケーションが cs_ctx_alloc を使用して要求した CS-Library 動作のバージョンを表します。
- アプリケーションは、CS_VERSION の値の取得だけが可能です。

- CS_VERSION の有効な値は、次のとおりです。
  - CS_VERSION_100 ― バージョン 10.0 の動作を示す。
  - CS_VERSION_110 ― バージョン 11.1 の動作を示す。
  - CS_VERSION_120 ― バージョン 12.0 の動作を示す。
  - CS_VERSION_125 ― バージョン 12.5 の動作を示す。
  - CS_VERSION_150 ― バージョン 15.0 の動作を示す。
  - CS_VERSION_155 ― バージョン 15.5 の動作を示す。
  - CS_VERSION_157 ― バージョン 15.7 の動作を示す。

参照

cs_ctx_alloc、ct_con_props、ct_config、ct_init

cs_conv_mult

説明

ある文字セットの文字データを別の文字セットに変換するための変換乗算子を取得します。

構文

```
CS_RETCODE cs_conv_mult(context,
                        srcloc,
                        destloc,
                        conv_multiplier)
CS_CONTEXT    *context;
CS_LOCALE     *srcloc;
CS_LOCALE     *destloc;
CS_INT        *conv_multiplier;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

srcloc

CS_LOCALE 構造体を指すポインタです。変換元変数の文字セットを示します。このパラメータを NULL にはできません。

destloc

CS_LOCALE 構造体を指すポインタです。変換先変数の文字セットを示します。このパラメータを NULL にはできません。

conv_multiplier

CS_INT 変数へのポインタ。cs_conv_mult は、*srcloc* で示された文字セットを *destloc* で示された文字セットへ変換するための変換乗算子を取得して、**conv_multiplier* に入れます。

戻り値

cs_conv_mult は次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_conv_mult の失敗の主な理由は、無効なパラメータの設定です。

例

iso_1 文字セットから eucjis 文字セットに変換するための変換乗算子を取得するコード例を次に示します。

```
#define EXIT_ON_FAIL(context, ret, msg) ¥
{ if (ret != CS_SUCCEED) ¥
    { ¥
      fprintf(stdout,"Fatal error(%ld):%s¥n",(long)ret,msg); ¥
      if (context != (CS_CONTEXT *) NULL) ¥
      { (CS_VOID)ct_exit(context, CS_FORCE_EXIT); ¥
        (CS_VOID)cs_ctx_drop(context); } ¥
      exit(-1); ¥
    } }


** usa_locale uses the iso_1 character set.
*/
ret = cs_loc_alloc(context, &usa_locale);
EXIT_ON_FAIL(context, ret, "cs_loc_alloc(usa) failed.");
ret = cs_locale(context, CS_SET, usa_locale,
                CS_SYB_CHARSET, "iso_1", CS_NULLTERM, NULL);
EXIT_ON_FAIL(context, ret, "cs_locale(usa, CHARSET) failed.");

/*
 ** japan_locale uses eucjis.
 */
 ret = cs_loc_alloc(context, &japan_locale);
 EXIT_ON_FAIL(context, ret, "cs_loc_alloc(japan) failed.");

ret = cs_locale(context, CS_SET, japan_locale,
                 CS_SYB_CHARSET, "eucjis", CS_NULLTERM, NULL);
 EXIT_ON_FAIL(context, ret, "cs_locale(japan, CHARSET) failed.");

/*
 ** Get the conversion multiplier for iso_1 to eucjis conversions.
 */
 ret = cs_conv_mult(context,
                    usa_locale, japan_locale, &conv_mult);
 EXIT_ON_FAIL(context, ret, "cs_conv_mult(usa, japan) failed.");

fprintf(stdout,
         "Conversion multiplier for iso_1 to eucjis is %ld.¥n",
         (long)conv_mult);
```

使用法

- cs_conv_mult はある文字セットの文字データを別の文字セットに変換するための変換乗算子を取得します。
- 変換乗算子によって、アプリケーションは文字データを他の文字セットに変換するための変換先のデータ領域の大きさを指定できます。
- 別の文字セットに変換すると、文字列の長さが変化するため、アプリケーションでは、変換結果を格納できるだけの十分な変換先データ領域があることを確認する必要があります。変換先文字セットがマルチバイト文字セットの場合には、変換元文字セットの 1 バイトの文字が変換先では数バイトの文字に変換されることがあります。
- 変換できない文字は、1 バイトの "?"、2 バイトの "??"、または 3 バイトの "0xefbfbd" に置き換えられます。
- 「変換乗算子」は、変換元文字セットの 1 バイトの文字が変換先文字セットで変換される可能性がある最大バイト数に等しい値です。
- 文字列を他の文字列に変換する場合、アプリケーションは次のように変換乗算子を使用して変換先データ領域の大きさを指定します。

```
bytes_needed = conv_mult
               * srclen
               * CS_SIZEOF(CS_BYTE)
               + NTB
```

  各パラメータの意味は次のとおりです。

  - *bytes_needed* は変換先データ領域の必要な長さ (バイト単位) です。
  - *conv_mult* は変換乗算子の値です。
  - *srclen* は変換元文字列の長さ (バイト単位) です。
  - *NTB* は、null 終了が要求された場合は 1、それ以外の場合は 0 です。

- 『Open Client/Open Server 開発者用国際化ガイド』を参照してください。

参照

cs_convert, cs_locale, cs_manage_convert

cs_convert

説明

データ値のデータ型、ロケール、フォーマットを、別のデータ型、ロケール、フォーマットへ変換します。

構文

```
CS_RETCODE cs_convert(context, srcfmt, srcdata,
                destfmt, destdata, resultlen)
CS_CONTEXT    *context;
CS_DATAFMT    *srcfmt;
CS_VOID       *srcdata;
CS_DATAFMT    *destfmt;
CS_VOID       *destdata;
CS_INT         *resultlen;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

srcfmt

変換元データ・フォーマットを記述している CS_DATAFMT 構造体を指すポインタです。**srcfmt* 内のフィールドは、次のように使用されます。

フィールド名	フィールド設定
datatype	変換元データの型を表す型定数 (CS_CHAR_TYPE、CS_BINARY_TYPE など)。
maxlength	**srcdata* バッファ内のデータの長さ。固定長データ型の場合または *srcdata* が NULL の場合には、この値は無視される。
locale	変換元データのローカライゼーション値が含まれている CS_LOCALE 構造体を指すポインタ。**context* からのローカライゼーション値を使用する場合は NULL。

他のすべてのフィールドは無視される。

CS_DATAFMT 構造体の一般情報については、『Open Client Client-Library/C リファレンス・マニュアル』の「CS_DATAFMT 構造体」を参照してください。

srcdata

変換元データを指すポインタです。変換元データが null 値であることを示すには、*srcdata* を NULL として渡します。*srcdata* が NULL の場合、cs_convert は *destfmt–>datatype* によって示されるデータ型に対する null 代入値を **destdata* に指定します。

表 2-18 (94 ページ) に各データ型のデフォルトの null 代入値の一覧を示します。アプリケーションで cs_setnull を呼び出すことによって、アプリケーション固有の NULL 代入値を定義できます。

destfmt

変換先データ・フォーマットを記述している CS_DATAFMT 構造体を指すポインタです。表 2-2 は、使用される **destfmt* 内のフィールドの一覧です。

表 2-2 : cs_convert の *destfmt パラメータの CS_DATAFMT フィールド

フィールド名	フィールド設定
datatype	変換先データ型を表す型定数 (CS_CHAR_TYPE、CS_BINARY_TYPE など)。
maxlength	*destdata* バッファの長さ。
locale	変換先データのローカライゼーション値が含まれている CS_LOCALE 構造体を指すポインタ。**context* からのローカライゼーション値を使用する場合は NULL。
format	次の記号のビットマスク • 変換先が文字およびテキストだけの場合には、データを NULL で終了するための CS_FMT_NULLTERM、または、変数の全長にわたってスペースを埋め込むための CS_FMT_PADBLANK を使用する。 • 変換先が文字、バイナリ、テキスト、およびイメージの場合には、変数の全長にわたって NULL を埋め込むための CS_FMT_PADNULL を使用する。 • 文字変換元から文字変換先に変換する場合、CS_FMT_SAFESTR を使用して変換先の一重引用符 (') を二重にする。CS_FMT_SAFESTR は CS_FMT_NULLTERM、CS_FMT_PADBLANK、または CS_FMT_PADNULL と結合できる。 • どの型の変換先でも、フォーマット情報が与えられていない場合は CS_FMT_UNUSED を使用する。
scale	変換先となる変数に使用する位取り。 変換元データと変換先データの型が同じ場合、変換先データが変換元データから *scale* の値を取得する必要があることを示すために、*scale* を CS_SRC_VALUE に設定できる。 *scale* は、*precision* 以下でなければなりません。
precision	変換先となる変数に使用する精度。 変換元データと変換先データの型が同じ場合には、変換先データが変換元データから *precision* の値を取得することを示すには、*precision* を CS_SRC_VALUE に設定する。 *precision* は、*scale* 以上でなければならない。

他のすべてのフィールドは無視される。

CS_DATAFMT 構造体の一般情報については、『Open Client Client-Library/C リファレンス・マニュアル』の「CS_DATAFMT 構造体」を参照してください。

destdata

変換先バッファ領域を指すポインタです。

resultlen

整数変数を指すポインタです。cs_convert は、**resultlen* を、**destdata* に入れられたデータの長さ (バイト単位) に設定します。変換が失敗した場合、cs_convert は、**resultlen* を CS_UNUSED に設定します。

resultlen はオプションのパラメータです。NULL として渡すことができます。

データ型変換表

表 2-3 に、cs_convert によりサポートされているデータ型変換を示します。変換元のデータ型は一番左のカラムにあり、変換先データ型は表の上部にあります。"•" は変換がサポートされていることを示します。空白は変換がサポートされていないことを示します。

表 2-3：データ型変換表

Open Client のデータ型	CS_BINARY	CS_LONGBINARY	CS_VARBINARY	CS_BIT	CS_CHAR	CS_LONGCHAR	CS_VARCHAR	CS_DATETIME	CS_DATETIME4	CS_TINYINT	CS_SMALLINT	CS_INT	CS_DECIMAL	CS_NUMERIC	CS_FLOAT	CS_REAL	CS_MONEY	CS_MONEY4	CS_BOUNDARY	CS_SENSITIVITY	CS_TEXT	CS_IMAGE	CS_UNICHAR	CS_DATE	CS_TIME	CS_BIGINT	CS_USMALLINT	CS_UINT	CS_UBIGINT	CS_UNITEXT	CS_XML
CS_BINARY	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•		•	•	•		•
CS_LONGBINARY	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•		•	•	•		•
CS_VARBINARY	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•		•	•	•		•
CS_BIT	•	•	•	•	•	•	•			•	•	•	•	•	•	•					•	•	•				•	•	•		
CS_CHAR	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•	•		•
CS_LONGCHAR	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•	•		•
CS_VARCHAR	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•	•		•
CS_DATETIME			•		•	•	•	•	•												•		•	•	•						
CS_DATETIME4			•		•	•	•	•	•												•		•	•	•						
CS_TINYINT	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•				•	•	•		
CS_SMALLINT	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•				•	•	•		
CS_INT	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•				•	•	•		

Open Client のデータ型	CS_BINARY	CS_LONGBINARY	CS_VARBINARY	CS_BIT	CS_CHAR	CS_LONGCHAR	CS_VARCHAR	CS_DATETIME	CS_DATETIME4	CS_TINYINT	CS_SMALLINT	CS_INT	CS_DECIMAL	CS_NUMERIC	CS_FLOAT	CS_REAL	CS_MONEY	CS_MONEY4	CS_BOUNDARY	CS_SENSITIVITY	CS_TEXT	CS_IMAGE	CS_UNICHAR	CS_DATE	CS_TIME	CS_BIGINT	CS_USMALLINT	CS_UINT	CS_UBIGINT	CS_UNITEXT	CS_XML
CS_DECIMAL	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•				•	•	•		
CS_NUMERIC	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•				•	•	•		
CS_FLOAT	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•				•	•	•		
CS_REAL	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•				•	•	•		
CS_MONEY	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•				•	•	•		
CS_MONEY4	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•				•	•	•		
CS_BOUNDARY					•	•	•												•		•										
CS_SENSITIVITY					•	•	•													•	•										
CS_TEXT	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	•	•		•
CS_IMAGE	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•		•	•	•	•	•
CS_UNICHAR	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•		•	•	•		
CS_DATE			•		•	•	•	•	•												•		•	•							
CS_TIME			•		•	•	•	•	•												•		•		•						
CS_BIGINT	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•			•	•	•	•		
CS_USMALLINT	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•			•		•	•		
CS_UINT	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•			•	•		•		
CS_UBIGINT	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•			•	•	•			•	•	•			
CS_UNITEXT	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•	•	•	•
CS_XML	•	•	•		•	•	•														•	•									

日時データ型変換表

表 2-4 に、cs_convert でサポートされる日時データ型のデータ型変換を示します。変換元のデータ型は一番左のカラムにあり、変換先データ型は表の上部にあります。"•" は変換がサポートされていることを示します。空白は変換がサポートされていないことを示します。

表 2-4：日時データ型変換

Open Client のデータ型	CS_BINARY	CS_LONGBINARY	CS_VARBINARY	CS_BIT	CS_CHAR	CS_LONGCHAR	CS_VARCHAR	CS_DATETIME	CS_DATETIME4	CS_BIGDATETIME	CS_TINYINT	CS_SMALLINT	CS_INT	CS_DECIMAL	CS_NUMERIC	CS_FLOAT	CS_REAL	CS_MONEY	CS_MONEY4	CS_BOUNDARY	CS_SENSITIVITY	CS_TEXT	CS_IMAGE	CS_UNICHAR	CS_DATE	CS_TIME	CS_BIGINT	CS_BIGTIME	CS_UNITEXT
CS_BINARY	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•		•	
CS_LONGBINARY	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•		•	
CS_VARBINARY	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•		•	
CS_BIT	•	•	•	•	•	•	•				•	•	•	•	•	•	•					•	•	•					
CS_CHAR	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	
CS_LONGCHAR	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	
CS_VARCHAR	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	
CS_DATETIME			•		•	•	•	•	•	•												•		•	•	•		•	
CS_DATETIME4			•		•	•	•	•	•	•												•		•	•	•		•	
CS_BIGDATETIME			•		•	•	•	•	•	•												•		•	•	•		•	•
CS_TINYINT	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•					
CS_SMALLINT	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•					
CS_INT	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•					
CS_DECIMAL	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•					
CS_NUMERIC	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•					
CS_FLOAT	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•					
CS_REAL	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•					
CS_MONEY	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•					
CS_MONEY4	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•					
CS_BOUNDARY					•	•	•													•		•							
CS_SENSITIVITY					•	•	•														•	•							
CS_TEXT	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•		•	
CS_IMAGE	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•			•
CS_UNICHAR	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•		•	
CS_DATE			•		•	•	•	•	•	•												•		•	•				
CS_TIME			•		•	•	•	•	•	•												•		•		•		•	
CS_BIGINT	•	•	•	•	•	•	•				•	•	•	•	•	•	•	•	•			•	•	•			•		
CS_BIGTIME			•		•	•	•	•	•	•												•		•		•		•	•
CS_UNITEXT	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			•	•	•	•	•	•	•	•

CS_BIGDATETIME および CS_BIGTIME 間の変換と、以下のデータ型の間での変換はサポートされていません。

- CS_BLOB
- CS_LONG
- CS_UBIGINT
- CS_UINT
- CS_USHORT
- CS_USMALLINT
- CS_XML

CS_BIGDATETIME および CS_BIGTIME 間の変換は、すべて既存の日時変換および時間変換と同様に処理されます。

戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_convert の失敗の主な理由は、要求した変換を CS-Library がサポートしていないことです。

cs_convert は変換エラーが発生すると、CS-Library エラー・メッセージを生成します。「エラー処理」(5 ページ) を参照してください。

使用法

- 特定の変換が可能かどうかを調べるには、cs_will_convert を使用します。
- Client-Library アプリケーションでは、ct_bind が自動的に暗黙的データ変換を設定します。これによって、アプリケーションは、プログラム変数にバインドされる結果データを明示的に変換する必要がなくなります。
- cs_set_convert を呼び出すと、アプリケーションでカスタム変換ルーチンをインストールできます。特定の型を変換するためのカスタム・ルーチンがインストールされると、cs_convert または ct_bind はその型変換が必要になるたびにカスタム・ルーチンを呼び出します。
- cs_convert は、標準データ型とユーザ定義データ型の間でデータ型変換を行うことができます。これらの型変換を行うには、アプリケーションが cs_set_convert を呼び出し、適切なカスタム変換ルーチンをインストールする必要があります。

- 『Open Client Client-Library/C リファレンス・マニュアル』の「データ型」のページを参照してください。また、Adaptive Server Enterprise データ型については、『ASE リファレンス・マニュアル』を参照してください。

特定の変換

- *binary* データ型や *image* データ型との間のデータ型変換は、変換するデータに *character* データまたは *text* データが含まれていない限り、単なるバイト・コピーです。character データまたは text データを binary または image に変換する場合、cs_convert は、文字列が "0x" から始まるかどうかに関係なく、この文字列を 16 進数として解釈します。binary データの長さと固定長データの長さが一致している必要があります。データ長が一致しないと、アンダフローまたはオーバフローが発生します。
- *money*、*character*、または *text* の値を *float* に変換すると、精度が損なわれます。float 値を character または text に変換するときにも同じことが起こります。
- binary データ型や image データ型との間で変換を行うときにデータの長さが異なると、アンダフロー・エラーやオーバフロー・エラーになることがあります。たとえば、1 バイトの binary データを integer データに変換する場合に起こります。エラーを防ぐには、データ型の CS_TINYINT (1 バイト整数) を使用します。
- CS_MONEY の最大値は $922,337,203,685,477.5807、CS_MONEY4 の最大値は $214,748.3648 であるため、*float* 値を *money* に変換するとオーバフローすることがあります。
- *integer* データまたは *float* データを、*character* または *text* に変換するときにオーバフローが起こると、変換後の値の最初の文字にエラーを示すアスタリスク (*) が付きます。
- 変換先が *bit* のとき、変換される値が 0 でない場合は、bit 値は 1 に設定され、変換される値が 0 の場合は、bit 値は 0 に設定されます。
- *decimal* データまたは *numeric* データを decimal データまたは numeric データに変換する場合には、*destfmt–>scale* および *destfmt –>precision* の中で CS_SRC_VALUE を使用して、変換先データと変換元データの位取りおよび精度が同一である必要があることを示します。CS_SRC_VALUE は変換元データが decimal または numeric である場合だけに有効です。

- LOB ロケータ型 (CS_TEXTLOCATOR_TYPE、CS_IMAGELOCATOR_TYPE、CS_UNITEXTLOCATOR_TYPE) からの変換、および LOB ロケータ型への変換は特定の規則に従います。

  LOB ロケータ型から CS_CHAR_TYPE への変換では、24 バイトのロケータを、先行 "0x" を付けずに 16 進数文字列に変換します。CS_LOCATOR 構造体内のプリフェッチ・データは無視されます。

  LOB ロケータ型から LOB ロケータ型への変換は許可されます。ただし、LOB ロケータ型の変換元と変換先の型は同じである必要があります。たとえば、image LOB ロケータ型から text LOB ロケータ型への変換は許可されません。

  LOB ロケータから text、image、または unitext 型への変換では、CS_LOCATOR 変数からプレフェッチ・データを取得します。したがって、LOB ロケータのデータ型は、変換先のデータ型と同じである必要があります。すなわち、CS_TEXTLOCATOR_TYPE から CS_TEXT_TYPE、CS_IMAGELOCATOR_TYPE から CS_IMAGE_TYPE、および CS_UNITEXTLOCATOR_TYPE から CS_UNITEXT_TYPE への変換のみが許可されます。

表 2-5 : サポートされている LOB ロケータの変換

	CS_TEXTLOCATOR_TYPE	CS_IMAGELOCATOR_TYPE	CS_UNITEXTLOCATOR_TYPE
CS_CHAR_TYPE	•	•	•
CS_TEXT_TYPE	•		
CS_IMAGE_TYPE		•	
CS_UNITEXT_TYPE			•
CS_TEXTLOCATOR_TYPE	•		
CS_IMAGELOCATOR_TYPE		•	
CS_UNITEXTLOCATOR_TYPE			•

注意 Open Client および Open Server 15.0 以降では、unichar データ型がサポートされています。『Open Client Client-Library/C プログラマーズ・ガイド』の「第 3 章 Open Client および Open Server のデータ型の使い方」を参照してください。

文字セット間の変換

- cs_convert は、次の場合に文字セット変換を実行します。
  - *srctype* と *desttype* が両方とも文字ベースの型であり、かつ

- *srcfmt–>locale* が指定する文字セットが *destfmt–>locale* と異なる場合。

文字ベースの型とは、CS_CHAR、CS_LONGCHAR、CS_TEXT、CS_UNITEXT、CS_VARCHAR、CS_XML です。

- 文字セット変換を実行するアプリケーションを作成するには、次の手順に従います。

  a　cs_loc_alloc を 2 回呼び出して、*src_locale* と *dest_locale* の 2 つの CS_LOCALE 構造体を割り付けます。これらの構造体はそれぞれ、変換元データと変換先データのロケールを記述するように設定されます。

  b　*src_locale* に対応する文字セットを設定するため、cs_locale を呼び出します。この呼び出しでは、ロケール名または文字セット名のどちらかを指定できます。

  文字セット名を使用するには、*action* を CS_SET に、*type* を CS_SYB_CHARSET に、*buffer* を文字セット名にして渡します。この手順で *dest_locale* の文字セットも設定します。

  ロケール名を使用するには、*action* を CS_SET に、*type* を CS_LC_CTYPE に、*buffer* をロケール名にして渡します (ロケール名に対応する文字セットが使用される)。この手順で *dest_locale* の文字セットも設定します。

  c　(オプション) cs_conv_mult を呼び出して、*src_locale* の文字セットと *dest_locale* の文字セットとの変換に使用する変換乗算子を取得します。変換乗算子は、結果が変換先スペースでオーバフローする可能性があるかどうかを判断するために使用できます。

  d　CS_DATAFMT 構造体を設定して、変換元データと変換先データのデータ型、長さ、フォーマットを記述します。変換元の CS_DATAFMT 構造体の *locale* フィールドに *src_locale* を設定し、変換先の CS_DATAFMT 構造体の locale フィールドに *dest_locale* を設定します。

  e　cs_convert を呼び出して変換を実行します。この手順を、設定済みの CS_LOCALE および CS_DATAFMT 構造体を使用して、必要な回数繰り返します。

  f　終了したら、cs_loc_drop を呼び出して、*src_locale* と *dest_locale* の割り付けを解除します。

参照　cs_conv_mult, cs_manage_convert, cs_set_convert, cs_setnull, cs_will_convert

cs_ctx_alloc

説明

CS_CONTEXT 構造体を割り付けます。

構文

CS_RETCODE cs_ctx_alloc(version, ctx_pointer)

```
CS_INT          version;
CS_CONTEXT   **ctx_pointer;
```

パラメータ

version

対象となる CS-Library 動作のバージョンを示します。次の記号値のいずれか 1 つを指定します。

version の値	意味	サポートされる機能
CS_VERSION_100	10.0 の動作	初期バージョン
CS_VERSION_110	11.1 の動作	Unicode 文字セットのサポート。 外部設定ファイルを使用して Client-Library プロパティの設定を制御する。
CS_VERSION_120	12.0 の動作	上記の機能すべて。
CS_VERSION_125	12.5 の動作	unichar サポート、ワイド・データおよびワイド・カラム、SSL。
CS_VERSION_150	15.0 の動作	BCP パーティション、BCP 計算カラム、長い識別子、UnilibR、Adaptive Server Enterprise のデフォルト・パケット・サイズ、スクロール可能カーソル、およびクラスタのサポート。同時に unitext、xml、bigint、usmallint、uint、ubigint の各データ型もサポートする。Sybase ライブラリ名の変更に注意。
CS_VERSION_155	15.5 の動作	CS_BIGDATETIME データ型と CS_BIGTIME データ型、マイクロ秒の精度の time データ、ct_send_data の強化、Open Server 動的なリスナ、Open Client CS_RES_NOXNLMETADATA 応答機能、FIPS-140-2 準拠のパスワード暗号化。

version の値	意味	サポートされる機能
CS_VERSION_157	15.7 の動作	ラージ・オブジェクト (LOB) ロケータ・サポート、ストアド・プロシージャ・パラメータとしての LOB、ロー内とロー外の LOB のサポート、Bulk-Library と *bcp* による非実体化カラムの処理後続ゼロ維持のサポート、名前のないアプリケーションの設定に関する新しい処理、TCP ソケット・バッファ・サイズの設定、可変長のローの拡張、カーソル・クローズ時のロックの解放ロー・フォーマットのキャッシュ

ctx_pointer

ポインタ変数のアドレスです。cs_ctx_alloc は、**ctx_pointer* に、新しく割り付けた CS_CONTEXT 構造体のアドレスを設定します。

エラーの場合、cs_ctx_alloc は **ctx_pointer* を NULL に設定します。

戻り値

cs_ctx_alloc は次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_MEM_ERROR	十分なメモリを割り付けることができなかったために、ルーチンは失敗した。
CS_FAIL	他の理由でルーチンは失敗した。

cs_ctx_alloc の失敗の主な理由は、システム環境の設定の誤りです。cs_ctx_alloc は、ロケール・ファイルを読み込み、割り付けたコンテキストに対してデフォルトのローカライゼーション値を指定します。そのため、CS-Library がロケール・ファイルを取得できなかったり、ロケール・ファイルの設定に誤りがあると、cs_ctx_alloc は失敗します。

注意　cs_ctx_alloc が CS_FAIL を返した場合、詳細なエラー情報が標準エラー (SDTERR) と *sybinit.err* ファイルに送られます。このファイルは、現在の作業ディレクトリに作成されます。

ほとんどのシステムでは、SYBASE 環境変数または論理名がロケール・ファイルのロケーションを指定します。CS-Library ローカライゼーション値に必要な環境設定については、『Open Client/Server 設定ガイド Windows 版』または『Open Client/Server 設定ガイド UNIX 版』を参照してください。

上記以外の cs_ctx_alloc の失敗の理由は、次のとおりです。

- メモリが不足している
- ローカライゼーション・ファイルが見つからない
- LANG 環境変数の値がロケール・ファイル内のエントリと矛盾する

注意 標準エラー・デバイスのあるプラットフォームでは、CS-Library がロケール・ファイルを取得できない場合、cs_ctx_alloc はアメリカ英語のエラー・メッセージを標準エラー・デバイスに出力します。標準エラー・デバイスのない Windows およびその他のプラットフォームでは、アプリケーションの作業ディレクトリ内にある *sybinit.err* というテキスト・ファイルにアメリカ英語のエラー・メッセージが書き込まれます。

例

```
/*
 ** ex_init()
 */

CS_RETCODE CS_PUBLIC
 ex_init(context)
 CS_CONTEXT  **context;
 {
      CS_RETCODE       retcode;
      CS_INT             netio_type = CS_SYNC_IO;

     /* Get a context handle to use */
      retcode = cs_ctx_alloc(CS_VERSION_125, context);
      if (retcode != CS_SUCCEED)
      {
           ex_error("ex_init:cs_ctx_alloc() failed");
           return retcode;
      }

      /* Initialize Open Client */
       ...CODE DELETED.....

      /* Install client and server message handlers */
       ...CODE DELETED.....
```

```
        if (retcode != CS_SUCCEED)
         {
              ct_exit(*context, CS_FORCE_EXIT);
              cs_ctx_drop(*context);
              *context = NULL;
         }
        return retcode;
    }
```

使用法

- 「コンテキスト構造体」とも呼ばれる CS_CONTEXT 構造体には、アプリケーション・コンテキストを記述する情報が含まれています。たとえば、コンテキスト構造体には、デフォルトのローカライゼーション情報が含まれており、コンテキスト構造体によって、使用される CS-Library のバージョンが定義されます。
- コンテキスト構造体の割り付けは、Client-Library または Server-Library のアプリケーションで行う最初のステップです。
- CS_CONTEXT 構造体の割り付け後、一般に Client-Library アプリケーションは、cs_config と ct_config の両方またはいずれかを呼び出して、コンテキストをカスタマイズし、コンテキスト内で 1 つまたは複数の接続を作成します。Server-Library アプリケーションは、cs_config および srv_props を呼び出すことによって、コンテキストをカスタマイズします。
- コンテキスト構造体の割り付けを解除するには、アプリケーションは、cs_ctx_drop を呼び出します。
- cs_ctx_global でもコンテキスト構造体を割り付けることができます。cs_ctx_alloc と cs_ctx_global の違いは、cs_ctx_alloc は呼び出されるたびに新しいコンテキスト構造体を割り付けるのに対し、cs_ctx_global は、最初に呼び出されたときに一度だけ新しいコンテキスト構造体を割り付けることです。その後 cs_ctx_global を呼び出すと、既存のコンテキスト構造体を指すポインタが返されます。

- マルチスレッド・アプリケーションのシグナル処理で sigwait を使用できるようにするため、cs_ctx_alloc と cs_ctx_global はどちらも、それらがマルチスレッド・アプリケーションで最初に実行されたときにシグナルをブロックします。これらのシグナルは、Open Client/Open Server ライブラリによって制御される、1 つの専用スレッドを除いて、すべてブロックされます。このスレッドは、対応するシグナル・ハンドラが ct_callback または srv_signal ルーチンを使ってインストールされている場合にシグナルをブロックします。別のスレッドが後で生成され、このシグナルの sigwait を呼び出し、シグナルを受信したときに適切なユーザー指定のシグナル・ハンドラ関数を実行します。この動作の変更方法、およびアプリケーションでスレッド・シグナルそのものを処理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「第 2 章 Client-Library トピックス」を参照してください。

参照

ct_con_alloc、ct_config、cs_ctx_drop、cs_ctx_global、cs_config

cs_ctx_drop

説明

CS_CONTEXT 構造体の割り付けを解除します。

構文

CS_RETCODE cs_ctx_drop(context)

CS_CONTEXT *context;

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

戻り値

cs_cxt_drop は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

オープンしている接続がコンテキストに含まれていると、cs_ctx_drop は CS_FAIL を返します。

例

```
/*
 ** ex_ctx_cleanup()
 **
 ** Parameters:
 **  context  Pointer to context structure.
```

```
**  status   Status of last interaction with Client-
**           Library.If not ok, this routine will perform
**                 a force exit.
**
** Returns:
** Result of function calls from Client-Library.
     */
     CS_RETCODE CS_PUBLIC
     ex_ctx_cleanup(context, status)
     CS_CONTEXT       *context;
     CS_RETCODE      status;
     {
        CS_RETCODE       retcode;
        CS_INT             exit_option;
        exit_option = (status != CS_SUCCEED) ? CS_FORCE_EXIT :
           CS_UNUSED;
        retcode = ct_exit(context, exit_option);
        if (retcode != CS_SUCCEED)
        {
           ex_error("ex_ctx_cleanup:ct_exit() failed");
           return retcode;
        }
        retcode = cs_ctx_drop(context);
        if (retcode != CS_SUCCEED)
        {
           ex_error("ex_ctx_cleanup:cs_ctx_drop() failed");
           return retcode;
        }
        return retcode;
     }
```

使用法

- CS_CONTEXT 構造体は、一連のサーバ接続に対して、特定のコンテキストまたはオペレーティング環境を記述します。
- CS_CONTEXT 構造体の割り付けを解除すると、CS_CONTEXT を再び使用することはできません。アプリケーションは、新しい CS_CONTEXT を割り付けるために cs_ctx_alloc を呼び出します。

  注意　Sybase は、アプリケーション・プログラムごとに、コンテキスト・ハンドラを 1 つだけサポートします。

- Client-Library アプリケーションは、ct_exit を呼び出してコンテキストに対応する Client-Library 領域を解放するまでは、cs_ctx_drop を呼び出して CS_CONTEXT 構造体の割り付けを解除することはできません。

参照

cs_ctx_alloc、ct_close、ct_exit

cs_ctx_global

説明

CS_CONTEXT 構造体を返します。

構文

CS_RETCODE cs_ctx_global(version, ctx_pointer)

```
CS_INT          version;
CS_CONTEXT   **ctx_pointer;
```

パラメータ

version

対象となる CS-Library 動作のバージョンを示します。次の記号値のいずれか 1 つを指定します。

version の値	意味	サポートされる機能
CS_VERSION_100	10.0 の動作	初期バージョン
CS_VERSION_110	11.1 の動作	Unicode 文字セットのサポート。 外部設定ファイルを使用して Client-Library プロパティの設定を制御する。
CS_VERSION_120	12.0 の動作	上記の機能すべて。
CS_VERSION_125	12.5 の動作	unichar サポート、ワイド・データおよびワイド・カラム、SSL。
CS_VERSION_150	15.0 の動作	BCP パーティション、BCP 計算カラム、長い識別子、Unilib、Adaptive Server Enterprise のデフォルト・パケット・サイズ、スクロール可能カーソル、およびクラスタのサポート。同時に unitext、xml、bigint、usmallint、uint、および ubigint の各データ型もサポートする。Sybase ライブラリ名の変更に注意。
CS_VERSION_155	15.5 の動作	CS_BIGDATETIME データ型と CS_BIGTIME データ型、マイクロ秒の精度の time データ、ct_send_data の強化、Open Server 動的なリスナ、Open Client CS_RES_NOXNLMETADATA 応答機能、FIPS-140-2 準拠のパスワード暗号化。

version の値	意味	サポートされる機能
CS_VERSION_157	15.7 の動作	ラージ・オブジェクト (LOB) ロケータ・サポート、ストアド・プロシージャ・パラメータとしての LOB、ロー内とロー外の LOB のサポート、Bulk-Library と *bcp* による非実体化カラムの処理後続ゼロ維持のサポート、名前のないアプリケーションの設定に関する新しい処理、TCP ソケット・バッファ・サイズの設定、可変長のローの拡張、カーソル・クローズ時のロックの解放ロー・フォーマットのキャッシュ

アプリケーションがすでに CS_CONTEXT 構造体を割り付けている場合、*version* は以前に要求したバージョンと一致しなければなりません。

ctx_pointer

ポインタ変数のアドレスです。cs_ctx_global は、新しく割り付けた CS_CONTEXT 構造体またはすでに割り付けられている CS_CONTEXT 構造体のアドレスを ***ctx_pointer* に設定します。

