Users Guide for Access Services

Enterprise Connect™ Data Access Option for ODBC 15.7

Contents

Conventions

These style and syntax conventions are used in Sybase® documentation.

Style Conventions

Name	Example
Files, directories	`econnect\ServerName\cfg`
Programs, utilities, procedures, commands	the **set** statement
Properties	**Allocate**
Options	**connect**
Code examples, text on screens	`** Prepare the statement`
Variables in command line displays (integer, in this example)	`ClientIdleTimeout=`*integer*
Syntax statements that display options for a command	`sp_columns` *`table_name`* `[,` *`table_owner`*`][,` *`table_qualifier`*`][,` *`column_name`*`]`

Syntax Conventions

Key	Definition
{ }	Curly braces indicate that you must choose at least one of the enclosed options. Do not type the braces when you enter the command.
[ ]	Brackets mean that choosing one or more of the enclosed options is optional. Do not type the brackets when you enter the command.
( )	Parentheses are to be typed as part of the command.
|	The vertical bar means you can select only one of the options shown.
,	The comma means you can choose as many of the options shown as you like, separating your choices with commas that you type as part of the command.

Introduction

When using DB2 ODBC drivers that support the IBM Distributed Relational Database Architecture (DRDA) protocol, ECDA Option for ODBC supports access to DB2 UDB on z/OS, Windows, Linux, and UNIX platforms.

When using Microsoft SQL Server ODBC drivers, ECDA Option for ODBC supports access to Microsoft SQL Server databases running on Windows platforms only.

ECDA Option for ODBC

ECDA Option for ODBC is a Sybase solution that gives client applications ODBC data access.

It combines the functionality of the ECDA Option for ODBC architecture with ODBC to provide dynamic SQL access to target data, as well as the ability to support stored procedures and text and image pointers.

The ECDA Option for ODBC provides access management, copy management, and remote systems management. It comprises:

- The DirectConnect™ server, which provides management and support functions for DirectConnect service libraries.
- An access service library, which accesses data from ODBC-accessible target databases, such as DB2 UDB and Microsoft SQL Server.
- Access services, which contain specific sets of configuration properties relating to the target to be accessed and define how each access service behaves.

Using the IBM Distributed Relational Database Architecture (DRDA) protocol, ECDA Option for ODBC supports access to DB2 UDB on z/OS, Windows, Linux, and UNIX platforms.

For more information about ECDA architecture, see the *Enterprise Connect Data Access Overview Guide*.

ODBC Driver

ECDA Option for ODBC provides basic connectivity to non Sybase datasources, using the ODBC back-end (server-side) driver that you purchase for your target database, such as IBM or Microsoft SQL.

Following the vendor's instructions, you install the ODBC driver on the same server as ECDA Option for ODBC and then configure ECDA Option for ODBC to use that ODBC driver for access to your database.

Note: Verify that your ODBC driver is compatible with Sybase driver manager software.

Because ODBC drivers have varying degrees of functionality, it is important that when working with non-Sybase-provided third-party ODBC drivers, you carefully integrate and test them to be sure they meet your needs.

DirectConnect Server Routing Process

The DirectConnect server routes each client request for an access service to the appropriate access service library.

The routing process can access the service in one of these ways:

- Directly – specify the exact name of the access service. If the access service is defined correctly, the DirectConnect server matches the request with the access service.
- Using service name redirection – map your access service connections to allow client requests to be routed to assigned access services based on user profiles. This feature allows you to centrally manage client access to access services.

See *Enterprise Connect Data Access Mainframe Connect Server Administration Guide > Service Name Redirection*.

ECDA Option for ODBC Configuration Properties

You can configure ECDA Option for ODBC properties on the server level, the access service library level, or an individual access service level.

DirectConnect configuration properties are grouped as:

- Server configuration files – consist of the properties that manage a particular DirectConnect server.
- Access service library configuration files – consist of general library configuration values and configuration sets for all access services associated with a particular access service library.
- Access service configuration properties – define a particular access service and are stored in the access service library configuration file.

When you install a DirectConnect server, the default configurations allow the server to run. For each access service you create within each server, you must provide additional configuration properties that define the connectivity to your target database system.

See the *Enterprise Connect Data Access Mainframe Connect Server Administration Guide* for information on configuring the DirectConnect server.

You can set access services to be enabled at start-up through a configuration setting. If this value is set to no, you must manually enable the access service before it can be used.

You can configure properties using the DirectConnect Manager or a text editor. Sybase recommends that you use DirectConnect Manager for these reasons:

- Changes that you make with a text editor do not take effect until you restart the server. However, most changes that you make with DirectConnect Manager can take effect immediately.
- You can use DirectConnect Manager as a guide to the properties that can be changed, as well as the valid values for each property.

See also

- *DirectConnect Manager* on page 7

Server External Files

The DirectConnect server manages external files that reside in various subdirectories.

See the appropriate installation guide for your database system and platform.

Table 1. Server External Files

File Name	Description
License file	The license file contains licensing information entered by the client for the products and features that are being used. This site-specific file contains descriptions of server nodes that can run the license daemons, various vendor daemons, and licenses for the features and the supported products.
Log file	The log file contains operational information that you can use to correct problems. Although the file is maintained in US English, client messages appear in the client language. The log file resides in the server `log` subdirectory.
Server configuration file	`server.cfg` contains all server configuration information. It resides in the server `cfg` subdirectory.
Access service library files	This dynamically loaded shared library represents each access service library. The DirectConnect server identifies the library by the file name. To install, load, or access a library, verify that the executable file for that library exists in the server *`install_dir`*`/DC-15_0/svclib` subdirectory for UNIX, or the *`install_dir`*`\DC-15_0\svclib` subdirectory for Windows.

File Name	Description
Access service library configuration file	This file contains information for the access service library and all of its access services. Each access service library has a configuration collection. The server defines the file format, but each configuration property is defined by the access service library, regardless of whether the property is managed at the access service library or the access service level. The configuration files reside in the server `cfg` subdirectory. For information on configuring access service library properties, see the appropriate *Access Service Guide* for your database system.
Service name redirection file	This optional file contains all information necessary to redirect incoming requests for access service names to other access services. The file resides in the server `cfg` subdirectory.
Trace file	This file is the only active trace file for the system and it provides debugging information for Sybase Product Support Engineers and Technical Support personnel. You can set it on and off through server configuration. Although the trace file is maintained in US English, client messages appear in the client language. The trace file resides in the `log` subdirectory.

See the *Enterprise Connect Data Access and Mainframe Connect Server Administration Guide* for more information on the service name redirection file and the server configuration file.

See also

- *Access Service Library Configuration* on page 9
- *Logging Properties* on page 39
- *Tracing Properties* on page 51

DirectConnect Manager

DirectConnect Manager graphically represents each DirectConnect object on a tree list or an “icon map,” which is a customizable workspace where you can add or remove objects.

When you add a DirectConnect server to DirectConnect Manager, its server name, access service library, and any access services appear on the tree list or the icon map.

DirectConnect Manager communicates asynchronously with DirectConnect servers, which means you can continue to use DirectConnect Manager while a command is being processed.

You can configure properties using DirectConnect Manager or a text editor. However, Sybase strongly recommends that you use DirectConnect Manager for these reasons:

- Changes that you make with a text editor do not take effect until you restart the server, while changes that you make with DirectConnect Manager can be made to take effect immediately.
- You can use DirectConnect Manager as a guide to the properties that can be changed, as well as the valid values for each property.
- DirectConnect Manager can perform all of its management functions remotely. With DirectConnect Manager, you do not need physical access to the DirectConnect server machine or directory.
- DirectConnect Manager provides management services to multiple servers at the same time, including the ability to copy access service configurations from one server to another.

For more information about DirectConnect Manager features, use the online help menu option.

You can install DirectConnect Manager and its required components from the DirectConnect Client CD.

Note: When you install a DirectConnect product on a Windows or UNIX platform or machine, you must install DirectConnect on a separate platform or machine; doing so allows you to control any ECDA product from any machine.

Internationalization

Internationalization consists of character code set conversion and cultural formatting.

- Code set conversion involves converting the hexadecimal representation of a character from a code set in a target database to a code set in a client application, or the reverse.

- Cultural formatting involves designating decimal separators, monetary signs, date and time separators, and a 3-digit grouping symbol. Cultural formatting in DirectConnect is performed through the use of configuration properties.

Localization

You can localize the messages that are generated by the target database manager and passed to the client without changes, and the messages generated in ECDA Option for ODBC.

- The target database manager can be any application between the DirectConnect server and the target data file, including the ODBC driver.
- ECDA Option for ODBC does not localize database manager messages.

  For information on how to set up localization of such messages, see your database manager and the ODBC driver documentation.

Access Service Library Configuration

Customize the access service library and the individual access services for the ECDA Option for ODBC, which includes DB2 UDB, Microsoft SQL Server, and ODBC-accessible database servers.

Configuration Process

Customize the access service library and the individual access services for the ECDA Option for ODBC.

To create additional access services, configure, and edit existing properties in the access service library configuration file, use either:

- The DirectConnect Manager to modify the access service library configuration file and dynamically change the properties without stopping and starting the server, or,
- A text editor to edit the access service library configuration file that resides on the DirectConnect server. Upon completion, you must stop and restart the server for the changes to take effect.

Note: Sybase recommends that you use the DirectConnect Manager to modify the access service configuration file. See DirectConnect Manager online help.

The access service library uses some configuration information from the DirectConnect server. See the *Enterprise Connect Data Access and Mainframe Connect Server Administration Guide*.

Configuration File Description

Use the DirectConnect Manager or a text editor to modify and save the configuration file named `dcany.cfg`.

See the *ECDA Option for ODBC Installation Guide* for your platform to find the location of the configuration file within the ECDA Option for ODBC directory structure.

Configuration File Format

Use a configuration file to define the access service library and the ODBC driver.

An access service library configuration file consists of:

- A primary section [*Service Library*] that groups access service library properties. The name is hard-coded and cannot be changed.
- Access service sections [*Service Name*], in brackets.
- Subsections {*Subsection Name*}, in braces. Subsections group the properties by type.

- Configuration properties and values.

You can include comments. Enter each comment on a separate line and begin with a semicolon or the “#” symbol in column one.

Configuring Access Service Properties

Configure the access service with a specific set of configuration properties.

1. Open an access service library.
2. Enter site-specific values for all required properties.
3. Enter only the values that differ from the default values for properties that are not required.

   Note: Configuration properties are not case-sensitive.

   Rules:

   - Server properties apply to all access services created for that server.
   - Service library properties apply to the service library.
   - Access service properties apply to specific access services.

Configuration File Templates

The ECDA configuration target database template shows all configuration properties, organized as they appear in the configuration file layout.

In this template, two required properties have entries for ODBC data targets: `ConnectionSpec1` and `EnableAtStartup`.

```
[Service Library]
{Client Interaction}
ODBCDriverManager=
SvclibDescription=

{Logging}
LogSvcLibStatistics=
```

```
[Service Name]
```

```
{ACS Required}
ConnectionSpec1=ODBCDataSource
```

```
{Catalog Stored Procedures}
CSPColumnODBCVersion=
CSPExclusions=
CSPIncludeAlias=
CSPIncludeSynonym=
CSPIncludeSystem=
CSPIncludeTable=
CSPIncludeView=
DatatypeInfo=
```

```
{Client Interaction}
ClientDecimalSeparator=
ClientIdleTimeout=
EnableAtStartup=
MaxResultSize=
MaxRowsReturned=
MaxSvcConnections=
quoted_identifiers=
SendWarningMessages=
ServiceDescription=
StripBinaryZero=
StripString=TextSize=
TextSize=
TransactionMode=
Version=
```

```
{Data Conversion Errors}
CharConvertError=
DateTimeConvertError=
DefaultDate=
DefaultNum=
DefaultTime=
NumConvertError=
```

```
{Datatype Conversion}
BinaryResults=
DateResults=
DateTimeResults=
DecimalResults=
FloatResults=
Int2Results=
Int4results=
RealResults=
TimeResults=
TinyIntResults=
XNLChar=
XNLVarChar=
```

```
{Logging}
LogConnectionStatistics=
LogReceivedSQL=
LogRequestStatistics=
LogServiceStatistics=
LogTargetActivity=
LogTransferStatistics=
LogTransformedSQL=
```

```
{Target Interaction}
Allocate=
DelimitSQLRequests=
DisableROLock=
IsolationLevel=
QuotedStringDelimeter=
ReturnNativeError=
SQLOdbcCursors=
SQLTransformation=
StopCondition=
TargetDBMS=
TargetDecimalSeparator=
{Tracing}
TraceEvents=
TraceInterface=
TraceTarget=
```

```
{Transfer}
BulkCommitCount=
TransferBatch=
TransferBatchSeparator=
TransferErrorAction=
TransferErrorCount=
TransferExpress=
TransferPacketSize=
```

See also

- *Configuration Property Categories* on page 15

Changing Configuration Property Values

Use the DirectConnect Manager or a text editor to change the default access service configuration property values.

1. Using the DirectConnect Manager:

   For instructions on how to use DirectConnect Manager to edit the access service configuration file (`dcany.cfg`), see *DirectConnect Manager online help > Managing Access Services > Modifying Access Service Configuration Properties*.

   Note: Before you can use DirectConnect Manager to update the access service properties, you must have installed DirectConnect Manager as outlined in the installation guide for your platform. Also, you must identify and establish a connection between the DirectConnect server and DirectConnect Manager. See *Connecting DirectConnect Manager to a DirectConnect Server* in the DirectConnect Manager online help.

   For additional information, use the verbose mode that is available with DirectConnect Manager.
2. Using the text editor:
   a) Locate and open the access service library configuration file `dcany.cfg`.
   b) Update the service library configuration properties as needed.
   c) Open the access service file `server.cfg` and change the access service property values as needed.

      List each property and value under the appropriate subsection. If the subsection is not shown, you must add it.
   d) Save the file.
   e) Stop the server, then restart for the changes to take effect.

Creating Additional Services

Use the DirectConnect Manager or the text editor to create additional access services.

Rules for access service names:

- Service names must be unique but not case-sensitive.
- Service names cannot exceed 31 characters for Windows or UNIX operating systems.
- The initial character must be an alphabetic character (a–z, A–Z); subsequent characters can be alphabetic characters, numbers, or the underscore (_) character.

1. Using DirectConnect Manager:

   For instructions on how to use DirectConnect Manager to create a service, see *DirectConnect Manager online help > Managing Access Services > Creating a New Service* or *Copying a Service*.
2. Using the text editor:
   a) Open the Access Service Library configuration file, `dcany.cfg`.
   b) Create a section for each new service and then add:
      - The service name, in brackets.
      - Required properties below the service name, grouped in the {ACS Required} subsection.
      - Property value overrides listed below the service name, grouped by subsection.

   c) Save the file.
   d) Stop the server, then restart for the changes to take effect.
   e) Enable a client machine to connect to a new access service.
   f) Enter the access service name in the `sql.ini` configuration file on Windows NT machines or the `interfaces` file on UNIX client machines.

      See the *Enterprise Connect Data Access Installation Guide* for your platform for instructions about editing the `sql.ini` or `interfaces` file.

      Note: If you choose to use service name redirection, create an assigned service name entry in the service name redirection file.

      See *Enterprise Connect Data Access and Mainframe Connect Server Administration Guide > Service Name Redirection*.

See also

- *Configuration Property Categories* on page 15

Code Page Translation for ODBC-Based Products

ODBC drivers incorporate code page translation within their normal data processing.

The ECDA Option for ODBC uses this functionality to simplify code page translation.

For ODBC-based products, code page translation can take place between the DirectConnect server and the target database or between the client and the DirectConnect server.

DirectConnect Server and the Target Database

The ODBC driver uses the server-platform-configured code page value as its client code page.

Depending on the platform, you can find the code page value:

- On Windows – use the Windows Registry ACP value. To locate the value, use the Registry Editor called `regedit` to navigate through the registry tree to `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\NIs\CodePage`, which represents the platform's ODBC code page. On the right panel, scroll to the ACP value.
- On Linux and UNIX – use the **locale** command to determine the operating system, the platform, and ODBC code page value.

