# New Features Bulletin Open Server™ 15.7 and SDK 15.7 for Windows, Linux, and UNIX

Document ID: DC20155-01-1570-06

Last revised: January 29, 2013

| Topic | Page |
|---|---|
| Product platforms and compatibilities | 4 |
| Solaris SPARC 64-bit patch level | 7 |
| Product components | 7 |
| Open Server | 7 |
| Software Developer's Kit | 8 |
| SDK DB-Library Kerberos Authentication Option | 10 |
| New features for ESD #7 | 10 |
| Open Client 15.7 and Open Server 15.7 features | 11 |
| SDK 15.7 features for Adaptive Server Enterprise Drivers and Providers | 15 |
| Adaptive Server Enterprise extension module for Python | 17 |
| Adaptive Server Enterprise extension module for PHP | 23 |
| New features for ESD #6 | 27 |
| Open Client 15.7 and Open Server 15.7 features | 27 |
| Adaptive Server Enterprise extension module for Python | 30 |
| Adaptive Server Enterprise extension module for Perl | 30 |
| New features for ESD #5 | 61 |
| Adaptive Server ADO.NET Data Provider support for Transact-SQL queries with COMPUTE clause | 61 |
| New SSIS Custom Data Flow Destination component for faster data transfers to Adaptive Server | 62 |
| jConnect dynamic logging levels | 64 |
| Package name changed in jConnect for converter classes | 64 |
| Increased PreparedStatement parameter limit in jConnect | 65 |
| New SkipRowCountResults connection property for Adaptive Server ODBC Driver | 65 |
| Support for AF_UNIX sockets in Adaptive Server ODBC Driver | 66 |

| Topic | Page |
|---|---|
| AdjustLargePrecisionAndScale connection property for Adaptive Server ODBC Driver | 67 |
| New features for ESD #4 | 67 |
| Open Client 15.7 and Open Server 15.7 features in ESD #4 | 68 |
| SDK 15.7 features for jConnect and Adaptive Server Drivers and Providers in ESD #4 | 74 |
| Adaptive Server Enterprise extension module for Python in ESD #4 | 85 |
| Adaptive Server Enterprise extension module for PHP in ESD #4 | 85 |
| Adaptive Server Enterprise database driver for Perl in ESD #4 | 87 |
| New features for ESD #3 | 88 |
| Skipping installation of samples, documentation, and debug files | 88 |
| Open Client 15.7 and Open Server 15.7 features in ESD #3 | 88 |
| Adaptive Server Enterprise extension module for Python in ESD #3 | 90 |
| New features for ESD #1 | 92 |
| Open Client 15.7 and Open Server 15.7 features in ESD #1 | 92 |
| SDK 15.7 features for jConnect and Adaptive Server Drivers and Providers in ESD #1 | 93 |
| Adaptive Server Enterprise extension module for Python in ESD #1 | 95 |
| Open Client 15.7 and Open Server 15.7 features | 96 |
| Large object locator support | 96 |
| In-row and off-row LOB support | 104 |
| Bulk-Library select into logging | 104 |
| Bulk-Library and bcp handling of nonmaterialized columns | 105 |
| Support for preserving trailing zeros | 106 |
| New DB-Library overflow errors | 106 |
| New nameless application configuration settings handling | 106 |
| TCP socket buffer size configuration | 107 |
| isql64 and bcp64 for all 64-bit products | 108 |
| Support for expanded variable-length rows | 108 |
| Row format caching | 109 |
| Support for releasing locks at cursor close | 110 |
| Large objects as stored procedure parameters | 111 |
| SDK 15.7 features for jConnect and Adaptive Server Enterprise Drivers and Providers | 121 |
| ODBC driver version information utility | 122 |

| Topic | Page |
|---|---|
| SupressRowFormat2 connection string property | 123 |
| Enhancement to UseCursor Property | 123 |
| Logging without ODBC Driver Manager tracing | 124 |
| jConnect setMaxRows enhancement | 125 |
| TDS ProtocolCapture | 125 |
| ODBC data batching without binding parameter arrays | 126 |
| Optimized batching in jConnect | 128 |
| jConnect parameter batching without row accumulation | 129 |
| jConnect batch update enhancement to execute past errors | 130 |
| Support for releasing locks at cursor close | 130 |
| select for update support | 130 |
| Support for expanded variable-length rows | 131 |
| Support for nonmaterialized columns | 131 |
| In-row and off-row LOB storage support | 132 |
| Large objects as stored procedure parameters | 132 |
| Large object locator support | 132 |
| Adaptive Server Enterprise extension module for Python | 151 |
| Adaptive Server Enterprise extension module for PHP | 151 |
| Adaptive Server Enterprise database driver for Perl | 151 |
| Deprecated features | 152 |
| DCE service libraries | 152 |
| dsedit_dce utility files | 152 |
| Unsupported platforms | 152 |
| Accessibility features | 153 |

# Product platforms and compatibilities

Table 1 lists the platforms that support Open Server™ and SDK:

***Table 1: Platforms that support Open Server and SDK***

| Platform |
|---|
| HP-UX Itanium 32-bit |
| HP-UX Itanium 64-bit |
| IBM AIX 32-bit |
| IBM AIX 64-bit |
| Linux x86 32-bit |
| Linux x86-64 64-bit |
| Linux on POWER 32-bit |
| Linux on POWER 64-bit |
| Microsoft Windows x86 32-bit |
| Microsoft Windows x86-64 64-bit |
| Solaris SPARC 32-bit |
| Solaris SPARC 64-bit |
| Solaris x86 32-bit |
| Solaris x86-64 64-bit |

**Note** Not all Open Server and SDK components are available on the platforms listed above. See “Product components” on page 7 for the complete list of components available on each platform.

Table 2 lists the platforms, compilers, and third-party products Open Server and SDK products are built and tested on:

***Table 2: Open Server and SDK platform compatibility matrix***

| Platform | Operating system level | C and C++ compilers | COBOL compiler | Kerberos version | Lightweight Directory Access (LDAP) | Secure Sockets Layer (SSL) | Perl version | PHP version | Python version |
|---|---|---|---|---|---|---|---|---|---|
| HP-UX Itanium 32-bit | HP 11.31 | HP ANSI C A.06.17 | MF SE 5.1 | MIT 1.4.1 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br><br>CSI-Crypto 2.7M1 | 5.14 (DBI 1.616) | n/a | n/a |
| HP-UX Itanium 64-bit | HP 11.31 | HP ANSI C A.06.17 | MF SE 5.1 | MIT 1.4.1 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br><br>CSI-Crypto 2.7M1 | n/a | 5.3.6 | 2.6, 2.7, and 3.1 (DBAPI 2.0) |

| Platform | Operating system level | C and C++ compilers | COBOL compiler | Kerberos version | Lightweight Directory Access (LDAP) | Secure Sockets Layer (SSL) | Perl version | PHP version | Python version |
|---|---|---|---|---|---|---|---|---|---|
| IBM AIX 32-bit | AIX 6.1 | XL C 10.1 | MF SE 5.1 | Cybersafe Trustbroker 2.1, MIT 1.4.1 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br><br>CSI-Crypto 2.7M1 | 5.14 (DBI 1.616) | n/a | n/a |
| IBM AIX 64-bit | AIX 6.1 | XL C 10.1 | MF SE 5.1 | MIT 1.4.3 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br><br>CSI-Crypto 2.7M1 | n/a | 5.3.6 | 2.6, 2.7, and 3.1 (DBAPI 2.0) |
| Linux x86 32-bit | 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 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br><br>CSI-Crypto 2.7M1 | n/a | n/a | n/a |
| Linux x86-64 64-bit | 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 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br><br>CSI-Crypto 2.7M1 | 5.14 (DBI 1.616) | 5.3.6 | 2.6, 2.7, and 3.1 (DBAPI 2.0) |
| Linux on POWER 32-bit | Red Hat Enterprise Linux 5.3 | XL C 10.1 | None planned | MIT 1.4.1 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br><br>CSI-Crypto 2.7M1 | 5.14 (DBI 1.616) | n/a | n/a |
| Linux on POWER 64-bit | Red Hat Enterprise Linux 5.3 | XL C 10.1 | MF SE 5.1 | MIT 1.4.1 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br><br>CSI-Crypto 2.7M1 | n/a | 5.3.6 | 2.6, 2.7, and 3.1 (DBAPI 2.0) |
| Microsoft Windows x86 32-bit | Windows 2008 R2 Service Pack 1<br><br>Windows XP Service Pack 1 (ODBC/ OLE DB only) | 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 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br><br>CSI-Crypto 2.7M1 | n/a | n/a | n/a |

| Platform | Operating system level | C and C++ compilers | COBOL compiler | Kerberos version | Lightweight Directory Access (LDAP) | Secure Sockets Layer (SSL) | Perl version | PHP version | Python version |
|---|---|---|---|---|---|---|---|---|---|
| Microsoft Windows x86-64 64-bit | Windows 2008 R2 Service Pack 1<br>Windows XP Service Pack 1 (ODBC/ OLE DB only) | Microsoft Visual Studio 2005 Service Pack 1 (C/C++) | MF SE 5.1 | Cybersafe Trustbroker 2.1 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br>CSI-Crypto 2.7M1 | Active Perl 5.14.1 (DBI 1.616) | 5.3.6 | 2.6, 2.7, and 3.1 (DBAPI 2.0) |
| Solaris SPARC 32-bit | Solaris 10 | Solaris Studio 12.1 | MF SE 5.1 | Cybersafe Trustbroker 2.1, MIT 1.4.2 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br>CSI-Crypto 2.7M1 | 5.14 (DBI 1.616) | n/a | n/a |
| Solaris SPARC 64-bit | Solaris 10, patch level 144488-17 or later, patch level 119963-24 or later for *SUNWlibC* | Solaris Studio 12.1 | MF SE 5.1 | Cybersafe Trustbroker 2.1, MIT 1.4.2 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br>CSI-Crypto 2.7M1 | n/a | 5.3.6 | 2.6, 2.7, and 3.1 (DBAPI 2.0) |
| Solaris x86 32-bit | Solaris 10 | Solaris Studio 12.1 | MF SE 5.1 | MIT 1.4.2 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br>CSI-Crypto 2.7M1 | 5.14 (DBI 1.616) | n/a | n/a |
| Solaris x86-64 64-bit | Solaris 10 | Solaris Studio 12.1 | MF SE 5.1 | MIT 1.4.2 | OpenLDAP 2.4.31 with OpenSSL 1.0.1b | Certicom SSL Plus 5.2.2 (SBGSE 2.0)<br>CSI-Crypto 2.7M1 | n/a | 5.3.6 | 2.6, 2.7, and 3.1 (DBAPI 2.0) |

LEGEND: n/a = script is not available or does not work with SDK on that platform.

**Note** For the most current Open Server and SDK certifications support, see the Sybase® platform certifications page at http://certification.sybase.com/ucr/search.do.

**Note** Microsoft has ended mainstream support for Visual Studio 2005. Although, SDK currently supports Visual Studio Compiler 2005 and later versions, Sybase recommends that you move to Visual Studio 2010 as soon as possible.

### Solaris SPARC 64-bit patch level

For the Solaris SPARC 64-bit platform, the Solaris 10 operating system kernel patch level must be 144488-17 or later (patch bundle June 30th, 2011 or later). You must also apply patch 119963-24 or later to the *SUNWlibC* package.

## Product components

Open Server 15.7 and SDK 15.7 introduce new features, such as Bulk-Library select into logging, large object stored procedure parameter support, support for nonmaterialized columns in Adaptive Server® Enterprise, and updates to jConnect™ for JDBC™ and Adaptive Server drivers and providers. Open Server 15.7 and SDK 15.7 also support the Perl, PHP, and Python scripting languages for use with Adaptive Server.

The products released with 15.7 are:

- Open Server
- Software Developer’s Kit
- SDK DB-Library Kerberos Authentication Option

### Open Server

Open Server is a set of APIs and supporting tools you can use to create custom servers to respond to client requests submitted through Open Client™ or jConnect for JDBC routines. Table 3 lists the Open Server components and the platforms these components are supported on.

***Table 3: Open Server components and supported platforms***

| Open Server components | Platforms |
|---|---|
| Open Server Server-Library | All platforms |
| Open Server Client-Library | All platforms |
| Language modules | All platforms |

## Software Developer's Kit

The Software Developer's Kit (SDK) is a set of libraries and utilities you can use to develop client applications. Table 4 lists the SDK components and the platforms these components are supported on.

***Table 4: SDK components and supported platforms***

| SDK components | Platforms |
|---|---|
| Open Client Client-Library | All platforms |
| Open Client DB-Library™ | All platforms |
| Embedded SQL™/C (ESQL/C) | All platforms |
| Embedded SQL/COBOL (ESQL/COBOL) | • HP HP-UX Itanium 32-bit<br>• HP HP-UX Itanium 64-bit<br>• IBM AIX 64-bit<br>• Linux x86 32-bit<br>• Linux x86-64 64-bit<br>• Linux on POWER 32-bit<br>• Linux on POWER 64-bit<br>• Microsoft Windows x86 32-bit<br>• Microsoft Windows x86-64 64-bit<br>• Solaris SPARC 32-bit<br>• Solaris SPARC 64-bit<br>• Solaris x86 32-bit<br>• Solaris x86-64 64-bit |

| SDK components | Platforms |
|---|---|
| Extended Architecture (XA) | • HP HP-UX Itanium 32-bit<br>• HP HP-UX Itanium 64-bit<br>• IBM AIX 32-bit<br>• IBM AIX 64-bit<br>• Linux x86-64 64-bit<br>• Microsoft Windows x86 32-bit<br>• Microsoft Windows x86-64 64-bit<br>• Solaris SPARC 32-bit<br>• Solaris SPARC 64-bit<br>• Solaris x86 32-bit<br>• Solaris x86-64 64-bit |
| jConnect for JDBC | All platforms |
| Adaptive Server Enterprise ODBC Driver by Sybase | • HP HP-UX Itanium 64-bit<br>• IBM AIX 64-bit<br>• Linux on POWER 64-bit<br>• Linux x86 32-bit<br>• Linux x86-64 64-bit<br>• Microsoft Windows x86 32-bit<br>• Microsoft Windows x86-64 64-bit<br>• Solaris SPARC 64-bit<br>• Solaris x86-64 64-bit |
| Adaptive Server Enterprise OLE DB Provider by Sybase | • Microsoft Windows x86 32-bit<br>• Microsoft Windows x86-64 64-bit |
| Adaptive Server Enterprise ADO.NET Data Provider | • Microsoft Windows x86 32-bit<br>• Microsoft Windows x86-64 64-bit |
| Language modules | All platforms |
| Adaptive Server Enterprise extension module for Python | • HP-UX Itanium 64-bit<br>• IBM AIX 64-bit<br>• Linux x86-64 64-bit<br>• Linux on POWER 64-bit<br>• Microsoft Windows x86-64 64-bit<br>• Solaris SPARC 64-bit<br>• Solaris x86-64 64-bit |

| SDK components | Platforms |
|---|---|
| Adaptive Server Enterprise extension module for PHP | • HP-UX Itanium 64-bit<br>• IBM AIX 64-bit<br>• Linux x86-64 64-bit<br>• Linux on POWER 64-bit<br>• Microsoft Windows x86-64 64-bit<br>• Solaris SPARC 64-bit<br>• Solaris x86-64 64-bit |
| Adaptive Server Enterprise database driver for Perl | • HP-UX Itanium 32-bit<br>• IBM AIX 32-bit<br>• Linux x86-64 64-bit<br>• Linux on POWER 32-bit<br>• Microsoft Windows x86-64 64-bit<br>• Solaris SPARC 32-bit<br>• Solaris x86 32-bit |

### SDK DB-Library Kerberos Authentication Option

The Sybase SDK DB-Library Kerberos Authentication Option allows the MIT Kerberos security mechanism to be used on DB-Library and is available on:

- Linux x86 32-bit
- Microsoft Windows x86 32-bit
- Solaris SPARC 32-bit
- Solaris SPARC 64-bit

## New features for ESD #7

ESD #7 introduces updated functionality for Open Client 15.7 and Open Server 15.7, SDK 15.7, Adaptive Server Enterprise extension module for Python 15.7, and Adaptive Server Enterprise extension module for PHP 15.7.

## Open Client 15.7 and Open Server 15.7 features

Open Client 15.7 and Open Server 15.7 have been enhanced to support Client-Library connection string properties, remote password encryption, and *libsybsspiwrapper64.dll* for Windows 64-bit.

### Client-Library supports connection string properties

Client-Library now supports the API routine, connect_string():

#### ct_connect_string()

Description

Connects to a server by specifying a connection string. This function provides the same functionality as ct_connect(). It also provides a mechanism to set certain attributes at connection time.

Syntax

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

Parameters

- *connection* – a pointer to a CS_CONNECTION structure. A CS_CONNECTION structure contains information about a particular client/server connection. Use ct_con_alloc to allocate a CS_CONNECTION structure, and ct_con_props to initialize that structure with login parameters.
- *connection_string* – a string containing attribute names and values.
- *length* – the length, in bytes, of the connection string.

Return value

ct_connect returns:

| Returns | Indicates |
|---|---|
| CS_SUCCEED | The routine completed successfully. |
| CS_FAIL | The routine failed. |
| CS_PENDING | Asynchronous network I/O is in effect. See the Asynchronous programming section in the *Open Client Client-Library/C Reference Manual*. |
| CS_BUSY | An asynchronous operation is already pending for this connection. See the Asynchronous programming section in the *Open Client Client-Library/C Reference Manual*. |

The connection string is a semicolon-separated string of name=value parts:

1. Name – a case-insensitive value that can be delimited by an equal sign (=) or semicolon (;). An attribute can have multiple synonyms. For example, server and servername refer to the same attribute.
2. Equals sign (=) – indicates the start of the value to be assigned to the Name. If there is no equals sign, the Name is assumed to be of Boolean type with a value of true.
3. Value – a string that is terminated by a semicolon (;). To skip the termination, use a backslash (\) if a semicolon or another back slash is present in the value. Values can be of type boolean, integer, or string. Valid values for Boolean types are true, false, on, off, 1, and 0.

For example:

```
Username=me; Password=mypassword; Servername=SYBASE
```

**Valid attribute names and values**

| Name | Description | Value |
|---|---|---|
| ANSINull | Determines whether evaluation of NULL-valued operands in SQL equality (=) or inequality (!=) comparisons is ANSI-compliant.<br>If the value is true, Adaptive Server enforces the ANSI behavior that *= NULL* and *is NULL* are not equivalent. In standard Transact-SQL®, *= NULL* and *is NULL* are considered to be equivalent.<br>This option affects *<> NULL* and *is not NULL* behavior in a similar fashion. | Boolean value.<br>The default is false. |
| BulkLogin | Determines whether a connection is enabled to perform a bulk-copy operation. | Boolean value.<br>The default is false. |
| ChainXacts | If true, Adaptive Server uses chained transaction behavior, that is, each server command is considered to be a distinct transaction.<br>Adaptive Server implicitly executes a begin transaction before any of these statements: delete, fetch, insert, open, select, and update. You must still explicitly end or roll back the transaction.<br>If false, an application must specify explicit begin transaction statements paired with commit or rollback statements. | Boolean value.<br>The default is false. |

| Name | Description | Value |
|---|---|---|
| Charset | Specifies the charset to be used on this connection. | String value. |
| Confidentiality | Whether data encryption service is performed on the connection. | Boolean value.<br>The default is false. |
| CredentialDelegation | Determines whether to allow the server to connect to a second server with the user’s delegated credentials. | Boolean value.<br>The default is false. |
| DetectReplay | Determines whether the connection’s security mechanism detects replayed transmissions. | Boolean value.<br>The default is false. |
| DetectOutOfSequence | Determines whether the connection’s security mechanism detects transmissions that arrive out of sequence. | Boolean value.<br>The default is false. |
| Integrity | Determines whether the connection’s security mechanism performs data integrity checking. | Boolean value.<br>The default is false. |
| Interfaces | The path and name of the interfaces file. | String value. |
| Keytab | The name and path to the file from which a connection’s security mechanism reads the security key to go with the *username* value. | String value.<br>The default is NULL, that is, the user must have established credentials before connecting. |
| Locale | Determines which language and character set to use for messages, datatype conversions, and datetime formats. | String value. |
| Language | Determines which language set to use for messages, datatype conversions, and datetime formats. | String value. |
| LoginTimeout | Specifies the login timeout value. | Integer value. |
| MaxConnect | Specifies the maximum number of simultaneously open connections that a context may have. | Integer value.<br>Default value is 25. Negative and zero values are not allowed. |

| Name | Description | Value |
|---|---|---|
| MutualAuthentication | Determines whether the server is required to authenticate itself to the client. | Boolean value. The default is false. |
| NetworkAuthentication | Determines whether the connection's security mechanism performs network-based user authentication. | Boolean value. The default is false. |
| PacketSize | Specifies the TDS packet size. | Integer value. |
| Password | Specifies the password used to log in to the server. | String value. |
| PasswordEncryption | Determines whether the connection uses asymmetrical password encryption. | Boolean value. The default is false. |
| SecurityMechanism | Specifies the name of the network security mechanism that performs security services for the connection. | String value. The default value depends on security driver configuration. |
| Server Servername | Specifies the name of the server to which you are connected. | String value. |
| ServerPrincipalName | Specifies the network security principal name for the server to which a connection is opened. | String value. The default is NULL, which means that the connection assumes the server principal name is the same as its *ServerName* value. |
| TDS_Keepalive | Determines whether to use the KEEPALIVE option. | Boolean value. The default is true. |
| Timeout | Specifies the connection timeout value. | Integer value. |
| UID User Username | Specifies the name used to log in to the server. | String value. |

### Remote password encryption

Open Server supports the retrieval of remote password pairs for connections using extended plus encrypted passwords (EPEP). The retrieving properties, including SRV_T_NUMRMTPWDS and SRV_T_RMTPWDS, work with srv_thread_props(). If the client supports the EPEP protocol, the SRV_T_NUMRMTPWDS property returns the number of decrypted remote password pairs, and the SRV_T_RMTPWDS property returns the password pairs.

