新機能ガイド

SAP Open Server™ および SDK for SAP ASE 16.0

Windows、Linux、および UNIX 版

ドキュメント ID： DC00071-01-1570100-01
改訂： 2014 年 3 月

目次

製品のプラットフォームと互換性

SAP® Open Server™ と SDK for SAP® ASE をサポートするプラットフォームを以下に示します。

- HP-UX Itanium 32 ビット版
- HP-UX Itanium 64 ビット版
- IBM AIX 32 ビット版
- IBM AIX 64 ビット版
- Linux x86 32 ビット版
- Linux x86-64 64 ビット版
- Linux on POWER 32 ビット版
- Linux on POWER 64 ビット版
- Microsoft Windows x86 32 ビット版
- Microsoft Windows x86-64 64 ビット版
- Solaris SPARC 32 ビット版
- Solaris SPARC 64 ビット版
- Solaris x86 32 ビット版
- Solaris x86-64 64 ビット版

注意： SAP Open Server と SDK for SAP ASE のすべてのコンポーネントが上記のプラットフォーム上で使用できるとは限りません。 各プラットフォームで使用できるコンポーネントの完全なリストについては、「製品コンポーネント」を参照してください。

SAP Open Server と SDK for SAP ASE のプラットフォーム互換性の一覧

Open Server および SDK for SAP ASE 製品が構築およびテストされているプラットフォーム、コンパイラ、サードパーティ製品を以下の表に示します。

プラットフォーム	オペレーティングシステムレベル	C および C++ コンパイラ	COBOL コンパイラ	Kerberos のバージョン	LDAP (Lightweight Directory Access)	Secure Sockets Layer (SSL)	Perl のバージョン	PHP のバージョン	Python のバージョン
HP-UX Itanium 32 ビット版	HP 11.31	HP ANSI C A.06.17	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	該当なし	該当なし
HP-UX Itanium 64 ビット版	HP 11.31	HP ANSI C A.06.17	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	該当なし	5.3.6	2.6、2.7、および 3.1 (DBAPI 2.0)
IBM AIX 32 ビット版	AIX 6.1	XL C 10.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.1	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	該当なし	該当なし

プラットフォーム	オペレーティングシステムレベル	C および C++ コンパイラ	COBOL コンパイラ	Kerberos のバージョン	LDAP (Lightweight Directory Access)	Secure Sockets Layer (SSL)	Perl のバージョン	PHP のバージョン	Python のバージョン
IBM AIX 64 ビット版	AIX 6.1	XL C 10.1	MF SE 5.1	MIT 1.4.3	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	該当なし	5.3.6	2.6、2.7、および 3.1 (DBAPI 2.0)
Linux x86 32 ビット版	Red Hat Enterprise Linux 5.3	gcc 4.1.2 20060404 kernel 2.6.9-55.ELsmp	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	該当なし	該当なし	該当なし
Linux x86-64 64 ビット版	Red Hat Enterprise Linux 5.3 (Nahant Update 4)	gcc 4.1.2 20060404 kernel 2.6.9-55.ELsmp	MF SE 5.1	MIT 1.4.3	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	5.3.6	2.6、2.7、および 3.1 (DBAPI 2.0)
Linux on POWER 32 ビット版	Red Hat Enterprise Linux 5.3	XL C 10.1	サポート予定なし	MIT 1.4.1	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	該当なし	該当なし

プラットフォーム	オペレーティングシステムレベル	C および C++ コンパイラ	COBOL コンパイラ	Kerberos のバージョン	LDAP (Lightweight Directory Access)	Secure Sockets Layer (SSL)	Perl のバージョン	PHP のバージョン	Python のバージョン
Linux on POWER 64 ビット版	Red Hat Enterprise Linux 5.3	XL C 10.1	MF SE 5.1	MIT 1.4.1	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	該当なし	5.3.6	2.6、2.7、および 3.1 (DBAPI 2.0)
Microsoft Windows x86 32 ビット版	Windows 2008 R2 Service Pack 1	Microsoft Visual Studio 2005 Service Pack 1 (C/C++)	MF SE 5.1	Cybersafe Trustbroker 4.0、MIT 2.6.4	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	該当なし	該当なし	該当なし
Microsoft Windows x86-64 64 ビット版	Windows 2008 R2 Service Pack 1	Microsoft Visual Studio 2005 Service Pack 1 (C/C++)	MF SE 5.1	Cybersafe Trustbroker 2.1	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	Active Perl 5.14.1 (DBI 1.616)	5.3.6	2.6、2.7、および 3.1 (DBAPI 2.0)
Solaris SPARC 32 ビット版	Solaris 10	Solaris Studio 12.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.2	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certicom SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	該当なし	該当なし

プラットフォーム	オペレーティングシステムレベル	C および C++ コンパイラ	COBOL コンパイラ	Kerberos のバージョン	LDAP (Light-weight Di-rectory Access)	Secure Sock-ets Layer (SSL)	Perl のバージョン	PHP のバージョン	Python のバージョン
Solaris SPARC 64 ビット版	Solaris 10、パッチレベル 144488-17 以降、パッチレベル 119963-24 以降 (SUNWlibC 向け)	Solaris Studio 12.1	MF SE 5.1	Cybersafe Trustbroker 2.1、MIT 1.4.2	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certi-com SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	該当なし	5.3.6	2.6、2.7、および 3.1 (DBAPI 2.0)
Solaris x86 32 ビット版	Solaris 10	Solaris Studio 12.1	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certi-com SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	5.14 (DBI 1.616)	該当なし	該当なし
Solaris x86-64 64 ビット版	Solaris 10	Solaris Studio 12.1	MF SE 5.1	MIT 1.4.2	OpenLDAP 2.4.31 (OpenSSL 1.0.1b を含む)	Certi-com SSL Plus 5.2.2 (SBGSE 2.0) CSI-Crypto 2.7M1	該当なし	5.3.6	2.6、2.7、および 3.1 (DBAPI 2.0)

凡例: 該当なし = そのプラットフォーム版でスクリプトが使用できない、または SDK for SAP ASE と連動しない。

注意： SAP Open Server および SDK for SAP ASE の動作確認に関する最新のサポートについては、SAP® Sybase® Platform Certification ページを参照してください。 http://certification.sybase.com/ucr/search.do

Microsoft は、Visual Studio 2005 に対するメインストリームサポートを終了しています。 ただし、現時点で SDK for SAP ASE は Visual Studio Compiler 2005 以降のバージョンをサポートしています。なるべく早く Visual Studio 2010 に移行することをおすすめします。

Solaris SPARC 64 ビット版のパッチレベル

Solaris SPARC 64 ビット版プラットフォームの場合、Solaris 10 オペレーティングシステムのカーネルパッチレベルは 144488-17 以降でなければなりません (2011 年 6 月 30 日付け以降のパッチバンドル)。

また、パッチ 119963-24 以降を `SUNWlibC` パッケージに適用する必要があります。

FIPS 互換プラットフォームのサポート

SAP Open Server と、SAP® jConnect™ for JDBC (SAP jConnect) を除くすべての SDK for SAP ASE コンポーネントは FIPS に準拠しており、次のプラットフォームで使用できます。

プラットフォーム	FIPS 準拠
HP-UX Itanium 32 ビット版	15.7 SP121
HP-UX Itanium 64 ビット版	15.7 SP120
IBM AIX 32 ビット版	なし
IBM AIX 64 ビット版	なし
Linux x86 32 ビット版	15.7 SP121
Linux x86-64 64 ビット版	15.7 SP120
Linux on POWER 32 ビット版	15.7 SP120
Linux on POWER 64 ビット版	なし
Microsoft Windows x86 32 ビット版	15.7 SP120
Microsoft Windows x86-64 64 ビット版	15.7 SP121
Solaris SPARC 32 ビット版	15.7 SP120
Solaris SPARC 64 ビット版	15.7 SP121
Solaris x86 32 ビット版	15.7 SP121
Solaris x86-64 64 ビット版	15.7 SP121

SAP jConnect for JDBC の FIPS 準拠

バージョン 16.0 から、SAP jConnect に、FIPS 140-2 認定 Java Cryptography Extension (JCE) プロバイダが同梱されます。 この JCE プロバイダはデフォルトで使用され、FIPS 140-2 準拠の暗号化を提供します。

製品コンポーネント

ここでは、SAP Open Server と SDK for SAP ASE の製品コンポーネントについて説明します。

SAP Open Server 16.0 と SDK for SAP ASE 16.0 は、SAP® Adaptive Server® Enterprise で使用される Perl、PHP、および Python のスクリプト言語もサポートします。

SAP Open Server

SAP® Open Server は、SAP Open Client™ または SAP jConnect ルーチン経由で送信されたクライアント要求に応答するカスタムサーバの作成に使用可能な API とサポートツールのセットです。

表 1 : SAP Open Server のコンポーネントとサポートされているプラットフォーム

Open Server のコンポーネント	プラットフォーム
SAP Open Server Server-Library	すべてのプラットフォーム
SAP Open Server Client-Library	すべてのプラットフォーム
言語モジュール	すべてのプラットフォーム

SDK for SAP ASE

SDK for SAP ASE は、クライアントアプリケーションの開発に使用可能なライブラリとユーティリティのセットです。

表 2 : SDK for SAP ASE のコンポーネントとサポートされているプラットフォーム

SDK for SAP ASE のコンポーネント	プラットフォーム
SAP Open Client Client-Library	プラットフォーム
SAP® Open Client DB-Library™	すべてのプラットフォーム
SAP® Embedded SQL™/C (ESQL/C)	すべてのプラットフォーム

SDK for SAP ASE のコンポーネント	プラットフォーム
Embedded SQL/COBOL (ESQL/COBOL)	• HP HP-UX Itanium 32 ビット版 • HP HP-UX Itanium 64 ビット版 • IBM AIX 64 ビット版 • Linux x86 32 ビット版 • Linux x86-64 64 ビット版 • Linux on POWER 32 ビット版 • Linux on POWER 64 ビット版 • Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版 • Solaris SPARC 32 ビット版 • Solaris SPARC 64 ビット版 • Solaris x86 32 ビット版 • Solaris x86-64 64 ビット版
Extended Architecture (XA)	• HP HP-UX Itanium 32 ビット版 • HP HP-UX Itanium 64 ビット版 • IBM AIX 32 ビット版 • IBM AIX 64 ビット版 • Linux x86-64 64 ビット版 • Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版 • Solaris SPARC 32 ビット版 • Solaris SPARC 64 ビット版 • Solaris x86 32 ビット版 • Solaris x86-64 64 ビット版
SAP jConnect	すべてのプラットフォーム
SAP Adaptive Server Enterprise ODBC ドライバ	• HP HP-UX Itanium 64 ビット版 • IBM AIX 64 ビット版 • Linux on POWER 64 ビット版 • Linux x86 32 ビット版 • Linux x86-64 64 ビット版 • Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版 • Solaris SPARC 64 ビット版 • Solaris x86-64 64 ビット版

SDK for SAP ASE のコンポーネント	プラットフォーム
SAP Adaptive Server Enterprise ADO.NET Data Provider	• Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版
言語モジュール	すべてのプラットフォーム
Python 用 SAP Adaptive Server Enterprise 拡張モジュール	• HP-UX Itanium 64 ビット版 • IBM AIX 64 ビット版 • Linux x86-64 64 ビット版 • Linux on POWER 64 ビット版 • Microsoft Windows x86-64 64 ビット版 • Solaris SPARC 64 ビット版 • Solaris x86-64 64 ビット版
PHP 用 SAP Adaptive Server Enterprise 拡張モジュール	• HP-UX Itanium 64 ビット版 • IBM AIX 64 ビット版 • Linux x86-64 64 ビット版 • Linux on POWER 64 ビット版 • Microsoft Windows x86-64 64 ビット版 • Solaris SPARC 64 ビット版 • Solaris x86-64 64 ビット版
Perl 用 SAP Adaptive Server Enterprise データベースドライバ	• HP-UX Itanium 32 ビット版 • IBM AIX 32 ビット版 • Linux x86-64 64 ビット版 • Linux on POWER 32 ビット版 • Microsoft Windows x86-64 64 ビット版 • Solaris SPARC 32 ビット版 • Solaris x86 32 ビット版
dbisql ツール	• HP HP-UX Itanium 64 ビット版 • IBM AIX 64 ビット版 • Linux on POWER 64 ビット版 • Linux x86-64 64 ビット版 • Solaris SPARC 64 ビット版 • Solaris x86-64 64 ビット版 • Microsoft Windows x86 32 ビット版 • Microsoft Windows x86-64 64 ビット版

SDK for SAP ASE のコンポーネント	プラットフォーム
Kerberos Authentication for DB-Library	• Linux x86 32 ビット版 • Microsoft Windows x86 32 ビット版 • Solaris SPARC 32 ビット版 • Solaris SPARC 64 ビット版

16.0 の新機能

バージョン 16.0 では、SAP Open Client 16.0 および SAP Open Server 16.0、SDK for SAP ASE 16.0、および PHP 用 SAP Adaptive Server Enterprise 拡張モジュール 16.0 を対象として、機能の更新、サポートの打ち切り、ブランド変更が行われています。

SAP Open Client 16.0 と SAP Open Server 16.0 の機能

SAP Open Client と SAP Open Server では、`ocs.cfg` ファイルに新しいプロパティが導入され、ユーザはこれを利用して設定を適用できるようになりました。

名前付きアプリケーションの NAMED_APP_DEFAULT セクション

`NAMED_APP_DEFAULT` セクションを使用すると、ユーザは一連の設定値を `ocs.cfg` ファイルに含まれていないすべての名前付きアプリケーションに適用できます。

`ocs.cfg` ランタイム設定ファイルに指定のない名前付きアプリケーションのプロパティは、`NAMED_APP_DEFAULT` セクションで設定できます。 たとえば、`ocs.cfg` 内の固有のセクションがない名前付きアプリケーションのすべてで FIPS モードをオンにするには、次のように指定できます。

```
[NAMED_APP_DEFAULT]
CS_PROP_FIPSMODE = CS_TRUE ;
```

SDK DB-Library Kerberos Authentication Option の変更

バージョン 16.0 以降、DB-Library Kerberos Authentication Option が SDK for SAP ASE にコンポーネントとして追加され、別途購入可能なオプションではなくなります。

SDK for SAP ASE コンポーネントの詳細については、「製品コンポーネント」>「SDK for SAP ASE」を参照してください。

SDK DB-Library Kerberos Authentication Component のインストールと有効化の詳細は、使用しているプラットフォームの『SDK for SAP ASE /SAP Open Server 16.0 インストールガイド』を参照してください。

既知の問題については、使用しているプラットフォームの SDK for SAP ASE と SAP Open Server のバージョン 16.0 リリースノートを参照してください。

Adaptive Server Enterprise のドライバおよびプロバイダ用の SDK for SAP ASE 16.0 の機能

バージョン 16.0 では、SAP jConnect と SAP Adaptive Server ADO.NET Data Provider を対象とする機能が更新されています。

FIPS 140-2 に準拠した SAP jConnect for JDBC

バージョン 16.0 以降、SAP jConnect に、FIPS 140-2 認定 Java Cryptography Extension (JCE) プロバイダが付属しています。 この JCE プロバイダは、デフォルトで、パスワードの暗号化と SSL 接続に使用されます。

`JCE_PROVIDER_CLASS` プロパティを設定すると、そのプロパティで指定された JCE プロバイダが使用されます。 FIPS 140-2 準拠が必要な場合は、指定した JCE プロバイダが FIPS 140-2 認定であることを確認してください。

FIPS140-2 準拠の暗号化の使用を強制するには、`ENABLE_FIPS` 接続プロパティを true に設定します。

SAP ASE ADO.NET バージョン 16.0 用の .NET Framework コンポーネント

SAP ASE ADO.NET バージョン 16.0 では .NET Framework のバージョン 3.5 と 4.0 がサポートされます。

表 3 : SDK for SAP ASE に付属する .NET Framework 3.5 および 4.0 コンポーネント

.NET Framework のバージョン	コンポーネント
3.5	• `Sybase.AdoNet35.AseClient.dll` (ADO.NET プロバイダ) • `Sybase.VSIntegration35.ASE.dll` (DDEX プロバイダ) • `Sybase.AdoNet35.AseDestination.dll` (SSIS コンポーネント) • `Sybase.AdoNet35.AseReportingServices.dll` (SSRS コンポーネント) • `Sybase.AdoNet35.EnterpriseLibrary.dll` (Enterprise Library コンポーネント) • `AseGacUtility35.exe` • `AseRegistar35.exe`
4.0	• `Sybase.AdoNet4.AseClient.dll` (ADO.NET プロバイダ) • `Sybase.VSIntegration4.ASE.dll` (DDEX プロバイダ) • `Sybase.AdoNet4.AseDestination.dll` (SSIS コンポーネント) • `Sybase.AdoNet4.AseReportingServices.dll` (SSRS コンポーネント) • `Sybase.AdoNet4.EnterpriseLibrary.dll` (Enterprise Library コンポーネント) • `AseGacUtility4.exe` • `AdoNetRegistrar4.exe`

参照：

• 廃止された機能 (17 ページ)

PHP 用 SAP Adaptive Server Enterprise 拡張モジュール

PHP 用 SAP ASE 拡張モジュールが拡張され、SAP ASE 結果セットのデータ型がサポートされるようになりました。

PHP ドライバの拡張データ型のサポート

バージョン 16.0 以降、PHP 用 SAP ASE ドライバでは、言語クエリ結果セット内のすべての SAP ASE データ型がサポートされます。

また、バージョン 16.0 以降の PHP 用 SAP ASE ドライバでは、リモートプロシージャコールパラメータの拡張データ型もサポートされます。`sybase_rpc_bind_param_ex()` API が更新され、このようなデータ型の受け入れが可能になりました。

PHP 用 SAP ASE ドライバは、PHP ドライバ拡張で使用可能になった PHP datetime ISO 8601 解析関数を使用します。 プラットフォームによっては、PHP ビルドプロセスに date/time 関数のエクスポートを明示的に指示することが必要になります。たとえば、これらの設定はプラットフォームに応じて次のように使用できます。

- Windows - *PHP_DLL_DEF_SOURCES* 変数の参照を `ext¥date¥php_date.def` ファイルに追加します。 Makefile 変数は、PHP date core 拡張関数の EXPORTS を含む `ext¥date¥php_date.def` への参照です。
- Linux - gcc コンパイラを使用して PHP 環境を構築する場合は、configure スクリプトの CFLAGS で '--fvisibility=hidden' が設定されていないことを確認します。
- AIX - xlc コンパイラを使用している場合は、PHP 環境をコンパイルする前に CFLAGS に '-bexpall' が含まれていることを確認します。

サポートされているデータ型の詳細については、『PHP 用 Adaptive Server Enterprise 拡張モジュールプログラマーズガイド』を参照してください。

Microsoft Windows x64 上で VC9/VS2008 を使用してコンパイルされる PHP ドライバ

バージョン 16.0 以降、PHP 用 SAP ASE ドライバは、Microsoft Windows x64 プラットフォーム上で Microsoft Visual C++ 9/Visual Studio 2008 を使用してコンパイルされます。

そのため、PHP ドライバは、VC9/VS2008 を使用してコンパイルされた 64 ビット PHP バージョン 5.3.6 ランタイムでのみロードが可能になります。

注意： Microsoft Visual C++ 9 コンパイラは Visual Studio 2008 に付属しています。

SDK for ASE SDK 16.0 における dbisql ツールのサポート

Interactive SQL が SDK for SAP ASE バージョン 16.0 に追加されました。**dbisql** で起動します。

『Adaptive Server Enterprise 15.7 SP100 ユーティリティガイド』の「第 7 章: グラフィックモードでの Interactive SQL の使用」を参照してください。

廃止された機能

バージョン 16.0 以降、SAP Open Server および SDK for SAP ASE ではいくつかのドライバのサポートが廃止されています。

バージョン 16.0 以降、SDK for SAP ASE と SAP Open Server でサポート対象外になったものは次のとおりです。

- .NET 2.0 および 3.0 Framework
- OLE DB プロバイダ

SDK for SAP ASE バージョン 16.0 で削除されたディレクトリとコンポーネントは次のとおりです。

- `DataAccess¥ADONET¥dll¥Sybase.AdoNet2.AseClient.dll`
- `DataAccess¥ADONET¥dll ¥Sybase.AdoNet2.AseReportingServices.dll`
- `DataAccess¥ADONET¥dll ¥Sybase.EnterpriseLibrary.AseClient.dll`
- `DataAccess¥ADONET¥dll¥Sybase.VSIntegration.ASE.dll`
- `DataAccess¥ADONET¥dll¥AdoNetRegistrar.exe`
- `DataAccess¥ADONET¥dll¥AseGacUtility.exe`
- `DataAccess¥ADONET¥dll¥Microsoft.VC80.ATL`
- `DataAccess¥ADONET¥dll¥Microsoft.VC80.CRT`
- `DataAccess¥ADONET¥dll¥Microsoft.VC80.MFC`
- `DataAccess¥ADONET¥dll¥Microsoft.VC80.MFCLOC`
- `DataAccess¥ADONET¥dll¥Microsoft.VC80.OpenMP`
- `DataAccess64¥ADONET¥dll¥Sybase.AdoNet2.AseClient.dll`
- `DataAccess64¥ADONET¥dll¥Sybase.AdoNet2.AseDestination.dll`

- DataAccess64¥ADONET¥dll ¥Sybase.AdoNet2.AseReportingServices.dll
- DataAccess64¥ADONET¥dll ¥Sybase.EnterpriseLibrary.AseClient.dll
- DataAccess64¥ADONET¥dll¥Sybase.VSIntegration.ASE.dll
- DataAccess64¥ADONET¥dll¥AdoNetRegistrar.exe
- DataAccess64¥ADONET¥dll¥AseGacUtility.exe
- DataAccess64¥ADONET¥dll¥Microsoft.VC80.ATL
- DataAccess64¥ADONET¥dll¥Microsoft.VC80.CRT
- DataAccess64¥ADONET¥dll¥Microsoft.VC80.MFC
- DataAccess64¥ADONET¥dll¥Microsoft.VC80.MFCLOC
- DataAccess64¥ADONET¥dll¥Microsoft.VC80.OpenMP

- DataAccess¥OLEDB¥dll
- DataAccess¥OLEDB¥dll¥locales
- DataAccess¥OLEDB¥dll¥Microsoft.VC80.ATL
- DataAccess¥OLEDB¥dll¥Microsoft.VC80.CRT
- DataAccess¥OLEDB¥dll¥Microsoft.VC80.MFC
- DataAccess¥OLEDB¥dll¥Microsoft.VC80.MFCLOC
- DataAccess¥OLEDB¥dll¥Microsoft.VC80.OpenMP
- DataAccess¥OLEDB¥dll¥sybdrvoledb.dll
- DataAccess¥OLEDB¥samples
- DataAccess¥OLEDB¥sp
- DataAccess¥bin¥sybdrvadm.exe
- DataAccess¥bin¥sybdrvadm.ico
- DataAccess¥bin¥dsnmigrate.exe

- DataAccess64¥OLEDB
- DataAccess64¥OLEDB¥dll¥locales
- DataAccess64¥OLEDB¥dll¥Microsoft.VC80.ATL
- DataAccess64¥OLEDB¥dll¥Microsoft.VC80.CRT
- DataAccess64¥OLEDB¥dll¥Microsoft.VC80.MFC
- DataAccess64¥OLEDB¥dll¥Microsoft.VC80.MFCLOC
- DataAccess64¥OLEDB¥dll¥Microsoft.VC80.OpenMP
- DataAccess64¥OLEDB¥dll¥sybdrvoledb64.dll

- DataAccess64¥OLEDB¥samples
- DataAccess64¥OLEDB¥sp
- DataAccess64¥bin¥sybdrvadm.exe
- DataAccess64¥bin¥sybdrvadm.ico
- DataAccess64¥bin¥dsnmigrate.exe

SP121 の新機能

SP121 では、新たに次のプラットフォームが OpenSSL 用の FIPS に準拠しています。

- HP-UX Itanium 32 ビット版
- Linux x86 32 ビット版
- Microsoft Windows x86-64 64 ビット版
- Solaris SPARC 64 ビット版
- Solaris x86 32 ビット版
- Solaris x86-64 64 ビット版

参照：

- FIPS 互換プラットフォームのサポート (6 ページ)

SP120 の新機能

SP120 では Certicom 暗号化サービスが置き換えられています。また、Open Client 15.7、Open Server 15.7、および PHP 用 Adaptive Server Enterprise 拡張モジュール 15.7 を対象とする機能とプロパティに新規導入と更新が適用されています。

Certicom の代替製品

機密情報の格納と転送を保護するための暗号化サービスを提供する Certicom ソフトウェアは、SAP® Sybase 製品ではサポートされなくなります。 これらのサービスは、他のプロバイダによって置き換えられます。

Open Server と、jConnect for JDBC を除くすべての SDK コンポーネントでは OpenSSL がサポートされます。

jConnect for JDBC は、Java VM で提供される、要求されたアルゴリズムをサポートできる JCE プロバイダを使用します。 たとえば、Oracle Java VM では通常、SUN JCE プロバイダが使用されます。 実際に使用されるプロバイダは、Java VM のセキュリティ設定によって異なります。

ODBC、OLE DB、ADO.NET、Open Client、および Open Server における OpenSSL

Open Server と、jConnect for JDBC を除くすべての SDK コンポーネントでは OpenSSL がサポートされます。

この変更により、次の証明書ユーティリティがサポートされなくなります。

- **certreq**
- **certauth**
- **certpk12**

かわりに、Open Server と SDK に **openssl** ユーティリティが含まれます。ユーティリティの場所は次のとおりです。

- (UNIX) `$SYBASE/$SYBASE_OCS/bin`
- (Windows) `%SYBASE%¥%SYBASE_OCS%¥bin`

certreq、**certauth**、**certpk12** で実装されたすべての証明書管理タスクを実行するには **openssl** ツールを使用します。

http://www.openssl.org/docs/apps/openssl.html を参照してください。

FIPS 準拠の有効化

15.7 SP120 から、クライアントライブラリはデフォルトで厳格な FIPS 準拠を有効にしなくなったため、アプリケーションで準拠を有効にする必要があります。

FIPS モードの OpenSSL は、OpenSSL のセキュリティによって厳格に制御されます。 このため、クライアントライブラリで FIPS 準拠を有効にする前に、サーバの SSL 証明書が FIPS の要件に準拠していることを確認してください。 要件に準拠していない場合、FIPS モードを有効にすると、サーバへの接続は失敗します。

これは、Certicom FIPS モジュールで正しく動作していた証明書の一部が、OpenSSL の使用時に動作しなくなる可能性があることも意味しています。

サーバの SSL 証明書に関する FIPS 140-2 の要件は次のとおりです。

- MD5 アルゴリズムは FIPS 140-2 に準拠していないため、MD5 を FIPS 準拠のアルゴリズムに置き換える必要があります。
- プライベートキーは pkcs8 のフォーマットに準拠し、FIPS 140-2 準拠の OpenSSL アルゴリズムを使用して暗号化されている必要があります。
- デジタル署名に RSA 暗号化アルゴリズムを使用する場合、RSA キーのサイズは 1024 ビット以上である必要があります。

詳細については、『Adaptive Server Enterprise SP60 新機能ガイド』を参照してください。

ODBC、OLE DB、および ADO.NET ドライバにおける FIPS 準拠の有効化

OpenSSL で FIPS モードを有効または無効にするための新しい接続プロパティ `EnableFIPS` が ODBC、OLE DB、および ADO.NET ドライバに追加されました。

- ODBC で FIPS モードを有効にするには、`EnableFIPS=1` を接続文字列に追加します。
- OLE DB で FIPS モードを有効にするには、`EnableFIPS=true` をプロバイダ文字列に追加します。
- ADO.NET で FIPS モードを有効にするには、`EnableFIPS=true` を接続文字列に追加します。

デフォルトでは、`EnableFIPS` プロパティは無効になっています (false または 0 に設定されている)。 各クライアントプロセスで開くことができる接続のタイプは 1 つのみです。 1 つの接続が FIPS である必要がある場合、すべての接続で FIPS を使用する必要があります。

注意： Sybase 独自のパスワード暗号化によって使用されるアルゴリズムは、FIPS に準拠していません。このため、FIPS モードを有効にする際は、サーバで RSA パスワード暗号化モードがサポートされていることを確認してください。

次の場合、「セキュリティコンテキストの設定中にエラーが発生しました」と表示される可能性があります。

FIPS を使用しない場合	SSL 接続に使用する `trusted.txt` ファイルの形式が正しくない。`trusted.txt` のパスとファイルを確認する。
FIPS モードが有効になっている場合	• Microsoft Windows で、ドライバが推奨ベースアドレスを取得しなかったことにより、OpenSSL のコア内フィンガープリントチェックが失敗した。 Microsoft Process Explorer を使用すると、実行中のプロセスのベースアドレスを表示できる。 • OpenSSL のフィンガープリントが、認識できない理由により失敗した。

Microsoft Windows では、ドライバをメモリにロードする際の推奨ベースアドレスがあります。 推奨ベースアドレスは次のとおりです。

- ODBC: `0xF800000`
- OLE DB: `0xF500000`
- ADO.NET: `0xF200000`

Open Client と Open Server における FIPS 準拠の有効化

Open Client と Open Server では、FIPS 140-2 準拠を有効にすることができます。

- アプリケーションでは、`ocs.cfg` ファイルのコンテキストプロパティ CS_PROP_FIPSMODE を CS_TRUE に設定する必要があります。または
- 環境変数 SYBOCS_FIPS_MODE を 1 に設定します。

Microsoft Windows で FIPS 準拠を有効にする場合、メモリにロードする Open Client の推奨ベースアドレスは `0xFB00000` です。 これは、OpenSSL のコア内フィンガープリントチェックでベースアドレスの競合が発生しないようにすることを目的としています。

注意： この推奨ベースアドレスが使用できない場合、Open Client は「FIPS fingerprint check failed」というエラーで初期化に失敗します。

Perl、Python、および PHP における FIPS 準拠の有効化

Perl、Python、および PHP では、FIPS 140-2 準拠を有効にすることができます。
Perl、Python、および PHP で FIPS 140-2 準拠を有効にするには、`FIPSMode` 接続プロパティを true に設定します。

Microsoft Windows で FIPS 準拠を有効にする場合、メモリにロードする際の推奨ベースアドレスは `0xFB00000` です。 これは、OpenSSL のコア内フィンガープリントチェックでベースアドレスの競合が発生しないようにすることを目的としています。

注意： この推奨ベースアドレスが使用できない場合、「FIPS fingerprint check failed」というエラーで初期化に失敗します。

FIPS プラットフォームの可用性

Certicom の置き換えにより、SP120 に付属する FIPS 140-2 準拠の OpenSSL 用暗号化モジュールが次のプラットフォームで使用できるようになります。

- HP-UX Itanium 64 ビット版
- Linux x86-64 64 ビット版
- Linux on POWER 32 ビット版
- Microsoft Windows x86 32 ビット版
- Solaris SPARC 32 ビット版

参照：

• FIPS 互換プラットフォームのサポート (6 ページ)

jConnect for JDBC によって使用される JCE プロバイダ

jConnect は、要求されたアルゴリズムをサポートできる、Java VM 内の JCE プロバイダを使用します。

たとえば、Oracle Java VM では通常、SUN JCE プロバイダが使用されます。 実際に使用されるプロバイダは、Java VM のセキュリティ設定によって異なります。

特定の JCE プロバイダを使用するための jConnect の設定

特定の JCE プロバイダを使用するよう jConnect を設定します。
JCE_PROVIDER_CLASS 接続プロパティを、プロバイダ用の文字列クラス名に設定します。

注意： JCE プロバイダの jar ファイルが CLASSPATH に配置されていることを確認してください。

jConnect for JDBC における FIPS 準拠の有効化

通常、Java VM で提供されるデフォルトの JCE プロバイダは FIPS 140-2 認定を受けていません。 FIPS 140-2 準拠の接続を設定するには、FIPS 140-2 認定 JCE プロバイダにアクセスする必要があります。

FIPS 140-2 認定 JCE プロバイダにアクセスできれば、次のプロパティを設定することにより、jConnect を FIPS 140-2 に準拠するよう設定できます。

1. `ENABLE_FIPS` 接続プロパティのブール値を TRUE に設定します。
2. `JCE_PROVIDER_CLASS` 接続プロパティを、FIPS 140-2 認定 JCE プロバイダ用の文字列クラス名に設定します。

Open Client 15.7 と Open Server 15.7 の機能

Open Client 15.7 と Open Server 15.7 では、Client-Library の機能の更新および新しい接続プロパティがサポートされます。

パフォーマンスを向上させる新しい isql 引数

新しい `--fast` コマンドライン引数を使用すると、数値のみのカラムタイプで (大型) データセットの取得速度が向上します。

`--fast` 引数には、Sybase Adaptive Server® Enterprise および Open Client でサポートされているすべての整数型が含まれます。

注意： `--fast` 引数は、string、date、time、および datetime のデータ型には作用しません。

`--fast` オプションは、カラム出力のフォーマットが変更されるため、標準出力との互換性はありません。 標準の **isql** 出力フォーマットは保持されます。 また、`--fast` が使用されている場合は、行の折り返しが行われません。そのため、デフォルトの 80 カラム幅を保持するには、`--fast` とともに `-w` を指定する必要があります。

次の例では、カラム 80 で選択された出力が折り返されます。

```
isql -U sa -P --fast -w80
```

次の例では、折り返されません。

```
isql -U sa -P --fast
```

isql と bcp 用の --filemode オプション

(UNIX のみ) `--filemode` オプションを使用すれば、**isql** によって生成されたファイル (出力ファイル) と **bcp** によって生成されたファイル (**bcp** によって生成されたすべてのファイル) のファイルパーミッションを設定できます。 `--filemode` は、これらのファイルのデフォルトパーミッション設定より優先されます。

`--filemode <nnn>` オプションの使用により、ユーザは、一部の **bcp** 生成ファイルと **isql** 生成ファイルでデフォルトより緩和されたパーミッション設定を指定できます。

- **isql** では、`--filemode <nnn>` オプションを使用して緩和されたファイルパーミッションを設定すると、生成される出力ファイル (-o オプション) と **isql** からリダイレクトされる出力 ('go > filename' メソッドを使用) に影響します。 たとえば、「user」に読み込み/書き込みを許可し、「group」に読み込みのみを許可し、「other」に読み込み/書き込みを許可しない設定で **isql** 出力を作成する場合は、次のように指定できます。
  ```
  isql -U sa -P secret -o myoutput --filemode 640
  ```
- **bcp** では、`--filemode <nnn>` オプションを使用して緩和されたファイルパーミッションを設定すると、生成される次の **bcp** ファイルに影響します。
  - データファイル (**bcp** 出力データが収録されたファイル)
  - 出力ファイル (-o オプションを使用)
  - フォーマットファイル (-f オプションを使用)
  - エラーファイル (-e オプションを使用)

生成されるすべてのデータと出力ファイルのデフォルトパーミッション設定については、ESD #4 セクションの「Open Client ファイルと Open Server ファイルに対する厳密化されたパーミッション (UNIX のみ)」を参照してください。

参照：

• Open Client ファイルと Open Server ファイルに対する厳密化されたパーミッション (UNIX のみ) (109 ページ)

接続文字列プロパティの新しいキーワード

Client-Library では、API ルーチンの **ct_connect_string()** に新しいキーワードが追加されています。

ct_connect_string()

名前	説明	値
DataOrigin	データオリジンスタンピングサービスを有効にする。 データオリジンスタンピングサービスが、データがクライアントまたはサーバから送信されたことを確認する。	ブール値。 デフォルトは false。
FIPSMode	**FIPSMode** は、SSL 暗号化での FIPS 準拠アルゴリズムの使用を判別する。 コンテキストプロパティの **CS_PROP_FIPSMODE** も参照。	ブール値。 デフォルトは false。
HAFailover	高可用性フェールオーバを判別する。	ブール値。 デフォルトは false。
PasswordEncryptionOnRetry	キーワード **PasswordEncryption** は、特定の接続で **CS_SEC_ENCRYPTED_PASSWORD** と **CS_SEC_EXTENDED_ENCRYPTED_PASSWORD** の両方を使用可能にする。 **PasswordEncryptionOnRetry** は、古いサーバが新しいバージョンをサポートしていない場合に、そのサーバに接続するクライアントアプリケーションが、古い形式のパスワード暗号化を引き続き使用できることを保証する。	ブール値。 デフォルトは false。

例:

```
CS_SEC_DATAORIGIN    Boolean;
CS_PROP_FIPSMODE    Boolean;
CS_HAFAILOVER    Boolean;
CS_SEC_NON_ENCRYPTION_RETRY    Boolean;
```

PHP 用 Adaptive Server Enterprise 拡張モジュール

PHP 用 Adaptive Server Enterprise 拡張モジュールが拡張され、非デバッグ時間帯の PHP ドライバのロードがサポートされるようになりました。

非デバッグ PHP ランタイムにおける PHP デバッグドライバのロード

Sybase SDK のバージョン 15.7 SP 120 では、Linux と Windows 以外のすべてのサポート対象プラットフォームで、非デバッグ PHP ランタイムに PHP ドライバの通常バリアントとデバッグバリアントをロードできます。

デバッグ PHP ドライバは、デバッグ PHP ランタイムにロードされなくなりました。

Linux と Windows では、通常 PHP ドライバを通常 PHP ランタイムにロードし、デバッグ PHP ドライバをデバッグ PHP ランタイムにロードすることができます。

SP110 の新機能

SP110 では、Open Client 15.7、Open Server 15.7、SDK 15.7、Perl 用 Adaptive Server Enterprise Data Provider 15.7、および Python 用 Adaptive Server Enterprise 拡張モジュール 15.7 を対象とする機能とプロパティに新規導入および更新が適用されています。

Open Client 15.7 と Open Server 15.7 の機能

Open Client 15.7 と Open Server 15.7 では、Client-Library の機能の更新および新しい接続プロパティがサポートされます。

Open Server の新機能 srv_msgq_set_blocking_threshold

新しい API **srv_msgq_set_blocking_threshold()** の機能を使用すれば、送信元のスレッドをブロックすることなく、メッセージキュー内に保存可能なメッセージ数のスレッショルドを設定できます。

構文

```
CS_RETCODE srv_msgq_set_blocking_threshold(SRV_OBJID mqid, CS_INT
threshold)
```

パラメータ

- *mqid*
  ブロックスレッショルドを設定するメッセージキューの識別子。
- *threshold*
  送信元のスレッドをブロックすることなく、メッセージキューに入れることが可能なメッセージの最大数、もしくは、スレッショルドなしの `CS_NO_LIMIT` を設定する。

戻り値

戻り値	意味
CS_SUCCEED	スレッショルドが正しく設定されている。
CS_FAIL	スレッショルドが正しく設定されていない。

使用例

```
/*
** We want the threads to block if there are already 10 messages
```

```
** in the queue.
*/
ret = srv_msgq_set_blocking_threshold(mqid, 10);
```