Upon connection to the target, the ODBC driver queries the target database for its code page and compares the value to the server-platform-configured code page:

- If the values are not equal, the ODBC driver translates from the server-platform code page to the target code page.
- If the values are equal, the ODBC driver does not perform any translation. As an Open Server™ API, the ECDA Option for ODBC relies on Open Server for datatype conversion.

Client and the DirectConnect Server

For proper code page translation, the ECDA code page identified by Open Server must match the server-platform-configured code page value.

The default platform in the `locales.dat` configuration file identifies the Open Server code page.

The DirectConnect server configuration property, called `OSCodeSetConvert` determines whether the ECDA Option for ODBC allows Open Server to perform code page translation between the client and the DirectConnect server. Set the `OSCodeSetConvert` to:

- Yes to indicate that the DirectConnect server performs code page translation.
- No to indicate that the DirectConnect server does not perform any code page translation.

Configuration Property Categories

Configuration properties are classified by category; property, syntax, range, default values, acceptable values, and comments.

See also

- *Service Library Properties* on page 15
- *ACS Required Property* on page 17
- *Catalog Stored Procedures Properties* on page 18
- *Client Interaction Properties* on page 22
- *Data Conversion Error Properties* on page 30
- *Datatype Conversion Properties* on page 32
- *Logging Properties* on page 39
- *Target Interaction Properties* on page 46
- *Tracing Properties* on page 51
- *Transfer Properties* on page 53
- *Configuration Properties Quick Reference* on page 159

Service Library Properties

A list of service library properties and all of their services.

Syntax:

```
{Client Interaction}
ODBCDriverManager=
SvclibDescription=

{Logging}
LogSvcLibStatistics=
```

ODBCDriverManager {Client Interaction}

(UNIX) Specifies the full path name to the `ODBCDriverManager` access service library.

Syntax

```
ODBCDriverManager = ODBC Driver Manager library
```

where `ODBC Driver Manager library` is the full path name to the ODBC driver manager library.

Note: Enter the library name with the full path; otherwise, the program must search the entire library for the correct driver manager library.

Range
String value up to 255 characters.

Values
Default values are the names the unixODBC driver manager installed with the product (found by the library path):

- `libodbc.so.1.0.0` (Solaris, Linux)
- `libobdc.sl.1.0.` (HP)
- `libodbc.so.1` (AIX)

Comment
The driver manager library uses a generic name, `libdodbc.lib_ext`. The `ODBCDriverManager` configuration property defaults to the unixODBC driver manager version name, `libodbc.lib_ext.x`, which makes it possible to place the unixODBC driver manager library in the `DC-15_0/lib` directory already located in the library path.

SvclibDescription {Client Interaction}
Describes the access service library.

Syntax

```
SvclibDescription=char
```

This property applies to a description of the access service library.

Range
0–255 characters

Default
None

Comment
This property allows you to place descriptive information about the access service library in the configuration file.

LogSvcLibStatistics {Logging}
Specifies how often the access service library records accumulated statistics about connection requests to all access services associated with this access service library during the reporting interval.

Syntax

```
LogSvclibStatistics=integer
```

This is an access service library property that applies to the access service library as a whole.

Range
0–2147483646

Default
`0` (zero)

Values

- `integer` is a number of seconds.
- A value of `0` specifies that the access service library does not record statistics in the server log file.

Comments

- Use this property to:
  - Monitor load on the entire access service library.
  - Monitor load on the target database through the DirectConnect server.
- If you enable both `LogSvclibStatistics` (service library level) and `LogServiceStatistics` (service level) properties, Sybase recommends that you set the `LogSvclibStatistics` property to the same property value as the `LogServiceStatistics` or a multiple thereof. If you use DirectConnect Manager to change these two property values, set the `LogSvclibStatistics` property last for better synchronization.
- If the `LogSvclibStatistics` property value is greater than `0` (zero), the ECDA Option for ODBC records totals of the statistics for all access services in the access service library.

See also

- *LogServiceStatistics Data and Log Service Statistics* on page 43

ACS Required Property

Require site-specific values for ODBC data targets.

Be sure to enter this value for your installation.

The subsection heading and the name of the property must appear in the access service library configuration file as:

```
{ACS Required}
ConnectionSpec1=
```

ConnectionSpec1

Specifies an ODBC data source name (DSN) defined in the `ODBC system information` file.

- On Windows – the datasources are defined using the ODBC Administrator.
- On UNIX – the datasources are defined in an `odbc.ini` file.

Syntax

```
ConnectionSpec1=char
```

Range

1–255 characters

Default

None

Value

`char` is a valid datasource name configured in the `odbc.ini` file.

Catalog Stored Procedures Properties

Controls the information of an access service returned from the catalog stored procedures (CSPs).

Note: Many of the properties in this group are not supported, nor do they affect an access service. These properties are available for compatibility purposes only.

The subsection heading and a list of the properties must appear in the access service library configuration file as:

```
{Catalog Stored Procedures}

CSPColumnODBCVersion=
CSPExclusions=
CSPIncludeAlias=
CSPIncludeSynonym=
CSPIncludeSystem=
CSPIncludeTable=
CSPIncludeView=
DatatypeInfo=
```

CSPColumnODBCVersion

Specifies the ODBC version that catalog stored procedures results conform to.

Syntax

```
CSPColumnODBC Version = [ 2 | 3 ]
```

Default
`3`

Values

- `2` specifies Adaptive Server® Enterprise/Component Integration Services (CIS) version 12.0.
- `3` specifies Adaptive Server/CIS version 12.5 and later.

Comment
This property affects interoperability with Adaptive Server/CIS. Change the value to 2 if you see:

```
Error 11209 - 'column type mismatch in remote object' when executing
create existing table command to MSSQL server 2000SP 2.
```

CSPExclusions

Specifies an access service to limit access to information normally returned from **sp_tables** upon authorization.

Syntax

```
CSPExclusions=[ none | user | nonauth |nonauthpublic ]
```

Default
`user`

Values

- `none` specifies no exclusions, based on authorization to information.
- `user` specifies exclusions, based on specific user authorization to information.
- `nonauth` specifies that a user must be granted user or group authorization to access information.
- `nonauthpublic` specifies that a user must be granted user authorization, or that `PUBLIC` is granted some authorization.

Comment
This property is not supported and does not affect the ECDA Option for ODBC access service. It is available for compatibility purposes only.

CSPIncludeAlias

Specifies the access service to return information about aliases from **sp_tables**.

Syntax

```
CSPIncludeAlias=[no | yes]
```

Default
no

Values

- no specifies that the access service does not return alias information.
- yes specifies that the access service returns alias information.

Comment
This property is not supported and does not affect the ECDA Option for ODBC access service. It is available for compatibility purposes only.

CSPIncludeSynonym
Specifies the access service to return information about synonyms from **sp_tables**.

Syntax

```
CSPIncludeSynonym=[no | yes]
```

Default
no

Values

- no specifies the access service not to return synonym information.
- yes specifies the access service to return synonym information.

CSPIncludeSystem
Specifies the access service to return information about system tables from **sp_tables**.

Syntax

```
CSPIncludeSystem=[no | yes]
```

Default
no

Values

- no specifies the access service not to return system table information.
- yes specifies the access service to return system table information.

Comment
The user issuing **sp_tables** must be authorized to query these tables.

CSPIncludeTable

Specifies the access service to return information about tables from **sp_tables**.

Syntax

```
CSPIncludeTable=[yes | no]
```

Default

yes

Values

- yes specifies the access service to return table information.
- no specifies the access service not to return table information.

CSPIncludeView

Specifies the access service to return information about views from **sp_tables**.

Syntax

```
CSPIncludeView=[yes | no]
```

Default

yes

Values

- yes specifies the access service to return view information.
- no specifies the access service not to return view information.

DatatypeInfo

Specifies the type of datatype information returned from **sp_datatype_info**.

Syntax

```
DatatypeInfo=[transact | target]
```

Default

`transact`

Values

- `transact` specifies **sp_datatype_info** to return the Transact-SQL™ datatypes that map to the ODBC datatypes supported by the target.
- `target` specifies **sp_datatype_info** to return target datatype names.

Client Interaction Properties

Controls how an access service library or an access service interacts with client applications.

The subsection heading and a list of the properties must appear in the access service library configuration file as:

```
{Client Interaction}
ClientDecimalSeparator=
ClientIdleTimeout=
EnableAtStartup=
MaxResultSize=
MaxRowsReturned=
MaxSvcConnections=
quoted_identifier=
SendWarningMessages=
ServiceDescription=
StripBinaryZero=
StripString=
TextSize=
TransactionMode=
Version=
```

ClientDecimalSeparator

Specifies the character the client application uses to separate decimal numbers for presentation purposes.

Syntax

```
ClientDecimalSeparator=char
```

Default

. (period)

Values

- `char` indicates the character used by the client application as the decimal delimiter.
- A period (.) indicates that the client application uses a period as the decimal delimiter.

Comment

If your client application uses a different character as a decimal delimiter, verify that this application connects to an access service configuration set that uses the same client decimal delimiter character.

ClientIdleTimeout

Specifies number of minutes a client connection can remain inactive before an access service terminates the connection.

Syntax

```
ClientIdleTimeout=integer
```

Range
0–1024

Default
`0`

Values

- `integer` is the number of minutes a client connection can remain inactive before a conection is terminated.
- `0` indicates that an idle connection is never terminated.

Comments

- A connection is idle when:
  - A client application connects but does not issue a command.
  - A command completes processing, but the client does not issue a new command.
  - A large result set is returned from a SQL request, and the result screen paused for the specified timeout period.
- The access service checks client activity once per minute. Therefore, a `ClientIdleTimeout` value of *n* might allow a client to remain active for nearly *n* + 1 minutes.

EnableAtStartup

Specifies whether this access service starts when the DirectConnect server starts.

Syntax

```
EnableAtStartup=[no | yes]
```

Default
no

Values

- no means the access service does not start when the server starts.
- yes means the access service starts when the server starts.

Comment
If you are not using DirectConnect Manager, set this property to yes.

MaxResultSize

Specifies the maximum number of bytes an access service returns to the client application in a result set.

Syntax

```
MaxResultSize=integer
```

Range

0–unlimited

Default

`unlimited`

Values

- `integer` is a number of bytes.
- A value of `0` indicates that the result size is an unlimited value.

Comments

- The `MaxResultSize` value is approximate in that the access service checks at the end of each row to see if the value is exceeded.
- If the value is exceeded, the access service:
  - Sends the entire row to the client application (not a partial row)
  - Does not send any of the remaining rows in the result set

MaxRowsReturned

Specifies the maximum number of rows an access service returns to the client application in a result set.

Syntax

```
MaxRowsReturned=integer
```

Range

0–unlimited

Default

`unlimited`

Values

- `integer` is a number of rows.
- A value of `0` indicates that the number of rows returned is an unlimited value.

Comment

If the `MaxRowsReturned` value is exceeded, the value is returned to the client as an error.

MaxSvcConnections

Specifies the maximum number of client connections that can log in to the access service at one time.

Syntax

```
MaxSvcConnections=integer
```

Range

1–*n*, where *n* is the maximum number of client connections allowed for the access service.

Default

`MaxConnections` property value of the DirectConnect server

Value

`integer` is a number of client connections.

Comments

- If you set the `Allocate` property value to `request`, this allows more actual clients to be supported because unused clients are not counted as connections.
- The DirectConnect server `MaxConnections` property determines the maximum number of client connections.

  See **MaxConnections** in the *Enterprise Connect Data Access and Mainframe Connect Server Administration Guide*.

quoted_identifier

Specifies whether to enable or disable delimited identifiers.

Delimited identifiers are object names enclosed in double quotes. You can use them to avoid certain restrictions on object names. You can delimit only table, view, and column names.

Delimited identifiers can:

- Be reserved words
- Begin with non alphabetic characters
- Include characters that ordinarily are not allowed

Syntax

```
quoted_identifer = [on | off]
```

Default

off

Values

- on means that a quoted string used as an identifier is recognized.
- off means that a quoted string is not recognized as an identifier.

Comments

- Delimited identifiers follow the conventions for identifiers for DB2.
- Before you create or reference a delimited identifier, issue:

```
set quoted_identifier on
```

Example 1

Each time you use the delimited identifier in a statement, you must enclose it in double quotes.

```
create table "1one" (col 1 char(3))
 create table "include spaces" (col1 int)
```

or:

```
create table "grant" ("add" int)
 insert into "grant" ("add") values (3)
```

When the `quoted_identifier` configuration property is set to on, use single quotes around character or date strings. Delimiting strings with double quotes causes Adaptive Server to treat them as identifiers.

Example 2

Inserts a character string into `col1` of `1one` when the quoted identifier "1one" is set to on:

```
insert into "1one"(col1) values ('abc')
```

Example 3

Inserts a single quote into a column, use two consecutive single quotation marks and values "a'b" into `col1`

```
insert "1one"(col1) values('a"b')
```

SendWarningMessages

Specifies whether an access service returns warning messages to the client application.

Syntax

```
SendWarningMessages=[no | yes]
```

Default

no

Values

- no specifies not to return warning messages to the client application.
- yes specifies to return warning messages to the client application.

ServiceDescription

Allows you to place descriptive information about each access service in the configuration file.

Syntax

```
ServiceDescription=char
```

Range
0–255 characters

Default
None

Value
`char` is a user-defined character string.

StripBinaryZero

Specifies whether binary zeros are removed from the incoming language commands.

Syntax

```
StripBinaryZero=[yes | no]
```

Default
no

Values

- no specifies that binary zeros are not removed from the incoming language commands.
- yes specifies that binary zeros are removed from the incoming language commands.

StripString

Removes the configured character string from the beginning of an incoming language event.

Syntax

```
StripString= character string
```

Default
blank

Values

- `character string` is the configured string that is to be removed from the beginning of the incoming language string. It must match the incoming language string exactly, including case and spaces.

TextSize

Specifies the maximum number of bytes in character columns an access service returns to the client application.

Syntax

```
TextSize= integer
```

Range

1 - 2147483647

Default

2147483647

Values

- `integer` is a number of bytes.
- A value of `0` defaults to `2147483647`.

Comments

- Data exceeding the `TextSize` length is truncated without a warning message.
- Character data longer than `XNLCHAR` or `XNLVARCHAR` bytes is returned as CS_TEXT.
- For ECDA Option for ODBC for Microsoft SQL Server targets, how the data is queried determines the text and image results processing. Queries with a select list of a single `text` or `image` column results in data streaming. If data is streamed, a maximum of 2,147,483,647 bytes may be returned per `text` and `image` value. Queries with a select list containing more than one column results in bound data. All bound data, including text and image is limited to 32,767 bytes. The **TextSize** configuration property applies to both streaming and bound character data.

Note: ECDA Option for ODBC supports text data manipulation using text pointers only for a Microsoft SQL Server datasource.

TransactionMode

Specifies whether the access service or the client application manages **commit** and **rollback** statements.

Syntax

```
TransactionMode=[short | long]
```

Default

`short`

Values

- `long` gives commitment control to the client application.
- `short` issues a **commit** or a **rollback** after each request.

Comments

- The access service holds open the connection to the datasource until the client application issues a **commit** or **rollback**, or until the `ClientIdleTimeout` value is exceeded.
- If `ClientIdleTimeout` is exceeded, the transaction rolls back.

Version

Specifies the version string for ECDA Option for ODBC.

Syntax

```
Version= versionstring
```

Default

The access service default version string

Value

`versionstring` is the version string that is reported to client applications.

Comments

- This property allows you to customize a version string for client applications that rely on a string that is different from the access service default.
- If you customize an alternate version string:
  - The string format cannot contain embedded new lines.
  - You can insert a space after the equal sign for readability in the configuration file. However, when an access service sends the version string to the client application, it removes any leading and trailing white space.
- You can obtain the access service default version string by issuing **sp_helpserver**.

See also

- *sp_helpserver* on page 149

Data Conversion Error Properties

Controls the action an access service takes when it encounters data conversion errors.

The subsection heading and a list of the properties must appear in the access service library configuration file as:

```
{Data Conversion Errors}
CharConvertError=
DateTimeConvertError=
DefaultDate=
DefaultNum=
DefaultTime=
NumConvertError=
```

CharConvertError

Specifies the action an access service takes when it encounters a results column that is too long for the target column.