エラーの場合、cs_ctx_global は ***ctx_pointer* を NULL に設定します。

戻り値

cs_ctx_global は次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_MEM_ERROR	十分なメモリを割り付けることができなかったために、ルーチンは失敗した。
CS_FAIL	他の理由でルーチンは失敗した。

cs_ctx_global の失敗の主な理由は、次のとおりです。

- メモリ不足
- *version* 値が以前に要求したバージョンと一致していない

注意　cs_ctx_global が CS_FAIL を返した場合、詳細なエラー情報が標準エラー (SDTERR) と *sybinit.err* ファイルに送られます。このファイルは、現在の作業ディレクトリに作成されます。

アプリケーションで cs_ctx_global を実行するため、このルーチンを初めて呼び出すときに、設定の問題が原因でこの呼び出しが失敗することがあります。この章の cs_ctx_alloc の「戻り値」を参照してください。

使用法

- cs_ctx_alloc でもコンテキスト構造体を割り付けることができます。cs_ctx_alloc と cs_ctx_global の唯一の違いは、cs_ctx_alloc が呼び出されるたびに新しいコンテキスト構造体を割り付けるのに対し、cs_ctx_global は最初に呼び出されたときに一度だけ新しいコンテキスト構造体を割り付けることです。その後 cs_ctx_global を呼び出すと、既存のコンテキスト構造体を指すポインタが返されます。
- cs_ctx_global は、複数の独立したモジュールから 1 つのコンテキスト構造体へアクセスする必要があるアプリケーションで使用されます。
- マルチスレッド・アプリケーションのシグナル処理で sigwait を使用できるようにするため、cs_ctx_alloc と cs_ctx_global はどちらも、それらがマルチスレッド・アプリケーションで最初に実行されたときにシグナルをブロックします。これらのシグナルは、Open Client/Open Server ライブラリによって制御される、1 つの専用スレッドを除いて、すべてブロックされます。このスレッドは、対応するシグナル・ハンドラが ct_callback または srv_signal ルーチンを使ってインストールされている場合にシグナルをブロックします。別のスレッドが後で生成され、このシグナルの sigwait を呼び出し、シグナルを受信したときに適切なユーザー指定のシグナル・ハンドラ関数を実行します。この動作の変更方法、およびアプリケーションでスレッド・シグナルそのものを処理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「第 2 章 Client-Library トピックス」を参照してください。
- この章の cs_ctx_alloc を参照してください。

参照

cs_ctx_alloc、cs_ctx_drop、cs_config、ct_con_alloc、ct_config

cs_diag

説明

インライン・エラー処理を管理します。

構文

```
CS_RETCODE cs_diag(context, operation, type, index,
        buffer)

CS_CONTEXT    *context;
CS_INT        operation;
CS_INT        type;
CS_INT        index;
CS_VOID       *buffer;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

operation

実行するオペレーションです。表 2-6 (46 ページ) に *operation* の有効値を示します。

type

operation の値に応じて、*type* は、メッセージ情報を受け取る構造体のタイプ、処理の対象となるメッセージのタイプ、またはその両方を示します。

値は次のとおりです。

type の値	意味
SQLCA_TYPE	SQLCA 構造体。
SQLCODE_TYPE	長整数型の SQLCODE 構造体。
SQLSTATE_TYPE	6 要素の文字配列である SQLSTATE 構造体。
CS_CLIENTMSG_TYPE	CS_CLIENTMSG 構造体。CS-Library メッセージを示すときにも使用する。

index

対象となるメッセージのインデックスです。最初のメッセージのインデックスが 1、2 番目のメッセージは 2、以下同様に続きます。

buffer

データ領域を指すポインタ

operation の値に応じて、buffer は構造体または CS_INT を指します。

戻り値

cs_diag は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。
CS_NOMSG	アプリケーションが、有効なインデックスの中で最大のものより大きいインデックスを持つメッセージを取得しようとした。たとえば、アプリケーションがメッセージ番号 3 を取得しようとしたが、使用可能なメッセージが 2 つしかなかった。

cs_diag の失敗の主な理由は、次のとおりです。

- 無効な *context*
- メモリの割り付けができない
- パラメータの組み合わせが無効

使用法

表 2-6 : cs_diag パラメータの使用法のまとめ

operation の値	cs_diag	type の値	index の値	buffer の値
CS_INIT	インライン・エラー処理を初期化する。	CS_UNUSED	CS_UNUSED	NULL
CS_MSGLIMIT	保管するメッセージの最大数を設定する。	CS_CLIENTMSG_Type	CS_UNUSED	整数値を指すポインタ。
CS_CLEAR	このコンテキストに対するメッセージ情報をクリアする。 *buffer* が NULL でない場合、cs_diag は、ブランクおよび NULL の両方またはいずれかを適切に使用して初期化することによって、**buffer* 構造体もクリアする。	*type* の有効値のいずれか。	CS_UNUSED	タイプが *type* によって定義されている構造体を指すポインタ、または NULL。

operation の値	cs_diag	type の値	index の値	buffer の値
CS_GET	特定のメッセージを取得する。	*type* の有効値のいずれか。	1 から始まる取得するメッセージのインデックス	タイプが *type* によって定義されている構造体を指すポインタ。
CS_STATUS	現在保管されているメッセージの数を返す。	CS_CLIENTMSG_Type	CS_UNUSED	整数値を指すポインタ。

- CS-Library の呼び出しが含まれているアプリケーションは、次のいずれかの方法で CS-Library メッセージを処理できます。
  - アプリケーションは、cs_config を呼び出して、CS-Library メッセージ・コールバックをインストールできます。
  - アプリケーションは、cs_diag を使用して、CS-Library メッセージをインラインで処理できます。

  アプリケーションは、インライン方式とコールバック方式を切り替えることができます。
  - CS-Library メッセージ・コールバックをインストールすると、インライン・メッセージ処理はオフになります。保存されたメッセージはすべて破棄されます。
  - 同様に、cs_diag(CS_INIT) は、アプリケーションの CS-Library メッセージ・コールバックを「削除」します。cs_diag(CS_INIT) が呼び出されたときに、アプリケーションにメッセージ・コールバックがインストールされている場合、アプリケーションの cs_diag に対する最初の CS_GET 呼び出しは、この影響に対する警告メッセージを取得します。

  CS-Library メッセージ・コールバックがインストールされておらず、インライン・メッセージ処理が不可能な場合、CS-Library はメッセージ情報を破棄します。
- cs_diag は、特定のコンテキストに対するインライン・メッセージ処理を管理します。アプリケーションが 2 つ以上のコンテキストを持つ場合、各コンテキストに対して cs_diag を別々に呼び出す必要があります。

- マルチスレッド・アプリケーションでは、cs_diag は cs_diag を呼び出したスレッドによって実行された CS-Library 呼び出しに関するメッセージだけをレポートします。『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。
- cs_diag によって、アプリケーションはメッセージ情報を取得し、CS_CLIENTMSG、SQLCA、SQLCODE、または SQLSTATE 構造体に格納できます。メッセージを取得する時に、cs_diag は、*type* によって指定されたタイプの構造体を *buffer* が指しているものと想定します。

  取得したメッセージを SQLCA 構造体、SQLCODE 構造体、または SQLSTATE 構造体に保管するアプリケーションでは、CS-Library のコンテキスト・プロパティ CS_EXTRA_INF を CS_TRUE に設定する必要があります。これは、通常では CS-Library のエラー処理メカニズムによって返されない情報が SQL 構造体に格納されるからです。

  SQL 構造体を使用していないアプリケーションも、CS_EXTRA_INF を CS_TRUE に設定できます。この場合は、追加の情報が標準の CS-Library メッセージとして返されます。
- cs_diag は、新しいメッセージを保存できる十分な内部記憶領域がない場合、読み込まれていないすべてのメッセージを破棄し、メッセージの保存を停止します。次回、*operation* を CS_GET に設定して呼び出されたとき、空き領域の問題を示すために特別なメッセージを返します。

  このメッセージを返した後、cs_diag はメッセージの保存を再開します。

インライン・エラー処理の初期化

- インライン・エラー処理を初期化するには、*operation* に CS_INIT を指定して cs_diag を呼び出します。
- 通常、コンテキストがインライン・エラー処理を使用する場合、アプリケーションは、割り付け直後に cs_diag を呼び出して、コンテキストに対するインライン・エラー処理を初期化する必要があります。

メッセージのクリア

- アプリケーションは、コンテキストに対するメッセージ情報をクリアするために、*operation* に CS_CLEAR を設定して、cs_diag を呼び出します。
  - cs_diag は、*type* 値と対応しているデータ型の構造体を *buffer* が指していると想定します。
  - ct_diag は、**buffer* 構造体にブランクと NULL の両方またはいずれかを適宜設定して、この構造体をクリアします。
- アプリケーションが *operation* に CS_CLEAR を設定して、明示的に cs_diag を呼び出すまで、メッセージ情報はクリアされません。メッセージを取得してもメッセージ・キューからは削除されません。

メッセージの取得

- メッセージを取得するために、アプリケーションは、CS_GET を *operation* に、メッセージを取得する構造体のタイプを *type* に、1 から始まる対象となるメッセージのインデックスを *index* に、さらに、適切なタイプの構造体を **buffer* に設定して、cs_diag を呼び出します。
- cs_diag は、**buffer* 構造体にメッセージ情報を格納します。
- 指定されている最大有効インデックスよりも大きいインデックスを持つメッセージを取得しようとすると、cs_diag は、使用可能なメッセージがないことを示す CS_NOMSG を返します。
- これらの構造体の詳細については、『Open Client Client-Library/C リファレンス・マニュアル』の「SQLCA 構造体」、「SQLCODE 構造体」、「SQLSTATE 構造体」、「CS_CLIENTMSG 構造体」を参照してください。

メッセージの制限

- メモリに制限のある環境でアプリケーションを実行する場合に、そのアプリケーション内で CS-Library によって保存されるメッセージの数を制限できます。
- 保存できるメッセージの数を制限するため、アプリケーションは *operation* を CS_MSGLIMIT に設定し、*type* を CS_CLIENTMSG_TYPE に設定して、cs_diag を呼び出します。
- 指定したメッセージの制限数に達すると、CS-Library は新しいメッセージをすべて破棄します。
- アプリケーションは、現在保存されているメッセージの数よりも少ないメッセージ数を設定できません。

- CS-Library は、デフォルトではメッセージを無制限に保存します。アプリケーションでこのデフォルトの動作に戻すには、メッセージの制限数を CS_NO_LIMIT に設定します。

メッセージ数の取得

- 現在のメッセージ数を取得するには、アプリケーションは、*operation* を CS_STATUS に設定し、*type* を CS_CLIENTMSG_TYPE に設定して cs_diag を呼び出します。

参照 ct_callback、ct_options

cs_dt_crack

説明 マシン読み込み可能な日時値を、ユーザが使用できるフォーマットに変換します。

構文

```
CS_RETCODE cs_dt_crack(context, datetype, dateval,
                daterec)

CS_CONTEXT    *context;
CS_INT        datetype;
CS_VOID       *dateval;
CS_DATEREC    *daterec;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

datetype

**dateval* のデータ型を示す記号値です。

datetype の値	意味
CS_DATE_TYPE	CS_DATE **dateval*
CS_TIME_TYPE	CS_TIME **dateval*
CS_DATETIME_TYPE	CS_DATETIME **dateval*
CS_DATETIME4_TYPE	CS_DATETIME4 **dateval*
CS_BIGDATETIME_TYPE	CS_BIGDATETIME **dateval*
CS_BIGTIME_TYPE	CS_BIGTIME **dateval*

dateval

変換する日付、時刻、または日時の値を指すポインタです。

daterec

CS_DATEREC 構造体へのポインタ。cs_dt_crack は、変換した日付、時刻、または日時の値をこの構造体に格納します。CS_DATEREC は次のように定義されています。

```
typedef struct cs_daterec {
     CS_INT      dateyear;      /* year              */
     CS_INT      datemonth;     /* month             */
     CS_INT      datedmonth;    /* day of month      */
     CS_INT      datedyear;     /* day of year       */
     CS_INT      datedweek;     /* day of week       */
     CS_INT      datehour;      /* hour              */
     CS_INT      dateminute;    /* minute            */
     CS_INT      datesecond;    /* second            */
     CS_INT      datemsecond;   /* millisecond       */
     CS_INT      datetzone;     /* timezone          */
     CS_INT      datesecfrac;   /* second fractions  */
     CS_INT      datesecprec;   /* precision         */

} CS_DATEREC;
```

各パラメータの意味は次のとおりです。

- *dateyear* は、1900 以上の値です。
- *datemonth* は、0 から 11 の値です。
- *datedmonth* は、1 から 31 の値です。
- *datedyear* は、1 から 366 の値です。
- *datedweek* は、0 から 6 の値です。
- *datehour* は、0 から 23 の値です。
- *dateminute* は、0 から 59 の値です。
- *datesecond* は、0 から 59 の値です。
- *datemsecond* は、0 から 999 の値です。
- *datetzone* は、今後の使用のために予約されています。*cs_dt_crack* はこのフィールドを設定しません。
- *datesecfrac* は、秒以下の数です。このフィールドは、ミリ秒より大きい精度の日時データ型でのみ使用されます。
- *datesecprec* は精度を表します。CS_BIGDATETIME および CS_BIGTIME の場合、このフィールドは常に 10^6 です。このフィールドは、ミリ秒より大きい精度の日時データ型でのみ使用されます。

これらの数字の意味は、アプリケーションのロケールに応じて変わります。たとえば、ローカライゼーション情報で週の最初の日が日曜日であると指定されている場合には、値 2 の *datedweek* は火曜日を表します。ローカライゼーション情報で、週の最初の日を月曜日であると指定されている場合には、値 2 の *datedweek* は水曜日を表します。

アプリケーションで cs_dt_info を呼び出すと、有効な日付関連ローカライゼーション値を確認できます。

タイム・ゾーン・フィールド (*datetzone*) は、今後の使用のために予約されています。このフィールドは設定しません。

『Open Client Client-Library/C リファレンス・マニュアル』の「国際化のサポート」を参照してください。

戻り値

cs_dt_crack は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_dt_crack の失敗の主な理由は、無効なパラメータの設定です。

使用法

- cs_dt_crack は、日付、時刻、または日時の値をその整数要素に変換し、変換後の整数要素を CS_DATEREC 構造体に格納します。
- 日時値は内部フォーマットで保管されます。たとえば、CS_DATETIME 値は、1900 年 1 月 1 日からの日数に、真夜中からの 300 分の 1 秒の数を加えた数として保管されます。cs_dt_crack は、このような値を、アプリケーションが容易に使用できるフォーマットに変換します。
- ミリ秒までの精度の日時データ型の場合、秒以下は *datemsecond* フィールドに格納され、*datesecfrac* フィールドは使用されません。マイクロ秒以上の精度の日時データ型の場合、秒以下は *datesecfrac* フィールドに格納され、*datemsecond* フィールドは使用されません。このため、cs_dt_crack ルーチンを呼び出すアプリケーションは、使用している日時データ型に基づいて秒以下がどこに格納されているかを判断する必要があります。

参照

cs_dt_info

cs_dt_info

説明

言語特有の日付、時刻、または日時の情報を設定または取得します。

構文

```
CS_RETCODE cs_dt_info(context, action, locale, type,
        item, buffer, buflen, outlen)

CS_CONTEXT    *context;
CS_INT        action;
CS_LOCALE     *locale;
CS_INT        type;
CS_INT        item;
CS_VOID       *buffer;
CS_INT        buflen;
CS_INT        *outlen;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

日付、時刻、または日時情報を取得するときに *locale* が NULL の場合、cs_dt_info は、このコンテキスト構造体に含まれているデフォルトのロケール情報を使用します。

action

次の記号値のいずれかです。

action の値	**cs_dt_info**
CS_SET	日付、時刻、または日時変換フォーマットの設定
CS_GET	日付、時刻、または日時情報の取得

locale

CS_LOCALE 構造体を指すポインタです。ロケール構造体には日時情報を含む、ロケール情報が含まれています。

日時情報を設定する場合は、*locale* を指定する必要があります。

日時情報を取得する場合には、*locale* に NULL を設定することもできます。*locale* が NULL の場合、cs_dt_info は、**context* に格納されているデフォルトのロケール情報を使用します。

type

対象となる情報のタイプです。表 2-7 に、*type* の有効値の一覧を示します。

item

情報を取得する場合、*item* は取得するタイプ・カテゴリの項目番号です。たとえば、最初の月の名前を取得する場合、アプリケーションは、*type* に CS_MONTH を設定し、*item* に 0 を設定して渡します。

日時変換フォーマットを設定する場合には、*item* に CS_UNUSED を設定して渡してください。

buffer

日時情報を取得する場合、*buffer* は、cs_dt_info が要求された情報を格納する領域を指します。

**buffer* が要求された情報を保持するだけの大きさを持たないことを *buflen* が示す場合、cs_dt_info は **outlen* を要求された情報の長さに設定し、CS_FAIL を返します。

日時変換フォーマットを設定する場合、*buffer* は変換フォーマットを表す記号値を指します。

buflen

**buffer* の長さ (バイト単位)

item が CS_12HOUR の場合、*buflen* に CS_UNUSED を設定して渡します。

outlen

整数変数を指すポインタです。

cs_dt_info は、**outlen* を、要求された情報の長さ (バイト単位) に設定します。

要求された情報が *buflen* バイトよりも大きい場合、アプリケーションは **outlen* の値を使用して、情報の保持に必要なバイト数を判断します。

戻り値

cs_dt_info は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_dt_info の失敗の主な理由は、無効なパラメータの設定です。

使用法　　**表 2-7：cs_dt_info パラメーター覧**

type の値	cs_dt_info	action の有効値	item の有効値	*buffer の値
CS_MONTH	月名文字列を取得する。	CS_GET	0 － 11	文字列。
CS_SHORTMONTH	省略形の月名文字列を取得する。	CS_GET	0 － 11	文字列。
CS_DAYNAME	曜日名文字列を取得する。	CS_GET	0 － 6	文字列。
CS_DATEORDER	日付順文字列を取得する。	CS_GET	CS_UNUSED	日時フォーマットの月、日、年の位置を示す 3 つの文字 "m"、"d"、"y" を含んでいる文字列。
CS_12HOUR	言語が 12 時間表示フォーマットを使用しているかどうかを取得する。	CS_GET	CS_UNUSED	12 時間表示を使用している場合は CS_TRUE、24 時間表示を使用している場合は CS_FALSE。
CS_DT_CONVFMT	日時変換フォーマットを設定または取得する。	CS_GET または CS_SET	CS_UNUSED	記号値。有効値の一覧については、次のコメント・セクションを参照。

- cs_dt_info は、ネイティブ言語特有の日時情報を設定または取得します。
  - cs_dt_info は、ネイティブ言語の日付部分名、日付部分の順序情報、日時フォーマット情報、および言語が 12 時間表示日付フォーマットを使用しているかどうかを示す情報を返すことができます。
  - cs_dt_info は、日時フォーマット情報を設定できます。
- *locale* が NULL の場合、cs_dt_info は ***context* 内のネイティブ言語のロケール情報を探します。アプリケーションは、*property* を CS_LOC_PROP に設定して cs_config を呼び出し、CS_CONTEXT のロケール情報を設定します。

  ロケール情報を特に設定しないと、CS_CONTEXT のデフォルト・ロケール情報は、コンテキストが割り付けられたときに、CS-Library がオペレーティング・システムから取得する情報になります。オペレーティング・システムからのロケール情報が使用できない場合、CS-Library は、新しいコンテキスト内のプラットフォーム特有のローカライゼーション値を使用します。

- ロケールの日付順文字列は、文字データ型から CS_DATE、CS_DATETIME、または CS_DATETIME4 に変換するときに表現があいまいな日付文字列を解釈する方法を表します。ロケールの日付順文字列を取得するには、*type* に CS_DATEORDER を指定して cs_dt_info を呼び出します。たとえば、「04/05/96」は、「1996 年 4 月 5 日」とも「1996 年 5 月 4 日」とも解釈できます。前者に対応する日付順文字列は「mdy」で、後者は「dmy」です。

  アプリケーションはロケールの日付順文字列を直接設定できませんが、cs_locale を呼び出し、日付を変換するときに使用される各国言語を変更できます。これを行うには、アプリケーションで、*action* に CS_SET、*type* に CS_LC_TIME、**buffer* にロケール名を設定して cs_locale を呼び出します。アプリケーションは、異なる日付順文字列を使用するために設定された各国言語のロケールを指定できます。各国言語の日付順文字列は、次のように設定されます。

  - 各国言語に対して、*$SYBASE/locales/messages* サブディレクトリ内の各国言語サブディレクトリに *common.loc* ファイルが配置されます。
  - common.loc ファイルの [datetime] セクションにある "dateformat" 設定は日付順文字列を指定します。次に例を示します。

    ```
    [datetime]
    dateformat=dmy
    ```

  『Open Client/Server 設定ガイド Windows 版』または『Open Client/Server 設定ガイド UNIX 版』を参照してください。

- 日付変換フォーマットは、CS_DATE、CS_TIME、CS_DATETIME、CS_DATETIME4、CS_BIGDATETIME、または CS_BIGTIME の値が文字ベースのデータ型に変換される場合の結果のフォーマットを記述します。日付変換フォーマットを設定または取得するには、*type* に CS_DT_CONVFMT を設定して cs_dt_info を呼び出します。
- 表 2-8 は、CS_CHAR から CS_DATETIME、CS_DATE、または CS_TIME への変換において、*type* が CS_DT_CONVFMT の場合に **buffer* に対して有効な値の一覧です。この変換フォーマットは、文字列をこれらの日時データ型のいずれかに変換した場合の結果の記述にも使用できます。

表 2-8：type が CS_DT_CONVFMT (cs_dt_info) の場合の *buffer の値

記号値	CS_DATETIME を変換した CS_CHAR (例：Aug 24 2009 5:36PM)	CS_DATE から変換された CS_CHAR (例：Aug 24 2009)	CS_TIME を変換した CS_CHAR (例：5:36PM)
CS_DATES_HM	hh:mm 17:36	hh:mm 00:00	hh:mm 17:36
CS_DATES_HMA	hh:mm[AM|PM] 17:36:00AM	hh:mm 12:00AM	hh:mm 17:36:00AM
CS_DATES_HMS	hh:mm:ss 17:36:00	hh:mm:ss 00:00:00	hh:mm:ss 17:36:00
CS_DATES_HMS_ALT	hh:mm:ss 17:36:32	hh:mm:ss 00:00:00	hh:mm:ss 17:36:32
CS_DATES_HMSZA	hh:mm:ss:zzz[AM|PM] 5:36:00:000PM	hh:mm:ss:zzz[AM|PM] 12:00:00:000AM	hh:mm:ss:zzz[AM|PM] 5:36:00:000PM
CS_DATES_HMSZ	hh:mm:ss:zzz 17:36:00:000	hh:mm:ss:zzz 00:00:00:000	hh:mm:ss:zzz 17:36:00:000
CS_DATES_LONG	mon dd yyyy hh:mm:ss:zzz [AM|PM] Aug 24 2009 05:36:00:000PM	mon dd yyyy Aug 24 2009	hh:mm:ss:zzz[AM|PM] 05:36:00:000PM
CS_DATES_LONG_ALT	mon dd yyyy hh:mm:ss:zzz [AM|PM] Aug 24 2009 05:36:00:000PM	mon dd yyyy hh:mm:ss:zzz [AM|PM] Aug 24 2009 12:00:00:000 AM	mon dd yyyy hh:mm:ss:zzz [AM|PM] Jan 01 1900 05:36:00:000 PM
CS_DATES_MDYHMS	mon dd yyyy hh:mm:ss Aug 24 2009 17:36:00	mon dd yyyy Aug 24 2009	hh:mm:ss 17:36:00
CS_DATES_MDYHMS_ALT	mon dd yyyy hh:mm:ss Aug 24 2009 17:36:00	mon dd yyyy hh:mm:ss Aug 24 2009 00:00:00	mon dd yyyy hh:mm:ss Jan 1 1900 17:36:00
CS_DATES_SHORT	mon dd yyyy hh:mm [AM|PM] Aug 24 2009 17:36:00	mon dd yyyy Aug 24 2009	hh:mm[AM|PM] 17:36:00AM
CS_DATES_SHORT_ALT	mon dd yyyy hh:mm [AM|PM] Aug 24 2009 17:36:00	mon dd yyyy hh:mm [AM|PM] Aug 24 2009 12:00AM	mon dd yyyy hh:mm [AM|PM] Jan 1 1900 17:36:00
CS_DATES_DMY1	dd/mm/yy 24/08/09	dd/mm/yy 24/08/09	
CS_DATES_DMY1_YYYY	dd/mm/yyyy 24/08/2009	dd/mm/yyyy 24/08/2009	

記号値	CS_DATETIME を変換した CS_CHAR (例：Aug 24 2009 5:36PM)	CS_DATE から変換された CS_CHAR (例：Aug 24 2009)	CS_TIME を変換した CS_CHAR (例：5:36PM)
CS_DATES_DYM1	dd/yy/mm 24/09/08	dd/yy/mm 24/09/08	
CS_DATES_DYM1_YYYY	dd/yyyy/mm 24/2009/08	dd/yy/mm 24/2009/08	
CS_DATES_MDY1	mm/dd/yy 08/24/09	mm/dd/yy 08/24/09	
CS_DATES_MDY1_YYYY	mm/dd/yyyy 08/24/2009	mm/dd/yyyy 08/24/2009	
CS_DATES_MYD1	mm/yy/dd 08/09/24	mm/yy/dd 08/2009/24	
CS_DATES_MYD1_YYYY	mm/yyyy/dd 08/2009/24	mm/yyyy/dd 08/2009/24	
CS_DATES_YDM1	yy/dd/mm 09/24/08	yy/dd/mm 09/24/08	
CS_DATES_YDM1_YYYY	yyyy/dd/mm 2009/24/08	yyyy/dd/mm 2009/24/08	
CS_DATES_YMD1	yy.mm.dd 09.08.24	yy.mm.dd 09.08.24	
CS_DATES_YMD1_YYYY	yyyy.mm.dd 2009.08.24	yyyy.mm.dd 2009.08.24	
CS_DATES_DMY2	dd.mm.yy 24.08.09	dd.mm.yy 24.08.09	
CS_DATES_DMY2_YYYY	dd.mm.yyyy 24.08.2009	dd.mm.yyyy 24.08.2009	
CS_DATES_MDY2	mon dd, yy Aug 24,09	mon dd, yy Aug 24,09	
CS_DATES_MDY2_YYYY	mon dd, yyyy Aug 24.2009	mon dd, yyyy Aug 24.2009	
CS_DATES_YMD2	yy/mm/dd 09/08/24	yy/mm/dd 09/08/24	
CS_DATES_YMD2_YYYY	yyyy/mm/dd 2009/08/24	yyyy/mm/dd 2009/08/24	
CS_DATES_DMY3	dd-mm-yy 24-08-09	dd-mm-yy 24-08-09	

記号値	CS_DATETIME を変換した CS_CHAR (例：Aug 24 2009 5:36PM)	CS_DATE から変換された CS_CHAR (例：Aug 24 2009)	CS_TIME を変換した CS_CHAR (例：5:36PM)
CS_DATES_DMY3_YYYY	dd-mm-yyyy 24-08-2009	dd-mm-yyyy 24-08-2009	
CS_DATES_MDY3	mm-dd-yy 08-24-09	mm-dd-yy 08-24-09	
CS_DATES_MDY3_YYYY	mm-dd-yyyy 08-24-2009	mm-dd-yyyy 08-24-2009	
CS_DATES_YMD3	yymmdd 090824	yymmdd 090824	
CS_DATES_YMD3_YYYY	yyyymmdd 20090824	yyyymmdd 20090824	
CS_DATES_YMDTHMS 23	yyyy-mm-ddThh:mm:ss 2009-08-24T17:36:00	yyyy-mm-dd 2009-08-24	hh:mm:ss 17:36:00
CS_DATES_DMY4	dd mon yy 24 Aug 09	dd mon yy 24 Aug 09	
CS_DATES_DMY4_YYYY	dd mon yyyy 24 Aug 2009	dd mon yyyy 24 Aug 2009	

- 表 2-9 は、CS_CHAR と、CS_BIGDATETIME および CS_BIGTIME との変換において、*type* が CS_DT_CONVFMT の場合に **buffer* に対して有効な値の一覧です。

表 2-9：type が CS_DT_CONVFMT (cs_dt_info) の場合の *buffer の値

記号値	CS_BIGDATETIME から変換された CS_CHAR (例：Aug 24 2009 5:36PM)	CS_BIGTIME から変換された CS_CHAR (例：5:36PM)
CS_DATES_HMSUSA、または CS_DATES_HMSUSA_YYYY	hh:mm:ss.zzzzzz[AM|PM] `5:36:00.000000PM`	hh:mm:ss.zzzz[AM|PM] `5:36:00.000000PM`
CS_DATES_HMSUS、または CS_DATES_HMSUS_YYYY	hh:mm:ss.zzzzzz `17:36:00.000000`	hh:mm:ss.zzzzzz `17:36:00.000000`
CS_DATES_LONGUSA	mon dd yy hh:mm:ss.zzzzzz[AM|PM] `Aug 24 09 5:36:00.000000PM`	mon dd yy hh:mm:ss.zzzzzz[AM|PM] `Jan 1 01 5:36:00.000000PM`
CS_DATES_LONGUSA_YYYY	mon dd yyyy hh:mm:ss.zzzzzz[AM|PM] `Aug 24 2009 5:36:00.000000PM`	mon dd yyyy hh:mm:ss.zzzzzz[AM|PM] `Jan 1 0001 5:36:00.000000PM`
CS_DATES_LONGUS	mon dd yy hh:mm:ss.zzzzzz `Aug 24 09 17:36:00.000000`	mon dd yy hh:mm:ss.zzzzzz `Jan 1 01 17:36:00.000000`
CS_DATES_LONGUS_YYYY	mon dd yyyy hh:mm:ss.zzzzzz `Aug 24 2009 17:36:00.000000`	mon dd yy hh:mm:ss.zzzzzz `Jan 0 0001 17:36:00.000000`
CS_DATES_YMDHMSUS_YYYY	yyyy-mm-dd hh:mm:ss.zzzzzz `2009-08-24 17:36:00.000000`	yyyy-mm-dd hh:mm:ss.zzzzzz `0001-01-01 17:36:00.000000`

- cs_locale(CS_SET,CS_LC_TIME) 呼び出しまたは cs_locale (CS_SET、CS_LC_ALL) 呼び出しは、指定された各国言語用のデフォルト設定に日時変換情報をリセットします。

参照

cs_dt_crack, cs_locale

cs_loc_alloc

説明

CS_LOCALE 構造体を割り付けます。

構文

CS_RETCODE cs_loc_alloc(context, loc_pointer)

```
CS_CONTENT   *context;
CS_LOCALE    **loc_pointer;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

loc_pointer

ポインタ変数のアドレスです。cs_loc_alloc は、**loc_pointer* に、新しく割り付けた CS_LOCALE 構造体のアドレスを設定します。

戻り値

cs_loc_alloc は次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_loc_alloc の失敗の主な理由は、メモリ不足です。

使用法

- Open Client/Open Server アプリケーションは、CS_LOCALE 構造体を使用して、コンテキスト、スレッド、接続、またはデータ要素に対して、カスタム・ローカライゼーション値を定義できます。カスタム・ローカライゼーション値を定義するため、アプリケーションは次の処理を実行します。
  - cs_loc_alloc を呼び出して CS_LOCALE 構造体を割り付けます。
  - cs_locale (CS_SET) を呼び出して、カスタム値を持つ CS_LOCALE をロードします。
  - CS_LOCALE を使用してコンテキストまたは接続に対して CS_LOC_PROP プロパティを設定します。srv_thread_props を呼び出し、スレッドに対して SRV_T_LOCALE プロパティを設定します。プログラム変数を記述する CS_DATAFMT 構造体内の CS_LOCALE を使用します。または、Open Client/Open Server ルーチンへのパラメータとして CS_LOCALE を使用します。
  - cs_loc_drop を呼び出し、CS_LOCALE の割り当てを解除します。

- ローカライゼーション値は次のものを定義します。
  - Open Client/Open Server および Adaptive Server Enterprise のメッセージに使用する言語および文字セット。
  - 日付および時刻の表現方法
  - データと文字データ型間で変換するときに使用する文字セット
  - cs_strcmp によって使用されるソート順を定義するために使用する照合順

参照

cs_ctx_alloc, cs_loc_drop, cs_locale

cs_loc_drop

説明

CS_LOCALE 構造体の割り付けを解除します。

構文

```
CS_RETCODE cs_loc_drop(context, locale)

CS_CONTEXT    *context;
CS_LOCALE     *locale;
```

パラメータ

context

CS_LOCALE が割り付けられているコンテキストを表す CS_CONTEXT 構造体を指すポインタです。

locale

CS_LOCALE 構造体を指すポインタです。

戻り値

cs_loc_drop は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

使用法

- CS_LOCALE 構造体には、ローカライゼーション情報が含まれています。
- CS_LOCALE 構造体の割り付けが解除されると、この構造体を再び使用することはできません。新しい CS_LOCALE 構造体を割り付けるには、アプリケーションは cs_loc_alloc を呼び出します。
- アプリケーションは、まだ使用中の CS_LOCALE の割り付けを解除しないように注意してください。CS_DATAFMT 構造体から参照されている CS_LOCALE は、使用中と見なされます。

- アプリケーションは、cs_config または ct_con_props を呼び出して、コンテキストまたは接続に対する CS_LOC_PROP プロパティを設定した後、CS_LOCALE 構造体の割り付けを解除できます。これは、cs_config および ct_con_props が、ユーザが指定した CS_LOCALE 構造体への直接参照を設定するのではなく、CS_LOCALE 構造体の情報をコピーするためです。

参照　cs_loc_alloc, cs_locale

cs_locale

説明

CS_LOCALE 構造体へのローカライゼーション値のロード、または以前に CS_LOCALE 構造体のロードに使用したロケール名の取得を行います。

構文

```
CS_RETCODE cs_locale(context, action, locale, type,
        buffer, buflen, outlen)

CS_CONTEXT    *context;
CS_INT        action;
CS_LOCALE     *locale;
CS_INT        type;
CS_CHAR       *buffer;
CS_INT buflen;
CS_INT        *outlen;
```