### *libsybsspiwrapper64.dll* for Windows 64-bit

Use the *libsybsspiwrapper64.dll* wrapper library to allow Kerberos security driver to use the Windows Security Support Provider Interface (SSPI) routines on Windows 64-bit platform.

To use this feature, you must edit *libtcl64.cfg* to include *libsybsspiwrapper64.dll*. For example:

```
[SECURITY]

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

**Note** This library is stored in the *%SYBASE%\OCS-15_0\lib3p64* directory.

## SDK 15.7 features for Adaptive Server Enterprise Drivers and Providers

ESD #7 introduces new functionality for Adaptive Server ODBC Driver 15.7 and the Ribo utility.

### New *CancelQueryOnFreeStmt* connection property for Adaptive Server ODBC Driver

If a Microsoft Access form that is using the Adaptive Server ODBC Driver to execute a query that returns large result set is closed before the entire result set is processed, Microsoft Access remains unresponsive until the ODBC Driver completes processing the entire result set.

In version 15.7 ESD #7, a new connection property CancelQueryOnFreeStmt addresses this issue. When this connection property is set to 1, whenever a form is closed, the Adaptive Server ODBC Driver cancels any pending results and returns control to the Microsoft Access application immediately. When set to 0 (default value), there is no change in Adaptive Server ODBC Driver behavior.

## New efficient method to set client connection attributes

In version 15.7 ESD #7, Adaptive Server ODBC Driver adds support for setting client connection attributes efficiently using the ODBC SQLSetConnectAttr API. The attribute values set are visible in the Adaptive Server sysprocesses table and help distinguish different client connections.

To set these attributes in versions earlier than 15.7 ESD #7, application programs had to explicitly call set statements to set corresponding attributes resulting in additional executions on the server. When the SQLSetConnectAttr API is used, the driver defers executing the set statements, attaching them to the next statement that is executed.

**Note** Since the set statements are not executed immediately after SQLSetConnectAttr API is called, the values set are invisible on Adaptive Server until the next statement is executed.

SQLSetConnectAttr supports these attributes:

- SQL_ATTR_CLIENT_NAME – sets the client name, using the command set *clientname <value>*.
- SQL_ATTR_CLIENT_HOST_NAME – sets the client host name, using the command set *clienthostname <value>*.
- SQL_ATTR_CLIENT_APPL_NAME – sets the client application name, using the command set *clientapplname <value>*.

The value of these attributes is truncated to 30 bytes.

Use the ODBC SQLGetConnectAttr to retrieve the value of these attributes. However, it does not reflect any changes to the server value made outside of this interface.

### Enhanced support for data-at-exec feature in Adaptive Server ODBC Driver

In Adaptive Server ODBC Driver version 15.7 ESD #7, the data-at-exec feature has been enhanced to support bulk and batch operations resulting in lower memory utilization and increased performance for applications.

In earlier versions, all of the data for bound parameters had to be fully loaded before calling SQLBulkOperations or executing a batch. In ESD #7, the application does not need to preload any parameter data, it can be sent in chunks using SQLPutData. When using the Adaptive Server ODBC Driver batch protocol (SQLExecute/SQLExecDirect with SQL_ATTR_BATCH_PARAMS), data-at-exec is supported as long as SQL_ATTR_PARAMSET_SIZE is set to 1. Using data-at-exec for LOB columns requires the server to support LOB parameters.

### New *-n* command line option in Ribo utility

In versions earlier than 15.7 ESD #7, Ribo utility translated the entire raw *.tds* dump file in to a single translation file regardless of the size. Ribo utility has been enhanced to translate a raw *.tds* dump file into multiple files of manageable file sizes using a new command line option, -n. You specify the maximum size for a single translation file, in KB, with the -n option. When the translation output file results in a size greater than the value specified in -n option, a new file will be created.

The output file name follows this naming convention:

```
<output_file_part1_of_5> <output_file_part2_of_5>
```

where <output_file> is a file specified by the user, appended with partX_ofY, where *X* is the current part and *Y* is the number of parts into which the translated output is divided.

**Note** The -n flag takes effect when the translation is performed.

## Adaptive Server Enterprise extension module for Python

The Adaptive Server Enterprise extension module for Python has been enhanced to support Data Source Name style (DSN-style) connection properties, new sample programs, and blklib.

## Support for DSN-style connection string properties

The connect() method supports for DSN-style connection properties as described as follows:

### connect()

Constructs a connection object representing a connection to a database. The method accepts these keyword arguments:

- user – the user login name that the connection uses to log in to a server.
- password – the password that a connection uses when logging in to a server.
- servername – defines the Adaptive Server name to which client programs connect. If you do not specify servername, the DSQUERY environment variable defines the Adaptive Server name.
- dsn – the data source name. The data source name is a semicolon-separated string of name=value parts:
    - Name – a case-insensitive value that can be delimited by an equal sign (=) or semicolon (;). An attribute can have multiple synonyms. For example, server and servername refer to the same attribute.
    - Equals sign (=) – indicates the start of the value to be assigned to the Name. If there is no equals sign, the Name is assumed to be of boolean type with a value of true.
    - Value – a string that is terminated by a semicolon (;). To skip the termination, use a backslash (\) if a semicolon or another back slash is present in the value. Values can be of type boolean, integer, or string. Valid values for boolean types are true, false, on, off, 1, and 0.

  For example:

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

### Valid attribute names and values

The following table describes valid attribute names and values for the dsn keyword argument:

| Name | Description | Value |
|---|---|---|
| ANSINull | Determines whether evaluation of NULL-valued operands in SQL equality (=) or inequality (!=) comparisons is ANSI-compliant.<br>If the value is true, Adaptive Server enforces the ANSI behavior that = *NULL* and *is NULL* are not equivalent. In standard Transact-SQL, = *NULL* and *is NULL* are considered to be equivalent.<br>This option affects <> *NULL* and *is not NULL* behavior in a similar fashion. | Boolean value.<br>The default is false. |
| BulkLogin | Determines whether a connection is enabled to perform a bulk-copy operation. | Boolean value.<br>The default is false. |
| ChainXacts | If true, Adaptive Server uses chained transaction behavior, that is, each server command is considered to be a distinct transaction.<br>Adaptive Server implicitly executes a begin transaction before any of these statements: delete, fetch, insert, open, select, and update. You must still explicitly end or roll back the transaction.<br>If false, an application must specify explicit begin transaction statements paired with commit or rollback statements. | Boolean value.<br>The default is false. |
| Charset | Specifies the charset to be used on this connection. | String value. |
| Confidentiality | Whether data encryption service is performed on the connection. | Boolean value.<br>The default is false. |
| CredentialDelegation | Determines whether to allow the server to connect to a second server with the user's delegated credentials. | Boolean value.<br>The default is false. |
| DetectReplay | Determines whether the connection's security mechanism detects replayed transmissions. | Boolean value.<br>The default is false. |
| DetectOutOfSequence | Determines whether the connection's security mechanism detects transmissions that arrive out of sequence. | Boolean value.<br>The default is false. |

| Name | Description | Value |
|---|---|---|
| Integrity | Determines whether the connection’s security mechanism performs data integrity checking. | Boolean value.<br>The default is false. |
| Interfaces | The path and name of the interfaces file. | String value. |
| Keytab | The name and path to the file from which a connection’s security mechanism reads the security key to go with the *username* value. | String value.<br>The default is NULL, that is, the user must have established credentials before connecting. |
| Locale | Determines which language and character set to use for messages, datatype conversions, and datetime formats. | String value. |
| Language | Determines which language set to use for messages, datatype conversions, and datetime formats. | String value. |
| LoginTimeout | Specifies the login timeout value. | Integer value. |
| MaxConnect | Specifies the maximum number of simultaneously open connections that a context may have. | Integer value.<br>Default value is 25. Negative and zero values are not allowed. |
| MutualAuthentication | Determines whether the server is required to authenticate itself to the client. | Boolean value.<br>The default is false. |
| NetworkAuthentication | Determines whether the connection’s security mechanism performs network-based user authentication. | Boolean value.<br>The default is false. |
| PacketSize | Specifies the TDS packet size. | Integer value. |
| Password | Specifies the password used to log in to the server. | String value. |
| PasswordEncryption | Determines whether the connection uses asymmetrical password encryption. | Boolean value.<br>The default is false. |

| Name | Description | Value |
|---|---|---|
| SecurityMechanism | Specifies the name of the network security mechanism that performs security services for the connection. | String value.<br>The default value depends on security driver configuration. |
| Server<br>Servername | Specifies the name of the server to which you are connected. | String value. |
| ServerPrincipalName | Specifies the network security principal name for the server to which a connection is opened. | String value.<br>The default is NULL, which means that the connection assumes the server principal name is the same as its *ServerName* value. |
| TDS_Keepalive | Determines whether to use the KEEPALIVE option. | Boolean value.<br>The default is true. |
| Timeout | Specifies the connection timeout value. | Integer value. |
| UID<br>User<br>Username | Specifies the name used to log in to the server. | String value. |

## New sample programs

Several new samples are available for Adaptive Server Enterprise extension module for Python:

### dsnconnect

Demonstrates how to connect to a server using a dsn.

### blk

Uses the bulk-copy routines to copy data to a server table. The data is then retrieved and shown.

### blkmany

Uses the bulk-copy routines to copy data and multiple rows at a time.

### blkiter

Demonstrates how to use the Python iteration protocol to bulk-copy-out rows of a table.

### blktypes

Demonstrates how to use different Python object types (default, NULL values, and so on) as values in a bulk operation.

## blklib support

The blklib feature is an extension to the Python DB-API, which enables you to bulk-copy rows. The blklib feature includes an object interface, methods, and attributes.

### BulkCursor object constructor

Description

Python extension module that provides a connection object to establish a connection to the database. The connection object includes a method for creating a new BulkCursor object, which manages the context of a bulk operation.

The BulkCursor object can be constructed only from a connection object that was established with a property marking the connection for use in a bulk operation.

Usage

```
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()

Description

The close() method of the BulkCursor object closes a bulk operation. Once this method has been called, the bulk cursor object cannot be used. close() takes no arguments.

Usage

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

```
bblk.close()
```

### copy()

Description

The copy() method of the BulkCursor object initializes a bulk operation. This method accepts the following arguments:

- tablename – a string specifying the name of the table for the bulk operation.
- direction – this is a keyword argument with these values: *in* and *out*.

Usage

```
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()

Description

The done() method of the BulkCursor object marks the completion of a bulk operation. To start another operation, call the copy() method.

Usage

```
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()
```

## Adaptive Server Enterprise extension module for PHP

Adaptive Server Enterprise extension module for PHP has been enhanced to support DSN style connection properties.

## Support for DSN-style connection properties

sybase_connect() and sybase_pconnect() APIs support DSN-style connection properties. When you call either of these APIs using only the servername parameter, servername must contain a valid DSN (data source name) string. The data source name is a semicolon (;) separated string of name=value parts as explained as follows:

1. Name – a case-insensitive value that can be delimited by an equal sign (=) or semicolon (;). An attribute can have multiple synonyms. For example, server and servername refer to the same attribute.
2. Equals sign (=) – indicates the start of the value to be assigned to the Name. If there is no equals sign, the Name is assumed to be of boolean type with a value of true.
3. Value – a string that is terminated by a semicolon (;). To skip the termination, use a backslash (\) if a semicolon or another back slash is present in the value. Values can be of type boolean, integer, or string. Valid values for boolean types are true, false, on, off, 1, and 0.

For example:

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

### Valid attribute names and values

The following table describes valid attribute names and values for the dsn keyword argument:

| Name | Description | Value |
|---|---|---|
| ANSINull | Determines whether evaluation of NULL-valued operands in SQL equality (=) or inequality (!=) comparisons is ANSI-compliant.<br>If the value is true, Adaptive Server enforces the ANSI behavior that *= NULL* and *is NULL* are not equivalent. In standard Transact-SQL, *= NULL* and *is NULL* are considered to be equivalent.<br>This option affects *<> NULL* and *is not NULL* behavior in a similar fashion. | Boolean value.<br>The default is false. |
| BulkLogin | Determines whether a connection is enabled to perform a bulk-copy operation. | Boolean value.<br>The default is false. |

| Name | Description | Value |
|---|---|---|
| ChainXacts | If true, Adaptive Server uses chained transaction behavior, that is, each server command is considered to be a distinct transaction.<br><br>Adaptive Server implicitly executes a begin transaction before any of these statements: delete, fetch, insert, open, select, and update. You must still explicitly end or roll back the transaction.<br><br>If false, an application must specify explicit begin transaction statements paired with commit or rollback statements. | Boolean value.<br><br>The default is false. |
| Charset | Specifies the charset to be used on this connection. | String value. |
| Confidentiality | Whether data encryption service is performed on the connection. | Boolean value.<br><br>The default is false. |
| CredentialDelegation | Determines whether to allow the server to connect to a second server with the user's delegated credentials. | Boolean value.<br><br>The default is false. |
| DetectReplay | Determines whether the connection's security mechanism detects replayed transmissions. | Boolean value.<br><br>The default is false. |
| DetectOutOfSequence | Determines whether the connection's security mechanism detects transmissions that arrive out of sequence. | Boolean value.<br><br>The default is false. |
| Integrity | Determines whether the connection's security mechanism performs data integrity checking. | Boolean value.<br><br>The default is false. |
| Interfaces | The path and name of the interfaces file. | String value. |
| Keytab | The name and path to the file from which a connection's security mechanism reads the security key to go with the *username* value. | String value.<br><br>The default is NULL, that is, the user must have established credentials before connecting. |
| Locale | Determines which language and character set to use for messages, datatype conversions, and datetime formats. | String value. |

| Name | Description | Value |
|---|---|---|
| Language | Determines which language set to use for messages, datatype conversions, and datetime formats. | String value. |
| LoginTimeout | Specifies the login timeout value. | Integer value. |
| MaxConnect | Specifies the maximum number of simultaneously open connections that a context may have. | Integer value. Default value is 25. Negative and zero values are not allowed. |
| MutualAuthentication | Determines whether the server is required to authenticate itself to the client. | Boolean value. The default is false. |
| NetworkAuthentication | Determines whether the connection's security mechanism performs network-based user authentication. | Boolean value. The default is false. |
| PacketSize | Specifies the TDS packet size. | Integer value. |
| Password | Specifies the password used to log in to the server. | String value. |
| PasswordEncryption | Determines whether the connection uses asymmetrical password encryption. | Boolean value. The default is false. |
| SecurityMechanism | Specifies the name of the network security mechanism that performs security services for the connection. | String value. The default value depends on security driver configuration. |
| Server Servername | Specifies the name of the server to which you are connected. | String value. |
| ServerPrincipalName | Specifies the network security principal name for the server to which a connection is opened. | String value. The default is NULL, which means that the connection assumes the server principal name is the same as its *ServerName* value. |

| Name | Description | Value |
|---|---|---|
| TDS_Keepalive | Determines whether to use the KEEPALIVE option. | Boolean value.<br>The default is true. |
| Timeout | Specifies the connection timeout value. | Integer value. |
| UID<br>User<br>Username | Specifies the name used to log in to the server. | String value. |

### ***dsnconnect.php*** **sample program**

The *dsnconnect.php* sample program connects to a server using a DSN connection string. It optionally prints the server name, the user account, and the current database.

# New features for ESD #6

ESD #6 introduces updated functionality for Open Client 15.7 and Open Server 15.7, Data Source Name (DSN) connection properties support for Adaptive Server Enterprise extension module for Python 15.7, and Adaptive Server Enterprise extension module for Perl 15.7.

## Open Client 15.7 and Open Server 15.7 features

Open Client 15.7 and Open Server 15.7 have been enhanced to support bulk-copy-in with LOB datatype, the new SYBOCS_IFILE environment variable, LDAP and SSL version, parameter format suppression, extended plus encrypted password, and BCP --quoted-fname option.

### Bulk-copy-in with *LOB* datatype

In previous versions, if you marked an LOB column for transfer using blk_textxfer() API to copy LOB data into a database table consisting of both in-row and off-row values, all subsequent columns of this datatype were also required to be marked for transfer using blk_textxfer() API, and could not use blk_rowxfer(). With ESD#6, this limitation is removed and you can use blk_textxfer() followed by blk_rowxfer() API call.

## New SYBOCS_IFILE environment variable

Use SYBOCS_IFILE to specify the location of the interfaces file instead of the default *$SYBASE/interfaces*. If the application sets the CS_IFILE property in CT-Library, the property setting takes precedence.

## LDAP and SSL version support

The Sybase-provided OpenLDAP library (*libsybaseldap.so/dll*) uses OpenLDAP version 2.4.31 and OpenSSL version 1.0.1b for the connections to an LDAP server.

## Parameter format suppression

Open Client and Open Server now support parameter format suppression for dynamic statements in Adaptive Server Enterprise.

**Note** Starting with ESD #3, Open Client has been supporting the parameter format suppression. However, ESD #6 introduces Open Server support for parameter format suppression.

## Open Server support for extended plus encrypted password

When a client connection supports extended plus encrypted password (EPEP), Open Server handles the login negotiation, including decrypting of the password. The negotiation takes place before the SRV_CONNECT handler is called. In the SRV_CONNECT event handler, applications can simply retrieve the password with the existing SRV_T_PWD property and inspect the used password encryption protocol with a new property.

To try out Open Server password encryption, you can connect to the 'lang' sample using isql with the -X option, which turns on password encryption in isql.

**Note** From 15.0 release, Open Client supported the strong login password encryption. However, with ESD#6, Open Server supports the strong login password encryption.

### SRV_T_PWD

This property is used with srv_thread_props() to retrieve the password. If the client supports the EPEP protocol, SRV_T_PWD automatically returns the decrypted password.

### SRV_PWD_ENCRYPT_VERSION

This new public enumerated type in Open Server has these values:

- SRV_NOENCRYPT_PWD (0)
- SRV_ENCRYPT_PWD (1) (Not implemented in Open Server)
- SRV_EXTENDED_ENCRYPT_PWD (2) (Not implemented in Open Server)
- SRV_EXTENDED_PLUS_ENCRYPT_PWD (3)

### SRV_T_PWD_ENCRYPT_VERSION

Use this new read-only property along with the srv_thread_props() function to retrieve the protocol version of the password encryption that retrieved the password. The type and possible values of this property are described in "SRV_PWD_ENCRYPT_VERSION" on page 29.

**Note** You cannot use this property to avoid clear-text transmission of passwords. When Open Server reads the client-supported password encryption versions, the password may already have been transmitted in clear text. However, you can use this property to verify that all client applications use the required password encryption algorithm.

### SRV_S_DISABLE_ENCRYPT

Use the SRV_S_DISABLE_ENCRYPT property to disable support for the native password negotiation. If this property is set, Open Server does not start the password negotiation protocols. The default value for this SRV_S_DISABLE_ENCRYPT is CS_FALSE.

## BCP --quoted-fname option

The current syntax of the command line parameter for BCP is "--quoted-fname".

The system accepts the string “quoted-fname” without blank space in between string. You can place the new parameter anywhere after data file names in the list of commandline parameters.

To use data file names containing special characters, besides using this option, quote your file names within double quotation marks each preceded by a backslash (\"). If the file names contain double quotation marks, put a backslash preceding each double quotation mark in the file names.

***Table 5: Examples***

| Data file name | With the updated syntax |
|---|---|
| fnamepart1,fnamepart2 | \” fnamepart1,fnamepart2\” |
| fnamepart1”fnamepart2 | \” fnamepart1\”fnamepart2\” |
| “fnamepart1”fnamepart2” | \”\” fnamepart1\”fnamepart2\”\” |

## Adaptive Server Enterprise extension module for Python

The Adaptive Server Enterprise extension module for Python has been enhanced to support DSN style connection properties.

### Support for DSN style connection properties

The connect() method accepts a new keyword argument named dsn.

The keyword argument is a string that specifies connection information. The syntax of a dsn string is:

name1=*value1*;name2=*value2*;...

Here name1 normally corresponds to a connection property or option.

The name string does not contain escaped characters. To show the equal sign and semicolon in the value string, escape those characters by preceding each with a backslash.

## Adaptive Server Enterprise extension module for Perl

The Adaptive Server Enterprise extension module for Perl has been enhanced to support new attributes and methods, new Perl database and statement handle attributes, multiple statements, dynamic SQL, bind parameters, stored procedures, private driver methods, text and image data handling, and error handling.

## Support for DSN style connection properties

The driver uses a DSN mechanism that allows certain attributes to be set at connection time.

The DSN attribute syntax is the same as the Open Source DBD::Sybase driver. Therefore, you need not change Perl scripts or maintain different versions for DBD::Sybase versus DBD::SybaseASE. However, DBD::SybaseASE does not support some attributes that are considered obsolete. See “Currently unsupported DSN syntax” on page 34.

### SybaseASE driver connect syntax

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

The dbi:SybaseASE: section obtains the package name of the driver so it can be loaded. When the DSN is passed into the driver, the system removes this part and the remaining string holds the key and value pairs to be dissected.

---

**Note** The *$user_id* and *$password* credentials are separate API arguments; they are not part of the DSN string.

---

The %attrib argument is an optional, comma-separated chain of key-value pairs that set options at connection time. They are passed into the driver and handled during a connect() call. For example:

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

### Attributes and methods

These attributes are currently supported when connecting to a server:

- server – specifies the server to which you are connecting. The driver currently assumes this option is set. If server is not specified, use the ENV{"DSQUERY"} mechanism to obtain a server name.
- database – specifies which database within the server is the target database at connect time. If no database is specified, the master database is used.
- hostname – specifies, in the value section, the host name that is stored in the sysprocesses table for this process. If no hostname is specified, the host on which the Perl application executes is used.

- language – specifies the locale to be used on this connection. If no language is specified, the internal default locale named CS_LC_ALL is used.
- charset – specifies the charset to be used on this connection. If no charset is specified, the internal default that is, utf8, is used.
- host; port – specifies the combination of host and port to use instead of relying on the interfaces file entries.

---

**Note** In the Perl DSN syntax, host and port are separate options. An alternative DSN form similar to the following is not currently supported:

```
host:port=mumbles:1234
```

When the host and port DSN options are provided with the intent of not using the interface file, the host and port must suffice to connect. If the DSN attribute “server=” is also provided with the host and port combination, the connection fails.

Therefore, the usage of either host and port must be used to establish a connection or server alone must be used. The two DSN attributes (server versus host/port) are mutually exclusive.

---

- timeout – specifies the connection timeout value. Set to 0 or a negative value for no timeout.
- loginTimeout – specifies the login timeout value, in seconds. The default value is 60 seconds. Set loginTimeout=*value in seconds* to enable this attribute.
- tds_keepalive – specifies the KEEP_ALIVE attribute on the connection. Set tds_keepalive=*1* to enable this attribute.
- packetSize – specifies the TDS packet size for the connection. By default, the lower bound, which is set in the driver, is 2048. The maximum value is determined by the server, and is not set in the driver.
- maxConnect – increases or decreases the number of connections allowed. The range of values is 1 – 128; the default is 25.
- encryptPassword – specifies whether to use password encryption. Set encryptPassword=*1* to enable this attribute.
- sslCAFile – specifies an alternate location for the *trusted.txt* file. Specify an absolute path of up to 256 characters.

- scriptName – specifies the chosen name of the top-level Perl script that drives the application. This name appears in the sysprocesses table as the application name. Absence of this value gives a default application name that is obtained from the Perl internal environment. This value can be as many as 256 characters.

  **Note** The application name fed into the SybaseASE Driver is either set through the DSN scriptName option or is derived from the Perl internal environment.

- interfaces – specifies an alternate location to the Sybase interfaces file. Same constraints apply to the sslCAFile and scriptName options.

You can repeat attribute values as long as they are recognized by the driver. Illegal attributes cause the DBI->connect() call to fail.

**Note** The attribute names follow the Open Source Sybase Perl driver.

DSN-specific example:

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

Alternatively, use the DSQUERY environment variable:

```
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-specific example combination:

```
$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);
```

### Currently unsupported DSN syntax

- tdsLevel
- kerberos; for example:

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

- bulkLogin; for example:

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

- serverType

## Currently supported database handle attributes

Currently supported database handle attributes include:

| Attribute | Description | Default |
|---|---|---|
| dbh->{AutoCommit} = (0\|1); | Disables or enables AutoCommit. | 0 (off) |
| dbh->{LongTruncOK} = (0\|1); | Disables or enables truncation of text and image types. | 0 |

| Attribute | Description | Default |
|---|---|---|
| dbh->{LongReadLen}=(int); | Sets the default read chunk size for text and image data. For example:<br>dbh->{LongReadLen} = 64000. | 32767 |
| dbh->{syb_show_sql} =(0\|1); | If set, the current statement is included in the error string returned by the $dbh->errstr mechanism. | 0 |
| dbh->{syb_show_eed} = (0\|1); | If set, the extended error information is included in the error string returned by $dbh->errstr. | 0 |
| dbh->{syb_chained_txn} = (0\|1); | If set, CHAINED transactions are used when AutoCommit is off.<br>Use this attribute only during the connect() call:<br>`$dbh = DBI->connect("dbi:SybaseASE:", $user, $pwd, {syb_chained_txn => 1});`<br>Using syb_chained_txn at any time with AutoCommit turned off forces a commit on the current handle.<br>When set to 0, an explicit BEGIN TRAN is issued as needed. | 0 |
| dbh->{syb_use_bin_0x} = (0\|1); | If set, BINARY and VARBINARY values are prefixed with '0x' in the result string. | 0 |
| dbh->{syb_binary_images} = (0\|1); | If set, image data is returned in raw binary format. Otherwise, image data is converted into a hexadecimal string. | 0 |
| dbh->{syb_quoted_identifier} =(0\|1); | Allows identifiers that conflict with Sybase reserved words if they are quoted using "identifier." | 0 |
| dbh->{syb_rowcount}=(int); | If set to a nonzero value, the number of rows returned by a SELECT, or affected by an UPDATE or DELETE statement are limited to the *rowcount* value.<br>Setting it back to 0 clears the limit. | 0 |
| dbh->{syb_flush_finish} = (0\|1); | If set, the driver drains any results remaining for the current command by actually fetching them. This can be used instead of a ct_cancel() command issued by the driver. | 0 |

| Attribute | Description | Default |
|---|---|---|
| dbh->{LongReadLen}=(int); | Sets the default read chunk size for text and image data. For example:<br>dbh->{LongReadLen} = 64000. | 32767 |
| dbh->{syb_show_sql} =(0\|1); | If set, the current statement is included in the error string returned by the $dbh->errstr mechanism. | 0 |
| dbh->{syb_show_eed} = (0\|1); | If set, the extended error information is included in the error string returned by $dbh->errstr. | 0 |
| dbh->{syb_chained_txn} = (0\|1); | If set, CHAINED transactions are used when AutoCommit is off.<br>Use this attribute only during the connect() call:<br>`$dbh = DBI->connect("dbi:SybaseASE:", $user, $pwd, {syb_chained_txn => 1});`<br>Using syb_chained_txn at any time with AutoCommit turned off forces a commit on the current handle.<br>When set to 0, an explicit BEGIN TRAN is issued as needed. | 0 |
| dbh->{syb_use_bin_0x} = (0\|1); | If set, BINARY and VARBINARY values are prefixed with '0x' in the result string. | 0 |
| dbh->{syb_binary_images} = (0\|1); | If set, image data is returned in raw binary format. Otherwise, image data is converted into a hexadecimal string. | 0 |
| dbh->{syb_quoted_identifier} =(0\|1); | Allows identifiers that conflict with Sybase reserved words if they are quoted using "identifier." | 0 |
| dbh->{syb_rowcount}=(int); | If set to a nonzero value, the number of rows returned by a SELECT, or affected by an UPDATE or DELETE statement are limited to the *rowcount* value.<br>Setting it back to 0 clears the limit. | 0 |
| dbh->{syb_flush_finish} = (0\|1); | If set, the driver drains any results remaining for the current command by actually fetching them. This can be used instead of a ct_cancel() command issued by the driver. | 0 |

| Attribute | Description | Default |
|---|---|---|
| dbh->{LongReadLen}=(int); | Sets the default read chunk size for text and image data. For example:<br>dbh->{LongReadLen} = 64000. | 32767 |
| dbh->{syb_show_sql} =(0\|1); | If set, the current statement is included in the error string returned by the $dbh->errstr mechanism. | 0 |
| dbh->{syb_show_eed} = (0\|1); | If set, the extended error information is included in the error string returned by $dbh->errstr. | 0 |
| dbh->{syb_chained_txn} = (0\|1); | If set, CHAINED transactions are used when AutoCommit is off.<br>Use this attribute only during the connect() call:<br>`$dbh = DBI->connect("dbi:SybaseASE:", $user, $pwd, {syb_chained_txn => 1});`<br>Using syb_chained_txn at any time with AutoCommit turned off forces a commit on the current handle.<br>When set to 0, an explicit BEGIN TRAN is issued as needed. | 0 |
| dbh->{syb_use_bin_0x} = (0\|1); | If set, BINARY and VARBINARY values are prefixed with '0x' in the result string. | 0 |
| dbh->{syb_binary_images} = (0\|1); | If set, image data is returned in raw binary format. Otherwise, image data is converted into a hexadecimal string. | 0 |
| dbh->{syb_quoted_identifier} =(0\|1); | Allows identifiers that conflict with Sybase reserved words if they are quoted using "identifier." | 0 |
| dbh->{syb_rowcount}=(int); | If set to a nonzero value, the number of rows returned by a SELECT, or affected by an UPDATE or DELETE statement are limited to the *rowcount* value.<br>Setting it back to 0 clears the limit. | 0 |
| dbh->{syb_flush_finish} = (0\|1); | If set, the driver drains any results remaining for the current command by actually fetching them. This can be used instead of a ct_cancel() command issued by the driver. | 0 |

| Attribute | Description | Default |
|---|---|---|
| dbh->{syb_date_fmt} = datefmt string | This private method sets the default date conversion and display formats. See "Default date conversion and display format" on page 49. | |
| dbh->{syb_err_handler} | Perl subroutine that can be created to execute an error handler or report before the regular error handling takes place. Useful for certain classes of warnings. See "Error handling" on page 52. | 0 (not present) |
| dbh->{syb_failed_db_fatal} = (0\|1) | If the DSN has a database=*mumbles* attribute/value pair and this database does not exist at connection time, the DBI->connect() call fails. | 0 |
| dbh->{syb_no_child_con} = (0\|1); | If set, the driver disallows multiple active statement handles on the dbh. In this case, a statement can be prepared but must be executed to completion before another statement prepare is attempted. | 0 |
| dbh-> {syb_cancel_request_on_erro r}= (0\|1); | If set, when a multistatement set is executed and one statement fails, sth->execute() fails. | 1 (on) |
| dbh-> {syb_bind_empty_string_as_ null} = (0\|1); | If set, a NULLABLE column attribute returns an empty string (one space) to represent the NULL character. | 0 |
| dbh->{syb_disconnect_in_child} = (0\|1); | Handles closed connections across a fork. The DBI causes connections to be closed if a child dies. | 0 |
| dbh->{syb_enable_utf8} = (0\|1); | If set, UNICHAR, UNIVARCHAR, and UNITEXT are converted to utf8. | 0 |
| sth->syb_more_results} = (0\|1); | See "Multiple result sets" on page 41. | |
| sth->{syb_result_type} = (0\|1); | If set, returns the numeric result number instead of the symbolic CS_ version. | 0 |
| sth->{syb_no_bind_blob} = (0\|1); | If set, image or text columns are not returned upon sth->{fetch} or other variations. See "Text and image data handling" on page 50. | 0 |

| Attribute | Description | Default |
|---|---|---|
| sth->{syb_do_proc_status} = (0\|1); | Forces $sth->execute() to fetch the return status of a stored procedure executed in the SQL stream.<br>If the return status is nonzero, $sth->execute() returns `undef` (that is, it fails).<br>Setting this attribute does not affect existing statement handles. However, it affects those statement handles that are created after setting it.<br>To revert behavior of an existing $sth handle, execute:<br>$sth->{syb_do_proc_status} = 0; | 0 |

### Unsupported database handle options

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

**Note** Perl scripts attempting to use these options generate an error.

## Perl supported datatypes

The Perl driver currently supports string, numeric, and date and time datatypes.

| String types | Numeric types | Date and time datatypes |
|---|---|---|
| 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 | |

**Note** Perl returns numeric and decimal types as strings. Other datatypes are returned in their respective formats.

The default time/date format used by the Sybase ASE driver is the short format, for example, Aug 7 2011 03:05PM.

This format is based on the C (default) locale. See “Default date conversion and display format” on page 49 for other date and time formats supported.

## Using multiple statements

Adaptive Server can handle multistatement SQL in a single batch as shown in this example:

```
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();
```

If any of these statements fail, sth->execute() returns `undef`. If AutoCommit is on, statements that complete successfully may have inserted data in the table, which may not be the result you expect or want.

## Multiple result sets

The Perl driver allows you to prepare multiple statements with one call and execute them with another single call. For example, executing a stored procedure that contains multiple selects returns multiple result sets.

Results of multiple statements prepared with one call are returned to the client as a single stream of data. Each distinct set of results is treated as a normal single result set, which means that the statement handle's fetch() method returns `undef` at the end of each set.

The CT-Lib API ct_fetch() returns `CS_END_RESULTS` that the driver converts to `undef` after the last rows have been retrieved.

The driver allows the application to obtain the result type by checking sth->{syb_result_type}. You can then use the sth->{syb_more_results} statement handle attribute to determine if there are additional result sets still to be returned. The (numerical) value returned by sth->{syb_results_type} is one of:

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

Example for multiple result sets:

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

Sybase recommends that you use this if you expect multiple result sets.

**Note** The Perl driver currently does not support cursors using the ct_cursor() API. Therefore, the driver does not report CS_CURSOR_RESULT.

### Multiple active statements on a DatabaseHandle (dbh)

There can be multiple active statements on a single database handle by opening a new connection in the $dbh->prepare() method if there is already an active statement handle on this $dbh.

The dbh->{syb_no_child_con} attribute controls whether this feature is on or off. By default, DatabaseHandle is off, which indicates that multiple statement handles are supported. If it is on, multiple statements on the same database handle are disabled.

**Note** If AutoCommit is off, multiple statement handles on a single $dbh are unsupported. This avoids deadlock problems that may arise. Also, using multiple statement handles simultaneously provides no transactional integrity, as different physical connections are used.

## Supported character lengths

Supported character lengths for different types of identifiers.

The names of Sybase identifiers, such as tables and columns, can exceed 255 characters in length.

Logins, application names, and password lengths that are subject to TDS protocol limits cannot exceed 30 characters.

## Configuring locale and charsets

You can configure the Perl driver of CT-Library locale and charset using the DSN attributes charset and language. The driver's default character set is *UTF8* and the default locale is *CS_LC_ALL*.

## Dynamic SQL support, placeholders, and bind parameters

The Perl driver supports dynamic SQL, including parameter usage, as shown in this example:

```
$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";
}
```

**Note** The Perl driver supports the '?' style parameter, but ':1' placeholder types. You cannot use placeholders to bind a text or image datatype.

DBD::SybaseASE uses the Open Client ct_dynamic() family of APIs for the prepare() method. See the *Sybase Open Client C Programmers guide* for information about "?" style placeholder constraints and general dynamic SQL usage.

This is another example showing dynamic SQL support:

```
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);
```

**Note** The last statement throws an extended error information (EED) as the date is invalid. In the Perl script, set dbh->{syb_show_eed} = *1* before execution to write the Adaptive Server error message in the dbh->errstr.

Another example that illustrates the "?" style placeholder:

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


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

## Stored procedure support for placeholders

The Adaptive Server Enterprise database driver for Perl supports stored procedures that include both input and output parameters. Stored procedures are handled in the same way as any other Transact-SQL® statement. However, Sybase stored procedures return an extra result set that includes the return status that corresponds to the return statement in the stored procedure code. This extra result set, named CS_STATUS_RESULT with numeric value 4043, is a single row and is always returned last.

The driver can process the stored procedure using a special attribute, $sth->{syb_do_proc_status}. If this attribute is set, the driver processes the extra result set, and places the return status value in $sth->{syb_proc_status}. An error is generated if the result set is a value other than 0.

### Examples

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

This example illustrates the use of positional parameters:

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

```
        $sth->execute('one', 'two');
```

You cannot mix positional and named parameters in the same prepare statement; for example, this statement fails on the first parameter:

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

If the stored procedure returns data using output parameters, you must declare them first:

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

You cannot call stored procedures with bound parameters, as in:

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

This works as follows:

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

Because stored procedures almost always return more than one result set, use a loop until syb_more_results is 0:

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

**Parameter examples**

```
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
```

If your stored procedure returns only OUTPUT parameters, you can use:

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

This returns an array for all the OUTPUT parameters in the procedure call and ignores any other results. The array is undefined if there are no OUTPUT parameters or if the stored procedure fails.

## Generic example

```
$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};
}
```

The OUTPUT parameters are returned as a single row in a special result set.

## Parameter types

The driver does not attempt to determine the correct parameter type for each parameter. The default for all parameters is driver defaults all parameters to the ODBC style SQL_CHAR value, unless you use bind_param() with a type value set to a supported bind type.

The driver supports these ODBC style bind types:

- 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

The ODBC types are mapped in the driver to equivalent Adaptive Server datatypes. See the *Adaptive Server Enterprise ODBC Driver by Sybase User Guide 15.7*.

Execute the stored procedure, sp_datatype_info to get a full list of supported types for the particular Adaptive Server. For example:

```
$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...
```

---

**Note** Once you have set a column type for a parameter, you cannot change it unless you deallocate and retry the statement handle. When binding SQL_NUMERIC or SQL_DECIMAL data, you may get fatal conversion errors if the scale or the precision exceeds the size of the target parameter definition.

---

For example, consider this stored procedure definition:

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

```
$sth->bind_param(1, 3.456, SQL_NUMERIC);
```

which generates this error:

```
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.
```

Set the arithabort option as follows to ignore these errors:

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

See the Adaptive Server reference documentation.

## Supported private driver methods

dbh->syb_isdead() returns a true or false representation of the state of the connection. A false return value may indicate a specific class or errors on the connection, or that the connection has failed.

$sth->syb_describe() returns an array that includes the description of each output column of the current result set. Each element of the array is a reference to a hash that describes the column.

You can set the description fields such as NAME, TYPE, SYBTYPE, SYBMAXLENGTH, MAXLENGTH, SCALE, PRECISION, and STATUS, as shown in this example:

```
$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) {
    ....
}
```

**Note** The STATUS field is a string which can be tested for the following values: CS_CANBENULL, CS_HIDDEN, CS_IDENTITY, CS_KEY, CS_VERSION_KEY, CS_TIMESTAMP and CS_UPDATABLE, CS_UPDATECOL and CS_RETURN.

See the Open Client documentation.

## Default date conversion and display format

You can set your own default date conversion and display format using the syb_data_fmt() private method.

Sybase date format depends on the locale settings for the client. The default date format is based on the 'C' locale, for example, Feb 16 2012 12:07PM.

This same default locale supports several additional input formats:

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

Use dbh->{syb_date_fmt} with a string as argument, to change the date input and output format.

***Table 6: Supported date/time formats***

| Date format | Example |
|---|---|
| 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 |
| LONGMS | Nov 15 2011 11:30:33.532315PM |

The Adaptive Server Enterprise database driver for Perl supports all date and time values supported up to version 15.7.

## Text and image data handling

The Adaptive Server Enterprise database driver for Perl supports image and a text type for LONG/BLOB data. Each type can as much as 2GB of binary data.

The default size limit for text/image data is 32KB. Use the LongReadLen attribute to change this limit, which is set by a call to the fetch() API.

You cannot use bind parameters to insert text or image data.

When using regular SQL, image data is normally converted to a hex string, but you can use the syb_binary_images handle attribute to change this behavior. As an alternative, you can use a Perl function similar to $binary = pack("H*", $hex_string); to perform the conversion.

As the DBI has no API support for handling BLOB style (text/image) types, the *SybaseASE.pm* file includes a set of functions you can install, and use in application-level Perl code to call the Open Client ct_get_data() style calls. The syb_ct_get_data() and syb_ct_send_data() calls are wrappers to the Open Client functions that transfer text and image data to and from Adaptive Server.

**Example**

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

You can use the syb_ct_get_data() call to fetch the image/text data in raw format, either in one piece or in chunks. To enable this call, set the dbh->{syb_no_bind_blob} statement handle to *1*.

The syb_ct_get_data() call takes these arguments: the column number (starting at 1) of the query, a scalar reference, and a byte count. A byte count of 0 reads as many bytes as possible. The image/text column must be last in the select list for this call to work.

The call sequence is:

```
$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() returns the number of bytes that were fetched, if you are fetching chunks of data, you can use:

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

### Other TEXT/IMAGE APIs

The syb_ct_data_info() API fetches or updates the CS_IODESC structure for the image/text data item you want to update, as shown in this example:

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

- *$action* – CS_SET or CS_GET.
- *$column* – the column number of the active select statement (ignored for a CS_SET operation).
- *$attr* – a hash reference that sets the values in the structure.

You must fist call syb_ct_data_info() with CS_GET to fetch the CS_IODESC structure for the image/text data item you want to update. Then update the value of the total_txtlen structure element to the length (in bytes) of the image/text data you are going to insert. Set the log_on_update to true to enable full logging of the operation.

Calling syb_ct_data_info() with a CS_GET fails if the image/text data for which the CS_IODESC is being fetched is NULL. Use standard SQL to update the NULL value to non-NULL value (for example, an empty string) before you retrieve the CS_IODESC entry.

In this example, consider updating the data in the image column where the id column is 1:

1 Find the CS_IODESC data for the data:

```
$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. Update with the CS_IODESC values:

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

3. Set the size of the new data item to be inserted and make the operation unlogged:

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

4. To transfer the data in a single chunk:

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

5. To commit the operation:

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

## Error handling

All errors from the Adaptive Server database driver for Perl and CT-Lib are propagated into the DBI layer.

Exceptions include errors or warnings that must be reported during driver start-up, when there is no context available yet.

The DBI layer performs basic error reporting when the PrintError attribute is enabled. Use DBI trace method to enable tracing on DBI operations to track program- or system-level problems.

Examples of adding more detailed error messages (server messages) are as follows:

- Set dbh->{syb_show_sql} = *1* on the active dbh to include the current SQL statement in the string returned by $dbh->errstr.
- Set dbh->{syb_show_eed} = *1* on the active dbh to add extended error information (EED) such as duplicate insert failures and invalid date formats to the string returned by $dbh->errstr.
- Use the syb_err_handler attribute to set an ad hoc error handler callback (that is, a Perl subroutine) that gets called before the normal error handler performs its processing. If this subroutine returns 0, the error is ignored. This is useful for handling PRINT statements in Transact-SQL, and showplan output and dbcc output.

  The subroutine is called with parameters that include the Sybase error number, the severity, the state, the line number in the SQL batch, the server name (if available), the stored procedure name (if available), the message text, the SQL text and the strings "client" or "server" to denote type.

## Configuring security services

Use the *ocs.cfg* and *libtcl.cfg* files to configure security options.

For a connection, use *ocs.cfg* to set directory and security properties. Edit *libtcl.cfg* to load security and directory service drivers.

**Note** In the *ocs.cfg* file, add an entry for the application name so you can set that driver-specific option.

To encrypt passwords, use the encryptPassword DSN option. For example:

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

## Examples

Use sample programs to view the basic usage of stored procedure and retrieve rows from the pubs2 authors table.

### Example 1

Use the following sample program to view the basic usage of stored procedures in Perl. This program connects to a server, creates two stored procedures, calls prepare, binds, or executes the procedures, prints the results to STDOUT, disconnects, and exits the program.

```
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;
```

**Example 2**

Use the following sample program to retrieve rows from the pubs2 authors table, insert them into tempdb, and append new rows for batch insert. The program then prints the updated authors table to STDOUT, disconnects, and exits.

```
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;
```

## New features for ESD #5

ESD #5 introduces new functionality for jConnect 7.07, Adaptive Server ODBC Driver 15.7, and Adaptive Server ADO.NET Data Provider15.7.