Syntax

```
CharConvertError=[reject | truncate]
```

Default

`reject`

Values

- `reject` rejects the row containing the error and issues a warning message.
- `truncate` inserts data to the length of the target column, truncates the remaining data, and issues a warning message.

DateTimeConvertError

Specifies the action an access service takes when it encounters rows with date, time, or `datetime` data values that are out of range for the target datatype.

Syntax

```
DateTimeConvertError=[reject | null | default]
```

Default

`reject`

Values

- `reject` rejects the row containing the error and issues a warning message.

- `null` inserts a NULL into the column and issues a warning message.
- `default` inserts the default date and time values, as configured in the `DefaultDate` and `DefaultTime` properties, into the column, and issues a warning message.

DefaultDate

Specifies the value an access service inserts into columns with date conversion errors when `DateTimeConvertError` is set to `default`.

Syntax

```
DefaultDate=yyyy-mm-dd
```

Default

`1900-01-01`

Value

`yyyy-mm-dd` is the default, where:

- `yyyy` is the year.
- `mm` is the month.
- `dd` is the day.

DefaultNum

Specifies the value an access service inserts into columns with numeric conversion errors when `NumConvertError` is set to `default`.

Syntax

```
DefaultNum=integer
```

Default

`0`

Values

`integer` is a valid number that replaces the value that caused the conversion error.

DefaultTime

Specifies the value that is inserted into columns with time conversion errors when `DateTimeConvertError` is set to `default`.

Syntax

```
DefaultTime=hh.mm.ss
```

Default

`00.00.00`

Value

`hh.mm.ss` is the default, where:

- `hh` is the hour in 24-hour clock time.
- `mm` is the minute.
- `ss` is the second.
- A period is used as the delimiter.

NumConvertError

Specifies the action an access service takes when it encounters rows with numeric data values that are out of range for the target datatype.

Syntax

```
NumConvertError=[reject | null | default]
```

Default

`reject`

Values

- `reject` rejects the row containing the error and issues a warning message.
- `null` inserts a NULL into the column and issues a warning message.
- `default` inserts the default numeric value configured in the `DefaultNum` property into the column and issues a warning message.

Datatype Conversion Properties

Controls how an access service converts target database datatypes to Open Client™ and Open Server datatypes before sending the data to the client application.

The subsection heading and a list of the properties must appear in the access service library configuration file as:

```
{Datatype Conversion}
BinaryResults=
DateResults=
DateTimeResults=
DecimalResults=
FloatResults=
Int2Results=
Int4results=
RealResults=
TimeResults=
TinyIntResults=
XNLChar=
XNLVarChar=
```

To provide portability across the DBMS, the names of the configuration properties refer to generic datatypes. The description includes specific target database datatypes to which these generic datatypes correspond.

Note: Datatype conversion properties control conversion of outgoing data from the DBMS. These properties do not control conversion of incoming data from client applications.

BinaryResults

Specifies the Open Server datatype to which returned binary results are converted.

Syntax

```
BinaryResults=[binary | char]
```

Default

`binary`

Values

- `binary` indicates that results of 255 bytes or less are returned as CS_BINARY. Those with 256 bytes or more are returned as CS_IMAGE.
- `char` indicates that results of 255 bytes or less are returned as CS_CHAR. Those with 256 bytes or more are returned as CS_TEXT.

Comments

If the `BinaryResults` configuration property is set to `char`, the access service changes the binary string to a character string and converts the character string to the client code set.

DateResults

Specifies the Open Client and Open Server datatype to which an access service converts to date results.

Syntax

```
DateResults=[datetime | datetime4 | char_iso | char_usa |
char_eur | char_jis | char_odbc]
```

Default

`datetime`

Values

- `datetime` returns an 8-byte `datetime` datatype with a range of legal values from January 1, 1753 to December 31, 2079, and a precision of 1/300th of a second (3.33 milliseconds).

- `datetime4` returns a 4-byte `datetime` datatype with a range of legal values from January 1, 1900 to June 6, 2079, and a precision of one minute.
- `char_iso` returns character data in the format `yyyy-mm-dd`.
- `char_usa` returns character data in the format `mm/dd/yyyy`.
- `char_eur`returns character data in the format `dd.mm.yyyy`.
- `char_jis`returns character data in the format `yyyy-mm-dd`.
- `char_odbc` returns character data in the format `yyyy-mm-dd`.

Comments

If the DATE value is outside the range of the Adaptive Server DATETIME, the `DateTimeConvertError` property determines the desired datatype conversion.

DateTimeResults

Specifies the Open Client and Open Server datatype to which an access service converts to ODBC timestamp results.

Syntax

```
DateTimeResults=[datetime | datetime4 | char_iso | char_usa |
char_eur | char_jis | char_odbc]
```

Default

`datetime`

Values

- `datetime` returns an 8-byte `datetime` datatype with a range of legal values from January 1, 1753 to December 31, 2079, and a precision of 1/300th of a second (3.33 milliseconds).
- `datetime4` returns a 4-byte `datetime` datatype with a range of legal values from January 1, 1900 to June 6, 2079, and a precision of one minute.
- `char_iso` returns character data in the format `yyyy-mm-dd-hh.mm.ss.nnnnnn`.
- `char_usa` returns character data in the format `mm/dd/yyyy hh:mm` AM or PM.
- `char_eur` returns character data in the format `dd.mm.yyyy hh.mm.ss`.
- `char_jis` returns character data in the format `yyyy-mm-dd hh:mm:ss`.
- `char_odbc` returns character data in the format `yyyy-mm-dd hh:mm:ss.nnnnnn`.

Comments

- Use `char_iso` to retain the most precision.

- You can convert ODBC TIMESTAMP to one of the character formats to retain more precision. CS_DATETIME has less precision (1/300ths of a second) than ODBC TIMESTAMP, which can have a precision of up to six fractional places.
- A string representation of an ODBC TIMESTAMP starts with a digit and has a length of at least 16 characters. The complete string representation of an ODBC TIMESTAMP has this form: `yyyy-mm-dd-hh:mm:ss.nnnnnn`.

  Trailing blanks can be included.
- You can omit leading zeros from the month, day, and hour part of the ODBC TIMESTAMP. Also, you can truncate microseconds or omit them entirely. If you choose to omit any digit of the microseconds portion, an implicit specification of 0 is assumed.

DecimalResults

Specifies the Open Client and Open Server datatype to which an access service converts to decimal results.

Syntax

```
DecimalResults=[autoconvert | int | float | real | char | money |
money4 | bcd]
```

Default

`autoconvert`

Values

- `autoconvert` allows the access service to choose the appropriate datatype to return according to following conversion scheme:
  - If scale = 0 and precision is less than or equal to 9, the access service returns CS_INT.
  - If scale is less than or equal to 4, and precision minus scale is less than or equal to 19, the access service returns CS_MONEY.
  - If scale is greater than 2, and precision minus scale is greater than 14, the access service returns CS_FLOAT.
- `int` returns a 4-byte integer type.
- `float` returns an 8-byte float type.
- `real` returns a 4-byte float type.
- `char` returns a character type. However, decimal points are not aligned consistently with those in the ODBC DECIMAL columns.
- `money` returns an 8-byte money type.
- `money4` returns a 4-byte money type.
- `bcd` is valid only if you have columns described in binary coded decimal (BCD) format. The access service returns BCD columns as CS_BINARY or CS_VARBINARY with this format:
  - If precision is even, the first nibble is 0.

- Intervening digits are represented in BCD format with one nibble per digit.
- The final nibble indicates the sign: C is positive and D is negative.
- No indication of decimal position is given. The client application is responsible for determining decimal position.

Comment
In some ODBC datasources, the ODBC decimal type is not supported. In such cases, it is not supported as a Sybase datatype.

FloatResults
Specifies the Open Client and Open Server datatype to which an access service converts to float results.

Syntax

```
FloatResults=[float | real | char]
```

Default
`float`

Values

- `float` returns an 8-byte float type.
- `real` returns a 4-byte float type.
- `char` returns a character type.

Int2Results
Specifies the Open Client and Open Server datatype to which an access service converts to small integer results.

Syntax

```
Int2Results=[smallint | char]
```

Default
`smallint`

Values

- `smallint` returns a 2-byte integer type.
- `char` returns a character type.

Int4Results

Specifies the Open Client and Open Server datatype to which an access service converts to integer results.

Syntax

```
Int4Results=[int | char]
```

Default

`int`

Values

- `int` returns a 4-byte integer type.
- `char` returns a character type.

RealResults

Specifies the Open Client and Open Server datatype to which an access service converts to real results.

Syntax

```
RealResults=[float | real | char]
```

Default

`real`

Values

- `float` returns an 8-byte float type.
- `real` returns a 4-byte float type.
- `char` returns a character type.

Note: ECDA Option for ODBC ignores the `char` setting.

TimeResults

Specifies the Open Client and Open Server datatype to which an access service converts to time results.

Syntax

```
TimeResults=[ datetime | datetime4 | char_iso | char_usa |
char_eur | char_jis | char_odbc ]
```

Default

`datetime`

Values

- `datetime` returns an 8-byte `datetime` datatype with a range of legal values from January 1, 1753 to December 31, 9999, and a precision of 1/300th of a second (3.33 milliseconds).
- `datetime4` returns a 4-byte `datetime` datatype with a range of legal values from January 1, 1900 to June 6, 2079, and a precision of one minute.
- `char_iso` returns character data in the format `hh.mm.ss`.
- `char_usa` returns character data in the format `hh:mm` AM or PM.
- `char_eur` returns character data in the format `hh.mm.ss`.
- `char_jis` returns character data in the format `hh:mm:ss`.
- `char_odbc` returns character data in the format `hh:mm:ss.nnnnnn`.

TinyIntResults

Specifies the Open Client and Open Server datatype to which an access service converts to `TinyIntResults`.

Syntax

```
TinyIntResults=[ smallint, tinyint ]
```

Default

`smallint`

Values

- `smallint` returns an 2-byte integer.
- `tinyint` returns a 1-byte integer.

XNLChar

Specifies the maximum size of both `char` and `binary` results. If the maximum size is exceeded, the datatype is promoted to `text` and `image`, respectively.

Syntax

```
XNLChar= integer
```

Default

256

Values

`integer` is a valid number between 256 and 2147483647 (two gigabytes).

Comments
Sybase recommends that this value matches the maximum size of the `char` and `binary` datatypes of the back-end database. It is common for this limit to be the same for the `char` and `binary` datatypes.

XNLVarChar

Specifies the maximum size of both `varchar` and `varbinary` results. If the maximum size is exceeded, the datatype is promoted to `text` and `image`, respectively.

Syntax

```
XNLVarChar=integer
```

Default
256

Values
`integer` is a valid number between 256 and 2147483647.

Comments
Sybase recommends that the value matches the maximum size of the `varchar` and `varbinary` datatypes of the back-end database. It is common for this limit to be the same for the `varchar` and `varbinary` datatypes.

Logging Properties

Controls whether access service data is recorded in the DirectConnect server log file.

See the *Enterprise Connect Data Access and Mainframe Connect Server Administration Guide*.

The subsection heading and a list of the properties must appear in the access service library configuration file as:

```
{Logging}
LogConnectionStatistics=
LogReceivedSQL=
LogRequestStatistics=
LogServiceStatistics=
LogTargetActivity=
LogTransferStatistics=
LogTransformedSQL=
```

Logging Statistics Properties

The log statistics properties record similar types of data.

You can use these properties independently or in combination to record statistics.

Property Name	Description
`LogConnectionStatistics`	Records accumulated statistics about requests made by each client connection. Statistics are recorded when the client disconnects.
`LogRequestStatistics`	Records statistics about individual SQL requests.
`LogServiceStatistics`	Records accumulated statistics about requests made by all connections to this access service.
`LogTransferStatistics`	Records statistics about individual transfer requests.

LogConnectionStatistics

Specifies whether the access service records accumulated statistics about each connection.

Syntax

```
LogConnectionStatistics=[no | yes]
```

Default

no

Values

- no means connection statistics are not recorded.
- yes means connection statistics are recorded.

Comments

- Connection statistics are recorded in the server log file when the client disconnects from the access service.
- You can use this property to monitor the activity of particular clients.

LogConnectionStatistics Data and Log Service Statistics

The statistics data recorded when `LogConnectionStatistics` is set to on.

Log Field Data	Description
Buffer size	The number of bytes of SQL (after transformation) in the SQL request sent to the database.
Service processing time	The elapsed time, in seconds, from the time the DB2 access service receives a SQL statement until it sends the statement to the database.

Log Field Data	Description
DBMS processing time	The elapsed time, in seconds, from the time the DB2 access service sends the SQL statement to the database until the database returns the first result row to the client application.
Time to receive rows	The elapsed time, in seconds, from the time the database sends the first result row to the client application until the client application receives the last result row (in the format `ss.nnn`).
Total processing time	The elapsed time, in seconds, from the time the DB2 access service receives the SQL statement until the client application receives the last result row.
Number of rows returned	The number of result rows returned to the client application.
Number of conversion errors	The number of result rows that contain data conversion errors.
Number of kilobytes returned	The number of kilobytes returned to the client application (to a scale of 3 in the format `n.nnn`).
Number of events	The total number of events that occurred during the time period.

LogReceivedSQL

Specifies whether the access service records SQL statements as the statements are received from client applications.

Syntax

```
LogReceivedSQL=[no|yes]
```

Default

no

Values

- no means SQL statements are not recorded as they are received.
- yes means SQL statements are recorded as they are received.

LogRequestStatistics

Specifies whether the access service records statistics about each SQL request.

Syntax

```
LogRequestStatistics=[no|yes]
```

Default

no

Values

- no means statistics about each request are not recorded.
- yes means statistics about each request are recorded.

Comments

- Use this property to:
  - Aid performance tuning on a specific type of request.
  - Analyze data throughput.
  - Monitor the types of requests by users.

LogRequestStatistics Data and Log Service Statistics

The statistics recorded when `LogRequestStatistics` is set to on.

Log Field Data	Description
Buffer size	The number of bytes of SQL (after transformation) in the SQL request sent to the database.
Service processing time	The elapsed time, in seconds, from the time the DB2 access service receives a SQL statement until it sends the statement to the database
DBMS processing time	The elapsed time, in seconds, from the time the DB2 access service sends the SQL statement to the database until the database returns the first result row to the client application.
Time to receive rows	The elapsed time, in seconds, from the time the database sends the first result row to the client application until the client application receives the last result row (in the format `ss.nnn`).
Total processing time	The elapsed time, in seconds, from the time the DB2 access service receives the SQL statement until the client application receives the last result row.
Number of rows returned	The number of result rows returned to the client application.
Number of conversion errors	The number of result rows that contain data conversion errors.
Number of kilobytes returned	The number of kilobytes returned to the client application (to a scale of 3 in the format `n.nnn`).
Command name	SQL keyword for request, such as **select**, **disconnect**, and **insert**.

LogServiceStatistics

Specifies how often the access service records accumulated statistics.

Syntax

```
LogServiceStatistics=integer
```

Range

0–2147483646

Default

0

Values

- `integer` is a number of seconds.
- A value of `0` specifies that the access service does not record access service statistics.

Comment

Use this property to:

- Monitor the load on a particular access service.
- Monitor usage of a particular access service.

LogServiceStatistics Data and Log Service Statistics

The statistics recorded when the `LogServiceStatistics` property is set to on.

Log Field Data	Description
Buffer size	The number of bytes of SQL (after transformation) in the SQL request sent to the database.
Service processing time	The elapsed time, in seconds, from the time the DB2 access service receives a SQL statement until it sends the statement to the database.
DBMS processing time	The elapsed time, in seconds, from the time the DB2 access service sends the SQL statement to the database until the database returns the first result row to the client application.
Time to receive rows	The elapsed time, in seconds, from the time the database sends the first result row to the client application until the client application receives the last result row (in the format `ss.nnn`).
Total processing time	The elapsed time, in seconds, from the time the DB2 access service receives the SQL statement until the client application receives the last result row.