パラメータ

context

CS_LOCALE が割り付けられているコンテキストを表す CS_CONTEXT 構造体を指すポインタです。

action

次の記号値のいずれかです。

action の値	cs_locale
CS_SET	新しいローカライゼーション値を持つ CS_LOCALE をロードする。
CS_GET	CS_LOCALE をロードするために使用したロケール名を取得する。

locale

CS_LOCALE 構造体を指すポインタです。*action* が CS_SET の場合、cs_locale はこの構造体を変更します。action が CS_GET の場合、cs_locale は構造体を調べ、前に CS_LOCALE をロードするために使用したロケール名を調べます。

type

次の記号値のいずれかです。

type の値	意味
CS_LC_ALL	ローカライゼーション情報のすべてのタイプ。 **注意** CS_LC_ALL は「設定専用」のタイプである。つまり、*type* が CS_LC_ALL の場合、*action* は CS_SET でなければならない。
CS_LC_COLLATE	照合順 (「ソート順」とも呼ぶ)。Open Client が文字データをソートや比較するときに使用する。
CS_LC_CTYPE	文字セット。文字データ型に、または文字データ型から変換するとき、Open Client は文字セットを使用する。
CS_LC_MESSAGE	Open Client/Open Server および Adaptive Server Enterprise のエラー・メッセージに使用する言語および文字セット。
CS_LC_TIME	日時と文字型との間で変換するときに使用する言語および文字セット。CS_LC_TIME は、月名およびその省略形、日付部分の順序、および am/pm 文字列を使用するかを制御する。
CS_SYB_LANG、 CS_SYB_CHARSET、 CS_SYB_SORTORDER、 CS_SYB_LANG_CHARSET	これらの値の詳細については、「cs_locale での言語名、文字セット名、ソート順名の使用」(68 ページ) を参照。

警告！ Open Server アプリケーション・プログラマは、Open Server アプリケーション全体に適用する CS_LOCALE 構造体を設定する場合には、type を CS_LC_ALL に設定する必要があります。

buffer

action が *CS_SET* の場合、*buffer* は、ロケール名、文字セット名、言語名、ソート順名、または言語セットと文字セットの組み合わせを表す文字列を指します。

action が *CS_GET* の場合、*buffer* は、cs_locale がロケール名、文字セット名、言語名、ソート順名、または言語セットと文字セットの組み合わせを格納する領域を指します。出力については、すべての名前が null で終了します。バッファには、名前と null ターミネータのための十分な長さが必要です。

buflen

**buffer* の長さ (バイト単位)

action が CS_SET で、**buffer* の値が NULL で終了している場合には、*buflen* に CS_NULLTERM を設定して渡します。

outlen

整数変数を指すポインタです。

action が CS_SET の場合、*outlen* は使用されません。

action が CS_GET のときに *outlen* を指定すると、cs_locale は **outlen* をロケール名の長さ (バイト単位) に設定します。

名前が *buflen* バイトよりも大きい場合、アプリケーションは **outlen* の値を使用して、名前を保持するために必要なバイト数を判断できます。

action が CS_SET の場合、またはアプリケーションが戻り値の長さ情報を必要としない場合には、*outlen* に NULL を設定して渡すことができます。

戻り値

cs_locale は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_locale の失敗の主な理由は、次のとおりです。

- *action* が CS_SET の場合、**buffer* ロケール名が Sybase ロケール・ファイルにない。
- *action* が CS_GET であり、*buflen* によって **buffer* データ領域が小さすぎることが示された。
- ローカライゼーション・ファイルが見つかりません。

使用法

注意　cs_locale の動作は、プラットフォーム固有の設定によって異なります。『Open Client/Server 設定ガイド Windows 版』または『Open Client/Server 設定ガイド UNIX 版』のローカライゼーションの章を読み、Client-Library のローカライゼーション・メカニズムを十分に理解してください。ローカライゼーションに関するプログラミングの問題については、『Open Client/Open Server 開発者用国際化ガイド』を参照してください。

- cs_locale(CS_SET) は、CS_LOCALE 構造体にローカライゼーション値をロードします。cs_locale(CS_GET) は、CS_LOCALE 構造体から現在の設定を取得します。
- 「ロケール名」は、言語／文字セット／ソート順の組み合わせを表す文字列です。たとえば、ロケール名「fr」は言語／文字セット／ソート順の組み合わせで French、iso_1、binary を表します。
  - Sybase では、デフォルトのロケール・ファイル内にいくつかのロケール名をあらかじめ定義してあります。
  - システム管理者は新しいロケール名を定義して、これらのロケール名を Sybase ロケール・ファイルに追加できます。ロケール名の追加方法については、『Open Client/Server 設定ガイド Windows 版』または『Open Client/Server 設定ガイド UNIX 版』を参照してください。
- 『Open Client/Server 開発者用国際化ガイド』を参照してください。

CS_LOCALE 構造体のロード

- アプリケーションは、CS_LOCALE を使用して、コンテキスト、接続、またはデータ要素に対してカスタム・ローカライゼーション値を定義する前に、CS_LOCALE を初期化または「ロード」する必要があります。
- cs_locale(CS_SET) はローカライゼーション値を使用して CS_LOCALE 構造体をロードします。ローカライゼーション値はロケール名を与えることで指定できます。文字セット、言語、ソート順を名前で直接指定することもできます。
- ロケール名を指定する場合、*buffer* には Sybase ロケール・ファイルにあるエントリに対応する名前を指定する必要があります。

  デフォルトのロケール名を指定するときには、*buffer* に NULL を設定して渡すこともできます。この場合、cs_locale はオペレーティング・システムで使用するロケール名を取得します。オペレーティング・システム環境で適切なロケール名が見つからない場合は、cs_locale はプラットフォーム固有のデフォルトのロケール名を使用します。

  対象となるローカライゼーション項目は、ロケール・ファイル・エントリの設定をベースにしてロードされます。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 設定ガイド』を参照してください。

- 文字セット名、言語名、およびソート順名を直接指定する方法については、「cs_locale での言語名、文字セット名、ソート順名の使用」(68 ページ) を参照してください。
- CS_LOCALE にカスタム値をロードした後に、アプリケーションは次の操作が可能です。
  - *property* に CS_LOC_PROP を設定して cs_config を呼び出し、カスタム・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します。
  - *property* に CS_LOC_PROP を設定して ct_con_props を呼び出し、カスタム・ローカライゼーション値を接続構造体にコピーします。
  - カスタム・ローカライゼーション値を受け入れるルーチン (cs_dt_info、cs_strcmp、cs_time) へのパラメータとして CS_LOCALE を指定します。
  - 対象プログラム変数 (cs_convert、ct_bind) を記述する CS_DATAFMT 構造体に、CS_LOCALE を含めます。
- cs_config はロケール情報をコピーするので、アプリケーションは、cs_config を呼び出して CS_LOC_PROP プロパティを設定した後に、CS_LOCALE 構造体の割り付けを解除できます。同様に、アプリケーションは ct_con_props を呼び出して CS_LOC_PROP プロパティを設定した後に、CS_LOCALE の割り付けを解除できます。ただし、CS_DATAFMT 構造体が CS_LOCALE を使用している場合には、CS_DATAFMT が CS_LOCALE の参照をやめるまで、アプリケーションで CS_LOCALE の割り付けを解除させないでください。
- ロケール名が初めて参照されるときに、ロケール名が指定する言語、文字セット、ソート順に対するすべてのローカライゼーション情報が環境から読み込まれ、**context* に保存されます。ロケール名が再度参照されると、cs_locale は環境からではなく CS_CONTEXT から情報を読み込みます。

ロケール名の取得

- アプリケーションで、CS_LOCALE をロードするために使用したロケール名を取得するには、対象となるローカライゼーション情報の型を *type* に設定し、さらに CS_LOCALE 構造体を指すポインタを *locale* に設定して cs_locale(CS_GET) を呼び出します。
- cs_locale は、**buffer* に、CS_LOCALE をロードするために使用したロケール名を表す NULL 終了文字列を設定します。

cs_locale での言語名、文字セット名、ソート順名の使用

- アプリケーションは、ロケール名の代わりに、言語名、文字セット名、ソート順を使用して cs_locale を呼び出すことができます。
- 言語名、文字セット名、ソート順名を使用するためには、*type* を CS_SYB_LANG、CS_SYB_CHARSET、CS_SYB_SORTORDER、または CS_SYB_LANG_CHARSET に設定して cs_locale を呼び出します。次の表は、これらの *type* の値に対する cs_locale のパラメータの一覧です。

表 2-10 : cs_locale での言語名、文字セット名、ソート順名の使用

type の値	action の値	buffer の値	cs_locale
CS_SYB_LANG	CS_SET	言語名を指すポインタ	指定された言語情報を使用して CS_LOCALE をロードする。
	CS_GET	データ領域を指すポインタ	現在の言語名を **buffer* に指定する。名前は null で終了する。
CS_SYB_CHARSET	CS_SET	文字セット名を指すポインタ	指定された文字セット情報を使用して CS_LOCALE をロードする。
	CS_GET	データ領域を指すポインタ	現在の文字セット名を **buffer* に指定する。名前は null で終了する。
CS_SYB_SORTORDER	CS_SET	ソート順名を指すポインタ	指定されたソート順情報を使用して CS_LOCALE をロードする。
	CS_GET	データ領域を指すポインタ	現在のソート順名を **buffer* に指定する。名前は null で終了する。
CS_SYB_LANG_CHARSET	CS_SET	*language_name*.character_*set_name* という形式の文字列を指すポインタ	指定された言語情報および文字セット情報を使用して CS_LOCALE をロードする。
	CS_GET	データ領域を指すポインタ	*language_name.character_set_ name* という形式の文字列を **buffer* に指定する。名前は null で終了する。

- アプリケーションは、*type* を CS_LC_ALL に設定して cs_locale を呼び出すことによって、一貫した情報を CS_LOCALE 構造体にロードしておく必要があります。
- アプリケーションが言語名だけを指定した場合、cs_locale は、事前にロードされた CS_LOCALE 構造体で指定されている文字セットとソート順を使用します。

  アプリケーションが文字セット名だけを指定した場合、cs_locale は、事前にロードされた CS_LOCALE 構造体で指定されている言語とソート順を使用します。

アプリケーションがソート順名だけを指定した場合、cs_locale は、事前にロードされた CS_LOCALE 構造体で指定されている言語と文字セットを使用します。

言語、文字セット、およびソート順の組み合わせが無効な場合、cs_locale は CS_FAIL を返します。

- 有効な言語名は、*$SYBASE/locales* ディレクトリ内のサブディレクトリに対応します。有効な文字セット名は、*$SYBASE/charsets* ディレクトリ内のサブディレクトリに対応します。文字セットに有効なソート順名は、*$SYBASE/charsets/character_set_name* ディレクトリ内のファイル名から、サフィックスを除いた名前に対応します。
- 要求された言語または文字セットに対するローカライゼーション・ファイルが存在しないと、cs_locale は CS_FAIL を返します。

参照　cs_loc_alloc, cs_loc_drop

cs_locator

説明　プリフェッチされたデータ、サーバ内の LOB の全長、ロケータのポインタの文字表現などの情報を CS_LOCATOR 変数から取得します。

構文

```
CS_RETCODE cs_locator(ctx, action, locator, type,
                buffer, buflen, outlen)

CS_CONTEXT      *ctx;
CS_INT           action;
CS_LOCATOR      *locator;
CS_INT           type;
CS_VOID         *buffer;
CS_INT           buflen;
CS_INT          *outlen;
```

パラメータ

ctx

CS_CONTEXT 構造体を指すポインタです。

action

情報を設定するのか取得するのかを指定します。現時点で実行可能な唯一のアクションは CS_GET です。

locator

ロケータ変数を指すポインタ。

type

取得または設定する情報の種類。記号値は次のとおりです。

値	対処法	*buffer が指す対象	説明
CS_LCTR_LOBLEN	CS_GET	CS_BIGINT	サーバ内の LOB データの全長を取得する。
CS_LCTR_LOCATOR	CS_GET	CS_CHAR	ロケータ値を文字列として取得する。
CS_LCTR_PREFETCHLEN	CS_GET	CS_INT	ロケータ変数に含まれている、プリフェッチされた LOBデータの長さを取得する。
CS_LCTR_PREFETCHDATA	CS_GET	CS_CHAR	ロケータ変数に含まれている、プリフェッチされた LOB データを取得する。
CS_LCTR_DATATYPE	CS_GET	CS_INT	ロケータの型を取得する。有効な戻り値の型は、CS_TEXTLOCATOR_TYPE、CS_IMAGELOCATOR_TYPE、および CS_UNITEXTLOCATOR_TYPE。

buffer

データの格納先変数を指すポインタ。文字データは NULL で終了します。

buflen

**buffer* のバイト単位の長さ。

outlen

CS_INT 変数を指すポインタです。*outlen* が NULL 以外の場合、cs_locator() は **outlen* を、**buffer* に配置されたデータのバイト単位の長さに設定します。返されたデータが文字データ (プリフェッチされたデータやロケータ文字列など) の場合、**outlen* に返される長さには、NULL ターミネータが含まれます。cs_locator() が CS_TRUNCATED を返し、*outlen* が NULL でない場合、cs_locator() は必要なバッファ・サイズを **outlen* に返します。

戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_TRUNCATED	バッファが小さすぎるために結果がトランケートされている。
CS_FAIL	ルーチンが失敗した。

例

トランケートする必要のあるテキスト値の LOB ロケータを取得します。他のコード例については、『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_LOCATOR *lobloc;
CS_INT     prefetchsize;
CS_BOOL    boolval;
CS_INT     start, length;
CS_INT     outlen;
CS_CHAR    charbuf[1024];
CS_BIGINT  totallen;
...

/*
** Turn on option CS_LOBLOCATOR first and set the prefetchsize to 100.
*/

boolval = CS_TRUE;
ct_options(conn, CS_SET, CS_OPT_LOBLOCATOR, &boolval, CS_UNUSED, NULL);
prefetchsize = 100;
ct_options(conn, CS_SET, CS_OPT_LOBPREFETCHSIZE, ¥
    &prefetchsize, CS_UNUSED, NULL);

/*
** Allocate memory for the CS_LOCATOR.
*/
cs_locator_alloc(ctx, &lobloc);

/*
** Open a transaction and get the locator.The locator is only valid within a
** transaction.
*/
sprintf(cmdbuf, "begin transaction ¥
   select au_id, copy from pubs2..blurbs where au_id ¥
   like ‘486-29-%’ ");
ct_command(cmd, CS_LANG_CMD, cmdbuf , CS_NULLTERM, CS_UNUSED);
ct_send(cmd);

/*
** Process results.
*/
while ((results_ret = ct_results(...)) == CS_SUCCEED)
{
   ...

}
   /*
   ** Bind the locator and fetch it.
   */
   strcpy(prmfmt.name, "@locatorparam");
   prmfmt.namelen = CS_NULLTERM;
```

```
    prmfmt.datatype = CS_TEXTLOCATOR_TYPE;
    prmfmt.maxlength = CS_UNUSED;
    ...

    ct_bind(cmd, 1, &fmt, lobloc, NULL, &indicator);
    ct_fetch(cmd, CS_UNUSED, CS_UNUSED, CS_UNUSED, &count);
}

/*
** Use the cs_locator() routine to retrieve data from the fetched locator.
** Get the prefetch length and the prefetch data.
*/
cs_locator(ctx, CS_GET, lobloc, CS_LCTR_PREFETCHLEN, ¥
    (CS_VOID *)&prefetchsize, sizeof(CS_INT), &outlen);

cs_locator(ctx, CS_GET, lobloc, CS_LCTR_PREFETCHDATA, ¥
    (CS_VOID *)charbuf, sizeof(charbuf), &outlen);

/*
** Retrieve the total length of the LOB data in the server for this
** locator.
*/
cs_locator(ctx, CS_GET, lobloc, CS_LCTR_LOBLEN, ¥
    (CS_VOID *)&totallen,
    sizeof(totallen), &outlen);

/*
** Use the retrieved locator to perform an action to the LOB, pointed to by
** this locator in the server.
**
** Get a substring from the text in the server, using a parameterized
** language command.
*/
start = 10;
length = 20;
sprintf(cmdbuf, "select return_lob(text, substring(@locatorparam, ¥
    start, length))");
ct_command(cmd, CS_LANG_CMD, cmdbuf, CS_NULLTERM, CS_UNUSED);

/*
** Set the format structure and call ct_param()
*/
strcpy(prmfmt.name, "@locatorparam");
prmfmt.namelen = CS_NULLTERM;
prmfmt.datatype = CS_TEXTLOCATOR_TYPE;
prmfmt.format = CS_FMT_UNUSED;
```

```
    prmfmt.maxlength = CS_UNUSED;
    prmfmt.status = CS_INPUTVALUE;

    indicator = 0;
    ct_param(cmd, &prmfmt, (CS_VOID *)lobloc, CS_UNUSED, indicator);

    /*
    ** Send the locator commands to the server.
    */
    ct_send(cmd);

    /*
    ** Process results.
    */
    while ((results_ret = ct_results(...)) == CS_SUCCEED)
    {
        ...
    }

    /*
    ** Truncate the text to 20 bytes and commit the transaction.
    */
    sprintf(cmdbuf, "truncate lob @locatorparam (length) ¥

        commit transaction");
    ct_command(cmd, CS_LANG_CMD, cmdbuf, CS_NULLTERM, CS_UNUSED);
    ct_param(cmd, &prmfmt, (CS_VOID *)lobloc, CS_UNUSED, indicator);

    ct_send(cmd);

    /*
    ** Process results.
    */
    while ((results_ret = ct_results(...)) == CS_SUCCEED)
    {
        ...
    }

    /*
    ** The transaction is closed, deallocate the locator.
    */
    cs_locator_drop(ctx, lobloc);
```

参照　　cs_locator_alloc, cs_locator_drop

cs_locator_alloc

説明

CS_LOCATOR データ型構造体を割り付けます。

構文

CS_RETCODE cs_locator_alloc(ctx, locator)

```
CS_CONTEXT *ctx;
CS_LOCATOR **locator;
```

パラメータ

ctx

CS_CONTEXT 構造体を指すポインタです。

locator

割り付けられるロケータ変数のアドレス。**locator* を、新たに割り付けられた CS_LOCATOR 構造体のアドレスに設定します。

戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

参照

cs_locator, cs_locator_drop

cs_locator_drop

説明

CS_LOCATOR データ型構造体の割り付けを解除します。

構文

CS_RETCODE cs_locator_drop(ctx, locator)

```
CS_CONTEXT *ctx;
CS_LOCATOR *locator;
```

パラメータ

ctx

CS_CONTEXT 構造体を指すポインタです。

locator

割り付け解除されるロケータ変数を指すポインタ。

戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

参照

cs_locator, cs_locator_alloc

cs_manage_convert

説明

ユーザ定義の文字セット変換ルーチンをインストールまたは取得します。

構文

```
CS_RETCODE cs_manage_convert(context, action,
        srctype, srcname, srcnamelen,
        desttype, destname, destnamelen,
        conv_multiplier, func)

CS_CONTEXT      *context;
CS_INT          action;
CS_INT          srctype;
CS_CHAR         *srcname;
CS_INT          srcnamelen;
CS_INT          desttype;
CS_CHAR         *destname;
CS_INT          destnamelen;
CS_INT          *conv_multiplier;
CS_CONV_FUNC    *func;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

action

次の記号値のいずれかです。

action の値	**cs_manage_convert**
CS_SET	指定されたデータ型と文字セット名との変換に対して、変換ルーチンと変換乗算子をインストールする。
CS_GET	指定されたデータ型と文字セット名に対する現在の変換ルーチンと変換乗算子を取得する。
CS_CLEAR	指定されたデータ型と文字セット名に対する CS-Library のデフォルト変換ルーチンで現在の変換ルーチンを置き換え、クリアする。

srctype

変換元データのデータ型です。現在のリリースでは、*srctype* を CS_CHAR_TYPE に設定する必要があります。

srcname

srctype に対応する文字セット名です。この文字セット名は Sybase インストール・ディレクトリの *charsets* サブディレクトリ内にあるサブディレクトリ名に対応している必要があります。

srcnamelen

srcname のバイト単位の長さです。*srcname* が null で終了している場合は、*srcnamelen* を CS_NULLTERM として渡すことができます。

desttype

変換先データのデータ型です。現在のリリースでは、*desttype* を CS_CHAR_TYPE に設定する必要があります。

destname

変換先文字セットの名前です。この文字セット名は Sybase インストール・ディレクトリの *charsets* サブディレクトリ内にあるサブディレクトリ名に対応している必要があります。

destnamelen

destname の長さ (バイト単位) です。*destname* が null で終了している場合は、*destnamelen* を CS_NULLTERM にして渡すことができます。

conv_multiplier

CS_INT 変数のアドレスです。action が CS_SET の場合、指定された文字セット変換の変換乗算子として **conv_multiplier* を渡します。action が CS_GET の場合、**conv_multiplier* は指定された文字セット変換の変換乗算子を受け取ります。action が CS_CLEAR の場合、*conv_multiplier* を NULL にして渡します。

アプリケーションでのこの数値の用途については、「変換乗算子の意味」(79 ページ) を参照してください。

func

CS_CONV_FUNC 変数のアドレスです。この変数自体が文字セット変換ルーチンを指すポインタです。カスタム文字セット変換ルーチンのコーディングに関する要件については「カスタム文字セット変換ルーチンの定義」(79 ページ) を参照してください。

変換ルーチンがインストールされる場合、**func* はインストールされる変換ルーチンを指します。

変換ルーチンが取得される場合、cs_manage_convert は、*srcname* から *destname* への変換用に現在インストールされている文字セット変換ルーチンを指すか、カスタム・ルーチンがインストールされていない場合には NULL を指すように **func* を設定します。

変換ルーチンをクリアする場合には、**func* を NULL として渡してください。

注意　*func* は関数を示すポインタを指すポインタを表します。このパラメータの受け渡しについて特別な条件があります。「カスタム文字セット変換ルーチンのインストール」(81 ページ) のコード例を参照してください。

戻り値

cs_manage_convert は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_manage_convert の失敗の主な理由は、無効なパラメータの設定です。

使用法

- cs_manage_convert を使用すると、アプリケーションは、データを別の文字セットに変換するカスタム文字セット変換ルーチンをインストールできます。

文字セット変換

- Client-Library、CS-Library、Server-Library は、すべて文字セット変換を実行できます。文字セット変換は、アプリケーションが 2 つの文字データ型で変換を行うときに、変換元と変換先の文字セットが異なる場合に発生します。
  - CS-Library では、cs_convert が 2 つの文字データ型で変換を行うときに、*destfmt* の CS_DATAFMT 構造体が *srcfmt* の CS_DATAFMT 構造体とは異なるロケールを指定している (または異なるロケールをデフォルトとして使用している) 場合に、文字セット変換が実行されます。

  - Client-Library では、アプリケーションはカラムを文字データ型の変数にバインドして、接続のロケール (CS_LOC_PROP 接続プロパティ) とは異なる ct_bind の *datafmt* の CS_LOCALE を指すポインタを渡すことで、フェッチされた文字データ用の文字セット変換を要求できます。
  - Server-Library では、クライアントに渡された、またはクライアントから受け取ったすべての文字データは、クライアント・スレッドの文字セットと Open Server 文字セットの間で自動的に変換されます。

- 文字データ型は、CS_CHAR、CS_LONGCHAR、CS_TEXT、CS_UNITEXT、CS_UNICHAR、CS_VARCHAR、CS_XML です。
- cs_manage_convert に対し、アプリケーションは *srctype* と *desttype* を両方とも CS_CHAR_TYPE として渡す必要があります。ただし、CS-Library、Client-Library、Server-Library は、任意の 2 つの文字ベースの型の変換において、変換ロケールで指定されている文字セットが変換ルーチンに対応する文字セットならば、この変換ルーチンを呼び出します。
- カスタム変換ルーチンをインストールする主な理由は、間接的な変換を直接的な変換に置き換えることでパフォーマンスを改善させることです。

  カスタム文字セット変換ルーチンを使用すると、CS-Library が文字セットの直接変換を使用しない文字セット変換を実行するアプリケーションのパフォーマンスを向上できます。間接的な文字セット変換とは、まず、Unicode UTF-8 に変換し、次に Unicode UTF-8 から変換先文字セットに変換することです。この変換を実行するアプリケーションは、直接変換をサポートするカスタム・ルーチンをインストールすることでパフォーマンスを改善できます。

  たとえば、Open Server アプリケーションで ISO 8859-1 と EUC JIS の間の変換を実行するためのカスタム・ルーチンをインストールするとします。この直接変換は、Open Server で提供される間接変換 (ISO 8859-1 → Unicode UTF-8 → EUC JIS または ISO 8859-1 ← Unicode UTF-8 ← EUC JIS) よりも高速である場合があります。
- 特定の文字変換が直接か間接かを判断するには、変換元文字セットの変換設定ファイルの内容を確認します。変換先文字セット用のエントリがある場合は、その変換は直接変換です。文字セット設定ファイルについては、『Open Client/Open Server 開発者用国際化ガイド』で説明されています。
- 『Open Client/Server 開発者用国際化ガイド』を参照してください。

変換乗算子の意味

- アプリケーションは、指定された文字セット間での変換用に、cs_manage_convert に変換乗算子を指定する必要があります。
- 変換乗算子の値は、指定された文字セット間で変換を行うときに変換元の 1 バイトを変換できる変換先結果にある最大バイト値と同じです。
- アプリケーションは、cs_conv_mult を使って、指定した文字セット変換を行う変換乗算子を取得します。この数値によって、アプリケーションは変換に必要な変換先領域を判断できます。

カスタム文字セット変換ルーチンの定義

- カスタム文字セット変換ルーチンは次のように定義します。

```
CS_RETCODE CS_PUBLIC
 convfunc(context, srcfmt, srcdata,
          destfmt, destdata, destlen)
 CS_CONTEXT         *context;
 CS_DATAFMT         *srcfmt;
 CS_VOID            *srcdata;
 CS_DATAFMT         *destfmt;
 CS_VOID            *destdata;
 CS_INT             *destlen;
```

各パラメータの意味は次のとおりです。

- *context* は CS_CONTEXT 構造体へのポインタです。
- *srcfmt* は、変換元データを記述する CS_DATAFMT 構造体へのポインタです。*srcfmt–>maxlength* は、変換元データの実際の長さをバイト単位で記述します。
- *srcdata* は変換元データを指すポインタです。
- *destfmt* は、変換先データを記述する CS_DATAFMT 構造体へのポインタです。*destfmt–>maxlength* は、変換先データ領域の実際の長さをバイト単位で記述します。
- *destdata* は変換先データ領域を指すポインタです。

destlen は整数を指すポインタです。変換ルーチンでは、**destlen* を **destdata* に格納されているデータのバイト数に設定する必要があります。トランケートされた結果がルーチンによって書き込まれる場合は、トランケートする前のデータのバイト数を **destlen* に設定してください。

注意 CS_VARCHAR 構造体に変換する場合、変換ルーチンは CS_VARCHAR の *str* フィールドに書き込まれるバイト数を **destlen* と CS_VARCHAR の *len* フィールドの両方に設定します。

- cs_config は、カスタム変換ルーチン内から呼び出すことのできる唯一の CS-Library、Client-Library、または Server-Library の関数です。
- カスタム文字セット変換ルーチンが返す値を、表 2-11 に示します。
  - 変換ルーチンが表 2-11 の CS_SUCCEED 以外の値を返す場合は、アプリケーションは、示されたエラー条件に対応する Client-Library または CS-Library のメッセージを受け取ります。
  - 表 2-11 にない値を変換ルーチンが返す場合には、アプリケーションは Client-Library または CS-Library から "Unknown return code" というエラー・メッセージを受け取ります。

表 2-11：カスタム変換ルーチンの戻り値

戻り値	意味
CS_SUCCEED	変換に成功した。
CS_TRUNCATED	変換の結果がトランケートされた。
CS_MEM_ERROR	メモリの割り付け障害が発生した。
CS_EBADXLT	文字が変換できなかった。
CS_ENOXLT	要求された変換はサポートされていない。
CS_EDOMAIN	変換元の値がデータ型の正しい値のドメインの範囲外である。
CS_EDIVZERO	0 による除算はできない。
CS_EOVERFLOW	変換の結果がオーバフローした。
CS_EUNDERFLOW	変換の結果がアンダフローした。
CS_EPRECISION	変換の結果、精度が損なわれた。
CS_ESCALE	無効な位取り値が検出された。
CS_ESYNTAX	変換の結果、変換先の型に対して構文上正しくない値である。
CS_ESTYLE	スタイル・エラーのために変換オペレーションが停止した。

カスタム文字セット変換ルーチンのインストール

- 次に示すコードは、cs_manage_convert を呼び出してカスタム変換ルーチンをインストールする方法を示しています。このコードは、インストールされたルーチンが正しく定義されていることを前提としています (「カスタム文字セット変換ルーチンの定義」(79 ページ) を参照)。プログラム変数 *p_conv_func* は、変換ルーチンのアドレスを渡すために使用されます。

```
#define MULT_ISO_1_TO_EUCJIS 4
 CS_CONV_FUNC p_conv_func;
 CS_INT       conv_mult = MULT_ISO_1_TO_EUCJIS;

/*
 ** Install the routine charconv_iso_1_TO_eucjis() to convert
 ** character data from iso_1 character set to eucjis character
 ** set.
 */
 p_conv_func = charconv_iso_1_TO_eucjis;
 if (cs_manage_convert(context, CS_SET,
         CS_CHAR_TYPE, "iso_1", CS_NULLTERM,
         CS_CHAR_TYPE, "eucjis", CS_NULLTERM,
         &conv_mult, &p_conv_func )
     != CS_SUCCEED)
 {
   fprintf(stdout, "cs_manage_convert() failed!￥n");
   (CS_VOID)ct_exit(context, CS_FORCE_EXIT);
   (CS_VOID)cs_ctx_drop(context);
   exit(-1);
 }
```

参照　cs_conv_mult, cs_convert, cs_locale, cs_set_convert

cs_objects

説明　オブジェクトおよびオブジェクトに関連するデータを、保存、取得、クリアします。

構文

```
CS_RETCODE cs_objects(context, action, objname,
        objdata)

CS_CONTEXT      *context;
CS_INT          action;
CS_OBJNAME      *objname;
CS_OBJDATA      *objdata;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

action

次の記号値のいずれかです。

action の値	cs_objects
CS_SET	オブジェクトを保存する。
CS_GET	最初に見つかる一致オブジェクトを取得する。
CS_CLEAR	すべての一致オブジェクトをクリアする。

objname

オブジェクト名構造体を指すポインタです。**objname* は、対象となるオブジェクトに名前を付け、記述します。オブジェクト名構造体は次のように定義されます。

```
/*
 ** CS_OBJNAME
 */
 typedef struct _cs_objname
 {
   CS_BOOL          thinkexists;
   CS_INT           object_type;
   CS_CHAR          last_name[CS_MAX_CHAR];
   CS_INT           lnlen;
   CS_CHAR          first_name[CS_MAX_CHAR];
   CS_INT           fnlen;
   CS_VOID          *scope;
   CS_INT           scopelen;
   CS_VOID          *thread;
   CS_INT           threadlen;
 } CS_OBJNAME;
```

object_type、*last_name*、*first_name*、*scope*、および *thread* の 5 つのフィールドは、ストアド・オブジェクトを示すキーを構成します (「cs_objects のキーの命名」(86 ページ) を参照してください)。表 2-12 に CS_OBJNAME フィールドの説明を示します。

表 2-12 : CS_OBJNAME フィールド

フィールド	説明	注意
thinkexists	アプリケーションが、このオブジェクトが存在することを予期しているかどうかを示す。	*thinkexists* の値は cs_objects のリターン・コードに反映される。詳細は「戻り値」の項を参照。
object_type	オブジェクトのタイプ。	このフィールドは 5 つの部分から構成されるキーの最初の部分である。 *object_type* は、次の値のいずれかである。 • CS_CONNECTNAME • CS_CURSORNAME • CS_STATEMENTNAME • CS_CURRENT_CONNECTION • CS_WILDCARD (任意の値に一致する) • ユーザ定義値。ユーザ定義値は 100 以上。
last_name	対象となるオブジェクトに関連した姓があれば、その「姓」。	このフィールドは 5 つの部分から構成されるキーの 2 番目の部分である。
lnlen	*last_name* のバイト単位の長さ。	CS_NULLTERM は、NULL で終了する *last_name* を示す。 CS_UNUSED は、*last_name* の値が内部で使用されていないことを示す。 CS_GET および CS_CLEAR オペレーションでは、CS_WILDCARD はすべての *last_name* 値に一致する。
first_name	対象となるオブジェクトに関連した名前があれば、その「名前」。	このフィールドは 5 つの部分から構成されるキーの 3 番目の部分である。
fnlen	*first_name* のバイト単位の長さ。	CS_NULLTERM は、NULL で終了する *first_name* を示す。 CS_UNUSED は、*first_name* の値が内部で使用されていないことを示す。 CS_GET および CS_CLEAR オペレーションの場合、CS_WILDCARD はすべての *first_name* 値に一致する。

フィールド	説明	注意
scope	オブジェクトの対象範囲を記述するデータ。	このフィールドは 5 つの部分から構成されるキーの 4 番目の部分である。
scopelen	*scope* のバイト単位の長さ。	CS_NULLTERM は、NULL で終了する対象範囲データを示す。 CS_UNUSED は、**scope* の値が内部で使用されていないことを示す。 CS_GET および CS_CLEAR オペレーションの場合、CS_WILDCARD はすべての *scope* 値に一致する。
thread	マルチスレッド実行環境内のスレッドを区別するために使用される、プラットフォーム特有のデータ。	このフィールドは 5 つの部分から構成されるキーの 5 番目の部分である。
threadlen	*thread* のバイト単位の長さ。	CS_NULLTERM は、NULL で終了するスレッド・データを示す。 CS_UNUSED は、**thread* の値が内部で使用されていないことを示す。 CS_GET および CS_CLEAR オペレーションの場合、CS_WILDCARD はすべての *thread* 値に一致する。

objdata

オブジェクト・データ構造体を指すポインタです。***objdata* は、対象となるオブジェクトおよびそのオブジェクトに関連するすべてのデータです。オブジェクト・データ構造体は次のように定義されます。