注意

- デフォルト値 (**srv_msgq_set_blocking_threshold()** が呼び出されなかった場合) は **CS_NO_LIMIT** です。
- メッセージキューの動作を以前のバージョンの Open Server と同じにするには、スレッショルドを **CS_NO_LIMIT** に設定します。 保存されたメッセージがサーバワイドの最大数に到達すると、**srv_putmsgq()** によるブロックは行われずエラーになります。 メッセージのサーバワイドの最大値は、**SRV_S_MSGPOOL** プロパティを使用して指定します。
- スレッショルドを 0 (ゼロ) に設定すると、**srv_getmsgq()** を使用してメッセージが取得されるまで、このメッセージキューに対する **srv_putmsgq()** の呼び出しのすべてがブロックされます。
- スレッショルドを **CS_NO_LIMIT** 以外の負の値に設定することはできません。
- また、このスレッショルドは、メッセージキューに保存可能なメッセージのサーバワイドの最大値より大きい値に設定することはできません。 メッセージのサーバワイドの最大値は、**SRV_S_MSGPOOL** プロパティを使用して指定します。
- スレッショルドがキュー内の現在のメッセージ数より小さい値に設定された場合、新しいメッセージブロックを追加すると、キューから十分な数のメッセージが削除されて新しい上限に適合できるまで、呼び出し元のスレッドがブロックされます。
- スレッショルドがキュー内の現在のメッセージ数より大きい値に設定された場合は、メッセージがキューから削除されるたびにブロックされたスレッドが 1 つずつブロック解除されます。
- *SRV_M_WAIT* フラグを指定して **srv_putmsgq()** を呼び出した場合は、スレッショルド値が考慮されません。 このフラグが使用されていた場合は、メッセージ自体がもう一度キューから取り出されるまで、呼び出し元がブロックされます。

CS_DATAFMT フォーマット指定子

新しいフォーマット指定子 **CS_FMT_SUBS_ILL_CHAR** が **CS_DATAFMT** 構造の「format」ビットマスク要素に追加され、Adaptive Server から送信された不正文字の変換が可能になりました。

Adaptive Server が **enable permissive unicode** 設定パラメータを使用しているときに、クライアントが不正な Unicode 文字を受信することがあります。
CS_FMT_SUBS_ILL_CHAR を設定して、Unicode 以外のデータを正常に変換できるようにします。

15.7 より前のバージョンでは、不正文字が検出された段階でエラーがレポートされていました。

新しい接続プロパティ

Open Client と Open Server の新しい接続プロパティでは、接続時にデフォルトデータベースを指定できます。

- **CS_PROP_INITIAL_DATABASE** - 接続中の初期データベースの設定に使用されます。 接続時はログインの成功後に、パラメータ化された **use database** コマンドがサーバに送信されます。 **use database** コマンドが失敗しても、接続は成功します。 エラー処理がインラインで実行される場合は、**ct_diag()** を使用してキャッシュされたエラーを調べ、**use database** コマンドが成功したか、失敗したかが確認できます。 **ct_diag()** は、**ct_connect()** の完了後に呼び出します。 クライアントメッセージコールバックハンドラがインストールされている場合は、**use database** コマンドの結果としてそのハンドラが呼び出されます。 このハンドラは、生成されたメッセージをチェックして、**use database** コマンドが失敗した場合の処理方法を決定します。 また、**CS_FAIL** を返して接続を終了することも、**CS_SUCCEED** を返して失敗を無視することもできます。
- **CS_PROP_CURRENT_DATABASE** - **ct_connect()** の完了後に、コネクションで最後に使用したデータベースが格納されます。 このプロパティは、クライアントライブラリがサーバからの *ENVCHANGE* データベーストークンを確認したときに設定されます。
- **CS_PROP_USE_LAST_DATABASE** - **CS_HAFAILOVER** ともに使用されるブールプロパティで、ポストフェールオーバデータベースを最新の **use database** コマンドの結果に設定します。 true の場合は、**CS_PROP_INITIAL_DATABASE** が、サーバからの ENVCHANGE データベーストークンストリームの送信でレポートされるデータベース名に更新されます。 フェールオーバでは、この更新後の値によって接続データベースが設定されます。

新しいサーバプロパティ SRV_S_ADJUSTRECVPARAMLEN

SRV_S_ADJUSTRECVPARAMLEN プロパティを使用すると、**srv_descfmt** API はクライアントから受信したパラメータデータの最大長を調整して返すことができます。

バージョン 15.7 SP 110 では、Open Server アプリケーションで **SRV_S_ADJUSTRECVPARAMLEN** プロパティを *CS_TRUE* に設定すると、Open Server で受信データの文字セット変換を実行する場合に、**srv_descfmt** によってクライアントから受信したパラメータの最大長を取得してパラメータデータの格納に十分な長さに調整することができます。

既存のアプリケーションとの下位互換性の維持のため、**SRV_S_ADJUSTRECVPARAMLEN** プロパティはデフォルトで *CS_FALSE* に設定されています。

Adaptive Server Enterprise のドライバおよびプロバイダ用の SDK 15.7 の機能

SP110 では、Adaptive Server ODBC ドライバ 15.7 と jConnect for JDBC 7.07 を対象とする機能の更新が行われています。

Adaptive Server ODBC ドライバの共有メモリ診断

アプリケーションユーザと管理者は Adaptive Server ODBC ドライバにより、ドライバとデータベースのパフォーマンスを監視できます。

この情報には、アプリケーションを使用してプログラムから、または、新しいユーティリティの **aseodbcstatus** を外部的に使用して、アクセスすることができます。 このユーティリティを使用するには、共有メモリを使用するようにインストルメンテーションを設定する必要があります。

アプリケーションの変更を伴わない Adaptive Server ODBC ドライバのインストルメンテーションの有効化
ODBC インストルメンテーションは、次の環境変数のいずれかを設定して有効化します。

- SYBASE_ODBC_FORCE_INSTRUMENTATION=1 - インストルメンテーションを有効にするように Adaptive Server ODBC ドライバを設定します。 この環境変数の値が設定されていない場合、または 1 以外の値に設定されている場合は、インストルメンテーションをプログラムから有効にすることができます。
- SYBASE_ODBC_INSTRUMENTATION_FINE=1 - 文レベルでネットワークトラフィックをモニタリングするように Adaptive Server ODBC ドライバを設定します。 この変数が設定されている場合は、現在実行中の文とともにネットワーク時刻が保存されます。 この環境変数を 1 に設定すると、アプリケーションが Sybase ODBC ライブラリにアクセスする際に複数のスレッドを使用できなくなります。 この環境変数が設定されていない場合は、ネットワーク時刻がアプリケーションレベルで収集されます。

共有メモリインストルメンテーションの設定
共有メモリでは、**aseodbcstatus** ユーティリティを使用してインストルメンテーションデータを使用できます。 共有メモリセグメントは、SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM 環境変数によって有効化され、設定されます。

```
SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM=<number of diagnostic
sections to put in one shared memory segment>
(example 512)
```

共有メモリインストルメンテーションを有効にするには、SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM を 0 (デフォルト) より大きい値に設定します。

この環境変数を小さい数値に設定すると、Adaptive Server ODBC ドライバ が使用する共有メモリセグメントが増え、オペレーティングシステムによっては、パフォーマンスに影響する場合があります。 この環境変数を大きな値に設定した場合は、Adaptive Server ODBC ドライバ が必要以上に大きな共有メモリセグメントを使用することがあります。

アプリケーションが使用する文のおおよその数がわかっている場合は、SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM の値をその数より少し多めに設定します。 たとえば、アプリケーションが 250 ～ 350 の文を使用する場合は、SYBASE_ODBC_STATEMENT_DIAGNOSTICS_SHMEM の値を 360 に設定します。 アプリケーションが使用する文の数の範囲が広い (100 ～ 10000 など) 場合は、文の最大数を考慮して設定すると、アプリケーションですべての文が網羅的に使用されない場合にメモリが過剰に使用される可能性があります。 この場合は、設定値を小さくして、メモリブロック数の増加を許容します。 このような状況では、アプリケーションが通常使用する文の最小数の 2 倍に設定してみてください。

***aseodbcstatus** ユーティリティを使用したインストルメンテーションデータの取得*

インストルメンテーションデータを取得するには、**aseodbcstatus** ユーティリティを使用して共有メモリセグメントに接続し、インストルメンテーションデータを表示します。

aseodbcstatus が受け入れるパラメータは次のとおりです。

- **-help** - 有効なパラメータのリストを表示します。
- **-check <memory_area> <pid>** - 特定のプロセス ID に指定されたメモリ領域の使用可能性をチェックします。 メモリ領域が使用できない場合は、**aseodbcstatus** が 0 以外のステータスで終了します。
- **-print <memory_area> <pid>** - 特定のプロセス ID に指定されたメモリ領域に含まれるインストルメンテーションデータを出力します。 データが不明の場合 (使用された **asedobcstatus** のバージョンが ODBC ドライバのバージョンより古い場合など) は、**asedobcstatus** が 0 以外のステータスで終了します。
- **-statement_diagnostics <pid> <sid> <filter | all>** - 指定された文 ID **(<sid>)** のインストルメンテーションデータを出力します。 文 ID として -1 を渡すと、すべての文に関するデータが出力されます。 **filter** が渡された場合は、カウントが 0 以外のインストルメンテーションデータのみが表示されます。

aseodbcstatus ユーティリティは取得するデータを制御する複数のメモリ領域を備えています。 有効な値は次のとおりです。

- InstrumentationTimes - 特定の ID のグローバルインストルメンテーションデータ。 これは、プロセスで使用されるすべての接続と文に関する結合データデータです。
- InstrumentationTimesName - InstrumentationTimes と **statement_diagnostics** の両方のインストルメンテーションデータの各行の名前を順序に従って並べたリスト。
- StatementIDs - **statement_diagnostics** で使用された文 ID をリストします。

プログラムからのインストルメンテーションの使用
アプリケーションでは、環境、接続、および文の属性を直接使用して、インストルメンテーションを有効化し、アクセスすることができます。 環境属性と接続属性は同じで、どちらもアプリケーションに対してグローバルに動作します。 接続属性は、カスタム環境属性をサポートしないドライバマネージャを使用しているアプリケーションからアクセスできます。 属性は次のとおりです。

- **SQL_ATTR_INSTRUMENTATION** - インストルメンテーションの動作を制御します。 サポートされている値は次のとおりです。
  - **SQL_INSTRUMENTATION_ENABLE** - インストルメンテーションデータの収集をオンにします。
  - **SQL_INSTRUMENTATION_DISABLE** - インストルメンテーションデータの収集をオフにします。
  - **SQL_INSTRUMENTATION_CLEAR** - これは、文属性でサポートされる唯一の値です。 環境属性または接続属性に対して設定した場合は、**SQL_INSTRUMENTATION_CLEAR** によってグローバルインストルメンテーションデータがクリアされます。 文属性に対して設定した場合は、**SQL_INSTRUMENTATION_CLEAR** によってその文のインストルメンテーションデータがクリアされます。
  - **SQL_INSTRUMENTATION_CLEAR_ALL** - グローバルとすべての文インストルメンテーションがクリアされます。
  - **SQL_INSTRUMENTATION_FINE** - ロック、ネットワーク、および **select** 文とバッチのさまざまな側面など、より詳細なインストルメンテーションデータの収集を可能にします。
- **SQL_ATTR_INSTRUMENTATION_LOG** - **SQLWCHAR** 文字列としてフォーマットされたインストルメンテーションデータを取得します。 環境または接続に対して使用された場合、**SQL_ATTR_INTRUMENTATION_LOG** はグローバルインストルメンテーションデータを取得します。 文に対して使用された場合、**SQL_ATTR_INTRUMENTATION_LOG** はその文のインストルメンテーションデータのみを取得します。 文字列はセミコロン区切りリストとしてフォーマットされます。 それぞれの項目のフォーマットは、次のとおりです。

```
<instrumentation name>:<time in us>,<count >
```

次に例を示します。

```
Unknown:0,0; SocketRetrieve:75,19;
Waiting for lock XATransactionManager:0,0;
Holding lock XATransactionManager:0,
0; SQLAllocHandle:149,20;
```

64 ビット Linux でサポートされる Sybase iAnywhere ODBC Driver Manager

Adaptive Server Enterprise ODBC ドライババージョン 15.7 SP 110 では、Linux x86_64 と Linux Power 64 ビット上で、バージョン 16.0 の Sybase iAnywhere ODBC Driver Manager がサポートされます。

サポートされるプラットフォームの詳細については、Sybase Adaptive Server Enterprise ODBC ドライバの『ユーザーズガイド 15.7』を参照してください。

注意： バージョン 16 の Sybase iAnywhere ODBC Driver Manager は Microsoft Windows ではサポートされません。

jConnect の PRE_CACHE_DATATYPE_INFO 接続プロパティ

jConnect は **PRE_CACHE_DATATYPE_INFO** 接続プロパティを使用して、ログイン時にデータ型メタデータをキャッシュすることによって、それ以降のデータアクセスパフォーマンスを向上させます。

データ型メタデータを取得するために **Statement** またはその派生インタフェースを繰り返し使用する場合、**PRE_CACHE_DATATYPE_INFO** を true に設定することで、パフォーマンスを向上させることができます。

PRE_CACHE_DATATYPE_INFO を true に設定すると、さまざまな `ResultsetMetadata` API で使用される `isCaseSensitive` や `isSearchable` などのユーザ定義データ型に関する情報が接続時にキャッシュされます。 それ以降、この情報にアクセスする際はキャッシュにある情報を利用できます。

PRE_CACHE_DATATYPE_INFO が false (デフォルト) の場合、jConnect はユーザ定義データ型情報をキャッシュしません。

注意： 接続先のデータベースに存在するユーザ定義データ型の数によっては、接続の確立に要する時間が長くなることがあります。

Perl 用 Adaptive Server Enterprise 拡張モジュール

Perl 用 Adaptive Server Enterprise 拡張モジュールは、Kerberos 接続、データソース名スタイル (DSN スタイル) 更新後の接続の属性とメソッド、ロケールと charset の設定、および更新後のデータベースハンドル属性がサポートされます。

Perl ドライバの DSN スタイル接続プロパティ

Perl ドライバでは、いくつかの新しい DSN プロパティの追加と変更が行われています。

バージョン 15.7 SP 110 で現在サポートされているプロパティとその値の正式なリストを以下に示します。

SybaseASE ドライバの Connect 構文

DBI connect() メソッドでは、属性と値のペアの設定に次のルールが適用されます。 このメソッドの DSN 文字列パラメータには、前述のように、**dbi:SybaseASE:** の後ろに 1 つ以上の *name=value* 要素のセミコロン (;) 区切り文字列を続けて入力する必要があります。

- **Name** - 等号 (=) またはセミコロン (;) で区切ることが可能な値。大文字と小文字が区別されます。 1 つの属性に複数の同義語がある場合があります。 たとえば、server と servername は同じ属性を指します。
- **等号 (=)** - Name に割り当てる値の開始を示します。 等号がない場合、名前は true 値のブール型とみなされます。
- **Value** - セミコロン (;) で終了する文字列。 値内にセミコロンまたは別のバックスラッシュ (¥) がある場合は、バックスラッシュを使用します。 値は、ブール型、整数型、または文字列型で指定できます。 ブール型の有効な値は、true、false、on、off、1、0 です。

注意： 値がないブール名が存在する場合は、ブール型が true に設定されます。

有効な属性名と属性値

dsn キーワード引数の属性名と属性値をまとめたリストを以下に示します。

名前	説明	値
ANSINull	SQL の等号 (=) または不等号 (!=) の比較で NULL 値のオペランドの評価が ANSI 標準に準拠しているかどうかを判別する。 値が true の場合、Adaptive Server は、*=NULL* と *is NULL* は同義ではないとする ANSI の動作を適用する。 標準の Transact-SQL® では、*=NULL* と *is NULL* は同義とみなされる。 このオプションは、*<> NULL* と *is not NULL* の動作にも同様に作用する。	ブール値。 デフォルトは false。

名前	説明	値
BulkLogin	接続でバルクコピー操作を実行できるかどうかを判別する。	ブール値。 デフォルトは false。
ChainXacts	true の場合、Adaptive Server は連鎖トランザクション動作を使用する。つまり、各サーバコマンドが個別のトランザクションとみなされる。 Adaptive Server は次の各文の前に、begin transaction を暗黙的に実行する。 **delete**、**fetch**、**insert**、**open**、**select** および **update**。 ただし、トランザクションを明示的に終了またはロールバックする必要がある。 false の場合、アプリケーションは **begin transaction** 文を **commit** 文または **rollback** 文とペアにして明示的に指定する必要がある。	ブール値。 デフォルトは false。
Charset	この接続で使用する **charset** を指定する。	文字列値。 デフォルト文字セットは、現在、**iso_1** に設定されている。
Confidentiality	接続でデータの暗号化サービスを実行するかどうか。	ブール値。 デフォルトは false。
CredentialDelegation	ユーザの委任クレデンシャルを使用して、サーバに他のサーバへの接続が許可されているかどうかを判別する。	ブール値。 デフォルトは false。
DetectReplay	接続のセキュリティメカニズムがリプレイされた転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。
DetectOutOfSequence	接続のセキュリティメカニズムが不正なシーケンスの転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。
Hostname	クライアントマシンのホスト名。	文字列値。
HostPort	接続先サーバのホストとポートの組み合わせを指定する。	文字列値。 文字列のフォーマットは、“hostname portnumber” または “hostname: portnumber ”。

名前	説明	値
Integrity	接続のセキュリティメカニズムがデータ整合性チェックを実行するかどうかを判別する。	ブール値。 デフォルトは false。
Interfaces	interfaces ファイルのパスと名前。	文字列値。
Keytab	接続のセキュリティメカニズムが *username* 値とともに使用されるセキュリティキーを読み込むファイルのパスと名前。	文字列値。 デフォルトは NULL。つまり、接続前にユーザがクレデンシャルを確立しておく必要がある。
Locale	メッセージ、データ型変換、日時フォーマットで使用する言語と文字セットを判別する。	文字列値。
Language	メッセージ、データ型変換、および日時フォーマットで使用する言語セットを判別する。	文字列値。
LoginTimeout	ログインタイムアウト値を指定する。	整数値。
MaxConnect	あるコンテキストで同時にオープン可能な接続の最大数を指定する。	整数値。 デフォルト値は 25。 負の値および 0 は使用できない。
MutualAuthentication	サーバがクライアントに対して自身を認証する必要があるかどうかを判別する。	ブール値。 デフォルトは false。
NetworkAuthentication	接続のセキュリティメカニズムがネットワークベースのユーザ認証を実行するかどうかを判別する。	ブール値。 デフォルトは false。
PacketSize	TDS パケットサイズを指定する。	整数値。
Password	サーバへのログインに使用するパスワードを指定する。	文字列値。
PasswordEncryption **EncryptPassword**	接続が非対称のパスワード暗号化を使用するかどうかを判別する。	ブール値。 デフォルトは false。

名前	説明	値
SecurityMechanism	接続のためのセキュリティサービスを実行するネットワークセキュリティメカニズムの名前を指定する。	文字列値。 デフォルト値はセキュリティドライバ設定によって異なる。
Server **Servername**	接続先のサーバの名前を指定する。	文字列値。
ServerPrincipalName **Kerberos**	オープンする接続の接続先サーバのネットワークセキュリティプリンシパル名を指定する。	文字列値。 デフォルトは NULL。つまり、接続ではサーバのプリンシパル名がその *ServerName* 値と同じであるとみなされる。
ScriptName	サーバへのログインに使用されるアプリケーション名。	文字列値。
SslCAFile	信頼される CA 証明書が収録されたファイルへのパス。	文字列値。
TDSKeepalive	KEEPALIVE オプションを使用するかどうかを判別する。	ブール値。 デフォルトは true。
Timeout	接続タイムアウト値を指定する。	整数値。
UID **User** **Username**	サーバへのログインに使用する名前を指定する。	文字列値。

Python 用 Adaptive Server Enterprise 拡張モジュール

Python 用 Adaptive Server Enterprise 拡張モジュールは、バルク操作と LOB カラムのバルクコピーに対する新しいプロパティをサポートします。

バルクコピー操作のプロパティの設定

アプリケーションでは、特定のバルクプロパティを設定してから、バルクコピー操作を開始できます。

`blkcursor` オブジェクトの **copy()** メソッドを使用してプロパティを設定します。

このメソッドが受け入れる引数は次のとおりです。

- **name** - バルクコピー操作の実行対象のテーブル名。
- **direction** - これは、in と out の値を持つキーワード引数です。
- **properties** - 操作のプロパティ。 これは、name=value 要素のセミコロン区切り文字列です。
  - **Name** - 等号 (=) またはセミコロン (;) で区切ることが可能な値。大文字と小文字が区別されます。 1 つの属性に複数の同義語がある場合があります。
  - **等号 (=)** - Name に割り当てる値の開始を示します。 等号がない場合、Name は true の値を持つブール型とみなされます。
  - **Value** - セミコロン (;) で終了する文字列。 値内にセミコロンまたは別のバックスラッシュ (¥) が存在する場合は、バックスラッシュを使用します。 値は、ブール型、整数型、または文字列型のいずれかを使用できます。 ブール型の有効な値は、true、false、on、off、1、0 です。

注意： 値がないブール名が存在する場合、ブール型を true に設定する必要があります。

次に例を示します。

```
blk.copy(name="mytable", direction="in",
properties="IdStartNum=21")
```

有効な属性名と属性値

プロパティキーワード引数の有効な属性名と属性値を以下に示します。

名前	説明	値
Identity	テーブルの identity カラムの値が挿入される各ローに対して明示的に指定されているかどうか。 バルクコピーイン操作にプロパティ IdStartNum が設定されている場合はこのプロパティを true に設定できない。	ブール値。デフォルトは false。
IdStart-Num	挿入されるローの identity カラムの開始値。 最初に挿入されたローでこの値が使用され、後続の各ローで値が順に増加する。 バルクコピーイン操作に対してプロパティ Identity が true に設定されている場合はこのプロパティを設定できない。	整数値。デフォルト値なし。

Identity カラムを含むテーブルに対するバルクコピー操作

identity カラムを含むテーブルに対してバルクコピー操作を実行します。
identity カラムが含まれるバルクコピーイン操作でローを転送する場合は、デフォルトで、identity カラムの値が指定されません。 この値はサーバによって生成されます。

次に例を示します。

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulklogin=true;chainxacts=off")
cur = conn.cursor()
cur.execute("create table mytable (empid int identity, empname
varchar(20))")
            cur.close()
            blk = conn.blkcursor()

# Start bulk copy in operation. Do not specify values for identity
columns.
# Values will be generated by the server.
blk.copy(name="mytable", direction="in")
blk.rowxfer(["Joanne"])
blk.rowxfer(["John"])
blk.done()
```

- **IdStartnum** プロパティによる identity カラムの初期開始値の指定。
  次に例を示します。
  ```
  # Specify starting identity column value of 11 for the copy
  operation.
  blk.copy(name="mytable", direction="in",
  properties="IdStartNum=11")
  blk.rowxfer(["Max"])
  blk.rowxfer(["Danny"])
  blk.done()
  ```
- アプリケーションによる identity カラムの値の明示的な指定に使用される **Identity** プロパティ。
  次に例を示します。
  ```
  # Values for identity columns will have to be specified.
  blk.copy(name="mytable", direction="in",
  properties="identity=on")
  blk.rowxfer([21, "Maya"])
  blk.rowxfer([22, "Uma"])
  blk.done()
  ```

LOB カラムのバルクコピー

Python モジュールでは、text カラムと image (LOB) カラム に関連するバルクコピー操作がサポートされるようになりました。

LOB オブジェクトのコンストラクタ、型、およびメソッド
アプリケーションによって、特殊な値を格納するオブジェクトの作成に使用される専用のコンストラクタと型が提供されます。 アプリケーションでは、コンストラクタを使用してバルクコピーイン操作の Python オブジェクトを text カラムまたは image カラムとしてバインドする必要があります。 `blkcursor` メソッドに渡された場合は、モジュールが入力パラメータの適切な型を検出してそれに応じてパラメータをバインドします。

ラージオブジェクト (LOB) のサポート

Python は、ラージオブジェクト (LOB) データ型の *text* と *image* の使用をサポートします。

コンストラクタ:

Lob(type, obj) - LOB 値を保持するオブジェクトを作成します。

受け入れる引数は次のとおりです。

- `type` - LOB オブジェクトの型。 後述のように、値として TEXT または IMAGE を設定できます。
- `obj` - Python バッファオブジェクト。 バッファプロトコルをサポートする任意のオブジェクトです。 このようなオブジェクトには組み込みの bytearray が含まれます。

型

TEXT 型 - データベース内の text カラムを記述します。

IMAGE 型 - データベース内の image カラムを記述します。

LOB オブジェクトメソッド

.readinto(bytearray) - text または image カラムにバインドされた LOB オブジェクトのデータを取得するバルクコピーアウト操作で使用する必要があります。 このメソッドは読み取られたバイト数を返します。 カラム値のコピーが完了したことを示す場合は None オブジェクトを返します。 アプリケーションは、None オブジェクトが返されるまで、このメソッドを繰り返し呼び出す必要があります。 チャンクごとに読み取られるバイト数は、**bytearray** のサイズによって決まります。 このメソッドが受け入れる引数は次のとおりです。

bytearray - カラムからのデータが読み込まれ、この配列がコピーされます。

バルクコピーイン操作での LOB カラム

バルクコピーイン操作の場合、アプリケーションは `LOB()` コンストラクタを使用して text カラムまたは image カラムに転送する Python オブジェクトをマークする必要があります。

次に例を示します。

```
conn = sybpydb.connect(dsn="user=sa;bulklogin=true;chainxacts=off")
cur = conn.cursor()cur.execute("create table mytable (id int, t text,
i image)")
cur.close()
blk = conn.blkcursor()
blk.copy("mytable", direction="in")

# Transfer text and image data using a bytearray.
```

```
arr1 = bytearray(b"hello this is some text data")
lob1 = sybpydb.Lob(sybpydb.TEXT, arr1)
arr2 = bytearray(b"hello this is some image data")
lob2 = sybpydb.Lob(sybpydb.IMAGE, arr2)
blk.rowxfer([1, lob1, lob2])
```

Python では、複数の方法でファイルを開いて読み取ることができます。 以下の例では、バルクコピー操作でファイルを転送するためのメモリマップの使い方を示します。

```
# Transfer data from a file using memory maps.
fh1 = open("file1", "rb")
mp1 = mmap.mmap(fh1.fileno(), 0 , access=mmap.ACCESS_READ)
arr1 = bytearray(mp1)
lob1 = sybpydb.Lob(sybpydb.TEXT, arr1)
fh2 = open("file2", "rb")
mp2 = mmap.mmap(fh2.fileno(), 0 , access=mmap.ACCESS_READ)
arr2 = bytearray(mp2)
lob2 = sybpydb.Lob(sybpydb.IMAGE, arr2)
blk.rowxfer([2, lob1, lob2])
```

バルクコピーアウト操作での LOB カラム

text カラムと image カラムのバルクコピーアウト操作では、転送対象の text カラムと image カラムをローの最後に配置する必要があります。

text カラムと image カラムのデータは LOB オブジェクトとして返されます。 ローはテーブルから 1 つずつ転送する必要があります。 各ローの転送後に、ロー内の各 LOB オブジェクトのデータを取得する必要があります。 `readinto()` メソッドは、すべてのカラム値がコピーされたことを示す None オブジェクトが返されるまで繰り返し呼び出す必要があります。

次に例を示します。

```
# Method to read data from a lob object
def getlobdata(lob):
    outarr = bytearray()
    chunk = bytearray(1024)
    while True:
        len = lob.readinto(chunk);
        if (len == None):
             break
        outarr.extend(chunk[:len])
return outarr

blk.copy("mytable", direction="out")
# The rows should be transferred one by one.
row = blk.rowxfer()
print(row[0])
# Now read the lob data for the text column column
arr1 = getlobdata(row[1])
print(arr1.decode())
# Now read the lob data for the text column column
arr2 = getlobdata(row[2])
```

```
print(arr2.decode())
```

SP100 の新機能

SP100 では、バージョン管理番号の変更点と、Open Client 15.7、Open Server 15.7、および SDK 15.7 用に更新された機能が導入されます。

リリースバージョン番号の変更

これまで Sybase® のお客様にはメジャーリリースまたはマイナーリリースに続く ESD (Electronic Software Delivery) として知られてきたソフトウェアパッチを SP (サポートパッケージ) と呼ぶようになりました。SP には最大 3 桁までの数字が割り振られます。

『SAP 製品リリース方針 - すべての主要ソフトウェアリリース』 (https://service.sap.com/releasestrategy) を参照してください。このバージョン番号の変更によるアップグレード手順やダウングレード手順の変更はありません。

インストーラの変更点

SDK と Open Server のインストーラは、バージョンと下位互換性に関して強化されています。

- SDK と Open Server のインストーラは、インストールするバージョンがインストールディレクトリ内のバージョンと互換性があり、そのバージョンの上にインストールできるかどうかをチェックするようになりました。
  インストールするバージョンが、インストールディレクトリ内のバージョンに含まれるバグ修正を使用できない場合、このインストールは互換性がないとみなされます。
  すでにインストールされているバージョンが互換性がある場合、インストールは正常に実行されます。
  すでにインストールされているバージョンが、インストールするバージョンと互換性がない場合、インストールプロセスは停止します。 次のことができます。
  - エラーを無視して続行する。または
  - インストールをアボートし、 互換性のあるバージョンがソフトウェアダウンロードサイトで提供されているかどうかをチェックする。
- 下位互換性を確保するために、インストーラは 15.7 GA から 15.7 SP100 までのすべてのバージョンのセキュリティおよびディレクトリドライバファイルをインストールします。

Open Client 15.7 と Open Server 15.7 の機能

Open Client 15.7 と Open Server 15.7 は、MIT Kerberos ライブラリをサポートしています。

新しい MIT Kerberos ライブラリにおける Sybase Kerberos ドライバのサポート

Windows 64 ビット版用の新しい MIT Kerberos ライブラリであるバージョン 4.0.1 を、Sybase Kerberos ドライバ `libsybskrb64.dll` とともに使用できます。

Windows 64 ビット版で MIT Kerberos GSS ライブラリを使用するには、次のエントリを `%SYBASE%¥OCS-15_0¥ini¥libtcl64.cfg` ファイルの SECURITY セクションに追加します。

```
[SECURITY]
csfkrb5=libsybskrb64.dll secbase=@MYREALM libgss=C:
¥Kerberos_winx64¥bin¥gssapi64.dll
```

C:¥Kerberos_winx64 は MIT Kerberos がインストールされている場所です。

注意： Kerberos gssapi ライブラリのパスにはスペースを含めないでください。

Adaptive Server Enterprise のドライバおよびプロバイダ用の SDK 15.7 の機能

SP100 には、Adaptive Server ODBC ドライバ 15.7、jConnect 7.07、および Adaptive Server ADO.NET Data Provider 15.7 用の新しい機能が導入されています。

WindowsCharsetConverter 接続プロパティ

(Microsoft Windows のみ) バージョン 15.7 SP 100 以降、新しい接続プロパティ **WindowsCharsetConverter** によって、使用する変換ライブラリ (Sybase Unicode Infrastructure Library (Unilib) または Microsoft Unicode 変換ルーチン) をユーザが選択できるようになりました。

バージョン 15.5 以降、Windows プラットフォーム上の Adaptive Server Enterprise ADO.NET Data Provider、Adaptive Server Enterprise OLEDB Provider、および Adaptive Server Enterprise ODBC ドライバ では Sybase Unicode Infrastructure Library (Unilib) が文字セット変換に使用されます。

15.5 より前のバージョンでは、Microsoft Unicode 変換ルーチンが使用されます。

この 2 つのライブラリには、変換の実行方法に若干の違いがあります。

接続文字列内で、**WindowsCharsetConverter** を次のいずれかに設定します。

- 0 - (デフォルト) Unilib ライブラリを使用します。
- 1 - Microsoft Unicode 変換ルーチンを使用します。

注意： アプリケーションが Unilib との特定の変換差異に依存している場合は、Microsoft Unicode 変換ルーチンを使用します。

Windows 以外のオペレーティングシステムでは、文字セット変換に Unilib のみがサポートされます。**WindowsCharsetConverter** を 1 に設定しても何も作用しません。

Adaptive Server for SQL Server 2012 へのデータ転送を高速化する SSIS Custom Data Flow Destination コンポーネント

Adaptive Server ADO.NET Data Provider のディストリビューションには SQL Server Integration Services (SSIS) Custom Data Flow Destination コンポーネントが含まれています。これは、SQLServer 2012 と互換性があり、バルク挿入プロトコルを使用して Adaptive Server Enterprise へのデータ転送を高速化します。

このカスタムデータフロー変換先コンポーネントは、`AseBulkCopy` クラスでサポートされる Adaptive Server のバルク挿入プロトコルを使用します。 この名前付き `Sybase.AdoNet4.AseDestination.dll` コンポーネントは、Adaptive Server ADO.NET Data Provider とともにインストールされ、32 ビットシステムでは `%SYBASE%¥DataAccess¥ADONET¥dll` に、64 ビットシステムでは `%SYBASE%¥DataAccess64¥ADONET¥dll` にインストールされます。

SQLServer 2008 と互換性のある Custom Data Flow Destination コンポーネントのバージョンについては、ESD #5 セクションの「Adaptive Server へのデータ転送を高速化する新しい SSIS Custom Data Flow Destination コンポーネント」を参照してください。

注意： SQL Server 2008 からのデータ転送用 SSIS 変換先コンポーネントは、`Sybase.AdaptiveServerAdoNetDestination.dll` から `Sybase.AdoNet2.AseDestination.dll` に名前が変更されています。

Adaptive Server ADO.NET Destination SSIS コンポーネントの設定

Adaptive Server ADO.NET Destination SSIS コンポーネントは、Adaptive Server の変換先へのデータ転送を迅速に行います。

1. `Sybase.AdoNet4.AseDestination.dll` を `C:¥Program Files ¥Microsoft SQL Server¥110¥DTS¥PipelineComponents` および `C:`

¥Program Files (x86)¥Microsoft SQL Server¥110¥DTS ¥PipelineComponents にコピーします。

2. ステップ 1 で使用したローカルドライブの Microsoft SQL Server ディレクトリのいずれかから、SDK インストールで提供される AseGacUtility4.exe を使用して Sybase.AdoNet4.AseDestination.dll を登録します。
3. Windows で SQLServer 2012 Data Tools または SQL Server 2012 Data を起動するには、[スタート] > [すべてのプログラム] > [Microsoft SQL Server 2012] > [SQL Server Data Tools]を選択します。
4. [ファイル] > [新規作成] > [プロジェクト] > [Integration Services プロジェクト] を選択します。
   [SSIS] ツールボックスに Sybase 変換先コンポーネントが自動的に表示されます。
5. [制御フロー項目] ツールボックスから、制御フローオブジェクトをドラッグアンドドロップします。
6. [データフローの変換先] タブ、[データフローの変換元ツールボックス] タブの順に選択し、[Sybase AdoNet4 ASE Destination] と [ADO NET Source Component] を [データフロー] タブにドラッグアンドドロップします。
7. [接続マネージャー] ウィンドウに使用可能な変換元または変換先が表示されない場合は、[接続マネージャー] ウィンドウを右クリックして、[新しい ADO.NET 接続] を選択します。 既存のデータ接続がある場合はそれを選択し、そうでない場合は、[新規作成] をクリックします。
8. 変換先 Adaptive Server への接続を新たに作成するには、[ADO.NET の接続マネージャーの構成] ウィンドウで、[新規作成] をクリックして、[Sybase Adaptive Server Enterprise Data Provider] を選択します。
9. [接続マネージャー] ウィンドウで、接続のプロパティを入力します。
10. バルク挿入を有効にするには、[追加の接続プロパティ] テキストボックスに次のように入力します。
    ```
    enablebulkload=1
    ```

    注意： バルク挿入使用の詳細については、『Adaptive Server Enterprise ADO.NET Data Provider ユーザーズガイド』の「AseBulkCopy」を参照してください。
11. [OK] をクリックします。
12. データフローの ADO.NET 変換元に、接続とデータアクセスモードを設定します。 ADO.NET 変換元からデータフローパスに接続したら、[Sybase AdoNet4 ASE Destination] を右クリックして、[高度な編集の表示] を選択します。

13. [接続マネージャー] タブの [接続マネージャー] フィールドから ASE の接続を選択します。 [コンポーネントのプロパティ] タブで、TableName プロパティを変換先のテーブル名に設定します。
14. [入力列] タブを選択してから、[名前] を選択します。 これにより、変換元テーブルで指定されたすべてのカラムが選択されます。
15. [OK] をクリックして、接続を確立します。

    SQL Server Integration Service を使用したデータ転送の詳細については、Microsoft SSIS のマニュアルを参照してください。

Adaptive Server ADO.NET Data Provider での SSRS のサポート

Adaptive Server ADO.NET Data Provider のディストリビューションには、Microsoft SQL Server Reporting Services (SSRS) のカスタムデータ拡張機能コンポーネントが含まれているため、ユーザは認証情報をレポートサーバに格納することができます。

Adaptive Server SSRS コンポーネントでは、以下がサポートされます。

- Microsoft SQL Server 2008
- Microsoft SQL Server 2008 R2

この名前付き Sybase.AdoNet2.AseReportingServices コンポーネントは Adaptive Server ADO.NET Data Provider とともに次のディレクトリにインストールされます。 32 ビットシステムの場合は、`%SYBASE%¥DataAccess¥ADONET¥dll` で、64 ビットシステムの場合は、`%SYBASE%¥DataAccess64¥ADONET¥dll` です。

Adaptive Server ADO.NET SSRS コンポーネントの設定

Adaptive Server ADO.NET SSRS コンポーネントを設定します。

1. `Sybase.AdoNet2.AseReportingServices.dll` を `C:¥Program Files (x86)¥Microsoft Visual Studio 9.0¥Common7¥IDE ¥PrivateAssemblies` にコピーします。
2. テキストエディタを使用して `C:¥Program Files (x86)¥Microsoft Visual Studio 9.0¥Common7¥IDE¥PrivateAssemblies` から `RSReportDesigner.config` を開きます。
   - Data セクションの下に以下を入力します。

     ```
     <Extension Name="Sybase"
     Type="Sybase.AdoNet2.AseReportingServices.SybaseClientConnecti
     onWrapper,Sybase.AdoNet2.AseReportingServices"/>
     ```

   - Designer セクションの下に以下を入力します。

```
<Extension Name="Sybase"
Type="Microsoft.ReportingServices.QueryDesigners.GenericQueryD
esigner,Microsoft.ReportingServices.QueryDesigners"/>
```

3. `RSReportDesigner.config` ファイルを保存します。

Adaptive Server Enterprise のドライバおよびプロバイダ用の LDAPS 機能

LDAP URL で、`ldap` ではなく `ldaps` が指定されている場合は LDAP サーバへの SSL 接続が確立されます。

UNIX

`odbc.ini` (または接続文字列) で DSN に対して指定する必要がある属性の例を次に示します。

```
DSURL = ldaps://huey:636/dc=sybase,dc=com????
bindname=cn=Manager,dc=Sybase,dc=com?secret
DSServiceName = myAse
TrustedFile = /usr/u/sybase/config/trusted.txt
```

LDAP サーバの証明書への署名に使用した認証局の署名付き証明書を `trusted.txt` ファイルに追加する必要があります。

Windows

接続文字列で指定する必要がある属性の例を次に示します。