Log Field Data	Description
Number of rows returned	The number of result rows returned to the client application.
Number of conversion errors	The number of result rows that contain data conversion errors.
Number of kilobytes returned	The number of kilobytes returned to the client application (to a scale of 3 in the format `n.nnn`).
Number of events	The total number of events that occurred during the time period.
Number of successful connections	The total number of successful client connections that occurred during the time period.
Maximum number of client connections	The greatest number of client connections at any given time during the time period.

LogTargetActivity

Specifies whether the access service records interactions between the access service and the target database. A file is created for each connection.

Syntax

```
LogTargetActivity=[no|yes]
```

Default

no

Values

- no means the access service does not record these interactions with the target database.
- yes means the access service records these interactions with the target database:
  - Log in
  - Log out
  - Requests sent
  - Results received

LogTransferStatistics

Specifies whether the access service records statistics about transfers.

Syntax

```
LogTransferStatistics=[no|yes]
```

Default

no

Values

- no means transfer statistics are not recorded.
- yes means transfer statistics are recorded.

LogTransferStatistics Data
The access service fields that record transfer statistics.

Log Field Data	Description
Buffer size	The number of bytes in the **transfer** statement.
Transfer setup time	The elapsed time, in seconds, from the time the access service receives a **transfer** statement until it issues the source **select** against the secondary database. This includes connection time to the secondary database and the time the access service takes to obtain datatype information for target columns.
Source DBMS processing time	The elapsed time, in seconds, from the time the access service issues the **select** part of the **transfer** statement against the source database until it receives the first result row from the source of the transfer.
Time to transfer rows	The elapsed time, in seconds, from the time the source database returns the first row to the access service until the last row is inserted into the target of the transfer.
Total processing time	The elapsed time, in seconds, from the time the access service receives the **transfer** statement until the last row of the **transfer** data is inserted into the target database.
Number of rows transferred	The number of rows transferred.
Number of conversion errors	The number of rows that contain data conversion errors.
Number of kilobytes returned	The total number of kilobytes transferred (to a scale of 3 in the format `n.nnn`).
Command type	The first token (command) from the SQL buffer. This is always a **transfer**.

LogTransformedSQL
Specifies whether the access service records SQL as it is transformed and sent to the target database.

Syntax

```
LogTransformedSQL=[no|yes]
```

Default
no

Values

- no means SQL is not recorded as it is transformed and sent to the target database.
- yes means SQL is recorded as it is transformed and sent to the target database.

Target Interaction Properties

Controls how an access service interacts with the target database.

The subsection heading and a list of the properties must appear in the access service library configuration file as:

```
{Target Interaction}
Allocate=
DelimitSqlRequests=
DisableROLock=
IsolationLevel=
QuotedStringDelimeter=
ReturnNativeError=
SQLOdbcCursors=
SQLTransformation=
StopCondition=
TargetDBMS=
TargetDecimalSeparator=
```

Allocate

Controls when an access service allocates conversations with the target database system.

Syntax

```
Allocate=[connect | request]
```

Default
`connect`

Values

- `connect` allocates a conversation when the client connects and holds it open for the duration of the client connection.
- `request` allocates a new conversation each time the client application sends a request and deallocates the conversation after each request.

Note: There is a large performance penalty when using the request setting.

DelimitSqlRequests

Allows statements to be batched and executed as a single request to the target DBMS. Statements are delimited with a semicolon (;).

Syntax

```
DelimitSqlRequests=[yes|no]
```

Default

yes

Values

- no batches and executes statements as a single request to the target DBMS, allowing the creation of stored procedures and triggers.
- yes splits the statements that contain semicolon; each delimited statement is sent separately to the target DBMS.

DisableROLock

(Microsoft SQL Only)Disables a lock that is created in a `read-only` cursor.

Syntax

```
Disablerolocks=[yes |no]
```

Default

no

Values

- yes disables the lock on the `read-only` cursors.
- no retains the lock on the `read-only` cursors.

Comments

Sybase recommends setting `DisableROLock` to yes for increased concurrency when accessing shared tables. However, applications opening multiple cursors on the same table within the same connection must set `DisableROLock` to yes to avoid this error:

```
Connection is busy with results for another hstmt
```

IsolationLevel

Controls the level of locking and records access to ODBC-accessible tables.

Syntax

```
IsolationLevel=[ur | cr | rr | sr | vr | no]
```

Default

`no`

Values

See these descriptions of isolation levels in the *Microsoft ODBC 3.5 Programmer's Reference and SDK Guide*:

- `Dirty read` – transaction 1 changes a row. Transaction 2 reads the changed row before transaction 1 commits the change. If transaction 1 rolls back the change, transaction 2 reads a row that is considered to have never existed.
- `Nonrepeatable read` – transaction 1 reads a row. Transaction 2 updates or deletes that row and commits the change. If transaction 1 attempts to reread the row, it receives different row values or discovers that the row has been deleted.
- `Phantom` – transaction 1 reads a set of rows that satisfy some search criteria. Transaction 2 inserts a row that matches the search criteria. If transaction 1 reexecutes the statement to read the rows, it receives a different set of rows.

Acceptable values are:

- `ur` (uncommitted read) – dirty reads, nonrepeatable reads, and phantoms are possible.
- `cr` (committed read) – dirty reads are not possible. Nonrepeatable reads and phantoms are possible.
- `rr` (repeatable read) – dirty reads and nonrepeatable reads are not possible. Phantoms are possible.
- `sr` (serializable) – transactions can be serialized. Dirty reads, nonrepeatable reads, and phantoms are not possible. This is usually implemented by using locking protocols that reduce concurrency.
- `vr` (versioning) – transactions can be serialized, but this value provides higher concurrency. Dirty reads are not possible. This is usually implemented by using nonlocking protocols, such as record versioning.
- `no` (none) – uses the ODBC driver default level.

QuotedStringDelimiter

Specifies the character used for quoted strings.

Syntax

```
QuotedStringDelimiter= char
```

Range

0–1 characters

Default

`'`(single quote)

Values

`char` is the quoted string delimiter for your locale.

ReturnNativeError

Allows a non localized native error message and a native error severity to be returned to the client.

Syntax

```
ReturnNativeError = [yes|no]
```

Default

no

Values

- yes returns non localized native error messages to the client.
- no does not return non localized native error messages to the client.

SQLOdbcCursors

Controls ODBC cursor library usage.

Syntax

```
SQLOdbcCursors=[if_needed | odbc | driver | default]
```

Default

`if_needed`

Values

- `if_needed` uses the cursor library only if it is needed.
- `odbc` uses the ODBC cursor library.
- `driver` uses the cursor management capability of the driver.
- `default` uses the cursor library.

SQLTransformation

Specifies the mode the access service uses for SQL transformation.

Syntax

```
SQLTransformation=[passthrough |sybase | tsql0 | tsql1 | tsql2]
```

Default

passthrough

Values

- **passthrough** sends all SQL statements to the database system as received, without transformation. A client application uses **passthrough** mode to gain direct access to DBMS capabilities.
- **sybase** performs SQL transformation of selected statements. It also allows the use of multipart table names with the **view** command in SQL statements.

StopCondition

Specifies the conditions under which an access service stops processing results.

Syntax

```
StopCondition=[error | none | warning]
```

Default
`error`

Values

- `error` stops processing results only when an error occurs.
- `none` does not stop processing results when errors or warnings occur.
- `warning` stops processing results when an error or warning occurs.

TargetDBMS

Specifies whether SQL transformation are applied to specific dialects of DB2 for z/OS, or DB2 for Windows and UNIX.

Syntax

```
TargetDBMS=[notused | UDBLAN | UDBOS390 | UDBAS400]
```

Default
`notused`

Values

- `notused` transforms SQL to specific dialects.
- `UDBLAN` transforms SQL to specific dialects for DB2 for Windows and UNIX.
- `UDBOS390` transforms SQL to specific dialects for DB2 for z/OS.
- `UDBAS400` transforms SQL to specific dialects for DB2 for AS/400.

Comments

- Affects the generation of DB2 DDL statements, such as **create table** and **alter table**.

- Setting the value to `UDBLAN` or `UDBOS390` indicates which **create table** syntax to use when in **sybase** translation mode. The database part of the three-part table name is appended with or without the **IN DATABASE** clause. `UDBLAN` causes the translation to drop the DATABASE symbol from the **IN DATABASE** clause.

TargetDecimalSeparator

Specifies the character delimiter for the fractional part of decimal numbers passed between the access service and the target DBMS.

Syntax

```
TargetDecimalSeparator= char
```

Range

0–1 characters

Default

. (period)

Values

- `char` indicates the character used by the target as the decimal delimiter.
- A period (.) indicates that the target database uses a period as the decimal delimiter.

Tracing Properties

Controls whether access service data is written to the DirectConnect server trace file.

Because tracing degrades performance, use tracing only when instructed to do so by Sybase Technical Support.

Warning! To provide Sybase Technical Support with all the necessary trace data, the server trace file is allocated a maximum of 20MB of space. When the server trace file exceeds the maximum, it is copied to a file with the same file name and with an "*_old*" extension (`<filename>_old`).

See the *Enterprise Connect Data Access and Mainframe Connect Server Administration Guide* for information about the DirectConnect server trace file, and suggestions for deleting or backing up old log and trace files.

The subsection heading and a list of the properties must appear in the access service library configuration file as:

```
{Tracing}
TraceEvents=
TraceInterface=
TraceTarget=
```

TraceEvents

Specifies whether the access service traces the event handler layer of the access service library.

Syntax

```
TraceEvents=[no|yes]
```

Default

no

Values

- no does not trace the event handler layer.
- yes traces the event handler layer.

TraceInterface

Specifies whether the access service traces the interface layer of the connectivity library.

Syntax

```
TraceInterface=[no|yes]
```

Default

no

Values

- no does not trace the interface layer.
- yes traces the interface layer.

Comments

- The `TraceFilename` file contains the interface layer tracing.
- Traces major ODBC API calls, providing the function called, the return status, and key parameters.

TraceTarget

Specifies whether the access service traces the implementation layer of the access service library.

Syntax

```
TraceTarget=[no | yes]
```

Default

no

Values

- no does not trace the target layer.
- yes traces the target layer.

Comments
Traces data moving to and from the ODBC driver and key Sybase-to-ODBC data conversions.

Transfer Properties

Controls how an access service performs transfer processing.

The subsection heading and a list of the properties must appear in the access service library configuration file as:

```
{Transfer}
BulkCommitCount=
TransferBatch=
TransferBatchSeparator
TransferErrorAction=
TransferErrorCount=
TransferExpress=
TransferPacketSize=
```

BulkCommitCount

Specifies the number of bulk transfer and express transfer rows that are transferred before a **commit** is sent to the target connection.

Syntax

```
BulkCommitCount=integer
```

Range
0–32767

Default
`0`

Value
`integer` is the number of rows that are transferred before a **commit** is issued (between **commit** operations).

TransferBatch

Sets the number of rows that the DirectConnect server sends and fetches into the receive buffer.

Syntax

```
TransferBatch=integer
```

Range
0–32767

Default
1 (one)

Value
`integer` is the number of rows that the DirectConnect server sends and fetches in one batch.

Comments

- The access service builds the designated number of destination template and bulk copy statements in its request buffer before executing each request.
- If the value is 0, the access service sends all statements that fit the request buffer.
- An invalid value defaults to 0.

See also

- *Destination-Template Transfer Statement* on page 115

TransferBatchSeparator

Delimits the statements in a batch.

Syntax

```
TransferBatchSeparator=; (semicolon)
```

Default
`space`

Values

- `;` (semicolon) delimits statements for a DRDA target database.
- `space` delimits statements for Adaptive Server target databases.

Comments
When required, set the property to a semicolon.

TransferErrorAction

Specifies whether to roll back or commit a transfer batch when an error occurs.

Syntax

```
TransferErrorAction=[noaction | rollback]
```

Default
`noaction`

Values

- `noaction` commits the transfer batch when an error occurs.
- `rollback` issues a **rollback** when an error occurs, the `TransactionMode` is `long`, and the `TransferErrorCount` is exceeded.

Comments
To use the `rollback` property, you must set:

- `TransferErrorAction` to `rollback`.
- `TransferErrorCount` to a value greater than zero.
- `TransactionMode` to `long`.

TransferErrorCount

Specifies the number of error rows that are allowed during destination-template transfer before processing stops.

Syntax

```
TransferErrorCount=integer
```

Range
0–32767

Default
0

Value
`integer` is a number of error rows.

Comment
A value of 0 does not stop processing, regardless of the number of errors encountered.

TransferExpress

Specifies whether the bulk transfer statements are handled by express transfer processing.

Syntax

```
TransferExpress=[no | yes]
```

Default
no

Values

- no indicates standard bulk processing.
- yes indicates that express transfer processing.

Comments
Express transfer requires the transfer statement `secondaryname` to match a DSN in the ODBC system information (`odbc.ini`) file.

TransferPacketSize

Sets the packet size between the DirectConnect server and the target database.

Syntax

```
TransferPacketSize=integer
```

Default
512

Values
`integer` must be in multiples of 512 and cannot exceed the target packet size.

Warning! If you increase the packet size beyond the capabilities of Adaptive Server, the transfer login to the secondary database fails.

Comments
To determine the Adaptive Server packet size, use **sp_configure**.

Example 1:

Returns the default packet size:

```
sp_configure “default network packet”
```

Example 2:

Returns the maximum size:

```
sp_configure “max network packet”
```

Setting Up Adaptive Server Enterprise

Set up Adaptive Server Enterprise for remote procedure calls (RPCs) against an access service.

Create an Adaptive Server user ID and password that matches an access service user ID and password.

See the *Adatpive Server Enterprise Installation Guide* for your platform to set up an Adaptive Server.

1. Configure Adaptive Server for remote access.

a) Log in to Adaptive Server as sa and check the current **sp_configure** setting:

```
sp_configure 'remote access'
```

- If the returned value is `1`, Adaptive Server is configured for remote access.
- If the returned value is `0`, enter:

  ```
  sp_configure 'remote access',1
  ```

  Restart Adaptive Server.

2. Define the access service as a remote server.

a) Enter each access service name in the Adaptive Server `SYSSERVERS` table:

```
sp_addserver service_name service_class
```

where:

- *service_name* is the access service to set up as a remote server.
- *service_class* must be direct_connect.

The access service name is case-sensitive and must match the name you used to define connectivity between Adaptive Server and the access service.

b) Verify that the access service is successfully defined as a remote server:

```
sp_helpserver server_name
```

This command returns a result set with a list of the services defined to Adaptive Server.

c) Use **sp_addexternlogin** to add an external login password to allow access to the access services.

d) Verify Adaptive Server/CIS connectivity using the Transaction-SQL extension **connect to** *server_name*. This is the same server name you created in a previous step using **sp_addserver**. This command establishes a **passthrough** mode connection to the access service that remains in effect until you issue a **disconnect** command.

e) Enable location transparency of remote data through remote object mapping:

- Use **sp_addobjectdef** to define the storage locations of remote objects.

- Use either **create table** or **create existing table** (as applicable) to map remote table schema to Adaptive Server/CIS.

f) Use **select** to perform joins across the access service after all the previous steps.

See the *Component Integration Services Guide* for more information, as well as instructions for configuring and tuning Adaptive Server/CIS.

3. Define connectivity between Adaptive Server and the access service.

In the Windows `sql.ini` file or the UNIX `interfaces` file, add a QUERY section with the same name and `ConnectionSpec` that was added in step 2 (`service_name`).

See the *Adaptive Server Enterprise System Administration Guide* and the *Adaptive Server Enterprise Reference Guide*.

RPC Issues

When Adaptive Server issues an RPC to an access service, it attempts to connect using a service name identical to the Adaptive Server identifier in the Windows `sql.ini` file or the UNIX `interfaces` file.

To support Adaptive Server RPC events, either:

- Define a service within the access service library using the same name as Adaptive Server, using the previous procedure, or,
- Set up a service name redirection to redirect the Adaptive Server to the appropriate service.

  See *Enterprise Connect Data Access and Mainframe Connect Server Administration Guide > Service Name Redirection*.

Setting Up ECDA Option for ODBC for Adaptive Server/CIS

Configure ECDA Option for ODBC so that Adaptive Server/CIS can configure it to transparently access data on a remote server as an access service.