### Adaptive Server ADO.NET Data Provider support for Transact-SQL queries with COMPUTE clause

Adaptive Server ADO.NET Data Provider now supports Transact-SQL queries that include a COMPUTE clause.

A COMPUTE clause lets you include detail and summary results in a single select statement.

The summary row follows the detail rows of a specific group, as shown here:

```
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)
```

When Adaptive Server ADO.NET Data Provider executes a select statement that includes a COMPUTE clause, the provider returns multiple result sets to the client. The number of result sets depends on the number of unique groupings available. Each group contains one result set for the detail rows and one result set for the summary. The client must process all result sets to fully process the rows returned; if it does not, only the detail rows of the first group of data are included in the first result set returned.

See the *Adaptive Server Enterprise Transact-SQL Users Guide* for more information about the COMPUTE clause.

See the *ADO.NET Programmers Guide* on the Microsoft Web site for more information about processing multiple result sets.

## New SSIS Custom Data Flow Destination component for faster data transfers to Adaptive Server

Adaptive Server ADO.NET Data Provider distribution now includes a SQL Server Integration Services (SSIS) Custom Data Flow Destination component, which performs faster data transfer in to Adaptive Server destinations. The faster data transfers use the Adaptive Server bulk-insert protocol supported by AseBulkCopy class. This component, named SybaseAdaptiveServerAdoNetDestination, is installed along with the Adaptive Server ADO.NET Data Provider and the assembly files in: *%SYBASE%\DataAccess\ADONET\SybaseAdaptiveServerAdoNetDestination.dll* (32–bit systems) and *%SYBASE%\DataAccess64\ADONET\SybaseAdaptiveServerAdoNetDestination.dll* (64–bit systems).

### ❖ Configuring Adaptive Server ADO.NET Destination SSIS component

1. Copy the *SybaseAdaptiveServerAdoNetDestination.dll* to *C:\Program Files\Microsoft SQL Server\100\DTS\PipelineComponents* and *C:\Program Files (x86)\Microsoft SQL Server\100\DTS\PipelineComponents*.

2 From any Microsoft SQL Server directory on your local drive, register the *SybaseAdaptiveServerAdoNetDestination.dll* assembly.

---

**Note** If you have not run the SDK setup on this machine, see the DDEX Provider for Adaptive Server section in the *Adaptive Server Enterprise ADO.NET Data Provider Users Guide* to manually register the Adaptive Server ADO.NET Data Provider and the DDEX Provider.

---

3 Start SQL Server Business Intelligence Studio.

4 On the Toolbox tab, right-click Data Flow Destinations and select Choose Items. The Choose Toolbox Items window appears.

5 Select the SSIS Data Flow Items tab. Click Sybase Adaptive Server Enterprise ADO NET Destination, then click OK. Select Toolbox | Data Flow Destinations to see the Sybase Adaptive Server ADO NET Destination component.

6 To create an SSIS project, select File | New | Project | Integration Services Project menu. Create or drag and drop a Control Flow object from the Control Flow Items toolbox.

7 From the Data Flow Destinations and Data Flow Sources Toolbox tab, drag and drop Sybase Adaptive Server ADO NET Destination Component and ADO NET Source Component onto the Data Flow tab.

8 If a source or destination connection is not available in Connection Managers window, right-click in the Connection Managers window, and select New ADO.NET Connection. Select the already existing Data connection, or click New.

9 To create a new connection to the destination Adaptive Server, click New button in the Configure ADO.NET Connection Manager window, and then select Sybase Adaptive Server Enterprise Data Provider.

10 In the Connection Manager window, enter your connection properties.

11 To enable bulk insert, in the Additional Connection Props text box, enter:

```
enablebulkload=1
```

---

**Note** See AseBulkCopy in the *Adaptive Server Enterprise ADO.NET Data Provider Users Guide* for more details about utilizing bulk insert functionality.

---

12 Click OK.

13 For the ADO.NET Source in your Data Flow, setup the connection and data access mode. After you connect the data flow path from your ADO.NET Source, right-click Sybase Adaptive Server ADO NET Destination Component, and choose Show Advanced Edit.

14 From the Connection Manager tab, select ASE connection from the Connection Manager field. From the Component Properties tab, set the TableName property to the destination table name.

15 Select the Input Columns tab, and select the Name check box. This will select all the columns specified by the source table.

16 Click OK.

The connection is established. See *Microsoft SSIS* documentation for more information about data transfer.

## jConnect dynamic logging levels

jConnect has been enhanced to allow application users to set message granularity to Level.FINE, Level.FINER, and Level.FINEST. For example:

- When a user sets the logging level to Level.FINE on SybConnection class, jConnect reports:

  Dr1_Col setClientInfo(Properties)

- Level.FINER on SybConnection class reports:

  Dr1_Co1 setClientInfo(Properties.size = [3])

- Level.FINEST on SybConnection class reports:

  Dr1_Co1 setClientInfo(Properties = [[ClientUserValue, ApplicationNameValue, ClientHostnameValue]])

See *jConnect for JDBC Programmers Reference*.

## Package name changed in jConnect for converter classes

In jConnect 7.07, the package name and file path for all character-set converter classes has been changed. The character set converter class files has been moved from *com/sybase/jdbc4/utils* to *com/sybase/jdbc4/charset*. Package name changes for character-set converter classes in jConnect 7.07 include:

- com.sybase.jdbc4.utils.TruncationConverter has been changed to com.sybase.jdbc4.charset.TruncationConverter
- com.sybase.jdbc4.utils.PureConverter has been changed to com.sybase.jdbc4.charset.PureConverter

**Note** If you have declared classes that extend character-set converter classes to use the full package name, you must change the package name from com.sybase.jdbc4.utils to com.sybase.jdbc4.charset.

Sybase recommends that you use wildcard character imports instead of coding the class reference. For example:

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

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

The converter class references for package name are resolved by the import statements.

## Increased PreparedStatement parameter limit in jConnect

In previous versions, the maximum number of parameters for PreparedStatement was limited to 2048. jConnect 7.07 now supports 32767 parameters, when connected to Adaptive Server that also supports the larger limit.

## New SkipRowCountResults connection property for Adaptive Server ODBC Driver

The SkipRowCountResults connection property can be used to control how the ODBC Driver treats statements that return row count results. UPDATE, INSERT and DELETE statements return row count results. SELECT statements return result sets. An ODBC application may execute a batch of statements that uses a mix statements returning row counts or result sets.

When SkipRowCountResults is set to 1 (the default), the Adaptive Server ODBC Driver skips any row count results. After executing a batch of statements using SQLExecDirect or SQLExecute, the ODBC application is positioned on the first result set. Subsequent calls to SQLMoreResults will skip over row count results and the application is positioned on the next available result set.

When SkipRowCountResults is set to 0, the Adaptive Server ODBC Driver stops at each result set or row count. After executing a batch of statements using SQLExecDirect or SQLExecute, the application is positioned on the first available result which can be either a result set or a row count. The ODBC application can use SQLFetch to retrieve a result set or SQLRowCount to retrieve the row count results. Subsequent calls to SQLMoreResults will position the application to the next available result, which can be either a result set or row count.

## Support for AF_UNIX sockets in Adaptive Server ODBC Driver

The Adaptive Server ODBC Driver now supports AF_UNIX sockets to communicate to Adaptive Server. This support is currently limited to the Linux x86-64 64-bit platform. You can use the AF_UNIX socket when both the ODBC application and Adaptive Server are located on the same host, and both are configured to use AF_UNIX sockets. The AF_UNIX sockets provide better performance than TCP/IP sockets. To enable AF_UNIX sockets from ODBC, set these connection strings properties:

- networklibraryname=afunix – informs the Adaptive Server ODBC Driver that AF_UNIX socket is used.
- server=<full path to the pipe> – path to the AF_UNIX socket. For example, */tmp/test/demo_socket*.

See the *Sybase Adaptive Server Enterprise* documentation for more information on configuring Adaptive Server to use AF_UNIX sockets.

### AdjustLargePrecisionAndScale connection property for Adaptive Server ODBC Driver

In versions earlier than 15.7, the Adaptive Server ODBC Driver did not support calls to SQLSetDescField(), to set scale and precision of numeric or decimal columns. Any calls to this API were ignored, and the Adaptive Server ODBC Driver set the precision and scale of the column based on the value received. As Adaptive Server supports a precision larger than the ODBC numeric structure, the Adaptive Server ODBC Driver further scaled down the values received from the server as needed to accommodate them within the ODBC numeric structure. In versions 15.7 and later, the Adaptive Server ODBC Driver no longer ignores the calls to SQLSetDescField() that set the precision and scale of the numeric or decimal column. It is therefore possible to find that ODBC Applications that worked before now receive data overflow errors with the new Adaptive Server ODBC Driver. The AdjustLargePrecisionAndScale property allows the earlier behavior to continue, and enables the Adaptive Server ODBC Driver to select the optimal precision and scale to accommodate the value received from the server.

By default, AdjustLargePrecisionAndScale is 0, which causes the Adaptive Server ODBC Driver to accept the calls made to SQLSetDescField() API to set precision or scale.

When you set the AdjustLargePrecisionAndScale connection property to 1, the Adaptive Server ODBC Driver ignores any calls made to SQLSetDescField() API to set precision or scale, and uses the precision and scale of actual data value.

For more information about SQLSetDescField(), see the Microsoft Developers Network at http://msdn.microsoft.com/.

## New features for ESD #4

ESD #4 introduces new functionality for Open Client 15.7 and Open Server 15.7, SDK 15.7, Adaptive Server Enterprise extension module for Python 15.7, Adaptive Server Enterprise extension module for PHP 15.7, and Adaptive Server Enterprise data provider for Perl 15.7.

## Open Client 15.7 and Open Server 15.7 features in ESD #4

Open Client 15.7 and Open Server 15.7 have been enhanced to provide new functionality including stricter permissions for Open Client and Open Server files (UNIX), batched parameters, and new safe string handling routines.

### Stricter permissions for Open Client and Open Server files (UNIX only)

Starting with ESD#4, newly generated Open Client and Open Server files have the following stricter permissions:

***Table 7: Files and their permission settings***

| Files | Permission |
|---|---|
| Interfaces files | rw- r-- r-- (644) |
| BCP data file | rw- r-- --- (640) |
| BCP format file | rw- r-- --- (640) |
| BCP output file | rw- --- --- (600) |
| BCP error file | rw- --- --- (600) |
| ISQL output file (-o option) | rw- --- --- (600) |
| ISQL Command history file | rw- --- --- (600) |
| ISQL temporary file | rw- --- --- (600) |
| ISQL output redirection | rw- --- --- (600) |
| Open Server log file | rw- --- --- (600) |
| LDAP debug log file | rw- --- --- (600) |
| Kerberos debug log file | rw- --- --- (600) |
| Netlib trace output file | rw- --- --- (600) |
| DCL trace output file | rw- --- --- (600) |

**Note** These permissions apply to newly generated files only; existing files retain their permissions (typically rw- rw- rw- (666)).

**Note** Permissions of files on Microsoft Windows remain unchanged.

## New SYBOCS_TCL_CFG environment variable for setting alternate path to libtcl*.cfg files

Starting with ESD#4, you can use the new SYBOCS_TCL_CFG environment variable to set the alternate full path name of the *libtcl.cfg* and *libtcl64.cfg* files. For example:

Windows:

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

UNIX:

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

By default, the *libtcl.cfg* and *libtcl64.cfg* files are searched in the *%SYBASE%\%SYBASE_OCS%\ini* directory on Windows and in the *$SYBASE/$SYBASE_OCS/config* directory on UNIX.

You can also use the CS_LIBTCL_CFG property to set the alternate path for the *libtcl.cfg* and *libtcl64.cfg* files.

## New isql command line option --URP to set universal remote password

Use the new --URP command line option to enable setting the universal remote password for clients accessing Adaptive Server.

isql --URP *remotepassword*

*remotepassword* is the universal remote password.

Examples

```
%isql --URP “ASEremotePW”
```

## New linux64 and nthread_linux64 settings for SYBPLATFORM

`linux64` and `nthread_linux64` (for threaded applications) are now valid settings for the *SYBPLATFORM* environment variable that can be used for compiling Open Client and Open Server sample applications on Linux x86-64 64-bit. The existing `linuxamd64` and `nthread_linuxamd64` settings remain valid for the same use.

## LAN Manager driver for Microsoft Windows 64-bit

Open Client and Open Server includes *libsybsmssp64.dll,* which is a 64-bit LAN Manager driver for Microsoft Windows x86-64 64-bit. *libsybsmssp64.dll* is located in *%SYBASE%\%SYBASE_OCS%\dll*; its behavior is similar to the 32-bit driver *libsybsmss.dll*.

## Support for batched parameters

Starting with ESD #4, Open Client and Open Server allow multiple sets of command parameters to be sent without ending the command itself. In an Open Client application, use the new ct_send_params() routine repeatedly to transfer parameters without needing to process the results of the previous command and without needing to resend the command itself. In an Open Server application, set SRV_S_PARAM_BATCHING property to CS_TRUE.

### ct_send_params

Description

Send command parameters in batches.

Syntax

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

Parameters

- *cmd*

  A pointer to a CS_COMMAND structure.

- *reserved*

  Set to CS_UNUSED. This is a placeholder reserved for possible future use.

Return value

ct_send_params returns:

| Returns | Indicates |
|---|---|
| CS_SUCCEED | The routine completed successfully. |
| CS_FAIL | The routine failed. |

Usage

A call to this function sends the parameters indicated earlier using ct_param() or ct_setparam(). To stop sending parameters, use a ct_send() call after the last ct_send_params() call. This signals the end of the parameters and completes the current command.

- The first ct_send_params() call sends the actual command, the parameter formats for all parameters, and the first set of parameters to the server. Subsequent calls only send more parameters without format.
- The network buffer containing the parameters gets flushed during every call to ct_send_params() so that the server can start processing the command.
- Unlike ct_send(), ct_send_params() does not end the current command. You can call ct_send_params() repeatedly to send multiple sets of parameters.

- The handling of the results starts only after a ct_send() call to complete the command. If ct_results() is called before ct_send(), an error results.

### Rebinding using ct_setparam()

When sending multiple sets of parameters, an application may need to point CT-Library to other locations in memory than for the previous set of parameters. To rebind the parameters, use ct_setparam() to provide a different location for the data. Here is the existing ct_setparam() declaration:

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

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

Provide new values for *data*, *datalenp* and *indp* parameters in ct_setparam() call to bind to different memory locations.

After a ct_send_params() call, the format of the parameters cannot be changed. Any calls to ct_setparam() made after a call to ct_send_params() must therefore pass a NULL value for *datafmt*.

Only parameters initially bound with ct_setparam() can be rebound.

### Batched parameters support to Server-Library

To enable batched parameter support in Open Server Server-Library, set the SRV_S_PARAM_BATCHING server property to CS_TRUE. For example, before srv_run():

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

Then, srv_xferdata() has two new return codes when a command contains multiple sets of command parameters.

- CS_PARAMS_MORE - indicates parameters have been successfully copied and there are more parameters in the batch.
- CS_PARAMS_END - indicates parameters have been successfully copied. This is the last set of parameters in the batch.

### Example programs

Two new CT-Library sample programs are available:

- *batch_lang.c* - demonstrates how ct_send_params() can be used with a language statement. This sample uses ct_send_params() repeatedly to insert lines read from a file into a table. Since it uses the same location for the parameters for every line read, it does not need to call ct_param() or ct_setparam() in between calls to ct_send_params().
- *batch_dynamic.c* - uses dynamic SQL and sends parameters to the server for which the data resides at different memory locations. Therefore, this sample also demonstrates how ct_setparam() can be used to rebind to different variables before calling ct_send_params() again.

The *ctos* sample program has been updated to include:

- Turn on the SRV_S_PARAM_BATCHING server property.
- Use ct_setparams() to bind CT-Lib to the location of the data.
- Handle the new return values from srv_xferdata()
- Call ct_send_params() for each set of command parameters.

## New CS-Library string handling routines

### cs_strlcpy

Description

Safe string copy function. Copies at most *target_size*-1 characters from *source_str* to *target_str*, truncating if necessary. The result is always a null terminated string except when *source_str* or *target_str* are NULL, or *target_size* is 0.

Syntax

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

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

Parameters

- *target_str*

  The target string where source string is to be copied.
- *source_str*

  The source string to be copied.
- *target_size*

Size of the target string

Return value

- 0 if *source_str* is NULL, *target_str* is NULL, or *target_size* is 0.
- *target_size* in case of an overflow.
- strlen(*source_str*) in all other cases.

### cs_strlcat

Description

Safe string concatenation function. Appends at most *target_size* - strlen(*target_str*) - 1 characters of *source_str* to *target_str*. The result is always a null terminated string, except when *source_str* or *target_str* are NULL, or *target_size* is 0, or the string pointed to by *target_str* is longer than *target_size* bytes.

Syntax

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

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

Parameters

- *target_str*

  The target string where source string is to be appended.
- *source_str*

  The source string to be appended.
- *target_size*

  Size of the target string

Return value

- 0 if *source_str* is NULL, *target_str* is NULL, or *target_size* is 0
- *target_size* in case of an overflow
- strlen(*target_str*) + strlen(*source_str*) in all other cases

### cs_snprintf

Description

A common snprintf like function for all platforms, providing formatted output conversion. The result is always a null terminated string.

Syntax

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

Parameters

- *str*

  String into which the output is written to.
- *size*

Maximum number of bytes to write.

- *format*

  Character string composed of zero or more conversion directives.

Return value

None

## SDK 15.7 features for jConnect and Adaptive Server Drivers and Providers in ESD #4

ESD #4 introduces new functionality for jConnect for JDBC 7.07, Adaptive Server Enterprise ODBC Driver 15.7, Adaptive Server Enterprise OLE DB Provider 15.7, and Adaptive Server Enterprise ADO.NET Data Provider 15.7.

### Granular and predicated permissions

Starting with Adaptive Server 15.7 ESD #2, role-privilege management model has been enhanced:

- New grantable system privileges that are granular have been added to enforce principles of Separation of Duties (SOD) and Least Privilege (LP). These grantable system permissions can be server-wide privileges or database-wide privileges.
- System-defined roles *sa_role*, *sso_role*, *oper_role*, *replication_role*, and *keycustodian_role* are now reconstructed as *privilege containers* consisting of a set of explicitly granted privileges.
- Custom roles can now be created from out-of-box system-defined roles by granting or revoking privileges.
- CREATE PROCEDURE statement now supports a new EXECUTE AS OWNER | CALLER option. Then, ASE checks runtime permissions, executes DDL, and resolves object names as procedure owner or as procedure caller.
- The enhanced role-privilege management model is enabled by using the new enable granular permissions configuration option.

See Adaptive Server Enterprise 15.7 ESD #2 documentation.

jConnect for JDBC, Adaptive Server Enterprise ODBC Driver, Adaptive Server Enterprise OLE DB Provider, and Adaptive Server Enterprise ADO.NET Data Provider support the new role-privilege management model when connected to an Adaptive Server with the new model enabled.

To support returning information about the predicate used to grant predicated permissions, the following methods return an additional column named PREDICATE:

- ODBC – SQLColumnPrivileges() and SQLTablePrivileges()
- JDBC – ResultSet getColumnPrivileges() and ResultSet getTablePrivileges()
- OLE DB – IDBSchemaRowset::GetRowset(DBSCHEMA_COLUMN_PRIVILEGES) and IDBSchemaRowset::GetRowset(DBSCHEMA_TABLE_PRIVILEGES)

If granular permissions are set up on the database, the methods return additional rows to convey the granular permissions.

There is no change in the behavior of the ADO.NET methods.

## alter table drop column without datacopy

Adaptive Server version 15.7 ESD #2 allows you to drop columns from a table without performing a data copy. This reduces the amount of time required for alter table drop column to run. See Adaptive Server Enterprise 15.7 ESD #2 documentation.

jConnect for JDBC, Adaptive Server Enterprise ODBC Driver, Adaptive Server Enterprise OLE DB Provider, and Adaptive Server Enterprise ADO.NET Data Provider support this feature for normal DML operations (insert, delete, update, and merge) when connected to an Adaptive Server with the feature enabled. You do not need any special configuration to use the feature; it is automatically supported.

jConnect for JDBC and Adaptive Server Enterprise ODBC Driver also support this feature for bulk copy when connected to an Adaptive Server with the feature enabled.

This feature is not available for nonmaterialized or virtual computed columns, encrypted columns, and XML Columns.

## Fast logged bulk insert

Adaptive Server version 15.7 ESD #2 allows you to fully log bcp in fast mode, providing full data recovery. Previous versions of bcp in fast mode logged only page allocations. See Adaptive Server Enterprise 15.7 ESD #2 documentation.

In jConnect for JDBC, set ENABLE_BULK_LOAD connection property to the new value LOG_BCP to enable full logging.

In ODBC Driver, set EnableBulkLoad connection property to new value 3 to enable full logging. Alternatively, set the SQL_ATTR_ENABLE_BULK_LOAD connection attribute to the desired level in the ODBC application:

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

This allows a single connection to use different types of bulk load.

In ADO.NET Provider, set EnableBulkLoad connection property to new value 3 to enable full logging.

## Dynamic logging

Starting with ESD #4, jConnect for JDBC supports logging mechanism by implementing standard Java Logger mechanism. Now, the application can get handle of jConnect's logger and turn logging on or off as and when required. See *jConnect for JDBC Programmers Reference*.

## Dynamic client information setting

Starting with ESD #4, you can set new values for jConnect for JDBC client information properties (ApplicationName, ClientUser, ClientHostName) using setClientInfo() and getClientInfo() standard methods even after the connection has been established:

## Dynamic connection property setting

Starting with ESD #4, you can set new values for jConnect for JDBC connection properties using setClientInfo() and getClientInfo() standard methods even after the connection has been established. See *jConnect for JDBC Programmers Reference* for the list of connection properties that can be dynamically set.

## Exception handling

Exception handling in jConnect for JDBC has been enhanced. You can use getCause() method to get the cause of the exception when the exception message contains directive to use getcause().