```
/*
 ** CS_OBJDATA
 */
 typedef struct _cs_objdata
 {
   CS_BOOL           actuallyexists;
   CS_CONNECTION    *connection;
   CS_COMMAND        *command;
   CS_VOID           *buffer;
   CS_INT              buflen;
 } CS_NAMEDATA;
```

表 2-13 に、CS_OBJDATA フィールドの説明を示します。

表 2-13 : CS_OBJDATA フィールド

フィールド	説明	注意
actuallyexists	このオブジェクトが実際に存在するかどうかを示す	一致するオブジェクトが見つかった場合、cs_objects は *actuallyexists* を CS_TRUE に設定する。 一致するオブジェクトが見つからなかった場合、cs_objects は *actuallyexists* を CS_FALSE に設定する。
connection	オブジェクトが存在する接続を表す CS_CONNECTION 構造体を指すポインタ	
command	オブジェクトが関係しているコマンド領域を表す CS_COMMAND 構造体を指すポインタ	NULL の場合もある。
buffer	データ領域を指すポインタアプリケーションは、*buffer* を使用して、保存されたオブジェクトにデータを関連付けることができる	*action* が CS_SET の場合、**buffer* にはオブジェクトに関連付けるデータが含まれる。 *action* が CS_GET の場合、cs_objects は、取得されるオブジェクトに関連付けられるデータを **buffer* に設定する。
buflen	**buffer* の長さ (バイト単位)	*action* が CS_SET の場合、*buflen* は **buffer* に含まれるデータの長さである。CS_NULLTERM は、NULL で終了するデータを示す。CS_UNUSED は、保存されているオブジェクトに関連するデータが存在しないことを示す。 *action* が CS_GET の場合、*buflen* は **buffer* の最大容量である。cs_objects を実行すると、*buflen* の値は、**buffer* にコピーされたバイト数で上書きされる。*buflen* が CS_UNUSED の場合、cs_objects はデータの長さを *buflen* に上書きするが、**buffer* にはコピーしない。

戻り値

cs_objects は、*action* および *objname–>thinkexists* として渡される値によって CS_SUCCEED または CS_FAIL を返します (表 2-12 (83 ページ) を参照)。表 2-14 は各組み合わせに対する戻り値を示します。

表 2-14 : cs_objects の戻り値

cs_objects の呼び出し		cs_objects の戻り値		
action の値	objname → thinkexists の値	不一致	Last-name 一致	完全一致
CS_GET	CS_TRUE	CS_FAIL	CS_FAIL	CS_SUCCEED
CS_GET	CS_FALSE	CS_SUCCEED	CS_SUCCEED	CS_SUCCEED
CS_SET	CS_TRUE	CS_FAIL	CS_FAIL	CS_SUCCEED
CS_SET	CS_FALSE	CS_SUCCEED	CS_SUCCEED	CS_FAIL
CS_CLEAR	CS_TRUE	CS_FAIL	CS_FAIL	CS_SUCCEED
CS_CLEAR	CS_FALSE	CS_SUCCEED	CS_SUCCEED	CS_SUCCEED

使用法

表 2-15 : cs_objects パラメータの使用法の一覧

action の値	objname の値	objdata の値
CS_SET	オブジェクトの 5 つの部分から構成されるキー	保存するオブジェクト、およびそのオブジェクトとともに保存する追加データ
CS_GET	オブジェクトの 5 つの部分から構成されるキー	取得されたオブジェクトに設定
CS_CLEAR	オブジェクトの 5 つの部分から構成されるキー	CS_UNUSED

- cs_objects は、名前によって構造体およびデータ項目を取得する必要があるプリコンパイラ・アプリケーション内で有効です。

cs_objects のキーの命名

- cs_objects は、**objname* 構造体の *object_type*、*last_name*、*first_name*、*scope*、*thread* という 5 つのフィールドから構成されるキーを使用します。
  - CS_SET オペレーションでは、cs_objects はこのキーを使用して **objdata* オブジェクトを格納します。
  - CS_GET オペレーションでは、cs_objects はこのキーを使用してオブジェクト情報を取得し **objdata* に格納します。
  - CS_CLEAR オペレーションでは、cs_objects はキーに一致するオブジェクトをすべてクリアします。
- 表 2-16 は、cs_objects がキー・フィールドが一致するかどうかを調べるときの規則を説明します。

表 2-16：cs_objects のキー一致規則

***objname キー値**	**保管されたキー値が CS_UNUSED である場合**	**保管されたキー値が他の有効値である場合**
CS_WILDCARD	一致	一致
CS_UNUSED	一致	不一致
他の有効値	不一致	名前が一致し、相互に同じ長さの場合は一致

- cs_objects は、次の 2 とおりの一致を扱います。
  - キーの *last_name*、*scope*、および *thread* の部分が一致する「姓一致」
  - キーの 5 つの部分がすべて一致する「完全一致」

  action および *objname–>thinkexists* とともに cs_objects が扱う一致タイプにより、リターン・コードが決まります。
- CS_GET および CS_CLEAR オペレーションでは、1 つまたは複数の ***objname* キー・フィールドに対して CS_WILDCARD を指定できます。
  - CS_GET オペレーションでは、cs_objects は、検出された最初の一致オブジェクトが反映するように ***objdata* を設定します。
  - CS_CLEAR オペレーションでは、cs_objects はすべての一致オブジェクトをクリアします。

「現在の接続」オブジェクトの取得

- CS_CURRENT_CONNECTION オブジェクトがアプリケーションによってすでに保存されている場合、このアプリケーションは現在の接続を次のいずれかの方法で取得できます。
  - cs_objects を呼び出します。このとき、*objname–>object_type* に CS_CURRENT_CONNECTION を、*lnlen* に CS_UNUSED を、*fnlen* に CS_UNUSED を設定します。cs_objects は *objname* の *last_name* と *first_name* フィールドを無視し、*objdata –>buffer* に現在の接続の名前を、*objdata–>buflen* にこの接続名の長さを設定します。

- *objname–>object_type* に CS_CONNECTNAME を設定し、*objname–>last_name* と *objname–>lnlen* に新たに取得された接続名と接続名の長さを指定して、cs_objects を呼び出します。cs_objects は、*objdata* に取得された接続を設定します。

警告！アプリケーションは、完了コールバック・ルーチンの中から cs_objects(CS_SET) を呼び出すことはできません。

参照 cs_ctx_alloc

cs_prop_ssl_localid

説明 ローカル ID (証明書) ファイルへのパスを指定する。

構文

```
typedef struct _cs_sslid
{
      CS_CHAR     *identity_file;
      CS_CHAR     *identity_password;
} CS_SSLIDENTITY
```

パラメータ

identity_file

デジタル証明書およびそれに関連付けられたプライベート・キーを含むファイルへのパスを提供します。

CS_GET は、CS_CONNECTION で設定されている場合にのみ、使用されている *indentity_file* を返します。

identity_password

プライベート・キーを暗号化するのに使用されます。

cs_set_convert

説明 ユーザ定義の変換ルーチンをインストールまたは取得します。

構文

```
CS_RETCODE cs_set_convert(context, action, srctype,
        desttype, func)

CS_CONTEXT      *context;
CS_INT          action;
CS_INT          srctype;
CS_INT          desttype;
CS_CONV_FUNC    *func;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。CS_CONTEXT 構造体は Client-Library アプリケーション・コンテキストを定義します。

action

次の記号値のいずれかです。

action の値	cs_set_convert
CS_SET	変換ルーチンをインストールする。
CS_GET	このタイプのカレント変換ルーチンを取得する。
CS_CLEAR	CS-Library のこのタイプのデフォルト変換ルーチンで置き換えることによって、現在の変換ルーチンをクリアする。

srctype

変換元データのデータ型です。

desttype

変換先データのデータ型です。

func

カスタム変換関数を指すポインタである、CS_CONV_FUNC 変数を指すポインタです。「カスタム変換ルーチンの定義」(90 ページ) では、カスタム変換関数のプロトタイプについて説明しています。

変換ルーチンをインストールする場合、**func* はインストールしたい変換ルーチンを指します。

変換ルーチンを取得する場合、cs_set_convert は、現在インストールされている変換ルーチンを指すように **func* を設定します。

変換ルーチンをクリアする場合には、**func* を NULL として渡してください。

注意　*func* は関数を示すポインタを指すポインタを表します。このパラメータの受け渡しについて特別な条件があります。「カスタム変換ルーチンのインストール」(92 ページ) のコード例を参照してください。

戻り値

cs_set_convert は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_set_convert の失敗の主な理由は、無効なパラメータの設定です。

使用法

- アプリケーションは、カスタム変換ルーチンをインストールすることによって、次のデータ型間でデータを変換できます。
  - 標準の Open Client または Open Server のデータ型
  - 標準データ型とユーザ定義のデータ型
  - ユーザ定義データ型
- 特定の変換に対してカスタム・ルーチンがインストールされると、指定された型変換が必要なときに、クライアント／サーバ・ライブラリは透過的にカスタム・ルーチンを呼び出します。
- Client-Library または Server-Library のアプリケーションは、次の宣言でユーザ定義データ型を作成します。

  ```
  typedef CS_SMALLINT          EMPLOYEE_ID;
  ```

  Open Client ルーチンの ct_bind および cs_convert は、整数記号定数を使用してデータ型を識別するので、アプリケーションでユーザ定義型に対して型定数を宣言すると便利です。ユーザ定義データ型は、CS_USERTYPE 以上の値として定義します。

  ```
  #define EMPLOYEE_ID_TYPE     CS_USERTYPE + 1;
  ```

  ユーザ定義型と CS-Library 標準データ型間の変換を可能にするために、アプリケーションは cs_set_convert を呼び出して、新しい型のためのユーザ定義変換ルーチンをインストールします。
- カスタム変換ルーチンをクリアするには、アプリケーションは *action* に CS_CLEAR を、*func* に NULL を設定して、cs_set_convert を呼び出します。cs_set_convert は、カスタム・ルーチンを CS-Library の適切なタイプのデフォルト変換ルーチン (ある場合) に置き換えます。
- アプリケーションで cs_setnull を呼び出して、ユーザ定義型の NULL 代入値を定義できます。

カスタム変換ルーチンの定義

- カスタム変換ルーチンは次のように定義されます。

```
CS_RETCODE CS_PUBLIC
 convfunc(context, srcfmt, srcdata,
          destfmt, destdata, destlen)
 CS_CONTEXT       *context;
 CS_DATAFMT         *srcfmt;
 CS_VOID            *srcdata;
 CS_DATAFMT         *destfmt;
 CS_VOID            *destdata;
 CS_INT             *destlen;
```

各パラメータの意味は次のとおりです。

- *context* は CS_CONTEXT 構造体へのポインタです。
- *srcfmt* は、変換元データを記述する CS_DATAFMT 構造体へのポインタです。*srcfmt–>maxlength* は、変換元データの実際の長さをバイト単位で記述します。
- *srcdata* は変換元データを指すポインタです。
- *destfmt* は、変換先データを記述する CS_DATAFMT 構造体へのポインタです。*destfmt–>maxlength* は、変換先データの実際の長さをバイト単位で記述します。
- *destdata* は変換先データ領域へのポインタです。
- *destlen* は整数を指すポインタです。変換が成功した場合、カスタム・ルーチンは **destlen* を **destdata* に入れられたバイト数に設定します。

- cs_config は、カスタム変換ルーチン内から呼び出すことのできる唯一の CS-Library、Client-Library、または Server-Library の関数です。
- 次の表は、カスタム変換ルーチンにおける有効な戻り値を示します。CS-Library は、CS_SUCCEED 以外の値が返ってきた場合は CS-Library エラーを示します。表 2-17 に記述されているように、他の値が返される場合は、エラー条件を示します。
  - 変換ルーチンから返される値が、表 2-17 にリストされている CS_SUCCEED 以外の値の場合は、その値が示すエラー条件に対応する Client-Library または CS-Library のメッセージがアプリケーションに返されます。
  - 変換ルーチンが表 2-17 にない値を返す場合には、アプリケーションは Client-Library または CS-Library から "Unknown return code" というエラー・メッセージを受け取ります。

表 2-17：カスタム変換ルーチンの戻り値

戻り値	意味
CS_SUCCEED	変換に成功した。
CS_TRUNCATED	変換の結果がトランケートされた。
CS_MEM_ERROR	メモリの割り付け障害が発生した。
CS_EBADXLT	文字が変換できなかった。
CS_ENOXLT	要求された変換はサポートされていない。
CS_EDOMAIN	変換元の値がデータ型の正しい値のドメインの範囲外である。
CS_EDIVZERO	0 による除算はできない。
CS_EOVERFLOW	変換の結果がオーバフローした。
CS_EUNDERFLOW	変換の結果がアンダフローした。
CS_EPRECISION	変換の結果、精度が損なわれた。
CS_ESCALE	無効な位取り値が検出された。
CS_ESYNTAX	変換の結果、変換先の型に対して構文上正しくない値である。
CS_ESTYLE	スタイル・エラーのために変換オペレーションが停止した。

カスタム変換ルーチンのインストール

次に示すコードは、cs_set_convert を呼び出してカスタム変換ルーチン MyConvert をインストールする方法を示しています。この変換ルーチンは、CS_CHAR から MY_USER_TYPE によって示されているユーザ定義型への変換を行います。このコードは、MyConvert が正しく定義されたカスタム変換ルーチンであることを前提としています (「カスタム変換ルーチンの定義」(90 ページ) を参照)。プログラム変数 *myfunc* を使用して、変換ルーチンのアドレスを渡しています。

```
#define MY_USER_TYPE (CS_USER_TYPE + 2)

CS_CONV_FUNC p_conv_func;

p_conv_func = MyConvert;
 if (cs_set_convert(context, CS_SET, CS_CHAR_TYPE, MY_USER_TYPE,
     &p_conv_func) != CS_SUCCEED)
 {
     fprintf(stdout, "cs_set_convert(MY_USER_TYPE) failed!￥n");
     (CS_VOID)ct_exit(context, CS_FORCE_EXIT);
     (CS_VOID)cs_ctx_drop(context);
     exit(1);
 }
```

参照 cs_convert、cs_manage_convert、cs_setnull、ct_bind

cs_setnull

説明

NULL データをバインドまたは変換するときに使用する NULL 代入値を定義します。

構文

```
CS_RETCODE cs_setnull(context, datafmt, buffer,
        buflen)

CS_CONTEXT    *context;
CS_DATAFMT    *datafmt;
CS_VOID       *buffer;
CS_INT        buflen;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタ。cs_setnull はこのコンテキストの NULL 代入値を定義します。

datafmt

NULL 代入値が定義されるデータ型を記述する CS_DATAFMT 構造体を指すポインタです。

buffer

NULL 代入値を指すポインタです。***buffer* のデータ型は *datafmt –>type* と一致する必要があります。

buflen

**buffer* の長さ (バイト単位)

戻り値

cs_set_null は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_setnull の失敗の主な理由は、次のとおりです。

- メモリ割り付けエラー
- 無効なパラメータの設定

使用法

- ANSI スタイルのバインドが有効な場合、CS-Library は NULL 代入値を使用しません。ANSI スタイルのバインドをアクティブな状態にするために、アプリケーションは Client-Library プロパティ CS_ANSI_BINDS を CS_TRUE に設定します。

- ANSI スタイルのバインドが無効で、変換元のデータが NULL の場合、CS-Library は対象データを、その対象型に対してあらかじめ定義された NULL 代入値に設定します。たとえば、すべての型の NULL 値を CS_CHAR 対象に変換すると、結果は空の文字列になります。
- Client-Library アプリケーションでは、NULL 代入値はコンテキスト・レベルで定義されます。Client-Library 接続が割り付けられると、接続は、その親コンテキストから NULL 代入値を取得します。
- NULL の変換元値を CS_CHAR または CS_BINARY の変換先変数に変換するときに、CS-Library は最初に 0 バイトを変換先に入れます。次に、変換先を記述する CS_DATAFMT 構造体の *format* フィールドを使用して、埋め込むか、または NULL で終了するかを判断します。
- 特定のデータ型の NULL 代入値を CS-Library のデフォルトに戻すには、*buffer* を NULL に設定して cs_setnull を呼び出します。
- CS-Library および Client-Library は、次のデフォルトの NULL 代入値を使用します。

表 2-18 : デフォルトの null 代入値

送信先の型	null 代入値
CS_BINARY_TYPE	空の配列
CS_VARBINARY_TYPE	空の配列
CS_BIT_TYPE	0
CS_CHAR_TYPE	空の文字列
CS_VARCHAR_TYPE	空の文字列
CS_DATE_TYPE	4 バイトのゼロ
CS_TIME_TYPE	4 バイトのゼロ
CS_BIGDATETIME_TYPE	8 バイトのゼロ
CS_BIGTIME_TYPE	8 バイトのゼロ
CS_DATETIME_TYPE	8 バイトのゼロ
CS_DATETIME4_TYPE	4 バイトのゼロ
CS_TINYINT_TYPE	0
CS_SMALLINT_TYPE	0
CS_INT_TYPE	0
CS_DECIMAL_TYPE	0.0 (デフォルトの位取りおよび精度)
CS_NUMERIC_TYPE	0.0 (デフォルトの位取りおよび精度)
CS_FLOAT_TYPE	0.0
CS_REAL_TYPE	0.0

送信先の型	null 代入値
CS_MONEY_TYPE	$0.0
CS_MONEY4_TYPE	$0.0
CS_BOUNDARY_TYPE	空の文字列
CS_SENSITIVITY_TYPE	空の文字列
CS_TEXT_TYPE	空の文字列
CS_UNITEXT_TYPE	空の文字列
CS_IMAGE_TYPE	空の配列
CS_XML_TYPE	空の文字列

参照　cs_set_convert, cs_will_convert

cs_snprintf

説明　すべてのプラットフォームを対象とする、snprintf に類似する関数。フォーマットされた出力の変換を行います。この結果は常に null で終了する文字列になる。

構文　void cs_snprintf(char *str, size_t size, const char *format, ...)

パラメータ

str
　出力先の文字列。

size
　書き込みの最大バイト数。

format
　ゼロまたは 1 以上の変換ディレクティブで構成される文字列。

戻り値　なし

cs_strbuild

説明

ネイティブ言語メッセージ文字列を構築します。

構文

```
CS_RETCODE cs_strbuild(context, buffer, buflen,
        resultlen, text, textlen
        [, formats, formatlen]
        [, arguments]);

CS_CONTEXT      *context;
CS_CHAR         *buffer;
CS_INT          buflen;
CS_INT          *resultlen;
CS_CHAR         *text;
CS_INT          textlen;
CS_CHAR         *formats;       /* Optional */
CS_INT          formatlen;      /* Optional */
< オプション引数 >
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

buffer

cs_strbuild によって終了メッセージが格納される領域を指すポインタです。終了メッセージは、NULL で終了していないことに注意してください。アプリケーションは、**resultlen* を使用して、**buffer* に記述されたメッセージの長さを調べる必要があります。

buflen

**buffer* データ領域の長さ (バイト単位) です。

resultlen

整数変数を指すポインタです。cs_strbuild は、**resultlen* を、**buffer* に入れられたデータの長さ (バイト単位) に設定します。

text

メッセージの未終了テキストを指すポインタです。**text* 文字列には、メッセージ・テキストおよび変数に対するプレースホルダが含まれています。プレースホルダの形式は *%integer*! (%1!、%2! など) です。整数は、特定のプレースホルダに対してどの引数を置き換えるかを示します。引数は左から右に番号が付けられます。

textlen

text* の長さ (バイト単位) です。text* が NULL で終了する場合には、*textlen* を CS_NULLTERM にして渡してください。

フォーマット

**text* 文字列内の各プレースホルダに対する sprintf スタイルのフォーマット指定子を 1 つ含んでいる文字列を指すポインタです。

formatlen

formats* の長さ (バイト単位) です。formats* が NULL で終了する場合には、*formatlen* を CS_NULLTERM にして渡してください。

arguments

formats* 文字列に応じて文字列に変換され、buffer* に入れられるメッセージを生成するために **text* 文字列で置き換えられる値です。

各プレースホルダに対して、引数が 1 つ存在しなければなりません。最初の値は最初のフォーマットおよび %1! プレースホルダに対応し、2 番目の値は 2 番目のフォーマットおよび %2! プレースホルダに対応し、以下同様に続きます。

十分な引数を指定せずに cs_strbuild を呼び出すと、予測できない結果になります。

引数の指定が多すぎると、余分な引数は無視されます。

戻り値

cs_str_build は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

使用法

- cs_strbuild は、値のプレースホルダを含んでいるテキスト、型の情報および値の表示属性を含んでいるフォーマット文字列、および値を表す複数の引数から、印刷可能なネイティブ言語のメッセージ文字列を構築します。
- エラー・メッセージのパラメータの順序は、言語によって異なります。cs_strbuild によって、アプリケーションは sprintf と同様の方法でエラー・メッセージを構築できるため、ある言語から他の言語にエラー・メッセージを容易に変換できます。

  たとえば、ストアド・プロシージャの中でキーワードが誤って使用されたことをユーザに通知するエラー・メッセージがあるとします。このメッセージには、3 つの引数が必要です。使い方に誤りのあるキーワード、そのキーワードのある行、およびストアド・プロシージャの名前です。この場合、英語のローカライゼーション・ファイルのメッセージ・テキストは、次のようになります。

  ```
  The keyword ‘%1!‘ is misused in line %2! of stored
  procedure ‘%3!‘.
  ```

  日本語のローカライゼーション・ファイルでは、同じメッセージが次のようになります。

ストアド・プロシージャ '%3!' の行 %2! において、誤ったキーワード '%1!' が使用されています。

上記のいずれかのメッセージを呼び出す cs_strbuild は、次のようになります。

```
cs_strbuild(context, &mybuffer, buflength,
    &resultlength, messagetext, CS_NULLTERM,
    "%s, %d, %s", CS_NULLTERM,
    keyword, linenum, sp_name);
```

唯一異なる点は、*messagetext* の内容です。

- cs_strbuild のフォーマット指定子を複数指定する場合には、他の文字で区切って指定するか、または連続して指定できます。これによって、既存の英語のメッセージ文字列をフォーマット・パラメータとして使用できます。1 番目のフォーマット指定子は %1! プレースホルダを記述し、2 番目のフォーマット指定子は %2! プレースホルダを記述し、以下同様に続きます。

参照 cs_dt_crack, cs_dt_info, cs_locale

cs_strcmp

説明 指定されたソート順を使用して 2 つの文字列を比較します。

構文

```
CS_RETCODE cs_strcmp(context, locale, type, str1,
        len1, str2, len2, result)

CS_CONTEXT      *context;
CS_LOCALE       *locale;
CS_INT          type;
CS_CHAR         *str1;
CS_INT          len1;
CS_CHAR         *str2;
CS_INTI         len2;
CS_INT          *result;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

locale

CS_LOCALE 構造体を指すポインタです。CS_LOCALE 構造体には、cs_strcmp がソート順を定義するために使用する照合順などのロケール情報が含まれています。

アプリケーションで CS_LOCALE 構造体の中の照合順を変更するには、*type* を CS_LC_COLLATE または CS_SYB_SORTORDER に設定し、cs_locale を呼び出します。

locale を NULL にできます。*locale* が NULL の場合、cs_strcmp は、*context* が指す CS_CONTEXT 構造体内で定義されているローカライゼーション情報を使用します。CS_CONTEXT が割り付けられるときにデフォルトのローカライゼーション情報が取得されるので、ローカライゼーション情報は常にコンテキスト・レベルで定義されます。

type

実行する比較のタイプです。

type が CS_COMPARE の場合、cs_strcmp はアルファベット順に比較します。

type が CS_SORT の場合、ソートされたリストでの表示順で値が比較されます。アルファベット順では等しい文字列を、ソートされたリストの異なる場所に置くことができます。

str1

比較の 1 番目の文字列を指すポインタです。

len1

str1* の長さ (バイト単位) です。str1* が NULL で終了する場合は、*len1* を CS_NULLTERM として渡してください。

str2

比較の 2 番目の文字列を指すポインタです。

len2

str2* の長さ (バイト単位) です。str2* が NULL で終了する場合は、*len2* を CS_NULLTERM として渡してください。

result

比較の結果を指すポインタです。次の表は、**result* の有効値の一覧です。

result の値	**意味**
<0	*str1* がアルファベット順で *str2* よりも小さいか、またはソートされたリスト内で *str1* が *str2* の前にある。
0	*str1* がアルファベット順で *str2* と等しいか、または *str1* と *str2* が同一である。
>0	*str1* がアルファベット順で *str2* よりも大きいか、またはソートされたリスト内で *str1* が *str2* の後にある。

戻り値

cs_strcmp は、次の値を返します。

戻り値	**意味**
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

使用法

- cs_strcmp は、2 つの値を比較して、比較結果を **result* に設定します。
- 一部の言語には、特定のソート順ではアルファベット順に等しい文字列が、異なる文字を含んでいる場合があります。文字列はアルファベット順で等しくても、ソートされたリストに文字列を入れるときに使用する標準の順序があります。

  アプリケーションは、cs_strcmp を使用して、アルファベット順またはソートされたリストの順序を反映した文字列の比較を実行できます。たとえば、ソートされたリスト内に小文字の前に大文字を入れるソート順を指定します。

  - 文字列 "ABC" と文字列 "abc" は同等である。

    type に CS_COMPARE を設定して、"ABC" (*str1*) と "abc" (*str2*) を比較する cs_strcmp を呼び出すと、0 が設定された *result* が返されます。

  - "ABC" を "abc" より前にソートする。

    type に CS_SORT を指定して cs_strcmp を呼び出し、"ABC" (*str1*) と "abc" (*str2*) を比較すると、返される *result* は 0 より小さい値に設定されます。

- cs_strcmp は、使用するソート順を決定するため **locale* を検証します (*locale* が NULL の場合は **context* を検証します)。

- CS_LOCALE 構造体の中のソート順を変更するには、アプリケーションは *type* を CS_LC_COLLATE または CS_SYB_SORTORDER に設定し、cs_locale を呼び出す。
- CS_CONTEXT 構造体の中のソート順を変更するには、必要なソート順を指定して CS_LOCALE 構造体を設定してから、cs_config を呼び出してコンテキストの CS_LOC_PROP プロパティを設定する。

参照　cs_cmp, cs_locale, cs_config

cs_strlcat

説明　セーフ文字列の連結関数。最大で *source_str* の *target_size* - strlen(*target_str*) - 1 文字が *target_str* に付加されます。この結果は、*source_str* または *target_str* が NULL である場合、*target_size* が 0 である場合、あるいは *target_str* でポイントされる文字列が *target_size* バイトより長い場合を除き、常に null で終了する文字列になります。

構文

```
CS_RETCODE cs_strlcat(target_str, source_str, target_size)

CS_CHAR     *target_str;
CS_CHAR     *source_str;
CS_INT      *target_size;
```

パラメータ

target_str
ソース文字列の追加先であるターゲット文字列。

source_str
追加されるソース文字列。

target_size
ターゲット文字列のサイズ。

戻り値

- 0　*source_str* が NULL、*target_str* が NULL、または *target_size* が 0 の場合。
- *target_size* - オーバーフローの場合。
- strlen(*target_str*) + strlen(*source_str*) - 他のすべてのケース。

cs_strlcpy

説明

セーフ文字列のコピー機能。最大で *target_size*-1 文字が *source_str* から *target_str* にコピーされます。必要に応じてトランケートが行われます。この結果は、*source_str* または *target_str* が NULL である場合、あるいは、*target_size* が 0 である場合を除き、常に null で終了する文字列になります。

構文

CS_RETCODE cs_strlcpy(target_str, source_str, target_size)

```
CS_CHAR         *target_str;
CS_CHAR         *source_str;
CS_INT           *target_size;
```

パラメータ

target_str
ソース文字列のコピー先であるターゲット文字列。

source_str
コピー元のソース文字列。

target_size
ターゲット文字列のサイズ。

戻り値

- 0 *source_str* が NULL、*target_str* が NULL、または *target_size* が 0 の場合。
- *target_size* - オーバーフローの場合。
- strlen(*source_str*) - 他のすべてのケース。

cs_time

説明

現在の日付と時刻を取得します。

構文

CS_RETCODE cs_time(context, locale, buffer, buflen,
outlen, daterec)

```
CS_CONTEXT    *context;
CS_LOCALE     *locale;
CS_VOID       *buffer;
CS_INT        buflen;
CS_INT        *outlen;
CS_DATEREC    *daterec;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

locale

CS_LOCALE 構造体を指すポインタです。CS_LOCALE 構造体は、cs_time が現在の日時文字列を作成するときに使用するフォーマット情報などのロケール情報を含んでいます。

locale を NULL にできます。*locale* が NULL の場合、cs_time は、*context* が示す CS_CONTEXT 構造体に定義されたローカライゼーション情報を使用します。CS_CONTEXT が割り付けられるときにデフォルトのローカライゼーション情報が取得されるので、ローカライゼーション情報は常にコンテキスト・レベルで定義されます。

buffer

cs_time によって現在の日付と時刻を表す文字列が格納される領域を指します。

buffer はオプションのパラメータであり、NULL として渡すことができます。*buffer* が NULL の場合には、*daterec* を指定する必要があります。

buflen

**buffer* の長さ (バイト単位)

buffer が指定されており、現在の日時を保持するには **buffer* の大きさが十分でないことが *buflen* によって示されると、cs_time は **outlen* を日時文字列の長さに設定し、CS_FAIL を返します。

buffer が NULL の場合は、*buflen* を CS_UNUSED として渡してください。

outlen

整数変数を指すポインタです。

cs_time は、**outlen* を現在の日時文字列の長さ (バイト単位) に設定します。

文字列が *buflen* バイトよりも大きい場合は、**outlen* の値を使用すると、その文字列を保持するために必要なバイト数を判断できます。

buffer が NULL の場合、*outlen* を NULL として渡してください。

アプリケーションが戻り値の長さの情報を無視する場合、*outlen* を NULL にして渡すことができます。

daterec

cs_time によって現在の日付と時刻が格納される CS_DATEREC 構造体を指すポインタです。cs_time は CS_DATEREC 構造体の *datemsecond* および *datetzone* フィールドに値を設定しないことに注意してください。

この章 cs_dt_crack を参照してください。

daterec はオプションのパラメータであり、NULL として渡すことができます。*daterec* が NULL の場合は、*buffer* を指定する必要があります。

戻り値

cs_time は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

cs_time の失敗の主な理由は、次のとおりです。

- 無効なパラメータの設定。
- *buflen* が、フォーマットされた日時文字列を保持するには **buffer* データ領域の大きさが十分でないことを示した。

使用法

- cs_time は、文字列フォーマットまたは CS_DATEREC 構造体、あるいはその両方で、現在の日付と時刻を返します。
- cs_time は、**context* に含まれているロケール情報に基づいて日付と時刻をフォーマットします。

参照

cs_config, cs_dt_crack, cs_dt_info, cs_locale

cs_validate_cb

説明

ct_callback によって登録される、Client-Library コールバック・ルーチン。

構文

```
typedef struct _cs_sslcertfield
{
        CS_VOID      *value;
        CS_INT        field_id;
        CS_INT        length;
} CS_SSLCERT_FIELD;

typedef struct _cs_sslcert
{
```

```
    CS_INT field_count;
    CS_INT extension_count;
    CS_UINT                 start_date;
    CS_UINT                 end_date;
    CS_SSLCERT_FIELD    *fieldptr;
    CS_SSLCERT_FIELD    *extensionptr;
} CS_SSLCERT;

typedef CS_INT (CS_PUBLIC * CS_CERT_CB) PROTOTYPE ((
    CS_VOID         *user_data,
    CS_SSLCERT    *certptr,
    CS_INT          cert_count,
    CS_INT          valid
    ));
```

パラメータ

certptr

cert_count 個の要素を持つ CS_SSLCERT の配列を指すポインタです。コールバックから戻るときに、使用されていたメモリがすべて解放されます。

注意　配列は null で終了していません。

fieldptr

field_count 要素を指すポインタです。

extensionptr

extension_count 要素を指すポインタです。

cs_will_convert

説明

Client/Server ライブラリ内で特定のデータ型の変換が可能かどうかを示します。

構文

```
CS_RETCODE cs_will_convert(context, srctype, desttype,
        result)

CS_CONTEXT    *context;
CS_INT          srctype;
CS_INT          desttype;
CS_BOOL       *result;
```

パラメータ

context

CS_CONTEXT 構造体を指すポインタです。

srctype
変換元データ (CS_BYTE_TYPE、CS_CHAR_TYPE など) のデータ型を表す記号定数です。

desttype
変換先データのデータ型を表す記号定数です。

result
ブール型変数へのポインタです。cs_will_convert は ***result*** を、データ型変換をサポートしているときは CS_TRUE に、データ型変換をサポートしていないときには CS_FALSE に設定します。

戻り値

cs_will_convert は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

例

```
/*
 ** ex_display_column()
 */


CS_RETCODE CS_PUBLIC
 ex_display_column(context, colfmt, data, datalength,
      indicator)
 CS_CONTEXT       *context;
 CS_DATAFMT           *colfmt;
 CS_VOID         *data;
 CS_INT               datalength;
 CS_SMALLINT     indicator;
 {
    char         *null = "NULL";
    char         *nc   = "NO CONVERT";
    char         *cf   = "CONVERT FAILED";
    CS_DATAFMT      srcfmt;
    CS_DATAFMT      destfmt;
    CS_INT       olen;
    CS_CHAR      wbuf[MAX_CHAR_BUF];
    CS_BOOL      res;
    CS_INT          i;
    CS_INT          disp_len;

 if (indicator == CS_NULLDATA)
 {
```

```
        olen = strlen(null);
        strcpy(wbuf, null);
    }
    else
    {
        cs_will_convert(context, colfmt->datatype,
             CS_CHAR_TYPE, &res);

        if (res != CS_TRUE)
    {
           olen = strlen(nc);
           strcpy(wbuf, nc);
         }
         else
         {
           srcfmt.datatype  = colfmt->datatype;
           srcfmt.format    = colfmt->format;
           srcfmt.locale    = colfmt->locale;
           srcfmt.maxlength = datalength;

           destfmt.maxlength = MAX_CHAR_BUF;
           destfmt.datatype  = CS_CHAR_TYPE;
           destfmt.format    = CS_FMT_NULLTERM;
           destfmt.locale = NULL;

           if (cs_convert(context, &srcfmt, data,
              &destfmt, wbuf, &olen) != CS_SUCCEED)
           {
              olen = strlen(cf);
              strcpy(wbuf, cf);
           }
           else
           {
              /*
              ** output length include null
              ** termination
              */
              olen -= 1;
           }

       }
    }
    fprintf(stdout, "%s", wbuf);

    disp_len = ex_display_dlen(colfmt);
    for (i = 0; i < (disp_len - olen); i++)
```

```
    {
       fputc(' ', stdout);
    }

       return CS_SUCCEED;
    }
```

