SYBASE®
An SAP® Company

Programmers Guide

Adaptive Server® Enterprise Database Driver for Perl 15.7

DOCUMENT ID: DC01694-01-1570-03
LAST REVISED: June 2012

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

Contents

Adaptive Server Enterprise Database Driver for Perl

The Adaptive Server® Enterprise database driver for the Perl scripting language allows Perl developers to connect to an Adaptive Server database and query or change information using a Perl script.

Perl Driver Module

DBD::SybaseASE is the Adaptive Server database driver for the Perl scripting language.

The DBD::SybaseASE database driver for the Perl scripting language 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-Library.

Using DBI and DBD::SybaseASE, your Perl scripts can directly access Adaptive Server Enterprise database servers.

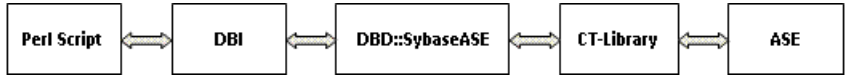
The generic Perl DBI API specification defines a set of methods that provide a database interface that is independent of the actual database being used.

The Perl DBI programmable API calls are documented at *http://search.cpan.org/~timb/DBI-1.616/DBI.pm* .

Note: The DBD::SybaseASE driver cannot function without the DBI. The DBI contains all user-visible APIs.

Perl Driver Data Flow

The following diagram describes how data is moved through the required components.



Required Components

Access to an Adaptive Server database using the Perl programming language requires the following components:

- Perl installation – generic core database API that is database-vendor-agnostic.
- DBD::SybaseASE – database driver for the Perl scripting language.
- CT-Library – (CT-Lib API) is part of the Open Client suite. CT-Library sends commands to Adaptive Server and processes results.
- Adaptive Server Enterprise

- Perl

Version Requirements

For information about platform support, see the *Software Developers Kit and Open Server Installation Guide* for your platform.

- Adaptive Server Enterprise – version 15.7 or later.
- Open Client and Open Server – version 15.7 or later.
- Perl – version 5.14.0 or 5.14.1.
- DBD::SybaseASE driver – no specific version requirements.
- CT-Library – (CT-Lib API) version 15.7.
- Perl DBI – version 1.616.

  The Sybase® installer does not check for a Perl installation or if the driver dependencies are installed on the target system.

Note: The build mode of the Perl driver released for your platform also dictates the build mode of your Perl installation and the DBI. As an example, for Linux the driver is released in 64-bit mode with threading enabled. This means Perl must be configured in full 64-bit mode with threading enabled. The build mode requirement also applies to the DBI interface.

Installing and Configuring the Driver for Perl

The database driver for Perl is a component you can install through the Sybase Installer.

The database driver for Perl is an optional installation component when you choose **Custom** as the installation type. The driver is installed by default if the installation type you choose is **Typical** or **Full**. For installation and configuration instructions, see the *Software Developers Kit and Open Server Installation Guide* for your platform.

Developing Perl Applications

Use the Perl DBI API to develop Perl applications.

SybaseASE Driver Connect Syntax

To use a Perl script to connect to an Adaptive Server, the server must be running and the client installation must be complete with all components installed.

```
"dbi:SybaseASE:attr=value;attr=value", $user_id, $password,
%attrib);
```

Attributes and Methods

These attributes are currently supported when connecting to a server:

- **server** – specifies the server to which you are connecting.

- **database** – specifies which database within the server is the default database at connect time.
- **hostname** — specifies the host name that is stored in the sysprocesses table for this process.

Attribute values can be repeated as long as they are recognized by the driver.

The following is an example of the SybaseASE driver syntax using the **DBI->connect** method:

```
my $dbh = DBI->connect(“dbi:SybaseASE:server=mumbles;database=prod”,
"john_doe",  "xyz" );
```

Currently supported database handle attributes:

Table 1. Database handle attributes

Attribute	Description	Default
dbh->{AutoCommit} = (0|1);	Disables or enables AutoCommit.	0
dbh->{LongTruncOK} = (0|1);	Disables or enables truncation of text and image types.	0
dbh->{LongReadLen}=(int);	Sets the default read chunk size for TEXT and IMAGE data. Example: **dbh->{LongReadLen} = 64000**.	32767
dbh->{syb_show_sql} =(0|1);	If set, the current statement is included in the error string returned by **$dbh->errstr**.	0
dbh->{syb_show_eed} = (0|1);	If set, the extended error information is included in the error string returned by **$dbh->errstr**.	0
dbh->{syb_chained_txn} = (0|1);	If set, CHAINED transactions are used when AutoCommit is off. Use this attribute only during the `connect()` call: `$dbh = DBI->connect('dbi:SybaseASE:', $user, $pwd, {syb_chained_txn => 1});` Using **syb_chained_txn** at any time with AutoCommit turned off forces a commit on the current handle. When set to 0, an explicit **BEGIN TRAN** is issued as needed.	0

Attribute	Description	Default
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 in to a Hexadecimal string.	0
dbh->{syb_quoted_identifier}=(0|1);	Allows identifiers that clash with Sybase re-served words to be used when quoted using "identifier".	0
dbh->{syb_rowcount}=(int);	If set to a nonzero value, the number of rows returned by a SELECT, or affect an UPDATE or DELETE statement are limited to the *rowcount* value. Setting it back to 0 clears the limit.	0
sth->{syb_do_proc_status} = (0|1);	Forces **$sth->execute()** to fetch the return status of a stored procedure executed in the SQL stream. If the return status is nonzero, **$sth->execute()** return `undef` (that is, fail). Setting this attribute does not affect existing statement handles; only those that are created after setting it. To revert behavior of an existing **$sth** handle, execute: **$sth->{syb_do_proc_status} = 0**;	0
dbh->{syb_flush_finish}	If set, **$dbh->finish** drains any results remaining for the current command by actually fetching them. This can be used in place of the driver issuing a **ct_cancel()** command.	
_date_fmt	This private method sets the default date conversion and display formats. Currently, the only supported option is `locale = “C”` which must be set, otherwise the conversion fails.	

Perl Supported Datatypes

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

String Types

- char
- varchar
- binary
- varbinary
- text
- image
- unichar
- univarchar

Numeric Types

- integer
- smallint
- tinyint
- money
- smallmoney
- float
- real
- double
- numeric
- decimal
- bit
- bigint

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

Date and Time Datatypes

- datetime
- date
- time
- bigtime
- bigdatetime

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” locale and must be either set as `LC_ALL` or `LANG` to `“C”` before using the driver.

For information about datatypes, see *Using Open Client and Server Datatypes* in the *Open Client Client-Library/C Programmers Guide*.

Using Multiple Statements

The Perl driver allows multiple statements to be prepared with one call and then executed with one call.

The Sybase ASE Database Driver for Perl currently allows for 25 concurrent database connections to any database. Results of multiple statements prepared with one call are sent back to the client as a single stream of data. Each distinct set of results is treated as a normal single result set, this means that the 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 data has been retrieved.

You can use the `syb_more_results` statement handle attribute to check for result sets in the pipeline.

Example:

```
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.

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.

Login, application names, password lengths cannot exceed 30 characters.

Database Parameter Support

Configure the database parameters `locale` and `charset`.

The database parameters `locale` and `charset` can be configured through the driver and set as follows:

- At the environment level, either set `LC_ALL` or `LANG` to `“C”`.
- Use the default character set customary for your platform. The most commonly used default character sets are iso_1 and for HP/UX, roman8.

Currently there is no DSN support to change the locale or charset at connection time.

See the *Open Client and Open Server International Developers Guide* for more information about character sets.

Dynamic SQL Support

Dynamic SQL is supported by the driver for Perl, including usage of parameters.

Use '?' style marker for the parameters.

Default Date Conversion and Display Format

You can set your own default date conversion and display format using the `_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:

Feb 16 2012 12:07PM

Under this same default locale, Sybase understands several additional input formats:

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

Use `_date_fmt()` (accessible using **$dbh->func()**) to change the date input and output format.

Table 2. Support Date 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

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

LONG/BLOB Data Handling

Adaptive Server Enterprise database driver for Perl supports an IMAGE and a TEXT type for LONG/BLOB data. Each type can hold up to 2GB of binary data.

The default size limit for TEXT/IMAGE data is 32KB. Use the LongReadLen attribute to change this limit.

Bind parameters cannot be used to insert TEXT or IMAGE data.

Automatic Key Generation

Adaptive Server Enterprise database driver for Perl 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
```

Sequence generators are not supported; use stored procedures to generate sequential identity values.