```
DSURL = ldaps://huey:636/dc=sybase,dc=com????
bindname=cn=Manager,dc=Sybase,dc=com?secret
DSServiceName = myAse
```

LDAP サーバの証明書への署名に使用した認証局の署名付き証明書を Microsoft Certificate Store にインストールする必要があります。

SAP jConnect での SSL サポート

15.7 SP 100 より前のバージョンの SAP jConnect で SSL ソケットを使用するには、[SybSocketFactory] インタフェースの実装を作成し、**SYBSOCKET_FACTORY** 接続プロパティを設定してその実装を使用する必要があります。

SAP jConnect には、SSL ソケットを使用して SAP Adaptive Server に接続するためのサポートが組み込まれています。 新しい接続プロパティ **ENABLE_SSL** の設定に応じた動作は次のとおりです。

- false - (デフォルト) SAP jConnect では SSL ソケットは使用されません。
- true - SAP jConnect で SSL ソケットが使用され、対象の SAP Adaptive Server で SSL ソケット接続が有効になっている必要があります。 SAP jConnect では、**SYBSOCKET_FACTORY** 接続プロパティは無視されます。

注意： DriverManager.setLoginTimeout プロパティを使用してログインタイムアウトを設定して、SSL が有効になっていない SAP Adaptive Server で SSL 接続を試行中に、接続のタイムアウトを許可することをおすすめします。

SSL ソケット機能は、次の標準の Java プロパティに依存します。

- **javax.net.ssl.keyStore**
- **javax.net.ssl.keyStorePassword**
- **javax.net.ssl.trustStore**
- **javax.net.ssl.trustStorePassword**
- **javax.net.ssl.trustStore**
- **javax.net.ssl.trustStoreType**

Java の標準プロパティの詳細については、Java J2SE 6 のドキュメントを参照してください。

ESD #7 の新機能

ESD #7 では、Open Client 15.7 と Open Server 15.7、SDK 15.7、Python 用 Adaptive Server Enterprise 拡張モジュール 15.7、および PHP 用 Adaptive Server Enterprise 拡張モジュール 15.7 を対象とする機能の更新が適用されています。

Open Client 15.7 と Open Server 15.7 の機能

Open Client 15.7 および Open Server 15.7 が拡張され、Client-Library 接続文字列プロパティ、リモートパスワード暗号化、および Windows 64 ビット版の `libsybsspiwrapper64.dll` がサポートされるようになりました。

接続文字列プロパティをサポートする Client-Library

Client-Library では、API ルーチンの **ct_connect_string()** がサポートされるようになりました。

ct_connect_string()
接続文字列を指定してサーバに接続します。

ct_connect_string() 関数は、**ct_connect()** と同じ機能を提供します。 また、接続時に特定の属性を設定するメカニズムも提供します。

構文

```
CS_RETCODE ct_connect_string(connection, connection_string, length)
CS_CONNECTION    *connection;
CS_CHAR                  *connection_string;
CS_INT                       length;
```

パラメータ

- *connection* - CS_CONTEXT 構造体へのポインタ。 CS_CONNECTION 構造体には、特定のクライアント/サーバ接続に関する情報が含まれています。 CS_CONNECTION 構造体を割り付けるには、**ct_con_alloc** を使用します。
- *connection_string* - 属性名と属性値を含む文字列。
- *length* - **connection_string* のバイト単位の長さ。 **connection_string* が null で終了している場合は、length を CS_NULLTERM として渡します。 connection_string が NULL の場合は、length を 0 または CS_UNUSED として渡します。

戻り値
ct_connect の戻り値は次のとおりです。

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。
CS_BUSY	この接続では、非同期操作が保留中である。『Open Client Client-Library/C リファレンスマニュアル』の「非同期プログラミング」を参照。

接続文字列は、name=value 要素のセミコロン区切り文字列です。

1. Name - 等号 (=) またはセミコロン (;) で区切ることが可能な値。大文字と小文字が区別されます。 1 つの属性に複数の同義語がある場合があります。 たとえば、**server** と **servername** は同じ属性を指します。
2. 等号 (=) - Name に割り当てる値の開始を示します。 等号がない場合は、Name が true 値のブール型とみなされます。
3. Value - セミコロン (;) で終了する文字列。 値内にセミコロンまたは別のバックスラッシュ (¥) がある場合は、バックスラッシュを使用します。 値は、ブール型、整数型、または文字列型で指定できます。 ブール型の有効な値は、true、false、on、off、1、0 です。

   注意： 値がないブール名がある場合、ブール型は true に設定する必要があります。

次に例を示します。

```
ct_connect_string(conn, "Username=me; Password=mypassword;
Servername=ASE", CS_NULLTERM);
```

有効な属性名と属性値

dsn キーワード引数に対する有効な属性名と属性値を以下の表に示します。

名前	説明	値
ANSINull	SQL の等号 (=) または不等号 (!=) の比較で NULL 値のオペランドの評価が ANSI 標準に準拠しているかどうかを判別する。 値が true の場合、Adaptive Server は、*= NULL* と *is NULL* は同義ではないとする ANSI の動作を適用する。 標準の Transact-SQL® では、*= NULL* と *is NULL* は同義とみなされる。 このオプションは、*<> NULL* と *is not NULL* の動作にも同様に作用する。	ブール値。 デフォルトは false。
BulkLogin	接続でバルクコピー操作を実行できるかどうかを判別する。	ブール値。 デフォルトは false。
ChainXacts	true の場合、Adaptive Server は連鎖トランザクション動作を使用する。つまり、各サーバコマンドが個別のトランザクションとみなされる。 Adaptive Server は次の各文の前に、begin transaction を暗黙的に実行する。 **delete**、**fetch**、**insert**、**open**、**select** および **update**。 ただし、トランザクションを明示的に終了またはロールバックする必要がある。 false の場合、アプリケーションは begin transaction 文を commit 文または rollback 文とペアにして明示的に指定する必要がある。	ブール値。 デフォルトは false。
Charset	この接続で使用する **charset** を指定する。	文字列値。
Confiden-tiality	接続でデータの暗号化サービスを実行するかどうか。	ブール値。 デフォルトは false。
Credential-Delegation	ユーザの委任クレデンシャルを使用してサーバを他のサーバに接続させるかどうかを判別する。	ブール値。 デフォルトは false。
DetectRe-play	接続のセキュリティメカニズムがリプレイされた転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。
DetectOu-tOfSe-quence	接続のセキュリティメカニズムが不正なシーケンスの転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。

名前	説明	値
Integrity	接続のセキュリティメカニズムがデータ整合性チェックを実行するかどうかを判別する。	ブール値。 デフォルトは false。
Interfaces	interfaces ファイルのパスと名前。	文字列値。
Keytab	接続のセキュリティメカニズムが *username* 値と一緒に使用されるセキュリティキーを読み込むファイルのパスと名前。	文字列値。 デフォルトは NULL。つまり、接続前にユーザがクレデンシャルを確立しておく必要がある。
Locale	メッセージ、データ型変換、日時フォーマットで使用する言語と文字セットを判別する。	文字列値。
Language	メッセージ、データ型変換、日時フォーマットで使用する言語セットを判別する。	文字列値。
LoginTimeout	ログインタイムアウト値を指定する。	整数値。
MaxConnect	あるコンテキストで同時にオープン可能な接続の最大数を指定する。	整数値。 デフォルト値は 25。 負の値および 0 は使用できない。
MutualAuthentication	サーバがクライアントに対して自身を認証する必要があるかどうかを判別する。	ブール値。 デフォルトは false。
NetworkAuthentication	接続のセキュリティメカニズムがネットワークベースのユーザ認証を実行するかどうかを判別する。	ブール値。 デフォルトは false。
PacketSize	TDS パケットサイズを指定する。	整数値。
Password	サーバへのログインに使用するパスワードを指定する。	文字列値。
PasswordEncryption	接続が非対称のパスワード暗号化を使用するかどうかを判別する。	ブール値。 デフォルトは false。
SecurityMechanism	接続のためのセキュリティサービスを実行するネットワークセキュリティメカニズムの名前を指定する。	文字列値。 デフォルト値はセキュリティドライバ設定によって異なる。

名前	説明	値
Server **Servername**	接続先のサーバの名前を指定する。	文字列値。
ServerPrin-cipalName	オープンする接続の接続先サーバのネットワークセキュリティプリンシパル名を指定する。	文字列値。 デフォルトは NULL。このとき接続は、サーバのプリンシパル名がその *ServerName* 値と同じであるとみなす。
TDS_Keepa-live	KEEPALIVE オプションを使用するかどうかを判別する。	ブール値。 デフォルトは true。
Timeout	接続タイムアウト値を指定する。	整数値。
UID **User** **Username**	サーバへのログインに使用する名前を指定する。	文字列値。

リモートパスワード暗号化

Open Server は、Extended Plus Encrypted Passwords (EPEP) を使用した接続用のリモートパスワードペアの取得をサポートします。

SRV_T_NUMRMTPWDS、SRV_T_RMTPWDS などのプロパティの取得は、**srv_thread_props()** を使用して行います。 クライアントが EPEP プロトコルをサポートしている場合は、SRV_T_NUMRMTPWDS プロパティが復号化されるリモートパスワードペアの数を返し、SRV_T_RMTPWDS プロパティがパスワードのペアを返します。

Windows 64 ビット用の libsybsspiwrapper64.dll

`libsybsspiwrapper64.dll` ラッパライブラリを使用して、Windows 64 ビット版プラットフォームで Kerberos セキュリティドライバによる Windows セキュリティサポートプロバイダインターフェイス (SSPI) ルーチンの使用を可能にします。

この機能を使用するには、`libtcl64.cfg` を編集して `libsybsspiwrapper64.dll` を含める必要があります。 次に例を示します。

```
[SECURITY]csfkrb5=LIBSYBSKRB64 secbase=@MYREALM libgss=C:¥Sybase
¥release¥OCS-15_0¥lib3p64¥libsybsspiwrapper64.dll
```

注意： このライブラリは `%SYBASE%¥OCS-15_0¥lib3p64` ディレクトリに保存されています。

Adaptive Server Enterprise のドライバおよびプロバイダ用の SDK 15.7 の機能

ESD #7 では、Adaptive Server ODBC ドライバ 15.7 と Ribo ユーティリティを対象とする新しい機能が導入されています。

Adaptive Server ODBC ドライバ 用の新しい CancelQueryOnFreeStmt 接続プロパティ

Adaptive Server ODBC ドライバ を使用して大量の結果セットを返すクエリを実行している Microsoft Access フォームが、結果セット全体の処理が終わる前に閉じられた場合は、ODBC ドライバ が結果セット全体の処理を完了するまで Microsoft Access が無応答状態になります。

バージョン 15.7 ESD #7 の新しい接続プロパティ **CancelQueryOnFreeStmt** はこの問題を解決します。 この接続プロパティが 1 に設定されると、フォームが閉じられるときは常に、Adaptive Server ODBC ドライバ がすべての保留中の結果をキャンセルして、すぐに制御を Microsoft Access アプリケーションに返します。 0 (デフォルト値) に設定された場合、Adaptive Server ODBC ドライバ の動作に変化はありません。

クライアント接続属性を設定するための新しい効率的な方法

バージョン 15.7 ESD #7 では、ODBC **SQLSetConnectAttr** API を使用したクライアント接続属性の設定を効率化するサポートが、Adaptive Server ODBC ドライバに追加されています。 設定した属性値は Adaptive Server の **sysprocesses** テーブルで参照可能で、異なる複数のクライアント接続の区別に役立ちます。

15.7 ESD #7 より前のバージョンでこれらの属性を設定するには、アプリケーションプログラムで明示的に **set** 文を呼び出して対応する属性を設定しなければならないため、サーバで別途実行する必要がありました。 **SQLSetConnectAttr** API を使用する場合は、ドライバが **set** 文の実行を遅らせ、実行される次の文に付加します。

注意： **set** 文は **SQLSetConnectAttr** API の呼び出し直後に実行されるわけではないため、その次の文が実行されるまで、設定された値を Adaptive Server 上で参照することができません。

SQLSetConnectAttr でサポートされる属性は、次のとおりです。

- **SQL_ATTR_CLIENT_NAME** - コマンドセット *clientname<value>* を使用してクライアント名を設定します。

- **SQL_ATTR_CLIENT_HOST_NAME** - コマンドセット *clienthostname <value>* を使用してクライアントホスト名を設定します。
- **SQL_ATTR_CLIENT_APPL_NAME** - コマンドセット *clientapplname <value>* を使用してクライアントアプリケーション名を設定します。

これらの属性の値は 30 バイトにトランケートされます。 これらの属性の値を取得するには、ODBC **SQLGetConnectAttr** を使用します。 ただし、このインタフェース外で行われたサーバの値の変更は反映しません。

Adaptive Server ODBC ドライバの data-at-exec 機能の拡張サポート

Adaptive Server ODBC ドライババージョン 15.7 ESD #7 では、data-at-exec 機能が拡張され、バルク操作とバッチ操作がサポートされるようになりました。これにより、メモリ使用量が減少し、アプリケーションのパフォーマンスが向上しています。

以前のバージョンでは、**SQLBulkOperations** を呼び出したりバッチを実行したりする前に、バインドされたパラメータのすべてのデータを完全にロードしておく必要がありました。 ESD #7 では、パラメータのデータをアプリケーションで事前にロードする必要はありません。データは **SQLPutData** を使用してチャンクで送信できます。 Adaptive Server ODBC ドライバのバッチプロトコル (**SQLExecute**/**SQLExecDirect** と **SQL_ATTR_BATCH_PARAMS**) を使用している場合は、**SQL_ATTR_PARAMSET_SIZE** が 1 に設定されていれば、data-at-exec がサポートされます。 LOB カラムに対して data-at-exec を使用するには、サーバが LOB パラメータをサポートしている必要があります。

Ribo ユーティリティの新しい -n コマンドラインオプション

Ribo ユーティリティが拡張され、新しいコマンドラインオプション `-n` を使用することで未加工の `.tds` ダンプファイルが扱いやすいファイルサイズの複数のファイルに変換されるようになりました。

15.7 ESD #7 より前のバージョンでは、Ribo ユーティリティは、サイズに関係なく、未加工の `.tds` ダンプファイル全体を単一の変換ファイルに変換していました。 Ribo ユーティリティが拡張され、新しいコマンドラインオプション `-n` を使用することで未加工の `.tds` ダンプファイルが扱いやすいファイルサイズの複数のファイルに変換されるようになりました。 単一の変換ファイルの最大サイズを `-n` オプションを使用して KB 単位で指定します。 変換出力ファイルのサイズが `-n` オプションで指定された値より大きくなった場合は、新しいファイルが作成されます。

出力ファイル名は次の命名規則に従います。

<output_file_part1_of_5> <output_file_part2_of_5>

ここで、**<output_file>** はユーザが指定したファイルで **partX_ofY** が付加されます。この *X* は現在の構成部分を表し、*Y* は変換後の出力の分割後の構成部分の合計数を表します。

注意： `-n` フラグは、変換の実行時に効力を発揮します。

Python 用 Adaptive Server Enterprise 拡張モジュール

Python 用 Adaptive Server Enterprise 拡張モジュールが強化され、データソース名スタイル (DSN スタイル) 接続プロパティ、新しいサンプルプログラム、および **blklib** がサポートされるようになりました。

DSN スタイル接続文字列プロパティのサポート

connect() メソッドでは、DSN スタイル接続プロパティのサポートが追加されました。

connect()
データベースへの接続を表す Connection オブジェクトを構成します。

このメソッドは、次のキーワード引数を受け入れます。

- **user** - サーバへのログイン時に接続が使用するユーザログイン名。
- **password** - サーバへのログイン時に接続が使用するパスワード。
- **servername** - クライアントプログラムが接続する Adaptive Server の名前。**servername** を指定しない場合、DSQUERY 環境変数で Adaptive Server の名前を定義します。
- **dsn** - データソース名。 データソース名は、セミコロンで区切られた名前=値の文字列です。
  - Name - 等号 (=) またはセミコロン (;) で区切ることが可能な値。大文字と小文字が区別されます。 1 つの属性に複数の同義語がある場合があります。たとえば、**server** と **servername** は同じ属性を指します。
  - 等号 (=) - Name に割り当てる値の開始を示します。等号がない場合、名前は true 値のブール型とみなされます。
  - Value - セミコロン (;) で終了する文字列。 値内にセミコロンまたは別のバックスラッシュ (¥) がある場合は、バックスラッシュを使用します。 値は、ブール型、整数型、または文字列型で指定できます。 ブール型の有効な値は、true、false、on、off、1、0 です。

    注意： 値がないブール名がある場合、ブール型は true に設定する必要があります。

  次に例を示します。

```
sybpydb.connect(user='name', password='password string',
                          dsn='servername=Sybase;timeout=10')
```

有効な属性名と属性値

dsn キーワード引数に対する有効な属性名とその値を以下の表に示します。

名前	説明	値
ANSINull	SQL の等号 (=) または不等号 (!=) の比較で NULL 値のオペランドの評価が ANSI 標準に準拠しているかどうかを判別する。 値が true の場合、Adaptive Server は、*= NULL* と *is NULL* は同義ではないとする ANSI の動作を適用する。 標準の Transact-SQL では、*= NULL* と *is NULL* は同義とみなされる。 このオプションは、*<> NULL* と *is not NULL* の動作にも同様に作用する。	ブール値。 デフォルトは false。
BulkLogin	接続でバルクコピーオペレーションを実行できるかどうかを判別する。	ブール値。 デフォルトは false。
ChainXacts	true の場合、Adaptive Server は連鎖トランザクション動作を使用する。つまり、各サーバコマンドが個別のトランザクションとみなされる。 Adaptive Server は次の各文の前に、begin transaction を暗黙的に実行する。 **delete**、**fetch**、**insert**、**open**、**select** および **update**。 ただし、トランザクションを明示的に終了またはロールバックする必要がある。 false の場合、アプリケーションは begin transaction 文を commit 文または rollback 文と対で明示的に指定する必要がある。	ブール値。 デフォルトは false。
Charset	この接続で使用する **charset** を指定する。	文字列値。
Confiden-tiality	接続でデータの暗号化サービスを実行するかどうか。	ブール値。 デフォルトは false。
Credential-Delegation	ユーザの委任クレデンシャルを使用してサーバを他のサーバに接続させるかどうかを判別する。	ブール値。 デフォルトは false。
DetectRe-play	接続のセキュリティメカニズムがリプレイされた転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。

名前	説明	値
DetectOutOfSequence	接続のセキュリティメカニズムが不正なシーケンスの転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。
Integrity	接続のセキュリティメカニズムがデータ整合性チェックを実行するかどうかを判別する。	ブール値。 デフォルトは false。
Interfaces	interfaces ファイルのパスと名前。	文字列値。
Keytab	接続のセキュリティメカニズムが *username* 値と一緒に使用されるセキュリティキーを読み込むファイルのパスと名前。	文字列値。 デフォルトは NULL。つまり、接続前にユーザがクレデンシャルを確立しておく必要がある。
Locale	メッセージ、データ型変換、日時フォーマットで使用する言語と文字セットを判別する。	文字列値。
Language	メッセージ、データ型変換、日時フォーマットで使用する言語セットを判別する。	文字列値。
LoginTimeout	ログインタイムアウト値を指定する。	整数値。
MaxConnect	あるコンテキストで同時にオープン可能な接続の最大数を指定する。	整数値。 デフォルト値は 25。 負の値および 0 は使用できない。
MutualAuthentication	サーバがクライアントに対して自身を認証する必要があるかどうかを判別する。	ブール値。 デフォルトは false。
NetworkAuthentication	接続のセキュリティメカニズムがネットワークベースのユーザ認証を実行するかどうかを判別する。	ブール値。 デフォルトは false。
PacketSize	TDS パケットサイズを指定する。	整数値。
Password	サーバへのログインに使用するパスワードを指定する。	文字列値。
PasswordEncryption	接続が非対称のパスワード暗号化を使用するかどうかを判別する。	ブール値。 デフォルトは false。

名前	説明	値
SecurityMechanism	接続のためのセキュリティサービスを実行するネットワークセキュリティメカニズムの名前を指定する。	文字列値。 デフォルト値はセキュリティドライバ設定によって異なる。
Server **Servername**	接続先のサーバの名前を指定する。	文字列値。
ServerPrincipalName	接続がオープンされるサーバのネットワークセキュリティプリンシパル名を指定する。	文字列値。 デフォルトは NULL。このとき接続は、サーバのプリンシパル名がその *ServerName* 値と同じであるとみなす。
Keepalive	KEEPALIVE オプションを使用するかどうかを判別する。	ブール値。 デフォルトは true。
Timeout	接続タイムアウト値を指定する。	整数値。
UID **User** **Username**	サーバへのログインに使用する名前を指定する。	文字列値。

新しいサンプルプログラム

Python 用 Adaptive Server Enterprise 拡張モジュールには、新しいサンプルがいくつか用意されています。

dsnconnect
dsn を使用してサーバに接続する方法を示します。

blk
バルクコピールーチンを使用して、データをサーバテーブルにコピーします。 次に、データを取得して表示します。

blkmany
バルクコピールーチンを使用して、データおよび複数のローを一度にコピーします。

blkiter
Python 反復プロトコルを使用してテーブルのローをバルクコピーアウトする方法を示します。

blktypes
各種 Python オブジェクト型 (デフォルト、NULL 値など) をバルクオペレーションの値として使用する方法を示します。

blklib サポート

blklib 機能は Python DB-API の拡張機能で、これにより、ローのバルクコピーを実行できます。 **blklib** 機能には、オブジェクトインタフェース、メソッド、および属性が含まれます。

BulkCursor オブジェクトコンストラクタ

データベースとの接続を確立する接続オブジェクトを提供する Python 拡張モジュールです。 接続オブジェクトには、バルクオペレーションのコンテキストを管理する新しい `BulkCursor` オブジェクトを作成するためのメソッドが含まれます。

バルクオペレーションで使用される接続をマーク付けするプロパティで構築された接続オブジェクトからのみ `BulkCursor` オブジェクトを作成できます。

使用法

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")cur = conn.cursor()
cur.execute("create table mytable (i int, c char(10))")
blk = conn.blkcursor()
```

close()
`BulkCursor` オブジェクトの **close()** メソッドはバルクオペレーションを終了します。 このメソッドを呼び出すと、バルクカーソルオブジェクトは使用できなくなります。 **close()** に引数はありません。

使用法

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")
blk = conn.blkcursor()
bblk.close()
```

copy()
`BulkCursor` オブジェクトの **copy()** メソッドはバルクオペレーションを開始します。

このメソッドは、次の引数を受け入れます。

- **tablename** - バルクオペレーション用のテーブルの名前を指定する文字列
- **direction** - *in* と *out* の値を持つキーワード引数

使用法

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")
cur = conn.cursor()
cur.execute("create table mytable (i int, c char(10))")
blk = conn.blkcursor()
blk.copy("mytable", direction="out")
```

done()

`BulkCursor` オブジェクトの **done()** メソッドはバルクオペレーションの完了をマーク付けします。 別のオペレーションを開始するには、**copy()** メソッドを呼び出します。

使用法

```
import sybpydb
conn = sybpydb.connect(dsn="user=sa;bulk=true")
cur = conn.cursor()
cur.execute("create table mytable (i int, c char(10)
blk = conn.blkcursor()
blk.copy("mytable", direction="in")
...
blk.done()
blk.copy("mytable", direction="out")
...
blk.done()
blk.close()
```

PHP 用 Adaptive Server Enterprise 拡張モジュール

PHP 用 Adaptive Server Enterprise 拡張モジュールが強化され、DSN スタイル接続プロパティがサポートされるようになりました。

DSN スタイル接続プロパティのサポート

sybase_connect() API と **sybase_pconnect()** API は DSN スタイル接続プロパティをサポートをします。

servername パラメータのみを使用して **sybase_connect()** API または **sybase_pconnect()** API を呼び出す場合、**servername** に有効な DSN (データソース名) 文字列が含まれている必要があります。 データソース名は、セミコロン (;) で区切られた名前=値の文字列です。それぞれの構成要素は次のとおりです。

1. Name - 等号 (=) またはセミコロン (;) で区切ることが可能な値。大文字と小文字が区別されます。 1 つの属性に複数の同義語がある場合があります。 たとえば、**server** と **servername** は同じ属性を指します。
2. 等号 (=) - Name に割り当てる値の開始を示します。 等号がない場合、名前は true 値のブール型とみなされます。

3. Value - セミコロン (;) で終了する文字列。 値内にセミコロンまたは別のバックスラッシュ (¥) がある場合は、バックスラッシュを使用します。 値は、ブール型、整数型、または文字列型で指定できます。 ブール型の有効な値は、true、false、on、off、1、0 です。

   注意： 値がないブール名がある場合、ブール型は true に設定する必要があります。

次に例を示します。

```
Username=name;Password=pwd;Timeout=10
```

有効な属性名と属性値

dsn キーワード引数に対する有効な属性名とその値を以下の表に示します。

名前	説明	値
ANSINull	SQL の等号 (=) または不等号 (!=) の比較で NULL 値のオペランドの評価が ANSI 標準に準拠しているかどうかを判別する。 値が true の場合、Adaptive Server は、*= NULL* と *is NULL* は同義ではないとする ANSI の動作を適用する。 標準の Transact-SQL では、*= NULL* と *is NULL* は同義とみなされる。 このオプションは、*<> NULL* と *is not NULL* の動作にも同様に作用する。	ブール値。 デフォルトは false。
BulkLogin	接続でバルクコピーオペレーションを実行できるかどうかを判別する。	ブール値。 デフォルトは false。
ChainXacts	true の場合、Adaptive Server は連鎖トランザクション動作を使用する。つまり、各サーバコマンドが個別のトランザクションとみなされる。 Adaptive Server は次の各文の前に、begin transaction を暗黙的に実行する。 **delete**、**fetch**、**insert**、**open**、**select** および **update**。 ただし、トランザクションを明示的に終了またはロールバックする必要がある。 false の場合、アプリケーションは begin transaction 文を commit 文または rollback 文と対で明示的に指定する必要がある。	ブール値。 デフォルトは false。
CharSet	この接続で使用する **charset** を指定する。	文字列値。

名前	説明	値
Confidentiality	接続でデータの暗号化サービスを実行するかどうか。	ブール値。 デフォルトは false。
CredentialDelegation	ユーザの委任クレデンシャルを使用してサーバを他のサーバに接続させるかどうかを判別する。	ブール値。 デフォルトは false。
DetectReplay	接続のセキュリティメカニズムがリプレイされた転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。
DetectOutOfSequence	接続のセキュリティメカニズムが不正なシーケンスの転送を検出するかどうかを判別する。	ブール値。 デフォルトは false。
Integrity	接続のセキュリティメカニズムがデータ整合性チェックを実行するかどうかを判別する。	ブール値。 デフォルトは false。
Interfaces	interfaces ファイルのパスと名前。	文字列値。
Keytab	接続のセキュリティメカニズムが *username* 値と一緒に使用されるセキュリティキーを読み込むファイルのパスと名前。	文字列値。 デフォルトは NULL。つまり、接続前にユーザがクレデンシャルを確立しておく必要がある。
Locale	メッセージ、データ型変換、日時フォーマットで使用する言語と文字セットを判別する。	文字列値。
Language	メッセージ、データ型変換、日時フォーマットで使用する言語セットを判別する。	文字列値。
LoginTimeout	ログインタイムアウト値を指定する。	整数値。
MaxConnect	あるコンテキストで同時にオープン可能な接続の最大数を指定する。	整数値。 デフォルト値は 25。 負の値および 0 は使用できない。
MutualAuthentication	サーバがクライアントに対して自身を認証する必要があるかどうかを判別する。	ブール値。 デフォルトは false。
NetworkAuthentication	接続のセキュリティメカニズムがネットワークベースのユーザ認証を実行するかどうかを判別する。	ブール値。 デフォルトは false。

名前	説明	値
PacketSize	TDS パケットサイズを指定する。	整数値。
Password	サーバへのログインに使用するパスワードを指定する。	文字列値。
PasswordEncryption	接続が非対称のパスワード暗号化を使用するかどうかを判別する。	ブール値。 デフォルトは false。
SecurityMechanism	接続のためのセキュリティサービスを実行するネットワークセキュリティメカニズムの名前を指定する。	文字列値。 デフォルト値はセキュリティドライバ設定によって異なる。
Server **Servername**	接続先のサーバの名前を指定する。	文字列値。
ServerPrincipalName	接続がオープンされるサーバのネットワークセキュリティプリンシパル名を指定する。	文字列値。 デフォルトは NULL。このとき接続は、サーバのプリンシパル名がその *ServerName* 値と同じであるとみなす。
Keepalive	KEEPALIVE オプションを使用するかどうかを判別する。	ブール値。 デフォルトは true。
Timeout	接続タイムアウト値を指定する。	整数値。
UID **User** **Username**	サーバへのログインに使用する名前を指定する。	文字列値。

dsnconnect.php サンプルプログラム

`dsnconnect.php` サンプルプログラムは、DSN 接続文字列を使用してサーバに接続します。 必要に応じて、サーバ名、ユーザアカウント、および現在のデータベースを出力します。

ESD #6 の新機能

ESD #6 には、Open Client 15.7 と Open Server 15.7 の新機能、Python 用 Adaptive Server Enterprise 拡張モジュール 15.7 と Perl 用 Adaptive Server Enterprise 拡張モジュール 15.7 のデータソース名 (DSN) 接続プロパティのサポートが導入されています。

Open Client 15.7 と Open Server 15.7 の機能

Open Client 15.7 および Open Server 15.7 の機能が強化され、`LOB` データ型、新しい SYBOCS_IFILE 環境変数、LDAP と SSL のバージョン、パラメータフォーマットの省略、Extended Plus の暗号化パスワード、および BCP --quoted-fname オプションを伴うバルクコピーインがサポートされるようになりました。

LOB データ型を指定したバルクコピーイン

ESD #6 では、**blk_textxfer()** API 呼び出しの後で **blk_rowxfer()** API 呼び出しを行えます。

以前のバージョンでは、**blk_textxfer()** API を使用して LOB カラムに転送用のマーク付けをし、ロー内とロー外の両方の値からなるデータベーステーブルに LOB データをコピーした場合、このデータ型の後続のカラムもすべて **blk_textxfer()** API を使用して転送用のマーク付けをする必要があり、**blk_rowxfer()** API は使用できませんでした。ESD#6 ではこの制限が削除されたため、**blk_textxfer()** API 呼び出しの後で **blk_rowxfer()** API 呼び出しが行えます。

新しい SYBOCS_IFILE 環境変数

デフォルトの `$SYBASE/interfaces` の代わりに SYBOCS_IFILE を使用して、interfaces ファイルの場所を指定します。

アプリケーションで CT-Library の CS_IFILE プロパティが設定されている場合は、そのプロパティ設定が優先されます。

LDAP バージョンと SSL バージョンのサポート

Sybase が提供する OpenLDAP ライブラリ (`libsybaseldap.so/dll`) では、LDAP サーバへの接続に OpenLDAP バージョン 2.4.31 および OpenSSL バージョン 1.0.1b を使用します。

パラメータフォーマットの省略

Open Client と Open Server では、Adaptive Server Enterprise の動的文のパラメータフォーマットの省略がサポートされるようになりました。

注意： ESD #3 から、Open Client ではパラメータフォーマットの省略をサポートしています。 しかし、ESD #6 ではパラメータフォーマットの省略のサポートを Open Server に導入します。

Open Server の Extended Plus 暗号化パスワードのサポート

クライアント接続が Extended Plus の暗号化パスワードをサポートする場合、Open Server は、パスワードの復号化を含むログインネゴシエーションを処理します。

ログインネゴシエーションは、SRV_CONNECT ハンドラが呼び出される前に実行されます。 SRV_CONNECT イベントハンドラでは、アプリケーションは既存の SRV_T_PWD プロパティでパスワードを取得し、新しいプロパティで使用されたパスワード暗号化プロトコルを検査するだけです。

Open Server のパスワード暗号化を試すには、`-X` オプション付きの **isql** を使用して "lang" サンプルに接続できます。このオプションは **isql** でパスワード暗号化を有効化します。

注意： リリース 15.0 から、Open Client はログインパスワードの暗号化をサポートしています。 しかし ESD#6 では、Open Server のログインパスワード暗号化を強化してサポートしています。

SRV_T_PWD
このプロパティは **srv_thread_props()** とともに使用してパスワードを取得します。クライアントが EPEP プロトコルをサポートしている場合、SRV_T_PWD は復号化されたパスワードを自動的に返します。

SRV_PWD_ENCRYPT_VERSION
Open Server の新しい public 列挙型で、次の値があります。

- SRV_NOENCRYPT_PWD (0)
- SRV_ENCRYPT_PWD (1) (Open Server では実装されていません)
- SRV_EXTENDED_ENCRYPT_PWD (2) (Open Server では実装されていません)
- SRV_EXTENDED_PLUS_ENCRYPT_PWD (3)

SRV_T_PWD_ENCRYPT_VERSION
この新しい読み込み専用プロパティを **srv_thread_props()** 関数とともに使用して、パスワードを取得したパスワード暗号化のプロトコルバージョンを取得します。

このプロパティのタイプおよび有効な値は、SRV_PWD_ENCRYPT_VERSION で説明します。

注意： パスワードのクリアテキスト送信を避けるためにこのプロパティを使用することはできません。 クライアントでサポートするパスワード暗号化バージョンを Open Server が読み込む際に、パスワードはすでにクリアテキストで送信されている場合があります。 しかし、このプロパティを使用して、すべてのクライアントアプリケーションが必要なパスワード暗号化アルゴリズムを使用していることを確認できます。

SRV_S_DISABLE_ENCRYPT
SRV_S_DISABLE_ENCRYPT プロパティを使用して、ネイティブパスワードネゴシエーションのサポートを無効化します。 このプロパティが設定された場合は、Open Server はパスワードネゴシエーションプロトコルを開始しません。 この SRV_S_DISABLE_ENCRYPT のデフォルト値は CS_FALSE です。

BCP --quoted-fname オプション

BCP の現在のコマンドラインパラメータの構文は ”--quoted-fname” です。

システムでは、文字列の間に空白を含まない文字列 ”quoted-fname” を受け取ります。 コマンドラインパラメータのリストにあるデータファイル名の後の任意の場所に新しいパラメータを配置できます。

このオプションを使用する以外に特殊文字を含むデータファイル名を使用するには、二重引用符でファイル名を囲み、各引用符の前にバックスラッシュを付けます (¥")。 ファイル名に二重引用符を含む場合は、ファイル名中の各二重引用符の前にバックスラッシュを付けます。

表 4 : 例

データファイル名	更新された構文を適用
fnamepart1,fnamepart2	¥” fnamepart1,fnamepart2¥”
fnamepart1”fnamepart2	¥” fnamepart1¥”fnamepart2¥”
“fnamepart1”fnamepart2”	¥”¥” fnamepart1¥”fnamepart2¥”¥”

Python 用 Adaptive Server Enterprise 拡張モジュール

Python 用 Adaptive Server Enterprise 拡張モジュールが強化され、DSN スタイル接続プロパティがサポートされるようになりました。

DSN スタイル接続プロパティのサポート

connect() メソッドは **dsn** という名前の新しいキーワード引数を受け入れます。

このキーワード引数は、接続情報を指定する文字列です。 **dsn** 文字列の構文は次のとおりです。

```
name1=value1;name2=value2;...
```

この name1 は、通常、接続プロパティまたはオプションに対応します。

名前文字列にはエスケープされた文字を含みません。 値文字列に等号およびセミコロンを表示するには、バックスラッシュをそれぞれの文字の前に付けてエスケープします。

Perl 用 Adaptive Server Enterprise 拡張モジュール

Perl 用 Adaptive Server Enterprise 拡張モジュールが強化され、新しい属性およびメソッド、新しい Perl データベースおよび文ハンドル属性、複数の文、動的 SQL、バインドパラメータ、ストアドプロシージャ、プライベートドライバメソッド、text と image のデータ処理、およびエラー処理がサポートされるようになりました。

DSN スタイル接続プロパティのサポート

ドライバは、接続時に特定の属性を設定するための DSN メカニズムを使用します。

DSN 属性構文は、Open Source DBD::Sybase ドライバと同じです。 したがって、Perl スクリプトを変更したり、DBD::Sybase と DBD::SybaseASE で異なるバージョンを保持する必要はありません。 ただし、DBD::SybaseASE は、無効とみなされている一部の属性をサポートしません。 「現在サポートされていない DSN 構文」を参照してください。

SybaseASE ドライバの connect 構文
dbi:SybaseASE: セクションによってドライバのパッケージ名が取得されるので、次の構文でロードできます。

```
DBI->connect("dbi:SybaseASE:attr=value;attr=value", $user_id,
$password, %attrib);
```

DSN がドライバに渡されると、この部分が削除され、残りの文字列には、解析対象のキーと値のペアが格納されます。

注意： *$user_id* と *$password* のクレデンシャルは、別々の API 引数であり、DSN 文字列の一部ではありません。

%attrib 引数は省略可能で、接続時にオプションを設定する一連のキーと値のペアをカンマで区切って指定します。 これらは、**connect()** 呼び出し中にドライバに渡され、処理されます。 例を示します。

```
DBI->connect("dbi:SybaseASE:server=mumbles; user, password,
PrintError => 1, AutoCommit = 0);
```

属性とメソッド

サーバへの接続時にサポートされる属性は以下のとおりです。

属性	説明
server	接続先のサーバの名前を指定する。 ドライバは現在、このオプションが設定されていることを前提としている。 server を指定しない場合、ENV{"DSQUERY"} メカニズムを使用してサーバ名を取得する。
database	接続時に、サーバ内のどのデータベースをターゲットデータベースにするかを指定する。 database を指定しない場合、マスタデータベースが使用される。
hostname	値セクションで、このプロセスの `sysprocesses` テーブルに格納されるホスト名を指定する。 hostname を指定しない場合、Perl アプリケーションが実行されるホストが使用される。
language	この接続で使用するロケールを指定する。 language を指定しない場合、CS_LC_ALL という内部デフォルトロケールが使用される。
charset	この接続で使用する charset を指定する。 charset を指定しない場合、内部デフォルトの **utf8** が使用される。

属性	説明
host; port	interfaces ファイルのエントリを利用する代わりに、使用するホストとポートの組み合わせを指定する。 **注意：** Perl DSN 構文では、host と port は別々のオプションである。 次のような代替 DSN 形式は現在サポートされていない。 `host:port=mumbles:1234` interface ファイルを使用せずに host と port の DSN オプションを指定する場合、host と port だけで接続できなければならない。 DSN 属性 “server=” を host と port の組み合わせとともに指定すると、接続は失敗する。 そのため、host と port を使用するか、server のみを使用して、接続を確立する必要がある。 2 つの DSN 属性 (server と host/port) は相互に排他的である。
timeout	接続タイムアウト値を指定する。 timeout を指定しない場合は、0 か負の値を設定する。
loginTimeout	ログインタイムアウト値を秒単位で指定する。 デフォルト値は 60 秒。 この属性を有効にするには、**loginTimeout=*value in seconds*** を設定する。
tds_keepalive	接続の KEEP_ALIVE 属性を指定する。 この属性を有効にするには、**tds_keepalive=*1*** を設定する。
packetSize	接続の TDS パケットサイズを指定する。 ドライバで設定される下限値はデフォルトで 2048。 最大値はサーバにより決定され、ドライバでは設定されない。
maxConnect	許可される接続の数を増減する。 有効な値の範囲は 1 ～ 128。デフォルトは 25。
encryptPassword	パスワード暗号化を使用するかどうかを指定する。 この属性を有効にするには、**encryptPassword=*1*** を設定する。
sslCAFile	`trusted.txt` ファイルの代替ロケーションを指定する。 最大 256 文字の絶対パスを指定する。
scriptName	アプリケーションを起動する最上位の Perl スクリプトの選択名を指定する。 この名前はアプリケーション名として `sysprocesses` テーブルに表示される。 この値を指定しない場合、Perl 内部環境から取得されたデフォルトのアプリケーション名が使用される。 この値には最大 256 文字を使用できる。 **注意：** SybaseASE ドライバに渡されるアプリケーション名は、DSN **scriptName** オプションを使用して設定されるか、Perl 内部環境から取得されます。
interfaces	Sybase interfaces ファイルの代替ロケーションを指定する。 **sslCAFile** オプションと **scriptName** オプションには同じ制約が適用される。