使用法

- cs_will_convert を使用すると、cs_convert または ct_bind/ct_fetch が特定の変換を実行できるかどうかをアプリケーションで調べることができます。cs_convert は、サポートしていない変換を実行するために呼び出されると、CS_FAIL を返し、CS-Library エラーを生成します。
- cs_convert は、標準データ型とユーザ定義データ型間で変換できます。これらの型変換を可能にするには、アプリケーションが cs_set_convert を使用して、カスタム変換ルーチンをインストールする必要があります。ある変換に対してカスタム・ルーチンが指定されている場合には、cs_will_convert はその変換がサポートされていることを示します。

データ型変換表

cs_convert がサポートするデータ型変換の一覧は、cs_convert のマニュアル・ページに収録されています (表 2-3 (29 ページ) を参照)。

参照

cs_convert, cs_set_convert, cs_setnull

第 3 章

Bulk-Library

この章では、Bulk-Library について説明します。

Bulk-Library の概要

Bulk-Library/C は、Client-Library アプリケーションと Server-Library アプリケーションで Adaptive Server Enterprise のバルク・コピー・インタフェースを使用可能にするルーチンを提供します。

Adaptive Server Enterprise のバルク・コピー・インタフェースを使用すると、クライアント・アプリケーションのプログラム変数とサーバのデータベース・テーブルとの間で高速のデータ転送を実行できます。SQL の insert コマンドと select コマンドを使用したデータ転送方法の代わりにバルク・コピー・インタフェースを使用できます。

管理者は bcp ユーティリティを使用してバルク・コピーを実行できます。プログラマは Bulk-Library を使用して、カスタマイズしたバルク・コピー・ツールを作成できます。また、Bulk-Library は Open Server ゲートウェイ・アプリケーションでバルク・コピーのサポートを可能にするために必要なルーチンを提供します。

Adaptive Server Enterprise が暗号化カラムをサポートしている場合は、暗号化カラムのバルク・コピーもサポートされます。

注意 Bulk-Library/C ルーチンは、Open Client Client-Library と Open Server Server-Library のアプリケーションで使用するルーチンです。DB-Library™ は独自のバルク・コピー・インタフェースを提供しています。DB-Library のインタフェースについては『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。

クライアント・サイドのルーチンとサーバ・サイドのルーチン

Bulk-Library にはクライアント・サイドのルーチンとサーバ・サイドのルーチンがあります。

クライアント・サイドの Bulk-Library ルーチン

クライアント・サイドのルーチンでは、Client-Library プログラマが自分のプログラムからバルク・コピー・コマンドを実行できます。クライアント・サイドのルーチンでは、プログラムは次のことが実行できます。

- データベース・テーブルを移植するためにバルク・コピー・データをリモート・サーバに転送する
- データベース・テーブルの内容をプログラム・メモリに抽出する

サーバ・サイドの Bulk-Library ルーチン

サーバ・サイドのルーチンは Open Server で使用します。Open Server のプログラマは、サーバ・サイドのルーチンをクライアント・サイドのルーチンと共に使用して、Open Server ゲートウェイ経由でバルク・コピーによる転送を実行できます。ゲートウェイ・サーバは、クライアント・サイドのルーチンを使用してリモート・サーバからのバルク・コピー・データを取得し、サーバ・サイドのルーチンを使用してサーバ自身のクライアントにデータを転送します。引数として SRV_PROC (Open Server スレッド制御構造) ポインタを必要とするルーチンは、サーバ・サイドのルーチンです。

サーバ・サイドの Bulk-Library ルーチンでは、アプリケーションは Server-Library にリンクされている必要があります。このルーチンは、クライアント・サイドのルーチンと一緒に使用してください。

ヘッダ・ファイル

ヘッダ・ファイル *bkpublic.h* には、Bulk-Library の定義が記述されています。これは、Bulk-Library ルーチンの呼び出しを行うすべてのアプリケーション・ソース・ファイルに必須のファイルです。

Bulk-Library ルーチンを呼び出す Client-Library アプリケーションには、*bkpublic.h* だけをインクルードします。これは、*bkpublic.h* に *ctpublic.h* がインクルードされているためです。アプリケーションにこの 2 つのヘッダ・ファイルをインクルードしても、特に影響はありません。

Bulk-Library ルーチンを呼び出すゲートウェイ Open Server アプリケーションは、Server-Library に必要な他のインクルード・ファイルに加えて、*bkpublic.h* をインクルードする必要があります。*bkpublic.h* には Open Server のヘッダ・ファイルはインクルードされていません。

Bulk-Library とのリンク

ほとんどのプラットフォームでは、Bulk-Library は独立したライブラリ・ファイルであり、アプリケーションのリンク行に指定する必要があります。使用しているプラットフォームでのコンパイルとリンクの方法については『Open Client/Server プログラマーズ・ガイド補足』を参照してください。

CS_BLKDESC 構造体

Bulk-Library の呼び出しで実行されるすべてのバルク・コピー・オペレーションには、CS_BLKDESC 構造体が必要です。この構造体は、バルク記述子構造体とも呼ばれます。バルク記述子構造体は、特定のバルク・コピー操作を制御する隠し構造体です。

アプリケーションは blk_alloc(128 ページ) でバルク記述子構造体を割り付け、blk_drop(154 ページ) でバルク記述子が使用していたメモリを解放します。バルク記述子構造体の内部は文書化されていませんが、構造体のプロパティは blk_props(163 ページ) ルーチンで取り出したり、変更したりできます。

blk_alloc 以外のすべての Bulk-Library ルーチンでは、入力パラメータとして有効なバルク記述子構造体ポインタが必要です。

バルク記述子構造体は、Client-Library の接続構造体の子構造体と見なされています。バルク・コピー・オペレーションはリモート・サーバとの対話に Client-Library の接続を必要とします。

Bulk-Library クライアントのプログラミング

クライアント・サイドの Bulk-Library ルーチンは、Client-Library プログラムにバルク・コピー機能を提供します。Client-Library プログラマが、データベース以外のアプリケーションとのデータの交換、新しいデータベースへのデータのロード、データベース間のデータの移動を行うアプリケーションを開発しているときには、バルク・コピー機能が役に立ちます。

Client-Library アプリケーションは、Bulk-Library ルーチンを呼び出して、データベース・テーブルにデータをコピー・インしたり、データベース・テーブルからデータをコピー・アウトしたりできます。

- バルク・コピー・イン・オペレーションとは、クライアント・マシンからデータベース・テーブルにデータを移動することで、一般にデータベース・テーブルの移植に使用します。データベースへのバルク・コピーでは、Bulk-Library はロー (生) 形式の表データをネットワーク経由で転送します。データベースへのバルク・コピー・イン・オペレーションは、同じ SQL insert 文でデータを埋め込む操作よりもかなり高速になります。
- バルク・コピー・アウト・オペレーションでは、データがデータベース・テーブルからクライアント・プログラムのメモリ領域に移動されます。このオペレーションは通常、データの抽出に使用します。データ抽出では、バルク・コピーは SQL の同じ機能である select 文に比べて、特にパフォーマンスが優れているわけではありません。ただし、プログラマにとっては Bulk-Library インタフェースを使用する方が便利です。

注意　クライアント・サイドの Bulk-Library ルーチンの結果として発生したエラーは、Client-Library のエラーとしてレポートされます。アプリケーションは、Client-Library メッセージ・コールバックをインストールしてこのエラーを処理するか、ct_diag を使用してインラインで処理してください。

バルク・コピー・イン・オペレーション

アプリケーションは、Bulk-Library ルーチンを呼び出して、プログラム変数からデータベース・テーブルにデータをコピーできます。

データベースへのコピー処理で、SQL の insert の代わりにバルク・コピーを使用する最大の利点は、その速度です。

インデックスのないテーブルにデータをコピーするときには、バルク・コピーの「高速」バージョンを使います。Adaptive Server Enterprise では、高速転送実行中にはデータのロギングは行われません。転送が完了する前にシステムに障害が起きると、データベースには新しいデータは残りません。高速転送は、データベースのリカバリ性に影響するので、Adaptive Server Enterprise のオプション select into/bulkcopy がオンになっているときにだけ有効になります。アプリケーションは Adaptive Server Enterprise のシステム・プロシージャ sp_dboption を呼び出して、このオプションをオンにするか、または Client Library 接続プロパティ CS_BULK_LOGIN を使用します。

select into/bulkcopy オプションがオンになっていないときに、インデックスのないテーブルにデータをコピーしようとすると、Adaptive Server Enterprise はエラー・メッセージを生成します。

バルク・コピー・オペレーションの完了後、システム管理者はデータベースの今後のリカバリのために、データベースをダンプしてください。

データをインデックス付きのテーブルにコピーするときには、バルク・コピーの低速バージョンが自動的に使用され、ロー挿入のログが取られます。

バルク・コピー・インのプロセス

一般的なアプリケーションでは、次の手順でバルク・コピー・イン・オペレーションが実行されます。

1　Client-Library アプリケーションと同様の方法でアプリケーションを初期化し、Client-Library エラー処理を設定します。Bulk-Library は、クライアント・サイドのルーチンへの呼び出しで生成されたエラーを Client-Library のメッセージとしてレポートします。

2　使用する接続構造体を割り付けます。

3 ct_con_props を呼び出して、ターゲット・サーバへの接続に必要なプロパティを設定します。さらに、アプリケーションは CS_BULK_LOGIN プロパティを CS_TRUE に設定して、この接続でのバルク・コピーの実行を有効にする必要があります。

注意 スループットを向上させるために、プログラムで Tabular Data Stream™ (TDS) のパケット・サイズを調整できることもあります。デフォルトよりも大きなパケット・サイズにすると、通常はパフォーマンスが向上します。Adaptive Server Enterprise が大きな TDS パケット・サイズを受け入れるように設定されていることを確認してから、アプリケーションで CS_PACKET_SIZE 接続プロパティを設定します。設定可能なネットワーク・パケット・サイズの増加については『システム管理ガイド』、接続プロパティの詳細については『Open Client Client-Library/C リファレンス・マニュアル』を参照してください。

4 ct_connect を呼び出して接続をオープンします。

5 blk_alloc を呼び出してバルク記述子構造体を割り付けます。

6 blk_init を呼び出してバルク・コピー・オペレーションを初期化します。

7 ターゲット・テーブルのカラムごとに、アプリケーションは次の操作を実行します。

- (オプション) blk_describe を呼び出します。返されたターゲット・カラムの記述により、アプリケーションはカラムのデータ型やサイズを判断できます。
- (オプション) blk_default を呼び出します。デフォルトがテーブル・スキーマによって定義されている場合、カラムのデフォルト値が返されます。バルク・コピー・イン・オペレーションがカラムのデフォルト値を使用することを指示するには、アプリケーションは ***datalen* を 0 に設定して blk_bind を呼び出します。
- blk_bind を呼び出して変数をターゲット・カラムにバインドします。blk_textxfer を使用してカラムのデータを転送する場合は、blk_bind を呼び出すときに *buffer* を NULL に設定する必要があります。

カラムはスカラ変数または配列のどちらかにバインドできます。カラムがスカラ変数にバインドされると、blk_rowxfer_mult を呼び出すたびに、1 つのローのカラム値がバインドされた変数からデータベースに転送されます。配列バインドでは、配列が各カラムにバインドされ、blk_rowxfer_mult を呼び出すたびに複数のローが転送されます。どちらの場合でも、アプリケーションは *indicator* 変数と *datalen* 変数もカラムにバインドします。これらの変数は転送されるデータの条件を示すために使用されます。

この章の説明では、配列バインドが有効になっていないことを前提にしています。詳細については、「第 4 章 Bulk-Library ルーチン」の blk_bind を参照してください。

8　データを転送します。

転送するデータが残っていると、アプリケーションはテーブル・カラムにバインドされているプログラム変数にデータを入れ、次に blk_rowxfer_mult を呼び出してそのローを転送します。

バインドされたそれぞれのカラムに対して blk_rowxfer_mult を呼び出す前に、アプリケーションは次のように *datalen* 値と *indicator* 値を設定して、挿入する値を指定します。

datalen の値	**indicator の値**	**結果**
> 0	任意 (無視される)	blk_rowxfer_mult は、*buffer* から *datalen* バイト分をカラム値として読み込む。
0	0	カラムのデフォルト値が利用できれば挿入される。デフォルト値が利用できない場合は、NULL が挿入される。
0	-1	NULL が挿入される。

ローの中の複数のカラムのデータがまとめて転送される場合、アプリケーションは、各カラムに対して blk_textxfer をループで呼び出します。blk_textxfer によって転送されるデータは、ローの最後、つまりバインドされたカラムの後に存在します。

アプリケーションは、必要に応じて blk_done(CS_BLK_BATCH) を呼び出して、ローをまとめて送信できます。この呼び出しは、アプリケーションによる前回の blk_done の呼び出し以降に転送されたすべてのローを永続的に保存することを Adaptive Server Enterprise に指示します。

9 blk_done(CS_BLK_ALL) を呼び出してローの最後のバッチを送信し、バルク・コピー・オペレーションが完了したことを示します。

10 blk_drop を呼び出して、バルク記述子構造体の割り付けを解除します。

注意 アプリケーションは、blk_rowxfer_mult 呼び出しの合間に blk_bind を呼び出して、別のプログラム変数のアドレスや長さを指定できます。

バルク・コピー・イン・オペレーションのプログラム構造

ほとんどのアプリケーションでは、次に示すコード例のようなプログラム構造を使用してバルク・コピー・イン・オペレーションを実行します。

```
ct_con_props to set connection properties
 ct_connect to open the connection
 blk_alloc to allocate a CS_BLKDESC
 blk_init to initiate the bulk copy

for each column
       (optional:blk_describe to get a description of
            the column)
       (optional:blk_default to get the column's default
            value)
        blk_bind to bind the column to a program
            variable, or to mark the column for transfer
            via blk_textxfer
 endfor

while there's data to transfer
        if it's time to save a batch of rows
            blk_done(CS_BLK_BATCH)
        endif
        copy row values to program variables
        call blk_rowxfer_mult to transfer the row data

      if data is being transferred via blk_textxfer
            for each column to transfer
                 while there's data for this column
                     blk_textxfer to tranfer a chunk of data
                 endwhile
            endfor
        endif
 endwhile
```

```
blk_done(CS_BLK_ALL)
 blk_drop to deallocate the CS_BLKDESC
```

バルク・コピー・アウト・オペレーション

バルク・コピー・アウトのプロセスは、サーバからローを読み込み、カラム値をプログラム変数に入れます。

バルク・コピー・アウトのプロセス

一般的なアプリケーションでは、次の手順でバルク・コピー・アウト・オペレーションが実行されます。

1　ct_con_props を呼び出して接続のオープンに必要なプロパティを設定します。

2　ct_connect を呼び出して接続をオープンします。

3　blk_alloc を呼び出してバルク記述子構造体を割り付けます。

4　対象となるカラムごとに、アプリケーションは次の操作を行います。

- (オプション) blk_describe を呼び出してカラムの記述を取得します。アプリケーションがカラムのデータ型やサイズの情報を持っていない場合は、この手順が必要です。
- (オプション) blk_bind を呼び出してコピー元のカラムにプログラム変数をバインドします。blk_textxfer を使用してカラムのデータを転送する場合は、**buffer* を NULL に設定して blk_bind を呼び出します。

  カラムはスカラ変数または配列のどちらかにバインドできます。カラムがスカラ変数にバインドされると、blk_rowxfer_mult を呼び出すたびに、1 つのローのカラム値が、データベースにバインドされた変数に転送されます。配列バインドでは、配列が各カラムにバインドされ、blk_rowxfer_mult を呼び出すたびに、複数のカラム値が各配列に転送されます。

  この章の説明では、配列バインドを使用しないことを前提にしています。詳細については、「第 4 章 Bulk-Library ルーチン」の blk_bind を参照してください。

5 blk_rowxfer_mult をループで呼び出してデータを転送します。

アプリケーションは、blk_rowxfer_mult が CS_END_DATA を返すまで blk_rowxfer_mult を繰り返し呼び出して、各ローをプログラム変数に転送します。

ローの中の複数のカラムのデータがまとめて転送される場合、アプリケーションは、各カラムに対して blk_textxfer をループで呼び出します。blk_textxfer によって転送されるデータは、ローの最後、つまりバインドされたカラムの後に存在します。

たとえば、アプリケーションが 1、3、5、7、9 のカラムのバルク・コピーを行い、カラム 7 と 9 をコピーするために blk_textxfer を呼び出す必要があるとします。アプリケーションは、それぞれのカラムに対して blk_bind を呼び出しますが、カラム 7 と 9 に対しては *buffer* を NULL として渡します。blk_rowxfer_mult を呼び出して、テーブルから 1 つのローを転送した後で、アプリケーションは blk_textxfer をループで呼び出してカラム 7 のデータをコピーし、次に別のループで blk_textxfer を呼び出してカラム 9 のデータをコピーする必要があります。

6 blk_done(CS_BLK_ALL) を呼び出して、バルク・コピー・オペレーションが完了したことを示します。

7 blk_drop を呼び出して、バルク記述子構造体の割り付けを解除します。

注意 アプリケーションは、blk_rowxfer_mult 呼び出しの合間に blk_bind を呼び出して、別のプログラム変数のアドレスや長さを指定できます。

バルク・コピー・アウト・オペレーションのプログラム構造

ほとんどのアプリケーションでは、次に示すコード例のようなプログラム構造を使用してバルク・コピー・アウト・オペレーションを実行します。

```
ct_con_props to set connection properties
 ct_connect to open the connection
 blk_alloc to allocate a CS_BLKDESC
 blk_init to initiate the bulk copy
 for each column of interest
          (optional:blk_describe to get a description of
                the column)
          blk_bind to either bind the column to a program
```

```
                variable or to indicate that blk_textxfer will
                be used to transfer data for the column.
 endfor
while there's data to transfer
          call blk_rowxfer_mult to transfer the row data
          pull data from program variables to a permanent
                location, if desired.
          if data is being transferred via blk_textxfer
                for each column to transfer
                   while there's data for this column
                      blk_textxfer to tranfer a chunk of data
                   endwhile
                endfor
          endif
 endwhile
 blk_done(CS_BLK_ALL)
 blk_drop to deallocate the CS_BLKDESC
```

Secure Adaptive Server Enterprise とのデータのコピー

Secure Adaptive Server Enterprise テーブルにある各ローには sensitivity カラムがあり、このカラムにはローの sensitivity ラベルがあります。Secure Adaptive Server Enterprise はデータへアクセスするときに sensitivity ラベルを使用します。

Secure Adaptive Server Enterprise テーブルとの間でバルク・コピーを行うときに、アプリケーションはテーブルの sensitivity カラムをバルク・コピー・オペレーションの対象に含めるかどうかを選択できます。

sensitivity カラムをバルク・コピー・オペレーションの対象に含めるには、アプリケーションは BLK_SENSITIVITY_LBL プロパティを CS_TRUE に設定します。BLK_SENSITIVITY_LBL のデフォルト値は CS_FALSE で、sensitivity カラムが含まれないことを意味します。

sensitivity カラムへのコピーを行うユーザには、Secure Adaptive Server Enterprise 上で bcpin_labels_role が与えられている必要があります。ユーザがこの役割を持っていないと、バルク・コピー・オペレーションは失敗します。この役割の設定については、Secure Adaptive Server Enterprise のマニュアルを参照してください。

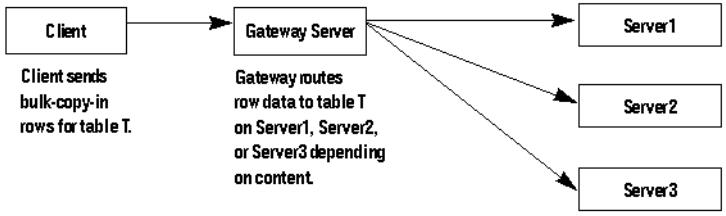
Bulk-Library ゲートウェイのプログラミング

サーバ・サイドの Bulk-Library ルーチンは、クライアント・サイドのルーチンと一緒にゲートウェイで使用するように設計されています。Bulk-Library ルーチンを呼び出すには、Open Server アプリケーションが (Client-Library 呼び出しで設定される) 有効な CS_CONNECTION 構造体を使用できる必要があります。

Open Server のバルク・コピー機能では、ゲートウェイ Open Server アプリケーションがバルク・コピー・データをフィルタできます。ゲートウェイ Open Server は、バルク・コピー・オペレーションの各ローを調べ、次に示すフィルタのいずれかを実装できます。

- 他のローを保持したまま、特定のローを破棄する
- リモート・サーバにすべてのローを送信する
- 次の図に示すように、ローの内容に基づいてバルク・コピー要求を複数のリモート・サーバにルート指定する

図 3-1: バルク・コピー要求をルート指定するゲートウェイ



ゲートウェイのクライアントは、「TDS text/image 挿入要求」と、「TDS バルク・コピー要求」の 2 種類のバルク・コピー要求を発行できます。TDS text/image 挿入要求の場合は、クライアントは text または image ストリームの送信を要求します。TDS バルク・コピー要求の場合は、クライアントは実際にバルク・コピー要求を開始します。どちらの場合も、要求処理には言語イベント (SRV_LANGUAGE) とバルク・イベント (SRV_BULK) の両方の処理が含まれます。

Open Server アプリケーションは、SRV_LANGUAGE と SRV_BULK の 2 つのイベント・ハンドラを使用して両方の要求を処理します。SRV_LANGUAGE イベント・ハンドラの内部では、クライアントがどのバルク要求を発行したかをアプリケーションが判断し、その情報を内部的に記録します。さらに、要求がバルク・コピーに対するものである場合には、アプリケーションはバルク記述子構造体を割り付け、初期化します。SRV_BULK イベント・ハンドラの内部では、アプリケーションは要求タイプを取得してから、それに従ってデータを処理します。

この項の説明では、ゲートウェイ・アプリケーションがバルク・コピー挿入要求と text/image 挿入要求の両方を受け入れることを前提にしています。text/image 挿入コマンドだけの処理の詳細については『Open Server Server-Library/C リファレンス・マニュアル』の「text および image」を参照してください。

注意　Bulk-Library は、サーバ・サイドのルーチンを呼び出した結果発生したエラーを Server-Library エラーとしてレポートします。サーバ・サイドの Bulk-Library ルーチンを呼び出すアプリケーションは、Server-Library エラー・ハンドラをインストールして、これらのエラーのノーティフィケーションを受信してください。

SRV_LANGUAGE イベント・ハンドラの内部構造

ゲートウェイ・アプリケーションでどちらのタイプのバルク要求も処理する場合には、"insert bulk" か "writetext bulk" のフレーズを解析するように SRV_LANGUAGE イベント・ハンドラをコーディングする必要があります。これらのフレーズは次の内容を示しています。

- "insert bulk" フレーズは、バルク・コピー要求の開始を示します。要求の処理は言語ハンドラで起動され、SRV_BULK ハンドラで終了します。
- "writetext bulk" フレーズは、クライアントが SRV_BULK イベント・ハンドラで処理される text または image のストリームのバイトを発行することを示します。

"Insert Bulk" 要求

"insert bulk" 言語要求のテキストは、次のようになります。

```
insert bulk tablename [with nodescribe]
```

"with nodescribe" はオプションです。

これに応答して、SRV_LANGUAGE イベント・ハンドラは次の処理を行います。

1 srv_thread_props を呼び出して、バルク・タイプを内部に記録します。この呼び出しでは、*cmd* を CS_SET に設定し、*property* を SRV_T_BULKTYPE に設定し、*bufp* が SRV_BULKLOAD の値を指すように設定します。

2 解析を継続してテーブル名を抽出します。テーブル名は blk_init ルーチンへの引数となります。テーブル名は、スライス情報なしの "*database.owner.tablename*" のフォームで表されます。スライスを bulk insert コマンドで使用する場合は、コロンとスライス番号をテーブル名から削除する必要があります。

3 blk_alloc を呼び出して、バルク記述子構造体 CS_BLKDESC を割り付けます。

4 blk_init を呼び出して、データ交換のクライアント側を初期化します。

5 "with nodescribe" が指定されている場合は、このデータがバッチの一部であり、バルク・データがロードされるテーブルがすでに記述されていることを意味しています。アプリケーションで blk_srvinit をもう一度呼び出す必要はありません。

   "with nodescribe" が指定されていない場合は、blk_srvinit を呼び出してデータ交換のサーバ側を初期化します。

"Writetext Bulk" 要求

"writetext bulk" 言語要求のテキストは、次のようになります。

```
writetext bulk dbname.tblname.colname textptr
[timestamp=timestamp] [with log]
```

タイムスタンプとロギングのインジケータはオプションです。

これに応答して、SRV_LANGUAGE イベント・ハンドラは次の処理を行います。

1　srv_thread_props を呼び出して、バルク・タイプを内部に記録します。この呼び出しでは、*cmd* を CS_SET に設定し、*property* を SRV_T_BULKTYPE に設定し、*bufp* が SRV_TEXTLOAD、SRV_IMAGELOAD、または SRV_UNITEXTLOAD の値を指すように設定します。

2　解析を続行して、オブジェクト名を抽出します。オブジェクト名の形式は一般に、"*dbname.tblname.colname*" です。この名前を、CS_IODESC 構造体の name フィールドと namelen フィールドに保管します。後で、ゲートウェイ・アプリケーションでデータ・ストリームがサーバに渡される場合には、SRV_BULK イベント・ハンドラの中でこのオブジェクト名を ct_data_info の引数として使用できます。

3　解析を継続してテキスト・ポインタを抽出します。テキスト・ポインタは大きな 16 進数として示されます。文字列から実際の CS_BINARY 値に変換されると、テキスト・ポインタとその長さは CS_IODESC 構造体の textptr と textptrlen フィールドに保管されます。

4　解析を続行して、タイムスタンプがある場合は抽出します。タイムスタンプの形式は、"timestamp = *large_hexadecimal_number*" です。文字列から実際の CS_BINARY 値に変換されると、タイムスタンプとその長さを CS_IODESC 構造体の timestamp フィールドと timestamplen フィールドに保管できるようになります。

5　最後に、解析を行ってロギング・インジケータを抽出します。ロギング・インジケータが存在する場合には、"with log" と示されます。このインジケータが存在する場合は、CS_IODESC 構造体の log_on_update フィールドを CS_TRUE に設定してください。

SRV_BULK イベント・ハンドラの内部構造

SRV_BULK イベント・ハンドラの内部では、アプリケーションはハンドラをトリガしたバルク要求に応答する必要があります。ただし、その応答はクライアントがどのタイプのバルク要求を発行したかによって異なります。要求タイプを取得するには、srv_thread_props を呼び出します。このとき、cmd を CS_GET に設定し、*property* を SRV_T_BULKTYPE に設定します。

要求タイプが SRV_TEXTLOAD、SRV_IMAGELOAD、または SRV_UNITEXTLOAD の場合は、srv_text_info ルーチンと srv_get_text ルーチンを使用して、クライアントからの text データまたは image データをまとまりに区切って読み込みます。詳細については『Open Server Server-Library/C リファレンス・マニュアル』の「text および image」を参照してください。

要求タイプが SRV_BULKLOAD の場合、アプリケーションはクライアント・サイドとサーバ・サイドのルーチンを組み合せて使用して、バルク・コピーのローを処理します。バルク・コピーのローを処理するには、SRV_BULK イベント・ハンドラは次の処理を行います。

1 blk_rowalloc を呼び出して CS_BLK_ROW 構造体を割り付けます。

CS_BLK_ROW 構造体は、クライアントから送信されたフォーマット済みバルク・コピー・ローを保持する隠し構造体です。

2 blk_getrow を呼び出して、クライアントのフォーマット済みローを取得します。この呼び出しは text、image、sensitivity、boundary 型のカラム以外のすべてのカラム・データを取得します。ゲートウェイはこれらを後で処理できます。ローに text、image、sensitivity、または boundary のデータが含まれている場合は、blk_getrow は CS_BLK_HASTEXT を返します。それ以外の場合は、CS_SUCCEED を返します。それ以上ローがない場合には、バルク・コピー・オペレーションは完了し、blk_getrow は CS_END_DATA を返します。

3 ゲートウェイがローの内容を調べる必要がある場合は (特定のリモート・サーバにローをルーティングしたり、データを拒否したりする場合など)、blk_colval を呼び出してバルク・ローの各カラムの値を調べます。

4 クライアント・サイドのルーチンの blk_sendrow を呼び出して、フォーマット済みローをリモート・サーバに送信します。

5 受信バルク・ローに text、image、sensitivity、boundary のデータがある場合は、ゲートウェイのサーバ部分は blk_gettext を呼び出して、ローの text、image、sensitivity、boundary 部分を取得します。ハンドラはクライアント・サイドのルーチンである blk_sendtext を呼び出して、このルーチンをリモート・サーバに送信します。

6 blk_rowdrop を呼び出して、blk_rowalloc によって割り付けられた CS_BLK_ROW 構造体の割り付けを解除します。

7　クライアント・サイドのルーチンである blk_done を呼び出して、バッチまたはバルク・コピー・オペレーションが完了したことを示します。

8　blk_drop を呼び出して、バルク記述子構造体の割り付けを解除します。

例

Open Server のオンラインのサンプル・プログラム *ctos.c* には、バルク・コピー要求を処理するコードが含まれています。

トピック名

可変長のローの拡張

Adaptive Server 15.7 では、データオンリーロック (DOL) ローの可変長カラムの最大オフセットは 32767 バイトに拡張されており、8K を超える論理ページ・サイズが設定されている Adaptive Server が長い可変長の DOL ローをサポートできるようになっています。

Adaptive Server の論理ページに移植するために使用される Open Client と Open Server の Bulk-Library 15.7 ルーチンは、拡張された DOL ローをサポートしています。この機能は Bulk-Library 15.7 以降では自動的にアクティブ化されますが、Adaptive Server では有効にする必要があります。

長い DOL ローを使用するように設定されているデータベースは、Bulk-Library 15.5 以前を使用しているアプリケーションから送信された DOL ローを受け入れることができます。ただし、Bulk-Library 15.7 を使用するアプリケーションが長い DOL ローを Adaptive Server 15.5 以前、または以前のバージョンの形式の DOL ローを予期するデータベースに送信することはできません。送信した場合は、次のいずれかのエラーが発生します。

- `BCP failed to create rows in target table.Column %1! would start at an offset over 8191 bytes; this starting location cannot be represented accurately in the table's (row) format.`

- `BCP failed to create rows in target table.Column %1! starts at an offset greater than %2! bytes; this starting location is not permitted by the current database configuration.`

エラーを修正するには、次のどちらかの手順に従います。

- テーブルのロック・スキームをデータオンリーロックから全ページロックに変更します。
- Adaptive Server 15.7 以降に接続している場合は、allow wide dol rows オプションをターゲット・データベースで有効にします。『ASE パフォーマンス&チューニング・シリーズ：物理データベースのチューニング』の「第 2 章 データの格納」を参照してください。

ロー内とロー外の LOB のサポート

Bulk-Library バージョン 15.7 では、Adaptive Server の text、image、unitext のラージ・オブジェクト (LOB) カラムをロー内に格納することをサポートしています。

Adaptive Server 15.7 では、ロー内記憶領域に格納するようにマークされている LOB カラムは、ローに十分な領域が残っている場合にロー内に格納されます。ロー内に書き込むことができるのは、バインドされている LOB データのみです。bcp ユーティリティは LOB データをバインドするので、該当するロー内の LOB データを送信します。『ASE Transact-SQL ユーザーズ・ガイド』の「第 21 章 ロー内／ロー外の LOB」を参照してください。

マテリアライズされていないカラム

Bulk-Library 15.7 は、Adaptive Server 15.7 のマテリアライズされていないカラムを処理できます。この機能により、マテリアライズされていないカラムを含む、変更済みの Adaptive Server テーブルに対してデータのバルク・コピー・インを実行する場合は、Bulk-Library および bcp のバージョン 15.7 以降のみを使用できます。それより前のバージョンの bcp を使用してマテリアライズされていないカラムへのデータのバルク・コピー・インを実行した場合は、Adaptive Server によってエラーが発生します。

第 4 章

Bulk-Library ルーチン

この章では各 Bulk-Library ルーチンについて説明します。

ルーチン	説明	ページ
blk_alloc	CS_BLKDESC 構造体を割り付ける。	128
blk_bind	プログラム変数とデータベース・カラムをバインドする。	131
blk_colval	フォーマットされたバルク・コピー・ローからカラム値を取得するサーバ・サイドのルーチン。	144
blk_default	カラムのデフォルト値を取得する。	146
blk_describe	データベース・カラムの記述を取得する。	148
blk_done	CS_CONTEXT 構造体を割り付ける。	150
blk_drop	CS_BLKDESC 構造体の割り付けを解除する。	154
blk_getrow	フォーマットされたバルク・コピー・ローを取得して保管する、サーバ・サイドのルーチン。	156
blk_gettext	送られてくるフォーマットされたバルク・コピー・ローの text、image、sensitivity、または boundary の部分を取得する、サーバ・サイドのルーチン。	158
blk_init	バルク・コピー・オペレーションを開始する。	160
blk_props	バルク記述子構造体のプロパティを設定または取得する。	163
blk_rowalloc	フォーマットされたバルク・コピー・ローのための領域を割り付けるサーバ・サイドのルーチン。	171
blk_rowdrop	フォーマットされたバルク・コピー・ローに対して、以前に割り付けられた領域を解放するサーバ・サイドのルーチン。	172
blk_rowxfer	ローの数を指定または受信しないで、バルク・コピー・オペレーション時に 1 つ以上のローを転送する。	173
blk_rowxfer_mult	バルク・コピー・オペレーション実行中に 1 つ以上のローを転送する。	176
blk_sendrow	blk_getrow で取得したフォーマットされたバルク・コピー・ローを送る、サーバ・サイドのルーチン。	181

ルーチン	説明	ページ
blk_sendtext	blk_sendtext で取得したフォーマットされたバルク・コピー・ロー内の text、image、sensitivity、または boundary データを送る、サーバ・サイドのルーチン。	183
blk_srvinit	必要に応じてサーバ・テーブル・カラムの記述をクライアントにコピーする、サーバ・サイドのルーチン。	185
blk_textxfer	バルク・コピー・オペレーション時にカラム・データをまとまりごとに転送する。	186

blk_alloc

説明

CS_BLKDESC 構造体を割り付けます。

構文

CS_RETCODE blk_alloc(connection, version, blk_pointer)

```
CS_CONNECTION    *connection;
CS_INT           version;
CS_BLKDESC       **blk_pointer;
```