## New jConnect connection properties for performance improvement

Starting with ESD #4, jConnect for JDBC has the following new connection properties for performance improvement:

| Property | Description | Default value |
|---|---|---|
| OPTIMIZE_STRING_ CONVERSIONS | Specifies whether or not to enable string conversion optimization.<br>This optimization behavior can improve jConnect performance when a client uses character datatypes in SQL prepared statement.<br>Values:<br>• 0 – the default value; string conversion optimization is not enabled.<br>• 1 – enable string conversion optimization when jConnect uses utf8 or server default character set.<br>• 2 – enable string conversion optimization for all cases. | 0 |
| SUPPRESS_PARAM_ FORMAT | When executing dynamic SQL prepared statements, jConnect client can use the SUPPRESS_PARAM_FORMAT connection string property to suppress parameter data (TDS_PARAMS). The client sends less parameter metadata where possible for better performance.<br>Values:<br>• 0 – TDS_PARAMFMT is not suppressed in select, insert, and update operations.<br>• 1 – the default value; TDS_PARAMFMT is suppressed where possible. | 1 |
| SUPPRESS_ROW_ FORMAT | In jConnect, client can use the SUPPRESS_ROW_FORMAT connection string property to force Adaptive Server to send TDS_ROWFMT or TDS_ROWFMT2 data only when the row format changes for a dynamic SQL prepared statement. Adaptive Server can send less data to the client if possible, resulting in better performance.<br>Values:<br>• 0 – TDS_ROWFMT or TDS_ROWFMT2 data is sent, even if the row format has not changed.<br>• 1 – the default; forces the server to send TDS_ROWFMT or TDS_ROWFMT2 only when the row format has changed. | 1 |

## New jConnect connection properties

Starting with ESD #4, jConnect for JDBC has the following new connection properties:

| Property | Description | Default value |
|---|---|---|
| EARLY_BATCH_ READ_THRESHOLD | Specifies the threshold on number of rows after which a reader thread should be started to drain out the server responses for a batch.<br>Set this value to -1 if the early read is not ever required. | -1 |
| STRIP_BLANKS | Forces the server to remove the preceding and trailing blanks in a string value before storing it in the table.<br>Values:<br>• 0 – the default value; string values sent by the client are stored ‘as is’.<br>• 1 – preceding and trailing blanks in a string value are removed before storing it in the table. | 0 |
| SUPPRESS_ CONTROL_TOKEN | Suppresses control tokens.<br>Values:<br>• 0 – the default value; control tokens are sent.<br>• 1 – control tokens are suppressed. | 0 |

## Notes on Hibernate support for JDBC

Hibernate is a collection of related projects enabling developers to utilize POJO-style domain models in their applications extending beyond Object or Relational Mapping. Out of the many modules, Hibernate-core module deals with Object Relational Mapping.

Dialect is a helper for Hibernate to communicate with the database in its language. Hibernate has created dialect files for versions of Adaptive Server Enterprise:

| Sybase Dialect file | ASE version |
|---|---|
| *Sybase11Dialect.java* | 11.9.2 |
| *Sybase15Dialect.java* | 15.0 |
| *Sybase157Dialect.java* | 15.7 |

**Note** Hibernate and Sybase actively test latest releases and create new dialects when required. All the updated dialects are part of scheduled Hibernate releases. This release schedule may not match Adaptive Server release schedule. If you need access to the updated dialect prior to release of the corresponding Hibernate release, they may be available at Hibernate on Sybase ASE at https://community.jboss.org/wiki/HibernateSybaseintegration.

## Support for SQL_ATTR_OUTPUT_NTS=SQL_FALSE

Adaptive Server Enterprise ODBC Driver now allows you to set the SQL_ATTR_OUTPUT_NTS attribute to SQL_FALSE so that the driver does not return string data null-terminated. Set the attribute before allocating any connection handle:

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

By default, the SQL_ATTR_OUTPUT_NTS attribute to SQL_TRUE and all output strings are null-terminated.

## Support for SQLLEN datatype of length 8-byte (Linux 64-bit only)

Adaptive Server Enterprise ODBC Driver for Linux x86-64 64-bit and Linux on POWER 64-bit now supports a 4-bytes SQLLEN datatype and an 8-bytes SQLLEN datatype.

Red Hat and SUSE provide the unixODBC Driver Manager as their driver manager. Versions of the unixODBC Driver Manager prior to 2.2.13 expect to use a 4-bytes SQLLEN datatype. The default configuration of the unixODBC Driver Manager in versions 2.2.13 and later, such as that provided by Red Hat Enterprise Linux 6 and later, expect an 8-bytes SQLLEN datatype. Accordingly, the Adaptive Server Enterprise ODBC Driver provides two versions of the driver. Please check the unixODBC Driver Manager version used by your 64-bit Linux system.

Starting with ESD #4, there are two driver shared library files and a soft link in the *DataAccess64/ODBC/lib/* directory:

- The *libsybdrvodb-sqllen4.so* - equivalent to the original *libsybdrvodb.so* file that supports a 4-bytes SQLLEN datatype
- The *libsybdrvodb-sqllen8.so* file - new version of the *libsybdrvodb.so* file that supports an 8-bytes SQLLEN datatype
- The *libsybdrvodb.so* soft link that points to the original driver shared library file, now named *libsybdrvodb-sqllen4.so*

There is no change when you want to continue using the 4-bytes SQLLEN datatype.

To use the 8-bytes SQLLEN datatype, modify the soft link to point to the *libsybdrvodb-sqllen8.so* file:

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

## ODBC deferred array binding

Adaptive Server Enterprise ODBC Driver now provides the extended SQLBindColumnDA() and SQLBindParameterDA() APIs that allow applications to bind all columns or parameters with a single API call. When you use these APIs, the pointers to column buffer or parameter buffer are reevaluated for each SQLExecute() or SQLExecDirect() call. Therefore, the application is able to change the buffers without another SQLBindCol() or SQLBindParameter() call. Because the calls to bind new pointers can be expensive, using the new extended APIs improves application performance where the same statement needs to be executed many times. Applications may also be able to save some memory copy operations by changing the buffer pointers before executing a query such that data is read from where available or copied to where needed.

See *Adaptive Server Enterprise ODBC Driver by Sybase Users Guide*.

## Bulk insert support for ODBC data batching

The ODBC data batching without binding parameter arrays feature introduced in 15.7 release has now been extended to support inserting batches using bulk insert protocol. To enable, set the EnableBulkLoad connection property to the desired bulk insert level (1, 2, or 3), and the HomogeneousBatch connection property to 2. See *Adaptive Server Enterprise ODBC Driver by Sybase Users Guide*.

For example, add `;enablebulkload=3;homogeneousbatch=2` in the connection string and simple insert statements executed in a batch are converted to fast-logged bulk insert statements.

Alternatively, set the connection properties programmatically using the SQL_ATTR_HOMOGENEOUS_BATCH and SQL_ATTR_ENABLE_BULK_LOAD connection attributes to achieve the same result:

```
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);
```

## Dynamic logging support without ODBC driver manager tracing

Adaptive Server Enterprise ODBC Driver 15.7 introduced the application logging without an ODBC driver manager tracing feature. See "Logging without ODBC Driver Manager tracing" on page 124. The application logging can be enabled (or disabled) for the duration of application execution.

ESD #4 extends this support by allowing you to dynamically enable or disable the application logging during application execution by setting the new SQL_OPT_TRACE environment attribute. Valid values are 0 (default) to disable or 1 to enable.

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

- Dynamic logging is enabled and disabled globally and affects all connections regardless of when they were opened and whether they are part of the environment handle used to set SQL_OPT_TRACE.
- By default, the log is written to the *sybodbc.log* file in the current directory. Use the SQL_OPT_TRACEFILE environment attribute to set a different file or file path.

  ```
  SQLSetEnvAttr(0, SQL_OPT_TRACEFILE, (SQLPOINTER)
  "logfilepath", SQL_NTS);
  ```

- Setting the LOGCONFIGFILE environment variable or registry value enables logging for the entire duration of application execution and overrides SQL_OPT_TRACE.
- If an ODBC Driver Manager is being used, setting SQL_OPT_TRACE turns on the Driver Manager tracing and has no impact on driver tracing.
- The client application can use a null handle when linking directly against the driver or an allocated handle when using Driver Manager tracing.
- *log4cplus* configuration file cannot be used with SQL_OPT_TRACE.

## Dynamic control of TDS protocol capture

Use the new SQL_ATTR_TDS_CAPTURE connection attribute of Adaptive Server Enterprise ODBC Driver to allow pause (SQL_CAPTURE_PAUSE) and resume (SQL_CAPTURE_RESUME) of TDS protocol capture.

```
// 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);
```

By default, TDS protocol capture operates for the duration of the connection when the ProtocolCapture connection property is set for the connection. Using SQL_ATTR_TDS_CAPTURE (with the ProtocolCapture connection property set) allows the application to selectively pause and resume TDS protocol capture for desired segments of program execution.

SQL_ATTR_TDS_CAPTURE can be set after a connection handle is allocated. It is not an error to pause or resume TDS protocol capture before a connection is established or for a connection that is not using TDS protocol capture. Pausing or resuming TDS protocol capture may be delayed by the driver to ensure the integrity of the capture stream. This ensures the write of full PDU packets for accurate capture consumption by Ribo and other protocol translator utilities.

Do not set SQL_ATTR_TDS_CAPTURE for applications that need to capture all TDS packets for a connection.

## Replication Server connection support

Adaptive Server Enterprise ODBC Driver can connect to Replication Server® to monitor and administer the server. Only valid Replication Server Administration commands sent by the ODBC Driver are supported by Replication Server. Set the BackEndType connection property to `Replication Server` for Replication Server connections.

## Comprehensive ADO.NET provider assembly files

Starting with ESD #4, Adaptive Server Enterprise ADO.NET Data Provider has only two provider assembly files that each contain all functionality:

- *Sybase.AdoNet2.AseClient.dll* – supports features of .NET 2.0, .NET 3.0, and .NET 3.5.
- *Sybase.AdoNet4.AseClient.dll* – supports features of .NET 4.1, and later.

The 32-bit versions of these files are installed in the *C:\Sybase\DataAccess\ADONET\dll* directory and the 64-bit versions are installed in the *C:\Sybase\DataAccess64\ADONET\dll* directory.

Update any build or deployment scripts that reference any of the DLLs that have been obsoleted.

## ADO.NET support for larger decimal precision/scale

Adaptive Server numeric and decimal datatypes support a maximum precision/scale of 38 and results from arithmetic operations can support precision/scale of up to 78, whereas the .NET Framework Decimal datatype can support a maximum precision/scale of 28. This can lead to data overflow when reading data of Adaptive Server numeric and decimal type or result of an arithmetic operation into the .NET Framework Decimal type.

Adaptive Server Enterprise ADO.NET Data Provider now supports AseDecimal - a structure that can support a precision/scale of 78. To use the AseDecimal structure to retrieve numeric or decimal values, set the new UseAseDecimal connection property to 1. By default, UseAseDecimal is set to 0 and the AseDecimal structure is not used.

## Visual Studio DDEX Connection dialog enhancement for additional connection properties (Doc CR 705592)

Adaptive Server Enterprise ADO.NET Data Provider now allows you to add additional connection properties in the Visual Studio DDEX Add Connection dialog.

- Connection properties can be specified as a semicolon(;)-separated list.
- Last connection property need not terminate with a semicolon(;).
- Properties without a value are ignored.

Currently, there are no warning or error messages to flag incorrect connection specifications.

## New connection strings for OLE DB applications

| Property names | Description | Required | Default value |
|---|---|---|---|
| ProtocolCapture | Enable this property to capture communication between an OLE DB application and the server.<br>See *Adaptive Server Enterprise OLE DB Provider Users Guide*. | No | Empty |

| Property names | Description | Required | Default value |
|---|---|---|---|
| RetryCount, RetryDelay | Control the connection retry behavior.<br>RetryCount is the number of times to attempt to connect to the server before reporting the connection failed. Between each retry, the driver delays for RetryDelay number of seconds.<br>By default, the OLE DB application does not retry the connection.<br>You can also specify these values in *SQL.INI* and LDAP interfaces:<br>• RetryCount can be specified as Retry Count in *SQL.INI* and as sybaseRetryCount in LDAP.<br>• RetryDelay can be specified as Loop Delay in *SQL.INI* and as sybaseRetryDelay in LDAP. | No | 0 |
| SuppressControlTokens | Specifies that Adaptive Server should not send TDS_CONTROL tokens.<br>Values:<br>• 0 – forces Adaptive Server to send TDS_CONTROL tokens where possible.<br>• 1– the default value; forces Adaptive Server to suppress TDS_CONTROL tokens. | No | 1 |
| SuppressParamFormat | Specifies that the OLE DB application should send parameter format tokens only when the format changes.<br>Values:<br>• 0 – forces the OLE DB application to always send the parameter format tokens on every execution.<br>• 1– the default value; requests the OLE DB application to suppress sending parameter format tokens when the format has already been set. | No | 1 |
| SuppressRowFormat | Specifies that Adaptive Server should send row format tokens only on first execution or when the format changes.<br>Values:<br>• 0 – forces Adaptive Server to send the format information on every execution.<br>• 1– the default value; requests Adaptive Server to suppress sending row format tokens when possible. | No | 1 |

| Property names | Description | Required | Default value |
|---|---|---|---|
| SuppressRowFormat2 | Specifies that Adaptive Server should send data using the TDS_ROWFMT byte sequence where possible instead of the TDS_ROWFMT2 byte sequence.<br>Values:<br>• 0 – the default value; forces Adaptive Server to send data in TDS_ROWFMT2 where possible.<br>• 1– forces Adaptive Server to send data in TDS_ROWFMT where possible.<br>See *Adaptive Server Enterprise OLE DB Provider Users Guide*. | No | 0 |

## Adaptive Server Enterprise extension module for Python in ESD #4

The Adaptive Server Enterprise extension module for Python has been enhanced to support new parameter datatype for dynamic statements and stored procedures.

### New parameter datatype support for dynamic statements and stored procedures

Starting with ESD #4, the Adaptive Server Enterprise extension module for Python supports decimal datatypes, money datatypes, and LOB as parameters for dynamic statements and stored procedures.

The Adaptive Server Enterprise extension module for Python also supports for date, time, datetime, and float parameters for stored procedures.

See the *Adaptive Server Enterprise Extension Module for Python Programmers Guide*.

## Adaptive Server Enterprise extension module for PHP in ESD #4

Starting with ESD #4, the Adaptive Server Enterprise extension module for PHP has the full set of APIs for application development:

| API type | API | Description |
|---|---|---|
| Connections: | sybase_close() | Closes the specified connection to ASE. |
| | sybase_connect() | Opens a connection to ASE. |
| | sybase_pconnect() | (**New**) Opens a persistent connection to ASE. |

| API type | API | Description |
|---|---|---|
| Queries: | sybase_affected_rows() | (**New**) Returns the number of rows affected by the last insert, delete, or update query on the specified connection. |
| | sybase_query() | Sends a query to the specified connection. The complete result set is automatically fetched and buffered. |
| | sybase_unbuffered_query() | (**New**) Sends a query to the specified connection. The complete result set is not automatically fetched and buffered as with sybase_query(). |
| Remote Procedure Calls: | sybase_rpc_bind_param_ex | (**New**) Binds a PHP variable to a remote procedure parameter. |
| | sybase_rpc_execute | (**New**) Executes the remote procedure call that was initialized with sybase_rpc_init(). |
| | sybase_rpc_init | (**New**) Returns a statement identifier pointing to the statement initialized for the remote procedure on the connection. |
| Result sets: | sybase_data_seek() | (**New**) Moves the internal row pointer on the result set associated with the result identifier to point to the specified row number. |
| | sybase_fetch_array() | (**New**) Fetch a result row as an associative array, a numeric array, or both. |
| | sybase_fetch_assoc() | Fetches one row of data from the result set associated with the specified result identifier in an associative array. |
| | sybase_fetch_field() | (**New**) Returns an object containing field information. |
| | sybase_fetch_object() | (**New**) Fetches one row of data from the result set associated with the specified result identifier as an object. |
| | sybase_fetch_row() | (**New**) Fetches one row of data from the result set associated with the specified result identifier in a numeric array. |
| | sybase_field_seek() | (**New**) Sets the internal pointer to the field offset requested. |
| | sybase_free_result() | Frees all memory associated with the result set. |
| | sybase_next_result() | (**New**) Returns a result set identifier pointing to the next result set on the connection. |
| | sybase_num_fields() | (**New**) Returns the number of fields in the result set. |
| | sybase_num_rows() | (**New**) Returns the number of rows in the result set of a select statement. |
| | sybase_use_result | (**New**) Stores the result set of the last unbuffered query on the connection and returns a result set identifier pointing to this stored result set. |

| API type | API | Description |
|---|---|---|
| Miscellaneous: | sybase_get_last_message() | (**New**) Returns the last message returned by the server. |
| | sybase_get_last_status | (**New**) Returns the last status result that was sent on the connection. |
| | sybase_select_db() | (**New**) Sets the current active database on the server referred to by the connection resource. |
| | sybase_set_message_handler() | (**New**) Sets a user-defined callback function that is to be called when a client or server message is received. |

See the *Adaptive Server Enterprise Extension Module for PHP Programmers Guide*.

## Adaptive Server Enterprise database driver for Perl in ESD #4

The Adaptive Server Enterprise database driver for Perl in ESD #4 has the following feature enhancements. See the *Adaptive Server Enterprise database driver for Perl Programmers Guide*.

- New database handle attributes
- New default date conversion and display format support using the new _data_fmt private method
- New LONG/BLOB data handling support

  Adaptive Server Enterprise database driver for Perl now supports an image and a text type for LONG/BLOB data. Each type can hold up to 2GB of binary data.
- New automatic key generation support

  Adaptive Server Enterprise database driver for Perl now supports an IDENTITY feature for automatic key generation. Declaring a table with an IDENTITY column generates a new value for each insert. The values are monotonically increasing, but are not guaranteed to be sequential. To fetch the value generated and used by the last insert:

  ```
  SELECT @@IDENTITY
  ```
- New parameter binding support

  Adaptive Server Enterprise database driver for Perl now directly supports parameter binding. Only the '?' style parameters are supported; the ":1" placeholder type parameters are not supported. Binding a text or image datatype parameter is not supported.

- New stored procedures with input and output parameters support

## New features for ESD #3

ESD #3 introduces new functionality for Open Client 15.7 and Open Server 15.7 and for Adaptive Server Enterprise extension module for Python 15.7.

### Skipping installation of samples, documentation, and debug files

Starting with ESD#3, you can choose to skip installation of sample files, documentation files, and debug files. By default, these files are installed when you install Open Server and SDK. To skip installation of these files:

- Use the new -DPRODUCTION_INSTALL=TRUE installer command-line argument when installing in GUI, console, and silent mode.
- Use the new PRODUCTION_INSTALL=TRUE property in the response file when installing in silent mode.

### Open Client 15.7 and Open Server 15.7 features in ESD #3

New features in ESD #3 include the CyberSafe Kerberos driver on 64-bit Microsoft Windows, scripting language enhancements, UNIX named sockets, and logging rejected rows.

#### CyberSafe Kerberos driver on 64-bit Microsoft Windows

Open Client and Open Server include *libsybskrb64.dll*, which is a 64-bit CyberSafe Trustbroker Kerberos driver library for Microsoft Windows x86-64 64-bit. *libsybskrb64.dll* is located in *%SYBASE%\%SYBASE_OCS%\dll*; its behavior is similar to the 32-bit CyberSafe TrustBroker Kerberos driver library *libsybskrb.dll*.

#### UNIX named sockets

This feature provides support for UNIX named sockets in Open Client and Open Server. This type of socket is also referred to as a UNIX domain socket.

This feature allows the use of UNIX named sockets for faster intrahost communication since the TCP stack does not need to be traversed for interprocess communication. To enable this feature, add entries to the directory service layer, specifying *afunix* instead of *tcp* for the transportation type.