属性値は、ドライバが認識する限り何度でも繰り返せます。 不正な属性は **DBI->connect()** 呼び出しが失敗する原因になります。

注意： 属性名は Open Source Sybase Perl ドライバに準拠します。

DSN 固有の例:

```
$dbh = DBI->connect("dbi:SybaseASE:server=mumbles", $user, $passwd);
```

または、DSQUERY 環境変数を使用します。

```
my $srv = $ENV{"DSQUERY"};
$dbh = DBI->connect("dbi:SybaseASE:server=$srv",  $user, $passwd);
$dbh = DBI-
>connect("dbi:SybaseASE:host=tzedek.sybase.com;port=8100", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:maxConnect=100", $user, $passwd);
$dbh = DBI->connect("dbi:SybaseASE:database=sybsystemprocs", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:charset=iso_1", $user, $passwd);
$dbh = DBI->connect("dbi:SybaseASE:language=us_english", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:packetSize=8192", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:interfaces=/opt/sybase/
interfaces", $user, $passwd);
$dbh = DBI->connect("dbi:SybaseASE:loginTimeout=240", $user,
$passwd);
$dbh = DBI->connect("dbi:SybaseASE:timeout=240", $user, $passwd);
$dbh = DBI->connect("dbi:Sybase:scriptName=myScript", $user,
$password);
$dbh = DBI->connect("dbi:SybaseASE:hostname=pedigree", $user,
$password);
$dbh = DBI->connect("dbi:SybaseASE:encryptPassword=1", $user,
$password);
$dbh = DBI>connect("dbi:SybaseASE:sslCAFile=/usr/local/sybase/
trusted.txt", $user, $password,
AutoCommit => 1);
```

DSN 固有の組み合わせの例:

```
$dbh = DBI->connect("dbi:SybaseASE:server=mumbles,
database=tempdb;packetSize=8192;
language=us_english;charset=iso_1;encryptPassword=1", $user, $pwd,
AutoCommit=>1, PrintError => 0);
```

現在サポートされていない DSN 構文
以下の DSN 構文は現在サポートされていません。

- **tdsLevel**
- **kerberos**: 例

  ```
  $dbh = DBI->connect("dbi:SybaseASE:kerberos=$serverprincipal",
  '', '');
  ```

- **bulkLogin**: 例

```
$dbh = DBI->connect("dbi:SybaseASE:bulkLogin=1", $user,
$password);
```

- **serverType**

現時点でサポートされているデータベースハンドル属性

現時点でサポートされているデータベースハンドル属性を以下の表に示します。

属性	説明	デフォルト
dbh->{AutoCommit} = (0|1);	AutoCommit を無効または有効にする。	0 (オフ)
dbh->{LongTruncOK} = (0|1);	`text` 型および `image` 型のトランケーションを無効または有効にする。	0
dbh->{LongReadLen}=(int);	`text` データと `image` データのデフォルトの読み取りチャンクサイズを設定する。 例は次のとおり。 **dbh->{LongReadLen} = 64000.**	32767
dbh->{syb_show_sql} =(0|1);	この属性を設定すると、**$dbh->errstr** メカニズムによって返されるエラー文字列に現在の文が含まれる。	0
dbh->{syb_show_eed} = (0|1);	この属性を設定すると、**$dbh->errstr** によって返されるエラー文字列に拡張エラー情報が含まれる。	0
dbh->{syb_chained_txn} = (0|1);	この属性を設定すると、AutoCommit が無効のときに連鎖トランザクションが使用される。 この属性は **connect()** 呼び出し中にのみ使用する。 `$dbh = DBI->connect("dbi:SybaseASE:", $user, $pwd, {syb_chained_txn => 1});` AutoCommit が無効のときに **syb_chained_txn** を使用すると、現在のハンドルで強制コミットが行われる。 この属性を 0 に設定すると、必要に応じて、明示的な **BEGIN TRAN** が発行される。	0
dbh->{syb_use_bin_0x} = (0|1);	この属性を設定すると、結果文字列で `BINARY` 値と `VARBINARY` 値にプレフィックスとして '0x' が付く。	0
dbh->{syb_binary_images} = (0|1);	この属性を設定すると、`image` データがローバイナリ形式で返される。 設定しない場合、`image` データは 16 進文字列に変換される。	0

属性	説明	デフォルト
dbh->{syb_quoted_identifier} =(0|1);	引用符付き識別子として使用すれば、Sybase 予約語と競合する識別子を許可する。	0
dbh->{syb_rowcount}=(int);	ゼロ以外の値に設定すると、**SELECT** によって返されるロー、**UPDATE** 文または **DELETE** 文の影響を受けるローの数が *rowcount* 値に制限される。 設定を 0 に戻すと、制限が解除される。	0
dbh->{syb_flush_finish} = (0|1);	この属性を設定すると、ドライバは現在のコマンドに対して残っている結果を、実際にフェッチして排出する。 これは、ドライバによって発行される **ct_cancel()** コマンドの代わりに使用できる。	0
dbh->{syb_date_fmt} = datefmt string	このプライベートメソッドはデフォルトの日付変換と表示フォーマットを設定する。 「デフォルトの日付変換と表示フォーマット」を参照。	
dbh->{syb_err_handler}	エラーハンドラを実行するため、または通常のエラー処理を行う前に報告するために作成できる Perl サブルーチン。 特定の警告クラスに便利である。 「エラー処理」を参照。	0 (存在しない)
dbh->{syb_failed_db_fatal} = (0|1)	DSN に **database=*mumbles*** 属性/値のペアがあり、このデータベースが接続時に存在しない場合、**DBI->connect()** 呼び出しは失敗する。	0
dbh->{syb_no_child_con} =(0|1);	この属性を設定すると、ドライバは、**dbh** に対する複数のアクティブな文ハンドルを許可しない。 この場合、文の準備はできるが、別の文の準備が試行される前に実行して完了する必要がある。	0
dbh->{syb_cancel_request_on_error}=(0|1);	この属性を設定すると、複数文のセットを実行して、1 つの文が失敗した場合、**sth->execute()** は失敗する。	1 (オン)
dbh->{syb_bind_empty_string_as_null}= (0|1);	この属性を設定すると、NULLABLE カラム属性は、NULL 文字を表す空の文字列 (1 つのスペース) を返す。	0
dbh->{syb_disconnect_in_child} = (0|1);	閉じた接続を分岐をまたいで処理する。 子が無効になっている場合、DBI により接続は閉じられる。	0
dbh->{syb_enable_utf8} = (0|1);	この属性を設定すると、`UNICHAR`、`UNIVARCHAR`、および `UNITEXT` は **utf8** に変換される。	0

属性	説明	デフォルト
sth->syb_more_results} = (0|1);	「複数の結果セット」を参照。	
sth->{syb_result_type} = (0|1);	この属性を設定すると、記号による CS_ バージョンの代わりに数値による結果が返される。	0
sth->{syb_no_bind_blob} = (0|1);	この属性を設定すると、**sth->{fetch}** またはその他のバリエーションにおいて `image` カラムまたは `text` カラムが返されない。「text と image のデータ処理」を参照。	0
sth->{syb_do_proc_status} = (0|1);	**$sth->execute()** を強制的に実行し、SQL ストリームで実行されたストアドプロシージャのリターンステータスをフェッチする。 リターンステータスがゼロ以外の場合、**$sth->execute()** は `undef` を返す (つまり、失敗)。 この属性を設定しても、既存の文ハンドルには影響しない。ただし、この属性の設定後に作成された文ハンドルには影響する。 既存の **$sth** ハンドルの動作を取り消すには、次を実行する。**$sth->{syb_do_proc_status} = 0;**	0

サポートされていないデータベースハンドルオプション

以下のデータベースハンドルオプションはサポートされていません。

- **dbh->{syb_dynamic_supported}**
- **dbh->{syb_ocs_version}**
- **dbh->{syb_server_version}**
- **dbh->{syb_server_version_string}**
- **dbh->{syb_has_blk}**

注意： これらのオプションを使用する Perl スクリプトではエラーが生成されます。

Perl でサポートされているデータ型

Perl ドライバでは現在、文字列、数値、日付と時刻のデータ型がサポートされています。

文字列データ型	数値データ型	日付と時刻のデータ型
char	integer	datetime
varchar	smallint	date
binary	tinyint	time
varbinary	money	bigtime
text	smallmoney	bigdatetime
image	float	
unichar	real	
univarchar	double	
	numeric	
	decimal	
	bit	
	bigint	

注意： Perl は numeric データ型と decimal データ型を文字列として返します。 その他のデータ型はそれぞれのフォーマットで返します。

Sybase ASE ドライバが使用するデフォルトの時刻/日付のフォーマットはショートフォーマットです (例: Aug 7 2011 03:05PM)。

このフォーマットは C (デフォルト) ロケールに基づいています。 サポートされている他の日付/時刻のフォーマットについては、「デフォルトの日付変換と表示フォーマット」を参照してください。

複数文の使用

Adaptive Server は 1 つのバッチで複数文の SQL を処理できます。

次に例を示します。

```
my $sth = $dbh->prepare("
    insert into publishers (col1, col2, col3) values (10, 12, 14)
    insert into publishers (col1, col2, col3) values (1, 2, 4)
    insert into publishers (col1, col2, col3) values (11, 13, 15)
    ");
my $rc = $sth->execute();
```

これらの文のいずれかが失敗すると、**sth->execute()** は `undef` を返します。 **AutoCommit** がオンの場合、正常に完了した文によってデータがテーブルに挿入されることがあり、予想した結果になるとは限りません。

複数の結果セット
Perl ドライバでは、複数の文を 1 回の呼び出しで準備し、別の 1 回の呼び出しで実行できます。 たとえば、複数の select を含むストアドプロシージャを実行すると、複数の結果セットが返ります。

1 つの呼び出しで準備された複数の文の結果は、単一のデータストリームとしてクライアントに返されます。 個々の結果セットは通常の単一の結果セットとして処理されます。つまり、文ハンドルの **fetch()** メソッドが各セットの最後に `undef` を返します。

CT-Lib API **ct_fetch()** は `CS_END_RESULTS` を返し、ドライバが最後のローを取得した後でこれを `undef` に変換します。

ドライバにより、アプリケーションは **sth->{syb_result_type}** をチェックして結果セットを取得できます。 その後、**sth->{syb_more_results}** 文ハンドル属性を使用して、返される結果セットがほかにもまだあるかどうかを確認することができます。 **sth->{syb_results_type}** により返される (数) 値は次のいずれかです。

- `CS_MSG_RESULT`
- `CS_PARAM_RESULT`
- `CS_STATUS_RESULT`
- `CS_COMPUTE_RESULT`
- `CS_ROW_RESULT`

複数の結果セットの例:

```
do {
    while($a = $sth->fetch) {
    ..for example, display data..
    }
} while($sth->{syb_more_results});
```

複数の結果セットが想定される場合は、これを使用することをおすすめします。

注意： Perl ドライバは現在、**ct_cursor()** API を使用したカーソルをサポートしていません。 したがって、ドライバは CS_CURSOR_RESULT を報告しません。

DatabaseHandle (dbh) の複数のアクティブ文
$dbh にアクティブな文ハンドルがすでにある場合、**$dbh->prepare()** メソッドで新しい接続を開くことにより、1 つのデータベースハンドルで複数の文をアクティブにすることができます。

dbh->{syb_no_child_con} 属性は、この機能のオンとオフを制御します。 デフォルトでは、DatabaseHandle はオフです。これは、複数の文ハンドルがサポートされ

ることを示します。 オンの場合、同じデータベースハンドル上で複数の文を使用することはできません。

注意： AutoCommit がオフの場合、 1 つの **$dbh** 上の複数の文ハンドルはサポートされません。 これにより、デッドロックの問題が発生するのを防ぐことができます。 また、複数の文ハンドルを同時に使用すると、複数の物理接続が使用されることになるので、トランザクションの整合性を確保できません。

サポートされている文字の長さ

さまざまなタイプの識別子でサポートされている文字の長さについて説明します。

テーブルやカラムなどの Sybase 識別子の名前の長さは、255 文字を超えてもかまいません。

TDS プロトコルの制限を受けるログイン、アプリケーション名、パスワードの長さは、30 文字を超えることはできません。

ロケールと文字セットの設定

DSN 属性 **charset** と **language** を使用して、Perl ドライバの CT-Library ロケールおよび文字セットを設定できます。

ドライバのデフォルトの文字セットは *UTF8*、デフォルトのロケールは *CS_LC_ALL* です。

動的 SQL のサポート、プレースホルダ、バインドパラメータ

Perl ドライバは、パラメータの使用など、動的な SQL をサポートします。

次に例を示します。

```
$sth = $dbh->prepare("select * from employee where empno = ?");

# Retrieve rows from employee where empno = 1024:
$sth->execute(1024);
while($data = $sth->fetch) {
    print "@$data¥n";
}
# Now get rows where empno = 2000:
$sth->execute(2000);
while($data = $sth->fetch) {
    print "@$data¥n";
}
```

注意： Perl ドライバは、'?' スタイルのパラメータをサポートしますが、':1' プレースホルダタイプをサポートしません。 プレースホルダを使用して `text` データ型および `image` データ型をバインドすることはできません。

DBD::SybaseASE は、**prepare()** メソッドに Open Client **ct_dynamic()** ファミリの API を使用します。 "?" スタイルのプレースホルダの制約と一般的な動的 SQL の使用方法については、『Sybase Open Client C プログラマーズガイド』を参照してください。

動的 SQL のサポートを示す別の例:

```
my $rc;
my $dbh;
my $sth;


# call do() method to execute a SQL statement.
#
$rc = $dbh->do("create table tt(string1 varchar(20), date datetime,
    val1 float, val2 numeric(7,2))");


$sth = $dbh->prepare("insert tt values(?, ?, ?, ?)");
$rc = $sth->execute("test12", "Jan 3 2012", 123.4, 222.33);


# alternate way, call bind_param() then execute without values in the
# execute statement.
$rc = $sth->bind_param(1, "another test");
$rc = $sth->bind_param(2, "Jan 25 2012");
$rc = $sth->bind_param(3, 444512.4);
$rc = $sth->bind_param(4, 2);
$rc = $sth->execute();

# and another execute, with args.....
$rc = $sth->execute("test", "Feb 30 2012", 123.4, 222.3334);
```

注意： 最後の文では、日付が無効なので、拡張エラー情報 (EED) がスローされます。 Perl スクリプトで、Adaptive Server エラーメッセージを **dbh->errstr** に書き込む前に **dbh->{syb_show_eed} =*1*** と設定します。

"?" スタイルのプレースホルダを示す別の例:

```
$sth = $dbh->prepare("select * from tt where date > ? and val1 > ?");
$rc = $sth->execute('Jan 1 2012', 120);

# go home....
$dbh->disconnect;
exit(0);
```

ストアドプロシージャによるプレースホルダサポート

Perl 用 Adaptive Server Enterprise データベースドライバは、入出力両方のパラメータを持つストアドプロシージャをサポートします。

ストアドプロシージャは、他の Transact-SQL 文と同じように処理されます。 ただし、Sybase ストアドプロシージャは、ストアドプロシージャコード内の return 文に対応するリターンステータスを含む追加の結果セットを返します。 数値 4043 の CS_STATUS_RESULT という名前のこの追加結果セットは単一ローであり、常に最後に返されます。

ドライバは、特殊な属性 **$sth->{syb_do_proc_status}** を使用してストアドプロシージャを処理できます。 この属性が設定されている場合、ドライバは追加結果セットを処理して、**$sth->{syb_proc_status}** にリターンステータス値を配置します。 結果セットが 0 以外の値の場合、エラーが生成されます。

例

```
$sth = $dbh->prepare("exec my_proc ¥@p1 = ?, ¥@p2 = ?");
    $sth->execute('one', 'two');
```

この例は、位置を指定するパラメータの使い方を示します。

```
$sth = $dbh->prepare("exec my_proc ?, ?");
  $sth->execute('one', 'two');
```

位置を指定するパラメータと名前付きパラメータを同じ prepare 文内に混在させることはできません。たとえば、次の文は最初のパラメータで失敗します。

```
$sth = $dbh->prepare("exec my_proc ¥@p1 = 1, ¥@p2 = ?");
```

ストアドプロシージャが出力パラメータを使用してデータを返す場合、まずそのパラメータを宣言しておくことが必要です。

```
$sth = $dbh->prepare(qq[declare @name varchar(50) exec getname abcd,
@name output]);
```

次の文のように、バインドされたパラメータを持つストアドプロシージャを呼び出すことはできません。

```
$sth = $dbh->prepare("exec my_proc ?");
  $sth->execute('foo');
```

次の文は機能します。

```
$sth = $dbh->prepare("exec my_proc 'foo'");
  $sth->execute('foo');
```

通常、ストアドプロシージャは複数の結果セットを返すので、syb_more_results が 0 になるまでループを使用してください。

```
do {
    while($data = $sth->fetch) {
```

```
        do something useful...
    }
 } while($sth->{syb_more_results});
```

パラメータの例

```
declare @id_value int, @id_name char(10)
    exec my_proc @name = 'a_string', @number = 1234,
        @id = @id_value OUTPUT, @out_name = @id_name OUTPUT
```

ストアドプロシージャが OUTPUT パラメータのみを返す場合、次の文を使用できます。

```
$sth = $dbh->prepare('select * .......');
    $sth->execute();
@results = $sth->syb_output_params();  #  this method is available in
SybaseASE.pm
```

これは、プロシージャコールですべての OUTPUT パラメータの配列を返し、他の結果を無視します。 OUTPUT パラメータがない場合、またはストアドプロシージャが失敗した場合、この配列は未定義になります。

一般的な例

```
$sth = $dbh->prepare("declare ¥@id_value int, ¥@id_name
        OUTPUT, @out_name = @id_name OUTPUT");
    $sth->execute();
    {
        while($d = $sth->fetch) {
            # 4042 is CS_PARAMS_RESULT
        if ($sth->{syb_result_type} == 4042) {
            $id_value  = $d->[0];
            $id_name   = $d->[1];
        }
    }
    redo if $sth->{syb_more_results};
}
```

OUTPUT パラメータは、特殊な結果セットでは 1 つのローとして返されます。

パラメータタイプ

ドライバは、パラメータごとに正しいパラメータタイプを確認するわけではありません。 全パラメータのデフォルト値は、**bind_param()** を使用して、サポートされているバインド型が設定されていない限り、ODBC スタイルの SQL_CHAR 値になります。

ドライバは次の ODBC スタイルのバインド型をサポートします。

- SQL_CHAR
- SQL_VARCHAR
- SQL_VARBINARY
- SQL_LONGVARCHAR

- SQL_LONGVARBINARY
- SQL_BINARY
- SQL_DATETIME
- SQL_DATE
- SQL_TIME
- SQL_TIMESTAMP
- SQL_BIT
- SQL_TINYINT
- SQL_SMALLINT
- SQL_INTEGER
- SQL_REAL
- SQL_FLOAT
- SQL_DECIMAL
- SQL_NUMERIC
- SQL_BIGINT
- SQL_WCHAR
- SQL_WLONGVARCHAR

ODBC 型は、対応する Adaptive Server データ型にドライバ内でマップされます。Sybase Adaptive Server Enterprise ODBC ドライバの『ユーザーズガイド 15.7』を参照してください。

特定の Adaptive Server でサポートされているデータ型の完全なリストを取得するには、ストアドプロシージャ **sp_datatype_info** を実行します。 次に例を示します。

```
$sth = $dbh->prepare("exec my_proc ¥@p1 = ?, ¥@p2 = ?");
    $sth->bind_param(1, 'one', SQL_CHAR);
    $sth->bind_param(2, 2.34, SQL_FLOAT);
    $sth->execute;
    ....
    $sth->execute('two', 3.456);
    etc...
```

注意： パラメータのカラムタイプを設定した後、文ハンドルを解放して再度実行しない限り、変更はできません。 SQL_NUMERIC データまたは SQL_DECIMAL データをバインドすると、総桁数または小数点以下桁数がターゲットパラメータ定義のサイズを超えた場合に、致命的な変換エラーが発生することがあります。

たとえば、ストアドプロシージャが次のように定義されているとします。

```
declare proc my_proc @p1 numeric(5,2) as...
    $sth = $dbh->prepare("exec my_proc ¥@p1 = ?");
    $sth->bind_param(1, 3.456, SQL_NUMERIC);
```

この場合、次のエラーが発生します。

```
DBD::SybaseASE::st execute failed: Server message number=241
severity=16 state=2 line=0 procedure=my_proc text=Scale error
```

```
during implicit conversion of NUMERIC value '3.456' to a
NUMERIC field.
```

これらのエラーを無視するには、**arithabort** オプションを次のように設定します。

```
$dbh->do("set arithabort off");
```

Adaptive Server のリファレンスマニュアルを参照してください。

サポートされているプライベートドライバメソッド

dbh->syb_isdead() は、接続の状態を表す true または false を返します。 false 戻り値は、特定のクラスまたは接続エラーを示します。つまり、接続の失敗を意味します。

$sth->syb_describe() は、現在の結果セットの各出力カラムの記述を含む配列を返します。 配列の各要素は、カラムを記述するハッシュの参照です。

次の例のように、NAME、TYPE、SYBTYPE、SYBMAXLENGTH、MAXLENGTH、SCALE、PRECISION、STATUS などの記述フィールドを設定できます。

```
$sth = $dbh->prepare("select name, uid from sysusers");
    $sth->execute;
    my @description = $sth->syb_describe;
    print "$description[0]->{NAME}¥n";                  # prints name
    print "$description[0]->{MAXLENGTH}¥n";    # prints 30
    etc, etc.
     ....
    while(my $row = $sth->fetch) {
    ....
}
```

注意： STATUS フィールドは、CS_CANBENULL、CS_HIDDEN、CS_IDENTITY、CS_KEY、CS_VERSION_KEY、CS_TIMESTAMP、CS_UPDATABLE、CS_UPDATECOL、CS_RETURN の値に対してテストできる文字列です。

Open Client のマニュアルを参照してください。

デフォルトの日付変換と表示フォーマット

syb_data_fmt() プライベートメソッドを使用して、独自のデフォルトの日付変換と表示フォーマットを設定できます。

Sybase の日付フォーマットは、クライアントのロケール設定によって異なります。 デフォルトの日付フォーマットは C ロケールに基づきます (例: Feb 16 2012 12:07PM)。

この同じデフォルトロケールで、次の追加入力フォーマットもサポートされます。

- 2/16/2012 12:07PM
- 2012/02/16 12:07
- 2012-02-16 12:07
- 20120216 12:07

日付入出力フォーマットを変更するには、引数が文字列の **dbh->{syb_date_fmt}** を使用します。

表 5 : サポートされている日付/時刻フォーマット

日付フォーマット	例
LONG	Nov 15 2011 11:30:11:496AM
SHORT	Nov 15 2011 11:30AM
DMY4_YYYY	Nov 15 2011
MDY1_YYYY	11/15/2011
DMY1_YYYY	15/11/2011
DMY2_YYYY	15.11.2011
DMY3_YYYY	15-11-2011
DMY4_YYYY	15 November 2011
HMS	11:30:11 AM
LONGMS	Nov 15 2011 11:30:33.532315PM

Perl 用 Adaptive Server Enterprise データベースドライバは、バージョン 15.7 までサポートしてきた日時の値のすべてをサポートします。

text と image のデータ処理

Perl 用 Adaptive Server Enterprise データベースドライバは、LONG/BLOB データの image 型と `text` 型をサポートします。 各データ型において、2GB のバイナリデータまで格納可能です。

text/image データのデフォルトサイズ制限は 32KB です。 この制限を変更するには **LongReadLen** 属性を使用します。これは、**fetch()** API の呼び出しにより設定されます。

text データまたは image データを挿入するためにバインドパラメータは使用できません。

標準 SQL を使用すると、image データは通常 16 進文字列に変換されますが、**syb_binary_images** ハンドル属性を使用してこの動作を変更できます。 また、

$binary = pack("H*", $hex_string); のような Perl 関数を使用してこの変換を実行することもできます。

DBI には BLOB スタイルの (`text/image`) データ型を処理するための API サポートがないため、`SybaseASE.pm` ファイルに格納されている関数セットをインストールし、アプリケーションレベルの Perl コードで使用して Open Client **ct_get_data()** スタイルの呼び出しを行うことができます。 **syb_ct_get_data()** と **syb_ct_send_data()** の呼び出しは、Adaptive Server 間で `text` データと `image` データを転送する Open Client 関数のラッパーです。

例

```
$sth->syb_ct_get_data($col, $dataref, $numbytes);
```

syb_ct_get_data() の呼び出しを使用して、image/text データをロー形式で 1 つずつ、またはまとめてフェッチすることができます。 この呼び出しを有効にするには、**dbh->{syb_no_bind_blob}** 文ハンドルを *1* に設定します。

syb_ct_get_data() の呼び出しは、クエリのカラム番号 (1 から始まる)、スカラ参照、およびバイト数の引数を受け取ります。 バイト数が 0 の場合、できるだけ多くのバイトが読み込まれます。 この呼び出しが機能するには、image/text カラムが select リストの最後に位置する必要があります。

呼び出しシーケンスは次のとおりです。

```
$sth = $dbh->prepare("select id, img from a_table where id = 1");
    $sth->{syb_no_bind_blob} = 1;
    $sth->execute;
    while($d = $sth->fetchrow_arrayref) {
        # The data is in the second column
        $len = $sth->syb_ct_get_data(2, ¥$img, 0);
}
```

syb_ct_get_data() は、フェッチされたバイト数を返します。データをまとめてフェッチする場合は、次の文を使用できます。

```
while(1) {
$len = $sth->syb_ct_get_data(2, $imgchunk, 1024);
... do something with the $imgchunk ...
 last if $len != 1024;
}
```

その他の TEXT/IMAGE API

syb_ct_data_info() API は、更新する image/text データ項目の CS_IODESC 構造体をフェッチまたは更新します。

次に例を示します。

```
$stat = syb_ct_data_info($action, $column, $attr)
```

- *$action* - CS_SET または CS_GET。
- *$column* - アクティブな select 文のカラム番号 (CS_SET オペレーションでは無視)。
- *$attr* - 構造体に値を設定するハッシュ参照。

最初に CS_GET を指定して **syb_ct_data_info()** を呼び出し、更新する image/text データ項目の CS_IODESC 構造体をフェッチする必要があります。 次に、**total_txtlen** 構造体要素の値を、挿入する image/text データの長さ (バイト単位) に更新します。 **log_on_update** を true に設定して、オペレーションの完全なロギングを有効にします。

CS_IODESC がフェッチされる image/text データが NULL である場合、CS_GET を指定して **syb_ct_data_info()** を呼び出すと失敗します。 CS_IODESC エントリを取得する前に、標準 SQL を使用して NULL 値を非 NULL 値 (たとえば、空の文字列) に更新します。

この例では、id カラムが 1 である image カラム内のデータを更新するとします。

1. このデータに対して CS_IODESC データを検出します。

```
$sth = $dbh->prepare("select img from imgtable where id = 1");
        $sth->execute;
    while($sth->fetch) {    # don't care about the data!
        $sth->syb_ct_data_info('CS_GET', 1);
    }
```

2. CS_IODESC 値で更新します。

```
$sth->syb_ct_prepare_send();
```

3. 挿入する新しいデータ項目のサイズを設定し、オペレーションのログが記録されないようにします。

```
$sth->syb_ct_data_info('CS_SET', 1, {total_txtlen
=> length($image), log_on_update => 0});
```

4. 次の文を使用して、1 つのまとまりとしてデータを転送します。

```
$sth->syb_ct_send_data($image, length($image));
```

5. 次の文を使用して、オペレーションをコミットします。

```
$sth->syb_ct_finish_send();
```

エラー処理

Perl および CT-Lib 用 Adaptive Server データベースドライバで発生したエラーはすべて、DBI レイヤに伝達されます。

例外には、ドライバの起動中、使用可能なコンテキストがまだない場合に報告する必要があるエラーや警告があります。

PrintError 属性が有効な場合、DBI レイヤは基本的なエラー報告を行います。 プログラムまたはシステムレベルの問題を追跡するには、DBI トレースメソッドを使用して DBI オペレーションのトレースを有効にします。

詳細なエラーメッセージ (サーバメッセージ) を追加する例は次のとおりです。

- アクティブな **dbh** で **dbh->{syb_show_sql} =*1*** を設定して、**$dbh->errstr** により返される文字列に現在の SQL 文を追加します。
- アクティブな **dbh** で **dbh->{syb_show_eed} =*1*** を設定して、**$dbh->errstr** により返される文字列に重複挿入エラーや無効な日付フォーマットなどの拡張エラー情報 (EED) を追加します。
- **syb_err_handler** 属性を使用して、標準エラーハンドラが処理を実行する前に呼び出される特定のエラーハンドラのコールバック (Perl サブルーチン) を設定します。 このサブルーチンが 0 を返す場合、エラーは無視されます。 これは、Transact-SQL の **PRINT** 文、**showplan** 出力、および **dbcc** 出力を処理する場合に便利です。
  このサブルーチンは、Sybase エラー番号、重大度、ステータス、SQL バッチの行番号、サーバ名 (存在する場合)、ストアドプロシージャ名 (存在する場合)、メッセージテキスト、SQL テキスト、およびタイプを表す文字列 "client" または "server" を含むパラメータを使用して呼び出されます。

セキュリティサービスの設定

`ocs.cfg` ファイルおよび `libtcl.cfg` ファイルを使用して、セキュリティオプションを設定します。

1. 接続については、`ocs.cfg` を使用してディレクトリとセキュリティのプロパティを設定します。

   注意： `ocs.cfg` ファイルで、ドライバ固有のオプションを設定できるように、アプリケーション名のエントリを追加します。
2. セキュリティサービスドライバとディレクトリサービスドライバをロードするように `libtcl.cfg` を編集します。
3. パスワードを暗号化するには、**encryptPassword** DSN オプションを使用します。 例:

```
DBI-
>connect("dbi:SybaseASE:server=mumbles;encryptPassword
=1", $user, $pwd);
```

例

サンプルプログラムを使用して、ストアドプロシージャの基本的な使用法を確認するとともに、pubs2 authors テーブルからローを取得します。

例 1

サンプルプログラムを使用して、Perl でのストアドプロシージャの基本的な使用法を確認します。

このプログラムは、サーバに接続し、2 つのストアドプロシージャを作成し、prepare を呼び出し、プロシージャをバインドまたは実行し、結果を STDOUT に出力した後、切断し、プログラムを終了します。

```
use strict;

use DBI qw(:sql_types);
use DBD::SybaseASE;

require_version DBI 1.51;

my $uid = "sa";
my $pwd = "";
my $srv = $ENV{"DSQUERY"} || die 'DSQUERY appears not set';
my $dbase = "tempdb";

my $dbh;
my $sth;
my $rc;

my $col1;
my $col2;
my $col3;
my $col4;

# Connect to the target server.
#
$dbh = DBI->connect("dbi:SybaseASE:server=$srv;database=$dbase",
    $uid, $pwd, {PrintError => 1});

# One way to exit if things fail.
#
if(!$dbh) {
    warn "Connection failed, check if your credentials are set
correctly?¥n";
    exit(0);
}

# Ignore errors on scale for numeric. There is one marked call below
# that will trigger a scale error in ASE. Current settings suppress
# this.
#
$dbh->do("set arithabort off")
```

```
        || die "ASE response not as expected";

# Drop the stored procedures in case they linger in ASE.
#
$dbh->do("if object_id('my_test_proc') != NULL drop proc
my_test_proc")
    || die "Error processing dropping of an object";

$dbh->do("if object_id('my_test_proc_2') != NULL drop proc
my_test_proc_2")
    || die "Error processing dropping of an object";

# Create a stored procedure on the fly for this example. This one
# takes input args and echo's them back.
#
$dbh->do(qq{
create proc my_test_proc ¥@col_one varchar(25), ¥@col_two int,
    ¥@col_three numeric(5,2), ¥@col_four date
as
    select ¥@col_one, ¥@col_two, ¥@col_three, ¥@col_four
}) || die "Could not create proc";

# Create another stored procedure on the fly for this example.
# This one takes dumps the pubs2..authors table. Note that the
# format used for printing is defined such that only four columns
#appear in the output list.
#
$dbh->do(qq{
create proc my_test_proc_2
as
    select * from pubs2..authors
}) || die "Could not create proc_2";

# Call a prepare stmt on the first proc.
#
$sth = $dbh->prepare("exec my_test_proc ¥@col_one = ?, ¥@col_two
= ?,
    ¥@col_three = ?, ¥@col_four = ?")
        || die "Prepare exec my_test_proc failed";

# Bind values to the columns. If SQL type is not given the default
# is SQL_CHAR. Param 3 gives scale errors if arithabort is disabled.
#
$sth->bind_param(1, "a_string");
$sth->bind_param(2, 2, SQL_INTEGER);
$sth->bind_param(3, 1.5411111, SQL_DECIMAL);
$sth->bind_param(4, "jan 12 2012", SQL_DATETIME);

# Execute the first proc.
#
$rc = $sth->execute || die "Could not execute my_test_proc";

# Print the bound args
#
```

```
dump_info($sth);

# Execute again, using different params.
#
$rc = $sth->execute("one_string", 25, 333.2, "jan 1 2012")
    || die "Could not execute my_test_proc";

dump_info($sth);

# Enable retrieving the proc status.
$sth->{syb_do_proc_status} = 1;

$rc = $sth->execute(undef, 0, 3.12345, "jan 2 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute("raisin", 1, 1.78, "jan 3 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute(undef, 0, 3.2233, "jan 4 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute(undef, 0, 3.2234, "jan 5 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute("raisin_2", 1, 3.2235, "jan 6 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

$rc = $sth->execute(undef, 0, 3.2236, "jan 7 2012")
    || die "Could not execute my_test_proc";
dump_info($sth);

# End of part one, generate blank line.
#
print "¥n";

# Undef the handles (not really needed but...).
#
undef $sth;
undef $rc;

# Prepare the second stored proc.
#
$sth = $dbh->prepare("exec my_test_proc_2")
        || die "Prepare exec my_test_proc_2 failed";

# Execute and print
#
$rc = $sth->execute || die "Could not execute my_test_proc_2";
dump_info($sth);
```

```
#
# An example of a display/print function.
#
sub dump_info {
    my $sth = shift;
    my @display;

    do {
    while(@display = $sth->fetchrow) {
     foreach (@display) {
        $_ = '' unless defined $_;
    }
     $col1 = $display[0];
     $col2 = $display[1];
     $col3 = $display[2];
     $col4 = $display[3];

    # Proc status is suppressed, assume proc
    # execution was always successful. Enable
    # by changing the write statement.
    #
    #write;
    write unless $col1 eq 0;
    }
} while($sth->{syb_more_results});
}

#
# The FORMAT template for this example.
#
format STDOUT_TOP =

Column1            Column2            Column3            Column4
------             ------             ------             ------
.

# Treat all data as left-justified strings
#
format STDOUT =
@<<<<<<<<<<<<   @<<<<<<<<<<<< @<<<<<<<<<<<<
@<<<<<<<<<<<<
$col1, $col2, $col3, $col4
.

# The End.....
#
$dbh->do("drop proc my_test_proc");
$dbh->do("drop proc my_test_proc_2");
$dbh->disconnect;
```

例 2

サンプルプログラムを使用して、pubs2 authors テーブルからローを取得し、tempdb に挿入して、バッチ挿入用の新しいローを追加します。 さらに、更新された authors テーブルを STDOUT に出力した後、切断し、終了します。

```
use strict;

use DBI ();
use DBD::SybaseASE ();

require_version DBI 1.51;

# trace(n) where n ranges from 0 - 15.
# use 2 for sufficient detail.
#DBI->trace(2); # 0 - 15, use 2 for sufficient detail

# Login credentials, handles and other variables.
#
my $uid = "sa";
my $pwd = "";
my $srv = $ENV{"DSQUERY"} || die 'DSQUERY appears not set';
my $dbase = "tempdb";
my $temp_table = "$dbase..authors";

my $rows;
my $col1;
my $col2;
my $dbh;
my $sth;
my $rc;

# Connect to the target server:
#
$dbh = DBI->connect("dbi:SybaseASE:server=$srv;database=$dbase",
    $uid, $pwd, {PrintError => 0, AutoCommit => 0})
    || die "Connect failed, did you set correct credentials?";

# Switch to the pubs2 database.
#
$rc = $dbh->do("use pubs2") || die "Could not change to pubs2";

# Retrieve 2 columns from pubs2..authors table.
#
$sth = $dbh->prepare(
    "select au_lname, city from authors where state = 'CA'")
    || die "Prepare select on authors table failed";

$rc = $sth->execute
    || die "Execution of first select statement failed";

# We may have rows now, present them.
#
$rows = dump_info($sth);
```

```
print "¥nTotal # rows: $rows¥n¥n";

# Switch back to tempdb, we take a copy of pubs2..authors
# and insert some rows and present these.
#
$rc = $dbh->do("use $dbase") || die "Could not change to $dbase";

# Drop the authors table in tempdb if present
#
$rc = $dbh->do("if object_id('$temp_table') != NULL drop table
$temp_table")
    || die "Could not drop $temp_table";

# No need to create a tempdb..authors table as the select into will
# do that.

$rc = $dbh->do("select * into $temp_table from pubs2..authors")
    || die "Could not select into table $temp_table";

# Example of a batch insert...
#
$sth = $dbh->prepare("
    insert into $temp_table
    (au_id, au_lname, au_fname, phone, address, city, state,
        country, postalcode) values
    ('172-39-1177', 'Simpson', 'John', '408 496-7223',
        '10936 Bigger Rd.', 'Menlo Park', 'CA', 'USA', '94025')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode) values
('212-49-4921', 'Greener', 'Morgen', '510 986-7020',
    '309 63rd St. #411', 'Oakland', 'CA', 'USA', '94618')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('238-95-4766', 'Karson', 'Chernobyl', '510 548-7723',
    '589 Darwin Ln.', 'Berkeley', 'CA', 'USA', '94705')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('267-41-4394', 'OLeary', 'Mich', '408 286-2428',
    '22 Cleveland Av. #14', 'San Jose', 'CA', 'USA', '95128')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('274-80-4396', 'Straight', 'Shooter', '510 834-2919',
    '5420 College Av.', 'Oakland', 'CA', 'USA', '94609')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
```

```
('345-22-1785', 'Smiths', 'Neanderthaler', '913 843-0462',
    '15 Mississippi Dr.', 'Lawrence', 'KS', 'USA', '66044')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('405-56-7012', 'Bennetson', 'Abra', '510 658-9932',
    '6223 Bateman St.', 'Berkeley', 'CA', 'USA', '94705')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('427-17-2567', 'Dullest', 'Annie', '620 836-7128',
    '3410 Blonde St.', 'Palo Alto', 'CA', 'USA', '94301')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('527-72-3246', 'Greene', 'Mstar', '615 297-2723',
    '22 Graybar House Rd.', 'Nashville', 'TN', 'USA', '37215')

insert into $temp_table
(au_id, au_lname, au_fname, phone, address, city, state,
    country, postalcode ) values
('672-91-3249', 'Yapan', 'Okiko', '925 935-4228',
    '3305 Silver Ct.', 'Walnut Creek', 'CA', 'USA', '94595')
");

$rc = $sth->execute || die "Could not insert row";

# Retrieve 2 columns from tempdb..authors table and present these
#
$sth = $dbh->prepare(
    "select au_lname, city from $temp_table where state = 'CA'")
    || die "Prepare select on $temp_table table failed";

$rc = $sth->execute
    || die "Execution of second select statement failed";

# Output
#
$rows = dump_info($sth);
print "¥nTotal # rows: $rows";
print "¥n";

sub dump_info {
    my $sth = shift;
    my @display;
    my $rows = 0;

while(@display = $sth->fetchrow) {
    $rows++;
    foreach (@display) {
    $_ = '' unless defined $_;
    }
```

```
    $col1 = $display[0];
    $col2 = $display[1];
    write;
    }
    $rows;
}

# The FORMAT template for this example.
#
format STDOUT_TOP =

Lastname            City
--------            -------
.

format STDOUT =

@<<<<<<<<<<<<    @<<<<<<<<<<<<
$col1, $col2
.

$dbh->disconnect;
```