パラメータ

connection

ct_con_alloc で割り付けられ、ct_connect でオープンされている CS_CONNECTION 構造体を指すポインタです。CS_CONNECTION 構造体は、特定のクライアント／サーバ接続の情報を含んでいます。

この接続に、保留中の結果があってはなりません。

version

対象となる Bulk-Library 動作のバージョンです。初期化時に、*version* の値と Client-Library のバージョン・レベルとの互換性をチェックします。*version* は次の値をとります。

値	意味	Client-Library バージョン・レベルとの互換性
BLK_VERSION_100	バージョン 10.0 の動作	CS_VERSION_110、CS_VERSION_100
BLK_VERSION_110	バージョン 11.0 の動作	BLK_VERSION_100 と同様
BLK_VERSION_120	バージョン 12.0 の動作	BLK_VERSION_100、110 と同様
BLK_VERSION_125	バージョン 12.5 の動作	BLK_VERSION_100、110、120 と同様
BLK_VERSION_150	バージョン 15.0 の動作	BLK_VERSION_100、110、120、125 と同様
BLK_VERSION_155	バージョン 15.5 の動作	BLK_VERSION_100、110、120、125、150 と同様

注意　context/ctlib が CS_VERSION_100 または CS_VERSION_110 のどちらに初期化されている場合でも、BLK_VERSION_100 は、Open Client/Server バージョン 11.x 以降でのみ使用できます。

アプリケーションの Client-Library バージョン・レベルは、接続の親コンテキスト構造体を初期化する ct_init に対する呼び出しによって決定されます。

blk_pointer

ポインタ変数のアドレスです。blk_alloc は、**blk_pointer* に、新しく割り付けた CS_BLKDESC 構造体のアドレスを設定します。

エラーの場合、blk_alloc は **blk_pointer* を NULL に設定します。

戻り値

blk_alloc は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

blk_alloc の失敗の主な理由は、メモリ不足です。

例

```
/*
 ** BulkCopyIn()
 ** Ex_tabname is globally defined.
 */
 CS_STATIC CS_RETCODE
 BulkCopyIn(connection)
 CS_CONNECTION    *connection;
 {
      CS_BLKDESC *blkdesc;
      CS_DATAFMT datafmt;/* variable descriptions */
      Blk_Data *dptr;/* data for transfer */
      CS_INT datalen[5];    /* variable data length */
      CS_INT      len;
      CS_INT numrows;

      /*
      ** Ready to start the bulk copy in now that all the
      ** connections have been made and have a table name.
      ** Start by getting the bulk descriptor and
      ** initializing.
      */
      if (blk_alloc(connection, BLK_VERSION_100, &blkdesc)
            != CS_SUCCEED)
      {
            ex_error("BulkCopyIn:blk_alloc() failed");
            return CS_FAIL;
 }
  if (blk_init(blkdesc, CS_BLK_IN,
            Ex_tabname, strlen(Ex_tabname)) == CS_FAIL)
 {
            ex_error("BulkCopyIn:blk_init() failed");
            return CS_FAIL;
 }
 /*
 ** Bind the variables to the columns and send the rows,
 ** and then clean up.
 */
 ...CODE DELETED.....

 return CS_SUCCEED;
 }
```

使用法

- CS_BLKDESC は「バルク記述子構造体」とも呼ばれ、構造体によるバルク・コピー・データの送受信を制御します。CS_BLKDESC は、特定のバルク・コピー・オペレーションについての情報を含んでいる隠し構造体です。
- blk_alloc を呼び出す前に、アプリケーションは Client-Library ルーチンである ct_con_alloc および ct_connect を呼び出して、CS_CONNECTION 構造体を割り付け、接続をオープンする必要があります。
- blk_alloc は、バルク・コピー・オペレーションで最初に呼び出されるルーチンです。
- 1 つの接続に複数の CS_BLKDESC 構造体と CS_COMMAND 構造体を割り付けることができます。しかし、一度にアクティブにできる CS_BLKDESC 構造体または CS_COMMAND 構造体は 1 つだけです。この章の blk_init (160 ページ) を参照してください。
- CS_BLKDESC 構造体の割り付けを解除するために、アプリケーションは blk_drop を呼び出すことができます。

参照

blk_drop、blk_init、ct_con_alloc、ct_connect

blk_bind

説明

プログラム変数をデータベース・カラムへバインドします。

構文

```
CS_RETCODE blk_bind(blkdesc, colnum, datafmt, buffer,
datalen, indicator)

CS_BLKDESC    *blkdesc;
CS_INT        colnum;
CS_DATAF      *datafmt;
CS_VOID       *buffer;
CS_INT        *datalen;
CS_SMALLINT   *indicator;
```

パラメータ

blkdesc

バルク・コピー・オペレーションの制御ブロックの役目を果たす CS_BLKDESC を指すポインタです。blk_alloc は CS_BLKDESC 構造体を割り付けます。

colnum

プログラム変数にバインドするカラムの番号です。テーブル内の最初のカラムは、カラム番号 1 です。2 番目のカラム番号は 2、以下同様に続きます。表示されているカラムのみカウントされます。

注意 ct_options パラメータを CS_OPT_HIDE_VCC または CS_OPT_SHOW_FI に設定すると、表示されるカラムに影響します。

CS_OPT_HIDE_VCC が CS_TRUE に設定されている場合は、仮想計算カラム (VCC) は公開されず、blk_bind においてカラム番号で表されることもありません。同様に、CS_OPT_SHOW_FI が CS_FALSE のままの場合は、関数インデックス (FI) は公開されず、blk_bind においてカラム番号で表されることもありません。

『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。

---

datafmt

カラムにバインドするプログラム変数を記述する CS_DATAFMT 構造体を指すポインタです。

表 4-1 は、blk_bind が使用する **datafmt* 内のフィールドと、フィールドの一般情報の一覧です。blk_bind は、使用しないフィールドを無視します。

表 4-1 : blk_bind が使用する CS_DATAFMT 構造体のフィールド

フィールド名	使用する場合	設定内容
name	未使用。	適用されない。
namelen	未使用。	適用されない。

フィールド名	使用する場合	設定内容
datatype	常時。	プログラム変数のデータ型を表す型定数 (CS_xxx_TYPE)。 『Open Client Client-Library/C リファレンス・マニュアル』の「データ型」にあるすべての型定数が有効。 Open Client のユーザ定義型は使用できない。 blk_bind は広範囲の型変換をサポートするため、*datatype* はカラムの型と一致している必要はない。たとえば、変数型 CS_FLOAT_TYPE を指定することで、*money* カラムを CS_FLOAT プログラム変数にバインドできる。blk_rowxfer_mult (176 ページ) または blk_rowxfer (173 ページ) はデータ転送時に適切な変換を行う。Client-Library によって提供されているデータ変換のリストについては、「第 2 章 CS-Library ルーチン」 の cs_convert (27 ページ) を参照。 *datatype* が CS_BOUNDARY_TYPE または CS_SENSITIVITY_TYPE の場合、プログラム変数 **buffer* の型は CS_CHAR でなければならない。

フィールド名	使用する場合	設定内容

format	コピー・アウト・オペレーション中に、文字型またはバイナリ型対象の変数へバインドするとき。それ以外の場合は、CS_FMT_UNUSED。	次の送信先タイプのビットマスクと関連の記号：
		• 文字および text の変換先タイプの場合：
		  • データを NULL で終了させるには、CS_FMT_NULLTERM。
		  • 変数の全長までスペースを埋め込ませるには、CS_FMT_PADBLANK。
		• 文字、バイナリ、text、および image の送信先タイプの場合：
		  • 変数の全長まで NULL を埋め込ませるには、CS_FMT_PADNULL。
		• すべての型の送信先タイプの場合：
		  • フォーマット情報が提供されない場合には、CS_FTM_UNUSED。
		• 配列バインドを使用する場合、CS_BLK_ARRAY_MAXLEN がバルク・コピー・イン・オペレーションの唯一のフォーマットである。「配列バインド」(143 ページ) を参照してください。

maxlength	可変長データ型にバインドするとき。 固定長データ型にバインドするときには、*maxlength* は無視される。	**buffer* プログラム変数の最大長。 文字またはバイナリの変数をバインドするときには、*maxlength* は、NULL ターミネータのような特殊な終了バイトを入れるために必要な領域を含めた、プログラム変数の合計最大長を示す必要がある。 バルク・コピー・イン・オペレーション中の *maxlength* は、**buffer* プログラム変数からコピーされるデータの最大長を指定する。 バルク・コピー・アウト・オペレーション中の *maxlength* は、**buffer* プログラム変数の長さである。

フィールド名	使用する場合	設定内容
scale	数値または 10 進数の変数にバインドするときだけ。	プログラム変数の位取り。 変換元データと変換先データの型が同じ場合、変換先データが変換元データから *scale* の値を取得する必要があることを示すために、*scale* を CS_SRC_VALUE に設定できる。 *scale* は、*precision* 以下でなければなりません。
precision	数値または 10 進数の送信先にバインドするときのみ。	プログラム変数の精度。 変換元データと変換先データの型が同じ場合には、変換先データが変換元データから *precision* の値を取得することを示すには、*precision* を CS_SRC_VALUE に設定する。 *precision* は、*scale* 以上でなければならない。
status	未使用。	適用されない。
count	常時。	*count* は、blk_rowxfer_mult (176 ページ) または blk_rowxfer (173 ページ) の 1 回の呼び出しで転送するローの数。*count* が 1 よりも大きい場合は、配列バインドが有効であるとみなされる。 バルク・コピー・アウト・オペレーション中、*count* が使用可能なローの数よりも大きい場合は、使用可能なローだけがコピーされる。 *count* は、1 つの例外を除いて、転送されるすべてのカラムに対して同じ値を持たなければならない。この例外とは、アプリケーションでは 0 と 1 の count を混在させることができる点である。これは、*count* が 0 の場合には、1 件のローが転送されるためである。
usertype	未使用。	適用されない。
locale	指定すると、*locale* が使用される。指定しない場合、デフォルトのローカライゼーションが適用される。	**buffer* プログラム変数のロケール情報を含んでいる CS_LOCALE 構造体を指すポインタ。

buffer

colnum によって指定されたカラムにバインドされるプログラム変数のアドレスです。

バルク・コピー・イン・オペレーションの場合、**buffer* は、blk_rowxfer_mult によるデータ・コピーのコピー元プログラム変数です。

バルク・コピー・アウト・オペレーションの場合、*buffer** は、blk_rowxfer_mult がコピーするデータを入れるコピー先プログラム変数です。*datafmt–>maxlength* が、**buffer* はコピーされたデータを保持するのに十分な大きさではないことを示す場合、blk_rowxfer_mult はロー転送時にデータをトランケートします。データがトランケートされた場合、Bulk-Library は、**indicator* を使用可能なデータの実際の長さに設定します。

buffer が NULL の場合は、blk_textxfer ルーチンを使用してデータが転送されます。

datalen

**buffer* データの長さ (バイト単位) を指すポインタです。

バルク・コピー・イン・オペレーションの場合

- **buffer* が NULL でない場合、**datalen* は **buffer* プログラム変数に含まれているデータの実際の長さを表します。ローを転送するための blk_rowxfer_mult または blk_rowxfer を呼び出す前に、アプリケーションがこの長さを設定する必要があります。可変長データの場合、ローごとに長さが変わります。配列バインド以外の固定長データの場合には、**datalen* が CS_UNUSED であることがあります。**datalen* が 0 である場合、**indicator* の値を使用して、カラムのデフォルト値または NULL のどちらを挿入するかが判断されます。詳細については、表 4-2 (141 ページ) を参照してください。
- **buffer* が NULL (データが blk_textxfer で転送されることを示す) である場合、**datalen* は転送される値の全体の長さを示します。

バルク・コピー・アウト・オペレーションの場合

- **datalen* は、**buffer* にコピーされるデータの実際の長さを表します。blk_rowxfer_mult または blk_rowxfer は、ローを転送するために呼び出されるたびに **datalen* を設定します。
- blk_rowxfer_mult または blk_rowxfer は、ローを転送するために呼び出されるたびに *datalen* を設定するので、*datalen* パラメータは、blk_bind() と blk_rowxfer()、または blk_rowxfer_mult() を呼び出す関数にローカルである必要があります。そうでない場合、無効な結果が発生します。

indicator

CS_INT 変数を指すポインタです。配列バインドの場合は CS_INT の配列を指します。ローを転送するときに、blk_rowxfer_mult または blk_rowxfer がインジケータの内容を読み込んで、バルク・コピー・データについての特定の条件を判断します。

戻り値

blk_bind は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

バルク・コピー・オペレーションを初期化する blk_init をアプリケーションが呼び出していなかったときに、blk_bind は CS_FAIL を返します。

例

```
/*
 ** BulkCopyIn()
 ** BLKDATA and DATA_END are defined in the bulk copy
 ** example program.
 */

 CS_STATIC CS_RETCODE

 BulkCopyIn(connection)
 CS_CONNECTION    *connection;
 {
      CS_BLKDESC *blkdesc;
      CS_DATAFMT datafmt;/* variable descriptions */
      Blk_Data *dptr;/* data for transfer */
      CS_INT datalen[5];    /* variable data length */
      CS_INT      len;
      CS_INT numrows;

    /*
    ** Ready to start the bulk copy in now that all the
    ** connections have been made and have a table name.
    ** Start by getting the bulk descriptor initializing.
    */
    ...CODE DELETED.....

    /*
    ** Bind the variables to the columns and
    ** transfer the data.
    */
    datafmt.locale = 0;
    datafmt.count = 1;
    dptr = BLKDATA;
    while (dptr->pub_id != DATA_END)
    {
         datafmt.datatype = CS_INT_TYPE;
         datafmt.maxlength = sizeof(CS_INT);
         datalen[0] = CS_UNUSED;

        if (blk_bind(blkdesc, 1, &datafmt, &dptr->pub_id,

             &datalen[0], NULL) != CS_SUCCEED)
         {
             ex_error("BulkCopyIn:blk_bind(1) failed");
             return CS_FAIL;
         }
```

```
        datafmt.datatype = CS_CHAR_TYPE;
        datafmt.maxlength = MAX_PUBNAME - 1;
        datalen[1] = strlen(dptr->pub_name);
        if (blk_bind(blkdesc, 2, &datafmt, dptr->pub_name,
             &datalen[1], NULL) != CS_SUCCEED)
        {
            ex_error("BulkCopyIn:blk_bind(2) failed");
            return CS_FAIL;
        }
        datafmt.maxlength = MAX_PUBCITY - 1;
        datalen[2] = strlen(dptr->pub_city);
        if (blk_bind(blkdesc, 3, &datafmt, dptr->pub_city,
          &datalen[2], NULL) != CS_SUCCEED)
        {
            ex_error("BulkCopyIn:blk_bind(3) failed");
            return CS_FAIL;
        }
        datafmt.maxlength = MAX_PUBST - 1;
        datalen[3] = strlen(dptr->pub_st);
        if (blk_bind(blkdesc, 4, &datafmt, dptr->pub_st,
             &datalen[3], NULL) != CS_SUCCEED)
        {
            ex_error("BulkCopyIn:blk_bind(4) failed");
            return CS_FAIL;
        }
        datafmt.maxlength = MAX_BIO - 1;
        datalen[4] = strlen((char *)dptr->pub_bio);
        if (blk_bind(blkdesc, 5, &datafmt, dptr->pub_bio,
             &datalen[4], NULL) != CS_SUCCEED)
       {
            ex_error("BulkCopyIn:blk_bind(5) failed");
            return CS_FAIL;
       }
       if (blk_rowxfer (blkdesc) == CS_FAIL)
       {
            ex_error("BulkCopyIn:blk_rowxfer() failed");
            return CS_FAIL;
       }
       dptr++;
   }

   /* Mark the operation complete and then clean up */
   ...CODE DELETED.....

   return CS_SUCCEED;
}
```

使用法

- blk_bind はクライアント・サイドのルーチンです。
- blk_bind はプログラム変数をデータベースのテーブル・カラムにバインドします。変数がバインドされると、その後に続く blk_rowxfer_mult 呼び出しでは、データベースとバインドされた変数との間でロー・データがコピーされます。コピー方向は、アプリケーションでその前に行った blk_init 呼び出しによって決まります。
- データベースにコピーする場合、アプリケーションは、データベース・テーブル内のカラムごとに blk_bind を 1 回呼び出す必要があります。データベースからコピーする場合、アプリケーションは対象としないカラムに対して blk_bind を呼び出す必要はありません。
- カラム値が blk_textxfer で転送されることを示すために、アプリケーションは *buffer* を NULL に設定して blk_bind を呼び出します。一般的なアプリケーションでは、大量の text や image の値を転送するときに blk_textxfer を使用します。

  text、image、boundary、または sensitivity データ型のカラムに blk_textxfer で転送されることを示すマークが付けられると、その後に続くこれらのデータ型のカラムもすべて blk_textxfer で転送するものとしてマーク付けする必要があります。たとえば、アプリケーションは、ローの最初の text カラムにマークを付けて blk_textxfer による転送対象としておいて、その後の text カラムはプログラム変数にバインドする、ということはできません。
- アプリケーションは、複数の blk_rowxfer_mult 呼び出しの間に blk_bind を呼び出して、変数のアドレスまたは長さの変更を反映できます。アプリケーションが 1 つのカラムまたは変数に対して blk_bind を複数回呼び出した場合、最後のバインドだけが有効です。
- アプリケーションは、blk_describe を呼び出して、特定のカラムのフォーマットを記述する CS_DATAFMT 構造体を初期化できます。

バルク・コピー・イン・オペレーションの場合の *blk_bind*

表 4-2 は、バルク・コピー・イン・オペレーションで使用する blk_bind のパラメータ値のリストです。*datafmt* フィールドについては、表 4-1 (132 ページ) を参照してください。

表 4-2 : バルク・コピー・インの場合の blk_bind パラメータ値

blk_bind を呼び出す場合	buffer の値	datalen の値	*indicator の値
blk_rowxfer_mult によるカラム値の読み込み元であるスカラまたは配列変数にバインドするとき。	プログラム変数または配列のアドレス	*buffer から読み込まれる値の長さを示す変数または配列を指すポインタ。 • *datalen が 1 以上の場合、*datalen 値は *buffer から読み込まれ、カラム値として送信される。 • *datalen が 0 の場合、*indicator 値を使用してカラムのデフォルト値 (使用可能な場合) または NULL のどちらを挿入するかを判断する。	インジケータ値をカラムに供給するプログラム変数または配列のアドレス。 *datalen が 0 の場合にのみ、*indicator は考慮される。 • *indicator が 0 の場合、カラムのデフォルト値 (使用可能な場合) が挿入される。使用可能なデフォルト値がない場合、NULL が挿入される。 • *indicator が -1 の場合、常に NULL が挿入される。
blk_textxfer を使用してカラム値が転送されることを示すとき。	NULL	blk_textxfer を使用して送信されるデータの全体の長さ。 この場合、*datafmt –>maxlength は無視される。	無視される。

Bulk-Library アプリケーションがバルク・コピー・イン・オペレーションで blk_bind を呼び出す場合、blk_bind に渡される *buffer*、*datalen*、*indicator* の各ポインタが記録されます。これらのポインタが指す領域にあるデータは、blk_rowxfer または blk_rowxfer_mult により読み込まれるまで有効でなければなりません。

バルク・コピー・アウト・オペレーションの場合の *blk_bind*

表 4-3 は、バルク・コピー・アウト・オペレーションで使用する blk_bind のパラメータ値のリストです。*datafmt* フィールドについては、表 4-1 (132 ページ) を参照してください。

表 4-3 : バルク・コピー・アウトの場合の blk_bind パラメータ値

blk_bind を呼び出す場合	buffer の値	*datalen の値	*indicator の値
blk_rowxfer_mult によるカラム値の書き込み先のスカラまたは配列変数にバインドするとき。	プログラム変数または配列のアドレス	blk_rowxfer_mult (176 ページ) が **buffer* に書き込まれる値の長さを入れる変数または CS_INT 配列変数を指すポインタ。	インジケータ値をカラムに供給するプログラム変数または配列のアドレス。 blk_rowxfer_mult は **indicator* を次のように設定する。 • -1 はデータが null であることを示す。 • 0 は正しいデータであることを示す。 • 1 以上の値はトランケーションの発生を示す。値は使用可能なデータの実際の長さとなる。
blk_textxfer を使用してカラム値が転送されることを示すとき。	NULL	無視される。 この場合、*datafmt –>maxlength* は **buffer* データ領域の長さを表す。	無視される。

データベースへのバルク・コピーの場合の NULL 値の指定

- コピー・インする場合、アプリケーションは blk_rowxfer_mult を呼び出す前に、**datalen* を 0、**indicator* を 0 に設定して、カラムのデフォルト値を使用するように blk_rowxfer_mult に指示できます。そのカラムに対してデフォルト値が定義されていない場合は、blk_rowxfer_mult によって NULL 値が挿入されます。
- blk_rowxfer_mult に対して、カラムのデフォルト値に関係なく NULL を挿入させるには、**datalen* を 0 に設定し、**indicator* を -1 に設定してから blk_rowxfer_mult を呼び出します。

バインドのクリア

- バインドをクリアするには、*buffer*、*datafmt*、*datalen*、および *indicator* を NULL に設定して blk_bind を呼び出してください。この方法で呼び出さないと、バルク・コピー・オペレーションの終了を示すために、アプリケーションが *type* を CS_BLK_ALL に設定して blk_done を呼び出すまで、バインドは有効な状態になります。

- すべてのバインドをクリアするには、*buffer*、*datafmt*、*datalen*、および *indicator* を NULL として、*colnum* を CS_UNUSED として渡してください。アプリケーションは、通常、配列バインドに使用されているカウントを変更する必要があるときに、すべてのバインドをクリアします。

配列バインド

- 配列バインドは、プログラム変数の配列にカラムをバインドする処理です。ロー転送時に、1 回の blk_rowxfer_mult 呼び出しにより、カラムの複数のローの値を変数の配列に転送するか、または変数の配列の値をカラムの複数のローに転送します。アプリケーションは、*datafmt–>count* を 1 よりも大きい値に設定して配列バインドを示します。
- 配列バインドはバルク・コピー・インとバルク・コピー・アウト・オペレーションでは異なる動作をします。
- 配列バインドを使用するバルク・コピー・イン・オペレーションでは、配列を指す *buffer*、*datalen*、*indicator* を使用して blk_bind を呼び出す必要があります。長さとインジケータの変数は、それぞれバッファ配列内の対応するデータを記述します。固定長データでは、*buffer* は常に固定長値の配列を指すポインタになります。可変長データ (特に文字データまたはバイナリ・データ) では、*buffer* はバイト配列を指すポインタです。可変長データの場合は、値のパック方法として「緩やか (loose)」と「密集 (dense)」があります。アプリケーションは各カラムのパック方法を指定するため、*datafmt–>format* フィールドにフラグを設定します。その方法は次のとおりです。
  - *datafmt–>format* に CS_BLK_ARRAY_MAXLEN ビットを設定すると、配列の値の緩やかなパックが指定されます。blk_rowxfer_mult は次のように計算されるバイト位置から *datalen*[*i*-1] バイト分を読み込むことで値 *i* を取得します。

    ```
    (i -1) * datafmt->maxlength
    ```

  - CS_BLK_ARRAY_MAXLEN ビットが *datafmt–>format* で設定されていない場合は、blk_rowxfer_mult で転送されるカラム値が密集パックされている必要があります。カラム配列に各値が埋め込みなしで格納されます。blk_rowxfer_mult は次のように計算されるバイト位置から *datalen*[*i*-1] バイト分を読み込むことで、値 *i* を取得します。

    ```
    datalen[i-2] + datalen[i-3] + ...+datalen[0]
    ```

つまり、最初の値は 0 で始まり、2 番目の値は *datalen*[0]、3 番目の値は *datalen*[1] + *datalen*[0]、以下同様になります。

たとえば、"girl"、"boy"、"man"、および "woman" の値を受け取る文字カラムがあり、このカラムが *datafmt–>maxlength* が 7 として渡されてバインドされると仮定します。緩やかな配列バインドでは *buffer* と *datalen* の内容は次のようになります。

```
buffer : girl    boy     man     woman
          0       7       14      21
 datalen:4, 3, 3, 5
```

密集してパックされる配列バインドでは *buffer* と *datalen* の内容は次のようになります。

```
buffer : girlboymanwoman
          0    4   7   10
 datalen:4, 3, 3, 5
```

- バルク・コピー・アウト・オペレーションでは、blk_bind で実行される配列バインドは ct_bind で実行される配列バインドと同じ動作になります。バルク・コピー・アウトのカラム配列は、常に緩やかにパックされます。
- バルク・コピー・アウト・オペレーション中に配列バインドを使用していると、blk_rowxfer_mult が書き込み先配列に書き込みを行っているときに、変換、メモリ、トランケーションのエラーが発生する可能性があります。この場合、blk_rowxfer_mult は書き込み先配列に部分的な結果を書き込み、CS_ROW_FAIL を返します。
- どちらの方向の場合も、配列バインドが有効のときは、アプリケーションで blk_textxfer を使用してデータを転送することはできません。

参照 blk_describe, blk_default, blk_init

blk_colval

説明 フォーマットされたバルク・コピー・ローからカラム値を取得するサーバ・サイドのルーチンです。

構文

```
CS_RETCODE blk_colval(srvproc, blkdescp, rowp, colnum,
        valuep, valuelen, outlenp)

SRV_PROC      *srvproc;
CS_BLKDESC    *blkdescp;
```

```
CS_BLK_ROW   *rowp;
CS_INT       colnum;
CS_VOID      *valuep;
CS_INT       valuelen;
CS_INT       *outlen;
```

パラメータ

srvproc

バルク・コピー・ローを送信するクライアントに関連付けられている SRV_PROC 構造体を指すポインタです。この構造体は、Open Server のアプリケーションとクライアント間の通信およびデータを管理するために Server-Library が使用するすべての情報を含んでいます。

blkdescp

バルク・コピー・データの情報が含まれている CS_BLKDESC 構造体を指すポインタです。この構造体は、あらかじめ blk_alloc 呼び出しで割り付け、blk_init を呼び出して初期化しておく必要があります。この構造体は、送られてくるフォーマットされたバルク・コピー・ローを解釈するときに使用します。

rowp

前回の blk_getrow 呼び出しによってデータが格納された CS_BLK_ROW 構造体を指すポインタです。

CS_BLK_ROW 構造体は、クライアントから送信されたフォーマット済みバルク・コピー・ローを保持する隠し構造体です。

colnum

対象となるカラムのカラム番号です。カラム番号は 1 から始まります。

valuep

バルク・コピー・ローのカラム値が入れられるアプリケーション・バッファを指すポインタです。

valuelen

valuep が指すバッファのバイト単位のサイズです。

outlen

CS_INT 変数を指すポインタです。blk_colval は ***outlen* をカラム・データのサイズ (バイト単位) で設定します。

戻り値

blk_colval は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

使用法

- blk_colval はサーバ・サイドのルーチンです。このルーチンは、フォーマットされたバルク・コピー・ローから特定のカラム値を取得し、アプリケーション・バッファに保管します。
- このルーチンは暗黙のデータ変換は行いません。cs_convert を使用してデータを変換してください。
- blk_colval 呼び出し後にカラム値を調べるため、アプリケーションはこの blk_colval を呼び出す前にそのカラムのデータ型を知っておく必要があります。
- Open Server アプリケーションでは、このルーチンを使用して、text、image、sensitivity、または boundary のカラムを取得することはできません。これらのカラムを取得するときには、blk_gettext を使用してください。

参照

blk_getrow, blk_gettext

blk_default

説明

カラムのデフォルト値を取得します。

構文

```
CS_RETCODE blk_default(blkdesc, colnum, buffer,
                buflen, outlen)
CS_BLKDESC    *blkdesc;
CS_INT        colnum;
CS_VOID       *buffer;
CS_INT        buflen;
CS_INT        *outlen;
```

パラメータ

blkdesc

バルク・コピー・オペレーションの制御ブロックの役目をしている CS_BLKDESC を指すポインタです。blk_alloc は CS_BLKDESC 構造体を割り付けます。

colnum

対象となるカラムの番号です。テーブル内の最初のカラムは、カラム番号 1 です。2 番目のカラム番号は 2、以下同様に続きます。

buffer

blk_default がデフォルト値を入れる領域を指すポインタです。

buflen

**buffer* データ領域の長さ (バイト単位) です。

outlen
整数変数を指すポインタです。

このパラメータを指定すると、blk_default は **outlen* をデフォルト値の長さ (バイト単位) に設定します。

デフォルト値が *buflen* バイトよりも大きい場合は、**outlen* の値を使用すると、値を保持するために必要なバイト数を判断できます。

戻り値

blk_default は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

アプリケーションがバルク・コピー・オペレーションを初期化する blk_init を呼び出していないと、blk_default は CS_FAIL を返します。

使用法

- blk_default は、クライアント・サイドのルーチンです。
- アプリケーションは、特定のターゲット・カラムに対してデフォルト値が定義されているかどうかを調べるために blk_default を呼び出すことができます。定義されている場合は、そのデフォルト値を調べます。

  この情報は、バルク・コピー・ローをデータベースに格納する準備をするときに有効です。カラムのデフォルト値を使用するかどうかを指定するには、アプリケーションが **datalen* と **indicator* の値を指定します (*datalen* と *indicator* は、blk_bind によってカラムへバインドされたプログラム変数のアドレスです)。「データベースへのバルク・コピーの場合の NULL 値の指定」(142 ページ) を参照してください。
- 対象となるカラムにデフォルト値がない場合には、blk_default は **outlen* を CS_NO_DEFAULT に設定して、CS_SUCCEED を返します。
- アプリケーションは、バルク・コピー・イン・オペレーションの場合にだけ、blk_default を使用してカラムのデフォルトを取得できます。blk_init(CS_BLK_IN) で CS_SUCCEED が返されるまでは、アプリケーションで blk_default を呼び出すことはできません。

参照

blk_bind, blk_describe, blk_init

blk_describe

説明

データベース・カラムの記述を取得します。

構文

CS_RETCODE blk_describe(blkdesc, colnum, datafmt)

```
CS_BLKDESC    *blkdesc;
CS_INT        colnum;
CS_DATAFMT    *datafmt;
```

パラメータ

blkdesc

バルク・コピー・オペレーションの制御ブロックの役目を果たしている CS_BLKDESC を指すポインタです。blk_alloc は CS_BLKDESC 構造体を割り付けます。

colnum

対象となるカラムの番号です。テーブル内の最初のカラムは、カラム番号 1 です。2 番目のカラム番号は 2、以下同様に続きます。

datafmt

CS_DATAFMT 構造体を指すポインタです。blk_describe は、*colnum* が参照するデータベース・カラムの記述を **datafmt* に入れます。

バルク・コピー・イン・オペレーションの場合、blk_describe は、CS_DATAFMT の各フィールドを次のように設定します。

表 4-4 : blk_describe がバルク・コピー・インの場合に設定する CS_DATAFMT のフィールド

フィールド名	blk_describe が設定する内容
name	存在する場合、NULL で終了されたカラム名。*namelen* が 0 の場合は NULL。
namelen	NULL ターミネータを含まない名前の実際の長さ。*name* が NULL の場合は 0。
datatype	カラムのデータ型を表す型定数。『Open Client Client-Library/C リファレンス・マニュアル』の「データ型」のページに記載されているすべての型定数が有効。ただし、CS_VARCHAR_TYPE と CS_VARBINARY_TYPE は除く。
maxlength	カラムに対するデータの可能な最大長。
scale	カラムの位取り。
precision	カラムの精度。

バルク・コピー・アウト・オペレーションの場合、blk_describe は CS_DATAFMT の各フィールドを次のように設定します。

表 4-5：バルク・コピー・アウトの場合に blk_describe により設定される CS_DATAFMT のフィールド

フィールド名	blk_describe が設定する内容
name	存在する場合、NULL で終了されたカラム名。*namelen* が 0 の場合は NULL。
namelen	NULL ターミネータを含まない名前の実際の長さ。 *name* が NULL の場合は 0。
datatype	カラムのデータ型。『Open Client Client-Library/C リファレンス・マニュアル』の「データ型」のページに記載されているすべてのデータ型が有効。
maxlength	カラムに対するデータの可能な最大長。
scale	カラムの位取り。
precision	カラムの精度。
status	次の記号値のビット・マスク。ビット単位で OR (論理和) をとる。 • そのカラムが NULL 値を持てることを示す CS_CANBENULL。 • 今まで公開されていた隠しカラムであることを示す CS_HIDDEN。隠しカラムは、CS_HIDDEN_KEYS プロパティをバルク記述子の親接続用に設定する場合に公開される。 • カラムが identity カラムであることを示す CS_IDENTITY。 • そのカラムがテーブルのキーの一部であることを示す CS_KEY。 • そのカラムがローのバージョン・キーの一部であることを示す CS_VERSION_KEY。
usertype	カラムの Adaptive Server Enterprise ユーザ定義データ型があれば、そのデータ型。*usertype* は、*datatype* の代わりではなく、*datatype* に追加されて設定される。
locale	データのロケール情報を持つ CS_LOCALE 構造体を指すポインタ。

戻り値

blk_describe は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

colnum が正しい結果カラムを表していない場合、blk_describe は CS_FAIL を返します。

使用法

- blk_describe は、クライアント・サイドのルーチンです。

- blk_describe はデータベース・カラムのフォーマットを記述します。アプリケーションは、この情報を使用して以下のことを行います。
  - (データベースのバルク・コピー・アウトのための) ローの取得に必要な割り付け格納領域のデータ型とサイズの条件を判断します。
  - プログラム変数のデータ型とデータベース・カラムとの互換性を判断します (cs_will_convert を呼び出して変換がサポートされるかどうかを判断し、必要ならばデータ長をチェックする)。
  - エラー・チェックを実行します。たとえば、バルク・コピー・アプリケーションのデバッグ・バージョンが blk_describe を呼び出して、テーブル・カラムのフォーマットについての予測を確認する場合があります。
- 通常、アプリケーションでは、互換性のあるプログラム変数の有効な型およびサイズを判断するときに、カラム記述が使用されます。
- CS_DATAFMT に関する詳細については、『Open Client Client-Library/C リファレンス・マニュアル』の「CS_DATAFMT 構造体」を参照してください。

参照

blk_default, blk_init

blk_done

説明

完了したバルク・コピー・オペレーション、または完了したバルク・コピー・バッチにマークを付けます。

構文

```
CS_RETCODE blk_done(blkdesc, type, outrow)

CS_BLKDESC    *blkdesc;
CS_INT        type;
CS_INT        *outrow;
```