For example, a traditional interfaces file entry may look as follows:

*MYSERVER*

```
master tcp unused myhost 8600
query tcp unused myhost 8600
```

To use UNIX named sockets instead of TCP for local clients while still using TCP for remote, the above entries become:

*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
```

## Logging rows rejected by the client

A new bcp option named --clienterr errorfile has been added to log any rejected row and its associated error message into an error file, if the row was rejected by the client due to errors detected by the client, such as conversion or format errors.

If you use the --clienterr option without the -e option, client error messages are written into the error file. However, server error messages are not written into the error file.

If you use the --clienterr option with the -e option, bcp does not proceed with the copy in or copy out operation.

## Increased bcp maximum rows handling capacity

The maximum number of rows that bcp can handle has been increased from *INT32_MAX* to *UINT64_MAX* (which is 18446744073709551615).

## Parameter format suppression

Open Client now support parameter format suppression for dynamic statements in Adaptive Server Enterprise.

## Adaptive Server Enterprise extension module for Python in ESD #3

The Adaptive Server Enterprise extension module for Python has been enhanced to support stored procedures with input and output parameters, compute rows, and localized error messages.

### Accessing stored procedures using Python

The Adaptive Server Enterprise extension module for Python adds support for passing input and output parameters to stored procedures. Use the callproc() method of the Cursor object to call a stored procedure. If there is an error in executing the stored procedure, callproc() throws an exception and you can retrieve the status value using the proc_status attribute. This support is an extension to the Python DBAPI specification.

This is a sample Python application with multiple row results:

```
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)
```

To specify output parameters, the extension module provides the OutParam constructor. This support is an extension to the Python DBAPI specification. The callproc() method returns a list of all the parameters passed to the method. If there are output parameters, and no result sets generated from the store procedure, the list contains the modified output values as soon as callproc() completes. However, if there are result sets, the list does not contain modified output values until all result sets from the stored procedure have been retrieved using the fetch*() methods and a call to nextset() is made to check if there are any more result sets. The nextset() method must be called even if only one result set is expected.

This is a sample Python application with output parameters:

```
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()
```

More examples of different output parameter types are available in the sample program *callproc.py*.

## Compute rows using Python

The Adaptive Server Enterprise extension module for Python adds support for compute rows. An example of compute row processing is available in the sample program *compute.py*.

## Localized error messages

The Adaptive Server Enterprise extension module for Python now supports localization of error messages.

# New features for ESD #1

ESD #1 introduces new functionality for Open Client 15.7 and Open Server 15.7, SDK 15.7, and Adaptive Server Enterprise extension module for Python 15.7.

## Open Client 15.7 and Open Server 15.7 features in ESD #1

New features in ESD #1 include the FIPS-certified SSL filter and support for the Adaptive Server Enterprise database driver for Perl and the Adaptive Server Enterprise extension module for PHP on 64-bit Windows.

### FIPS-certified SSL filter

The Sybase SSL filter is now Federal Information Processing Standard (FIPS) 140-2 compliant for these platforms supporting Certicom SSL:

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

The shared object SSL filter files for Linux on POWER 32-bit and 64-bit have been renamed from *libsybfcsissl.so* to *libsybfssl.so* and from *libsybfcsissl64.so* to *libsybfssl64.so*. The sample *libtcl.cfg* file has also been updated:

```
[FILTERS]
;ssl=libsybfssl.so
```

The SSL filter DLL for Microsoft Windows x86-64 64-bit has been renamed from *libsybfcsissl64.dll* to *libsybfssl64.dll*. The sample *libtcl64.cfg* file has also been updated:

```
[FILTERS]
;ssl=libsybfssl64
```

### ASE database driver for Perl and ASE extension module for PHP supported on 64-bit Windows

The Adaptive Server Enterprise database driver for Perl is now supported on the Microsoft Windows 64-bit platform for use with ActivePerl 5.14.1 and DBI 1.616.

The Adaptive Server Enterprise extension module for PHP is now supported on the Microsoft Windows 64-bit platform for use with PHP version 5.3.6.

## SDK 15.7 features for jConnect and Adaptive Server Drivers and Providers in ESD #1

ESD #1 introduces support for suppressing parameter format metadata and row format metadata to improve performance.

### Suppressing parameter format metadata to improve prepared statement performance

You can improve the performance of prepared statements with the ODBC driver by suppressing parameter format metadata when the prepared statements are reexecuted. Adaptive Server 15.7 ESD#1 and later supports parameter format metadata suppression.

To suppress parameter format metadata, set the DynamicPrepare connection property to 1, and then use the SuppressParamFormat connection string property.

The valid SuppressParamFormat connection string property values are:

- 0 – parameter format metadata is not suppressed in prepared statements.

- 1– the default value; parameter format metadata is suppressed where possible.

---

**Note** You can suppress parameter format metadata in prepared statements only if the connected Adaptive Server supports this feature. If the DynamicPrepare and SuppressParamFormat parameters are both set to 1 but the connected Adaptive Server does not support the suppression of parameter format metadata, Adaptive Server ignores the parameter settings.

---

Example

This ODBC connection string suppresses parameter format metadata in prepared statements:

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;
SuppressParamFormat=1;
```

## Suppressing row format metadata to improve query performance

You can improve the performance of repeatedly executed queries with the ODBC driver and ADO.NET Data Provider by instructing Adaptive Server to suppress row format metadata (TDS_ROWFMT or TDS_ROWFMT2) for queries that are reexecuted in a session. Adaptive Server 15.7 ESD#1 and later supports row format metadata suppression.

To suppress row format metadata, use the SuppressRowFormat connection string property.

The valid SuppressRowFormat connection string property values are:

- 0 – row format metadata is not suppressed.
- 1 – the default value; Adaptive Server does not send row format metadata where possible.

---

**Note** You can suppress row format metadata only if the connected Adaptive Server supports this feature. If the SuppressRowFormat parameter is set to 1 but the connected Adaptive Server does not support the suppression of row format metadata, Adaptive Server ignores the parameter setting.

---

Example

This ODBC connection string suppresses row format metadata:

```
DSN=sampledsn;UID=user;PWD=password;;DynamicPrepare=1;
SuppressRowFormat=1;
```

### SuppressRowFormat2 and SQLBulkOperations

Do not use the SuppressRowFormat2 connection string property with an ODBC program that uses the SQLBulkOperations API. Enabling SuppressRowFormat2 suppresses information that SQLBulkOperations requires and results in an error.

## Adaptive Server Enterprise extension module for Python in ESD #1

As of ESD #1, the Adaptive Server Enterprise extension module for Python supports Python versions 2.6, 2.7, and 3.1.

You can install the Adaptive Server Enterprise extension module for Python from the SDK installer. For installation instructions, see the *Software Developers Kit and Open Server Installation Guide* and the *Software Developers Kit and Open Server Release Bulletin*. For information about using the Adaptive Server Enterprise extension module for Python, see the *Adaptive Server Enterprise Extension Module for Python Programmers Guide*.

### Configuring Adaptive Server Enterprise extension module for Python

#### Python module search path

Python searches for an imported module in the list of directories specified with the Python variable *sys.path*. This variable is initialized from the directory containing the application, and in the list of directories specified by the environment variable PYTHONPATH, which uses the same syntax as the shell variable PATH, that is, a list of directory names. If you have not set PYTHONPATH, or if the module file is not found, the search continues in an installation-dependent default path. To use the Adaptive Server Enterprise extension module for Python in an application, you must set either PYTHONPATH, or the Python variable *sys.path* to one of the following directory paths (these are the default directories where the different versions of the Adaptive Server Python extension module are installed):

| Platform | Python Version | Default Installation Path |
|---|---|---|
| Windows | 2.6 | *$SYBASE\$SYBASE_OCS\python\python26_64\dll* |
| | 2.7 | *$SYBASE\$SYBASE_OCS\python\python27_64\dll* |
| | 3.1 | *$SYBASE\$SYBASE_OCS\python\python31_64\dll* |
| All other platforms | 2.6, 2.7 | *$SYBASE/$SYBASE_OCS/python/python26_64r/lib* |
| | 3.1 | *$SYBASE/$SYBASE_OCS/python/python31_64r/lib* |

# Open Client 15.7 and Open Server 15.7 features

This section describes the new features introduced in Open Client 15.7 and Open Server 15.7.

## Large object locator support

Open Client and Open Server version 15.7 support large object (LOB) locators. A LOB locator contains a logical pointer to LOB data in Adaptive Server rather than the data itself, thereby reducing the amount of data that passes through the network between Adaptive Server and its clients.

Adaptive Server 15.7 includes Transact-SQL® commands and functions that operate on LOB data using LOB locators. You can invoke these commands and functions as language commands from the Client-Library. See Chapter 21, "In-Row Off-Row LOB" in the *Adaptive Server Enterprise Transact-SQL Users Guide*.

### Client-Library changes

The CS_LOCATOR datatype supports LOB locator. The cs_locator_alloc() and cs_locator_drop() APIs allocate and deallocate memory for CS_LOCATOR variables. cs_locator() has been added to retrieve information from a CS_LOCATOR variable.

Client-Library routines cs_convert() and ct_bind() have been enhanced to handle CS_LOCATOR variables.

#### CS_LOCATOR

CS_LOCATOR is an opaque datatype that stores locator values and optional prefetched data. Use cs_locator_alloc() to allocate memory for a CS_LOCATOR variable before binding the incoming locator to the variable, otherwise, an error occurs. When the variable is no longer needed, use cs_locator_drop() to free its memory.

CS_LOCATOR variables can be reused, however, the current locator value in Adaptive Server is valid only until the transaction ends.

The type constants for CS_LOCATOR are:

- CS_TEXTLOCATOR_TYPE – for text LOBs.
- CS_IMAGELOCATOR_TYPE – for image LOBs.

- CS_UNITEXTLOCATOR_TYPE – for unitext LOBs.

Use cs_convert() to retrieve the locator's prefetched data and the character representation of the locator value from the CS_LOCATOR variable. Converting CS_LOCATOR to a CS_CHAR returns the locator's hexadecimal value as a string. Converting the locator to CS_TEXT_TYPE, CS_IMAGE_TYPE, or CS_UNITEXT_TYPE returns the locator's prefetched data.

***Table 8: Supported LOB locator conversions***

| | **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 |

LEGEND: X = supported conversion.

When working with locator datatypes:

- ct_bind() ignores the *maxlength* value of CS_DATAFMT because Client-Library considers the length of locator datatypes as fixed. Memory required for any optional prefetched data that is sent with the locator is allocated internally for its entire length. The *maxlength* value does not influence the length of the prefetched data.
- You can bind an incoming LOB locator to CS_CHAR_TYPE. You cannot, however, directly bind a locator to CS_TEXT_TYPE, CS_IMAGE_TYPE, or CS_UNITEXT_TYPE.

## cs_locator()

Retrieves information from a CS_LOCATOR variable, such as prefetched data, the total length of the LOB in the server, or the character representation of the locator pointer.

Syntax

```
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;
```

Parameters

- *ctx* – a pointer to a CS_CONTEXT structure.
- *action* – specifies whether to set or retrieve information. Currently, the only action allowed is CS_GET.
- *locator* – a pointer to the locator variable.
- *type* – type of information to retrieve or set. Symbolic values:

| Value | Action | *buffer points to | Description |
|---|---|---|---|
| CS_LCTR_LOBLEN | CS_GET | CS_BIGINT | Retrieves the total length of the LOB data in the server. |
| CS_LCTR_LOCATOR | CS_GET | CS_CHAR | Retrieves the locator value as a character string. |
| CS_LCTR_PREFETCHLEN | CS_GET | CS_INT | Retrieves the length of the prefetched LOB data contained in the locator variable. |
| CS_LCTR_PREFETCHDATA | CS_GET | CS_CHAR | Retrieves the prefetched LOB data contained in the locator variable. |
| CS_LCTR_DATATYPE | CS_GET | CS_INT | Retrieves the locator type. Valid return types are CS_TEXTLOCATOR_TYPE, CS_IMAGELOCATOR_TYPE, and CS_UNITEXTLOCATOR_TYPE. |

- *buffer* – a pointer to the variable to store data to. Character data is NULL terminated.
- *buflen* – **buffer* length, in bytes.
- *outlen* – a pointer to a CS_INT variable. If *outlen* is not NULL, cs_locator() sets **outlen* to the length, in bytes, of the data placed in **buffer*. If the data returned is a character data (for example, a prefetched data or locator string), the length returned in **outlen* includes the NULL terminator. If cs_locator() returns CS_TRUNCATED and *outlen* is not NULL, then cs_locator() returns the required buffer size in **outlen*.

Returns

| Return Value | Meaning |
|---|---|
| CS_SUCCEED | The routine completed successfully. |
| CS_TRUNCATED | The result has been truncated because the buffer is too small. |
| CS_FAIL | The routine failed. |

### cs_locator_alloc()

Allocates a CS_LOCATOR datatype structure.

Syntax

```
CS_RETCODE cs_locator_alloc(ctx, locator)

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

Parameters

- *ctx* – a pointer to a CS_CONTEXT structure.
- *locator* – the address of a locator variable to be allocated. Sets **locator* to the address of a newly allocated CS_LOCATOR structure.

Returns

| Return Value | Meaning |
|---|---|
| CS_SUCCEED | The routine completed successfully. |
| CS_FAIL | The routine failed. |

### cs_locator_drop()

Deallocates a CS_LOCATOR datatype structure.

Syntax

```
CS_RETCODE cs_locator_drop(ctx, locator)

CS_CONTEXT *ctx;
CS_LOCATOR *locator;
```

Parameters

- *ctx* – a pointer to a CS_CONTEXT structure.
- *locator* – a pointer to the locator variable to be deallocated.

Returns

| Return Value | Meaning |
|---|---|
| CS_SUCCEED | The routine completed successfully. |
| CS_FAIL | The routine failed. |

### *isql* enhancement

isql displays the LOB locator value in its hexadecimal character form. Prefetched data stored in CS_LOCATOR does not appear.

**Example** Converts LOB data to locators, and displays the locator value:

```
1> set send_locator on
2> go

1> select * from testable
2> go
charcol         textcol
--------------- --------------------------------------------------
```

```
Hello           0x48656c6c6f20576f726c642e2048657265204920616d2e2e
```

## Open Server support for large object locators

LOB locator functionality has been added to Server-Library, allowing Open Server applications to pass LOB locator language commands from the client to back-end servers. To pass LOB locators from servers to client applications, an Open Server application allocates memory for a *CS_LOCATOR* variable, and binds and receives the LOB information from the server.

srv_bind() and srv_descfmt() have been enhanced to handle CS_TEXT_LOCATOR_TYPE, CS_IMAGE_LOCATOR_TYPE, and CS_UNITEXT_LOCATOR_TYPE.

## Large object locator support

These connection capabilities indicate support for sending and receiving LOB locators:

- CS_DATA_LOBLOCATOR – a read-only request capability that is implicitly set when client applications are initialized with CS_VERSION_157, indicating that the Client-Library can send LOB locators to the server.
- CS_DATA_NOLOBLOCATOR – a response capability that a client application sets to inform servers not to send LOB locators even though the underlying Client-Library supports them.

### Requesting LOB locators from the server

By default, when selecting LOB columns or values, Adaptive Server sends LOB data instead of LOB locators, regardless of the negotiated LOB locator support. To explicitly request LOB locators or to request prefetched data, set these query-processing options using ct_options():

- CS_OPT_LOBLOCATOR – a Boolean that, when set to CS_TRUE, requests the server to return a locator instead of a LOB value. Set this option before sending the query to the server. The default is CS_FALSE.
- CS_OPT_LOBPREFETCHSIZE – an integer that specifies the size of the prefetched data that the server must send. For image locators, this size indicates the number of prefetched data bytes; for text and unitext locators, the number of characters.

CS_OPT_LOBPREFETCHSIZE has a default value of 0, which informs the server not to send prefetched data. A value of -1 retrieves the entire LOB data for the requested LOB along with its locator.

Locator values and optional prefetched data are stored in the CS_LOCATOR datatype. Clients must allocate memory for CS_LOCATOR variables before requesting for locator data.

**Example** Retrieves the LOB locator for a text value that needs to be truncated. See the *Open Client Client-Library/C Reference Manual* for more code examples.

```
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);
```

## In-row and off-row LOB support

Bulk-Library version 15.7 supports in-row storage of text, image, and unitext large object (LOB) columns in Adaptive Server.

In Adaptive Server 15.7, LOB columns that are marked for in-row storage are stored in-row when there is enough space available in the row. Only bound LOB data can be written in-row. The bcp utility binds LOB data, thus sending in-row LOB data as applicable. See Chapter 21, "In-Row Off-Row LOB" in the *Adaptive Server Enterprise Transact-SQL Users Guide*.

## Bulk-Library *select into* logging

To process a select into existing table statement that inserts rows into a proxy table, Adaptive Server uses the Bulk-Library to generate a bulk-copy operation. However, full logging is not available for regular bulk-copy operations. The BLK_CUSTOM_CLAUSE property enables Adaptive Server to distinguish between ordinary bulk-copy operations and bulk-copy operations that have resulted from an insert into statement affecting a proxy table. Bulk-copy operations that result from such an insert into statement can then be appended with the custom clause specified by the BLK_CUSTOM_CLAUSE property. Adaptive Server can detect this clause and perform full logging.

### BLK_CUSTOM_CLAUSE

An application can use the blk_props Bulk-Library routine to set or retrieve BLK_CUSTOM_CLAUSE:

***Table 9: Client/Server BLK_CUSTOM_CLAUSE property***

| Property name | Description | *buffer is | Applies to | Notes |
|---|---|---|---|---|
| BLK_CUSTOM_CLAUSE | A custom, application-specific SQL clause to add after the existing with clause of the insert bulk command. | A character string containing the custom clause. | IN copies only | Supported only by server versions that support the custom SQL clause. Currently used only by internal products. |

- A select into operation is allowed only if the Adaptive Server select into/bulkcopy/pllsort database option is set to on.
- For full logging of a select into operation, the Adaptive Server full logging for select into database option must be set to on.

Example

BLK_CUSTOM_CLAUSE is set with blk_props:

```
blk_props(blkdesc, CS_SET, BLK_CUSTOM_CLAUSE,
(CS_VOID *)"from select_into", CS_NULLTERM, NULL);
```

Adaptive Server generates a bulk copy operation with the specified custom clause appended:

```
insert bulk mydb.mytable with nodescribe from
select_into
```

where mydb and mytable are the affected database and table.

## Bulk-Library and *bcp* handling of nonmaterialized columns

Bulk-Library has been enhanced to handle nonmaterialized columns in Adaptive Server 15.7. With this enhancement, you can use Bulk-Library and bcp version 15.7 and later to bulk-copy-in data into Adaptive Server tables that are altered and contain nonmaterialized columns. Adaptive Server raises an error when you use earlier versions of bcp to bulk-copy-in data into nonmaterialized columns.

## Support for preserving trailing zeros

Open Client and Open Server version 15.7 support the disable varbinary truncation configuration parameter introduced in Adaptive Server 15.7. This parameter specifies whether Adaptive Server preserves or truncates trailing zeros from varbinary and binary null data.

Versions of Adaptive Server earlier than 15.7 and versions of bcp, and bulklib earlier than 15.7 truncate trailing zeros for varbinary datatypes. Versions of Adaptive Server 15.7 or later and versions of bcp, and bulklib 15.7 or later can truncate or preserve the trailing zeros of varbinary datatypes.

By default, disable varbinary truncation is 0 (off) for the server. Set it to 1 (on) to enable the feature.

## New DB-Library overflow errors

Use of a DB-Library routine that causes in an integer overflow results in this error:

```
302 = SYBEINTOVFL, "DB-LIBRARY internal error: The
arithmetic operation results in integer overflow."
```

Multiplication of the scrollopt and nrows parameters of the dbcursoropen DB-Library routine that causes an overflow results in this error:

```
301 = SYBCOPNOV, "dbcursoropen(): The multiplication of
scrollopt and nrows results in overflow."
```

## New nameless application configuration settings handling

You can now set whether the *ocs.cfg* runtime configuration file is parsed for application-specific settings for nameless applications (CS_APPNAME is not explicitly set by the application) and whether any settings found are applied to the application. The executable name obtained from the operating system is set as CS_APPNAME for the application and is used to parse the runtime configuration file.

Set CS_USE_DISCOVERED_APPNAME to CS_TRUE in the DEFAULT section of the *ocs.cfg* runtime configuration file to enable this feature.

When CS_USE_DISCOVERED_APPNAME is set to CS_FALSE (default), the runtime configuration file is not parsed for the nameless application.

Use CS_SANITIZE_DISC_APPNAME to specify whether the discovered application name (executable name obtained from the operating system) for a nameless application (CS_APPNAME is not explicitly set by the application) is used for parsing the runtime configuration file as is, after converting to uppercase, or after converting to lowercase.

You can set CS_SANITIZE_DISC_APPNAME in the DEFAULT section of the *ocs.cfg* runtime configuration file to any of these values:

- CS_CNVRT_UPPERCASE – convert discovered name to uppercase before use.
- CS_CNVRT_LOWERCASE – convert discovered name to lowercase before use.
- CS_CNVRT_NOTHING (default) – use the discovered name as it.

## TCP socket buffer size configuration

You can set the size of TCP input and output buffers using the Open Client and Open Server context/connection and server properties. Open Client and Open Server applications use these property settings to set buffer sizes with the operating system setsockopt command. Because setsockopt must be invoked before the TCP connect and accept commands, you must set these Open Client and Open Server properties before attempting to create a connection.

### Properties

The context/connection properties for setting TCP input and output buffer sizes are CS_TCP_RCVBUF and CS_TCP_SNDBUF.

***Table 10: Client-Library properties for buffer size configuration***

| Property | Meaning | *buffer value | Level |
|---|---|---|---|
| CS_TCP_RCVBUF | Size of the input buffer for the client application | A positive integer | Context, connection |
| CS_TCP_SNDBUF | Size of the output buffer for the client application | A positive integer | Context, connection |

Context example

```
ct_config(*context, CS_SET, CS_TCP_RCVBUF, &bufsize,
CS_UNUSED, NULL);
```

Connection example

```
ct_con_props(*connection, CS_SET, CS_TCP_RCVBUF,
&bufsize, CS_UNUSED, NULL);
```

The server properties for setting TCP input and output buffer sizes are SRV_S_TCP_RCVBUF and SRV_S_TCP_SNDBUF.

***Table 11: Server properties for buffer size configuration***

| **Property** | **SET/ CLEAR** | **GET** | **bufp when cmd is CS_SET** | **bufp when cmd is CS_GET** |
|---|---|---|---|---|
| SRV_S_TCP_RCVBUF | Yes | Yes | A CS_INT | A CS_INT |
| SRV_S_TCP_SNDBUF | Yes | Yes | A CS_INT | A CS_INT |

Server example