ESD #5 の新機能

ESD #5 には、jConnect 7.07、Adaptive Server ODBC ドライバ 15.7、および Adaptive Server ADO.NET Data Provider 15.7 の新機能が導入されています。

Adaptive Server ADO.NET Data Provider の COMPUTE 句を使用する Transact-SQL クエリのサポート

Adaptive Server ADO.NET Data Provider では、**COMPUTE** 句を含む Transact-SQL クエリがサポートされるようになりました。

COMPUTE 句を使用すると、単一の **select** 文にディテールと計算結果を組み込むことができます。 計算ローは、次に示すように特定グループのディテールローの後に表示されます。

```
select type, price, advance from titles order by type compute
sum(price), sum(advance) by type
```

```
type           price       advance
------------   ---------   ----------
UNDECIDED      NULL        NULL
Compute Result:
---------------------- -------------------------
NULL                   NULL
type           price       advance
------------   ---------   ----------
business       2.99        10,125.00
business       11.95       5,000.00
business       19.99       5,000.00
business       19.99       5,000.00
Compute Result:
---------------------- -------------------------
54.92                  25,125.00
...
...

(24 rows affected)
```

Adaptive Server ADO.NET Data Provider が **COMPUTE** 句を含む **select** 文を実行すると、複数の結果セットがクライアントに返されます。 結果セットの数は、使用できるユニークなグループの数によって異なります。 各グループには、ディテールローの結果セットが 1 つと計算結果の結果セットが 1 つ組み込まれます。 クライアントが返されたローを完全に処理するには、すべての結果セットを処理する必要があります。このように処理しない場合は、最初のデータグループのディテールローのみが、最初に返される結果セットに組み込まれます。

COMPUTE 句の詳細については、『Adaptive Server Enterprise Transact-SQL ユーザーズガイド』を参照してください。

複数の結果セットを処理する方法の詳細については、Microsoft の Web サイトで『ADO.NET プログラマーズガイド』を参照してください。

Adaptive Server へのデータ転送を高速化する新しい SSIS Custom Data Flow Destination コンポーネント

Adaptive Server ADO.NET Data Provider のディストリビューションには SQL Server Integration Services (SSIS) Custom Data Flow Destination コンポーネントが含まれるようになりました。これは、Adaptive Server の変換先へのデータ転送を高速化します。

この高速データ転送は、**AseBulkCopy** クラスでサポートされる Adaptive Server のバルク挿入プロトコルを使用します。 この名前付き SybaseAdaptiveServerAdoNetDestination コンポーネントは、Adaptive Server ADO.NET Data Provider およびアセンブリファイルとともに次のディレクトリにインストールされます。 `%SYBASE%¥DataAccess¥ADONET ¥SybaseAdaptiveServerAdoNetDestination.dll` (32 ビットシステム) および `%SYBASE%¥DataAccess64¥ADONET ¥SybaseAdaptiveServerAdoNetDestination.dll` (64 ビットシステム)

SQL Server 2008 の Adaptive Server ADO.NET Destination SSIS コンポーネントの設定

Adaptive Server ADO.NET Destination SSIS コンポーネントを設定します。

1. `Sybase.AdoNet2.AseDestination.dll` を `C:¥Program Files ¥Microsoft SQL Server¥100¥DTS¥PipelineComponents` および `C: ¥Program Files (x86)¥Microsoft SQL Server¥100¥DTS ¥PipelineComponents` にコピーします。
2. ローカルドライブの Microsoft SQL Server ディレクトリのいずれかから、SDK インストールで提供される AseGacUtility を使用して `Sybase.AdoNet2.AseDestination.dll` アセンブリを登録します。
3. SQL Server Business Intelligence Studio を起動します。
4. [ツールボックス] タブで、[データフローの変換先] タブを右クリックし、[アイテムの選択] を選択します。
   [ツールボックスアイテムの選択] ウィンドウが表示されます。

5. [SSIS データフロー項目] タブを選択します。 [Sybase Adaptive Server Enterprise ADO NET Destination]、[OK] の順にクリックします。 [ツールボックス] > [データフローの変換先] を選択して、Sybase Adaptive Server ADO NET Destination コンポーネントを確認します。
6. SSIS プロジェクトを作成するには、メニューから [ファイル] > [新規作成] > [プロジェクト] > [Integration Services プロジェクト] を選択します。 制御フローオブジェクトを作成するか、[制御フロー項目] ツールボックスからドラッグアンドドロップします。
7. [データフローの変換先] および [データフローの変換元ツールボックス] タブから、[Sybase Adaptive Server ADO NET Destination Component] と [ADO NET Source Component] を [データフロー] タブにドラッグアンドドロップします。
8. [接続マネージャー] ウィンドウで使用可能な変換元または変換先がない場合は、[接続マネージャー] ウィンドウを右クリックして、[新しい ADO.NET 接続] を選択します。 既存のデータ接続を選択するか、[新規] をクリックします。
9. 変換先 Adaptive Server への接続を新たに作成するには、[ADO.NET の接続マネージャーの構成] ウィンドウで、[新規] ボタンをクリックして、[Sybase Adaptive Server Enterprise Data Provider] を選択します。
10. [接続マネージャー] ウィンドウに、接続のプロパティを入力します。
11. バルク挿入を有効にするには、[追加の接続プロパティ] テキストボックスに次のように入力します。 `enablebulkload=1`

注意： バルク挿入機能使用の詳細は、『Adaptive Server Enterprise ADO.NET Data Provider ユーザーズガイド』の「**AseBulkCopy**」を参照してください。

12. [OK] をクリックします。
13. データフローの ADO.NET 変換元に、接続とデータアクセスモードを設定します。 ADO.NET 変換元からデータフローパスに接続したら、Sybase Adaptive Server ADO NET Destination Component を右クリックして、[高度な編集の表示] を選択します。
14. [接続マネージャー] タブの [接続マネージャー] フィールドから ASE の接続を選択します。 [コンポーネントのプロパティ] タブで、TableName プロパティを変換先のテーブル名に設定します。
15. [入力列] タブを選択し、[名前] チェックボックスをオンにします。 これで、変換元テーブルで指定されたすべてのカラムが選択されます。
16. [OK] をクリックします。

注意： SQL Server 2008 からのデータ転送用 SSIS 変換先コンポーネントは、`Sybase.AdaptiveServerAdoNetDestination.dll` から `Sybase.AdoNet2.AseDestination.dll` に名前が変更されています。

接続が確立されます。 データ転送の詳細については、Microsoft SSIS のマニュアルを参照してください。

jConnect 動的ロギングレベル

jConnect が強化され、アプリケーションユーザがメッセージの細分性を Level.FINE、Level.FINER、および Level.FINEST に設定できるようになりました。

次に例を示します。

- ユーザが **SybConnection** クラスでロギングレベルを Level.FINE に設定すると、jConnect は次の内容を報告します。
  Dr1_Col setClientInfo(Properties)
- **SybConnection** クラスで Level.FINER に設定した場合は、次のように報告されます。
  Dr1_Co1 setClientInfo(Properties.size = [3])
- **SybConnection** クラスで Level.FINEST に設定した場合は、次のように報告されます。
  Dr1_Co1 setClientInfo(Properties = [[ClientUserValue, ApplicationNameValue, ClientHostnameValue]])

『jConnect for JDBC プログラマーズリファレンス』を参照してください。

jConnect でのコンバータクラスのパッケージ名の変更

jConnect 7.07 では、すべての文字セットコンバータクラスのパッケージ名およびファイルパスが変更されています。

文字セットコンバータクラスファイルは `com/sybase/jdbc4/utils` から `com/sybase/jdbc4/charset` に移動しました。 jConnect 7.07 での文字セットコンバータクラスのパッケージ名の変更は次のとおりです。

- **com.sybase.jdbc4.utils.TruncationConverter** は **com.sybase.jdbc4.charset.TruncationConverter** に変更
- **com.sybase.jdbc4.utils.PureConverter** は **com.sybase.jdbc4.charset.PureConverter** に変更

注意： フルパッケージ名を使用するために文字セットコンバータクラスを拡張するクラスを宣言している場合は、パッケージ名を **com.sybase.jdbc4.utils** から **com.sybase.jdbc4.charset** に変更する必要があります。

クラス参照をコーディングする代わりにワイルドカード文字のインポートの使用をおすすめします。 次に例を示します。

```
import com.sybase.jdbc4.charset.*;

import com.sybase.jdbc4.utils.*;
```

パッケージ名のコンバータクラス参照はインポート文で解決されます。

jConnect での PreparedStatement パラメータ制限数の増加

以前のバージョンでは、**PreparedStatement** のパラメータの最大数は 2048 に制限されていました。jConnect 7.07 では、より大きな制限数をサポートする Adaptive Server に接続した場合、32767 のパラメータをサポートするようになりました。

Adaptive Server ODBC ドライバの新しい SkipRowCountResults 接続プロパティ

SkipRowCountResults 接続プロパティは、ローカウントの結果を返す文を ODBC ドライバで処理する方法を制御するために使用できます。

UPDATE 文、**INSERT** 文および **DELETE** 文はローカウントの結果を返します。 **SELECT** 文は結果セットを返します。 ODBC アプリケーションは、ローカウントまたは結果セットを返す混合文を使用するバッチを実行する場合があります。

SkipRowCountResults が 1 (デフォルト) に設定されている場合は、Adaptive Server ODBC ドライバはいずれのローカウントの結果もスキップします。 **SQLExecDirect** または **SQLExecute** を使用して文のバッチを実行後、ODBC アプリケーションは最初の結果セットに配置されます。 **SQLMoreResults** への後続の呼び出しではローカウントの結果はスキップされ、アプリケーションは次に利用可能な結果セットに配置されます。

SkipRowCountResults が 0 に設定されている場合、Adaptive Server ODBC ドライバは、各結果セットまたはローカウントで停止します。 **SQLExecDirect** または **SQLExecute** を使用して文のバッチを実行後、アプリケーションは最初に利用可能な結果 (結果セットまたはローカウント) に配置されます。 ODBC アプリケーションは **SQLFetch** を使用して結果セットを取得するか、**SQLRowCount** を使用してローカウントの結果を取得します。 **SQLMoreResults** への後続の呼び出しでアプリケーションは次に利用可能な結果 (結果セットまたはローカウントのいずれか) に配置されます。

Adaptive Server ODBC ドライバでの AF_UNIX ソケット のサポート

Adaptive Server ODBC ドライバでは、Adaptive Server と通信するための **AF_UNIX** ソケットがサポートされるようになりました。

現在、このサポートは、Linux x86-64 64 ビット版プラットフォームに限定されています。 ODBC アプリケーションおよび Adaptive Server の両方が同じホストにあり、AF_UNIX ソケットを使用するために設定されている場合は、**AF_UNIX** ソケットを使用できます。 AF_UNIX ソケットは TCP/IP ソケットよりもパフォーマンスに優れています。 ODBC から AF_UNIX ソケットを有効にするには、次の接続文字列プロパティを設定します。

- **networklibraryname=afunix** - Adaptive Server ODBC ドライバに AF_UNIX ソケットが使用されていることを通知します。
- **server=<full path to the pipe>** - AF_UNIX ソケットへのパスです。 たとえば、`/tmp/test/demo_socket` です。

AF_UNIX ソケットを使用するための Adaptive Server の構成の詳細については、Sybase Adaptive Server Enterprise のマニュアルを参照してください。

Adaptive Server ODBC ドライバの AdjustLargePrecisionAndScale 接続プロパティ

15.7 より以前のバージョンの Adaptive Server ODBC ドライバでは、数値または 10 進数のカラムに精度および位取りを設定するための **SQLSetDescField()** への呼び出しをサポートしていませんでした。

API への呼び出しはいずれも無視され、Adaptive Server ODBC ドライバはカラムの精度および位取りを受け取った値に基づいて設定していました。 Adaptive Server は ODBC 数値構造体よりも高い精度をサポートしているため、Adaptive Server ODBC ドライバはサーバから受け取る値を必要に応じてスケールダウンして ODBC 数値構造体に格納していました。 バージョン 15.7 以降では、Adaptive Server ODBC ドライバは数値または 10 進数のカラムの精度および位取りを設定する **SQLSetDescField()** への呼び出しを無視しません。 そのため、新しい Adaptive Server ODBC ドライバにより、以前はそのまま動作していた ODBC アプリケーションがデータオーバフローエラーを受け取るようになることもあります。 **AdjustLargePrecisionAndScale** プロパティは以前の動作の継続を可能にし、Adaptive Server ODBC ドライバが最適な精度および位取りを選択してサーバーから受け取った値を格納できるようにします。

デフォルトで **AdjustLargePrecisionAndScale** が 0 の場合、Adaptive Server ODBC ドライバが、精度または位取りを設定する **SQLSetDescField()** API に対する呼び出しを受け入れるようにします。

AdjustLargePrecisionAndScale 接続プロパティを 1 に設定した場合は、Adaptive Server ODBC ドライバは、精度または位取りを設定するための **SQLSetDescField()** API に対する呼び出しを無視し、実際のデータ値の精度および位取りを使用します。

SQLSetDescField() についての詳細は、Microsoft Developers Networkhttp://msdn.microsoft.com/ を参照してください。

ESD #4 の新機能

ESD #4 には、Open Client 15.7 と Open Server 15.7、SDK 15.7、Python 用 Adaptive Server Enterprise 拡張モジュール 15.7、PHP 用 Adaptive Server Enterprise 拡張モジュール 15.7、Perl 用 Adaptive Server Enterprise データプロバイダ 15.7 の新機能が導入されています。

ESD #4 の Open Client 15.7 と Open Server 15.7 の機能

Open Client 15.7 と Open Server 15.7 は、Open Client ファイルと Open Server ファイルに対するより厳密なパーミッション (UNIX)、バッチパラメータ、新しいセーフ文字列処理ルーチンなどの新機能を提供するように強化されています。

Open Client ファイルと Open Server ファイルに対する厳密化されたパーミッション (UNIX のみ)

ESD#4 以降、新たに生成された Open Client ファイルと Open Server ファイルには厳密化されたパーミッションが与えられます。

表 6 : ファイルおよびそれらのパーミッション設定

ファイル	パーミッション
Interfaces ファイル	rw- r-- r-- (644)
BCP データファイル	rw- r-- --- (640)
BCP フォーマットファイル	rw- r-- --- (640)
BCP 出力ファイル	rw- --- --- (600)
BCP エラーファイル	rw- --- --- (600)
ISQL 出力ファイル (-o オプション)	rw- --- --- (600)
ISQL コマンド履歴ファイル	rw- --- --- (600)
ISQL 一時ファイル	rw- --- --- (600)
ISQL 出力リダイレクト	rw- --- --- (600)
Open Server ログファイル	rw- --- --- (600)
LDAP デバッグログファイル	rw- --- --- (600)

ファイル	パーミッション
Kerberos デバッグログファイル	rw- --- --- (600)
Netlib トレース出力ファイル	rw- --- --- (600)
DCL トレース出力ファイル	rw- --- --- (600)

注意： これらのパーミッションは、新たに生成されたファイルにのみ適用されます。既存のファイルの場合はそれぞれのパーミッションが保持されます (通常は rw- rw- rw- (666))。 Microsoft Windows の場合、ファイルのパーミッションは変更されずに保持されます。

libtcl*.cfg ファイルへの代替パスを設定するための新しい環境変数 SYBOCS_TCL_CFG

ESD#4 以降、新しい環境変数 SYBOCS_TCL_CFG を使用し、`libtcl.cfg` ファイルと `libtcl64.cfg` ファイルの代替フルパス名を設定できるようになりました。

次に例を示します。

Windows:

```
set SYBOCS_TCL_CFG c:¥joe¥libctl.cfg
```

UNIX:

```
%setenv SYBOCS_TCL_CFG /usr/u/joe/libtcl.cfg
```

デフォルトで、`libtcl.cfg` ファイルと `libtcl64.cfg` ファイルの検索場所は、Windows の場合は `%SYBASE%¥%SYBASE_OCS%¥ini` ディレクトリ、UNIX の場合は `$SYBASE/$SYBASE_OCS/config` ディレクトリです。

CS_LIBTCL_CFG プロパティを使用して `libtcl.cfg` ファイルと `libtcl64.cfg` ファイルの代替パスを設定することもできます。

ユニバーサルリモートパスワードを設定するための新しい isql コマンドラインオプション --URP

新しいコマンドラインオプション `--URP` を使用すると、Adaptive Server にアクセスするクライアントに対するユニバーサルリモートパスワードの設定を有効にすることができます。

```
isql  --URP  remotepassword
```

remotepassword がユニバーサルリモートパスワードです。

例:

```
%isql --URP “ASEremotePW”
```

SYBPLATFORM の新しい linux64 設定と nthread_linux64 設定

環境変数 *SYBPLATFORM* の有効な設定として `linux64` および `nthread_linux64` (スレッドアプリケーション用) が追加され、Open Client と Open Server のサンプルアプリケーションを Linux x86-64 64 ビット版でコンパイルするために使用できるようになりました。 既存の `linuxamd64` 設定と `nthread_linuxamd64` 設定は、同じ用途で引き続き有効です。

Microsoft Windows 64 ビット版用 LAN Manager ドライバ

Open Client と Open Server には `libsybsmssp64.dll` が含まれます。これは、Microsoft Windows x86-64 64 ビット版用の 64 ビット LAN Manager ドライバです。`libsybsmssp64.dll` は `%SYBASE%¥%SYBASE_OCS%¥dll` に格納されており、その動作は 32 ビットのドライバ `libsybsmss.dll` と似ています。

バッチパラメータのサポート

ESD #4 以降、Open Client と Open Server でコマンド自体を終了せずに複数のセットのコマンドパラメータを送信できるようになりました。

Open Client アプリケーションでは、新しい **ct_send_params()** ルーチンを繰り返し使用してパラメータを転送できます。このとき、前のコマンドの結果を処理することも、コマンド自体を再送することも必要としません。 Open Server アプリケーションでは、**SRV_S_PARAM_BATCHING** プロパティを CS_TRUE に設定します。

ct_send_params

コマンドパラメータをバッチ送信します。

構文

```
CS_RETCODE ct_send_params(
  CS_COMMAND *cmd,
  CS_INT reserved)
```

パラメータ

- *cmd*
  CS_COMMAND 構造体を指すポインタです。
- *reserved*
  CS_UNUSED に設定されます。 これは後で使用できるように予約されているプレースホルダです。

戻り値

ct_send_params で次の値が戻されます。

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
この関数を呼び出すと、**ct_param()** または **ct_setparam()** を使用してあらかじめ指定されたパラメータが送信されます。 パラメータの送信を停止するには、最後の **ct_send_params()** 呼び出しの後に **ct_send()** 呼び出しを使用します。 これはパラメータの終了を示すシグナルとなり、現在のコマンドが完了します。

- 最初の **ct_send_params()** 呼び出しによって、実際のコマンド、すべてのパラメータのパラメータフォーマット、および最初のパラメータセットがサーバに送信されます。 後続の呼び出しでは、フォーマットなしで各パラメータのみが送信されます。
- サーバでコマンドの処理を開始できるように、パラメータが格納されているネットワークバッファは **ct_send_params()** に対する呼び出しのたびにフラッシュされます。
- **ct_send()** とは異なり、**ct_send_params()** では現在のコマンドは終了しません。 **ct_send_params()** を繰り返し呼び出すことによって、複数のパラメータセットを送信できます。
- 結果の処理は、コマンドを完了するための **ct_send()** 呼び出しの後でのみ可能になります。 **ct_results()** が **ct_send()** の前に呼び出されると、エラーが発生します。

ct_setparam() を使用した再バインド

複数のパラメータセットを送信する場合、アプリケーションが指す CT-Library はメモリ内の前のパラメータセットのロケーション以外でなければならない場合があります。

パラメータを再バインドするには、**ct_setparam()** を使用して、別のデータロケーションを指定します。 既存の **ct_setparam()** 宣言を次に示します。

```
ct_setparam(cmd, datafmt, data, datalenp, indp)

  CS_COMMAND *cmd;
  CS_DATAFMT *datafmt;
  CS_VOID *data;
  CS_INT *datalenp;
  CS_SMALLINT *indp;
```

ct_setparam() 呼び出しの *data*、*datalenp*、*indp* の各パラメータに新しい値を指定して、別のメモリロケーションにバインドします。

ct_send_params() 呼び出しの後、該当するパラメータのフォーマットは変更できなくなります。 したがって、**ct_send_params()** 呼び出しの後に実行される **ct_setparam()** 呼び出しで、*datafmt* に NULL 値を渡す必要があります。

再バインドできるパラメータは、最初に **ct_setparam()** でバインドされていたパラメータだけです。

Server-Library に対するバッチパラメータのサポート

Open Server の Server-Library でバッチパラメータのサポートを有効にするには、**SRV_S_PARAM_BATCHING** サーバプロパティを CS_TRUE に設定します。

たとえば、**srv_run()** の前に次のように指定します。

```
if (srv_props(ctos_ctx->cx_context, CS_SET,
SRV_S_PARAM_BATCHING, (CS_VOID *)&cs_true, sizeof(cs_true), NULL) !=
CS_SUCCEED)
{...}
```

これで、コマンドに複数セットのコマンドパラメータが含まれている場合に **srv_xferdata()** に 2 つのリターンコードが新しく割り当てられるようになります。

- CS_PARAMS_MORE - パラメータが正常にコピーされ、複数のパラメータがバッチ内に存在することを示します。
- CS_PARAMS_END - パラメータが正常にコピーされたことを示します。 これがバッチ内の最後のパラメータセットです。

サンプルプログラム

2 つの新しい CT-Library サンプルプログラムが用意されています。

- `batch_lang.c` - **ct_send_params()** が言語文でどのように使用されるかを示します。 このサンプルプログラムでは、**ct_send_params()** を繰り返し使用して、ファイルから読み込まれた行をテーブルに挿入します。 読み込まれる行ごとにパラメータに同じロケーションが使用されるため、複数の **ct_send_params()** 呼び出しの間に **ct_param()** または **ct_setparam()** を呼び出す必要はありません。
- `batch_dynamic.c` - 動的 SQL を使用して、異なる複数のメモリロケーションにデータが存在するサーバにパラメータを送信します。 したがって、このサンプルプログラムは、**ct_send_params()** を再呼び出しする前に、各変数に再バインドするために **ct_setparam()** をどのように使用できるか、ということも示しています。

サンプルプログラム `ctos` に次の内容が追加されて、更新されています。

- SRV_S_PARAM_BATCHING サーバプロパティをオンにします。
- **ct_setparams()** を使用してデータのロケーションに CT-Lib をバインドします。
- **srv_xferdata()** からの新規リターン値を処理します。

- コマンドパラメータのセットごとに **ct_send_params()** を呼び出します。

新しい CS-Library 文字列処理ルーチン

cs_strlcpy、**cs_strlcat**、および **cs_snprintf** は 3 つの新しい CS-Library 文字列処理ルーチンです。

cs_strlcpy
セーフ文字列のコピー関数。 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、常に null で終了する文字列になります。例外は、*source_str* か *target_str* が NULL である場合、または、*target_size* が 0 である場合です。

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* - オーバフローの場合。
- **strlen(*source_str*)** - 他のすべてのケース。

cs_strlcat
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* - オーバフローの場合。
- **strlen(*target_str*) + strlen(*source_str*)** - 他のすべてのケース。

cs_snprintf
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ります。

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

ESD #4 で jConnect および Adaptive Server のドライバとプロバイダに対応する SDK 15.7 機能

ESD #4 には、jConnect for JDBC 7.07、Adaptive Server Enterprise ODBC ドライバ 15.7、Adaptive Server Enterprise OLE DB プロバイダ 15.7、および Adaptive Server Enterprise ADO.NET Data Provider 15.7 の新機能が導入されています。

細密なパーミッションと述部付きパーミッション

Adaptive Server 15.7 ESD #2 以降、役割権限の管理モデルが強化されています。

- 作業の分割 (SOD) と最小権限 (LP) の原則を強化するために、付与可能な細密な新しいシステム権限が追加されています。 これらの付与可能なシステム権限は、サーバワイドまたはデータベースワイドの権限にすることができます。
- システム定義ロールの *sa_role*、*sso_role*、*oper_role*、*replication_role* および *keycustodian_role* は、明示的に付与する一連の権限で構成された*権限コンテナ*として再構築されています。
- カスタム役割は、あらかじめ用意されたシステム定義役割を基に、権限を付与または取り消すことによって作成できるようになりました。
- **CREATE PROCEDURE** 文で新しいオプション **EXECUTE AS OWNER | CALLER** がサポートされるようになりました。 これにより、ASE でプロシージャの所有者または呼び出し元として、ランタイムパーミッションを検査し、DDL を実行して、オブジェクト名を解決できます。
- 強化された役割権限管理モデルを有効にするには、新しい **enable granular permissions** 設定オプションを使用します。

Adaptive Server Enterprise 15.7 ESD #2 のマニュアルを参照してください。

jConnect for JDBC、Adaptive Server Enterprise ODBC ドライバ、Adaptive Server Enterprise OLE DB プロバイダ、および Adaptive Server Enterprise ADO.NET Data Provider では、新しい役割権限管理モデルが使用可能な Adaptive Server に接続されている場合、新しい役割権限管理モデルがサポートされます。

述部付きの権限付与に使用された述部に関する情報を返すために、次のメソッドは PREDICATE という名前の追加カラムを返します。

- ODBC - **SQLColumnPrivileges()** および **SQLTablePrivileges()**
- JDBC - **ResultSetgetColumnPrivileges()** および **ResultSetgetTablePrivileges()**
- OLE DB - **IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMN_PRIVILEGES)** および **IDBSchemaRowset::GetRowset(DBSCHEMA_TABLE_PRIVILEGES)**

細密なパーミッションがデータベースに設定されている場合、これらのメソッドは細密なパーミッションを格納する追加のローを返します。

ADO.NET メソッドの動作は変更されません。

データコピーなしの alter table drop column

Adaptive Server バージョン 15.7 ESD #2 では、データのコピーなしでテーブルからカラムを削除できます。

これにより、**alter table drop column** の実行に要する時間が短縮されます。 Adaptive Server Enterprise 15.7 ESD #2 のマニュアルを参照してください。

jConnect for JDBC、Adaptive Server Enterprise ODBC ドライバ、Adaptive Server Enterprise OLE DB プロバイダ、および Adaptive Server Enterprise ADO.NET Data Provider は、この機能が有効化されている Adaptive Server に接続している場合、通常の DML オペレーション (**insert**、**delete**、**update**、**merge**) でこの機能がサポートされます。 この機能を使用するために、特別な設定を行う必要はありません。この機能は自動的にサポートされます。

jConnect for JDBC と Adaptive Server Enterprise ODBC ドライバでは、この機能が有効化されている Adaptive Server に接続している場合、バルクコピーにもこの機能がサポートされます。

この機能は、マテリアライズされていない (つまり仮想の) 計算カラム、暗号化カラム、および XML カラムには使用できません。

高速ログによるバルク挿入

Adaptive Server バージョン 15.7 ESD #2 では、**高速**モードですべてのログを記録する **bcp** を実行できます。これにより、完全なデータリカバリが可能になります。

以前のバージョンの**高速**モードによる **bcp** では、ページ割り付けのログのみが記録されていました。 Adaptive Server Enterprise 15.7 ESD #2 のマニュアルを参照してください。

jConnect for JDBC で、ENABLE_BULK_LOAD 接続プロパティに新しい値 LOG_BCP を設定し、完全ログを有効にします。

ODBC ドライバで、EnableBulkLoad 接続プロパティに新しい値 3 を設定し、完全ログを有効にします。 または、ODBC アプリケーションで、次のように SQL_ATTR_ENABLE_BULK_LOAD 接続属性を適切なレベルに設定します。

```
sr = SQLSetConnectAttr(hdbc, SQL_ATTR_ENABLE_BULK_LOAD,
(SQLPOINTER)3, SQL_IS_INTEGER);
```

これにより、1 つの接続で複数の異なるタイプのバルクロードを使用できるようになります。

ADO.NET Provider で、EnableBulkLoad 接続プロパティに新しい値 3 を設定し、完全ログを有効にします。

動的ロギング

ESD #4 以降、標準 Java Logger メカニズムを実装することによって、jConnect for JDBC にロギングメカニズムがサポートされます。

アプリケーションで jConnect のロガーを処理し、必要に応じてロギングのオンとオフを切り替えることができるようになりました。『jConnect for JDBC プログラマーズリファレンス』を参照してください。

クライアント情報の動的設定

ESD #4 以降、**setClientInfo()** 標準メソッドと **getClientInfo()** 標準メソッドを使用して、jConnect for JDBC のクライアント情報プロパティ (ApplicationName、ClientUser、ClientHostName) に新しい値を設定できます。この設定は、接続が確立された後でも可能です。

接続プロパティの動的設定

ESD #4 以降、**setClientInfo()** 標準メソッドと **getClientInfo()** 標準メソッドを使用して、jConnect for JDBC の接続プロパティに新しい値を設定できます。この設定は、接続が確立された後でも可能です。

動的設定が可能な接続プロパティのリストについては、『jConnect for JDBC プログラマーズリファレンス』を参照してください。

例外処理

jConnect for JDBC での例外処理が強化されています。 **getCause()** メソッドを使用すると、例外メッセージに **getcause()** を使用するディレクティブが含まれている場合に、例外の原因を取得できます。

パフォーマンス向上を目的とした新しい jConnect 接続プロパティ

ESD #4 以降、パフォーマンス向上のために、jConnect for JDBC に新しい接続プロパティのセットが提供されています。

プロパティ	説明	デフォルト値
OPTIMIZE_STRING_CONVER-SIONS	文字列変換の最適化を有効にするかどうかを指定する。 この最適化動作によって、SQL 準備文の実行にクライアントで文字データ型が使用される場合の jConnect のパフォーマンスを向上させることができる。 値: • 0 - デフォルト値。文字変換最適化は無効。 • 1 - jConnect が utf8 またはサーバのデフォルト文字セットを使用する場合に文字変換最適化を有効にする。 • 2 - すべての状況で文字列変換の最適化が有効になる。	0
SUPPRESS_PARAM_FORMAT	動的 SQL 準備文を実行するときに、jConnect クライアントで SUPPRESS_PARAM_FORMAT 接続文字列プロパティを使用して、パラメータデータ (TDS_PARAMS) を抑制できる。 クライアントは可能な場合には送信するパラメータメタデータを減らしてパフォーマンスを改善する。 値: • false - 選択、挿入、更新の各オペレーションでは TDS_PARAMFMT は抑制されない。 • true - デフォルト値。TDS_PARAMFMT が可能な限り抑制される。	true

プロパティ	説明	デフォルト値
SUPPRESS_ROW_ FOR-MAT	jConnect で、クライアントは SUPPRESS_ROW_FORMAT 接続文字列プロパティを使用して、準備された動的 SQL 文のローフォーマットが変更されたときに限り Adaptive Server が TDS_ROWFMT データまたは TDS_ROWFMT2 データを送信するように強制することができる。 Adaptive Server がクライアントに送信するデータを少なくすることができるため、パフォーマンスが向上する。 値: • false - ローフォーマットが変更されていない場合でも、TDS_ROWFMT データまたは TDS_ROWFMT2 データが送信される。 • true - デフォルト値。ローフォーマットが変更された場合にのみサーバが TDS_ROWFMT または TDS_ROWFMT2 を送信するように強制する。	true

新しい jConnect 接続プロパティ

ESD #4 以降、jConnect for JDBC に新しい接続プロパティのセットが追加されています。

プロパティ	説明	デフォルト値
EARLY_ BATCH_ READ_ THRESHOLD	読み込むローの数のスレッショルドを指定する。このスレッショルドに達すると、リーダスレッドではバッチへのサーバ応答の送信が減らされる。 早期読み込みが必要になることがない場合、この値を -1 に設定する。	-1
STRIP_ BLANKS	文字列値をテーブルに格納する前に、サーバで強制的に文字列値から先行ブランクおよび後続ブランクを削除する。 値: • false - デフォルト値。クライアントから送信された文字列値がそのまま格納される。 • true - 文字列値がテーブルに格納される前に、先行ブランクと後続ブランクがその文字列値から削除される。	false

プロパティ	説明	デフォルト値
SUPPRESS_CONTROL_TOKEN	コントロールトークンを抑制する。 値: • false - デフォルト値。コントロールトークンが送信される。 • true - コントロールトークンは抑制される。	false

Hibernate の JDBC サポートに関する注意

Hibernate は、POJO のドメインモデルを利用可能にする関連するプロジェクトの集合です。これにより、アプリケーションをオブジェクトマッピングや関係マッピングを超えて拡張できます。 多数のモジュールの中で、Hibernate の Core モジュールはオブジェクト関係マッピングを処理します。

Dialect は、データベースの言語でデータベースと通信する Hibernate のヘルパーです。 Hibernate では、Adaptive Server Enterprise の各バージョン用に次の Dialect ファイルが作成されています。

Sybase Dialect ファイル	ASE バージョン
`Sybase11Dialect.java`	11.9.2
`Sybase15Dialect.java`	15.0
`Sybase157Dialect.java`	15.7

注意： Hibernate と Sybase では、最新のリリースが積極的に評価され、必要に応じて新しい Dialect が作成されます。 更新された Dialect はすべて、定期的な Hibernate リリースに含められます。 このリリーススケジュールは Adaptive Server のリリーススケジュールと一致しない場合があります。 対応する Hibernate のリリース前に、更新された Dialect にアクセスする必要がある場合、それらの更新された Dialect は Hibernate on Sybase ASE で入手できる可能性があります。

SQL_ATTR_OUTPUT_NTS=SQL_FALSE のサポート

Adaptive Server Enterprise ODBC ドライバで、**SQL_ATTR_OUTPUT_NTS** 属性を `SQL_FALSE` に設定することにより、このドライバが null で終了した文字列データを返さないようにすることができるようになりました。

次のように、接続ハンドルの割り付けが行われる前にこの属性を設定します。

```
SQLSetEnvAttr(hEnv, SQL_ATTR_OUTPUT_NTS, (SQLPOINTER)SQL_FALSE,
SQL_IS_INTEGER)
```

デフォルトで、**SQL_ATTR_OUTPUT_NTS** 属性は SQL_TRUE に設定され、すべての出力文字列が null で終了します。

8 バイト長の SQLLEN データ型のサポート (Linux 64 ビット版のみ)

Linux x86-64 64 ビット版および Linux on POWER 64 ビット版の Adaptive Server Enterprise ODBC ドライバで、4 バイトの SQLLEN データ型と 8 バイトの SQLLEN データ型がサポートされるようになりました。

Red Hat と SUSE で、unixODBC ドライバマネージャがそれぞれのドライバマネージャとして提供されます。 2.2.13 より前のバージョンの unixODBC ドライバマネージャでは、4 バイトの SQLLEN データ型の使用が想定されています。 Red Hat Enterprise Linux 6 以降で提供されているデフォルト設定のように、2.2.13 以降のバージョンの unixODBC ドライバマネージャのデフォルト設定では、8 バイトの SQLLEN データ型が想定されています。 同様に、Adaptive Server Enterprise ODBC ドライバでは 2 つのバージョンのドライバが提供されます。 該当する 64 ビット版の Linux システムで使用されている unixODBC ドライバマネージャのバージョンを確認してください。

ESD #4 以降、DataAccess64/ODBC/lib/ ディレクトリに、次の 2 つのドライバ共有ライブラリファイルとソフトリンクが配置されます。

- libsybdrvodb-sqllen4.so - 4 バイトの SQLLEN データ型をサポートするオリジナルの libsybdrvodb.so ファイルに相当
- libsybdrvodb-sqllen8.so ファイル - 8 バイトの SQLLEN データ型をサポートする libsybdrvodb.so ファイルの新しいバージョン
- libsybdrvodb-sqllen4.so という名前になったオリジナルのドライバ共有ライブラリファイルを指す libsybdrvodb.so ソフトリンク

4 バイトの SQLLEN データ型を引き続き使用する場合、変更はありません。

8 バイトの SQLLEN データ型を使用する場合は、libsybdrvodb-sqllen8.so ファイルを指すソフトリンクを次のように変更します。

```
> cd DataAccess64/ODBC/lib
> rm libsybdrvodb.so
> ln -s libsybdrvodb-sqllen8.so libsybdrvodb.so
```

ODBC 遅延配列バインド

Adaptive Server Enterprise ODBC ドライバで、拡張された **SQLBindColumnDA()** API と **SQLBindParameterDA()** API が提供されるようになりました。これにより、一度の API 呼び出しで、すべてのカラムまたはパラメータをバインドできます。

これらの API を使用すると、カラムバッファまたはパラメータバッファへのポインタが、**SQLExecute()** 呼び出しまたは **SQLExecDirect()** 呼び出しごとに再評価され

ます。 したがって、アプリケーションでは別途 **SQLBindCol()** 呼び出しまたは **SQLBindParameter()** の呼び出しを行うことなくバッファを変更できます。 新しいポインタをバインドする呼び出しはリソースの消費が大きくなる可能性があるため、同一の文を何度も実行する必要がある場合に新しい拡張 API を使用することで、アプリケーションのパフォーマンスが向上します。 また、利用可能なデータを読み取ったり、必要な場所にコピーしたりするクエリを実行する前に、バッファのポインタを変更することにより、メモリコピーの操作を省略できる場合があります。

Adaptive Server Enterprise ODBC ドライバの『ユーザーズガイド』を参照してください。

ODBC データのバッチ処理用バルク挿入のサポート

15.7 リリースで導入された、ODBC データのバッチ処理では、パラメータ配列のバインド機能なしで、バルク挿入プロトコルを使用したバッチの挿入がサポートされます。

これを有効にするために、EnableBulkLoad 接続プロパティを適切なバルク挿入レベル (1、2、または 3) に設定し、HomogeneousBatch 接続プロパティを 2 に設定します。 Adaptive Server Enterprise ODBC ドライバの『ユーザーズガイド』を参照してください。

たとえば、`;enablebulkload=3;homogeneousbatch=2` を接続文字列に付加すると、バッチ処理で実行される単純な挿入文が高速ログによるバルク挿入文に変換されます。

または、SQL_ATTR_HOMOGENEOUS_BATCH 接続属性と SQL_ATTR_ENABLE_BULK_LOAD 接続属性を使用し、次のようにプログラムによって接続プロパティを設定しても同じ結果が得られます。

```
sr = SQLSetConnectAttr(hdbc, SQL_ATTR_HOMOGENEOUS_BATCH,
(SQLPOINTER)2, SQL_IS_INTEGER);
sr = SQLSetConnectAttr(hdbc,
SQL_ATTR_ENABLE_BULK_LOAD, (SQLPOINTER)3, SQL_IS_INTEGER);
```