パラメータ

blkdesc

バルク・コピー・オペレーションの制御ブロックの役目を果たす CS_BLKDESC を指すポインタです。blk_alloc は CS_BLKDESC 構造体を割り付けます。

type
次の記号値のいずれかです。

type の値	blk_done
CS_BLK_ALL	完了したバルク・コピー・イン・オペレーション、または完了したバルク・コピー・アウト・オペレーションにマークを付ける。
CS_BLK_BATCH	バッチ処理されたバルク・コピー・イン・オペレーションのローのバッチの完了を示すマークを付ける。
CS_BLK_CANCEL	バルク・コピー・バッチまたはバルク・コピー・オペレーションをキャンセルする。

outrow
整数変数を指すポインタです。*type* が CS_BLK_BATCH または CS_BLK_ALL である場合、blk_done は、**outrow* を、アプリケーションが最後に blk_done を呼び出してから Adaptive Server Enterprise にバルク・コピーしたローの数に設定します。*type* が CS_BLK_CANCEL の場合、**outrow* は 0 に設定されます。

戻り値

blk_done は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。
CS_PENDING	非同期ネットワーク I/O が有効。『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。

blk_done の失敗の主な理由は次のとおりです。

- *blkdesc* ポインタが正しくない。
- *type* の値が正しくない。

例

```
/*
 ** BulkCopyIn()
 */


CS_STATIC CS_RETCODE
 BulkCopyIn(connection)
 CS_CONNECTION    *connection;
 {
      CS_BLKDESC *blkdesc;
      CS_DATAFMT datafmt;/* variable descriptions */
```

```
        Blk_Data *dptr;/* data for transfer */
        CS_INT datalen[5];    /* variable data length */
        CS_INT      len;
        CS_INT numrows;

        /*
        ** Ready to start the bulk copy in now that all the
        ** connections have been made and have a table name.
        ** Start by getting the bulk descriptor initializing.
        */
        ...CODE DELETED.....

        /*

        ** Now to bind the variables to the columns and

        ** transfer the data
        */
        ...CODE DELETED.....

        /* ALL the rows sent so clear up */
        if (blk_done(blkdesc, CS_BLK_ALL, &numrows) == CS_FAIL)
        {
           ex_error("BulkCopyIn:blk_done() failed");
           return CS_FAIL;
        }
        if (blk_drop(blkdesc) == CS_FAIL)
        {
           ex_error("BulkCopyIn:blk_drop() failed");
           return CS_FAIL;
        }
         return CS_SUCCEED;
    }
```

使用法

- blk_done と呼ばれるクライアント・サイドのルーチンは、クライアントのみのアプリケーションとゲートウェイ・アプリケーションの両方で必要です。

  注意　CS_OPT_NOCOUNT を設定してから、接続でバルク・コピー・オペレーションを実行すると、blk_done が誤ってエラーをレポートします。

- *type* を CS_BLK_ALL に設定して blk_done を呼び出すと、バルク・コピー・オペレーションの完了を示すマークが付けられます。アプリケーションがバルク・コピー・オペレーションの完了を示すマークを付けた後は、blk_init を呼び出して新しいバルク・コピー・オペレーションを開始するまで、アプリケーションは Bulk-Library ルーチンを呼び出すことはできません (ただし、*blk_drop* と *blk_alloc* は除く)。
- *type* を CS_BLK_BATCH に設定して blk_done を呼び出すと、バルク・コピー・イン・オペレーションのローのバッチの完了を示すマークが付けられます。CS_BLK_BATCH は、バルク・コピー・イン・オペレーションのときにだけ有効です。
- *type* を CS_BLK_CANCEL に設定して blk_done を呼び出すと、現在のバルク・コピー・オペレーションがキャンセルされます。アプリケーションが最後に blk_done(CS_BLK_BATCH) を呼び出した後に転送されたローは、データベースに保存されません。アプリケーションがバルク・コピー・オペレーションをキャンセルした後は、blk_init を呼び出して新しいバルク・コピー・オペレーションを初期化するまで、バルク・コピー・ルーチンを呼び出すことはできません (ただし、blk_drop と blk_alloc は除く)。

バルク・コピー・イン・オペレーションでの *blk_done* の呼び出し

- アプリケーションがデータベースにデータをバルク・コピーするときに、ローが永続的に保存されるのは、blk_done を呼び出したときだけです。大量のデータを転送するときには、リカバリ性を高めるために、blk_done(CS_BLK_BATCH) を定期的に呼び出して、転送されたローをより小さな単位で「バッチ」できます。
- *n* ローを転送するたびに、または遠隔計測アプリケーションのように一定量のデータ転送後に小休止が発生するたびに、*type* を CS_BLK_BATCH に設定して blk_done を呼び出すことによって、アプリケーションはローをバッチ処理できます。これにより、アプリケーションが最後に blk_done を呼び出した後に送信されたすべてのローが永久に保存されます。

- 1 つのバッチのローを保存した後、アプリケーションの最初の blk_rowxfer または blk_rowxfer_mult の呼び出しによって、次のバッチが暗黙的に開始されます。
- アプリケーションは、最後のバッチのローを送るために type を CS_BLK_ALL に設定して blk_done を呼び出す必要があります。この呼び出しにより、ローが永久に保存され、バルク・コピー・オペレーションの完了を示すマークが付けられます。さらに、内部バルク・コピー・データ構造体がクリーンアップされます。

バルク・コピー・アウト・オペレーションでの *blk_done* の呼び出し

- バルク・コピー・アウト・オペレーションで最後のローを転送した後、アプリケーションは type を CS_BLK_ALL に設定して blk_done を呼び出します。これにより、バルク・コピー・オペレーションの完了を示すマークが付けられ、内部バルク・コピー・データ構造体がクリーンアップされます。

参照

blk_init, blk_rowxfer, blk_rowxfer_mult

blk_drop

説明

CS_BLKDESC 構造体の割り付けを解除します。

構文

```
CS_RETCODE blk_drop(blkdesc)

 CS_BLKDESC     *blkdesc;
```

パラメータ

blkdesc

blk_alloc により割り付けられている CS_BLKDESC を指すポインタです。

戻り値

blk_drop は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

例

```
/*
```

```
** BulkCopyIn()
*/

CS_STATIC CS_RETCODE
BulkCopyIn(connection)
CS_CONNECTION    *connection;
 {

CS_BLKDESC *blkdesc;
CS_DATAFMT datafmt;/* variable descriptions */
Blk_Data *dptr;/* data for transfer */
CS_INT datalen[5];    /* variable data length */
CS_INT      len;

CS_INT numrows;

/*
** Ready to start the bulk copy in now that all the
** connections have been made and have a table name.
** Start by getting the bulk descriptor initializing.
*/
...CODE DELETED.....

/*
** Now to bind the variables to the columns and
** transfer the data
*/
...CODE DELETED.....

/* ALL the rows sent so clear up */
if (blk_done(blkdesc, CS_BLK_ALL, &numrows) == CS_FAIL)
{
     ex_error("BulkCopyIn:blk_done() failed");
     return CS_FAIL;
}
if (blk_drop(blkdesc) == CS_FAIL)
{
     ex_error("BulkCopyIn:blk_drop() failed");
     return CS_FAIL;
}

return CS_SUCCEED;
}
```

使用法

- CS_BLKDESC は「バルク記述子構造体」とも呼ばれ、特定のバルク・コピー・オペレーションの情報が含まれます。
- バルク記述子構造体の割り付けが解除されると、再び使用することはできません。新しい CS_BLKDESC を割り付けるために、アプリケーションは blk_alloc を呼び出すことができます。
- blk_drop は、通常、blk_done の後に呼び出されます。blk_drop は、バルク・コピー・オペレーションで最後に呼び出される必要があります。

参照

blk_alloc, blk_done

blk_getrow

説明

フォーマットされたバルク・コピー・ローを取得して保管する、サーバ・サイドのルーチンです。

構文

```
CS_RETCODE blk_getrow(srvproc, blkdescp, rowp)

  SRV_PROC     *srvproc;
CS_BLKDESC    *blkdescp;
CS_BLK_ROW    *rowp;
```

パラメータ

srvproc

バルク・コピー・ローを送信するクライアントに関連付けられている SRV_PROC 構造体を指すポインタです。この構造体は、Open Server とクライアント間の通信およびデータを管理するために Server-Library が使用するすべての情報を含んでいます。

blkdescp

バルク・コピー・データの情報が含まれている CS_BLKDESC 構造体を指すポインタです。この構造体は、あらかじめ blk_alloc 呼び出しで割り付け、blk_init を呼び出して初期化しておく必要があります。この構造体は、送られてくるフォーマットされたバルク・コピー・ローを解釈するときに使用します。

rowp

フォーマットされたバルク・コピー・ローのための領域が含まれている CS_BLK_ROW 構造体を指すポインタです。blk_rowalloc を使用して事前に領域を割り付けてください。

CS_BLK_ROW 構造体は、クライアントから送信されたフォーマット済みバルク・コピー・ローを保持する隠し構造体です。

戻り値

blk_getrow は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_END_DATA	それ以上ローがない。
CS_BLK_HAS_TEXT	ローに text、image、sensitivity、または boundary のデータが含まれている。blk_gettext を使用して、text、image、sensitivity、または boundary のデータを取得する。CS_BLK_HAS_TEXT は、CS_SUCCEED と同様に正常な戻り値であることに注意。
CS_FAIL	ルーチンが失敗した。

使用法

- blk_getrow は、ゲートウェイ・アプリケーションで有用なサーバ・サイドのルーチンです。
- このルーチンは、送られてくるフォーマットされたバルク・コピー・ローを、*rowp* が指す CS_BLK_ROW 構造体にコピーします。ロー・データは次の blk_getrow 呼び出しまで保存されます。アプリケーションは blk_rowalloc を使用してそのローに対する領域をあらかじめ割り付けておく必要があります。
- blk_getrow によって 1 つのローを受け取った後は、blk_colval を使用して、任意のフィールド (text、image、sensitivity、boundary のフィールドを除く) の内容を調べることができます。
- text、image、sensitivity、および boundary のフィールドを取得するには、blk_gettext を使用してください。
- その後に blk_sendrow ルーチンを使用して、バルク・コピー・ローを他のサーバに送ることができます。
- アプリケーションは、送られてくるすべてのローを、ローがなくなるまで blk_getrow を使用して読み込む必要があります。
- blk_getrow から CS_END_DATA が返されたら、アプリケーションは blk_rowdrop を使用して、そのローに対して割り付けられた領域を削除する必要があります。

参照

blk_colval, blk_gettext, blk_rowalloc

blk_gettext

説明

送られてくるフォーマットされたバルク・コピー・ローの text、image、sensitivity、または boundary のデータを取得する、サーバ・サイドのルーチンです。

構文

```
CS_RETCODE blk_gettext(srvproc,blkdescp, rowp, bufp,  bufsize, outlenp)

SRV_PROC        *srvproc;
CS_BLKDESC      *blkdescp;
CS_BLK_ROW      *rowp;
CS_BYTE         *bufp;
CS_INT          bufsize;
CS_INT          *outlenp;
```

パラメータ

srvproc

バルク・コピー・ローを送信するクライアントに関連付けられている SRV_PROC 構造体を指すポインタです。この構造体は、Open Server のアプリケーションとクライアント間の通信およびデータを管理するための Server-Library が使用するすべての情報を含んでいます。

blkdescp

バルク・コピー・データの情報が含まれている CS_BLKDESC 構造体を指すポインタです。この構造体は、あらかじめ blk_alloc 呼び出しで割り付け、blk_init を呼び出して初期化しておく必要があります。この構造体は、送られてくるフォーマットされたバルク・コピー・ローを解釈するときに使用します。

rowp

前回の blk_getrow 呼び出しでクライアントから読み込まれ、フォーマットされたバルク・コピー・ローを指すポインタです。

CS_BLK_ROW 構造体は、クライアントから送信されたフォーマット済みバルク・コピー・ローを保持する隠し構造体です。

bufp

Bulk-Library によって text、image、sensitivity、または boundary データが入れられるアプリケーション・バッファを指すポインタです。

bufsize

bufp が指す領域のバイト単位のサイズです。

outlenp

CS_INT 変数を指すポインタです。この変数は、blk_gettext を使用して実際に読み込まれるバイトの数に設定されます。*bufsize* よりも小さい値となることもあります。ローのすべての text、image、sensitivity、または boundary の部分が読み込まれたかどうかを調べるには、CS_END_DATA のリターン・コードをチェックしてください。**outlenp* 値が *bufsize* よりも小さくても、必ずしもローの終わりを示さないことがあります。たとえば、outlenp 値が、ローの最後のカラムではない text、image、sensitivity、または boundary の部分の終わりを示すことがあります。

戻り値

blk_gettext は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_END_DATA	現在送られてきているバルク・コピー・ローに、それ以上の text、image、sensitivity、または boundary のデータがない。blk_getrow を呼び出して次のバルク・コピー・ローを取得する。
CS_FAIL	ルーチンが失敗した。

使用法

- blk_gettext は、ゲートウェイ・アプリケーションで有用なサーバ・サイドのルーチンです。
- このルーチンは、blk_getrow と blk_colval を使用して、フォーマットされたバルク・コピー・ローを受け取り、それらを Adaptive Server Enterprise に送ります。このルーチンは、ローの text、image、sensitivity、または boundary の部分を取得します。
- バルク・コピー・ローはフォーマットされています。そのため、すべての text、image、sensitivity、または boundary のフィールドは、これ以外の型のフィールドの後のローの終わりにあります。Adaptive Server Enterprise にローを送るには、最初に blk_getrow を呼び出して、ローの中でこれ以外の型のフィールドを含むすべての部分を取得してください。次に blk_colval を呼び出し、これ以外の型のフィールドを含んでいるローの部分を取得して保管します。このデータの送り先を決めてから、blk_sendrow を使用してリモート・サーバに送ってください。次に blk_gettext を呼び出して、text、image、sensitivity、または boundary のデータをアプリケーション・バッファにコピーします。最後に blk_sendtext を呼び出して、リモート・サーバへこの情報を送ります。

- 送られてくるバルク・コピー・ローに text、image、sensitivity、または boundary のフィールドが含まれている場合、blk_getrow は、CS_BLK_HAS_TEXT を返します。
- ローに text、image、sensitivity、または boundary のフィールドが含まれていない場合に blk_gettext を呼び出しても、誤りではありません。このルーチンは CS_END_DATA を返します。
- このルーチンは、blk_getrow の後に呼び出してください。また、バルク・コピー・ローを完全に読み込むために、CS_END_DATA を返すまでこのルーチンを呼び出す必要があります。
- サーバにローを送る前に、ゲートウェイ・アプリケーションは、blk_init を呼び出して、バルク・コピー・オペレーションの設定をしておく必要があります。
- バルク・コピー・オペレーションの初期化の対象になったテーブルと、クライアントがバルク・コピー・インしているテーブルは、同じテーブルでなければなりません。

参照

blk_colval, blk_getrow, blk_gettext, blk_sendtext

blk_init

説明

バルク・コピー・オペレーションを開始します。

構文

```
CS_RETCODE blk_init(blkdesc, direction, tablename,
        tnamelen)

CS_BLKDESC    *blkdesc;
CS_INT        direction;
CS_CHAR       *tablename;
CS_INT        tnamelen;
```

パラメータ

blkdesc

バルク・コピー・オペレーションを制御する CS_BLKDESC を指すポインタです。アプリケーションは、blk_alloc を呼び出して、CS_BLKDESC を割り付けます。

CS_BLKDESC の親接続は、blk_init を呼び出す時点でオープンされている必要があります。また、保留中の結果があってはなりません。

direction

バルク・コピー・オペレーションの方向を示すため、次の値のいずれか 1 つを指定します。

direction の値	blk_init
CS_BLK_IN	クライアントからサーバへローをアップロードする、バルク・コピー・オペレーションを開始する。
CS_BLK_OUT	サーバからクライアントへローをダウンロードする、バルク・コピー・オペレーションを開始する。

tablename

対象となるテーブルの名前を指すポインタです。有効なサーバ・テーブル名は、すべて使用できます。テーブル名にはピリオド (.) やコロン (:) を含めることはできません。

tnamelen

tablename* の長さ (バイト単位) です。tablename* が NULL で終了している場合には、*tnamelen* を CS_NULLTERM として渡してください。

戻り値

blk_init は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。
CS_PENDING	非同期ネットワーク I/O が有効。『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。

失敗の主な理由は、存在しないテーブルを指定することです。

例

```
/*
 ** BulkCopyIn()
 ** Ex_tabname is globally defined.
 */
 CS_STATIC CS_RETCODE
 BulkCopyIn(connection)
 CS_CONNECTION    *connection;
 {
     CS_BLKDESC *blkdesc;
     CS_DATAFMT datafmt;/* variable descriptions */
     Blk_Data *dptr;/* data for transfer */
     CS_INT datalen[5];    /* variable data length */
     CS_INT      len;
```

```
        CS_INT numrows;

        /*
        ** Ready to start the bulk copy in now that all the
        ** connections have been made and have a table name.
        ** Start by getting the bulk descriptor and
        ** initializing.
        */
        if (blk_alloc(connection, BLK_VERSION_100, &blkdesc)
            != CS_SUCCEED)
        {
            ex_error("BulkCopyIn:blk_alloc() failed");
            return CS_FAIL;
        }

        if (blk_init(blkdesc, CS_BLK_IN,
                   Ex_tabname, strlen(Ex_tabname)) == CS_FAIL)
        {
            ex_error("BulkCopyIn:blk_init() failed");
            return CS_FAIL;
        }

        /*
        ** Bind the variables to the columns and send the rows,
        ** and then clean up.
        */
        ...CODE DELETED.....

        return CS_SUCCEED;
    }
```

使用法

- blk_init は、バルク・コピー・オペレーションを開始します。
- blk_init は、クライアント・サイドのルーチンです。ただし、クライアントのみのアプリケーションとゲートウェイ・アプリケーションの両方で必要です。
- 同じ接続で複数の CS_BLKDESC 構造体と CS_COMMAND 構造体が存在できますが、一度にアクティブにできる CS_BLKDESC 構造体または CS_COMMAND 構造体は 1 つだけです。
  - 接続を他のオペレーションで使用するには、blk_init で開始したバルク・コピー・オペレーションを完了させる必要があります。
  - 他の Client-Library または Bulk-Library コマンドの結果を開始、送信、処理するために接続が使用されるときには、バルク・コピー・オペレーションは起動できません。

- バルク・コピー・オペレーションが完了したら、バルク・コピー・オペレーションの完了を示すマークを付け、内部の Bulk-Library データ構造体をクリーンアップするために、アプリケーションは、*type* を CS_BLK_ALL に設定して blk_done を呼び出す必要があります。

参照　blk_alloc, blk_bind, blk_done, blk_rowxfer_mult

blk_props

説明　バルク記述子構造体のプロパティを設定または取得します。

構文

```
CS_RETCODE blk_props(blkdesc, action, property,
        buffer, buflen, outlen)

CS_BLKDESC    *blkdesc;
CS_INT        action;
CS_INT        property;
CS_VOID       buffer;
CS_INT        buflen;
CS_INT        *outlen;
```

パラメータ

blkdesc

CS_BLKDESC 構造体を指すポインタです。バルク記述子構造体には、バルク・コピー・オペレーションに関する情報が含まれます。blk_alloc はバルク記述子構造体を割り付けます。

action

次の記号定数のいずれかです。

action の値	blk_props
CS_SET	プロパティの値を設定する。
CS_GET	プロパティの値を取得する。
CS_CLEAR	プロパティの値をデフォルト値にリセットすることによってクリアする。

property

対象であるプロパティを示す定数です。表 4-6 (165 ページ) に、有効な *property* 定数の一覧を示し、各プロパティについて説明します。

buffer
プロパティ値を設定する場合、*buffer* はプロパティの設定に使用する値を指します。

プロパティ値を取得する場合、*buffer* は blk_props が要求された情報を入れる領域を示します。

値の C データ型はプロパティによって異なります。対象となるプロパティのデータ型については、表 4-6 (165 ページ) を参照してください。

buflen
一般に、*buflen* は **buffer* のバイト単位の長さです。

プロパティ値を設定する場合、**buffer* の値が NULL で終了している場合には、*buflen* を CS_NULLTERM として渡します。

**buffer* が固定長または記号値の場合、*buflen* を CS_UNUSED として渡します。

outlen
整数変数を指すポインタです。

プロパティ値を設定する場合、*outlen* は使用されません。この場合、NULL として渡してください。

プロパティ値が取得され、*outlen* が提供された場合、blk_props は **outlen* を要求された情報の長さ (バイト単位) に設定します。

情報の大きさが *buflen* バイトを超える場合は、**outlen* の値を使用すると、その情報を保持するために必要なバイト数を判断できます。

戻り値

blk_props は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

使用法

- バルク記述子プロパティは特定のバルク・コピー・オペレーションの内容を定義します。
- アプリケーションが Bulk-Library プロパティを設定する場合、このアプリケーションは、blk_alloc を呼び出してバルク記述子構造体を割り付けた後、特定のバルク・コピー・オペレーションを開始するために blk_init を呼び出す前に、プロパティを設定する必要があります。
- アプリケーションは、blk_props を使用して次のプロパティを設定または取得できます。

表 4-6：Client/Server バルク記述子プロパティ

プロパティ名	説明	*buffer の値	適用対象	注意
BLK_CONV	クライアントによって実行される文字セット変換。	CS_TRUE または CS_FALSE。	コピーインのみ	現在のサーバ接続で文字セット変換が行われないようにするには、CS_NOCHARSETCNV_REQD を CS_TRUE に設定する。『Open Client Client-Library/C リファレンス・マニュアル』の「ct_con_props」を参照。
BLK_CUSTOM_CLAUSE	insert bulk コマンドの既存の with 句の後に追加するアプリケーション固有のカスタム SQL 句。	カスタム句を含む文字列。	コピーインのみ	カスタム SQL 句をサポートするサーバ・バージョンのみによってサポートされる。現時点では内部の製品のみによって使用されている。

プロパティ名	説明	*buffer の値	適用対象	注意
BLK_IDENTITY	テーブルの identity カラムの値が、挿入される各ローについて明示的に指定されているかどうかを示す。 BLK_IDSTARTNUM がバルク・コピー・イン・オペレーションに対して設定されている場合、このプロパティは CS_TRUE に設定できない。	CS_TRUE または CS_FALSE。 デフォルトは CS_FALSE で、identity 値が次のいずれかであることを示す。 • BLK_IDSTARTNUM が示す開始の値から計算された値 • データが挿入されたときに、テーブルの既存の identity 値に基づいて Adaptive Server Enterprise によって計算された値	コピーインのみ	
BLK_IDSTARTNUM	挿入されたローの identity カラムの開始の値。最初に挿入されたローではこの値が使われ、以降の各ローではこの値が増加していく。 BLK_IDENTITY がバルク・コピー・イン・オペレーションに対して CS_TRUE に設定されている場合、このプロパティは設定できない。	identity の最初の値を含む CS_NUMERIC 変数。 デフォルトはない。	コピーインのみ	

プロパティ名	説明	*buffer の値	適用対象	注意
BLK_NOAPI_CHK	Bulk-Library 呼び出しで、不正なパラメータ値とステータス遷移を検出するためのパラメータ・チェックとエラー・チェックを有効にできるかどうかを示す。	CS_TRUE または CS_FALSE。 デフォルトは CS_FALSE で、エラー・チェックが行われることを意味する。	コピーインおよびコピーアウト	
BLK_PARTITION	BCP_IN および BCP_OUT オペレーションの BCP パーティションをサポートするプロパティ。	パーティションの名前を含む文字列。	コピーインおよびコピーアウト	
BLK_SENSITIVITY_ LBL	テーブルの *sensitivity* カラムがバルク・コピー・オペレーションに含まれるかどうかを示す。	CS_TRUE または CS_FALSE (デフォルト)。	コピーインおよびコピーアウト	Secure Adaptive Server Enterprise のみ
BLK_SLICENUM	分割されたテーブルへのバルク・コピーに使用する。コピーされたローが挿入されるテーブルの分割番号を指定する。	分割番号を表す正の値を含む CS_INT 変数。 デフォルトは CS_UNUSED で、Adaptive Server Enterprise がランダムに分割番号を選択することを示す。	コピーインのみ	

BLK_CUSTOM_CLAUSE プロパティ

ローをプロキシ・テーブルに挿入する select into existing table 文を処理するために、ASE は Bulk-Library を使用してバルク・コピー・オペレーションを生成します。ただし、完全なロギングを通常のバルク・コピー・オペレーションに使用することはできません。BLK_CUSTOM_CLAUSE は、Adaptive Server が通常のバルク・コピー・オペレーションと、insert into 文から発生し、プロキシ・テーブルに影響するバルク・コピー・オペレーションとを区別できるようにします。このような insert into 文から発生するバルク・コピー・オペレーションは、BLK_CUSTOM_CLAUSE によって指定されるカスタム句に追加できます。Adaptive Server はこの句を検出し、完全なロギングを実行できます。

- select into オペレーションは、Adaptive Server の select into/bulkcopy/pllsort データベース・オプションがオンに設定されている場合にのみ許可されます。
- select into オペレーションを完全にロギングするには、Adaptive Server の full logging for select into データベース・オプションをオンに設定する必要があります。

例

BLK_CUSTOM_CLAUSE は blk_props によって設定されます。

```
blk_props(blkdesc, CS_SET, BLK_CUSTOM_CLAUSE,
(CS_VOID *)"from select_into", CS_NULLTERM, NULL);
```

Adaptive Server は、指定されたカスタム句が付加されたバルク・コピー・オペレーションを生成します。

```
insert bulk mydb.mytable with nodescribe from
select_into
```

ここで、mydb と mytable は、影響を受けるデータベースとテーブルです。

BLK_IDENTITY プロパティ

- BLK_IDENTITY は、テーブルの identity カラムがバルク・コピー・イン・オペレーションに含まれるかどうかを決定します。
- BLK_IDENTITY はバルク・コピー・アウト・オペレーションには影響しません。
- BLK_IDENTITY が CS_TRUE の場合、アプリケーションは identity カラムへのデータを提供しなければなりません。

  BLK_IDENTITY が CS_FALSE の場合、アプリケーションは identity カラムへのデータを提供する必要はありません。この場合、サーバがデフォルト値をカラムに提供します。

- BLK_IDENTITY を有効にするには、対象となるデータベース・テーブルに対して identity_insert を on に設定します。これは、identity カラムへの値の挿入を許可します。バルク・コピー・オペレーションが完了すると、テーブルの identity_insert オプションはオフに設定されます。

  『ASE リファレンス・マニュアル』を参照してください。

BLK_NOAPI_CHK プロパティ

- BLK_NOAPI_CHK を、CS_TRUE に設定すると、Bulk-Libary 呼び出しのパラメータ・チェックとステータス・チェックを無効にできます。デフォルトは CS_FALSE であり、各 Bulk-Library 呼び出しのパラメータ・チェックとステータス・チェックは有効になります。エラー・チェックは、次の 2 種類です。
  - 「パラメータ・チェック」は、アプリケーションが正しいパラメータと正しいパラメータの組み合わせを呼び出しに渡したかどうかを判断します。
  - 「ステータス・チェック」は、必要な順序で呼び出しが行われたことを確認します。たとえば、blk_init は blk_bind よりも前に呼び出される必要があります。

  デフォルトのエラー・チェックは、アプリケーションが適切な方法で Bulk-Library ルーチンを呼び出していることを確認します。API チェックが有効になっていると、アプリケーションが使用上のエラーを処理し、エラーを発見したルーチンが CS_FAIL を返すときに、エラー・メッセージが表示されます。

  警告！ API チェックが無効になっていると、Bulk-Library の使用方法エラーが発生し、これによって予期しない動作や、プログラムのクラッシュが発生することがあります。

- アプリケーションが十分にテストされ、完全にデバッグされていれば、API チェックを無効にした方がパフォーマンスを改善できます。また、Bulk-Library は最大限の効果を引き出すために Client-Library を内部で呼び出すことから、Client-Library での API チェックを無効にする必要もあります (無効にするには CS_NOAPI_CHK コンテキスト・プロパティを CS_TRUE に設定して ct_config を呼び出します)。
- BLK_NOAPI_CHK は、ネットワーク・エラーや変換オーバフローなどのエラーのように、十分に動作するアプリケーションでも起こり得るエラーのテストには影響しません。

BLK_SENSITIVITY_LBL プロパティ

- BLK_SENSITIVITY_LBL は、Secure Adaptive Server Enterprise への、および Secure Adaptive Server Enterprise からのバルク・コピー・オペレーションを実行するアプリケーションで有効です。
- BLK_SENSITIVITY_LBL は *sensitivity* カラムのデータがバルク・コピー・オペレーションに含まれるかどうかを決定します。デフォルトでは、*sensitivity* カラム・データはバルク・コピー・オペレーションに含まれません。
- BLK_SENSITIVITY_LBL は、バルク・コピー・インとバルク・コピー・アウトの両方に影響します。
- BLK_SENSITIVITY_LBL が CS_TRUE の場合、アプリケーションはバルク・コピー・イン・オペレーションで *sensitivity* カラムにデータを提供しなければなりません。またバルク・コピー・アウト・オペレーションでは、アプリケーションは *sensitivity* カラムのデータを受け取ります。

  BLK_SENSITIVITY_LBL が CS_FALSE の場合、アプリケーションはバルク・コピー・イン・オペレーションで *sensitivity* カラムにデータを提供する必要はありません。また、バルク・コピー・アウト・オペレーションでは、*sensitivity* カラムからのデータを受け取りません。
- BLK_SENSITIVITY_LBL は、Secure Adaptive Server Enterprise のコピーにのみ適用できます。BLK_SENSITIVITY_LBL が CS_TRUE の場合に、アプリケーションが標準の Adaptive Server Enterprise に対してバルク・コピー・オペレーションをしようとすると、blk_init は失敗します。
- *sensitivity* カラムへのコピーを実行するアプリケーション・ユーザに対して、Secure Adaptive Server Enterprise 上で bcpin_labels_role の役割がアクティブにされている必要があります。bcpin_labels_role が接続のユーザに対してアクティブでない場合、blk_init は失敗します。
- Secure Adaptive Server Enterprise のマニュアルを参照してください。

BLK_PARTITION プロパティ

- 提供できる名前は 1 つだけです。単一の BLKLIB オペレーションは常に、テーブル全体または単一のパーティションを処理します。パーティション名を指定しないと、BLKLIB は、特定のパーティションではなくテーブル全体を処理します。

- このプロパティは BCP_IN オペレーションと BCP_OUT オペレーションの両方に対して使用できます。BLK_PARTITION または BLK_SLICENUM のどちらかを使用できます。一方を設定すると、他方はクリアされます。
- BLK_PARTITION プロパティでは、CS_VERSION_155 または BLK_VERSION_155 を設定する必要はありません。

参照　blk_alloc, blk_init

blk_rowalloc

説明　フォーマットされたバルク・コピー・ローのための領域を割り付けるサーバ・サイドのルーチンです。

構文　CS_RETCODE blk_rowalloc(srvproc, row)

```
SRV_PROC       *srvproc;
CS_BLK_ROW     **row;
```

パラメータ

srvproc

フォーマットされたバルク・コピー・ローを送信するクライアントに関連付けられている SRV_PROC 構造体を指すポインタです。この構造体は、Open Server とクライアント間の通信およびデータを管理するために Server-Library が使用するすべての情報を含んでいます。

row

CS_BLK_ROW 構造体を指すポインタへのポインタです。

CS_BLK_ROW 構造体は、クライアントから送信されたフォーマット済みバルク・コピー・ローを保持する隠し構造体です。

戻り値　blk_rowalloc は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

使用法

- blk_rowalloc は、ゲートウェイ・アプリケーションで有用なサーバ・サイドのルーチンです。
- このルーチンは、blk_getrow がフォーマットされたバルク・コピー・ローを入れる領域を割り付けます。

- ローの領域は、すべての blk_getrow 呼び出しによって使用されます。
- すべてのローが取得され、リモート・サーバに送られたら、blk_rowdrop を呼び出して、ローに割り付けられた領域を削除してください。

参照

blk_getrow, blk_rowdrop, blk_gettext

blk_rowdrop

説明

フォーマットされたバルク・コピー・ローに対して、以前に割り付けられた領域を解放するサーバ・サイドのルーチンです。

構文

```
CS_RETCODE blk_rowdrop(srvproc, row)

SRV_PROC       *srvproc;
CS_BLK_ROW     *row;
```

パラメータ

srvproc

フォーマットされたバルク・コピー・ローを送信するクライアントに関連付けられている SRV_PROC 構造体を指すポインタです。この構造体は、Open Server のアプリケーションとクライアント間の通信およびデータを管理するために Server-Library が使用するすべての情報を含んでいます。

row

blk_rowalloc の呼び出しによって割り付けられた CS_BLK_ROW 構造体 (隠し構造体) を指すポインタです。

戻り値

blk_rowdrop は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。

使用法

- blk_rowdrop は、ゲートウェイ・アプリケーションで有用なサーバ・サイドのルーチンです。
- このルーチンは、blk_rowalloc によって以前に割り付けられた領域を解放します。
- このルーチンは、すべてのフォーマットされたバルク・コピー・ローが取得され、リモート・サーバに送られた後に呼び出さなければなりません。

参照　blk_getrow, blk_rowalloc, blk_gettext

blk_rowxfer

説明　ローの数を指定または受信せずに、バルク・コピー・オペレーション時に 1 つ以上のローを転送します。

構文

```
CS_RETCODE blk_rowxfer(blkdesc)

CS_BLKDESC      *blkdesc;
```

パラメータ

blkdesc

バルク・コピー・オペレーションの制御ブロックの役目を果たしている CS_BLKDESC を指すポインタです。blk_alloc は CS_BLKDESC 構造体を割り付けます。

戻り値　blk_rowxfer は、次の値を返します。

表 4-7：blk_rowxfer の戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。
CS_PENDING	非同期ネットワーク I/O が有効。『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。
CS_BLK_HAS_TEXT	ローの中に、blk_textxfer で転送されることを示すマークが付けられたカラムが 1 つ以上ある。 アプリケーションは、blk_rowxfer を呼び出して次のローを転送する前に、blk_textxfer を呼び出して、これらのカラムのローに対するデータを転送しなければならない。
CS_END_DATA	データベースからデータをコピー・アウトする場合、blk_rowxfer は CS_END_DATA を返して、すべてのローが転送されたことを示す。 データベースにデータをコピーする場合、blk_rowxfer は CS_END_DATA を返さない。