```
srv_props(cp, CS_SET, SRV_S_TCP_SNDBUF, bufp,
CS_SIZEOF(CS_INT), (CS_INT *)NULL);
```

- Set these parameters as appropriate for your application. For example, if the client is expected to be sending a large amount of data to the server, set CS_TCP_SNDBUF and SRV_S_TCP_RCVBUF to large values to increase the corresponding buffer sizes.
- By default, the socket buffer size is set to the maximum allowable size for the operating system.

## *isql64* and *bcp64* for all 64-bit products

64-bit versions of isql and bcp (isql64 and bcp64) are now available on all the UNIX and Windows platforms that Open Client and Open Server support.

In versions earlier than Open Server and SDK 15.5 ESD #9, only 64-bit isql.exe and bcp.exe are available on 64-bit Windows. If you have a script that references isql.exe or bcp.exe, and you intend to use the 64-bit version, you must change the reference in the script to isql64.exe or bcp64.exe.

## Support for expanded variable-length rows

In Adaptive Server 15.7, the maximum offset of a variable-length column for a data-only-locked (DOL) row has been expanded to 32767 bytes, which allows an Adaptive Server configured with a logical-page size greater than 8K to support wide, variable-length, DOL rows.

The Open Client and Open Server Bulk-Library 15.7 routines, used to populate Adaptive Server logical pages, support the extended DOL rows. This feature is automatically activated in Bulk-Library 15.7 and later, but must be enabled in Adaptive Server.

Databases that are configured for wide DOL rows can accept DOL rows sent from an application that uses Bulk-Library 15.5 or earlier. However, applications that use Bulk-Library 15.7 must not send wide DOL rows to Adaptive Server 15.5 or earlier, or to a database that expects DOL rows in the old format. Otherwise, one of these errors occur:

- `BCP failed to create rows in target table. Column %1! would start at an offset over 8191 bytes; this starting location cannot be represented accurately in the table's (row) format.`
- `BCP failed to create rows in target table. Column %1! starts at an offset greater than %2! bytes; this starting location is not permitted by the current database configuration.`

To correct the error:

- Change the locking scheme of the table from data-only-locked to allpages-locked.
- When connected to Adaptive Server 15.7 or later, enable the allow wide dol rows option in the target database. See Chapter 2, "Data Storage" in the Adaptive Server Enterprise *Performance and Tuning Series: Physical Database Tuning*.

## Row format caching

Open Client 15.7 supports caching row format information, which allows client applications to request data servers to not send the row format information each time a dynamic SQL statement is invoked. Row format caching reduces network traffic between the data server and client applications, thereby improving system performance.

By default, row format caching is enabled in Open Client 15.7. To disable it, set the CS_CMD_SUPPRESS_FMT response capability to CS_FALSE. Use ct_cmd_props() to check and set the value of CS_CMD_SUPPRESS_FMT.

To determine if the server supports row format suppression, check the value of CS_RES_SUPPRESS_FMT using ct_capability().

**Note** This feature is available only when a client application is connected to a server that supports row format caching.

## Support for releasing locks at cursor close

Open Client 15.7, Open Server 15.7, and the Embedded SQL C and COBOL 15.7 processors support the release_locks_on_close cursor option introduced in Adaptive Server 15.7. This feature allows read locks to be released if the cursor closes. See the Adaptive Server Enterprise *Reference Manual: Commands*.

### Client-Library usage

The *option* parameter in the ct_cursor syntax has been extended to include CS_CUR_RELLOCKS_ONCLOSE. Use this option to direct Adaptive Server to release shared locks after a cursor closes. To use with read-only cursors or scrollable cursors, use the bitwise OR operator, "|" (pipe):

- 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

Examples

**Example 1** Declares a cursor that releases its shared locks when it closes:

```
ct_cursor(cmd, CS_CURSOR_DECLARE, cursor_name,
   CS_NULLTERM, select_statement, CS_NULLTERM,
   CS_CUR_RELOCKS_ONCLOSE);
```

**Example 2** Declares an insensitive, scrollable cursor that releases its shared locks when it closes:

```
ct_cursor(cmd, CS_CURSOR_DECLARE, cursor_name,
   CS_NULLTERM, select_statement, CS_NULLTERM,
   CS_CUR_RELOCKS_ONCLOSE | CS_SCROLL_INSENSITIVE);
```

For a sample Open Client program that illustrates this feature, see *csr_disp_scrollcurs3.c*.

### Open Server usage

When client applications declare a cursor with the CS_CUR_RELLOCKS_ONCLOSE option specified, Open Server sets the curstatus (cursor status) field of the SRV_CURDESC structure to SRV_CUR_RELLOCKS_ONCLOSE.

For illustration, see *cursor.c* in the *ctos* example code.

### ESQL/C and ESQL/COBOL usage

SQL DECLARE syntax in ESQL/C and ESQL/COBOL has been extended to include the RELEASE_LOCKS_ON_CLOSE keyword:

```
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]}]
```

You cannot use RELEASE_LOCKS_ON_CLOSE with an UPDATE clause except in this form:

```
EXEC SQL declare cursor c1 release_locks_on_close
   cursor for select * from T for update of col_a
```

In this case, RELEASE_LOCKS_ON_CLOSE is ignored.

cpre and cobpre cannot generate these ct_cursor() options:

- CS_CUR_RELLOCKS_ONCLOSE | CS_READ_ONLY
- CS_CUR_RELLOCKS_ONCLOSE | CS_FOR_UPDATE

ESQL/C sample code is available in *example8.cp*; ESQL/COBOL sample code is available in *example7.pco*.

## Large objects as stored procedure parameters

Open Client and Open Server 15.7 support using text, unitext, and image as input parameters to stored procedures and as parameters to dynamic SQL statements.

Two connection capabilities have been added to facilitate login negotiation regarding the use of this feature:

- CS_RPCPARAM_LOB – client applications send this request capability to the server to determine whether large object (LOB) datatypes can be used as input parameters to stored procedures. The server clears this capability bit in the initial login negotiation when it cannot support the feature, and an error occurs when you attempt to send LOB parameters to such a server.
- CS_RPCPARAM_NOLOB – client applications send this response capability to request the server to withhold sending LOB data as parameters. This capability is turned on by default.

## Sending small amounts of LOB data as parameters

Sending a small amount of LOB data as an input parameter to stored procedures or as a parameter to a prepared SQL statement is the same as sending non-LOB parameters.

To send a small amount of LOB data, allocate memory for the command and data and directly send these to the server using ct_param() or ct_setparam().

You must set the *maxlength* field for the CS_DATAFMT structure when using text, unitext, or image parameters. The *maxlength* value indicates whether all of the LOB data is sent at once or streamed to the server. When *maxlength* is greater than zero, the LOB data is sent in one chunk. When *maxlength* is set to CS_UNUSED, the LOB data is sent in a stream, using a loop of ct_send_data() calls to send the data in chunks. A chunk length of zero indicates the end of the data stream.

**Example 1** Sends a small amount of LOB data as an input parameter to a stored procedure:

```
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);

...
```

**Example 2** Sends a small amount of LOB data using a prepared statement:

```
/*
** 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);
...
}
```

## Sending large amounts of LOB data as parameters

Large amounts of LOB data is sent in streams to the server to better manage resources. Use ct_send_data() in a loop to send data to the server in chunks.

To send a LOB data parameter in chunks, use these settings to define the parameter:

- Set the *datatype* field of the CS_DATAFMT structure to CS_TEXT_TYPE, CS_UNITEXT_TYPE, or CS_IMAGE_TYPE.
- Set *maxlength* field of the CS_DATAFMT structure to CS_UNUSED.
- Set the **data* pointer argument of the ct_param() function to NULL.
- Set the *datalen* argument of the ct_param() function to 0.

**Example 1** Sends a large LOB data parameter in chunks:

```
#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);
```

**Example 2** Sends LOB data as a stream using a prepared SQL statement:

```
/*
** 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);
```

## Retrieving LOB parameters in Open Server

Open Server can retrieve the complete LOB parameter data at once using srv_xferdata or in chunks using the new srv_get_data routine. Open Server retrieves LOB parameters in chunks when the parameter length has been set to CS_UNUSED. See “srv_get_data” on page 121.

**Example** Retrieves description of LOB parameters:

```
    /*
    ** 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

Reads a text, unitext or image parameter stream from a client, in chunks.

Syntax

```
CS_RETCODE srv_get_data(spp, bp, buflen, outlenp)

SRV_PROC *spp;
CS_BYTE *bp;
CS_INT buflen;
CS_INT *outlenp;
```

Parameters

- *spp* – a pointer to an internal thread control structure.
- *bp* – a pointer to a buffer where the data from the client is placed.
- *buflen* – size of the **bp* pointer. This indicates how many bytes are transferred in each chunk.
- *outlenp* – an output parameter, *outlenp* contains the number of bytes read into the **bp* buffer.

Return Values

- CS_SUCCEED – srv_get_data() ran successfully, more data is pending.
- CS_FAIL – the routine failed.
- CS_END_DATA – srv_get_data() has completed reading the entire text, unitext, or image parameter.

## SDK 15.7 features for jConnect and Adaptive Server Enterprise Drivers and Providers

This section describes the new features introduced in SDK 15.7 for jConnect, the Adaptive Server Enterprise ODBC Driver, the Adaptive Server Enterprise OLE DB Provider, and the Adaptive Server Enterprise ADO.NET Data Provider.

## ODBC driver version information utility

The odbcversion utility displays information about the ODBC driver.

Syntax

odbcversion
-version |
-fullversion |
-connect *dsn userid password*

Parameters

-version

displays a simple numeric version string for the ODBC driver.

-fullversion

displays the verbose version string for the ODBC driver.

-connect *dsn userid password*

displays the Adaptive Server version and the version of ODBC and OLEDB MDA scripts installed on that Adaptive Server. Three variables are required with this parameter: *dsn*, which is the data source name for the Adaptive Server, and the user ID and password used to connect to the Adaptive Server.

Example

Obtain the simple numeric version string of an ODBC driver used to connect to Adaptive Server:

```
odbcversion -version
```

The utility returns a numeric version string:

```
15.05.00.1015
```

Usage

When no parameters are specified, the odbcversion utility displays a list of valid parameters.

## SupressRowFormat2 connection string property

With Adaptive Server Enterprise ODBC Driver 15.7, Adaptive Server Enterprise OLE DB Provider 15.7, and Adaptive Server Enterprise ADO.NET Data Provider 15.7, you can use the SupressRowFormat2 connection string property to force Adaptive Server to send data using the TDS_ROWFMT byte sequence where possible instead of the TDS_ROWFMT2 byte sequence. TDS_ROWFMT contains less data than TDS_ROWFMT2—which includes catalog, schema, table, and column information—and can result in better performance for many small select operations. Because the server sends reduced result set metadata when SupressRowFormat2 is set to 1, some information is not available to client programs. If your application relies on the missing metadata, you should not enable this property.

Values:

- 0 – the default value; TDS_ROWFMT2 is not suppressed.
- 1 – forces the server to send data in TDS_ROWFMT where possible.

Example

This connection string forces the server to send data in TDS_ROWFMT where possible on a connection made with ADO.NET Data Provider.

```
Data Source='myASE';Port=5000;Database=myDB;
Uid=myUID;Pwd=myPWD;SupressRowFormat2=1
```

## Enhancement to UseCursor Property

You can use the UseCursor connection string property of Adaptive Server Enterprise ODBC Driver to determine how server-side cursors are used for SQL statements that generate result sets. This property has been updated to allow a client application to control which statements create server-side cursors (value 2).

Values:

- 0 – the default value. Server-side cursors are not used.
- 1 – server-side cursors are used for all statements that generate result sets.
- 2 – server-side cursors are used for statements that generate result sets only when the SQLSetCursorName ODBC function is called. Because cursors use more resources, this setting allows you to limit the use of server-side cursors to statements that benefit from them.

## Logging without ODBC Driver Manager tracing

In Adaptive Server Enterprise ODBC Driver 15.7, you can log calls to ODBC APIs without using ODBC Driver Manager tracing. This is useful when the driver manager is not used or when running on a platform that does not support tracing.

To enable this feature on Microsoft Windows, use the LOGCONFIGFILE environment variable or the Microsoft Windows registry. To enable on Linux, use LOGCONFIGFILE.

When using LOGCONFIGFILE, set the environment variable to the full path of the ODBC log's configuration file. LOGCONFIGFILE overrides any existing registry entry.

When using the Microsoft Windows registry, create an entry called LogConfigFile in *HKEY_CURRENT_USER\Software\Sybase\ODBC* or *HKEY_LOCAL_MACHINE\Software\Sybase\ODBC*, and set its value to the full path of the ODBC log's configuration file. For example:

```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Sybase\ODBC]
"LogConfigFile"="c:\\temp\\odbclog.properties"
```

To disable logging, delete or rename the *LogConfigFile* value.

**Note** The value specified in *HKEY_CURRENT_USER* overrides any value set in *HKEY_LOCAL_MACHINE*.

### Log configuration file

The configuration file controls the format and location of the ODBC log file. In this example, the line in bold specifies where the log file is saved:

```
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=false
```

```
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 enhancement

JDBC programs use Statement.setMaxRows(int max) to limit the number of rows returned by a result set. In jConnect 7.0 and earlier, the result of the select, insert, update, and delete statements are counted against the limit.

To be consistent with the JDBC specification, jConnect 7.07 introduces the SETMAXROWS_AFFECTS_SELECT_ONLY connection property, which, when set to true (the default), limits only the rows returned by select statements.

SETMAXROWS_AFFECTS_SELECT_ONLY is ignored when connected to Adaptive Server 15.5 or earlier.

## TDS ProtocolCapture

Adaptive Server Enterprise ODBC Driver 15.7 introduces the ProtocolCapture connection string property which specifies a file for receiving Tabular Data Stream™ (TDS) packets exchanged between an ODBC application and Adaptive Server. ProtocolCapture takes effect immediately so that TDS packets exchanged during connection establishment are written to a unique filename generated using the file prefix. TDS packets are written to the file for the duration of the connection. You can use Ribo and other protocol translation tools to interpret the TDS capture file.

For example, to specify *tds_capture* as the TDS tracing log file prefix, type:

```
Driver=AdaptiveServerEnterprise;server=server1;
port=port1;UID=sa;PWD=;ProtocolCapture=tds_capture;
```

The first connection generates *tds_capture0.tds,* the second generates *tds_capture1.tds,* and so forth.

## ODBC data batching without binding parameter arrays

When the same SQL statement is executed for different parameter values, client applications normally bind parameter arrays and execute each set of parameters using SQLExecute, SQLExecuteDirect, and SQLBulkOperations. In binding arrays to SQL parameters, memory for the array is allocated, and all data is copied to the array before the SQL statement is executed. This can lead to inefficient use of memory and resources when processing high volume of transactions.

In Adaptive Server Enterprise ODBC Driver 15.7, client applications can use SQLExecute to send parameters in batches to Adaptive Server, without binding the parameters as arrays. SQLExecute returns SQL_BATCH_EXECUTING until the last batch of parameters has been sent and processed. It returns the status of the execution after the final batch of parameters has been processed.

A call to SQLRowCount is valid only after the final SQLExecute statement has completed.

### Managing data batches

SQL_ATTR_BATCH_PARAMS, a Sybase-specific connection attribute, has been introduced to manage the batches of parameters sent to Adaptive Server. Set SQL_ATTR_BATCH_PARAMS using SQLSetConnectAttr.

Values:

- SQL_BATCH_ENABLED – informs Adaptive Server Enterprise ODBC Driver to batch the parameters. When in this state, the driver sends an error if a statement other than the statement being processed—the first statement executed after setting SQL_ATTR_BATCH_PARAMS to SQL_BATCH_ENABLED—by SQLExecute is executed on the connection.
- SQL_BATCH_LAST_DATA – specifies that the next batch of parameters is the last batch, and that the parameters contain data.
- SQL_BATCH_LAST_NO_DATA – specifies that the next batch of parameters is the last batch, and to ignore the parameters.
- SQL_BATCH_CANCEL – informs the Adaptive Server Enterprise ODBC Driver to cancel the batch and to roll back the transactions.

  Only uncommitted transactions can be rolled back.

- SQL_BATCH_DISABLED – (default value) Adaptive Server Enterprise ODBC Driver returns to this state after processing the last batch of parameters. You cannot manually set SQL_ATTR_BATCH_PARAMS to this value.

## Examples

**Example 1** Sends a batch of parameters to the server without binding parameter arrays:

```
// 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);
}
```

**Example 2** Ends and closes a batch:

```
// 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);
```

### Considerations

- This feature supports only statements and stored procedures that do not return a result set or have an output parameter.
- Asynchronous mode is not supported. While in batch mode, the application cannot execute any statement on the same connection other than the one being batched.
- SQL_DATA_AT_EXEC is not supported. Bind LOB parameters as normal parameters.
- When batching data without binding parameter arrays and SQL_ATTR_PARAM_STATUS_PTR is set, Adaptive Server Enterprise ODBC Driver retrieves the number of array elements from the StringLength parameter to SQLSetStmtAttr, and not from SQL_ATTR_PARAMSET_SIZE.

## Optimized batching in jConnect

jConnect for JDBC 7.07 implements an internal algorithm to speed up batch operations for PreparedStatement objects. This algorithm is invoked when the HOMOGENEOUS_BATCH connection property is set to true.

**Note** Homogeneous batching is available only when your client application is connected to a server that supports this feature. Adaptive Server Enterprise 15.7 introduces support for homogeneous batching.

This example illustrates a PreparedStatement batching operation using the addBatch and executeBatch methods:

```
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();
```

where `connection` is a connection instance, `prep_stmt` is a prepared statement instance, and `?` denotes parameter placeholders for the prepared statement.

### Homogeneous batching with LOB columns

When the HOMOGENEOUS_BATCH and ENABLE_LOB_LOCATORS properties are set to true, your client application cannot mix LOB and non-LOB prepared statement setter methods in the same batch. For example, this is invalid:

```
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();
```

where catchphrase is a column of type text. This code fails because the setString method and the setClob method are used in the same batch for the same column.

## jConnect parameter batching without row accumulation

jConnect for JDBC 7.07 adds the SEND_BATCH_IMMEDIATE connection property. When set to true, jConnect sends the parameters for the current row immediately after invoking PreparedStatement.addBatch(). This minimizes usage of client memory and gives the server more time to process the batch parameters.

The default SEND_BATCH_IMMEDIATE value is false, which, when set, signals jConnect to send the batch parameters only after invoking PreparedStatement.executeBatch(), as before.

## jConnect batch update enhancement to execute past errors

jConnect for JDBC 7.07 introduces the EXECUTE_BATCH_PAST_ERRORS connection property, which, when set to true, allows a batch update operation to ignore nonfatal errors encountered while executing individual statements and to complete the batch update. When set to false, the default, batch update is aborted when an error is encountered, as in earlier versions.

See *jConnect for JDBC Programmers Reference* for information about batch update usage and the interpretation of its results.

## Support for releasing locks at cursor close

Adaptive Server 15.7 extends the declare cursor syntax to include the release_locks_on_close option, which releases shared cursor locks at isolation levels 2 and 3 when a cursor is closed. Adaptive Server Enterprise ODBC Driver 15.7 and jConnect for JDBC 7.07 support the release-lock-on-close semantics.

To apply this functionality to all read-only cursors created on an Adaptive Server Enterprise ODBC Driver connection, set the ReleaseLocksOnCursorClose connection property to 1. The default ReleaseLocksOnCursorClose value is 0.

To apply on a jConnect for JDBC connection, set the RELEASE_LOCKS_ON_CURSOR_CLOSE connection property to true. The default RELEASE_LOCKS_ON_CURSOR_CLOSE value is false.

Settings applied through these connection properties are static and cannot be changed after the connection has been established. This setting takes effect only when connected to a server that supports release_locks_on_close.

For information about release_locks_on_close, see the Adaptive Server Enterprise *Reference Manual: Commands*.

## select for update support

Adaptive Server 15.7 supports select for update, which can lock rows for subsequent updates within the same transaction, and supports exclusive locks for updatable cursors. See Chapter 2, "Queries: Selecting Data from a Table" in the Adaptive Server Enterprise *Transact-SQL Users Guide*.

This functionality is automatically available to clients when the for update clause is added to a select statement and to any updatable cursors opened within the clients. See *Adaptive Server Enterprise ODBC Driver Users Guide* and *jConnect for JDBC Programmers Reference* for information about creating updatable cursors.

## Support for expanded variable-length rows

Versions of Adaptive Server earlier than 15.7 configured for 16K logical page sizes could not create data-only locked (DOL) tables with variable-length rows if a variable-length column began more than 8191 bytes after the start of the row. This limitation has been removed starting in Adaptive Server 15.7. See Chapter 2, "Data Storage" in the Adaptive Server Enterprise *Performance and Tuning Series: Physical Database Tuning*.

ODBC and JDBC clients do not need special configuration to use this feature. When connected to Adaptive Server version 15.7 that has been configured to receive wide DOL rows, these clients automatically insert records using the wide offset. An error message is received if a client attempts to send a wide DOL row to an earlier version of Adaptive Server, or to a Adaptive Server 15.7 for which the wide DOL row option is disabled.

## Support for nonmaterialized columns

The bulk insert routines in the Adaptive Server Enterprise ODBC Driver 15.7 have been enhanced to handle nonmaterialized columns in Adaptive Server 15.7. Earlier versions of the Adaptive Server Enterprise ODBC Driver cannot perform bulk inserts of data into Adaptive Server when a table definition contains nonmaterialized columns. Adaptive Server raises an error when earlier versions of the Adaptive Server Enterprise ODBC Driver attempt to perform bulk inserts into nonmaterialized columns.

## In-row and off-row LOB storage support

In Adaptive Server 15.7, LOB columns that are marked for in-row storage are stored in-row when there is adequate memory to hold the entire row. When the size of a row increases over its defined limit due to an update to any column in it, the LOB columns which are stored in-row are moved off-row to bring it within the limits. See Chapter 21, "In-Row Off-Row LOB" in the Adaptive Server Enterprise *Transact-SQL Users Guide*.

The bulk insert routines in Adaptive Server Enterprise ODBC Driver 15.7 and jConnect for JDBC 7.07 support the in-row and off-row storage of text, image, and unitext LOB columns in Adaptive Server. Bulk insert routines from earlier client versions always store LOB columns off row.

## Large objects as stored procedure parameters

Passing LOB data as stored procedure input parameters has also been introduced in Adaptive Server 15.7.

jConnect for JDBC 7.07, Adaptive Server Enterprise ODBC Driver 15.7, Adaptive Server Enterprise OLE DB Provider 15.7, and Adaptive Server Enterprise ADO.NET Data Provider 15.7 support using text, unitext, and image as input parameters in stored procedures and as parameter marker datatypes.

## Large object locator support

jConnect for JDBC 7.07 and Adaptive Server Enterprise ODBC Driver 15.7 support large object (LOB) locators. A LOB locator contains a logical pointer to LOB data rather than the data itself, reducing the amount of data that passes through the network between Adaptive Server and its clients. Server support for LOB locators was introduced in Adaptive Server 15.7.

jConnect for JDBC 7.07 accesses LOB data using server-side locators when connected to an Adaptive Server that supports LOB locators and autocommit is turned off. Otherwise, jConnect materializes LOB data at the client side. You can use the complete LOB API with client-side materialized LOB data, however, due to larger data, API performance may be different than when used with LOB locators.

Adaptive Server Enterprise ODBC Driver 15.7 clients cannot use LOB locators unless connected to an Adaptive Server that supports it.

**Note** When you are using LOB locators, retrieving a large result set that includes LOB data on each row may impact your application's performance. Adaptive Server returns a LOB locator as part of the result set and, to obtain LOB data, jConnect and ODBC Driver must cache the remaining result set. Sybase recommends that you keep result sets small, or that you enable cursor support to limit the size of data to be cached.

## jConnect for JDBC support

To enable LOB locator support, establish a connection to Adaptive Server with the ENABLE_LOB_LOCATORS connection property set to true. When enabled, client applications can access the locators using the Blob, Clob, and NClob classes from the java.sql package.

**Note** When both LOB locators and autocommit are enabled, jConnect automatically switches the LOB locators to client-side-materialized LOB locators even if the connected Adaptive Server is capable of supporting them. This increases the memory used by the client and may degrade performance. Therefore, it is advisable to use LOB locators with autocommit off.

For information about the Blob, Clob, and NClob classes, see the Java documentation.

## Adaptive Server Enterprise ODBC Driver support

To enable LOB locator support, establish a connection to Adaptive Server with the EnableLOBLocator connection property set to 1. When EnableLOBLocator is set to 0, the default value, the ODBC Driver cannot retrieve a locator for a LOB column. When enabling LOB Locators, the connection should be set to autocommit off.

You must also include the *sybasesqltypes.h* file in your program. The *sybasesqltypes.h* file is located in the *include* directory, under the ODBC installation directory.

### ODBC datatype mapping for locator support

The ODBC datatype mapping for the Adaptive Server locator datatypes are:

| ASE Datatype | ODBC SQL Type | ODBC C Type |
|---|---|---|
| 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 |

## Supported conversions

The supported conversions for the Adaptive Server locator datatypes are:

| | SQL_C_TEXT_ LOCATOR | SQL_C_IMAGE_ LOCATOR | SQL_C_UNITEXT_ LOCATOR |
|---|---|---|---|
| SQL_TEXT_LOCATOR | X | | |
| SQL_IMAGE_LOCATOR | | X | |
| SQL_UNITEXT_LOCATOR | | | X |
| SQL_LONGVARCHAR | | | |
| SQL_WLONGVARCHAR | | | |
| SQL_LONGVARBINARY | | | |

LEGEND: x = supported conversion.

## ODBC API methods that support LOB locators

- SQLBindCol – *TargetType* can be any of the ODBC C locator datatypes.
- SQLBindParameter – *ValueType* can be any of the ODBC C locator datatypes. *ParameterType* can be any of the ODBC SQL locator datatypes.
- SQLGetData – *TargetType* can be any of the ODBC C locator datatype.
- SQLColAttribute – the SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE descriptors can return any of the ODBC SQL locator datatype.
- SQLDescribeCol – the datatype pointer can be any of the ODBC SQL locator datatypes.

See *Microsoft ODBC API Reference*.

## Implicit conversion of prefetched LOB data

In Adaptive Server Enterprise ODBC Driver 15.7, when Adaptive Server returns a LOB locator, you can use SQLGetData and SQLBindCol to retrieve the underlying prefetched LOB data by binding the column to SQL_C_CHAR or SQL_C_WCHAR for text locators, or to SQL_C_BINARY for image locators.

Set the SQL_ATTR_LOBLOCATOR attribute to enable or disable locators in a connection. If EnableLOBLocator has been specified in the connection string, SQL_ATTR_LOBLOCATOR is initialized with the value of EnableLOBLocator, otherwise, it is set to SQL_LOBLOCATOR_OFF, the default value. To enable locators, set the attribute to SQL_LOBLOCATOR_ON. Use SQLSetConnectAttr to set the attribute's value and SQLGetConnectAttr to retrieve its value.

Use SQLSetStatementAttr to set SQL_ATTR_LOBLOCATOR_FETCHSIZE to specify the size—in bytes for binary data and in characters for character data—of the LOB data to retrieve. The default value, 0, indicates that prefetched data is not requested, while a value of -1 retrieves the entire LOB data.

**Note** If the underlying LOB data size of the column being retrieved exceeds the prefetched data size that you have set, native error 3202 is raised when an ODBC client attempts to directly retrieve the data. When this happens, the client can retrieve the complete data by calling SQLGetData to obtain the underlying locator and perform all of the operations available on locators.

**Example 1** Retrieves an image locator using SQLGetData when the prefetched data represents the complete LOB value:

```
//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);
```

**Example 2** Retrieves an image locator using SQLGetData when prefetched data represents a truncated LOB value:

```
//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);
```

### Accessing and manipulating LOBs using locators

The ODBC API does not directly support LOB locators. An ODBC client application must use Transact-SQL functions to operate on the locators and manipulate LOB values. Adaptive Server Enterprise ODBC Driver introduces several stored procedures to facilitate the use of the required Transact-SQL functions.

This section discusses how various operations can be performed on a LOB locator. The input and output values of the parameters can be of any type that Adaptive Server can implicitly convert to the stored procedure definitions.

For details about the Transact-SQL commands and functions listed here, see "Transact-SQL Functions" in the Adaptive Server Enterprise *Reference Manual: Building Blocks*.

#### Initializing a *text* locator

Use sp_drv_create_text_locator to create a text_locator and optionally initialize it with a value. sp_drv_create_text_locator accesses the Transact-SQL function create_locator.

| | |
|---|---|
| Syntax | sp_drv_create_text_locator [*init_value*] |
| Input Parameters | *init_value* – a varchar or text value used to initialize the new locator. |
| Output Parameters | None. |
| Result Set | A column of type text_locator. The LOB that the locator references has *init_value* when supplied. |

### Initializing a *unitext* locator

Use sp_drv_create_unitext_locator to create a unitext_locator and optionally initialize it with value. sp_drv_create_unitext_locator accesses the Transact-SQL function create_locator.

| | |
|---|---|
| Syntax | sp_drv_create_unitext_locator [*init_value*] |
| Input Parameters | *init_value* – a univarchar or unitext used to initialize the new locator. |
| Output Parameters | None. |
| Result Set | A column of type unitext_locator. The LOB that the locator references has *init_value* when supplied. |

### Initializing an *image* locator

Use sp_drv_create_image_locator to create an image_locator and optionally initialize it with value. sp_drv_create_image_locator accesses the Transact-SQL function create_locator.

| | |
|---|---|
| Syntax | sp_drv_create_image_locator [*init_value*] |
| Input Parameters | *init_value* – a varbinary or image used to initialize the new locator. |
| Output Parameters | None. |
| Result Set | A column of type image_locator. The LOB that the locator references has *init_value* when supplied. |

### Obtaining complete *text* value from a *text* locator

Use sp_drv_locator_to_text, which accesses the Transact-SQL function return_lob.

| | |
|---|---|
| Syntax | sp_drv_locator_to_text *locator* |
| Input Parameters | *locator* – text_locator to retrieve value of. |
| Output Parameters | None. |
| Result Set | A column containing the text value referenced by *locator*. |

### Obtaining complete *unitext* value from a *unitext* locator

Use sp_drv_locator_to_unitext, which accesses the Transact-SQL function return_lob.

| | |
|---|---|
| Syntax | sp_drv_locator_to_unitext *locator* |
| Input Parameters | *locator* – unitext_locator to retrieve value of. |

Output Parameters

None.

Result Set

A column containing the unitext value referenced by *locator*.

### Obtaining complete *image* value from an *image* locator

Use sp_drv_locator_to_image, which accesses the Transact-SQL function return_lob.

Syntax

```
sp_drv_locator_to_image locator
```

Input Parameters

*locator* – image_locator to retrieve value of.

Output Parameters

None.

Result Set

A column containing the image value referenced by *locator*.

### Obtaining a substring from a *text* locator

Use sp_drv_text_substring, which accesses the Transact-SQL function substring.

Syntax

```
sp_drv_text_substring locator, start_position, length
```

Input Parameters

- *locator* – a text_locator that references the data to manipulate.
- *start_position* – an integer specifying the position of the first character to read and retrieve.
- *length* – an integer specifying the number of characters to read.

Output Parameters

None.

Result Set

A column of type text containing the substring retrieved.

### Obtaining a substring from a *unitext* locator

Use sp_drv_unitext_substring, which accesses the Transact-SQL function substring.

Syntax

```
sp_drv_unitext_substring locator, start_position, length
```

Input Parameters

- *locator* – a unitext_locator that references the data to manipulate.
- *start_position* – an integer specifying the position of the first character to read and retrieve.
- *length* – an integer specifying the number of characters to read.

Output Parameters

None.

Result Set

A column of type unitext containing the substring retrieved.

### Obtaining a substring from an *image* locator

Use sp_drv_image_substring, which accesses the Transact-SQL function substring.

Syntax

sp_drv_image_substring *locator*, *start_position*, *length*

Input Parameters

- *locator* – an image_locator that references the data to manipulate.
- *start_position* – an integer specifying the position of the first byte to read and retrieve.
- *length* – an integer specifying the number of bytes to read.

Output Parameters

None.

Result Set

A column of type image containing the substring retrieved.

### Inserting *text* at specified position

Use sp_drv_text_setdata, which accesses the Transact-SQL function setadata.

Syntax

sp_drv_text_setdata *locator*, *offset*, *new_data*, *data_length*

Input Parameters

- *locator* – a text_locator that references the text column to insert into.
- *offset* – an integer specifying the position from which to start writing the new content.
- *new_data* – varchar or text data to insert.

Output Parameters

*data_length* – an integer containing the number of characters written.

Result Set

None.

### Inserting *unitext* at specified position

Use sp_drv_unitext_setdata, which accesses the Transact-SQL function setadata.

Syntax

sp_drv_unitext_setdata *locator*, *offset*, *new_data*, *data_length*

Input Parameters

- *locator* –a unitext_locator that references the unitext column to insert into.
- *offset* – an integer specifying the position from which to start writing the new content.
- *new_data* – univarchar or unitext data to insert.

Output Parameters

*data_length* – an integer containing the number of characters written.

Result Set

None.

### Inserting an *image* at specified position

Use sp_drv_image_setdata, which accesses the Transact-SQL function setadata.

Syntax

sp_drv_image_setdata *locator*, *offset*, *new_data*, *datalength*

Input Parameters

- *locator* – an image_locator that references the image column to insert in.
- *offset* – an integer specifying the position from which to start writing the new content.
- *new_data* – varbinary or image data to insert.

Output Parameters

*data_length* – an integer containing the number of bytes written.

Result Set

None.

### Inserting *text* referenced by a locator

Use sp_drv_text_locator_setdata, which accesses the Transact-SQL function setadata.

Syntax

sp_drv_text_locator_setdata *locator*, *offset*, *new_data_locator*, *data_length*

Input Parameters

- *locator* – a text_locator that references the text column to insert into.
- *offset* – an integer specifying the position from which to start writing the new content.
- *new_data_locator* – a text_locator that references the text data to insert.

Output Parameters

*data_length* – an integer containing the number of characters written.

Result Set

None.

### Inserting *unitext* referenced by a locator

Use sp_drv_unitext_locator_setdata, which accesses the Transact-SQL function setadata.

Syntax

sp_drv_unitext_locator_setdata *locator*, *offset*, *new_data_locator*, *data_length*

Input Parameters

- *locator* –a unitext_locator that references the unitext column to insert into.
- *offset* – an integer specifying the position from which to start writing the new content.
- *new_data_locator* – a unitext_locator that references the unitext data to insert.

Output Parameters — *data_length* – an integer containing the number of characters written.

Result Set — None.

### Inserting *image* referenced by a locator

Use sp_drv_image_locator_setdata, which accesses the Transact-SQL function setadata.

Syntax

```
sp_drv_image_locator_setdata locator, offset, new_data_locator,
  datalength
```

Input Parameters

- *locator* – an image_locator that references the image column to insert in.
- *offset* – an integer specifying the position from which to start writing the new content.
- *new_data_locator* – an image_locator that references the image data to insert.

Output Parameters — *data_length* – an integer containing the number of bytes written.

Result Set — None.

### Truncating underlying LOB data

Use truncate lob to truncate the LOB data referenced by a LOB locator. See the Adaptive Server Enterprise *Reference Manual: Commands*.

### Finding character length of underlying *text* data

Use sp_drv_text_locator_charlength to find the character length of a LOB column referenced by a text locator. sp_drv_text_locator_charlength accesses the Transact-SQL function char_length.

Syntax

```
sp_drv_text_locator_charlength locator, data_length
```

Input Parameters — *locator* – a text_locator that references the text column to manipulate.

Output Parameters — *data_length* – an integer specifying the character length of the text column referenced by *locator*.

Result Set — None.

### Finding byte length of underlying *text* data

Use sp_drv_text_locator_bytelength to find the byte length of a LOB column referenced by a text locator. sp_drv_text_locator_bytelength accesses the Transact-SQL function data_length.

| | |
|---|---|
| Syntax | sp_drv_image_locator_bytelength *locator*, *data_length* |
| Input Parameters | *locator* – a text_locator that references the text column to manipulate. |
| Output Parameters | *data_length* – an integer specifying the byte length of the text column referenced by *locator*. |
| Result Set | None. |

### Finding character length of underlying *unitext* data

Use sp_drv_unitext_locator_charlength to find the character length of a LOB column referenced by a unitext locator. sp_drv_unitext_locator_charlength accesses the Transact-SQL function char_length.

| | |
|---|---|
| Syntax | sp_drv_unitext_locator_charlength *locator*, *data_length* |
| Input Parameters | *locator* – a unitext_locator that references the unitext column to manipulate. |
| Output Parameters | *data_length* – an integer specifying the character length of the unitext column referenced by *locator*. |
| Result Set | None. |

### Finding byte length of underlying *unitext* data

Use sp_drv_unitext_locator_bytelength to find the byte length of a LOB column referenced by a unitext locator. sp_drv_unitext_locator_bytelength accesses the Transact-SQL function data_length.

| | |
|---|---|
| Syntax | sp_drv_image_locator_bytelength *locator*, *data_length* |
| Input Parameters | *locator* – a unitext_locator that references the unitext column to manipulate. |
| Output Parameters | *data_length* – an integer specifying the byte length of the unitext column referenced by *locator*. |
| Result Set | None. |

### Finding byte length of underlying *image* data

Use sp_drv_image_locator_bytelength to find the byte length of a LOB column referenced by an image locator. sp_drv_image_locator_bytelength accesses the Transact-SQL function data_length.

| | |
|---|---|
| Syntax | sp_drv_image_locator_bytelength *locator*, *data_length* |
| Input Parameters | *locator* – an image_locator that references the image column to manipulate. |
| Output Parameters | *data_length* – an integer specifying the byte length of the image column referenced by *locator*. |

Result Set — None.

### Finding position of a search string within the *text* column referenced by a locator

Use sp_drv_varchar_charindex, which accesses the Transact-SQL function charindex.

Syntax

```
sp_drv_varchar_charindex search_string, locator, start, position
```

Input Parameters

- *search_string* – the literal, of type varchar, to search for.
- *locator* – a text_locator that references the text column to search from.
- *start* – an integer specifying the position from which to begin searching. The first position is 1.

Output Parameters — *position* – an integer specifying the starting position of *search_string* in the LOB column referenced by *locator*.

Result Set — None.

### Finding position of a string referenced by a *text* locator within the *text* Column referenced by another locator

Use sp_drv_textlocator_charindex, which accesses the Transact-SQL function charindex.

Syntax

```
sp_drv_textlocator_charindex search_locator, locator, start, position
```

Input Parameters

- *search_locator* – a text_locator that points to the literal to search for.
- *locator* – a text_locator that references the text column to search from.
- *start* – an integer specifying the position from which to begin searching. The first position is 1.

Output Parameters — *position* – an integer specifying the starting position of the literal in the LOB column referenced by *locator*.

Result Set — None.

### Finding position of a search string within the *unitext* column referenced by a locator

Use sp_drv_univarchar_charindex, which accesses the Transact-SQL function charindex.

Syntax

```
sp_drv_univarchar_charindex search_string, locator, start, position
```

Input Parameters

- *search_string* – the literal, of type univarchar, to search for.

- *locator* – a unitext_locator that references the unitext column to search from.
- *start* – an integer specifying the position from which to begin searching. The first position is 1.

Output Parameters: *position* – an integer specifying the starting position of *search_string* in the LOB column referenced by *locator*.

Result Set: None.

### Finding position of a string referenced by a *unitext* locator within the *unitext* column referenced by another locator

Use sp_drv_unitext_locator_charindex, which accesses the Transact-SQL function charindex.

Syntax:

sp_drv_charindex_unitextloc_in_locator *search_locator*, *locator*, *start*, *position*

Input Parameters:

- *search_locator* – a unitext_locator that points to the literal to search for.
- *locator* – a unitext_locator that references the text column to search from.
- *start* – an integer specifying the position from which to begin searching. The first position is 1.

Output Parameters: *position* – an integer specifying the starting position of the literal in the LOB column referenced by *locator*.

Result Set: None.

### Finding position of a byte sequence within the column referenced by an *image* locator

Use sp_drv_varbinary_charindex, which accesses the Transact-SQL function charindex.

Syntax:

sp_drv_varbinary_charindex *byte_sequence*, *locator*, *start*, *position*

Input Parameters:

- *byte_sequence* – the byte sequence, of type varbinary, to search for.
- *locator* – an image_locator that references the image column to search from.
- *start* – an integer specifying the position from which to begin searching. The first position is 1.

Output Parameters: *position* – an integer specifying the starting position of *search_string* in the LOB column referenced by *locator*.

Result Set: None.

### Finding position of byte sequence referenced by an *image* locator within the *image* column referenced by another locator

Use sp_drv_image_locator_charindex, which accesses the Transact-SQL function charindex.

Syntax

sp_drv_image_locator_charindex *sequence_locator*, *locator*, *start*, *start_position*

Input Parameters

- *sequence_locator* – an image_locator that points to the byte sequence to search for.
- *locator* – an image_locator that references the image column to search from.
- *start* – an integer specifying the position from which to begin searching. The first position is 1.

Output Parameters

*start_position* – an integer specifying the starting position of the byte sequence in the LOB column referenced by *locator*.

Result Set

None.

### Checking if a *text_locator* reference is valid

Use sp_drv_text_locator_valid, which accesses locator_valid.

Syntax

sp_drv_text_locator_valid *locator*

Input Parameters

*locator* – the text_locator to validate.

Output Parameters

A bit representing one of these values:

- 0 – false, *locator* is invalid.
- 1 – true, *locator* is valid.

Result Set

None.

### Checking if a *unitext_locator* reference is valid

Use sp_drv_unitext_locator_valid, which accesses locator_valid.

Syntax

sp_drv_unitext_locator_valid *locator*

Parameters

*locator* – the unitext_locator to validate.

Output Parameters

A bit representing one of these values:

- 0 – false, *locator* is invalid.
- 1 – true, *locator* is valid.

Result Set

None.

### Checking if an *image_locator* reference is valid

Use sp_drv_image_locator_valid, which accesses locator_valid.

Syntax

sp_drv_image_locator_valid *locator*

Parameters

*locator* – the image_locator to validate.

Output Parameters

A bit representing one of these values:

- 0 – false, *locator* is invalid.
- 1 – true, *locator* is valid.

Result Set

None.

### Freeing or deallocating a LOB locator

Use deallocate locator. See the Adaptive Server Enterprise *Reference Manual: Commands*.

### Examples

**Example 1** Allocates handles and establishes a connection:

```
// 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);
```

**Example 2** Selects a column and retrieves a locator:

```
// 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;
}
```

**Example 3** Determines data length:

```
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;
```

**Example 4** Appends text to a LOB column:

```
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);
```

**Example 5** Retrieves LOB data from a LOB locator.

```
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;
```

**Example 6** Transfers data from a client application to a LOB locator.

```
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);
```

## Adaptive Server Enterprise extension module for Python

The Adaptive Server Enterprise extension module for Python provides a Sybase-specific Python interface for executing queries against an Adaptive Server database. This module implements the Python Database API specification version 2.0 with extensions and is for use with Python versions 2.6, 2.7, and 3.1. You can read the Python Database API specification at http://www.python.org/dev/peps/pep-0249.

You can install the Adaptive Server Enterprise extension module for Python from the SDK installer. For installation instructions, see the *Software Developers Kit and Open Server Installation Guide* and the *Software Developers Kit and Open Server Release Bulletin*. For information about using the Adaptive Server Enterprise extension module for Python, see the *Adaptive Server Enterprise Extension Module for Python Programmers Guide*.

## Adaptive Server Enterprise extension module for PHP

The Adaptive Server Enterprise extension module for PHP provides an interface for executing queries against an Adaptive Server database and handling query results and includes the PHP APIs necessary for database access. This module is for use with PHP version 5.3.6. For information about using the Adaptive Server Enterprise extension module for PHP, see the *Adaptive Server Enterprise Extension Module for PHP Programmers Guide*.

## Adaptive Server Enterprise database driver for Perl

The Adaptive Server Enterprise database driver for Perl is called through the generic Perl DBI interface and translates Perl DBI API calls into a form that is understood by Adaptive Server through the Open Client SDK using CT-Lib. It gives Perl scripts direct access to Adaptive Server Enterprise database servers. This driver is for use with Perl version 5.14 and DBI version 1.616.

You can read the Perl DBI specification at http://search.cpan.org/~timb/DBI-1.616/DBI.pm. For information about using the Adaptive Server Enterprise database driver for Perl, see the *Adaptive Server Enterprise Database Driver for Perl Programmers Guide*.

## Deprecated features

These features are not supported for the current release of Open Server and SDK.

### DCE service libraries

The Distributed Computing Environment (DCE) directory services library *libsybddce.dll* and the DCE security services library *libsybsdce.dll* have been removed from Open Client and Open Server for Windows 32-bit platforms. In versions of Open Client and Open Server earlier than 15.7, these libraries resided in the *%SYBASE%\OCS-15_0\dll* directory.

### dsedit_dce utility files

The dsedit_dce X-Windows defaults file, *OCS-15_0/xappdefaults/Dsedit_dce*, and the dsedit_dce help file, *OCS-15_0/sybhelp/dsedit_dceHelpTextMsgs*, have been removed.

### Unsupported platforms

Open Server and SDK do not support HP-UX PA-RISC and Mac OS.

## Accessibility features

Section 508 requires that U.S. Federal agencies' electronic and information technology is accessible to people with disabilities. Sybase strongly supports Section 508 and has made a range of Sybase products Section 508-compliant, including Open Client and Open Server version 15.7.

Documents in the 15.7 release are available in HTML specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger. Open Client and Open Server documentation has been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.