ODBC ドライバマネージャのトレースを使用しない動的ロギングサポート

Adaptive Server Enterprise ODBC ドライバ 15.7 には、ODBC ドライバマネージャのトレースを使用しないアプリケーションロギング機能が導入されています。

アプリケーションの実行中にアプリケーションロギングを有効化 (または無効化) できます。 「ODBC ドライバマネージャのトレースなしのロギング」を参照してください。

ESD #4 では、このサポートが拡張され、新しい SQL_OPT_TRACE 環境属性を設定することにより、アプリケーションの実行中にアプリケーションロギングを動的に有効化/無効化できるようになりました。 有効値は、無効化の 0 (デフォルト) と有効化の 1 です。

```
// enable logging
SQLSetEnvAttr(0, SQL_OPT_TRACE, (SQLPOINTER)1,
  SQLINTEGER);
// disable logging
SQLSetEnvAttr(0, SQL_OPT_TRACE, (SQLPOINTER)0,
  SQLINTEGER);
```

- 動的ロギングは、グローバルに有効化および無効化できます。また、開始されたタイミングや、SQL_OPT_TRACE の設定に使用する環境ハンドルに含まれているどうかに関係なく、すべての接続に影響します。
- デフォルトで、このログは現在のディレクトリの `sybodbc.log` ファイルに書き込まれます。 別のファイルまたはファイルパスを設定するには、SQL_OPT_TRACEFILE 環境属性を使用します。

  ```
  SQLSetEnvAttr(0, SQL_OPT_TRACEFILE, (SQLPOINTER) "logfilepath",
  SQL_NTS);
  ```
- LOGCONFIGFILE 環境変数またはレジストリ値の設定により、アプリケーションの実行の存続期間にわたってロギングが可能であり、またこの設定は SQL_OPT_TRACE よりも優先されます。
- ODBC ドライバマネージャを使用している場合、SQL_OPT_TRACE の設定によってドライバマネージャのトレースが有効になります。また、ドライバのトレースには影響しません。
- クライアントアプリケーションでは、ドライバに直接リンクするときは null ハンドルを使用でき、ドライバマネージャのトレースを使用するときは割り付け済みハンドルを使用できます。
- `log4cplus` 設定ファイルを SQL_OPT_TRACE に使用することはできません。

TDS プロトコル取得の動的制御

新しい Adaptive Server Enterprise ODBC ドライバの SQL_ATTR_TDS_CAPTURE 接続属性を使用すれば、TDS プロトコルの取得の一時停止 (SQL_CAPTURE_PAUSE) および再開 (SQL_CAPTURE_RESUME) が可能です。

```
// pause protocol capture
SQLSetConnAttr(hDBC, SQL_ATTR_TDS_CAPTURE,
   (SQLPOINTER) SQL_CAPTURE_PAUSE, SQLINTEGER);

// resume protocol capture
SQLSetConnAttr(hDBC, SQL_ATTR_TDS_CAPTURE,
   (SQLPOINTER) SQL_CAPTURE_RESUME, SQLINTEGER);
```

デフォルトでは、TDS プロトコル取得は、ProtocolCapture 接続プロパティが接続に対して設定されている場合、接続している期間中動作します。

SQL_ATTR_TDS_CAPTURE (ProtocolCapture 接続プロパティを設定) を使用することで、アプリケーションでは、プログラム実行の任意のセグメントに対して選択的に TDS プロトコル取得の一時停止および再開を行うことができます。

SQL_ATTR_TDS_CAPTURE は、接続ハンドルの割り付け後に設定できます。 TDS プロトコル取得の一時停止および再開を接続が確立される前に実行したり、TDS プロトコル取得を使用していない接続に対して実行してもエラーにはなりません。 取得ストリームの整合性を確保するために、TDS プロトコル取得の一時停止または再開がドライバで遅延することがあります。 これにより、すべての PDU パケットの書き込みが維持され、Ribo や他のプロトコル変換ユーティリティにより取得が正確に消費されます。

接続のすべての TDS パケットの取得を必要とするアプリケーションには、SQL_ATTR_TDS_CAPTURE を設定しないでください。

Replication Server 接続のサポート

Adaptive Server Enterprise ODBC ドライバを Replication Server® に接続し、このサーバのモニタリングおよび管理を実行できます。

ODBC ドライバから送信される有効な Replication Server 管理コマンドのみが Replication Server でサポートされます。 Replication Server 接続のために、**BackEndType** 接続プロパティを `Replication Server` に設定します。

包括的 ADO.NET プロバイダアセンブリファイル

ESD #4 以降、Adaptive Server Enterprise ADO.NET Data Provider に含まれているのは、2 つのプロバイダアセンプリファイルのみで、それぞれがすべての機能を備えています。

- `Sybase.AdoNet2.AseClient.dll` - .NET 2.0、.NET 3.0、および .NET 3.5 の機能がサポートされます。
- `Sybase.AdoNet4.AseClient.dll` - .NET 4.1 以降の機能がサポートされます。

これらのファイルの 32 ビット版は `C:¥Sybase¥DataAccess¥ADONET¥dll` ディレクトリにインストールされ、64 ビット版は `C:¥Sybase ¥DataAccess64¥ADONET¥dll` ディレクトリにインストールされます。

旧式の DLL を参照するビルドまたは展開スクリプトをすべて更新します。

decimal データ型の精度/位取りの増大に対応する ADO.NET のサポート

Adaptive Server Enterprise ADO.NET Data Provider で、AseDecimal がサポートされるようになりました。これは、78 桁 の精度/位取りをサポートできる構造体です。

Adaptive Server の numeric データ型と decimal データ型では最大 38 桁 の精度/位取りがサポートされ、算術演算の結果では最大 78 桁 の精度/位取りがサポートされます。.NET Framework の decimal データ型の場合は、最大 28 桁 の精度/位取りがサポートされます。 このため、Adaptive Server の numeric データ型と decimal データ型、または算術演算の結果を .NET Framework の decimal データ型に読み込む場合に、データのオーバフローが発生する可能性があります。

Adaptive Server Enterprise ADO.NET Data Provider で、AseDecimal がサポートされるようになりました。これは、78 桁 の精度/位取りをサポートできる構造体です。AseDecimal 構造体を使用して numeric 値または decimal 値を取得するには、新しい UseAseDecimal 接続プロパティを 1 に設定します。 デフォルトでは、UseAseDecimal は 0 に設定されているため、AseDecimal 構造体は使用されません。

追加接続プロパティに対応する Visual Studio DDEX Connection ダイアログの強化

Adaptive Server Enterprise ADO.NET Data Provider で、Visual Studio DDEX Add Connection ダイアログに他の接続プロパティを追加できるようになりました。

- 接続プロパティは、セミコロン (;) で区切ったリストとして指定できます。
- 最後の接続プロパティの末尾にセミコロン (;) を付ける必要はありません。
- 値のないプロパティは無視されます。

現時点では、誤った接続の指定にフラグを付ける警告メッセージまたはエラーメッセージはありません。

OLE DB アプリケーションの新しい接続文字列

OLE DB アプリケーションの新しい接続文字列のセットが導入されています。

プロパティ名	説明	必須	デフォルト値
ProtocolCapture	このプロパティで OLE DB アプリケーションとサーバ間の通信が取得できるようにする。 Adaptive Server Enterprise OLE DB プロバイダの『ユーザーズガイド』を参照。	いいえ	ブランク

プロパティ名	説明	必須	デフォルト値
RetryCount、RetryDelay	接続再試行動作を制御する。 **RetryCount** は、接続失敗をレポートする前にサーバへの接続を試行する回数。 再試行と再試行の間に、ドライバは **RetryDelay** 秒遅延する。 デフォルトでは、OLE DB アプリケーションは接続を再試行しない。 次のように、これらの値を SQL.INI インタフェースと LDAP インタフェースに指定することもできる。 • **RetryCount** は、**RetryCount** として SQL.INI に指定し、**sybaseRetryCount** として LDAP に指定できる。 • **RetryDelay** は、**LoopDelay** として SQL.INI に指定し、**sybaseRetryDelay** として LDAP に指定できる。	いいえ	0
SuppressControlTokens	Adaptive Server が TDS_CONTROL トークンを送信しないように指定する。 値: • 0 - 可能な限り Adaptive Server が TDS_CONTROL トークンを送信するように強制する。 • 1- デフォルト値。Adaptive Server が TDS_CONTROL トークンを抑制するように強制する。	いいえ	1
SuppressParamFormat	フォーマットが変更された場合にのみ OLE DB アプリケーションがそのパラメータフォーマットトークンを送信するように指定する。 値: • 0 - 実行のたびに常に OLE DB アプリケーションがパラメータフォーマットトークンを送信するように強制する。 • 1- デフォルト値。フォーマットがすでに設定されている場合に、そのパラメータフォーマットトークンの送信が抑制されるように OLE DB アプリケーションに要求する。	いいえ	1

プロパティ名	説明	必須	デフォルト値
SuppressRow-Format	Adaptive Server が、初回実行時にのみローフォーマットトークンを送信するように、または、フォーマットが変更された場合にのみそのローフォーマットトークンを送信するように指定する。 値: • 0 - Adaptive Server が実行のたびにローフォーマット情報を送信するように強制する。 • 1- デフォルト値。可能な限り Adaptive Server がローフォーマットトークンの送信を抑制するように要求する。	いいえ	1
SuppressRow-Format2	Adaptive Server が、可能な限り TDS_ROWFMT2 バイトシーケンスではなく、TDS_ROWFMT バイトシーケンスでデータを送信するように指定する。 値: • 0 - デフォルト値。Adaptive Server が可能な限り TDS_ROWFMT2 でデータを送信するように強制する。 • 1 - Adaptive Server が可能な限り TDS_ROWFMT でデータを送信するように強制する。 Adaptive Server Enterprise OLE DB プロバイダの『ユーザーズガイド』を参照。	いいえ	0

ESD #4 の Python 用 Adaptive Server Enterprise 拡張モジュール

Python 用 Adaptive Server Enterprise 拡張モジュール が強化され、動的文とストアドプロシージャ用の新しいパラメータデータ型がサポートされるようになりました。

動的文とストアドプロシージャの新しいパラメータデータ型のサポート

ESD#4 以降、Python 用 Adaptive Server Enterprise 拡張モジュールに、動的文とストアドプロシージャ用のパラメータとして decimal データ型、money データ型、および LOB がサポートされています。

Python 用 Adaptive Server Enterprise 拡張モジュールでは、date、time、datetime、float の各パラメータもストアドプロシージャにサポートされています。

『Python 用 Adaptive Server Enterprise 拡張モジュールプログラマーズガイド』を参照してください。

ESD #4 の PHP 用 Adaptive Server Enterprise 拡張モジュール

ESD #4 以降、PHP 用 Adaptive Server Enterprise 拡張モジュールに、アプリケーション開発用の API のセットがすべて提供されています。

API タイプ	API	説明
接続:	**sybase_close()**	指定された ASE との接続をクローズする。
	sybase_connect()	ASE との接続をオープンする。
	sybase_pconnect()	(**新規**) ASE との永続的な接続をオープンする。
クエリ:	**sybase_affected_rows()**	(**新規**) 指定された接続に関するクエリの前回の挿入、削除、または更新の影響を受けたローの数を返す。
	sybase_query()	指定された接続にクエリを送信する。 完全な結果セットが自動的にフェッチおよびバッファされる。
	sybase_unbuffered_query()	(**新規**) 指定された接続にクエリを送信する。 **sybase_query()** のような、完全な結果セットの自動フェッチとバッファは行われない。
リモートプロシージャコール:	**sybase_rpc_bind_param_ex**	(**新規**) PHP 変数をリモートプロシージャパラメータにバインドする。
	sybase_rpc_execute	(**新規**) **sybase_rpc_init()** で初期化されたリモートプロシージャコールを実行する。
	sybase_rpc_init	(**新規**) 接続のリモートプロシージャ用に初期化された文を指す文識別子を返す。
結果セット:	**sybase_data_seek()**	(**新規**) 指定のロー番号を指すように、結果識別子に関連付けられている結果セット上の内部ローポインタを移動する。
	sybase_fetch_array()	(**新規**) 結果ローを関連配列、数値配列、またはその両方としてフェッチする。

API タイプ	API	説明
	sybase_fetch_assoc()	関連配列内で指定された結果識別子に関連付けられた結果セットから、データの 1 つのローをフェッチする。
	sybase_fetch_field()	(**新規**) フィールド情報を含むオブジェクトを返す。
	sybase_fetch_object()	(**新規**) 指定された結果識別子に関連付けられた結果セットから、データの 1 つのローをオブジェクトとしてフェッチする。
	sybase_fetch_row()	(**新規**) 数値配列内で指定された結果識別子に関連付けられた結果セットから、データの 1 つのローをフェッチする。
	sybase_field_seek()	(**新規**) 要求したフィールドオフセットに内部ポインタを設定する。
	sybase_free_result()	結果セットに関連付けられているすべてのメモリを解放する。
	sybase_next_result()	(**新規**) 接続に関する次の結果セットを指す結果セット識別子を返す。
	sybase_num_fields()	(**新規**) 結果セット内のフィールド数を返す。
	sybase_num_rows()	(**新規**) **select** 文の結果セット内のロー数を返す。
	sybase_use_result	(**新規**) 接続に関する前回バッファされていないクエリの結果セットを保管して、その保管されている結果セットを指す結果セット識別子を返す。
その他:	**sybase_get_last_message()**	(**新規**) サーバから返された最後のメッセージを返す。
	sybase_get_last_status	(**新規**) 接続で送信された最後のステータスの結果を返す。
	sybase_select_db()	(**新規**) 接続リソースで参照されているサーバ上の現在アクティブなデータベースを設定する。
	sybase_set_message_handler()	(**新規**) クライアントまたはサーバのメッセージが受信されると呼び出されるユーザ定義のコールバック関数を設定する。

『PHP 用 Adaptive Server Enterprise 拡張モジュールプログラマーズガイド』を参照してください。

ESD #4 の Perl 用 Adaptive Server Enterprise データベースドライバ

ESD #4 では、Perl 用 Adaptive Server Enterprise データベースドライバで次の機能が強化されています。

『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』を参照してください。

- 新しいデータベースハンドル属性
- 新しい **_data_fmt** プライベートメソッドを使用した新しいデフォルト日付変換と表示形式のサポート
- 新しい LONG/BLOB データ処理のサポート

  Perl 用 Adaptive Server Enterprise データベースドライバで、LONG/BLOB データの image データ型と text データ型がサポートされるようになりました。 各データ型において、最大 2GB のバイナリデータまで格納可能です。
- 新しい自動キー生成のサポート

  Perl 用 Adaptive Server Enterprise データベースドライバでは、自動キー生成用の IDENTITY 機能がサポートされます。 IDENTITY カラムでテーブルを宣言すると、各挿入に対して新しい値が生成されます。 この値は単調に増加しますが、連続性については保証されません。 最後の挿入により生成され、使用された値をフェッチするには、次のようにします。

  ```
  SELECT @@IDENTITY
  ```
- 新しいパラメータバインドのサポート

  Perl 用 Adaptive Server Enterprise データベースドライバで、パラメータのバインドが直接サポートされるようになりました。 '?' スタイルのパラメータのみがサポートされます。":1" プレースホルダタイプのパラメータはサポートされません。 text データ型または image データ型のパラメータのバインドはサポートされません。
- 入力パラメータと出力パラメータを使用した新しいストアドプロシージャのサポート

ESD #3 の新機能

ESD #3 には、Open Client 15.7 と Open Server 15.7 の新機能、および Python 用 Adaptive Server Enterprise 拡張モジュール 15.7 の新機能が導入されています。

サンプルファイル、文書ファイル、デバッグファイルのインストールの省略

ESD#3 以降、サンプルファイル、文書ファイル、デバッグファイルのインストールの省略を選択できます。

デフォルトでは、Open Server と SDK をインストールするときに、これらのファイルがインストールされます。 次の方法で、これらのファイルのインストールを省略します。

- GUI モード、コンソールモード、サイレントモードでインストールする場合、新しい `-DPRODUCTION_INSTALL=TRUE` インストーラコマンドライン引数を使用します。
- サイレントモードでインストールする場合、新しい **PRODUCTION_INSTALL=TRUE** プロパティを応答ファイルに使用します。

ESD #3 の Open Client 15.7 と Open Server 15.7 の機能

ESD #3 に提供される新機能には、64 ビット版 Microsoft Windows 用の CyberSafe Kerberos ドライバ、スクリプト言語の機能強化、UNIX 名前付きソケット、および拒否ローのロギングが含まれます。

64 ビット版 Microsoft Windows 用の CyberSafe Kerberos ドライバ

Open Client と Open Server には `libsybskrb64.dll` が含まれます。これは、Microsoft Windows x86-64 64 ビット版用の 64 ビット CyberSafe Trustbroker Kerberos ドライバライブラリです。

`libsybskrb64.dll` は `%SYBASE%¥%SYBASE_OCS%¥dll` に格納されており、その動作は 32 ビットの CyberSafe TrustBroker Kerberos ドライバライブラリ `libsybskrb.dll` と似ています。

UNIX 名前付きソケット

この機能では、UNIX 名前付きソケットが Open Client と Open Server でサポートされます。 このタイプのソケットは、UNIX ドメインソケットとも呼ばれます。

この機能では、ホスト内通信を高速化するために UNIX 名前付きソケットを使用することができます。これは、プロセス間通信のために TCP スタックをトラバースする必要がないためです。 この機能を有効にするには、移送タイプに `tcp` ではなく `afunix` を指定するエントリをディレクトリサービスレイヤに追加します。

たとえば、従来のインタフェースファイルエントリには、次のようなものがあります。

```
MYSERVER

master tcp unused myhost 8600
query tcp unused myhost 8600
```

TCP をリモートで使用しながら、ローカルクライアントには TCP の代わりに UNIX 名前付きソケットを使用する場合、上記のエントリは次のようになります。

```
MYSERVER

master afunix unused //myhost/tmp/MYSERVER.socket
query afunix unused //myhost/tmp/MYSERVER.socket
master tcp unused myhost 8600
query tcp unused myhost 8600
```

クライアントで拒否されたローのロギング

クライアントで、変換エラーやフォーマットエラーなどのエラーが検出されたことが原因で、ローが拒否された場合、そのローのログを取るために、新しい **bcp** オプションである `--clienterr errorfile` とこれに関連するエラーメッセージがエラーファイルに追加されています。

`--clienterr` オプションを `-e` オプションなしで使用すると、クライアントエラーメッセージがエラーファイルに書き込まれます。 ただし、サーバエラーメッセージはエラーファイルに書き込まれません。

`--clienterr` オプションを `-e` オプションと共に使用すると、**bcp** はコピーインオペレーションまたはコピーアウトオペレーションに進みません。

bcp による最大ロー処理能力の向上

bcp で処理可能なローの最大数が *INT32_MAX* から *UINT64_MAX* (つまり 18446744073709551615) に増大しました。

パラメータフォーマットの省略

Open Client では、Adaptive Server Enterprise の動的文のパラメータフォーマットの省略がサポートされるようになりました。

ESD #3 の Python 用 Adaptive Server Enterprise 拡張モジュール

Python 用 Adaptive Server Enterprise 拡張モジュールが強化され、入力パラメータと出力パラメータ、計算ロー、およびローカライズされたエラーメッセージを使用したストアドプロシージャがサポートされるようになりました。

Python を使用したストアドプロシージャへのアクセス

Python 用 Adaptive Server Enterprise 拡張モジュールに、入力パラメータと出力パラメータをストアドプロシージャに渡す機能が追加されました。

Cursor オブジェクトの **callproc()** メソッドを使用して、ストアドプロシージャを呼び出します。 ストアドプロシージャの実行中にエラーが発生した場合、**callproc()** から例外がスローされるため、`proc_status` 属性を使用してそのステータス値を取得できます。 このサポートは、Python DBAPI 仕様の拡張機能です。

これは、ロー結果が複数になるサンプル Python アプリケーションです。

```
import sybpydb
#Create a connection.
conn = sybpydb.connect(user='sa')
# Create a cursor object.
cur = conn.cursor()
# Call the stored procedure
try:
    cur.callproc('myproc')
    continue = True
    while(continue == True):
        row = cur.fetchall()
        continue = cur.nextset()
except sybpydb.Error:
    print("Status=%d" % cur.proc_status)
```

出力パラメータを指定するために、この拡張モジュールで **OutParam** コンストラクタが提供されます。 このサポートは、Python DBAPI 仕様の拡張機能です。

callproc() メソッドが、このメソッドに渡されたすべてのパラメータのリストを返します。 出力パラメータがあり、ストアドプロシージャから生成された結果セットがない場合、**callproc()** が完了するとすぐに、変更された出力値がそのリストに格納されます。 ただし、結果セットがある場合は、ストアドプロシージャから生成されたすべての結果セットが **fetch*()** メソッドで取得されてから、**nextset()** の呼び出しによって、結果セットが残っていないことが確認されるまで、変更された出力値はリストに格納されません。 **nextset()** メソッドは、予想される結果セットが 1 つだけである場合でも呼び出す必要があります。

これは、出力パラメータのあるサンプル Python アプリケーションです。

```
import sybpydb
#Create a connection.
conn = sybpydb.connect(user='sa')
# Create a cursor object.
cur = conn.cursor()
cur.execute("""
    create procedure myproc
    @int1 int,
    @int2 int output
    as
    begin
        select @int2 = @int1 * @int1
    end
    """)
int_in = 300
int_out = sybpydb.OutParam(int())
vals = cur.callproc('pyproc', (int_in, int_out))
print ("Status = %d" % cur.proc_status)
print ("int = %d" % vals[1])
cur.connection.commit()
# Remove the stored procedure
cur.execute("drop procedure myproc")
cur.close()
conn.close()
```

各種の出力パラメータの例が、ほかにもサンプルプログラム `callproc.py` に提供されています。

Python を使用したローの計算

Python 用 Adaptive Server Enterprise 拡張モジュールに、ロー計算のサポートが追加されています。

ロー計算処理の例は、サンプルプログラム `compute.py` に提供されています。

ローカライズされたエラーメッセージ

Python 用 Adaptive Server Enterprise 拡張モジュールで、エラーメッセージのローカライズがサポートされるようになりました。

ESD #1 の新機能

ESD #1 には、Open Client 15.7 と Open Server 15.7 の新機能、SDK 15.7 の新機能、および Python 用 Adaptive Server Enterprise 拡張モジュール 15.7 の新機能が導入されています。

ESD #1 の Open Client 15.7 と Open Server 15.7 の機能

ESD #1 の新機能には、FIPS 検証済み SSL フィルタ、および Perl 用 Adaptive Server Enterprise データベースドライバと Windows 64 ビット版の PHP 用 Adaptive Server Enterprise 拡張モジュールのサポートが含まれます。

FIPS 検証済み SSL フィルタ

Certicom SSL をサポートするプラットフォームの Sybase SSL フィルタが、Federal Information Processing Standard (FIPS) 140-2 に準拠するようになりました。

- HP-UX Itanium 32 ビット版
- HP-UX Itanium 64 ビット版
- IBM AIX 32 ビット版
- IBM AIX 64 ビット版
- Linux x86 32 ビット版
- Linux x86-64 64 ビット版
- Linux on POWER 32 ビット版
- Linux on POWER 64 ビット版
- Microsoft Windows x86 32 ビット版
- Microsoft Windows x86-64 64 ビット版
- Solaris SPARC 32 ビット版
- Solaris SPARC 64 ビット版
- Solaris x86 32 ビット版
- Solaris x86-64 64 ビット版

Linux on POWER 32 ビット版と 64 ビット版の共有オブジェクト SSL フィルタファイルの名前が、`libsybfcsissl.so` から `libsybfssl.so` に、および `libsybfcsissl64.so` から `libsybfssl64.so` に変更されました。 サンプル `libtcl.cfg` ファイルも更新されています。

```
[FILTERS]
;ssl=libsybfssl.so
```

Microsoft Windows x86-64 64 ビット版の SSL フィルタ DLL の名前が、`libsybfcsissl64.dll` から `libsybfssl64.dll` に変更されました。 サンプル `libtcl64.cfg` ファイルも更新されています。

```
[FILTERS]
;ssl=libsybfssl64
```

64 ビット版の Windows でサポートされる Perl 用 ASE データベースドライバと PHP 用 ASE 拡張モジュール

Perl 用 Adaptive Server Enterprise データベースドライバが、Microsoft Windows 64 ビット版のプラットフォームでサポートされ、ActivePerl 5.14.1 および DBI 1.616 に使用できるようになりました。

PHP 用 Adaptive Server Enterprise 拡張モジュールが、Microsoft Windows 64 ビット版のプラットフォームでサポートされ、PHP バージョン 5.3.6 に使用できるようになりました。

ESD #1 で jConnect および Adaptive Server のドライバとプロバイダに対応する SDK 15.7 機能

ESD #1 では、パフォーマンスの向上を目的として、パラメータフォーマットメタデータとローフォーマットメタデータの省略がサポートされています。

準備文のパフォーマンス向上を目的としたパラメータフォーマットメタデータの省略

準備文が再実行されるときにパラメータフォーマットメタデータを省略することによって、ODBC ドライバでの準備文のパフォーマンスを向上させることができます。

Adaptive Server 15.7 ESD#1 以降、パラメータフォーマットメタデータの省略がサポートされています。
DynamicPrepare 接続プロパティを 1 に設定してから、SuppressParamFormat 接続文字列プロパティを使用します。

有効な SuppressParamFormat 接続文字列プロパティ値は、次のとおりです。

- 0 - 準備文のパラメータフォーマットメタデータは抑制されません。
- 1 - デフォルト値。パラメータフォーマットメタデータが可能な限り抑制されます。

注意： 準備文のパラメータフォーマットメタデータを抑制できるのは、接続している Adaptive Server でこの機能がサポートされている場合だけです。
DynamicPrepare パラメータと SuppressParamFormat パラメータの両方が 1 に設定さ

れていても、Adaptive Server でパラメータフォーマットメタデータの省略がサポートされていない場合、Adaptive Server でそのパラメータ設定は無視されます。

例

次の ODBC 接続文字列によって、準備文のパラメータフォーマットメタデータが省略されます。

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;SuppressParam
Format=1;
```

クエリのパフォーマンス向上を目的としたローフォーマットメタデータの省略

ODBC ドライバおよび ADO.NET Data Provider で繰り返し実行されるクエリのパフォーマンスを向上させるには、セッション内でクエリが再実行されるときにローフォーマットメタデータ (TDS_ROWFMT または TDS_ROWFMT2) を抑制します。

Adaptive Server 15.7 ESD#1 以降、ローフォーマットメタデータの省略がサポートされています。

SuppressRowFormat 接続文字列プロパティを使用します。

有効な SuppressRowFormat 接続文字列プロパティ値は、次のとおりです。

- 0 - ローフォーマットメタデータは抑制されません。
- 1 - デフォルト値。可能な限り Adaptive Server w はローフォーマットメタデータを送信しません。

注意： ローフォーマットメタデータを抑制できるのは、接続している Adaptive Server でこの機能がサポートされている場合だけです。 SuppressRowFormat パラメータが 1 に設定されていても、接続している Adaptive Server でローフォーマットメタデータの省略がサポートされていない場合、Adaptive Server でそのパラメータ設定は無視されます。

例

次の ODBC 接続文字列によってローフォーマットメタデータが省略されます。

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;
SuppressRowFormat=1;
```

SuppressRowFormat2 と SQLBulkOperations

SQLBulkOperations API を使用する ODBC プログラムに SuppressRowFormat2 接続文字列プロパティを使用しないでください。

SuppressRowFormat2 を有効にすると、**SQLBulkOperations** に必要な情報が省略され、エラーの原因になります。

ESD #1 の Python 用 Adaptive Server Enterprise 拡張モジュール

ESD #1 以降、Python 用 Adaptive Server Enterprise 拡張モジュールで Python バージョン 2.6、2.7 および 3.1 がサポートされています。

Python 用の Adaptive Server Enterprise 拡張モジュールは、SDK インストーラからインストールできます。 インストール手順については、『Software Developers Kit/Open Server インストールガイド』および『Software Developers Kit/Open Server リリースノート』を参照してください。 Python 用 Adaptive Server Enterprise 拡張モジュールの使用については、『Python 用 Adaptive Server Enterprise 拡張モジュールプログラマーズガイド』を参照してください。

Python 用 Adaptive Server Enterprise 拡張モジュールの設定

アプリケーションで Python 用 Adaptive Server Enterprise 拡張モジュールを使用するよう、PYTHONPATH またはデフォルトのインストールディレクトリのパスにある Python 変数 *sys.path* を設定します。

Python モジュールの検索パス

Python は、Python 変数 *sys.path* で指定されるディレクトリのリスト内で、インポートされたモジュールを検索します。

sys.path

sys.path 変数はアプリケーションを含むディレクトリと環境変数 PYTHONPATH で指定されるディレクトリのリストから初期化されます。PYTHONPATH はシェル変数 PATH (つまりディレクトリ名のリスト) と同じ構文を使用します。

PYTHONPATH が設定されていない、またはそのモジュールファイルが見つからない場合は、インストールに依存するデフォルトのパス内で検索を続行します。Python 用 Adaptive Server Enterprise 拡張モジュールをアプリケーションで使用するには、PYTHONPATH または Python 変数 *sys.path* を次のディレクトリパス (各バージョンの Adaptive Server Python 拡張モジュールがインストールされるデフォルトディレクトリ) のいずれか 1 つに設定する必要があります。

プラットフォーム	Python のバージョン	デフォルトインストールパス
Windows	2.6	`$SYBASE¥$SYBASE_OCS¥python¥python26_64¥dll`

プラットフォーム	Python のバージョン	デフォルトインストールパス
	2.7	$SYBASE¥$SYBASE_OCS¥python¥python27_64¥dll
	3.1	$SYBASE¥$SYBASE_OCS¥python¥python31_64¥dll
その他のプラットフォーム	2.6、2.7	$SYBASE/$SYBASE_OCS/python/python26_64r/lib
	3.1	$SYBASE/$SYBASE_OCS/python/python31_64r/lib

Open Client 15.7 と Open Server 15.7 の機能

Open Client と Open Server のバージョン 15.7 では、ラージオブジェクト (LOB) ロケータ、ロー内/ロー外の LOB などその他多くのサポートが新機能として導入されました。

ラージオブジェクトのロケータのサポート

LOB ロケータには、データ自体ではなく、Adaptive Server 内の LOB データへの論理ポインタが含まれているため、Adaptive Server とそのクライアント間のネットワークを通過するデータの量が削減されます。

Adaptive Server 15.7 には、LOB ロケータを使用して LOB データを操作するための Transact-SQL コマンドおよび関数が含まれています。 これらのコマンドや関数は、Client-Library からの言語コマンドとして呼び出すことができます。 『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。

Client-Library の変更

CS_LOCATOR データ型は LOB ロケータをサポートしています。 **cs_locator_alloc()** API および **cs_locator_drop()** API は、CS_LOCATOR 変数のメモリをそれぞれ割り付けおよび割り付け解除します。 CS_LOCATOR 変数からの情報を取得するために、**cs_locator()** が追加されています。

Client-Library ルーチンである **cs_convert()** と **ct_bind()** は、CS_LOCATOR 変数を処理できるように強化されています。

CS_LOCATOR

CS_LOCATOR は、ロケータの値とオプションのプリフェッチされたデータを格納する opaque データ型です。

受信ロケータを CS_LOCATOR 変数にバインドする前に、**cs_locator_alloc()** を使用してこの変数のメモリを割り付けてください。そうしないとエラーが発生します。 変数が無用になった場合は、**cs_locator_drop()** を使用してそのメモリを解放します。

CS_LOCATOR 変数は再使用できますが、Adaptive Server 内の現在のロケータ値はトランザクションが終了すると無効になります。

CS_LOCATOR の型定数は次のとおりです。

- CS_TEXTLOCATOR_TYPE - `text` LOB 用
- CS_IMAGELOCATOR_TYPE - `image` LOB 用
- CS_UNITEXTLOCATOR_TYPE - `unitext` LOB 用

ロケータのプリフェッチされたデータやロケータ値の文字表現を CS_LOCATOR 変数から取得するには、**cs_convert()** を使用します。 CS_LOCATOR を CS_CHAR に変換すると、ロケータの 16 進数値が文字列として返されます。 ロケータを CS_TEXT_TYPE、CS_IMAGE_TYPE、または CS_UNITEXT_TYPE に変換すると、ロケータのプリフェッチされたデータが返されます。

サポートされている LOB ロケータの変換
LOB ロケータの変換を以下の表に示します。

	CS_TEXT_ LOCATOR	**CS_IMAGE_ LOCATOR**	**CS_UNITEXT_ LOCATOR**
CS_CHAR_TYPE	X	X	X
CS_TEXT_TYPE	X		
CS_IMAGE_TYPE		X	
CS_UNITEXT_TYPE			X
CS_TEXT_LOCATOR	X		
CS_IMAGE_LOCATOR		X	
CS_UNITEXT_LOCATOR			X
記号の説明: X = サポートされている変換			

ロケータのデータ型を使用するときは、次の内容が適用されます。

- **ct_bind()** は CS_DATAFMT の *maxlength* 値を無視します。これは、Client-Library がロケータのデータ型の長さを固定と見なすためです。 ロケータを使用して送信される、オプションのプリフェッチされたデータに必要なメモリは、その長さ全体に対して内部で割り付けられます。 *maxlength* の値がプリフェッチされたデータの長さに影響することはありません。
- 受信 LOB ロケータは CS_CHAR_TYPE にバインドできます。 ただし、ロケータを CS_TEXT_TYPE、CS_IMAGE_TYPE、または CS_UNITEXT_TYPE に直接バインドすることはできません。

cs_locator()
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。
- *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	アクション	***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_PREFET-CHLEN	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_PREFETCH-DATA	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、CS_UNITEXTLOCATOR_TYPE。

- *buffer* - データの格納先変数数を指すポインタ。 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。 *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。

cs_locator_alloc()

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。
- *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。

cs_locator_drop()
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。
- *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。

isql 拡張機能
isql は、LOB ロケータ値を 16 進数文字の形式で表示します。 CS_LOCATOR に格納されている、プリフェッチされたデータは表示されません。

例
LOB データをロケータに変換し、ロケータの値を表示します。

```
1> set send_locator on
2> go

1> select * from testable
2> go
```

```
charcol         textcol
--------------- ----------------------------------------------------
Hello           0x48656c6c6f20576f726c642e204865726520492061d2e2e
```

ラージオブジェクトのロケータに対する Open Server のサポート

LOB ロケータの機能が Server-Library に追加されているので、Open Server アプリケーションは LOB ロケータの言語コマンドをクライアントからバックエンドサーバに渡すことができます。

LOB ロケータをサーバからクライアントアプリケーションに渡すために、Open Server アプリケーションでは *CS_LOCATOR* 変数のメモリを割り付け、LOB 情報をバインドしてサーバから受け取ります。

srv_bind() と **srv_descfmt()** は、CS_TEXT_LOCATOR_TYPE、CS_IMAGE_LOCATOR_TYPE、CS_UNITEXT_LOCATOR_TYPE を処理できるように強化されています。

ラージオブジェクトのロケータのサポート

次の接続機能は、LOB ロケータの送受信のサポートを示します。

- CS_DATA_LOBLOCATOR - クライアントアプリケーションが CS_VERSION_157 によって初期化されたときに暗黙的に設定される読み取り専用要求機能です。Client-Library が LOB ロケータをサーバに送信できることを示します。
- CS_DATA_NOLOBLOCATOR - クライアントアプリケーションが設定する応答機能。基本となる Client-Library によってサポートされている場合でも、LOB ロケータを送信しないようにサーバに伝えます。

サーバからの LOB ロケータの要求

デフォルトでは、LOB のカラムや値を選択すると、Adaptive Server ではネゴシエートされた LOB ロケータがサポートされているかどうかにかかわりなく、LOB ロケータの代わりに LOB データを送信します。

明示的に LOB ロケータを要求するか、プリフェッチされたデータを要求するには、**ct_options()** を使用して次のクエリ処理オプションを設定します。

- CS_OPT_LOBLOCATOR - CS_TRUE に設定されている場合に、LOB 値ではなくロケータを返すようにサーバに要求するブール値。 このオプションは、クエリをサーバに送信する前に設定します。 デフォルトは **CS_FALSE** です。
- CS_OPT_LOBPREFETCHSIZE - サーバが送信する必要のある、プリフェッチされたデータのサイズを指定する整数。 image ロケータの場合、このサイズはプリフェッチされたデータのバイト数を示し、text および unitext ロケータの場合は文字数を示します。
  CS_OPT_LOBPREFETCHSIZE のデフォルト値は 0 で、これはプリフェッチされたデータを送信しないようにサーバに指示します。 -1 の値は、要求された LOB の LOB データ全体をそのロケータと共に取得します。

ロケータの値とオプションのプリフェッチされたデータは CS_LOCATOR データ型に格納されます。 クライアントは CS_LOCATOR 変数のメモリを割り付けてから、ロケータデータを要求する必要があります。

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);
```

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。

Bulk-Library の select into ロギング

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、Adaptive Server 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 プロパティは、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 プロパティ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。

BLK_CUSTOM_CLAUSE

アプリケーションは、**blk_props** Bulk-Library ルーチンを使用して、BLK_CUSTOM_CLAUSE を設定または取得できます。

表 7 : Client/Server BLK_CUSTOM_CLAUSE プロパティ

プロパティ名	説明	*buffer の値	適用対象	注意
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。

- **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。

Bulk-Library と bcp によるマテリアライズされていないカラムの処理

Bulk-Library は、Adaptive Server 15.7 でマテリアライズされていないカラムを処理できるように強化されています。

この機能強化により、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。

後続ゼロ保持のサポート

Open Client と Open Server のバージョン 15.7 では、Adaptive Server 15.7 に導入されている **disable varbinary truncation** 設定パラメータがサポートされます。 このパラメータは、Adaptive Server で varbinary null データと binary null データの後続のゼロを保持するかトランケートするかを指定します。

バージョン 15.7 より前の Adaptive Server および バージョン 15.7 より前の **bcp** と **bulklib** では、varbinary データ型の後続のゼロがトランケートされます。 バージョン 15.7 以降の Adaptive Server および バージョン 15.7 以降の **bcp** と **bulklib** では、varbinary データ型の後続のゼロをトランケートまたは保持できます。

デフォルトでは、サーバに対して **disable varbinary truncation** は 0 (オフ) になっています。 この機能を有効にするには、1 (オン) に設定します。

新しい DB-Library オーバフローエラー

DB-Library のオーバフローに関連してエラーが発生します。

整数オーバフローを発生させる DB-Library ルーチンを使用すると、次のエラーが発生します。

```
302 = SYBEINTOVFL, "DB-LIBRARY internal error: The arithmetic
operation results in integer overflow."
```

dbcursoropen DB-Library ルーチンの scrollopt パラメータと nrows パラメータを掛け合わせると、次のオーバフローエラーが発生します。