Prerequisites

Verify that you are running the latest Adaptive Server/CIS update (ESD) for your platform.

Task

1. Configure these properties for ECDA Option for ODBC to a Microsoft SQL Server datasource:
   - **QuotedIdentifier= on**
   - **SqlTransformation= sybase**
   - **CSPColumnODBCVersion= 2** (Adaptive Server/CIS version 12.0)
   - Use the default, **CSPColumnODBCVersion= 3** (Adaptive Server/CIS 12.5 and later)
2. Configure the Microsoft SQL Server driver:
   - On Windows – choose "enabled quoted identifiers" for the Microsoft SQL Server driver in the ODBC administrator.
   - On UNIX – add the configuration property **QuotedId= yes** in the ODBC system information file (`odbc.ini`).

Adaptive Server/CIS with the Adaptive Server Transaction Model

Adaptive Server/CIS requires an access service to support some aspects of the Adaptive Server transaction model.

If the access service is configured correctly for Adatpive Server/CIS, and if it recognizes the Adaptive Server/CIS client application, it supports the transaction model. The model includes these statements:

- **begin tran**
- **prepare tran**
- **commit** or **rollback**

Note: This is a limited implementation of the Adaptive Server transaction model and is available only for Adaptive Server/CIS. Other applications cannot use these features.

To process Adaptive Server/CIS transactions, you must set the transaction mode and transform level parameters.

See the *Adaptive Server Enterprise Component Integration Services Guide*.

If all conditions are met, the transaction statements are handled as:

begin tran statement

The access service:

- Sets a flag to indicate it is in an Adaptive Server/CIS transaction.
- Sets the transaction mode to **long**.
- Returns a successful status, as if a **set** command was executed.

Other considerations:

- Nested transactions are not required nor are they supported.
- Adaptive Server/CIS does not issue this statement if outstanding changes were not committed or rolled back.
- The access service issues a **commit** prior to handling the statement.

prepare tran statement

If the access service is in an Adaptive Server/CIS transaction, it:

- Ignores this statement.
- Returns a successful status, as if a **set** command was executed.

This statement is issued in preparation for a **commit** statement.

commit or rollback statement

If the access service is in an Adatpive Server/CIS transaction, it:

- Resets the flag that indicates it is in an Adaptive Server/CIS transaction.
- Resets the mode to **short**.
- Returns a success or failure status of the **commit** or **rollback** statement to Adaptive Server/CIS.

Adaptive Server/CIS expects the access service to be in **short** transaction mode unless:

- Adaptive Server/CIS opened one or more read-only cursors.
- Adaptive Server/CIS issued a **begin tran** but did not issue a **commit** or **rollback**.

Because the access service automatically switches to **long** transaction mode when a cursor is declared or when a dynamic event **prepare tran** occurs, it backs out of this mode when the last cursor and prepared statement are deallocated and not in a **begin tran** block.

Note: Adaptive Server/CIS always issues dynamic events within its own transaction, so it is not necessary to back out when the last dynamic statement is freed.

In the event of a rollback, the access service issues a **commit** and sets the transaction mode to **short**.

ECDA Option for ODBC for Replication Server Settings

Replication Server® provides several scripts or settings that can help you set up a connection from the target database to Replication Server.

Use an access service that is configured to:

- **TransactionMode = long**
- **ReturnNativeError = yes**
- **SQLTransformation = sybase**
- **Allocate = connect**

See:

- *Replication Server Administration Guide Volume 1 > Manage Database Connections*.
- *Replication Server Administration Guide Volume 2 > Customize Database Operations*.
- *Replication Server Heterogeneous Replication Guide > IBM DB2 for Linux, UNIX, and Windows as Primary Data Server*.

Query Global Variables

A user or client application can query a global variable to find and view the property and processing values that affect that client connection.

A global variable represents either the current value of a **set** statement, or information about the processing state of a connection.

Global variables are preceded by two “@@” characters and are not case-sensitive. To query a global variable, issue:

```
select @@variable_name
```

where *variable_name* is the name of the relevant global variable.

Other rules that apply:

- All global variables can be queried regardless of the SQL transformation mode in effect for the connection.
- The **select** statement must be the only one in a batch.
- The statement can be terminated with a semicolon.

Example 1:

SQL statement to query a global variable for the Client Interaction property, `SendWarningMessages`:

```
select @@SendWarningMessages
```

See also

- *Configuration Properties Quick Reference* on page 159

Issue Set Statements

Unlike global variable queries, which allow you to view but not change any settings, a user or client application can issue a **set** statement to change those values that affect the current client connection.

These values remain in effect only for the duration of the client connection or until another **set** statement is issued.

Access service **set** statements are not case-sensitive. To set an access service configuration property value or processing value for the current connection:

```
set { property_name | processing_name } value
```

where:

- *property_name* is the configuration property.
- *processing_name* is the processing option.
- *value* is a valid value for the configuration property.

Example 1:

A **set** statement for the Client Interaction property `SendWarningMessages`:

```
set SendWarningMessages {no | yes}
```

See also

- *Configuration Properties Quick Reference* on page 159

SQL Statements

SQL transformation modes affect the way the access service modifies SQL statements that are written for one database but are processed against another.

Although the transformation mode primarily affects the way in which the access service treats incoming SQL statements, it also affects these functional areas:

- Transaction
- Datatype handling
- **set** statements
- Remote Stored Procedures (RSPs)
- Adaptive Server/CIS interoperability
- ODBC driver interoperability

When you configure the access service with a specific transformation mode, that mode is effective for all client connections unless you use a **set** statement to alter it for a specific connection.

To make dialects appear as common SQL, the access service supports **passthrough** and **sybase** standard transformation modes.

passthrough Mode

Use **passthrough** mode to grant the client direct access to the capabilities of a DBMS target.

The SQL dialect is for ODBC, not the actual DBMS dialect to which the ODBC driver is connected.

The access service performs virtually no SQL transformation. **passthrough** mode converts carriage returns and line feeds. All other commands are passed directly to the ODBC driver, and the results are returned to the client.

Note: In ECDA Option for ODBC, you can prepare multiple SQL statements in a single statement. When you do this, be sure to separate the statements with semicolons.

sybase Mode

Use **sybase** mode for maximum compatibility between different target databases.

sybase mode allows client applications to operate independently of the target they are accessing.

When sybase mode is in effect, the access service performs a limited amount of Transact-SQL® syntax transformation on the SQL text that it receives, including text found in these commands: **language**, **cursor declare**, **dynamic prepare**, and **dynamic execute immediate**.

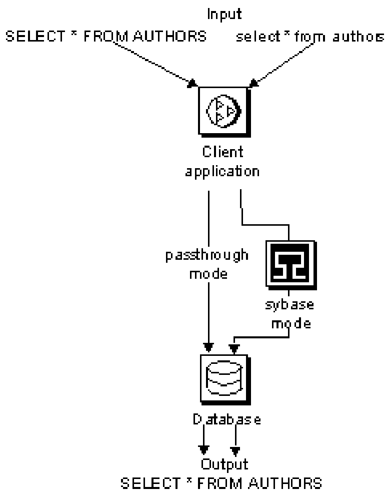
In **sybase** mode, the access service transforms the SQL text it receives into syntax that the target DBMS supports. If the access service receives syntax it does not recognize, it passes the text to the DBMS for execution.

Because an application uses this mode for purposes of compatibility with all access services provided by Sybase, it should not issue SQL commands that are unique to any single target DBMS.

The differences between **sybase** mode and **passthrough** mode:

- **passthrough** mode transfers similar dialect and syntax directly from the client application to the target.
- **sybase** mode performs translation functions, changing the **select** statement from lowercase in the client application to uppercase in the target.

Figure 1: passthrough and sybase Transformation Modes



Transformations in sybase Mode

sybase mode transforms the SQL syntax.

sybase mode transformations:

- Provides standard transformations of parameter marker names beginning with the “@” character
- Removes Transact-SQL comments in the form /*

- Converts Transact-SQL comparison operators (such as ! and =) into the target DBMS equivalent
- Converts single and double quotation marks used as string delimiters to the appropriate delimiter for the target DBMS
- Strips dollar signs from money constants

Unsupported syntax passes unchanged to the DBMS for processing.

Note: The **sybase** mode does not convert nonquoted tokens to uppercase.

Standard Transformations for Transact-SQL Commands

Standard transformations are provided only for portions of the Transact-SQL syntax that can be directly translated into target DBMS syntax.

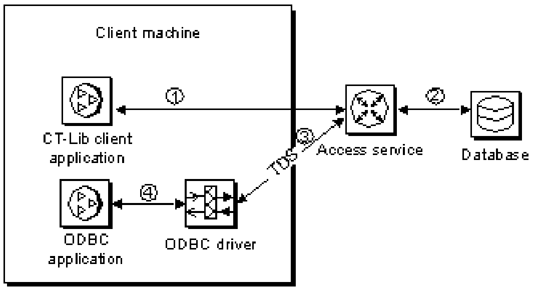
Command	Description of Transformation
alter table	Adds new columns to an existing table
begin transaction	Begins a new transaction
commit transaction	Commits all work performed for this transaction
create index	Creates a new index on a table
create table	Creates new tables
create view	Creates a new view
delete	Deletes rows from a table
delete (cursor)	Removes rows from a table
delete (dynamic)	Removes rows from a table
drop index	Removes an index from a table
drop table	Removes a table
drop view	Removes one or more views
execute	Runs system or user-defined stored procedures
grant	Assigns authorizations to users
insert	Adds new rows to a table or view
insert bulk	(Currently not supported)
prepare transaction	Prepares to commit a transaction
revoke	Revokes permission from users
rollback transaction	Rolls back or aborts the current transaction
select	Retrieves rows from database objects

Command	Description of Transformation
truncate table	Truncates a table by removing all rows (this statement is not logged and is not part of any transaction)
update	Adds or modifies data in existing rows
update (cursor)	Positional update: changes data in row made current by a read cursor
update (dynamic)	Dynamic update: changes data in existing rows
use	Accesses an existing database

Request Processing Flow

The client application issues a request through the access service to the database.

Figure 2: Request Processing Flow



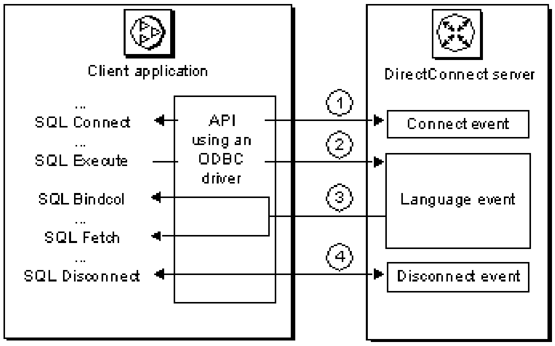
Request processing steps:

1. When the client application issues a request; such as a **select** statement, the client application program interface (API) receives the request and sends it to the access service.
2. The access service receives the request, transforms it as specified in its configuration, and executes it.
3. After the request processes, the access service converts target datatypes to Open Server datatypes and returns the results to the client application.
4. The client application disconnects from the access service.

ODBC Client API Processing

Use ODBC APIs to interact with the ODBC client application and the DirectConnect server.

Figure 3: ODBC Client API Example



The processing flow:

1. The ODBC client application initiates a **connect** event to a specific access service using the **SQLConnect** call. The ODBC driver uses the CT-Library API.
2. The ODBC application builds and executes the request using the **SQLExecDirect** function. This initiates a **language** event in the DirectConnect server.
3. The ODBC API uses **SQLBindCol** to assign local variables to specific columns. The **SQLFetch** call returns the resulting data to the application.
4. The ODBC API terminates the request with a **SQLDisconnect** call. This initiates a **disconnect** event in the DirectConnect server.

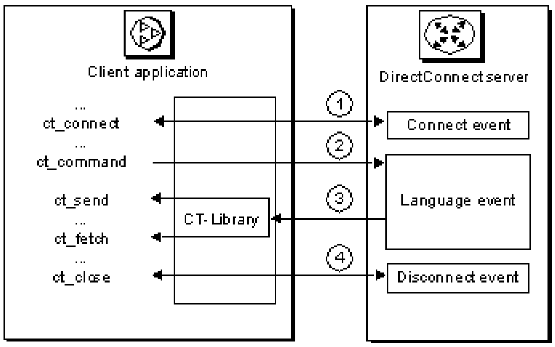
Procedure calls for the ODBC API are described in the *Microsoft ODBC 3.5 Programmers Reference and SDK Guide*.

Note: ECDA Option for ODBC supports the use of `text` or `image` pointers for Microsoft SQL Server only, such as: **ct_data**, **ct_get_data**, and **ct_put_data**.

CT-Library Client API Processing

Use CT-Library API to interact between a Sybase Open Client CT-Library API and the DirectConnect server.

Figure 4: CT-Library Client API Example



The processing flow:

1. When a CT-Library application issues a request, it uses the **ct_connect** call to initiate a **connect** event to a specific access service.
2. The CT-Library API executes the request using the **ct_command** and **ct_send** calls. This initiates a **language** event in the DirectConnect server.
3. The CT-Library API uses **ct_fetch** to return the requested results to the client application.
4. The CT-Library API terminates the request with **ct_close**. This call initiates a **disconnect** event in the DirectConnect server.

Note: CT-Library is used by both Replication Server and Adaptive Server/CIS to connect to the DirectConnect server.

Procedure calls for the CT-Library API are described in the *Sybase Open Client Client-Library/C Reference Manual*.

Note: Sybase no longer provides a back-end, server-side ODBC driver to access Microsoft SQL Server, DB2 UDB, or ODBC-accessible data sources. Instead, obtain an ODBC driver from the vendor of your database, and follow vendor instructions to configure it to be compatible with the Sybase ODBC driver manager.

See the *Enterprise Connect Data Access Installation Guide* for your platform.

Manage Processing Results

Manage processing results using the `Allocate`, `StopCondition`, and `TransactionMode` configuration properties together.

Allocate Configuration Property

Use `Allocate` to determine when a connection should be deallocated from the host server.

- If `Allocate` is set to `connect`, the connection is kept until the client issues some type of deallocation.
- If `Allocate` is set to `request`, the connection is established at the start of a request, and then dropped.

`Allocate` does not affect the transaction mode.

If `Allocate` is set to `request` and the transaction mode is `long`, the connection stays available until a batch ends on a **commit** or **rollback** statement or **exit**:

- If the batch ends on a **commit** or **rollback** statement, the transaction ends properly.
- If the batch ends with **exit**, the connection also ends and any uncommitted SQL statements are rolled back.

If `Allocate` is set to `request` and the transaction mode is `short`, after the request is processed, all SQL statements in the request are committed and the connection is dropped. If the request ends with a **rollback** statement, the SQL statements in the request are rolled back and the connection is dropped.

If `Allocate` is set to `request`, the transaction mode is `short`, and a **begin transaction** is encountered in the request, the SQL statements to that point are committed. The connection stays open, and the transaction mode is set to temporary `long`. When a **commit** or **rollback** statement is encountered, the transaction mode reverts to `short`, and the allocation reverts to `request`.

While in temporary `long` transaction mode, the connection stays open until the transaction ends, even if the transaction spans multiple requests. The `Allocate on request` message is ignored until the transaction ends.

StopCondition Configuration Property

Use `StopCondition` to specify whether the access service stops processing a request when it encounters an error or a warning.

Valid values are:

- `error`
- `warning`
- `none`

If you specify `none`, processing continues even when errors occur. This property is useful if you batch multiple statements in a request.

TransactionMode Configuration Property

Use `TransactionMode` to govern the transaction behavior. You can set it to `short` or `long`.

Short Transactions

If the transaction mode is set to `short`, the access service controls the commitment of requests.

After sending the request to the database, the access service automatically issues either:

- A **commit**, if the request succeeds, or,
- A **rollback**, if the request fails.

Behavior of Short Transactions and Begin Transaction

The **begin transaction** phrase affects the behavior of `short` transactions.

- If the **begin transaction** is issued prior to the end of the request, it triggers a **commit** of all previous statements in the request and temporarily sets the transaction mode to `long`.
- If the **begin transaction** is not issued prior to the end of a request, the request marks the end of the transaction, and the transaction is committed.
- You can issue a **commit** or **rollback** to end the temporary `long` mode. Doing this causes the transaction mode to revert to its status before the **begin transaction** was issued.
- If the batch is in temporary `long` mode and ends (exits) without a **commit** or **rollback**, the SQL statements are rolled back, and the transaction mode reverts to its original status.