Parameter Binding

Parameter binding is directly supported by Sybase.

Only the **?** style parameter is supported, not the **:1** placeholder types. Attempting to bind a TEXT or IMAGE datatype results in failure.

Stored Procedures

Adaptive Server Enterprise database driver for Perl supports stored procedures with input and output parameters.

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
]);
```

Stored procedures cannot be called with bound parameters:

Illegal:

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

Correct:

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

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

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

Error Handling

All errors from the Database Driver for Perl and CT-Lib are propagated into the DBI layer.

An exception is errors or warnings that must be reported during driver startup, when there is no context available yet.

The DBI performs basic error reporting when the **PrintError** attribute is enabled.

To track problems with your program or problems at the system level, enable tracing on DBI operations using the DBI **trace** method.

Configuring Security Services

Configure security options using the `ocs.cfg` and `libtcl.cfg` files.

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

Note: When using the `ocs.cfg` file, an entry for the application name should be added so that driver specific option can be set. The internal application name for the SybaseASE Driver is "SybaseASE". This name cannot be configured or changed at this time.

The DBD::SybaseASE driver does not currently have APIs to support the security and directory services from within the Perl application script. Currently, there is no possibility to enable security options through the DSN.

For more information, see *Configuration Files* in the *Open Client and Open Server Configuration Guide for UNIX.*

Additional Resources

Additional information about using the Perl driver.

- Building, testing and installation of the DBI driver:
  http://dbi.perl.org/
- The Perl DBI user programmable API calls:
  http://search.cpan.org/~timb/DBI-1.616/DBI.pm
- Open Client and Open Server documentation for configuration information:
  Open Client and Open Server Configuration Guide for UNIX > Configuration Files
- Initializing an application so that it executes using a specific language and related cultural conventions from a system configuration perspective:
  Open Client and Open Server Configuration Guide for UNIX > Localization
- Platform related issues for all the Open Client and Open Server products:
  Open Client and Open Server Programmers Supplement for UNIX
- Using the Open Client and Open Server runtime configuration file:
  Open Client Client-Library/C Reference Manual > Using the runtime configuration file > Open Client and Open Server runtime configuration file syntax
- Enabling an application to support multiple languages and cultural conventions:
  Open Client and Open Server International Developers Guide for UNIX > Understanding Internationalization and Localization
- Platform support:
  Software Developer Kit and Open Server Installation Guide for your platform.

Glossary

Glossary of term specific to scripting languages.

- **Client-Library –** part of Open Client, a collection of routines for use in writing client applications. Client-Library is designed to accommodate cursors and other advanced features in the Sybase product line.
- **CPAN –** Comprehensive Perl Archive Network. The Web site that holds a large collection of Perl software and documentation. See *http://www.cpan.org.*
- **CS-Library –** included with both the Open Client and Open Server products, a collection of utility routines that are useful to both Client-Library and Server-Library applications.
- **CT-Library –** (CT-Lib API) is part of the Open Client suite and is required to let an scripting application connect to Adaptive Server.
- **DBD –** database vendor-specific-driver that translates DBI database API calls into a form that is understood by the target database SDK.
- **DBI –** generic core database API that is database-vendor-agnostic and is the current standard for database access in a Perl application. See *http://dbi.perl.org*.
- **Driver –** the collection of Perl and C code that constitutes DBD::SybaseASE.
- **Extension or module –** the Perl language can be extended by modules that are written in Perl or a combination of Perl and C. In this document, extension and module denote the same.
- **Perl directory tree –** is one of:
  - The complete Perl installation that is installed as a binary module when the system is configured and has its operating system installed. A complete Perl installation is sometimes called the system (Perl) tree and owned by a system account (root, admin).
  - A private Perl (directory) tree, which has been built from source by a user other than the system account and is usually installed in a different location than the system Perl tree. This allows for new feature and bug-fix testing without compromising the system tree. A private directory tree is usually owned by the account that built the tree.
- **Perl script –** Perl is a scripting language that is widely used in system and database administration. See *http://www.perl.org*.
- **thread –** a path of execution through Open Server application and library code and the path's associated stack space, state information, and event handlers.
- **Transact-SQL –** an enhanced version of the database language SQL. Applications can use Transact-SQL to communicate with Adaptive Server Enterprise.

Index