```
301 = SYBCOPNOV, "dbcursoropen(): The multiplication of scrollopt and
nrows results in overflow."
```

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

ocs.cfg ルーチン設定ファイルを名前のないアプリケーション (CS_APPNAME がアプリケーションで明示的に設定されていない場合) のアプリケーション固有設定に対して解析するかどうか、および検出された設定をアプリケーションに適用するかどうかを設定できるようになりました。

オペレーティングシステムから取得された実行プログラム名は、CS_APPNAME としてアプリケーションに設定され、ランタイム設定ファイルの解析に使用されます。

この機能を有効化するには、ocs.cfg ランタイム設定ファイルの DEFAULT セクションで、CS_USE_DISCOVERED_APPNAME を CS_TRUE に設定します。

CS_USE_DISCOVERED_APPNAME が CS_FALSE に設定されている場合 (デフォルト)、名前のないアプリケーションに対するランタイム設定ファイルの解析は行われません。

CS_SANITIZE_DISC_APPNAME では、名前のないアプリケーション (CS_APPNAME がアプリケーションで明示的に設定されていない場合) に対して検出されたアプリケーション名 (オペレーティングシステムから取得された実行プログラム名) をランタイム設定ファイルの解析にそのまま使用するか、大文字か小文字に変換してから使用するかを指定します。

ocs.cfg ランタイム設定ファイルの DEFAULT セクションで、CS_SANITIZE_DISC_APPNAME に次の値を設定できます。

- CS_CNVRT_UPPERCASE - 検出された名前を大文字に変換してから使用する。
- CS_CNVRT_LOWERCASE - 検出された名前を小文字に変換してから使用する。
- CS_CNVRT_NOTHING (デフォルト) - 検出された名前をそのまま使用する。

TCP ソケットバッファサイズの設定

TCP 入出力バッファのサイズは、Open Client および Open Server のコンテキストプロパティまたは接続プロパティとサーバプロパティを使用して設定できます。

Open Client および Open Server アプリケーションは、これらのプロパティ設定を使用して、オペレーティングシステムの **setsockopt** コマンドでバッファサイズを設

定します。 **setsockopt** は TCP の connect および accept コマンドの前に呼び出す必要があるので、これらの Open Client および Open Server プロパティを設定してから接続を確立する必要があります。

プロパティ

TCP の入力バッファサイズと出力バッファサイズを設定するためのコンテキストプロパティと接続プロパティは、CS_TCP_RCVBUF と CS_TCP_SNDBUF です。

表 8 : バッファサイズ設定用の Client-Library プロパティ

プロパティ	意味	**buffer* の値	レベル
CS_TCP_RCVBUF	クライアントアプリケーションの入力バッファのサイズ	正の整数	コンテキスト、接続
CS_TCP_SNDBUF	クライアントアプリケーションの出力バッファのサイズ	正の整数	コンテキスト、接続

コンテキスト例

```
ct_config(*context, CS_SET, CS_TCP_RCVBUF, &bufsize, CS_UNUSED,
NULL);
```

接続例

```
ct_con_props(*connection, CS_SET, CS_TCP_RCVBUF, &bufsize,
CS_UNUSED, NULL);
```

TCP 入力および出力バッファサイズを設定するためのサーバプロパティは、SRV_S_TCP_RCVBUF と SRV_S_TCP_SNDBUF です。

表 9 : バッファサイズ設定用のサーバプロパティ

プロパティ	設定/クリア	取得	*cmd* が CS_SET のときの *bufp*	*cmd* が CS_GET のときの *bufp*
SRV_S_TCP_RCVBUF	はい	はい	CS_INT	CS_INT
SRV_S_TCP_SNDBUF	はい	はい	CS_INT	CS_INT

サーバ例

```
srv_props(cp, CS_SET, SRV_S_TCP_SNDBUF, bufp, CS_SIZEOF(CS_INT),
(CS_INT *)NULL);
```

- これらのパラメータをアプリケーションに応じて設定します。 たとえば、クライアントが大量のデータをサーバに送信することが想定される場合は、CS_TCP_SNDBUF と SRV_S_TCP_RCVBUF を大きな値に設定し、対応するバッファサイズを増加します。

- デフォルトで、ソケットのバッファサイズはオペレーティングシステムの最大許容サイズに設定されています。

すべての 64 ビット製品用 isql64 および bcp64

64 ビットバージョンの **isql** および **bcp** (**isql64** および **bcp64**) は、Open Client と Open Server によってサポートされているすべての UNIX および Windows プラットフォームで使用できます。

Open Server および SDK 15.5 ESD #9 より前のバージョンでは、64 ビットの Windows で 64 ビットの **isql.exe** と **bcp.exe** のみを使用できます。 **isql.exe** または **bcp.exe** を参照するスクリプトがあり、64 ビットバージョンを使用する場合は、スクリプトの参照を **isql64.exe** または **bcp64.exe** に変更する必要があります。

拡張された可変長ローのサポート

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、または古い形式の 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` はターゲットテーブルにローを作成できませんでした。 カラム `%1!` は `8191` バイト以上のオフセットで開始します。この開始ロケーションは、テーブルの （ロー） フォーマットでは正確に表すことができません。」
- 「`BCP` がターゲットテーブル内のローの作成に失敗しました。 カラム `%1!` が `%2!` バイトより大きいオフセットで開始しています。この開始位置は、現在のデータベース設定では使用できません。」

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。

ローフォーマットのキャッシュ

Open Client 15.7 は、ローフォーマット情報のキャッシュをサポートしています。これにより、クライアントアプリケーションはデータサーバに対し、動的 SQL 文が呼び出されるたびにローフォーマット情報を送信しないように要求できます。ローフォーマットをキャッシュすると、データサーバとクライアントアプリケーション間のネットワークトラフィックが減少するため、システムのパフォーマンスが向上します。

デフォルトでは、ローフォーマットのキャッシュは Open Client 15.7 で有効になります。 無効にするには、CS_CMD_SUPPRESS_FMT 応答機能を CS_FALSE に設定します。 CS_CMD_SUPPRESS_FMT の値をチェックし、設定するには **ct_cmd_props()** を使用します。

サーバがローフォーマットの省略をサポートしているかどうかを判断するには、**ct_capability()** を使用して CS_RES_SUPPRESS_FMT の値を確認します。

注意： この機能は、ローフォーマットのキャッシュをサポートしているサーバにクライアントアプリケーションが接続している場合にのみ使用できます。

カーソルクローズ時のロックの解放のサポート

Open Client 15.7、Open Server 15.7、および Embedded SQL C プロセッサと COBOL 15.7 プロセッサは、Adaptive Server 15.7 に導入された **release_locks_on_close** カーソルオプションをサポートしています。

この機能を使用すると、カーソルのクローズ時に読み込みロックを解放できます。『ASE リファレンスマニュアル: コマンド』を参照してください。

Client-Library の使用法

ct_cursor 構文の *option* パラメータは、CS_CUR_RELLOCKS_ONCLOSE を含めるように拡張されています。

このオプションを使用すると、カーソルのクローズ時に共有ロックを解放するよう Adaptive Server に指示できます。 読み取り専用カーソルまたはスクロール可能カーソルに使用するには、OR ビット処理演算子、"|" (パイプ) を使用します。

- CS_CUR_RELLOCKS_ONCLOSE
- CS_CUR_RELLOCKS_ONCLOSE | CS_READ_ONLY
- CS_CUR_RELLOCKS_ONCLOSE | CS_FOR_UPDATE
- CS_CUR_RELLOCKS_ONCLOSE | CS_SCROLL_CURSOR
- CS_CUR_RELLOCKS_ONCLOSE | CS_SCROLL_INSENSITIVE
- CS_CUR_RELLOCKS_ONCLOSE | CS_SCROLL_SEMISENSITIVE
- CS_CUR_RELLOCKS_ONCLOSE | CS_NOSCROLL_INSENSITIVE

例

- クローズ時に共有ロックを解放するカーソルを宣言します。

```
ct_cursor(cmd, CS_CURSOR_DECLARE, cursor_name,
   CS_NULLTERM, select_statement, CS_NULLTERM,
   CS_CUR_RELOCKS_ONCLOSE);
```

- クローズ時に共有ロックを解放する insensitive スクロール可能カーソルを宣言します。

```
ct_cursor(cmd, CS_CURSOR_DECLARE, cursor_name,
   CS_NULLTERM, select_statement, CS_NULLTERM,
   CS_CUR_RELOCKS_ONCLOSE | CS_SCROLL_INSENSITIVE);
```

この機能を示す Open Client サンプルプログラムについては、`csr_disp_scrollcurs3.c` を参照してください。

Open Server の使用法

クライアントアプリケーションが CS_CUR_RELLOCKS_ONCLOSE オプションを指定してカーソルを宣言した場合、Open Server では SRV_CURDESC 構造体の **curstatus** (カーソルステータス) フィールドを SRV_CUR_RELLOCKS_ONCLOSE に設定します。

`ctos` コード例の `cursor.c` の図を参照してください。

ESQL/C と ESQL/COBOL の使用法

ESQL/C と ESQL/COBOL の **SQL DECLARE** 構文は、**RELEASE_LOCKS_ON_CLOSE** キーワードを含むように拡張されています。

```
EXEC SQL DECLARE cursor_name
    [SEMI_SENSITIVE | INSENSITIVE]
```

```
    [SCROLL | NOSCROLL]
    [RELEASE_LOCKS_ON_CLOSE]
    CURSOR FOR "select stmt"
    [for {read only | update [ of column_name_list]}]
```

次のフォーム以外で **RELEASE_LOCKS_ON_CLOSE** に **UPDATE** 句を使用することはできません。

```
EXEC SQL declare cursor c1 release_locks_on_close
   cursor for select * from T for update of col_a
```

この場合、**RELEASE_LOCKS_ON_CLOSE** は無視されます。

cpre と **cobpre** は次の **ct_cursor()** オプションを生成できません。

- CS_CUR_RELLOCKS_ONCLOSE | CS_READ_ONLY
- CS_CUR_RELLOCKS_ONCLOSE | CS_FOR_UPDATE

ESQL/C コード例は `example8.cp` に含まれており、ESQL/COBOL コード例は `example7.pco` に含まれています。

ストアドプロシージャパラメータとしてのラージオブジェクト

Open Client と Open Server 15.7 は、ストアドプロシージャの入力パラメータおよび動的 SQL 文のパラメータとして `text`、`unitext`、`image` の使用をサポートしています。

この機能の使用に関するログインネゴシエーションを助長するために、2 つの接続機能が追加されています。

- CS_RPCPARAM_LOB - クライアントアプリケーションはこの要求機能をサーバに送信し、ラージオブジェクト (LOB) データ型をストアドプロシージャの入力パラメータとして使用できるかどうかを判断します。 サーバはこの機能をサポートできないときに、初期ログインネゴシエーションのこの機能ビットをクリアします。ユーザがこのようなサーバに LOB パラメータを送信しようとすると、エラーが発生します。
- CS_RPCPARAM_NOLOB - クライアントアプリケーションはこの応答機能を送信し、パラメータとして LOB データを送信しないようにサーバに要求します。 この機能はデフォルトにより有効になっています。

パラメータとしての少量の LOB データの送信

少量の LOB データをストアドプロシージャの入力パラメータまたは準備された SQL 文のパラメータとして送信するプロセスは、LOB 以外のパラメータを送信するプロセスと同じです。

少量の LOB データを送信するには、コマンドとデータ用のメモリを割り付け、**ct_param()** または **ct_setparam()** を使用してこれらをサーバに直接送信します。

`text`、`unitext`、または `image` パラメータを使用する場合は、CS_DATAFMT 構造体の *maxlength* フィールドを設定する必要があります。 *maxlength* 値は、すべての LOB データがサーバに一度に送信されるか、ストリーミングされるかを示します。 *maxlength* がゼロより大きい場合、LOB データは 1 つのまとまりで送信されます。 *maxlength* が CS_UNUSED に設定されている場合、LOB データはデータをまとまりで送信するための **ct_send_data()** 呼び出しのループを使用して、ストリームで送信されます。 ゼロのまとまりの長さは、データストリームの終わりを示します。

例 1
少量の LOB データをストアドプロシージャの入力パラメータとして送信します。

```
CS_TEXT textvar[50];
CS_DATAFMT paramfmt;
CS_INT datalen;
CS_SMALLINT ind;

...
ct_command(cmd, CS_RPC_CMD, ...)

/*
** Clear and setup the CS_DATAFMT structure, then pass
** each of the parameters for the RPC.
*/
memset(&paramfmt, 0, sizeof(paramfmt));

/*
** First parameter, an integer.
*/
strcpy(paramfmt.name, "@intparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_INT_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;
ct_param(cmd, &paramfmt, (CS_VOID *)&intvar,
   sizeof(CS_INT), ind))

/*
** Second parameter, a (small) text parameter.
*/
```

```
strcpy((CS_CHAR *)textvar, "The Open Client and Open
   Server products both include Bulk-Library and
   CS-Library. ");
datalen = sizeof(textvar);
strcpy(paramfmt.name, "@textparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_TEXT_TYPE;
paramfmt.maxlength = EX_MYMAXTEXTLEN;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;
ct_setparam(cmd, &paramfmt, (CS_VOID *)&textvar,
   &datalen, &ind);

ct_send(cmd);
ct_results(cmd, &res_type);

...
```

例 2
少量の LOB データを、準備文を使用して送信します。

```
/*
** Prepare the sql statement.
*/
sprintf(statement, "select title_id from mybooks where
   title like (?) ");

/*
** Send the prepared statement to the server
*/
ct_dynamic(cmd, CS_PREPARE, "my_dyn_stmt", CS_NULLTERM,
   statement, CS_NULLTERM);

ct_send(cmd);
handle_results(cmd);

/*
** Prompt user to provide a value for title
*/
printf("Enter title id value - enter an X if you wish
   to stop: ¥n");

while (toupper(title[0]) != 'X')
{
   printf("Retrieve detail record for title: ?");
   fgets(mytexttitle, 50,  stdin);

   /*
   ** Execute the dynamic statement.
   */

   ct_dynamic(cmd, CS_EXECUTE, "my_dyn_stmt",
   CS_NULLTERM, NULL, CS_UNUSED);
```

```
    /*
   ** Define the input parameter
   */

   memset(&data_format, 0, sizeof(data_format));
   data_format.status = CS_INPUTVALUE;
   data_format.namelen = CS_NULLTERM ;
   data_format.datatype = CS_TEXT_TYPE;
   data_format.format = CS_FMT_NULLTERM;
   data_format.maxlength = EX_MYMAXTEXTLEN;
   ct_setparam(cmd, &data_format,
      (CS_VOID *)mytexttitle, &datalen, &ind);

ct_send(cmd);
handle_results(cmd);
...
}
```

パラメータとしての大量の LOB データの送信

大量の LOB データは、リソースを効率的に管理するために、ストリームでサーバに送信されます。 **ct_send_data()** をループ内で使用して、まとまったデータをサーバに送信します。

LOB データパラメータをまとまりで送信するには、次の設定を使用してパラメータを定義します。

- CS_DATAFMT 構造体の *datatype* フィールドを CS_TEXT_TYPE、CS_UNITEXT_TYPE、または CS_IMAGE_TYPE に設定します。
- CS_DATAFMT 構造体の *maxlength* フィールドを CS_UNUSED に設定します。
- **ct_param()** 関数の **data* ポインタ引数を NULL に設定します。
- **ct_param()** 関数の `datalen` 引数を 0 に設定します。

例 1

大量の LOB データパラメータをまとまりで送信します。

```
#define BUFSIZE 2048

int fp;
char sendbuf[BUFSIZE]

/*
** Clear and setup the CS_DATAFMT structure, then pass
** each of the parameters for the RPC.
*/
memset(&paramfmt, 0, sizeof(paramfmt));
strcpy(paramfmt.name, "@intparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_INT_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;
```

```
ct_param(cmd, &paramfmt, (CS_VOID *)&intvar,
   sizeof(CS_INT), 0))

/*
** Text parameter, sent as a BLOB.
*/
strcpy(paramfmt.name, "@textparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_TEXT_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;

/*
** Although the actual data will not be sent here, we
** must invoke ct_setparam() for this parameter to send
** the parameter format (paramfmt) information to the
** server, prior to sending all parameter data.
** Set *data to NULL and datalen = 0, to indicate that
** the length of text data is unknown and we want to
** send it in chunks to the server with ct_send_data().
*/
ct_setparam(cmd, &paramfmt, NULL, 0, 0);

/*
** Another LOB parameter (image), sent in chunks with
** ct_send_data()
*/
strcpy(paramfmt.name, "@textparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_IMAGE_TYPE;
paramfmt.maxlength = CS_UNUSED;
paramfmt.status = CS_INPUTVALUE;
paramfmt.locale = NULL;

/*
** Just like the previous parameter, invoke
** ct_setparam() for this parameter to send the
** parameter format.
*/
ct_setparam(cmd, &paramfmt, NULL, 0, 0);

/*
** Repeat this sequence of filling paramfmt and calling
** ct_param() for any subsequent parameter that needs
** to be sent before finally sending the data chunks for
** the LOB type parameters.
*/
strcpy(paramfmt.name, "@any_otherparam");
paramfmt.namelen = CS_NULLTERM;
paramfmt.datatype = CS_MONEY_TYPE;
...

/*
** Send the first LOB (text) parameter in chunks of
```

```
** 'BUFSIZE' to the server. We must end with a 0 bytes
** write to indicate the end of the current parameter.
*/
fp = open("huge_text_file", O_RDWR, 0666);

do
{
   num_read = read(fp, sendbuf, BUFSIZE);
   ct_send_data(cmd, (CS_VOID *)sendbuf, num_read);
}  while (num_read != 0);

/*
** Repeat the ct_send_data() loop for the next LOB
** parameter.
** Send the image parameter in chunks of 'BUFSIZE'
** to the server as well and end with a 0 bytes write
** to indicate the end of the current parameter.
*/
fp = open("large_image_file", O_RDWR, 0666);
do
{
   num_read = read(fp, sendbuf, BUFSIZE);
   ct_send_data(cmd, (CS_VOID *)sendbuf, num_read);
} while (num_read != 0);

/*
** Ensure that all the data is flushed to the server
*/
ct_send(cmd);
```

例2

SQL 準備文を使用して、ストリームとして LOB データを送信します。

```
/*
** Prepare the sql statement.
*/
sprintf(statement, "select title_id from mybooks
   where title like (?) ");

/*
** Send the prepared statement to the server
*/
ct_dynamic(cmd, CS_PREPARE, "mydyn_stmt", CS_NULLTERM,
   statement, CS_NULLTERM);

ct_send(cmd);
handle_results();

/*
** Promt user to provide a value for title
*/
printf("Enter title id value - enter an X if you wish
   to stop: ¥n");

while (toupper(myblobtitle[0]) != 'X')
```

```
{
   printf("Retrieve detail record for title: ?");
   fgets(myblobtitle, 50,  stdin);

   /*
   ** Execute the dynamic statement.
   */
   ct_dynamic(cmd, CS_PREPARE, "my_dyn_stmt",
   CS_NULLTERM, statement, CS_NULLTERM);

   /*
   ** Define the input parameter, a TEXT type that we
   want to send in chunks to the server.
   */
   memset(&data_format, 0, sizeof(data_format)) ;
   data_format.namelen = CS_NULLTERM ;
   data_format.datatype = CS_TEXT_TYPE;
   data_format.maxlength = CS_UNUSED;
   data_format.status = CS_INPUTVALUE;
   ct_setparam(cmd, &data_format, NULL, 0, 0);

   /*
   ** Send the 'myblobtitle' data in chunks of
   ** 'CHUNKSIZE' to the server with ct_send_data() and
   ** end with 0 bytes to indicate the end of data for
   ** this parameter. This is just an example to show
   ** how chunks can be sent. (myblobtitle[] is used as
   ** a simple example. This could also be replaced by
   ** large file which would be read in chunks from disk
   ** for example).
   */
   bytesleft = strlen(myblobtitle);
   bufp = myblobtitle;

   do
   {
      sendbytes = min(bytesleft, CHUNKSIZE);
      ct_send_data(cmd, (CS_VOID *)bufp, sendbytes);
      bufp += bufp + sendbytes;
      bytesleft -= sendbytes;
   }  while (bytesleft > 0)

   /*
   ** End with 0 bytes to indicate the end of current
   data.
   */
   ct_send_data(cmd, (CS_VOID *)bufp, 0);

   /*
   ** Insure that all the data is sent to the server.
   */
   ct_send(cmd);
   handle_results(cmd)
   ...
}
```

```
/*
** Deallocate the prepared statement and finish up.
*/

ct_dynamic(cmd, CS_DEALLOC, "my_dyn_stmt", CS_NULLTERM,
   NULL, CS_UNUSED);

ct_send(cmd);
handle_results(cmd);
```

Open Server での LOB パラメータの取得

完全な LOB パラメータデータを **srv_xferdata** を使用して一度に取得するか、新しい **srv_get_data** ルーチンを使用してまとまりで取得します。

Open Server は、パラメータの長さが CS_UNUSED に設定されている場合に、LOB パラメータをまとまりで取得します。 「srv_get_data」を参照してください。

例
LOB パラメータの説明を取得します。

```
/*
** Retrieve the description of the parameters coming
** from client
*/

for (paramnum = 1; paramnum <= numparams; paramnum++)
{
   /*
   ** Get a description of the parameter.
   */
   ret = srv_descfmt(spp, CS_GET, SRV_RPCDATA,
      paramnum, &(paramfmtp[paramnum - 1]));

   /*
   ** Allocate space for the parameters and bind the
   ** data.
   */
   if (paramfmtp[paramnum-1].maxlength >= 0)
   {
      if (paramfmtp[paramnum-1].maxlength > 0)
      {
         data[paramnum-1] = calloc(1,
            paramfmtp[paramnum-1].maxlength);
      }
      else
      {
         ind[paramnum-1] = CS_NULLDATA;
      }
   }
   else
   {
      /*
      ** Allocate a large size buffer for BLOB data
```

```
      ** (which length is unknown yet)
      */
      blobbuf[blobnum] = malloc(BUFSIZE);
      blobnum++;
   }

   srv_bind(spp, CS_GET, SRV_RPCDATA, paramnum,
      &(paramfmtp[paramnum-1]), data[paramnum-1],
      &(len[paramnum-1]), &(ind[paramnum-1]))

   /*
   ** For every LOB parameter, call srv_get_data() in
   ** a loop as long as it succeeds
   /*
   for (i = 0; i < blobnum ; i++)
   {
      bufp = blobbuf[i];
      bloblen[i] = 0;
      do
      {
         ret = srv_get_data(spp, bufp, BUFSIZE,
            &outlen);
         bufp += outlen;
         bloblen[i] += outlen;
      } while (ret == CS_SUCCEED);

      /*
      ** Check for the correct return code
      */
      if (ret != CS_END_DATA)
      {
         return CS_FAIL;
      }

   }

   /*
   ** And receive remaining client data srv_xferdata()
   */
   ret = srv_xferdata(spp, CS_GET, SRV_RPCDATA);
}
```

srv_get_data

text、unitext、または image パラメータストリームをまとまった単位でクライアントから読み込みます。

構文

```
CS_RETCODE srv_get_data(spp, bp, buflen, outlenp)

SRV_PROC *spp;
CS_BYTE *bp;
CS_INT buflen;
CS_INT *outlenp;
```

パラメータ

- *spp* - 内部スレッド制御構 f 造体を指すポインタ。
- *bp* - クライアントからのデータが格納されているバッファを指すポインタ。
- *buflen* - **bp* ポインタのサイズ。 このサイズは、連続して転送される 1 つのデータのまとまりをバイト数で示したものです。
- *outlenp* - 出力パラメータ。*outlenp* には **bp* バッファに読み込まれるバイト数が含まれます。

戻り値

- CS_SUCCEED - **srv_get_data()** が正常に実行されました。保留中のデータがさらにあります。
- CS_FAIL - ルーチンが失敗しました。
- CS_END_DATA - **srv_get_data()** が `text`、`unitext`、または `image` パラメータ全体の読み取りを完了しました。

jConnect および Adaptive Server Enterprise のドライバおよびプロバイダに対応する SDK 15.7 機能

SDK 15.7 で jConnect、Adaptive Server Enterprise ODBC ドライバ、Adaptive Server Enterprise OLE DB プロバイダ、Adaptive Server Enterprise ADO.NET Data Provider に導入された新機能について説明します。

ODBC ドライバのバージョン情報ユーティリティ

odbcversion ユーティリティは、ODBC ドライバに関する情報を表示します。

構文
odbcversion `-version|-fullversion|-connect` *dsn userid password*

パラメータ
`-version`

ODBC ドライバの単純な数値のバージョン文字列を表示します。

`-fullversion`

ODBC ドライバの冗長バージョン文字列を表示します。

`-connect` *`dsn userid password`*

Adaptive Server のバージョンと、その Adaptive Server にインストールされている ODBC および OLEDB MDA スクリプトのバージョンを表示します。 このパラメータには 3 つの変数が必要です。 それらは、Adaptive Server のデータソース名である *dsn*、および Adaptive Server への接続に使用されるユーザ ID とパスワードです。

例
Adaptive Server への接続に使用される ODBC ドライバの単純な数値のバージョン文字列を取得します。

```
odbcversion -version
```

数値バージョン文字列が返されます。

```
15.05.00.1015
```

使用法
パラメータが指定されていない場合、**odbcversion** ユーティリティは有効なパラメータのリストを表示します。

SupressRowFormat2 接続文字列プロパティ

Adaptive Server Enterprise ODBC ドライバ 15.7、Adaptive Server Enterprise OLE DB プロバイダ 15.7、および Adaptive Server Enterprise ADO.NET Data Provider 15.7 に、SupressRowFormat2 接続文字列プロパティを使用することができます。これにより、Adaptive Server が可能な限り TDS_ROWFMT2 バイトシーケンスではなく、TDS_ROWFMT バイトシーケンスを使用してデータを送信するように強制することができます。

TDS_ROWFMT は、カタログ、スキーマ、テーブル、カラム情報を含む TDS_ROWFMT2 よりも少ないデータを含んでおり、多くの小さな **select** オペレーションでより優れたパフォーマンスを実現します。 SupressRowFormat2 が 1 に設定されている場合、サーバは縮小された結果セットメタデータを送信するので、一部の情報がクライアントプログラムで使用できなくなります。 欠如しているメタデータにアプリケーションが依存している場合、このプロパティは有効にしないでください。

値:

- 0 - デフォルト値。TDS_ROWFMT2 は抑制されません。
- 1 - サーバがデータをできる限り TDS_ROWFMT で送信するように強制します。

例
この接続文字列は、ADO.NET Data Provider との接続において、サーバがデータをできる限り TDS_ROWFMT で送信するように強制します。

```
Data Source='myASE';Port=5000;Database=myDB;
Uid=myUID;Pwd=myPWD;SupressRowFormat2=1
```

UseCursor プロパティの機能強化

Adaptive ServerEnterprise ODBC ドライバの UseCursor 接続文字列プロパティを使用すると、結果セットを生成する SQL 文に対して、サーバ側のカーソルがどのように使用されるかを判断できます。

このプロパティは、サーバ側のカーソル (値 2) を作成する文をクライアントアプリケーションが制御できるように更新されています。

値:

- 0 - デフォルト値。 サーバ側のカーソルは使用されません。
- 1 - サーバ側のカーソルが、結果セットを生成するすべての文に使用されます。
- 2 - **SQLSetCursorName** ODBC 関数が呼び出されたときにのみ、結果セットを生成する文にサーバ側のカーソルが使用されます。 カーソルはより多くのリソースを使用するので、この設定を使用すると、サーバ側カーソルから恩恵を得られる文のみにその使用を制限できます。

ODBC ドライバマネージャのトレースなしのロギング

Adaptive Server Enterprise ODBC ドライバ 15.7 では、ODBC ドライバマネージャのトレースを使用せずに ODBC API の呼び出しをロギングできます。 これは、ドライバマネージャが使用されないか、トレースをサポートしていないプラットフォームでドライバマネージャを実行している場合に便利です。

この機能を Microsoft Windows で有効にするには、LOGCONFIGFILE 環境変数または Microsoft Windows レジストリを使用します。 Linux で有効にするには、LOGCONFIGFILE を使用します。

LOGCONFIGFILE を使用するときは、環境変数を ODBC ログの設定ファイルのフルパスに設定します。 LOGCONFIGFILE は既存のレジストリエントリをすべて上書きします。

Microsoft Windows レジストリを使用する場合は、LogConfigFile というエントリを `HKEY_CURRENT_USER¥Software¥Sybase¥ODBC` または `HKEY_LOCAL_MACHINE¥Software¥Sybase¥ODBC` に作成し、その値を ODBC ログの設定ファイルのフルパスに設定します。 例を示します。

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER¥Software¥Sybase¥ODBC]
"LogConfigFile"="c:¥¥temp¥¥odbclog.properties"
```

ロギングを無効にするには、*LogConfigFile* の値を削除するか、名前を変更します。

注意： `HKEY_CURRENT_USER` に指定されている値は、`HKEY_LOCAL_MACHINE` に設定されている値を上書きします。

ログ設定ファイル

設定ファイルは ODBC ログファイルのフォーマットと場所を制御します。

この例では、太字の行がログファイルの保存先を指定します。

```
log4cplus.rootLogger=OFF, NULL
```

```
log4cplus.logger.com.sybase.dataaccess.odbc.api=TRACE, ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.parameter=TRACE,
ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api.parameter=false

log4cplus.logger.com.sybase.dataaccess.odbc.api.returncode=TRACE,
ODBCTRACE
log4cplus.additivity.com.sybase.dataaccess.odbc.api.returncode=fals
e

log4cplus.appender.NULL=log4cplus::NullAppender

log4cplus.appender.ODBCTRACE=log4cplus::FileAppender
log4cplus.appender.ODBCTRACE.File=c:¥temp¥odbc.log
log4cplus.appender.ODBCTRACE.layout=log4cplus::PatternLayout
log4cplus.appender.ODBCTRACE.ImmediateFlush=true
log4cplus.appender.ODBCTRACE.layout.ConversionPattern=%d{%H:%M:%S.
%q} %t %p
   %-25.25c{2} %m%n
```

jConnect setMaxRows の機能強化

JDBC プログラムでは **Statement.setMaxRows(int max)** を使用して、結果セットが返すロー数を制限します。 jConnect 7.0 以前では、**select**、**insert**、**update**、**delete** 文の結果が制限に対してカウントされます。

JDBC の仕様との一貫性を保つために、jConnect 7.07 には SETMAXROWS_AFFECTS_SELECT_ONLY 接続プロパティが導入されています。このプロパティは、true (デフォルト) に設定されている場合に **SELECT** 文が返すローのみを制限します。

Adaptive Server 15.5 以前に接続している場合、SETMAXROWS_AFFECTS_SELECT_ONLY は無視されます。

TDS ProtocolCapture

Adaptive Server Enterprise ODBC ドライバ 15.7 には、ODBC アプリケーションと Adaptive Server 間で交換される Tabular Data Stream™ (TDS) パケットを受信するためのファイルを指定する ProtocolCapture 接続文字列プロパティが導入されています。

ProtocolCapture の設定はすぐに反映されるので、接続の確立中に交換された TDS パケットはファイルプレフィクスを使用して生成された一意のファイル名に書き込まれます。 TDS パケットは、接続している期間中、ファイルに書き込まれま

す。 TDS 取得ファイルを解釈するには、Ribo および他のプロトコル変換ツールを使用できます。

たとえば、`tds_capture` を TDS トレースログファイルプレフィクスとして指定するには、次のように入力します。

```
Driver=AdaptiveServerEnterprise;server=server1;
port=port1;UID=sa;PWD=;ProtocolCapture=tds_capture;
```

最初の接続は `tds_capture0.tds` を生成し、2 番目の接続は `tds_capture1.tds` を生成します (以下同様)。

バインドパラメータ配列を使用しない ODBC データのバッチ処理

同じ SQL 文が異なるパラメータ値に対して実行される場合、クライアントアプリケーションは通常パラメータ配列をバインドし、**SQLExecute**、**SQLExecuteDirect**、**SQLBulkOperations** を使用してパラメータの各セットを実行します。

配列を SQL パラメータにバインドする際は、配列のメモリが割り付けられ、データがすべて配列にコピーされてから、SQL 文が実行されます。 これにより、大量のトランザクションを処理する場合にメモリとリソースの使用効率が低下することがあります。

Adaptive Server Enterprise ODBC ドライバ 15.7 では、クライアントアプリケーションは **SQLExecute** を使用して、パラメータを配列としてバインドせずに、パラメータを Adaptive Server にバッチで送信します。 **SQLExecute** は、最後のパラメータのバッチが送信および処理されるまで、SQL_BATCH_EXECUTING を返します。 最後のパラメータのバッチが処理されると、実行ステータスが返されます。

SQLRowCount の呼び出しは、最後の **SQLExecute** 文が完了した後でのみ有効です。

データバッチの管理

Adaptive Server に送信されるパラメータのバッチを管理するために、Sybase 固有の接続属性である SQL_ATTR_BATCH_PARAMS が導入されています。 SQL_ATTR_BATCH_PARAMS は **SQLSetConnectAttr** を使用して設定します。

値:

- SQL_BATCH_ENABLED - パラメータをバッチ処理するように Adaptive Server Enterprise ODBC ドライバに伝えます。 この状態では、現在処理中の文、つまり SQL_ATTR_BATCH_PARAMS を SQL_BATCH_ENABLED に設定した後に **SQLExecute** によって実行される最初の文以外の文が接続で実行されると、エラーが送信されます。

- SQL_BATCH_LAST_DATA - 次のパラメータのバッチが最後のバッチであり、パラメータにデータが含まれていることを指定します。
- SQL_BATCH_LAST_NO_DATA - 次のパラメータのバッチが最後のバッチであり、これらのパラメータを無視するように指定します。
- SQL_BATCH_CANCEL - バッチをキャンセルし、トランザクションをロールバックするように Adaptive Server Enterprise ODBC ドライバに伝えます。ロールバックできるのは、コミットされていないトランザクションのみです。
- SQL_BATCH_DISABLED - (デフォルト値) Adaptive Server Enterprise ODBC ドライバは、最後のパラメータのバッチを処理した後に、この状態に戻ります。SQL_ATTR_BATCH_PARAMS をこの値に手動で設定することはできません。

データバッチの管理に関する例

データバッチの管理に関する 2 つの例を示します。

例 1

パラメータ配列をバインドせずにパラメータのバッチをサーバに送信します。

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to start
// the batch
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
   (SQLPOINTER)SQL_BATCH_ENABLED, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Bind the parameters. This can be done once for the entire batch
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT,
   SQL_C_LONG, SQL_INTEGER, l1, 0, &c1, l1, &l1);
sr = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT,
SQL_C_CHAR, SQL_LONGVARCHAR, l2, 0, buffer, l2, &l2);
}

// Run a batch of 10 for (int i = 0; i < 10; i++)
{
   c1 = i;
   memset(buffer, 'a'+i, l2);
   sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
   printError(sr, SQL_HANDLE_STMT, stmt);
}
```

例 2

バッチを終了して閉じます。

```
// Setting the SQL_ATTR_BATCH_PARAMS attribute to end
// the batch
sr = SQLSetConnectAttr(dbc, SQL_ATTR_BATCH_PARAMS,
  (SQLPOINTER)SQL_BATCH_LAST_NO_DATA, SQL_IS_INTEGER);
printError(sr, SQL_HANDLE_DBC, dbc);

// Call SQLExecDirect one more time to close the batch
// - Due to SQL_BATCH_LAST_NO_DATA, this will not
// process the parameters
```

```
sr = SQLExecDirect(stmt, insertStmt, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
```

ODBC データのバッチ処理に関する考慮事項

ODBC データのバッチ処理機能に関するいくつかの考慮事項を示します。

- この機能は、結果セットを返さないか、出力パラメータを持たない文およびストアドプロシージャのみをサポートしています。
- 非同期モードはサポートされていません。 バッチモード中に、現在バッチ処理している文以外の文をアプリケーションが同じ接続で実行することはできません。
- SQL_DATA_AT_EXEC はサポートされていません。 LOB パラメータは通常のパラメータとしてバインドします。
- パラメータ配列をバインドせずにデータをバッチ処理する場合、SQL_ATTR_PARAM_STATUS_PTR が設定されているときに、Adaptive Server Enterprise ODBC ドライバでは SQL_ATTR_PARAMSET_SIZE ではなく、SQLSetStmtAttr の StringLength パラメータから複数の配列要素を取得します。

jConnect での最適化されたバッチ処理

jConnect for JDBC 7.07 では **PreparedStatement** オブジェクトのバッチオペレーションを高速化するための内部アルゴリズムを実装しています。

このアルゴリズムは HOMOGENEOUS_BATCH 接続プロパティが true に設定されている場合に呼び出されます。

注意： 同種バッチ処理は、この機能をサポートしているサーバにクライアントアプリケーションが接続している場合にのみ使用できます。 Adaptive Server Enterprise 15.7 には同種バッチ処理のサポートが導入されています。

次の例は、**addBatch** および **executeBatch** メソッドを使用した **PreparedStatement** バッチオペレーションを示します。

```
String sql = "update members set lastname = ? where member_id = ?";
prep_stmt = connection.prepareStatement(sql);
prep_stmt.setString(1, "Forrester");
prep_stmt.setLong(2, 45129);
prep_stmt.addBatch();
prep_stmt.setString(1, "Robinson");
prep_stmt.setLong(2, 45130);
prep_stmt.addBatch();
prep_stmt.setString(1, "Servo");
prep_stmt.setLong(2, 45131);
prep_stmt.addBatch();
prep_stmt.executeBatch();
```

ここで、connection は接続インスタンス、prep_stmt は準備文インスタンス、? は準備文のパラメータ用プレースホルダを表します。

LOB カラムの同種バッチ処理

HOMOGENEOUS_BATCH および ENABLE_LOB_LOCATORS プロパティが true に設定されている場合、クライアントアプリケーションが同じバッチ内で LOB 準備文と LOB 以外の準備文の setter メソッドを混合することはできません。

たとえば、以下は無効です。