Long Transactions

If the transaction mode is set to `long`, you control when the transaction ends by issuing a **commit** or **rollback**.

If the access service encounters a **begin transaction**, an error message is issued.

The client application issues a **commit** or **rollback** statement for each transaction. When the client application closes the connection, the access service issues a **rollback** before exiting.

Resulting Actions in Transaction Management

The actions that occur as a result of different combinations of `Allocate`, `StopCondition`, and `TransactionMode` configuration properties.

Table 2. State, Event, Result Table for Transaction Management

Configuration Property States and Conditions	Event	Resulting Activities
`TransactionMode` = `short` `Allocate` = `request` `StopCondition` = `error` or `warning`	If `StopCondition` is encountered.	Rolls back transaction at `StopCondition`. Does not continue request and ends the connection.
	If end of request with no `StopCondition`.	Commits transaction and ends the connection.
	If **begin transaction** is encountered in request.	Commits transaction. Switches to temporary `long` transaction mode and turns on **begin transaction**.
	If **commit** or **rollback** is encountered.	Issues the **commit** or **rollback**.
	If **cursor declare** or **dynamic prepare** is encountered.	Commits transaction. Switches to `long` transaction mode.
`TransactionMode` = `short` `Allocate` = `request` `StopCondition` = `none`	If error or warning is encountered.	Does not roll back. Continues request and keeps the connection until end of request.
	If end of request.	Commits transaction and ends the connection.
	If **begin transaction** is encountered in request.	Commits transaction. Switches to temporary `long` transaction mode and turns on **begin transaction**.

Configuration Property States and Conditions	Event	Resulting Activities
	If **commit** or **rollback** is encountered.	Issues the **commit** or **rollback**.
	If `cursor declare` or `dynamic prepare` is encountered.	Commits transaction. Switches to `long` transaction mode.
`TransactionMode` = `short` `Allocate` = `connect` `StopCondition` = `error` or `warning`	If `StopCondition` is encountered.	Rolls back transaction at `StopCondition`. Does not continue in request but keeps the connection.
	If end of request with no `StopCondition`.	Commits transaction and keeps the connection.
	If **begin transaction** is encountered in request.	Commits transaction. Switches to temporary `long` transaction mode and turns on **begin transaction**.
	If **commit** or **rollback** is encountered.	Issues the **commit** or **rollback**.
	If **cursor declare** or **dynamic prepare** is encountered.	Commits transaction. Switches to `long` transaction mode.
`TransactionMode` = `short` `Allocate` = `connect` `StopCondition` = `none`	If error or warning is encountered.	Does not roll back. Continues in request and keeps the connection.
	If end of request with no `StopCondition`.	Commits transaction and keeps the connection.
	If **begin transaction** is encountered in request.	Commits transaction. Switches to temporary `long` transaction mode and turns on **begin transaction**.
	If **commit** or **rollback** is encountered.	Issues the **commit** or **rollback**.
	If **cursor declare** or **dynamic prepare** is encountered.	Commits transaction. Switches to `long` transaction mode.
`TransactionMode` = `long` or temporary `long` `Allocate` = `request` or `connect` `StopCondition` = `error` or `warning`	If `StopCondition` is encountered.	Does not roll back transaction at `StopCondition`. Does not continue request but keeps the connection.
	If end of request with no `StopCondition`.	Keeps the connection.

Configuration Property States and Conditions	Event	Resulting Activities
	If end of request with **commit** or **rollback**.	Ends the connection.
	If **begin transaction** is encountered in request.	If not in **begin transaction** block, sets **begin transaction**. If in **begin transaction** block, ignores it and issues message.
	If **commit** or **rollback** is encountered.	Issues the **commit** or **rollback**. If in **begin transaction** block, ends **begin transaction**.
	If **cursor declare** or **dynamic prepare** is encountered.	Executes statement.
Same, plus **cursor** and **dynamic** statements all freed and in temporary `long` mode.	If **commit** or **rollback** is encountered.	Issues the **commit** or **rollback**. Ends **begin transaction** and reverts to previous transaction mode.
Same, plus **cursor** and **dynamic** statements `active` and in temporary `long` mode.	If **free** of a **cursor** or **dynamic**, means that all **cursors** and **dynamics** are now freed.	Executes **free**. If not in **begin transaction** block, reverts to previous transaction mode.
`TransactionMode = long` or temporary `long Allocate = request` or `connect StopCondition = none`	If `StopCondition` is encountered.	Does not roll back transaction at `StopCondition`. Continues in request and keeps the connection.
	If end of request with no `StopCondition`.	Keeps the connection.
	If end of request with **commit** or **rollback**.	Ends the connection.
	If **begin transaction** is encountered in request.	If not in **begin transaction** block, sets **begin transaction**. If in **begin transaction** block, then ignores and issues message.
`TransactionMode = long` or temporary `long Allocate = request` or `connect StopCondition = none`	If **commit** or **rollback** is encountered.	Issues the **commit** or **rollback**. If in **begin transaction** block, ends **begin transaction**.
	If `cursor declare` or `dynamic prepare` is encountered.	Executes statement.

Troubleshooting

Troubleshoot processing problems using the DirectConnect server log and trace files.

Configuration properties control whether data is recorded in the server log and trace files for each logging and tracing option. To configure logging and tracing properties, edit the `dcany.cfg` configuration file or use DirectConnect Manager.

See also

- *Access Service Library Configuration* on page 9

Tracing

Tracing properties allow you to record troubleshooting information for Sybase Technical Support.

Warning! To provide Sybase Technical Support with all the necessary trace data, the server trace file is allocated a maximum of 20MB of space. When the server trace file exceeds the maximum, it is copied to a file with the same file name and with an "`_old` " extension (`<filename>_old`).

See the *Enterprise Connect Data Access and Mainframe Connect Server Administration Guide*.

See also

- *Tracing Properties* on page 51

RPC Feature

The RPC feature allows a stored procedure to initiate an event in a remote database.

A client API can invoke a remote stored procedure (RSP) to:

- Invoke an external stored procedure
- Execute a language statement as an RPC
- Execute a **transfer** request

Creating and Executing Adaptive Server Remote Stored Procedures

Create an Adaptive Server stored procedure that executes a remote stored procedure (RSP).

1. Connect to Adaptive Server using **isql**.
2. At the prompt, enter:

```
create procedure newcust @custname varchar(nn),
 @custno varchar (nn) as
 begin
 execute servername. . .addcust
 @addname=custname, @addno=custno
end
```

where:

- @*custname* specifies the customer name.
- @*custno* specifies the customer number.
- *servername* specifies the access service instance to use. The three periods (...) after*servername* are required.
- *addcust* is the stored procedure name on the host.
- @*addname* is the stored procedure representing the new customer name.
- @*addno* is the stored procedure representing the new customer number.

Note: The new procedure must specify an existing RSP and provide any arguments that the RSP requires.

Example 1: Executes an RSP using isql:

```
c:>isql -Ssybase -Uuser -Ppasswrd
 1> execute newcust xxxx,yyyy
2> go
```

where:

- *xxxx* is the new customer name.
- *yyyy* is the new customer number.

The results obtained depend on the defined **addcust** stored procedure.

Executing a Language Statement as an RPC

Execute a SQL language statement to the access service through an RPC.

1. Connect to Adaptive Server using **isql**.
2. At the prompt, enter:

```
C:> isql -Sadaptiveserver -Uuser -Ppassword
 1> execute directconnect. . .dcon “select * from user.authors”
 2> go
```

where:

- *directconnect* is the name of the remote server. The three periods (...) after *directconnect* are required.
- **dcon** is the keyword of the RPC.

Warning! When using the keyword **dcon**, the access service inserts a space between each variable.

The access service RPC event handler is sensitive to several RPC keywords, including the keyword **dcon**. An RPC can have many parameters. Before the resulting string is executed, all parameters are concatenated. The access service translates the first parameter into a dynamic SQL statement, submits it to the target database, and returns the result set to the client application.

The event sequence is:

a. Adaptive Server determines whether the remote server *directconnect* is configured as a remote server:

- If the remote server is not configured correctly, the request fails.
- If the remote server is configured correctly, Adaptive Server checks for a site handler connection to the remote server.

b. Adaptive Server checks:

- If the site handler connection exists, then Adaptive Server connects to the remote server, triggering a connect event in the DirectConnect server.
- If the connect event processes successfully, then Adaptive Server triggers an RPC event in the DirectConnect server and submits the RPC called **dcon**. The first parameter to the RPC is the dynamic SQL language statement that is executed.
- If the site handler connection does not exist, then Adaptive Server establishes one.

This connection exists for the term of the RPC. It is reused if Adaptive Server submits other RPCs.

Rules for Using Language Statements as RPCs

Process the language statement as RPCs.

Adaptive Server has a strict model for processing language statements as RPC events:

1. It connects to the remote DirectConnect server and submits the RPC.
2. After it processes the results, it disconnects.

These quick connects and disconnects provide minimal network traffic.

Validation

When you log in to Adaptive Server, you must use a valid ECDA Option for ODBC target database user ID and password.

Transformation Mode and Syntax

All SQL transformation rules apply, including:

- If the DirectConnect server is configured for **passthrough** mode, the SQL within the double quotes must comply with the target SQL syntax.
- If the DirectConnect server is configured for **sybase** mode, the SQL must be in Sybase Transact-SQL dialect.

Commitment Control Rules

- If the access service is configured for long transactions, the SQL submitted must not be sensitive to a **commit**. For example, if the SQL is an **insert** statement that does not batch a **commit** into the statement, the **insert** rolls back using long transaction rules.
- If the access service is configured for short transactions, the SQL submitted is bound by short transactions that supply a **commit** by default.

Transfer RPC Event

The client application can create a stored procedure within Adaptive Server that invokes the access server library **transfer** function.

The access service library receives the **transfer** command as an RPC event with these arguments:

- Name of the RPC– “transfer”
- Argument 1– The secondary connection information ({to | from} “server userid pw”)
- Argument 2– Either the **bulk copy target** command or the destination-template **sourceselectstatement**.

- Argument 3– Either the bulk copy **sourceselectstatement** or the destination-template **sourceselectstatement**.

Example 1:

Shows a stored procedure that initiates a bulk-copy **transfer** statement:

```
create procedure replauth as
 begin
 execute servername. . .transfer
 "to 'servername2 userid password';",
 "with replace into authors;",
 "select * from authors;"
 end
```

where:

- **replauth** is the stored procedure.
- *servername* specifies the access service to handle the transfer. The three periods after *servername* are required.

  The access service library recognizes anything other than "transfer" in the next position as the name of an ODBC stored procedure.
- *servername2* specifies the secondary database for the **transfer** command.

Note: The RPC can have any number of parameters, because the RPC or parameters are concatenated and executed as a **transfer** command.

Executing a Transfer RPC

A client can log in to the Adaptive Server on which a procedure is defined and run the transfer RPC created.

1. Log in to Adaptive Server.
2. Run:

   ```
   execute replauth
   ```

   When Adaptive Server executes **replauth**, it passes an RPC to the access service. The access service returns any result rows or messages to the client application, not to Adaptive Server.

Using Triggers

Set up an Adaptive Server stored procedure as a trigger that executes automatically when the triggering condition is met.

1. Log in to Adaptive Server.
2. Create a trigger event.

Example 1: Creates a trigger that calls an RSP named “pcrsp” when the `phone` column is updated in the Adaptive Server `authors` table. In turn, “pcrsp” updates the `authors` table, using `au_id` to specify the row to update.

```
create trigger updatephone on authors as
 if update (phone)
 begin
 declare @ph varchar(14)
 declare @id varchar(14)
 declare @err int
 select @ph = inserted.phone from inserted
 select @id = inserted.au_id from inserted
 execute servername. . .pcrsp @phone=@ph,
 @au_id=@id
 select @err = @@error
 if (@err >> 0)
 begin
 print ‘error _ rolling back’
 rollback tran
 end
 else
 commit tran
 end
```

After it is created, `updatephone` starts whenever `phone` is updated:

```
C:>isql -Ssybase -Uuser -Ppasswrd
 1> update authors
 2> set phone=’xxx-xxx-xxxx’
 3> where au_id like ‘yyy-yy-yyyy’
 4> go
```

If the update fails, the access service rolls back the Adaptive Server transaction and shows this message:

```
@ERR >> 0
```

See:

- *Sybase Open Server-Library/C Reference Manual*
- *Sybase Open Client-Library/C Reference Manual*

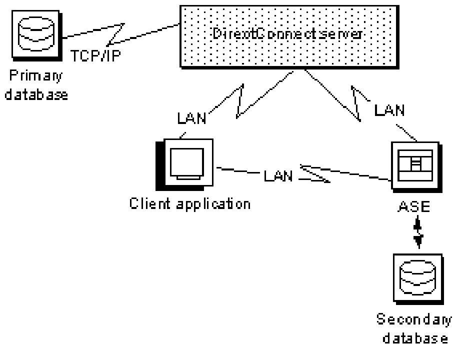
Transfer Process

The transfer process allows you to transfer rows and columns of data between tables in multiple databases from a client application.

Based on your needs and limitations, you can select from one of three transfer options: bulk copy, express, and destination-template.

Access Service Data Transfer Between Databases

The access service configured to access a particular primary database.The secondary database is an Adaptive Server database located on the LAN. The client application can transfer data from the primary database to Adaptive Server, or from Adaptive Server to the primary database.



During a transfer:

- Data flows from a table in the source database, through the DirectConnect server, to the target database. Although the client application initiates the transfer, the data does not flow through it.
- The DirectConnect server becomes a client to the secondary database.

Transfer Direction

Transfer the data from the primary to the secondary database, or from the secondary to the primary.

- A **transfer to** statement transfers data from the primary database to the secondary database. The primary database becomes the source database to the secondary database, which is the target.
- A **transfer from** statement transfers data from the secondary database to the primary database. The secondary database becomes the source database to the primary database, which is the target.

When you execute a bulk-copy **transfer from** statement, the secondary database (either Adaptive Server database or another database) is the source of the data to be transferred. The primary database becomes the destination database, or target.

Note: Sybase recommends that you use the **transfer from** command from the primary database when you transfer data between two access services. Using **transfer from** guarantees native datatype mapping and returns the proper datatype result set.

See also

- *Datatype Conversion for Transfer Processing* on page 93

Transfer Process – Unit of Work

A unit of work is one or more requests that execute, commit, or roll back as a group.

Bulk Copy and Express Transfer

Unit of work is based on the setting of the `BulkCommitCount` property.

- If `BulkCommitCount` is set to `0`, the entire transfer is treated as a unit of work. The access service performs a **commit** after the last row of data is inserted into the target table, even if value errors occurred for individual rows of the transfer.
- If `BulkCommitCount` is set to a non zero value, each block of `BulkCommitCount` rows is treated as a unit of work. The access service issues a **commit** after each block of `BulkCommitCount` rows.

Example 1:

If `BulkCommitCount` is set to 50, each block of 50 rows is treated as a unit of work, and a **commit** is issued after each 50 rows.

See also

- *Value Errors* on page 96

Destination-Template Transfer

When a destination-template **transfer** statement moves data from Adaptive Server to the primary database, the access service automatically sets the `StopCondition` property to `none`.

Subsequent **commit** and **rollback** processing is determined by whether `short` or `long` transactions are in effect:

- If `short` transactions are in effect, the access service issues a **commit** after each batch, whether or not errors occurred in the request. In this case, each batch of inserts is a unit of work.
- If `long` transactions are in effect, the access service issues a **commit** at the end of the entire transfer. Because `StopCondition` is set to `none`, the access service never issues a **rollback**. In this case, the entire transfer is a unit of work.

Transfer Targets

The **transfer** statement allows you to move data in either direction between the primary database and supported databases.

Supported databases:

- Adaptive Server
- Adaptive Server/CIS
- Other access service and legacy products or services:
  - DB2 UDB (z/OS, UNIX, and Windows)
  - Microsoft SQL Server
  - ODBC-accessible databases
- Any Open Server application that supports SQL.