戻り値	意味
CS_ROW_FAIL	ローのフェッチ中に復元可能なエラーが発生した。 バルク・コピー・アウト・オペレーションだけに適用される。 リカバリ可能なエラーとは、メモリ割り付けエラーや、ロー値をプログラム変数にコピーするときに発生する変換エラー (送信先バッファのオーバフローなど) などである。バッファ・オーバフロー・エラーの場合、blk_rowxfer は対応する **indicator* 変数を 0 よりも大きな値に設定する。インジケータ変数はアプリケーションの blk_bind の呼び出し時に指定されている必要がある。 blk_rowxfer が CS_ROW_FAIL を返した場合は、アプリケーションは blk_rowxfer の呼び出しを続行してローの取得を続行するか、または、ct_cancel を呼び出して残りの結果をキャンセルする。

例

```
/*
 ** BulkCopyIn()
 ** BLKDATA and DATA_END are defined in the bulk copy
 ** example program.
 */

 CS_STATIC CS_RETCODE
 BulkCopyIn(connection)
 CS_CONNECTION    *connection;
 {
     CS_BLKDESC *blkdesc;
     CS_DATAFMT datafmt;/* variable descriptions */
     Blk_Data *dptr;/* data for transfer */
     CS_INTdatalen[5];/* variable data length */
     CS_INT      len;
     CS_INT numrows;


    /*
     ** Ready to start the bulk copy in now that all the
     ** connections have been made and have a table name.
     ** Start by getting the bulk descriptor initializing.
     */
     ...CODE DELETED.....
    /*
     ** Now to bind the variables to the columns and
```

```
** transfer the data
*/
datafmt.locale = 0;
datafmt.count = 1;
dptr = BLKDATA;
while (dptr->pub_id != DATA_END)
{
    datafmt.datatype = CS_INT_TYPE;
    datafmt.maxlength = sizeof(CS_INT);
    datalen[0] = CS_UNUSED;

   if (blk_bind(blkdesc, 1, &datafmt, &dptr->pub_id,
        &datalen[0], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn:blk_bind(1) failed");
        return CS_FAIL;
    }
    datafmt.datatype = CS_CHAR_TYPE;
    datafmt.maxlength = MAX_PUBNAME - 1;
    datalen[1] = strlen(dptr->pub_name);
    if (blk_bind(blkdesc, 2, &datafmt, dptr->pub_name,
         &datalen[1], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn:blk_bind(2) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_PUBCITY - 1;
    datalen[2] = strlen(dptr->pub_city);
    if (blk_bind(blkdesc, 3, &datafmt, dptr->pub_city,
         &datalen[2], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn:blk_bind(3) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_PUBST - 1;
    datalen[3] = strlen(dptr->pub_st);
    if (blk_bind(blkdesc, 4, &datafmt, dptr->pub_st,
         &datalen[3], NULL) != CS_SUCCEED)
    {
        ex_error("BulkCopyIn:blk_bind(4) failed");
        return CS_FAIL;
    }
    datafmt.maxlength = MAX_BIO - 1;
    datalen[4] = strlen((char *)dptr->pub_bio);
    if (blk_bind(blkdesc, 5, &datafmt, dptr->pub_bio,
         &datalen[4], NULL) != CS_SUCCEED)
    {
```

```
                ex_error("BulkCopyIn:blk_bind(5) failed");
                return CS_FAIL;
            }
            if (blk_rowxfer (blkdesc) == CS_FAIL)
            {
                ex_error("BulkCopyIn:blk_rowxfer() failed");
                return CS_FAIL;
            }
            dptr++;
        }

        /* ALL the rows sent so clear up */
        ...CODE DELETED.....

        return CS_SUCCEED;
    }
```

使用法

- blk_rowxfer は、クライアント・サイドのルーチンです。
- blk_rowxfer の実行結果は、NULL *row_count* パラメータで blk_rowxfer_mult を呼び出した場合の結果と同じです。
- この章の blk_rowxfer_mult を参照してください。

参照

blk_bind, blk_rowxfer_mult, blk_textxfer

blk_rowxfer_mult

説明

バルク・コピー・オペレーションの場合に 1 つ以上のローを転送します。

構文

CS_RETCODE blk_rowxfer_mult(blkdesc, row_count)

```
CS_BLKDESC    *blkdesc;
CS_INT        *row_count;
```

パラメータ

blkdesc

バルク・コピー・オペレーションの制御ブロックの役目を果たしている CS_BLKDESC を指すポインタです。blk_alloc は CS_BLKDESC 構造体を割り付けます。

row_count

CS_INT 変数または NULL を指すポインタです。

バルク・コピー・アウト・オペレーションでは、blk_rowxfer_mult は読み込まれたローの数を設定した **row_count* を返します。*row_count* が NULL の場合には、アプリケーションはこの情報を使用できません (アプリケーションは blk_done を呼び出して、直前の blk_done 呼び出しの後に行われた blk_rowxfer_mult 呼び出しの累積回数から、転送されたローの数を判断できます。ただし、ロー・カウント変数を使用する方が簡単です)。

バルク・コピー・イン・オペレーションでは、blk_rowxfer_mult は **row_count* で指定された数のローをサーバに送信します。*row_count* が NULL または **row_count* が 0 の場合は、それ以前の blk_bind への呼び出しの *datafmt->count* で指定された数のローがサーバに送信されます。

row_count は配列バインドを実行するアプリケーションで使用されます。「配列バインド」(143 ページ) を参照してください。

戻り値

blk_rowxfer_mult は、次の値を返します。

表 4-8 : blk_rowxfer_mult の戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。
CS_PENDING	非同期ネットワーク I/O が有効。『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」を参照する。
CS_BLK_HAS_TEXT	ローの中に、blk_textxfer で転送されることを示すマークが付けられたカラムが 1 つ以上ある。 アプリケーションは、blk_textxfer を呼び出して次のローを転送する前に、blk_rowxfer_mult を呼び出して、これらのカラムのローのデータを転送する必要がある。
CS_END_DATA	データベースからデータをコピー・アウトする場合、blk_rowxfer_mult は CS_END_DATA を返して、すべてのローが転送されたことを示す。 データベースへデータをコピー・インする場合、blk_rowxfer_mult は CS_END_DATA を返さない。

戻り値	意味
CS_ROW_FAIL	ローのフェッチ中に復元可能なエラーが発生した。バルク・コピー・アウト・オペレーションだけに適用される。 blk_rowxfer_mult は **row_count* を設定して、転送されたローの数 (エラーのあるローを含む) を示す。エラーのあるローより後のローは転送されない。次に blk_rowxfer_mult を呼び出すと、エラーが発生したローの次のローから読み込みを開始する。 リカバリ可能なエラーとは、メモリ割り付けエラーや、ロー値をプログラム変数にコピーするときに発生する変換エラー (送信先バッファのオーバフローなど) などである。バッファ・オーバフロー・エラーの場合、blk_rowxfer_mult は対応する **indicator* 変数を 0 よりも大きな値に設定する。インジケータ変数は、アプリケーションの blk_bind の呼び出しで事前に指定されている必要がある。 blk_rowxfer_mult が CS_ROW_FAIL を返すと、アプリケーションは blk_rowxfer_mult の呼び出しを続行してローの取得を続けるか、または ct_cancel を呼び出して残りの結果をキャンセルできる。

blk_rowxfer_mult の失敗の主な理由は、変換エラーです。

使用法

- blk_rowxfer_mult は、クライアント・サイドのルーチンです。
- アプリケーションは、blk_rowxfer_mult を呼び出して (blk_bind によってバインドされた) プログラム変数とデータベース・テーブルとの間でローを転送します。
  - バルク・コピー・イン・オペレーションでは、blk_rowxfer_mult はプログラム変数からデータベースへデータをコピーします。
  - バルク・コピー・アウト・オペレーションでは、blk_rowxfer_mult はデータベースからデータをコピーし、このデータをプログラム変数に入れます。
- アプリケーション変数は、blk_rowxfer_mult でテーブル・カラムの内容の読み込みと書き込みができるように、最初に blk_bind によってテーブル・カラムにバインドされる必要があります。

blk_rowxfer_mult とバルク・コピー・イン・オペレーション

- データベースにローを転送するために、アプリケーションは blk_rowxfer_mult を繰り返し呼び出して、プログラム変数からデータベース・テーブルに値を転送します。データをデータベース・テーブルに転送するときに使用する Bulk-Library 呼び出し順序については、「バルク・コピー・イン・オペレーションのプログラム構造」(116 ページ) を参照してください。
- バルク・コピー・イン・オペレーション中に、blk_rowxfer_mult の **row_count* パラメータの値が、blk_bind に (*datafmt–>count* で) 渡された配列の長さを上書きします。呼び出しごとに転送されるローの数は、次のように判断できます。
  - アプリケーションが *row_count* パラメータとしてロー・カウント変数のアドレスを渡すと、blk_rowxfer_mult は、*datafmt–>count* または **row_count* のどちらか少ない件数のローを転送します。
  - アプリケーションが *row_count* を NULL で渡す場合、blk_rowxfer_mult は常に *datafmt–>count* の数のローを転送します。

  たとえば、アプリケーションが 103 個のローをアップロードしており、配列バインドを使用して 1 回に 10 個のローを転送していた場合には、アプリケーションは次のように処理を行います。
  - blk_bind のすべての呼び出しで *datafmt–>count* を 10 にして渡します。
  - blk_rowxfer_mult の 1 回目から 10 回目までの呼び出しでは、**row_count* を 10 に設定します。
  - blk_rowxfer_mult の最後の呼び出しに対して、**row_count* を 3 に設定します。
- 大量の text や image のカラム値を含むロー・データをアップロードするには、配列バインドは使わずに blk_textxfer を blk_rowxfer_mult とともに使用すると、一度に大量の値のまとまりを送信できます。詳細については、「大量の text 値または image 値のまとまりでの転送」(180 ページ) を参照してください。
- blk_rowxfer_mult が CS_FAIL を返す場合には、バルク・コピー・イン・オペレーションは自動的には終了しません。問題のあるローを訂正するか破棄した後で、アプリケーションは blk_rowxfer_mult の呼び出しを続行できます。

blk_rowxfer_mult とバルク・コピー・アウト・オペレーション

- データベースからローを転送するために、アプリケーションは blk_rowxfer_mult を繰り返し呼び出してサーバからカラム値を読み込み、それらをプログラム変数に入れます。データベース・テーブルからデータを読み込むときの Bulk-Library 呼び出し順序については、「バルク・コピー・アウト・オペレーションのプログラム構造」(118 ページ) を参照してください。
- データベースからバルク・コピー・アウトする場合の blk_rowxfer_mult の使用方法は、Client-Library の ct_fetch ルーチンの使用方法に似ています。
- blk_rowxfer_mult で読み込まれるローの数は、アプリケーションの blk_bind 呼び出しで *datafmt->count* として渡される値によって判断されます。blk_rowxfer_mult は、この数のローを読み込み、プログラム変数へデータを書き込みます。

  blk_rowxfer_mult への最後の呼び出し (テーブルの最後のローを取り出す呼び出し) の場合や、データが取り出される間にエラーが発生した場合には、読み込まれるローの数が少ない可能性があります。最後の呼び出しの場合は、リターン・コード CS_END_DATA で示されます。エラーが発生した場合は、CS_ROW_FAIL で示されます。どちらの場合も、blk_rowxfer_mult から返される **row_count* は、実際に読み込まれたロー数に設定されています。
- 大量の text や image のカラム値を含むロー・データをダウンロードするには、配列バインドは使わずに blk_textxfer を blk_rowxfer_mult とともに使用して、一度に大量の値のまとまりを読み込むことができます。詳細については次の項を参照してください。

大量の *text* 値または *image* 値のまとまりでの転送

- 配列バインドが有効でない場合に、アプリケーションでは、blk_textxfer を blk_rowxfer_mult とともに使用すると、大量の text や image の値を含むローを転送できます。この実行方法の詳細については、「Bulk-Library クライアントのプログラミング」(112 ページ) を参照してください。
- 大量の text カラムや image カラムのあるテーブルの場合、アプリケーションでは、すべてのデータを一括転送するよりも、text や image データを固定長のまとまりに分割して転送する方が便利なことがよくあります。もし、カラムをすべて一度に転送するのであれば、アプリケーションは値をすべてそのまま保持するのに十分なバッファ領域を確保する必要があります。

- 大量のカラム値をまとまりで転送するには、次の点に注意します。
  - アプリケーションはカラムに対する blk_bind 呼び出しで *buffer* を NULL として渡します。この設定は、このカラムのデータが blk_textxfer を使用して転送されることを指定します。バルク・コピー・イン・オペレーションでは、blk_bind の **datalen* パラメータでカラム値の長さを指定する必要もあります。
  - アプリケーションはローを転送するために blk_rowxfer_mult を呼び出します。blk_rowxfer_mult は CS_BLK_HAS_TEXT を返します。これは、Bulk-Library が、このローに、blk_textxfer で転送されるデータがさらにあると予想していることを示します。
  - アプリケーションは、転送を必要とする各カラムに対して、blk_textxfer が CS_END_DATA を返して、このカラムのデータがすべて転送されたことを示すまで、blk_textxfer をループで呼び出します。

参照　blk_bind, blk_textxfer

blk_sendrow

説明　*blk_getrow* から取得したフォーマットされたバルク・コピー・ローを送る、サーバ・サイドのルーチンです。

構文

```
CS_RETCODE blk_sendrow(blkdesc, row)

 CS_BLKDESC    *blkdesc;
 CS_BLK_ROW    *row;
```

パラメータ

blkdesc

バルク・コピー・オペレーションの制御ブロックの役目を果たしている CS_BLKDESC を指すポインタです。blk_alloc は CS_BLKDESC 構造体を割り付けます。

row

CS_BLK_ROW 構造体を指すポインタです。CS_BLK_ROW はクライアントから送信されるフォーマットされたバルク・コピー・ローを保持する隠し構造体です。CS_BLK_ROW 構造体にフォーマットされたローを格納するには、ゲートウェイ・アプリケーションはサーバ・サイドのルーチン blk_getrow を呼び出します。

戻り値　blk_sendrow は、次の値を返します。

表 4-9：blk_sendrow の戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。
CS_BLK_HAS_TEXT	ローが 1 つ以上の text、image、sensitivity、または boundary カラムを含んでいる。アプリケーションは、blk_getrow および blk_sendrow を呼び出して次のローを転送する前に、blk_gettext および blk_sendtext を呼び出して、このローのこれらのカラムを転送しなければならない。
CS_PENDING	非同期ネットワーク I/O が有効。『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。

使用法

- blk_sendrow はサーバ・サイドのルーチンです。
- ゲートウェイ・アプリケーションは、blk_sendrow を blk_getrow とともに使用します。これにより、ゲートウェイ・アプリケーションはフォーマットされたバルク・コピー・ローを Open Client アプリケーションから受け取り、これらのローを Adaptive Server Enterprise に送ることができます。
- blk_sendrow はゲートウェイ特有のルーチンで、blk_rowxfer や blk_rowxfer_mult の代わりに使用されます。アプリケーションで blk_sendrow を呼び出せるのは、blk_getrow を呼び出してフォーマットされたローを取得した後だけです。
- ゲートウェイ・アプリケーション内の呼び出し順序は次のとおりです。
  - フォーマットされたバルク・コピー・ローを取得する blk_getrow
  - フォーマットされたローを Adaptive Server Enterprise に送る blk_sendrow

  blk_getrow が CS_BLK_HAS_TEXT を返した場合には、blk_gettext が CS_END_DATA を返すまで、アプリケーションは次のルーチンをループで呼び出す必要があります。
  - text、image、sensitivity、または boundary データのまとまりを取得する blk_gettext
  - text、image、sensitivity、または boundary データのまとまりを送る blk_sendtext

転送される text、image、sensitivity、または boundary データのカラム数に関係なく、blk_gettext/blk_sendtext のループは 1 つだけ必要です。

参照　blk_init, blk_sendtext, blk_colval, blk_getrow, blk_gettext

blk_sendtext

説明　blk_getrow で取得した、フォーマットされたバルク・コピー・ロー内の text、image、sensitivity、または boundary のデータを送る、サーバ・サイドのルーチンです。

構文

```
CS_RETCODE blk_sendtext(blkdesc, row, buffer,
        buflen)

CS_BLKDESC    *blkdesc;
CS_BLK_ROW    *row;
CS_BYTE       *buffer;
CS_INT        buflen;
```

パラメータ

blkdesc

バルク・コピー・オペレーションの制御ブロックの役目を果たしている CS_BLKDESC を指すポインタです。blk_alloc は CS_BLKDESC 構造体を割り付けます。

row

CS_BLK_ROW 構造体を指すポインタです。CS_BLK_ROW 構造体は、クライアントから送信されたフォーマット済みバルク・コピー・ローを保持する隠し構造体です。CS_BLK_ROW 構造体にフォーマットされたローを格納するには、ゲートウェイ・アプリケーションは blk_getrow を呼び出します。

buffer

blk_sendtext が、text、image、sensitivity、または boundary データのまとまりを取得する領域を指すポインタです。

buflen

**buffer* データ領域の長さ (バイト単位) です。

戻り値　blk_sendtext は、次の値を返します。

表 4-10：blk_sendtext の戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。
CS_PENDING	非同期ネットワーク I/O が有効。『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。

使用法

- blk_sendtext はクライアント・サイドのルーチンです。
- ゲートウェイ・アプリケーションは、blk_sendtext を blk_gettext とともに使用します。これにより、ゲートウェイ・アプリケーションはフォーマットされたバルク・コピー・ローの text、image、sensitivity、または boundary データのまとまりを Open Client アプリケーションから受け取り、これらのデータを Adaptive Server Enterprise に送ることができます。
- blk_sendtext は、blk_textxfer の代わりに使用できるゲートウェイ特有のルーチンです。アプリケーションは、blk_gettext を呼び出して、フォーマットされたローの中の text、image、sensitivity、または boundary データのまとまりを取得した後にだけ blk_sendtext を呼び出すことができます。
- ゲートウェイ・アプリケーション内の呼び出し順序は次のとおりです。
  - フォーマットされたバルク・コピー・ローを取得する blk_getrow
  - フォーマットされたローを Adaptive Server Enterprise に送る blk_sendrow

  blk_sendrow が CS_BLK_HAS_TEXT を返した場合には、blk_gettext が CS_END_DATA を返すまで、アプリケーションは次のルーチンをループで呼び出す必要があります。
  - text、image、sensitivity、または boundary データのまとまりを取得する blk_gettext
  - text、image、sensitivity、または boundary データのまとまりを送る blk_sendtext

  転送される text、image、sensitivity、または boundary データのカラム数に関係なく、blk_gettext/blk_sendtext のループは 1 つだけ必要です。

参照

blk_init, blk_sendrow, blk_colval, blk_getrow, blk_gettext

blk_srvinit

説明

必要に応じてサーバ・テーブル・カラムの記述をクライアントにコピーする、サーバ・サイドのルーチンです。

構文

```
CS_RETCODE blk_srvinit(srvproc, blkdescp)

SRV_PROC       *srvproc;
CS_BLKDESC     *blkdescp;
```

パラメータ

srvproc

カラム記述を受け取るクライアントに関連付けられている SRV_PROC 構造体を指すポインタです。この構造体は、Open Server のアプリケーションとクライアント間の通信およびデータを管理するために Server-Library が使用するすべての情報を含んでいます。

blkdescp

バルク・コピー・データの情報が含まれている構造体を指すポインタです。この構造体は、あらかじめ blk_alloc 呼び出しで割り付け、blk_init を呼び出して初期化しておく必要があります。この構造体は、送られてくるフォーマットされたバルク・コピー・ローを正しく解釈するときに使用します。

戻り値

blk_srvinit は、次の値を返します。

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンは失敗した。アクションはなかった。

使用法

- blk_srvinit は、ゲートウェイ・アプリケーションで有用なサーバ・サイドのルーチンです。
- このルーチンは、クライアントの TDS のバージョンが 5.0 以降の場合に、CS_BLKDESC 構造体内の現在のサーバ・テーブル・カラムの記述をクライアントに送ります。
- クライアントからの「バルク挿入要求」に応答するときに、SRV_LANGUAGE イベント・ハンドラの中からこのルーチンを呼び出す必要があります。
- blk_srvinit が正常にクライアントに記述を返すと、Open Server アプリケーションの SRV_BULK イベント・ハンドラは、クライアントからのバルク・データの読み込みを開始できます。イベント・ハンドラは最初に blk_rowalloc を呼び出し、次に blk_getrow と blk_sendrow をループで呼び出して、バルク・コピー・ローを転送します。

- blk_init が CS_BLKDESC 構造体に記述を格納するため、ゲートウェイ・アプリケーションは、blk_srvinit を呼び出す前に blk_init を呼び出さなければなりません。

参照

blk_init, blk_getrow, blk_rowalloc, blk_sendrow

blk_textxfer

説明

バルク・コピー・オペレーション時にカラム・データをまとまりごとに転送します。

構文

```
CS_RETCODE blk_textxfer(blkdesc, buffer, buflen,
outlen)

CS_BLKDESC    *blkdesc;
CS_BYTE       *buffer;
CS_INT        buflen;
CS_INT        *outlen;
```

パラメータ

blkdesc

バルク・コピー・オペレーションの制御ブロックの役目を果たしている CS_BLKDESC を指すポインタです。blk_alloc は CS_BLKDESC 構造体を割り付けます。

buffer

blk_textxfer が、text、image、sensitivity、または boundary データのまとまりを取得する領域を指すポインタです。

buflen

**buffer* データ領域の長さ (バイト単位) です。

outlen

整数変数を指すポインタです。バルク・コピー・イン・オペレーションの場合、*outlen* は使用されません。NULL として渡してください。

バルク・コピー・アウト・オペレーションの場合、**outlen* は、**buffer* にコピーされるデータの長さ (バイト単位) を表します。

戻り値

blk_textxfer は、次の値を返します。

表 4-11：blk_textxfer の戻り値

戻り値	意味
CS_SUCCEED	ルーチンが正常に終了した。
CS_FAIL	ルーチンが失敗した。
CS_END_DATA	データベースからデータをコピーする場合、blk_textxfer は CS_END_DATA を返して、すべてのカラム値が送られたことを示す。 データベースにデータをコピーする場合、blk_textxfer は、blk_bind の **datalen* に等しい量のデータが送られたときに CS_END_DATA を返す。
CS_PENDING	非同期ネットワーク I/O が有効。『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。

例

```
/*
  ** BulkCopyIn()
  **
  ** BLKDATA and DATA_END are defined in the bulk copy
  ** example program.
  */

  CS_STATIC CS_RETCODE
  BulkCopyIn(connection)
  CS_CONNECTION    *connection;
  {
       CS_BLKDESC *blkdesc;
       CS_DATAFMT datafmt;/* variable descriptions */
       Blk_Data *dptr;/* data for transfer */
       CS_INT datalen[5];    /* variable data length */
       CS_INT      len;
       CS_INT numrows;

       /*
       ** Ready to start the bulk copy in now that all the
       ** connections have been made and have a table name.
       ** Start by getting the bulk descriptor initializing.
       */
       ...CODE DELETED.....

       /* Bind columns and transfer rows */
       dptr = BLKDATA;
       while (dptr->pub_id != DATA_END)
       {
```

```
datafmt.datatype = CS_INT_TYPE;
datafmt.count = 1;
datafmt.maxlength = sizeof(CS_INT);
datalen[0] = CS_UNUSED;

if (blk_bind(blkdesc, 1, &datafmt, &dptr->pub_id,
     &datalen[0], NULL) != CS_SUCCEED)
{
     ex_error("BulkCopyIn:blk_bind(1) failed");
     return CS_FAIL;
}
datafmt.datatype = CS_CHAR_TYPE;
datafmt.maxlength = MAX_PUBNAME - 1;
datalen[1] = strlen(dptr->pub_name);
if (blk_bind(blkdesc, 2, &datafmt, dptr->pub_name,
      &datalen[1], NULL) != CS_SUCCEED)
{
     ex_error("BulkCopyIn:blk_bind(2) failed");
     return CS_FAIL;
}
datafmt.maxlength = MAX_PUBCITY - 1;
datalen[2] = strlen(dptr->pub_city);
if (blk_bind(blkdesc, 3, &datafmt, dptr->pub_city,
      &datalen[2], NULL) != CS_SUCCEED)
{
     ex_error("BulkCopyIn:blk_bind(3) failed");
     return CS_FAIL;
}
datafmt.maxlength = MAX_PUBST - 1;
datalen[3] = strlen(dptr->pub_st);
if (blk_bind(blkdesc, 4, &datafmt, dptr->pub_st,
      &datalen[3], NULL) != CS_SUCCEED)
{
     ex_error("BulkCopyIn:blk_bind(4) failed");
     return CS_FAIL;
}
datafmt.datatype = CS_TEXT_TYPE;
datafmt.maxlength = MAX_BIO - 1;
datalen[4] = strlen((char *)dptr->pub_bio);
if (blk_bind(blkdesc, 5, &datafmt, NULL,
      &datalen[4], NULL) != CS_SUCCEED)
{
     ex_error("BulkCopyIn:blk_bind(5) failed");
     return CS_FAIL;
}
if (blk_rowxfer (blkdesc) == CS_FAIL)
{
```

```
                    ex_error("BulkCopyIn:EX_BLK - Failed on ¥
                         blk_rowxfer.");
                    return CS_FAIL;
               }
               if (blk_textxfer(blkdesc, dptr->pub_bio,
                    datalen[4], &len) == CS_FAIL)
               {
                    ex_error("BulkCopyIn:blk_rowxfer() failed");
                    return CS_FAIL;
               }
               dptr++;
          }
          /* ALL the rows sent so clear up */
          ...CODE DELETED.....

          return CS_SUCCEED;
     }
```

使用法

- blk_textxfer はクライアント・サイドのルーチンです。
- blk_textxfer は、大量の text または image の値を転送します。しかし、blk_textxfer ではデータ変換は実行されません。単にデータを転送するだけです。
- アプリケーションには、バルク・コピー・オペレーション時に text および image 値を転送する方法として、次の 2 とおりの方法があります。
  - アプリケーションは、text または image のデータを一般のデータと同様に扱うことができます。つまり、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し、blk_rowxfer_mult を使用してローを転送できます。通常、この方法は少量の text または image の値の場合には便利ですが、大量の値を転送する場合は便利ではありません。blk_rowxfer_mult を使用して値全体を転送する場合、アプリケーションは、カラム値全体を保持できる十分な大きさのプログラム変数を割り付ける必要があります。
  - blk_textxfer を使用すれば、アプリケーションは text または image のデータを複数のまとまりに分割して転送できます。この方法を使用すると、アプリケーションは転送する値よりも小さい転送バッファを使用できます。
- 特定のカラムを、blk_textxfer を使用して転送することを示すには、そのカラムに対して blk_bind を呼び出すときに *buffer* パラメータに NULL を指定します。データベース内へ転送する場合は、blk_bind の **datalen* パラメータとして、値の合計の長さを渡します。
- 参照先「第 3 章 Bulk-Library」。

バルク・コピー・イン・オペレーションでの *blk_textxfer* の使用

- アプリケーションの blk_bind 呼び出しはカラム順である必要はありませんが、blk_textxfer のカラムのデータは、カラム順に転送する必要があります。

  たとえば、アプリケーションがカラム 3 および 4 をバインドし、blk_textxfer で転送するカラムとしてカラム 2 および 1 にマークを付けたとします。アプリケーションは、まず blk_rowxfer_mult を呼び出してカラム 3 および 4 のデータをコピーします。次に、カラム 1 に対して blk_textxfer を呼び出してこのカラムのデータを転送してから、カラム 2 に対して blk_textxfer を呼び出す必要があります。

- データベースにデータをコピーする場合、text、image、boundary、または sensitivity データ型のカラムに、blk_textxfer で転送するためのマークを付けると、後続の同じ型のすべてのカラムにも、blk_textxfer で転送するためのマークを付ける必要があります。

  たとえば、アプリケーションは、ロー内の最初の text カラムに blk_textxfer で転送するためのマークを付けた場合、後続の text カラムをプログラム変数にバインドすることはできません。

- データベースにデータをコピーする場合、アプリケーションは、すべての text または image の値を転送するために必要な回数だけ blk_textxfer を呼び出さなければなりません。

バルク・コピー・アウト・オペレーションでの *blk_textxfer* の使用

- blk_textxfer を使用してデータベースからデータをコピーする場合には、バインドされたカラムに続くカラムだけを blk_textxfer で転送できます。つまり、blk_textxfer で転送されるカラムは、ローの終わりにある必要があります。

  たとえば、アプリケーションは、ロー内の最初の 2 つのカラムをプログラム変数へバインドして、3 番目のカラムに blk_textxfer で転送するためのマークを付けた場合、4 番目のカラムをバインドすることはできません。

- データベースからデータをコピーする場合、すべてのカラム値がコピーされると、blk_textxfer が CS_END_DATA を返します。

参照

blk_bind, blk_rowxfer_mult

付 録 A

エラー・メッセージ

メモリ割り当てに失敗しました

影響を受けた Open Server/SDK コンポーネント	Open Client
エラー番号	2
メッセージ	メモリ割り当てに失敗しました。
考えられる原因	Client-Library アプリケーションは、設定ファイルから読み取った情報を保管するためにメモリを割り当てることができません。
アクション	• マシン上のアプリケーションで現在利用できるメモリの分量をチェックします。 • 物理メモリまたは仮想メモリを増やします。 • 他のアプリケーションを終了してリソースを解放します。 • アプリケーションのメモリ使用量を定期的に監視して、メモリ・リークの可能性がないかチェックします。
追加情報	N/A
バージョン	Client-Library 10.0.x 以降

設定ファイルをオープンできません

影響を受けた Open Server/SDK コンポーネント	Open Client
エラー番号	3
メッセージ	設定ファイル %1! を読み込み用にオープンできません。
考えられる原因	アプリケーションは指定された設定ファイルを開くことができません。

アクション	• 指定された設定ファイルのパスとパーミッションをチェックします。 • 他のプロセスが現在ファイルに書き込んでいないことを確認します。
追加情報	N/A
バージョン	Client-Library 10.0.x 以降

設定が削除されました

影響を受けた Open Server/SDK コンポーネント	N/A
エラー番号	4
メッセージ	設定 %1! が削除されました (CS_FORCE)。
考えられる原因	バグのため、このエラーは現在表示されません。
アクション	N/A
追加情報	N/A
バージョン	N/A

設定ファイル名が長すぎます

影響を受けた Open Server/SDK コンポーネント	N/A
エラー番号	5
メッセージ	設定ファイル名 %1! が長すぎます。
考えられる原因	バグのため、このエラーは現在表示されません。
アクション	N/A
追加情報	N/A
バージョン	N/A

ファイル・フォーマット・エラー

影響を受けた Open Server/SDK コンポーネント　Open Client

エラー番号　6

メッセージ　ファイル・フォーマット・エラー。`%1!` 行目に無効な文字があります。

考えられる原因　指定された設定ファイルの読み取り中に、無効なマルチバイト文字が検出されました。

アクション　特定の設定ファイルに正しいマルチバイト文字が入力されていることを確認します。

追加情報　N/A

バージョン　Client-Library 10.0.x 以降

設定セクションがありません

影響を受けた Open Server/SDK コンポーネント　Open Client

エラー番号　7

メッセージ　設定セクション `%1!` がありません。

考えられる原因　セクション名またはインクルードされたセクション名が、指定された設定ファイルに見つかりません。

アクション　セクション名に対するセクション、またはインクルードされたセクション名を、指定された設定ファイル内に作成します。

追加情報　Open Client では、Client-Library アプリケーション用のランタイム設定ファイルを使用できます。ソフトウェアに含まれているデフォルトのランタイム設定ファイルはありません。したがって、設定ファイルを使用する必要がある場合、独自の設定ファイルを作成しなければなりません。必要に応じて、設定ファイルを作成、配置、およびファイルに名前を付けることができます。ただし、Open Client アプリケーションの起動時に、デフォルトで Open Client は Open Client ディレクトリの *config* サブディレクトリの下にある *ocs.cfg* という名前の設定ファイルを探します。ファイルが存在する場合、Open Client は含まれているアプリケーションの設定ファイルを探します。

ランタイム設定ファイルは、以下の図に示すように、各セクションに 1 つ以上の設定が含まれるセクションに分けられています。

```
[section name]
 keyword = value ; comment
 keyword = value
; more comments
[next section name]
... and so forth...
```

次の構文を使用して、あるセクションに別のセクション内のキーワードをインクルードできます。

```
[section
name]
 include = previous
section name
... more settings...
```

バージョン Client-Library 10.0.x 以降

設定ファイルの構文エラー

影響を受けた Open Server/SDK コンポーネント Open Client

エラー番号 8

メッセージ 設定ファイルの %1! 行目 (%2! の近く) に構文エラーがあります。

考えられる原因 指定された設定ファイルに構文エラーが発生しました。たとえば、セクション名が角カッコで囲まれていません。

アクション 指定された文字列付近の指定された行番号で、設定ファイルのエラーを見つけて修正します。

追加情報

Open Client では、Client-Library アプリケーション用のランタイム設定ファイルを使用できます。ソフトウェアに含まれているデフォルトのランタイム設定ファイルはありません。したがって、設定ファイルを使用する必要がある場合、独自の設定ファイルを作成しなければなりません。必要に応じて、設定ファイルを作成、配置、およびファイルに名前を付けることができます。ただし、Open Client アプリケーションの起動時に、デフォルトで Open Client は Open Client ディレクトリの *config* サブディレクトリの下にある *ocs.cfg* という名前の設定ファイルを探します。ファイルが存在する場合、Open Client は含まれているアプリケーションの設定ファイルを探します。

ランタイム設定ファイルは、以下の図に示すように、各セクションに 1 つ以上の設定が含まれるセクションに分けられています。

```
[section name]
 keyword = value ; comment
 keyword = value
; more comments
[next section name]
... and so forth...
```

バージョン　N/A

設定が使用中に削除されました

影響を受けた Open Server/SDK コンポーネント　N/A

エラー番号　9

メッセージ　設定 `%1!` が使用中に削除されました `(CS_FORCE)`。

考えられる原因　現在のバージョンの Open Cleint で使用されていません

アクション　N/A

追加情報　N/A

バージョン　N/A

索引

A

B

C

D

I

N

O

S

T

あ

い

え

お

か

け

こ

さ

し

せ

そ

た

つ

て

ね

は

ひ

ふ

へ

ま

め

も

ゆ

ろ

わ