```
String sql = "update members SET catchphrase = ? WHERE member_id
= ?";
prep_stmt = connection.prepareStatement(sql);
prep_stmt.setString(1, "Push the button, Frank!");
prep_stmt.setLong(2, 45129);
prep_stmt.addBatch();
Clob myclob = con.createClob();
myclob.setString(1, "Hi-keeba!");
prep_stmt.setClob(1, myclob);
prep_stmt.setLong(2, 45130);
prep_stmt.addBatch();
pstmt.executeBatch();
```

ここでのキャッチフレーズは、カラムのデータ型 text です。 **setString** メソッドと **setClob** メソッドが同じカラムに対して同じバッチで使用されているので、このコードは失敗します。

ローを累積しない jConnect パラメータのバッチ処理

jConnect for JDBC 7.07 には SEND_BATCH_IMMEDIATE 接続プロパティが追加されています。

これが true に設定されている場合、jConnect は **PreparedStatement.addBatch()** を呼び出した直後に現在のローのパラメータを送信します。 これにより、クライアントのメモリの使用が最小化され、サーバにはバッチパラメータを処理するための時間がより多く与えられます。

デフォルトの SEND_BATCH_IMMEDIATE 値は false で、これが設定されている場合はこれまでと同様に、**PreparedStatement.executeBatch()** を呼び出した後にのみバッチパラメータを送信するように jConnect に示します。

過去のエラーを実行するための jConnect バッチ更新の機能強化

jConnect for JDBC 7.07 には EXECUTE_BATCH_PAST_ERRORS 接続プロパティが導入されています。これが true に設定されている場合は、個々の文の実行中に遭遇した致命的でないエラーをバッチ更新オペレーションで無視し、バッチ更新を完了することができます。

デフォルトの false に設定されている場合は、以前のバージョンのように、エラーに遭遇するとバッチ更新は中止されます。

バッチ更新の使用法とその結果の解釈については、『jConnect for JDBC プログラマーズリファレンス』を参照してください。

カーソルクローズ時のロックの解放のサポート

Adaptive Server 15.7 では、**release_locks_on_close** オプションを含めるように **declare cursor** 構文が拡張されています。このオプションは、カーソルのクローズ時に独立性レベル 2 および 3 でカーソルの共有ロックを解放します。 Adaptive Server Enterprise ODBC ドライバ 15.7 と jConnect for JDBC 7.07 は、release-lock-on-close セマンティックをサポートしています。

この機能を Adaptive Server Enterprise ODBC Driver 接続で作成されたすべての読み取り専用カーソルに適用するには、ReleaseLocksOnCursorClose 接続プロパティを 1 に設定します。 ReleaseLocksOnCursorClose のデフォルト値は 0 です。

jConnect for JDBC 接続で適用するには、RELEASE_LOCKS_ON_CURSOR_CLOSE 接続プロパティを true に設定します。 デフォルトの RELEASE_LOCKS_ON_CURSOR_CLOSE 値は false です。

これらの接続プロパティによって適用された設定は静的で、接続の確立後に変更することはできません。 この設定は、**release_locks_on_close** をサポートしているサーバに接続されている場合にのみ有効です。

release_locks_on_close の詳細については、『ASE リファレンスマニュアル: コマンド』を参照してください。

select for update のサポート

Adaptive Server 15.7 は、同じトランザクション内の後続の更新用にローをロックできる **select for update** と、更新可能なカーソル用の排他ロックをサポートしています。

『ASE Transact-SQL ユーザーズガイド』の「第 2 章 クエリ: テーブルからのデータの選択」を参照してください。

この機能は、**for update** 句が **select** 文に追加されたときと、クライアント内で開いている更新可能なカーソルに追加されたときにクライアントで自動的に使用可能になります。 更新可能なカーソルの作成の詳細については、『ASE ODBC ドライバユーザーズガイド』および『jConnect for JDBC プログラマーズリファレンス』を参照してください。

拡張された可変長ローのサポート

16K 論理ページサイズを使用するように設定されている Adaptive Server 15.7 より前のバージョンでは、可変長カラムが行の先頭から 8191 バイトを超えた位置から始まっている場合、可変長ローを含むデータオンリーロック (DOL) テーブルは作成できませんでした。 この制限は、Adaptive Server 15.7 以降では取り除かれています。

『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。

ODBC および JDBC クライアントがこの機能を使用するにあたって特別な設定を行う必要はありません。 長い DOL ローを受信するように設定されている Adaptive Server バージョン 15.7 に接続すると、これらのクライアントは長いオフセットを使用して自動的にレコードを挿入します。 クライアントは、長い DOL ローを以前のバージョンの Adaptive Server に送信しようとするか、長い DOL ローオプションが無効になっている Adaptive Server 15.7 に送信しようとすると、エラーメッセージを受信します。

非実体化カラムのサポート

Adaptive Server Enterprise ODBC ドライバ 15.7 のバルク挿入ルーチンは、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るように強化されています。

以前のバージョンの Adaptive Server Enterprise ODBC ドライバでは、テーブル定義にマテリアライズされていないカラムが含まれている場合、Adaptive Server へのデータのバルク挿入は実行できません。 以前のバージョンの Adaptive Server Enterprise ODBC ドライバでマテリアライズされていないカラムにバルク挿入を実行しようとすると、エラーが発生します。

ロー内とロー外の LOB 記憶領域のサポート

Adaptive Server 15.7 では、ロー内に格納するようにマークされている LOB カラムは、ロー全体を格納するのに十分なメモリがある場合、ロー内に格納されます。

ロー内のカラムが更新されたために、ローのサイズが定義されている制限を超えた場合、ロー内に格納されている LOB カラムは制限を満たすためにロー外に移動されます。 『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。

Adaptive Server Enterprise ODBC Driver 15.7 および jConnect for JDBC 7.07 のバルク挿入ルーチンは、Adaptive Server の `text`、`image`、`unitext` の LOB カラムのロー内およびロー外の記憶領域をサポートしています。 以前のクライアントバージョンからのバルク挿入ルーチンでは、LOB カラムは常にロー外に格納されます。

ストアドプロシージャパラメータとしてのラージオブジェクト

ストアドプロシージャ入力パラメータとして LOB データを渡す機能も Adaptive Server 15.7 に導入されています。

jConnect for JDBC 7.07、Adaptive Server Enterprise ODBC ドライバ 15.7、Adaptive Server Enterprise OLE DB プロバイダ 15.7、Adaptive Server Enterprise ADO.NET Data Provider 15.7 は、ストアドプロシージャ内での入力パラメータ、およびパラメータマーカデータ型としての `text`、`unitext`、`image` の使用をサポートしています。

ラージオブジェクトのロケータのサポート

jConnect for JDBC 7.07 および Adaptive Server Enterprise ODBC ドライバ 15.7 は、ラージオブジェクト (LOB) ロケータをサポートしています。

LOB ロケータには、データ自体ではなく、LOB データへの論理ポインタが含まれているため、Adaptive Server とそのクライアント間のネットワークを通過するデータの量が削減されます。 LOB ロケータに対するサーバのサポートは、Adaptive Server 15.7 に導入されています。

jConnect for JDBC 7.07 では、LOB ロケータをサポートする Adaptive Server に接続しており、**autocommit** がオフになっている場合に、サーバ側のロケータを使用して LOB データにアクセスします。 それ以外の場合、jConnect はクライアント側で LOB データを実体化します。 クライアント側で実体化された LOB データを完全な LOB API で使用することはできますが、データ量が大きくなるため、LOB ロケータを使用した場合よりも API のパフォーマンスが低下することがあります。

Adaptive Server Enterprise ODBC ドライバ 15.7 クライアントは、LOB ロケータをサポートしている Adaptive Server に接続していない限り、LOB ロケータを使用できません。

注意： LOB ロケータを使用している場合、各ローに LOB データを含む大きな結果セットを取得すると、アプリケーションのパフォーマンスに影響が及ぶ場合があります。 Adaptive Server は LOB ロケータを結果セットの一部として返します。 LOB データを取得するには、jConnect と ODBC ドライバが残りの結果セットをキャッシュに格納する必要があります。 結果セットは小さいサイズを保持するか、カーソルのサポートを有効にしてキャッシュに格納するデータのサイズを制限することをおすすめします。

jConnect for JDBC のサポート

LOB ロケータのサポートを有効にするには、ENABLE_LOB_LOCATORS 接続プロパティを true に設定して Adaptive Server への接続を確立します。

有効にすると、クライアントアプリケーションは **java.sql** パッケージの **Blob**、**Clob**、**NClob** クラスを使用して、ロケータにアクセスできるようになります。

注意： LOB ロケータと **autocommit** の両方が有効になっている場合、jConnect では接続されている Adaptive Server でのサポートが可能であっても、クライアント側でマテリアライズされている LOB ロケータに自動的に切り替えます。 これにより、クライアントが使用するメモリが増加し、パフォーマンスが低下する場合が

あります。 したがって、**autocommit off** の状態で LOB ロケータを使用することをおすすめします。

Blob、**Clob**、**NClob** クラスの詳細については、Java のマニュアルを参照してください。

Adaptive Server Enterprise ODBC ドライバのサポート

LOB ロケータのサポートを有効にするには、EnableLOBLocator 接続プロパティを true に設定して Adaptive Server への接続を確立します。

EnableLOBLocator がデフォルト値である 0 に設定されている場合、ODBC ドライバは LOB カラムのロケータを取得できません。 LOB ロケータを有効にする場合は、接続を **autocommit off** に設定してください。

`sybasesqltypes.h` ファイルをプログラムに含める必要もあります。 `sybasesqltypes.h` ファイルは ODBC インストールディレクトリの下の `include` ディレクトリに含まれています。

ロケータをサポートするための ODBC データ型のマッピング

Adaptive Server ロケータのデータ型に対する ODBC データ型のマッピングを以下の表に示します。

ASE データ型	ODBC SQL 型	ODBC C 型
`text_locator`	SQL_TEXT_LOCATOR	SQL_C_TEXT_LOCATOR
`image_locator`	SQL_IMAGE_LOCATOR	SQL_C_ IMAGE_LOCATOR
`unitext_locator`	SQL_UNITEXT_LOCATOR	SQL_C_ UNITEXT_LOCATOR

サポートされている変換

Adaptive Server ロケータのデータ型に対してサポートされている変換を以下の表に示します。

	SQL_C_TEXT_ LOCATOR	SQL_C_IMAGE_ LOCATOR	SQL_C_UNI-TEXT_LOCATOR
SQL_TEXT_LOCATOR	X		
SQL_IMAGE_LOCATOR		X	
SQL_UNITEXT_LOCATOR			X
SQL_LONGVARCHAR			
SQL_WLONGVARCHAR			
SQL_LONGVARBINARY			

	SQL_C_TEXT_LOCATOR	SQL_C_IMAGE_LOCATOR	SQL_C_UNI-TEXT_LOCATOR
凡例: X = サポートされている変換			

LOB ロケータをサポートしているメソッド

ODBC API メソッドは LOB ロケータをサポートしています。

- **SQLBindCol** - *TargetType* には ODBC C ロケータの任意のデータ型を指定できます。
- **SQLBindParameter** - *ValueType* には ODBC C ロケータの任意のデータ型を指定できます。 *ParameterType* には ODBC SQL ロケータの任意のデータ型を指定できます。
- **SQLGetData** - *TargetType* には ODBC C ロケータの任意のデータ型を指定できます。
- **SQLColAttribute** - SQL_DESC_TYPE および SQL_DESC_CONCISE_TYPE 記述子は、ODBC SQL ロケータの任意のデータ型を返すことができます。
- **SQLDescribeCol** - データ型のポインタには、ODBC SQL ロケータの任意のデータ型を指定できます。

Microsoft の『ODBC API Reference』を参照してください。

プリフェッチされた LOB データの暗黙的変換

Adaptive Server Enterprise ODBC ドライバ 15.7 では、Adaptive Server が LOB ロケータを返した場合、**SQLGetData** および **SQLBindCol** を使用して、`text` ロケータ用の SQL_C_CHAR または SQL_C_WCHAR、あるいは `image` ロケータ用の SQL_C_BINARY にカラムをバインドすることで、基本となるプリフェッチされた LOB データを取得できます。

接続内のロケータを有効または無効にするには、SQL_ATTR_LOBLOCATOR 属性を設定します。 EnableLOBLocator が接続文字列内で指定されている場合、SQL_ATTR_LOBLOCATOR は EnableLOBLocator の値で初期化されます。それ以外の場合はデフォルト値である SQL_LOBLOCATOR_OFF に設定されます。 ロケータを有効にするには、属性を SQL_LOBLOCATOR_ON に設定します。 属性の値を設定するには **SQLSetConnectAttr** を使用し、属性の値を取得するには **SQLGetConnectAttr** を使用します。

SQLSetStatementAttr を使用して SQL_ATTR_LOBLOCATOR_FETCHSIZE を設定し、取得する LOB データのサイズを指定します。バイナリデータのサイズはバイト数で指定し、文字データのサイズは文字数で指定します。 デフォルト値の 0 は、プリフェッチされたデータが要求されないことを示し、-1 の値は LOB データ全体が取得されることを示します。

注意： 取得するカラムの基本となる LOB データのサイズが、設定されているプリフェッチされたデータのサイズを超えた場合は、ODBC クライアントがデータを直接取得しようとしたときにネイティブエラー 3202 が発生します。 これが発生した場合、クライアントは **SQLGetData** を呼び出すことで完全なデータを取得し、基本となるロケータを取得して、ロケータで利用できるオペレーションをすべて実行できます。

例 1

プリフェッチされたデータが完全な LOB 値を表しているときに、**SQLGetData** を使用して `image` ロケータを取得します。

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
(SQLPOINTER)SQL_LOCATOR_ON,
   0);

// Set size of prefetched LOB data
sr = SQLSetStatementAttr(stmt, SQL_ATTR_LOBLOCATOR_FETCHSIZE,
(SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);

printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);

sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

//Retrieve the binary data  (Complete Data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);
printError(sr, SQL_HANDLE_STMT, stmt);
```

```
//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);

SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
(SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

例 2

プリフェッチされたデータがトランケートされた LOB 値を表しているときに、**SQLGetData** を使用して `image` ロケータを取得します。

```
//Set Autocommit off
SQLRETURN sr;
sr = SQLSetConnectAttr(dbc, SQL_ATTR_AUTOCOMMIT,
   (SQLPOINTER)SQL_AUTOCOMMIT_OFF, 0);

//Enable LOB Locator for this exchange
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
   (SQLPOINTER)SQL_LOCATOR_ON, 0);

//Set size of prefetched LOB data
sr = SQLSetStatementAttr(stmt,
   SQL_ATTR_LOBLOCATOR_FETCHSIZE, (SQLPOINTER)32768, 0);

//Get a locator from the server
SQLLEN lLOBLen = 0;
Byte cBin[COL_SIZE];
SQLLEN lBin = sizeof(CBin);
unsigned char cLOC[SQL_LOCATOR_SIZE];
SQLLEN lLOC = sizeof(cLOC);

int id = 4;
SQLLEN l1 = sizeof(int);
SQLLEN idLen = sizeof(int);
sr = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, idLen,
   0, &id, idLen, &idLen);
printError(sr, SQL_HANDLE_STMT, stmt);

//Execute the select statement to return a locator
sr = SQLExecDirect(stmt, selectCOL_SQL, SQL_NTS);
printError(sr, SQL_HANDLE_STMT, stmt);
sr = SQLFetch(stmt);
printError(sr, SQL_HANDLE_STMT, stmt);

// Retrieve the binary data(Truncated data is returned)
sr = SQLGetData(stmt, 1, SQL_C_BINARY, cBin, lBin, &lBin);

if(sr == SQL_SUCCESS_WITH_INFO)
```

```
{
   SQLTCHAR errormsg[ERR_MSG_LEN];
   SQLTCHAR sqlstate[SQL_SQLSTATE_SIZE+1];
   SQLINTEGER nativeerror = 0;
   SQLSMALLINT errormsglen = 0;

   retcode = SQLGetDiagRec(handleType, handle, 1, sqlstate,
&nativeerror,
      errormsg, ERR_MSG_LEN, &errormsglen);

   printf("SqlState: %s Error Message: %s¥n", sqlstate, errormsg);

   //Handle truncation of LOB data; if data was truncated call
SQLGetData to
   // retrieve the locator.

   /* Warning returns truncated LOB data */
   if (NativeError == 32028) //Error code may change
   {
      BYTE ImageLocator[SQL_LOCATOR_SIZE];
      sr = SQLGetData(stmt, 1, SQL_C_IMAGE_LOCATOR, &ImageLocator,
         sizeof(ImageLocator), &Len);
      printError(sr, SQL_HANDLE_STMT, stmt);

      /*
         Perform locator specific calls using image Locator on a
separate
         statement handle if needed
      */
   }
}

//Cleanup
sr = SQLFreeStmt(stmt, SQL_UNBIND);
sr = SQLFreeStmt(stmt, SQL_RESET_PARAMS);
sr = SQLFreeStmt(stmt, SQL_CLOSE);

SQLEndTran(SQL_HANDLE_DBC, dbc,SQL_COMMIT);

//Disable LOB Locator for the future
sr = SQLSetConnectAttr(dbc, SQL_ATTR_LOBLOCATOR,
(SQLPOINTER)SQL_LOCATOR_OFF,
   0);
```

ロケータを使用した LOB のアクセスと操作

ODBC API は LOB ロケータを直接サポートしていません。 ODBC クライアントアプリケーションでは Transact-SQL 関数を使用して、ロケータに対するオペレーションを行い、LOB 値を操作する必要があります。 Adaptive Server Enterprise ODBC ドライバには、必要な Transact-SQL 関数の使用を助長するためのストアドプロシージャがいくつか導入されています。

LOB ロケータに対してはさまざまなオペレーションを実行できます。 パラメータの入出力値には、Adaptive Server がストアドプロシージャ定義に暗黙的に変換できる任意の型を指定できます。

ここに示す Transact-SQL コマンドおよび関数の詳細については、『ASE リファレンスマニュアル: ビルディングブロック』の「Transact-SQL 関数」を参照してください。

text ロケータの初期化

sp_drv_create_text_locator を使用して `text_locator` を作成し、必要に応じて値で初期化します。 **sp_drv_create_text_locator** は Transact-SQL 関数 **create_locator** にアクセスします。

構文

```
sp_drv_create_text_locator [init_value]
```

入力パラメータ

init_value - 新しいロケータの初期化に使用される `varchar` または `text` の値。

出力パラメータ

なし。

結果セット

`text_locator` 型のカラム。 ロケータが参照する LOB には、指定されている場合、*init_value* が含まれます。

unitext ロケータの初期化

sp_drv_create_unitext_locator を使用して `unitext_locator` を作成し、必要に応じて値で初期化します。 **sp_drv_create_unitext_locator** は Transact-SQL 関数 **create_locator** にアクセスします。

構文

```
sp_drv_create_unitext_locator [init_value]
```

入力パラメータ

init_value - 新しいロケータの初期化に使用される `univarchar` または `unitext`。

出力パラメータ

なし。

結果セット
`unitext_locator` 型のカラム。 ロケータが参照する LOB には、指定されている場合、*init_value* が含まれます。

image ロケータの初期化

sp_drv_image_unitext_locator を使用して `image_locator` を作成し、必要に応じて値で初期化します。 **sp_drv_create_image_locator** は Transact-SQL 関数 **create_locator** にアクセスします。

構文
```
sp_drv_create_image_locator [init_value]
```

入力パラメータ
init_value - 新しいロケータの初期化に使用される `varbinary` または `image`。

出力パラメータ
なし。

結果セット
`image_locator` 型のカラム。 ロケータが参照する LOB には、指定されている場合、*init_value* が含まれます。

text ロケータからの完全な text 値の取得

Transact-SQL 関数 **return_lob** にアクセスする **sp_drv_locator_to_text** を使用します。

構文
```
sp_drv_locator_to_text locator
```

入力パラメータ
locator - 値の取得対象となる `text_locator`。

出力パラメータ
なし。

結果セット
locator によって参照される `text` 値を含むカラム。

unitext ロケータからの完全な unitext 値の取得

Transact-SQL 関数 **return_lob** にアクセスする **sp_drv_locator_to_unitext** を使用します。

構文

```
sp_drv_locator_to_unitext locator
```

入力パラメータ

locator - 値の取得対象となる `unitext_locator`。

出力パラメータ

なし。

結果セット

locator によって参照される `unitext` 値を含むカラム。

image ロケータからの完全な image 値の取得

Transact-SQL 関数 **return_lob** にアクセスする **sp_drv_locator_to_image** を使用します。

構文

```
sp_drv_locator_to_image locator
```

入力パラメータ

locator - 値の取得対象となる `image_locator`。

出力パラメータ

なし。

結果セット

locator によって参照される `image` 値を含むカラム。

text ロケータからの部分文字列の取得

Transact-SQL 関数 **substring** にアクセスする **sp_drv_text_substring** を使用します。

構文

```
sp_drv_text_substring locator, start_position, length
```

入力パラメータ

- *locator* - 操作するデータを参照する `text_locator`。

- *start_position* - 読み込んで取得する最初の文字の位置を指定する integer。
- *length* - 読み込む文字数を指定する integer。

出力パラメータ
なし。

結果セット
取得された部分文字列を含む text 型のカラム。

unitext ロケータからの部分文字列の取得
Transact-SQL 関数 **substring** にアクセスする **sp_drv_unitext_substring** を使用します。

構文
```
sp_drv_unitext_substring locator, start_position, length
```

入力パラメータ

- *locator* - 操作するデータを参照する unitext_locator。
- *start_position* - 読み込んで取得する最初の文字の位置を指定する integer。
- *length* - 読み込む文字数を指定する integer。

出力パラメータ
なし。

結果セット
取得された部分文字列を含む unitext 型のカラム。

image ロケータからの部分文字列の取得
Transact-SQL 関数 **substring** にアクセスする **sp_drv_image_substring** を使用します。

構文
```
sp_drv_image_substring locator, start_position, length
```

入力パラメータ

- *locator* - 操作するデータを参照する image_locator。
- *start_position* - 読み込んで取得する最初のバイトの位置を指定する integer。
- *length* - 読み込むバイト数を指定する integer。

出力パラメータ
なし。

結果セット
取得された部分文字列を含む `image` 型のカラム。

指定した位置への text の挿入

Transact-SQL 関数 **setadata** にアクセスする **sp_drv_text_setdata** を使用します。

構文

```
sp_drv_text_setdata locator, offset, new_data, data_length
```

入力パラメータ

- *locator* - 挿入先の `text` カラムを参照する `text_locator`。
- *offset* - 新しいコンテンツの書き込み開始位置を指定する `integer`。
- *new_data* - 挿入する `varchar` または `text` データ。

出力パラメータ
data_length - 書き込まれた文字数を含む `integer`。

結果セット
なし。

指定した位置への unitext の挿入

Transact-SQL 関数 **setadata** にアクセスする **sp_drv_unitext_setdata** を使用します。

構文

```
sp_drv_unitext_setdata locator, offset, new_data, data_length
```

入力パラメータ

- *locator* - 挿入先の `unitext` カラムを参照する `unitext_locator`。
- *offset* - 新しいコンテンツの書き込み開始位置を指定する `integer`。
- *new_data* - 挿入する `univarchar` または `unitext` データ。

出力パラメータ
data_length - 書き込まれた文字数を含む `integer`。

結果セット
なし。

指定した位置への image の挿入

Transact-SQL 関数 **setadata** にアクセスする **sp_drv_image_setdata** を使用します。

構文

```
sp_drv_image_setdata locator, offset, new_data, datalength
```

入力パラメータ

- *locator* - 挿入先の `image` カラムを参照する `image_locator`。
- *offset* - 新しいコンテンツの書き込み開始位置を指定する `integer`。
- *new_data* - 挿入する `varbinary` または `image` データ。

出力パラメータ

data_length - 書き込まれたバイト数を含む `integer`。

結果セット

なし。

ロケータが参照する text の挿入

Transact-SQL 関数 **setadata** にアクセスする **sp_drv_text_locator_setdata** を使用します。

構文

```
sp_drv_text_locator_setdata locator, offset, new_data_locator,
data_length
```

入力パラメータ

- *locator* - 挿入先の `text` カラムを参照する `text_locator`。
- *offset* - 新しいコンテンツの書き込み開始位置を指定する `integer`。
- *new_data_locator* - 挿入先の `text` データを参照する `text_locator`。

出力パラメータ

data_length - 書き込まれた文字数を含む `integer`。

結果セット

なし。

ロケータが参照する unitext の挿入

Transact-SQL 関数 **setadata** にアクセスする **sp_drv_unitext_locator_setdata** を使用します。

構文

```
sp_drv_unitext_locator_setdata locator, offset, new_data_locator,
data_length
```

入力パラメータ

- *locator* - 挿入先の `unitext` カラムを参照する `unitext_locator`。
- *offset* - 新しいコンテンツの書き込み開始位置を指定する `integer`。
- *new_data_locator* - 挿入先の `unitext` データを参照する `unitext_locator`。

出力パラメータ

data_length - 書き込まれた文字数を含む `integer`。

結果セット

なし。

ロケータが参照する image の挿入

Transact-SQL 関数 **setadata** にアクセスする **sp_drv_image_locator_setdata** を使用します。

構文

```
sp_drv_image_locator_setdata locator, offset, new_data_locator,
   datalength
```

入力パラメータ

- *locator* - 挿入先の `image` カラムを参照する `image_locator`。
- *offset* - 新しいコンテンツの書き込み開始位置を指定する `integer`。
- *new_data_locator* - 挿入先の `image` データを参照する `image_locator`。

出力パラメータ

data_length - 書き込まれたバイト数を含む `integer`。

結果セット

なし。

基本となる LOB データのトランケート

truncate lob を使用して、LOB ロケータが参照している LOB データをトランケートします。

『ASE リファレンスマニュアル: コマンド』を参照してください。

基本となる text データの文字長の確認

sp_drv_text_locator_charlength を使用して、`text` ロケータが参照している LOB カラムの文字長を確認します。 **sp_drv_text_locator_charlength** は Transact-SQL 関数 **char_length** にアクセスします。

構文

```
sp_drv_text_locator_charlength locator, data_length
```

入力パラメータ
locator - 操作する `text` カラムを参照する `text_locator`。

出力パラメータ
data_length - *locator* が参照する `text` カラムの文字長を指定する `integer`。

結果セット
なし。

基本となる text データのバイト長の確認

sp_drv_text_locator_bytelength を使用して、`text` ロケータが参照している LOB カラムのバイト長を確認します。 **sp_drv_text_locator_bytelength** は Transact-SQL 関数 **data_length** にアクセスします。

構文

```
sp_drv_image_locator_bytelength locator, data_length
```

入力パラメータ
locator - 操作する `text` カラムを参照する `text_locator`。

出力パラメータ
data_length - *locator* が参照する `text` カラムのバイト長を指定する `integer`。

結果セット
なし。

基本となる unitext データの文字長の確認

sp_drv_unitext_locator_charlength を使用して、unitext ロケータが参照している LOB カラムの文字長を確認します。 **sp_drv_unitext_locator_charlength** は Transact-SQL 関数 **char_length** にアクセスします。

構文

```
sp_drv_unitext_locator_charlength locator, data_length
```

入力パラメータ
locator - 操作する unitext カラムを参照する unitext_locator。

出力パラメータ
data_length - *locator* が参照する unitext カラムの文字長を指定する integer。

結果セット
なし。

基本となる unitext データのバイト長の確認

sp_drv_unitext_locator_bytelength を使用して、unitext ロケータが参照している LOB カラムのバイト長を確認します。 **sp_drv_unitext_locator_bytelength** は Transact-SQL 関数 **data_length** にアクセスします。

構文

```
sp_drv_image_locator_bytelength locator, data_length
```

入力パラメータ
locator - 操作する unitext カラムを参照する unitext_locator。

出力パラメータ
data_length - *locator* が参照する unitext カラムのバイト長を指定する integer。

結果セット
なし。

基本となる image データのバイト長の確認

sp_drv_image_locator_bytelength を使用して、image ロケータが参照している LOB カラムのバイト長を確認します。 **sp_drv_image_locator_bytelength** は Transact-SQL 関数 **data_length** にアクセスします。

構文

```
sp_drv_image_locator_bytelength locator, data_length
```

入力パラメータ
locator - 操作する `image` カラムを参照する `image_locator`。

出力パラメータ
data_length - *locator* が参照する `image` カラムのバイト長を指定する `integer`。

結果セット
なし。

ロケータが参照する text カラム内の検索文字列位置の確認
Transact-SQL 関数 **charindex** にアクセスする **sp_drv_varchar_charindex** を使用します。

構文

```
sp_drv_varchar_charindex search_string, locator, start, position
```

入力パラメータ

- *search_string* - 検索対象となる `varchar` 型のリテラル。
- *locator* - 検索元の `text` カラムを参照する `text_locator`。
- *start* - 検索の開始位置を指定する整数。 最初の位置は 1 になります。

出力パラメータ
position - *locator* が参照する LOB カラム内の *search_string* の開始位置を指定する `integer`。

結果セット
なし。

別のロケータが参照している text カラム内の text ロケータが参照している文字列の位置の確認
Transact-SQL 関数 **charindex** にアクセスする **sp_drv_textlocator_charindex** を使用します。

構文

```
sp_drv_textlocator_charindex search_locator, locator, start,
position
```

入力パラメータ

- *search_locator* - 検索するリテラルを指す `text_locator`。
- *locator* - 検索元の `text` カラムを参照する `text_locator`。
- *start* - 検索の開始位置を指定する整数。 最初の位置は 1 になります。

出力パラメータ
position - *locator* が参照する LOB カラム内のリテラルの開始位置を指定する `integer`。

結果セット
なし。

ロケータが参照する unitext カラム内の検索文字列位置の確認
Transact-SQL 関数 **charindex** にアクセスする **sp_drv_univarchar_charindex** を使用します。

構文

```
sp_drv_univarchar_charindex search_string, locator, start, position
```

入力パラメータ

- *search_string* - 検索対象となる `univarchar` 型のリテラル。
- *locator* - 検索元の `unitext` カラムを参照する `unitext_locator`。
- *start* - 検索の開始位置を指定する整数。 最初の位置は 1 になります。

出力パラメータ
position - *locator* が参照する LOB カラム内の *search_string* の開始位置を指定する `integer`。

結果セット
なし。

別のロケータが参照している unitext カラム内の unitext ロケータが参照している文字列の位置の確認
Transact-SQL 関数 **charindex** にアクセスする **sp_drv_unitext_locator_charindex** を使用します。

構文

```
sp_drv_charindex_unitextloc_in_locator search_locator, locator,
start,
position
```

入力パラメータ

- *search_locator* - 検索するリテラルを指す `unitext_locator`。
- *locator* - 検索元の `text` カラムを参照する `unitext_locator`。
- *start* - 検索の開始位置を指定する整数。 最初の位置は 1 になります。

出力パラメータ
position - *locator* が参照する LOB カラム内のリテラルの開始位置を指定する `integer`。

結果セット
なし。

image ロケータが参照するカラム内のバイトシーケンス位置の確認
Transact-SQL 関数 **charindex** にアクセスする **sp_drv_varbinary_charindex** を使用します。

構文

```
sp_drv_varbinary_charindex byte_sequence, locator, start, position
```

入力パラメータ

- *byte_sequence* - 検索対象の `varbinary` 型のバイトシーケンス。
- *locator* - 検索元の `image` カラムを参照する `image_locator`。
- *start* - 検索の開始位置を指定する整数。 最初の位置は 1 になります。

出力パラメータ
position - *locator* が参照する LOB カラム内の *search_string* の開始位置を指定する `integer`。

結果セット
なし。

別のロケータが参照している image カラム内の image ロケータが参照しているバイトシーケンス位置の確認
Transact-SQL 関数 **charindex** にアクセスする **sp_drv_image_locator_charindex** を使用します。

構文

```
sp_drv_image_locator_charindex sequence_locator, locator, start,
start_position
```

入力パラメータ

- *sequence_locator* - 検索対象のバイトシーケンスを指す `image_locator`。
- *locator* - 検索元の `image` カラムを参照する `image_locator`。
- *start* - 検索の開始位置を指定する整数。 最初の位置は 1 になります。

出力パラメータ
start_position - *locator* が参照する LOB カラム内のバイトシーケンスの開始位置を指定する `integer`。

結果セット
なし。

text_locator の参照が有効であるかどうかの確認
locator_valid にアクセスする **sp_drv_text_locator_valid** を使用します。

構文
```
sp_drv_text_locator_valid locator
```

入力パラメータ
locator - 検証する `text_locator`。

出力パラメータ
次の値のいずれかを表す `bit`。

- 0 - false。*locator* は無効です。
- 1 - true。*locator* は有効です。

結果セット
なし。

unitext_locator の参照が有効であるかどうかの確認
locator_valid にアクセスする **sp_drv_unitext_locator_valid** を使用します。

構文
```
sp_drv_unitext_locator_valid locator
```

パラメータ
locator - 検証する `unitext_locator`。

出力パラメータ
次の値のいずれかを表す `bit`。

- 0 - false。*locator* は無効です。
- 1 - true。*locator* は有効です。

結果セット
なし。

image_locator の参照が有効であるかどうかの確認

locator_valid にアクセスする **sp_drv_image_locator_valid** を使用します。

構文

```
sp_drv_image_locator_valid locator
```

パラメータ
locator - 検証する `image_locator`。

出力パラメータ
次の値のいずれかを表す `bit`。

- 0 - false。*locator* は無効です。
- 1 - true。*locator* は有効です。

結果セット
なし。

LOB ロケータの割り付け解除

deallocate locator を使用して、LOB ロケータの解放と割り付け解除を行います。

『ASE リファレンスマニュアル: コマンド』を参照してください。

LOB ロケータの例

LOB ロケータに関する 6 つの例を示します。

例 1
ハンドルを割り付け、接続を確立します。

```
// Assumes that DSN has been named "sampledsn" and
// UseLobLocator has been set to 1.

SQLHENV environmentHandle = SQL_NULL_HANDLE;
SQLHDBC connectionHande = SQL_NULL_HANDLE;
SQLHSTMT statementHandle = SQL_NULL_HANDLE;
SQLRETURN ret;

SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &environmentHandle);
SQLSetEnvAttr(environmentHandle, SQL_ATTR_ODBC_VERSION,
SQL_ATTR_OV_ODBC3);
SQLAllocHandle(SQL_HANDLE_DBC, environmentHandle,
```

```
&connectionHandle);
Ret = SQLConnect(connectionHandle, "sampledsn",
   SQL_NTS, "sa", SQL_NTS, "Sybase",SQL_NTS);
```

例 2

カラムを選択し、ロケータを取得します。

```
// Selects and retrieves a locator for bk_desc, where
// bk_desc is a column of type text defined in a table
// named books. bk_desc contains the text "A book".

SQLPrepare(statementHandle, "SELECT bk_desc FROM books
   WHERE bk_id =1", SQL_NTS);

SQLExecute(statementHandle);
BYTE TextLocator[SQL_LOCATOR_SIZE];
SQLLEN Len = 0;
ret = SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);

If(Len == sizeof(TextLocator))
{
   Cout << Locator was created with expected size <<
   Len;
}
```

例 3

データ長を判別します。

```
SQLLEN LocatorLen = sizeof(TextLocator);
ret = SQLBindParameter(statementHandle, 1,
   SQL_PARAM_INPUT, SQL_C_TEXT_LOCATOR,
   SQL_TEXT_LOCATOR, SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), &LocatorLen);

SQLLEN CharLenSize = 0;
SQLINTEGER CharLen = 0;
ret = SQLBindParameter(statementHandle, 2,
SQL_PARAM_OUTPUT, SQL_C_LONG,SQL_INTEGER,0 , 0,
&CharLen, sizeof(CharLen), &CharLenSize);
SQLExecDirect(statementHandle,
   "{CALL sp_drv_text_locator_charlength( ?,?) }" , SQL_NTS);

cout<< "Character Length of Data " << charLen;
```

例 4

テキストを LOB カラムに追加します。

```
SQLINTEGER retVal = 0;
SQLLEN Col1Len = sizeof(retVal);
SQLCHAR appendText[10]=”abcdefghi on C++”;

SQLBindParameter(statementHandle, 14,
   SQL_PARAM_OUTPUT, SQL_C_SLONG, SQL_INTEGER, 0, 0, &retVal, 0,
Col1Len);
```

```
SQLBindParameter(statementHandle, 21, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, &TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);

SQLBindParameter(statementHandle, 32, SQL_PARAM_INPUT,
   SQL_C_SLONG, SQL_INTEGER, 0, 0, &charLen, 0, SQL_NULL_HANDLE);

SQLBindParameter(statementHandle, 43, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 10, 0, append_text,
   sizeof(append_text), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle,
   "{? = CALL sp_drv_setdata_text (?, ?, ?,?) }" , SQL_NTS);

SQLFreeStmt(statementHandle, SQL_CLOSE);
```

例 5

LOB データを LOB ロケータから取得します。

```
SQLCHAR description[512];
SQLLEN descriptionLength = 512;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_TEXT_LOCATOR, SQL_TEXT_LOCATOR,
   SQL_LOCATOR_SIZE, 0, TextLocator,
   sizeof(TextLocator), SQL_NULL_HANDLE);

SQLExecDirect(statementHandle, "{CALL sp_drv_locator_to_text(?)}",
SQL_NTS);

SQLFetch(statementHandle);

SQLGetData(statementHandle, 1,SQL_C_CHAR, description,
   descriptionLength, &descriptionLength)

Cout << "LOB data referenced by locator:" << description
   << endl;

Cout << "Expected LOB data:A book on C++" << endl;
```

例 6

クライアントアプリケーションのデータを LOB ロケータに転送します。

```
description = "A lot of data that will be used for a lot
   of inserts, updates and deletes"; descriptionLength = SQL_NTS;

SQLBindParameter(statementHandle, 1, SQL_PARAM_INPUT,
   SQL_C_CHAR, SQL_CHAR, 512, 0, description,
   sizeof(description), &descriptionLength);

SQLExecDirect(statementHandle,
   "{CALL sp_drv_create_text_locator(?)}", SQL_NTS);
```

```
SQLFetch(statementHandle);

SQLGetData(statementHandle, SQL_C_TEXT_LOCATOR,
   TextLocator, sizeof(TextLocator),&Len);
```

Python 用 Adaptive Server Enterprise 拡張モジュール

Python 用の Adaptive Server Enterprise 拡張モジュールは、Adaptive Server データベースに対してクエリを実行するための Sybase 固有の Python インタフェースです。

このモジュールは、拡張機能の付いた Python データベース API 仕様バージョン 2.0 を実装し、Python バージョン 2.6、2.7、3.1 で使用します。 Python データベース API の仕様については、http://www.python.org/dev/peps/pep-0249 を参照してください。

Python 用の Adaptive Server Enterprise 拡張モジュールは、SDK インストーラからインストールできます。 インストール手順については、『Software Developers Kit/Open Server インストールガイド』および『Software Developers Kit/Open Server リリースノート』を参照してください。 Python 用 Adaptive Server Enterprise 拡張モジュールの使用については、『Python 用 Adaptive Server Enterprise 拡張モジュールプログラマーズガイド』を参照してください。

PHP 用 Adaptive Server Enterprise 拡張モジュール

PHP 用の Adaptive Server Enterprise 拡張モジュールは、Adaptive Server データベースに対するクエリを実行し、クエリ結果を処理するためのインタフェースで、データベースへのアクセスに必要な PHP API を備えています。

このモジュールは PHP バージョン 5.3.6 で使用します。 PHP 用 Adaptive Server Enterprise 拡張モジュールの使用については、『PHP 用 Adaptive Server Enterprise 拡張モジュールプログラマーズガイド』を参照してください。

Perl 用 Adaptive Server Enterprise データベースドライバ

Perl 用の Adaptive Server Enterprise データベースドライバは、汎用 Perl DBI インタフェースで呼び出され、CT-Lib を使用した Open Client SDK 経由で、Perl DBI API 呼び出しを Adaptive Server が理解できる形式に変換します。

これにより、Perl スクリプトは Adaptive Server Enterprise データベースサーバに直接アクセスできるようになります。 このドライバは、Perl バージョン 5.14 および DBI バージョン 1.616 で使用します。

Perl DBI の仕様については、http://search.cpan.org/~timb/DBI-1.616/DBI.pm を参照してください。 Perl 用 Adaptive Server Enterprise データベースドライバの使用については、『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』を参照してください。

非推奨機能

Open Server と SDK の現在のリリースでは、一部のライブラリとユーティリティファイルをサポートしていません。

DCE サービスライブラリ

分散コンピューティング環境 (DCE) ディレクトリサービスライブラリ `libsybddce.dll` と DCE セキュリティサービスライブラリ `libsybsdce.dll` は、Windows 32 ビットプラットフォーム用の Open Client および Open Server から削除されています。

15.7 より前のバージョンの Open Client と Open Server では、これらのライブラリは `%SYBASE%¥OCS-15_0¥dll` ディレクトリに含まれていました。

dsedit_dce ユーティリティファイル

dsedit_dce X-Windows のデフォルトファイルである `OCS-15_0/xappdefaults/Dsedit_dce` と **dsedit_dce** ヘルプファイルである `OCS-15_0/sybhelp/dsedit_dceHelpTextMsgs` は削除されています。

サポートされていないプラットフォーム

Open Server と SDK では HP-UX PA-RISC と Mac OS をサポートしていません。

アクセシビリティ機能

第 508 条では、米国連邦機関の電子および情報技術は身体障害者が利用できるものと規定されています。 Sybase は、第 508 条を強力にサポートし、Open Client および Open Server バージョン 15.7 を含む広範な製品を、第 508 条に準拠させています。

リリース 15.7 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。

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」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。

索引