Datatype Conversion for Transfer Processing

Datatype conversion is handled differently by the access service for the different **transfer** types.

Bulk-Copy and Express Transfer Processing

After converting the incoming source database datatypes, the source datatypes are converted into the actual datatypes of the target columns. If the source and target columns have incompatible datatypes, the transfer ends with an error.

Destination-Template Transfer Processing
After converting the incoming source database datatypes, the datatype qualifiers specified are used, with the question marks in the template. When the question marks do not have qualifiers, the access service uses the datatypes of the source to determine the default qualifiers.

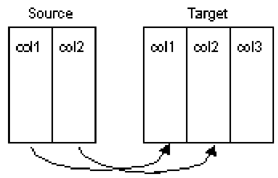
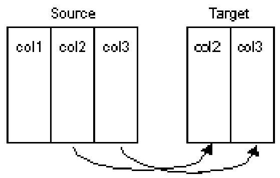
Warning! ECDA Option for ODBC cannot correctly transfer `varchar` values containing empty strings (zero-length non null strings), in the bulk copy and destination-template transfer process. Empty string `varchar` values are interpreted as NULL values. However, express transfer processes this empty string correctly.

Transfer Options for Processing Data

Bulk-copy transfer and express transfer allow you to quickly and directly transfer large amounts of compatible data between databases.

Destination-template transfer allows you the flexibility and control to insert the data in a template before sending it to the target database.

Table 3. Comparison of Two Transfer Command Types

Use Bulk-Copy and Express Transfer to:	Use Destination-Template Transfer to:
Execute the transfer quickly	Exercise more control over the transfer
Perform implicit datatype conversions	Move tables of data in which you need to explicitly specify datatype conversion, such as when the data is structurally incompatible
Move entire tables of data in which column types are compatible between the source and destination databases:  transfer to secondary_connection; with replace into target_table; select col1, col2 from source_table	Perform actions other than **INSERT** against a target database using input from the source database, such as **UPDATE**, **DELETE**, and **CREATE**  transfer to secondary_connection; select col2, col3 from source_table; insert into target_table values (?,?)

See also

- *Bulk-Copy Transfer and Express Transfer* on page 99
- *Destination-Template Transfer Statement* on page 115

Transfer Error Types and Error Handling

Transfer processing errors are classified as structural errors and values errors.

- Structural errors, for which the access service cancels the transfer process before any rows are transferred.
- Value errors, which occur on a row-by-row basis during the transfer process.

Structural Errors

Structural errors occur because of incompatible datatypes or number of columns.

Incompatible Datatypes

Datatypes are incompatible when source and target table datatypes cannot be mapped to one another. For example, `binary` to `datetime` transfers are not allowed.

- In bulk-copy and express transfers, before the access service moves any data from the source database to the target database, the access service compares datatypes in the source to datatypes in the target.
- In destination-template transfers, the access service compares the source table datatypes to the qualifiers set in the destination-template statement.

For all transfer types, if any columns have incompatible datatypes, the access service cancels the transfer without attempting to transfer any rows.

Incompatible Number of Columns

An unequal number of columns between the source and the target may cause a structural error.

- For bulk-copy and express transfer:
  - If the number of source columns exceeds the number of target columns, the access service cancels the transfer.
  - If the number of target columns exceeds the number of source columns, processing continues if all of the extra columns of the target table accept nulls. If any one of the extra columns in the destination table does not accept nulls, the access service cancels the transfer.
- For destination-template transfer:
  - If the number of columns returned by the **sourceselectstatement** exceeds the number of question marks in the destination-template **transfer** statement, the access service cancels the transfer.
  - If the number of question marks in the destination-template **transfer** statement exceeds the number of source columns, the keyword **null** replaces the extra question marks for each row of the transfer.

Value Errors

Value errors occur during bulk-copy transfer or destination-template transfer processing when the value being inserted either cannot be converted to the target column's datatype or is out of range for the target column's datatype.

Note: Express transfer does not support value error handling. If an error occurs, it aborts the entire transfer.

The access service handles value errors using these properties:

- `CharConvertError`
- `NumConvertError`
- `DatetimeConvertError`
- `DefaultDate`
- `DefaultTime`
- `DefaultNum`

If the `SendWarningMessages` property is set to yes, the access service sends a message to the client application when it encounters value errors.

In addition, the preceding properties can be used to fill in default values when datatype conversion fails during bulk-copy and destination-template transfers.

See also

- *Bulk-Copy Transfer and Express Transfer* on page 99
- *Access Service Library Configuration* on page 9

Transfer Processing Error Reporting

Obtain error information about bulk-copy and destination-template transfer processing.

- Include the `with report` phrase in the **transfer** statement. As a result, the access service returns a result set containing one `VARCHAR` column and one row that indicates the number of rows transferred, rejected, and modified during processing.
- Immediately following a **transfer** process, execute these statements:

  select `@@RejectedRowCount`

  select `@@DefaultedRowCount`

  These global variables return the number of rejected or defaulted rows.
- Set `SendWarningMessages` to yes, so that the access service returns warning messages to the client application when data conversion errors occur.

Control Processing with the TransferErrorCount Property

During bulk-copy and destination-template transfer processing, the access service automatically sets the `StopCondition` property to `none`.

It uses the value set in the `TransferErrorCount` property to determine how many error rows are allowed before processing stops. Set the value for the `TransferErrorCount` property with:

```
set TransferErrorCount nnn
```

The default setting is `0` (zero), which causes the access service to ignore errors.

Bulk-Copy Transfer and Express Transfer

Bulk-copy and express transfers initiate a direct transfer of data between two databases from the client application.

You can use a bulk-copy or express **transfer** statement to copy large amounts of data between similar tables. The express transfer feature transfers data faster than bulk-copy transfer, and because it uses the same syntax as the bulk-copy transfer, you can use it without modifying your applications.

Syntax

```
transfer [with report]

{to | from } 'secondaryname userid password';

with {insert | replace | truncate| alter table} into tablename;

sourceselectstatement
```

where:

- **transfer** must begin all transfer statements.
- `with report` is an optional phrase specified in the first line of the **transfer** statement. It instructs the access service to return processing information to the client application.

  This information is returned as a result set consisting of a `VARCHAR` column and a single row. The row contains the number of rows transferred, rejected, and modified during processing.
- *{to | from}* indicates the direction of the transfer:
  - *to* specifies that the data is transferred from the primary database to the secondary database.
  - *from* specifies that the data is transferred from the secondary database to the primary database.
- *secondaryname userid password* is a three-part character string that provides the information needed to connect to the secondary database:
  - *secondaryname* is the name used to identify the secondary database and must be recorded in these files:
    - For bulk-copy transfer, in the UNIX `interfaces` file or in the Windows `sql.ini` file.
    - For express transfer, the *secondaryname* must match a datasource name (DSN) in the `odbc.ini` file.
  - *userid* and *password* must be valid on the secondary database. If the password is NULL, you can substitute an asterisk for *password* and it will be corrected to a NULL

when sent to the secondary connection. Exactly three tokens are sent to the secondary connection.

All of the elements of the character string must be enclosed in single or double quotes in the order shown.

- *with {insert|replace|truncate} into* specifies whether the data is appended onto the target table (*insert*) or the existing data is deleted and replaced (*replace* or *truncate*).

  Note: When transferring data to Adaptive Server, the **truncate** option causes **transfer** to issue a **truncate** rather than a **delete** against the target table. For other target databases, **delete** and **truncate** are equivalent.

- *{with alter table} into* invokes a UDB command that disables logging for the transaction and truncates the table (for DB2 UDB only).

  Note: You must use this transfer syntax carefully: Errors render the table useless, and requiring restoration according to IBM's procedures. Read the IBM documentation pertaining to the **alter table** command and its option called `activate not logged initially`.

- *tablename* specifies the table into which data is inserted or replaced. The table must already exist because the **transfer** statement does not create a new one in the target database.
- **sourceselectstatement** specifies a SQL statement that is executed against the source database to produce the result set used in the transfer.

  This statement can be any statement the source database accepts, including stored procedures. SQL transformation is not performed on the **sourceselectstatement**. It must be in the source database SQL dialect.

See also

- *Express Transfer* on page 102

Bulk-Copy and Express Transfer Conditions

The conditions that allows you to determine type of **transfer** you want to select.

Table 4. When to Use Bulk-Copy and Express Transfers

Bulk-Copy Transfer	Express Transfer
When the source or destination database is Adaptive Server and does not use an Adaptive Server ODBC driver.	When you use ODBC drivers for both the source and destination database.
To exercise more control over transfer.	To execute the transfer quickly.
When you require diverse datatype conversions.	When you do not require any datatype conversions.

Limitations for Bulk-Copy and Express Transfers

- The **transfer** statement must be the only statement in a request.
- The table (the target) into which you want to transfer data must already exist because the **transfer** statement does not create new tables.
- The structure of the target table must match the structure of the source table.
- For bulk-copy and express transfer to work, the `secondaryname` to the secondary database must be recorded in these files:
  - For bulk-copy transfer, the Windows `sql.ini` file or the UNIX `interfaces` file.
  - For express transfer, the `secondaryname` must match a datasource name (DSN) in the `odbc.ini` file.
- Unicode datatypes are not supported.
- The first 32K of long character and long binary values are supported. Transfer processing truncates longer values without any warning.

Transfer Process

Transfer process for bulk-copy and express transfer from the client application through the access service to the database.

1. The client application initiates a transfer request.
2. The access service receives the transfer request and executes the **sourceselectstatement** against the source database to retrieve the schema of the result set, including column datatypes, length, precision, and scale.
3. The access service queries the target table for a description of the target table columns and compares this information to the structure of the result set with this criteria:
   - The target table must have at least as many columns as the result set.
   - The datatype of each result set column must be able to be converted to the datatype of the target column.

   If either of these tests fails, the access service stops processing the transfer and issues an error message.
4. If the **transfer** statement includes the **with replace** or **truncate** clause, the access service deletes data in the target table, provided the user ID of the person executing the request is authorized to do so. If the user ID is not authorized, the transfer fails.
5. The access service maps the columns from the result set to the columns in the target table in the same order. The access service attempts to insert NULL values (if allowable) in all columns in the target table that does not have corresponding columns in the result set.
6. The access service prepares an **insert** or equivalent bulk load statement for execution against the target table.
7. For bulk-copy transfer only, if conversion errors occur as rows are inserted (for example, a value is out of range), the invalid rows are handled according to the values set in these properties:

- `CharConvertError`
- `NumConvertError`
- `DatetimeConvertError`
- `DefaultDate`
- `DefaultTime`
- `DefaultNum`

The transfer continues processing. If the `SendWarningMessages` property is set to yes, the access service sends a warning message to the client application.

Express Transfer

Express transfer uses the same syntax as bulk-copy transfer; as a result, you do not need to modify your applications.

To improve performance transferring bulk data between datasources, Sybase has a bulk-copy transfer called “express transfer,” which uses ODBC bulk APIs to transfer data faster than bulk-copy transfer.

Example: To transfer data between Microsoft SQL Server and Adaptive Server requires an ODBC driver from Microsoft for the Microsoft SQL Server datasource and the Adaptive Server ODBC driver by Sybase.

To use this feature for Windows DB2 UDB, Microsoft SQL Server, or other ODBC-accessible datasources, you need to obtain a back-end ODBC driver. As of version 15.0, Sybase no longer provides back-end (server-side) drivers for them. However, Sybase does support these drivers:

- IBM DB2 CLI ODBC Driver for DB2 databases
- Data Direct Driver for DB2 and Microsoft SQL Server databases
- Microsoft driver for SQL Server databases, which you can download at no cost

See the *Enterprise Connect Data Access Installation Guide* for your platform for information about the ODBC drivers and driver managers.

Differences Between Bulk-Copy and Express Transfers

The functional differences between bulk-copy and express transfer statements in datatype conversion and error handling.

- Datatype conversion
  - In bulk-copy transfer, data being transferred is converted into intermediate Open Server datatypes.
  - In express transfer, the ODBC driver converts the datatypes automatically, and no conversion takes place in ECDA Option for ODBC.
- Error handling
  - If a datatype conversion error occurs, bulk-copy transfer supports error handling.

- Express transfer has only simple error handling: If an error occurs, it aborts the entire batch of rows if `BulkCommitCount` is greater than 1.

See also

- *Bulk-Copy and Express Transfer Errors* on page 108

Preparing to Use Express Transfer

Prepare to use the bulk-copy transfer, which can be interpreted as express transfer statements.

1. Set the `TransferExpress` property to yes, which causes the bulk-copy transfer statements to be interpreted as express transfer statements.
2. Enter the ODBC datasource name as the secondary name in the transfer statement.

   Express transfer is certified for target combinations, such as:
   - DB2 UDB z/OS, to and from Microsoft SQL Server, requires:
     - ECDA Option for ODBC
     - IBM DB2 CLI ODBC driver
   - DB2 UDB z/OS, to and from Adaptive Server, requires:
     - ECDA Option for ODBC
     - IBM DB2 CLI ODBC driver
     - Adaptive Server ODBC driver by Sybase
   - Microsoft SQL Server, to and from Adaptive Server, requires:
     - ECDA Option for ODBC
     - Microsoft SQL Server ODBC driver
     - Adaptive Server ODBC driver by Sybase

   The Adaptive Server ODBC Driver by Sybase is available on Windows and Linux platforms. For ECDA Option for ODBC on UNIX platforms, you can purchase a UNIX Adaptive Server driver from DataDirect Technologies. Obtain more information about the DataDirect drivers at the *DataDirect Web site http://www.datadirect.com/index.html*.

   Note: When you use an Adaptive Server driver, make the Adaptive Server connection the secondary connection. Use the Adaptive Server datasource name (DSN) for the secondary name.

Using Express Transfer from DB2 to Microsoft SQL Server

Use express transfer to transfer data from DB2 to Microsoft SQL Server.

Use an **isql** connection to DB2 through ECDA Option for ODBC (primary server), and an ODBC DSN (on the primary server machine) for a Microsoft SQL Server target called “MSQL-DSN.”

1. Connect to DB2 through the primary server:

   ```
   -isql -Sdcdb2udb - Uuserid -Ppassword
   ```

2. Set `TransferExpress` to yes.
3. Enter:

```
Transfer to ‘MSQL-DSN userid password’;
with insert into MSQL-table;
Select * from db2-table
```

Datatype Conversion for Express Transfer Statements

Acceptable source datatypes that the access service can convert into corresponding target destination datatypes.

If a column match is incompatible, the transfer ends with an error. If data conversion errors occur, you should try swapping the connections to the drivers used for the primary server and the secondary server.

Note: The datatype conversions identified may not be available for express transfer. Availability varies depending on the ODBC driver that is used.

Table 5. Datatype Conversions

Source Datatypes	Target Datatypes				
	CHAR **VARCHAR** **LONGVARCHAR** **TEXT (CLOB)**	**BIGINT** **DECIMAL** **DOUBLE FLOAT** **INTEGER** **MONEY** **MONEY4** **NUMERIC** **SMALLINT** **REAL** **TINYINT**	**DATETIME** **DATETIME4** **TIMESTAMP**	**BINARY** **VARBINARY** **LONGVARBINARY** **IMAGE (BLOB)**	**BIT**
CHAR VARCHAR LONGVARCHAR TEXT (CLOB)	x	x	x	x	x
BIGINT DECIMAL DOUBLE FLOAT INTEGER MONEY MONEY4 NUMERIC REAL SMALLINT TINYINT	x	x			

Source Datatypes	**Target Datatypes**				
	CHAR **VARCHAR** **LONGVARCHAR** **TEXT (CLOB)**	**BIGINT** **DECIMAL** **DOUBLE FLOAT** **INTEGER** **MONEY** **MONEY4** **NUMERIC** **SMALLINT** **REAL** **TINYINT**	**DATETIME** **DATETIME4** **TIMESTAMP**	**BINARY** **VARBINARY** **LONGVARBINARY** **IMAGE (BLOB)**	**BIT**
BINARY VARBINARY IMAGE (BLOB) LONGVARBINARY	x			x	
DATETIME DATETIME4 TIMESTAMP	x		x		
BIT	x	x			x

Bulk Copy Values Processing

The guidelines that are applicable for character, numeric, date, and binary datatype values.

Warning! The DirectConnect server cannot correctly transfer `varchar` values that contain empty strings (zero length non null strings), in the bulk-copy transfer process. Empty string `varchar` values are interpreted as `NULL` values.

Character Datatypes

Character datatypes (`CHAR`, `VARCHAR`, `TEXT`) can be converted to any other datatype.

Conversely, every datatype can be converted to character data. You must verify that the character string is able to be converted to the target datatype.

Example

The character string "450" can be converted to other numeric datatype such as `INTEGER` or `DECIMAL`, but the character string "Hello" causes a value error going to a numeric datatype.

Numeric Datatypes

Numeric datatypes can be converted to other numeric datatypes or to character datatypes, but they cannot be converted to binary or date datatypes.

Additional guidelines:

- All numeric conversions use rounding.
- Any loss of digits to the left of the decimal results in an error. For example, an integer of value `123` cannot be converted to a decimal (4,2) value without losing a digit to the left of the decimal point.
- Any loss of digits to the right of the decimal point is not considered an error. For example, a float of value `123.456` is converted to an integer of value `123` without an error.
- When you transfer data from a `decimal` column to a `float` column, the precision of the result is not better than the precision of the target column.

  For example, if you transfer data from a `decimal(15,0)` column to a `float` column, then back to a `decimal(15,0)` column, the results in the target `decimal(15,0)` column do not match the results of the source `decimal(15,0)` column, due to the float column precision.

Date Datatypes

Date datatypes can be converted to other date datatypes or to character strings.

However, they cannot be converted to numeric or binary datatypes.

Binary Datatypes

Binary datatypes can be converted to other binary datatypes or to character datatypes.

However, they cannot be converted to numeric or date datatypes.

Bulk-Copy and Express Transfer Errors

Bulk-copy transfer errors and express transfer errors are handled in different ways.

- Express transfer does not support errors regarding individual rows. If an error occurs, the entire transfer is aborted. Any value errors that occur are handled by the ODBC driver.
- Bulk-copy transfer errors are handled by the access service. Value errors occur during transfer processing when the value being inserted is out of range for the column datatype.

Bulk-Copy Value Processing Rules

The rules that are applicable for the bulk-copy transfer values.

- NULL values:

  If a source column contains NULL values but the destination does not allow them, the row is rejected.
- Numeric data:
  - All numeric conversions use rounding.
  - Any loss of digits to the left of the decimal results in an error. For example, an integer of value `123` could not be converted to a `decimal(4,2)` value without losing a digit to the left of the decimal point.
  - Any loss of digits to the right of the decimal point is not considered an error. For example, a `float` of value `123.456` would be converted to an integer value of `123` without error.
- Binary data:

  Binary data is transferred to a binary column without byte translation. A byte value in the source will have the same value in the destination.

Values That Cause Errors

Data values that can cause errors during bulk-copy transfer.

Source Input Datatypes	Target (destination) Datatypes	Error Conditions for Bulk Copy
`char, varchar, text`	`char, varchar, text, binary, varbinary, or image, LONG-VARCHAR`	Source data is longer than the destination column.

Source Input Datatypes	Target (destination) Datatypes	Error Conditions for Bulk Copy
`char, varchar, text`	`decimal, DECIMAL`	• Source is not a valid decimal string (must contain an optional leading sign and decimal digits). • Number of digits to the left of the decimal point is greater than the destination column precision minus scale. Any digits to the right of the decimal are lost without error.
`char, varchar, text`	`integer`	• Source is not a valid decimal string (must contain an optional leading sign, decimal digits, decimal point, and fractional decimal digits). • String of digits to the left of the decimal point is greater than `2147483647` (positive values) or less than `-2147483648` (negative values). Any digits to the right of the decimal are lost without an error being generated.
`char, varchar, text`	`smallint`	• Source is not a valid decimal string (must contain an optional leading sign, decimal digits, decimal point, and fractional decimal digits). • String of digits to the left of the decimal point is greater than `32767` (positive values), or it is less than `-32768` (negative values).
`char, varchar, text`	`tinyint`	• Source is not a valid positive integer string (must contain an optional leading "**+**" and decimal digits). • String of digits does not form an integer value between `0` and `255`.
`char, varchar, text`	`OS bit`	Source data length is greater than `1` or source value != "`0`" or "`1`."
`char, varchar, text`	`float. DOUBLE`	Source is not a valid floating point format string (must contain optional leading sign, decimal digits, optional decimal point, fractional decimal digits, and optional `E[+|-] nnn` exponent).

Source Input Datatypes	Target (destination) Datatypes	Error Conditions for Bulk Copy
`char, varchar, text`	`real`	• Source is not a valid floating point format string (must contain optional leading sign, decimal digits, optional decimal point, fractional decimal digits, and optional `E[+|-] nnn` exponent). • Target (destination) is Adaptive Server, and the value is greater than `3.402823466E38` or less than `-3.402823466E38`.
`char, varchar, text`	`date`	Source is not an ISO format date (`YYY-MM-DD`) or a valid Adaptive Server date/time string with the year later than `1753`.
`char, varchar, text`	`time`	Source is not an ISO format time (`HH.MM.SS`) or an `HH:MM:SS` format time.
`char, varchar, text`	`ODBC TIMESTAMP`	Source is not an ISO format date (`YYYY-MM-DD-HH.MM.SS`) or a `YYYY-MM-DD-HH.MM.SS.NNNNNN` date with `YYYY` greater than `0001`, or a valid Adaptive Server date/time string with the year later than `1753`.
`char, varchar, text`	`datetime`	Source is not an ISO format date, time, or timestamp with a year later than `1753` or a valid Adaptive Server date/time string with a year later than `1753`.
`char, varchar, text`	`datetime4`	Source is not an ISO format date, time, or timestamp with a year later than `1899` and the year, month, and day earlier than `Jun 7, 2079`, or a valid Adaptive Server date/time string with a year later than `1899` and the year, month, and day earlier than `Jun 7, 2079`.
`char, varchar, text`	`money`	Source is not a valid decimal string (must contain an optional leading sign, decimal digits, optional decimal point, and fractional decimal digits), and the value is greater than `922337203685477.5807`, or the value is less than `-922337203685477.5808`.
`char, varchar, text`	`money4`	Source is not a valid decimal string (must contain an optional leading sign, decimal digits, optional decimal point, and fractional decimal digits), and the value is greater than `214748.3647` or less than `-214748.3648`.

Source Input Datatypes	Target (destination) Datatypes	Error Conditions for Bulk Copy
`binary, varbinary, image`	`char, varchar, text, binary, varbinary, image, LONGVARCHAR`	Source data is longer than the destination column.
`byte, int, smallint`	`char, varchar, text`	Destination column is too small to hold the digits required to express the value. For example, the source value is `103` and the destination column is `char(2)`.
`byte, int, smallint`	`bit`	Source value !=`0` or `1`.
`smallint, int, float, real, money, money4, decimal`	`decimal`	Destination column precision minus scale is too small to hold the value. For example, a source data value of `98` requires destination column precision minus scale of `2`.
`money, money4, decimal, numeric`	`decimal`	If precision=scale=maximum, precision for the datatype does not transfer properly when the data value is `0`. A workaround is to alter the table to avoid one of these conditions.
`smallint`	`tinyint`	Source value is greater than `255` or less than `0`.
`int`	`smallint`	Source is greater than `32767` or less than `-32768`.
`int`	`money4`	Source is greater than `214748` or less than `-214748`.
`int`	`tinyint`	Source is greater than `255` or less than `0`.
`bit`	`decimal`	Target (destination) column precision minus scale is less than `1`.
`float, real`	`char, varchar, text`	Target (destination) column is too small to hold the digits required to express the value. For example, source is `1030303E+30` and destination column is `char(12)`.
`float, real, money`	`int`	Source value greater than `2147483647.0` or less than `-2147483648.0`.

Source Input Datatypes	Target (destination) Datatypes	Error Conditions for Bulk Copy
`float, real, money, money4, decimal`	`smallint`	Source is greater than `32767.0` or less than `-32768.0`.
`float, real, money, money4, decimal`	`tinyint`	Source is greater than `255.0` or less than `0.0`.
`float, real`	`money`	Source value is greater than `922337203685477.0` or less than `-922337203685477.0`. • Since `float` accuracy is 15 digits, a value of this magnitude is accurate only to the nearest dollar. • Since `real` accuracy is 7 digits, a value of this magnitude is accurate only to the nearest hundred million dollars.
`float, real, money, decimal`	`money4`	Source value is greater than `214748.3647` or is less than `-214748.3648`.
`float, real, money, money4, decimal`	`bit`	Source value !=`0.0` or `1.1`.
`money, money4, decimal`	`char, varchar, text`	Target (destination) column is too small to hold digits required to express value. For example, source is `100000000.001` and destination column is `char(12)`.
`datetime, datetime4`	`char, varchar, text`	Target (destination) column length is less than `19`.
`datetime`	`datetime4`	Date portion of source value is earlier than `Jan 1 1900` or later than `Jun 6 2079`.

Bulk-Copy Transfer Error Reporting

Obtain bulk-copy transfer error information.

- Including the **with report** phrase in the **transfer** statement, you receive a result set containing one `VARCHAR` column and one row indicating the number of rows transferred, rejected, and modified during processing.

- Executing **@@RejectedRowCount** or **@@DefaultedRowCount** immediately after a successful transfer. These global variables return the number of rejected or defaulted rows.
- Setting `SendWarningMessages` to yes, so the access service returns data conversion errors to the client application.

Destination-Template Transfer Statement

The destination-template transfer statement allows you to transfer data and send a result set into a template.

The destination-template transfer statement lets you:

- Create a full-image copy (**insert**)
- Create an incremental copy (**insert**)
- Modify column values (**update** or **delete**)
- Perform structural modifications (**create** or **alter**)
- Change database permissions (**grant** or **revoke**)
- Execute remote stored procedures (**execute** or **use**)
- Execute other arbitrary SQL statements

Destination-Template Transfer Processing

During a destination-template transfer, the access service inserts the data values it retrieves with the **sourceselectstatement** from the source database into the destination-template SQL clause.

This clause contains one question mark for each column in the result set of the **sourceselectstatement**. Each value from the result set is substituted for the corresponding question mark in the template on a row-by-row basis. The access service executes the resulting statement against the target database.When you use the destination-template **transfer** statement, these restrictions apply:

- The **transfer** statement must be the only statement in a request.
- The table into which you want to transfer data (the target) must already exist. The **transfer** statement does not create new tables in the transfer target.
- The structure of the target table must match the structure of the source table.
- For any transfer to work, the `ConnectionSpec` to the secondary connection must be recorded in the `sql.ini` file for Windows or in the `interfaces` file for UNIX.

Warning! The DirectConnect server cannot correctly transfer `varchar` values containing empty strings (zero length non null strings) in the destination-template transfer process. Empty string `varchar` values are interpreted as NULL values.

Syntax

```
transfer [with report]
{to | from} ‘secondaryname userid password’;
```

```
sourceselectstatement;
destinationtemplatestatement
```

where:

- **transfer** must begin all **transfer** statements.
- **with report** is an optional phrase specified in the first line of the **transfer** statement that instructs the access service to return processing information to the client application.

  This information is returned as a result set that consists of one `VARCHAR` column and one row. The row contains the number of rows transferred, rejected, or modified during processing.
- {*to* | *from*} indicates the direction of the transfer:
  - *to* specifies that the data is transferred from the primary database to the secondary database.
  - *from* specifies that the data is transferred from the secondary database to the primary database.
- *'secondaryname userid password'* is a character string that provides the information needed to connect to the secondary database:
  - *secondaryname* is the name used to identify the secondary database.
  - *userid* and *password* must be valid on the secondary database. You can substitute an asterisk for *password*, if a password is not specified.

  All of the elements of the character string must be enclosed in single or double quotes in the sequence.
- **sourceselectstatement** specifies a SQL statement that is executed against the source database to produce the result set that will be used in the transfer. This statement can be of any complexity acceptable to the source database, including stored procedures. SQL transformation is not performed on the **sourceselectstatement**. The transformation must be in the source database SQL syntax.
- **destinationtemplatestatement** is a SQL statement or any statement that is valid for the target database environment where it executes. SQL transformation is not performed on the **destinationtemplatestatement**. The transformation must be in the destination database SQL syntax.

  This statement can include question marks as placeholders for the data values that will be inserted. It can also include qualifiers to specify datatypes for the question mark placeholders in the **destinationtemplatestatement**.

To increase processing efficiency, you can batch destination-templates together for processing. Use the `TransferBatch` configuration property or a **set** statement to specify the number of templates to batch.

Datatype Qualifiers

Qualifiers tell the access service how to format data that is inserted for a placeholder.

If you do not supply a qualifier, the access service applies default transformations.

Qualification is required for date and time values. You can use the `?T`, `?t,` `?D`, and `?d` qualifiers for dates, or you can create a custom qualifier using special qualifiers.

Table 6. Destination-Template Transfer Datatype Qualifiers

Placeholder/ Qualifier	Definition
`?C`	Character string enclosed in quotes
`?N`	Numeric data, no quotes
`?D`	Standard format Adaptive Server datetime data enclosed in quotes
`?T`	Standard format ISO TIMESTAMP data enclosed in quotes, ‘`YYYY-MM-DD-hh.mm.ss.nnnnnn`’
`?d`	‘`mm/dd/yyyy`’
`?t`	‘`hh:mm:ss`’
`?X`	Standard format Adaptive Server hexadecimal data (for example, 0xffee) used for transferring binary data
`?x`	Standard format hexadecimal data (for example, X’FFEE’) used for transferring binary data
`?y`	‘`yy/mm/dd`’
`?G`	G‘<...>’ used for transferring graphic datatypes or formatting a graphic constant from binary character data
`?g`	GX‘<...>’ used for transferring graphic datatypes or formatting a graphic constant from binary character data (returns data in hexadecimal format)

Note: For each table, special circumstances are detailed in the text.

See the effects of the `?C`, `?N`, `?D`, and `?T` qualifiers.

Table 7. Effects of Qualifiers on Datatypes (1)

Open Server Datatype	Default	Effects (by Qualifier)			
		?C	?N	?D	?T
CS_CHAR CS_VARCHAR CS_TEXT	`?C`	Quote	No quote	Convert to Open Server `date-time` string, quote	Convert to ISO TIMESTAMP, quote
CS_BIT, CS_INT1 CS_INT2 CS_ INT4 CS_REAL CS_FLOAT	`?N`	Convert to `char`, quote	Convert to `char`, no quote	n/a	n/a
CS_MONEY CS_MONEY4 CS_DECIMAL	`?N`	Convert to `char`, quote	Convert to `char`, no quote	n/a	n/a
CS_DATETIME CS_DATETIME4	`?D`	'`MON DD YYYY hh:mm`' [AM or PM]	n/a	'`MON DD YYYY hh:mm:ss:nnn`'	'`YYYY-MM-DD-hh.mm.ss.nnnnnn`'

For CS_CHAR, CS_VARCHAR, and CS_TEXT used with the `?D` qualifier:

- If the source is an ISO TIMESTAMP, it is converted to ‘`Mon dd yyyy hh:mm:ss:nnn`’.
- If it is an ISO DATE, it is converted to ‘`Mon dd yy`’.
- If it is an ISO TIME, it is converted to ‘`Mon dd yy hh:mm:ss`’ using the value from the `DefaultDate` property as the date portion of the value.

For CS_CHAR, CS_VARCHAR, and CS_TEXT used with the `?T` qualifier, if the source is an ISO DATE or TIME, the `DefaultDate` and `DefaultTime` property values are used to fill in missing information.

See the effects of the `?y`, `?d`, `?t`, and `?x` qualifiers.

Table 8. Effects of Qualifiers on Datatypes (2)

Open Server Datatype	Default	Effects (by Qualifier)			
		?y	?d	?t	?x
CS_CHAR CS_VARCHAR CS_TEXT	`?C`	Convert and quote	Convert and quote	Convert and quote	Convert to hex; leading X' trailing '. For example, X'ab70'
CS_BIT, CS_INT1 CS_INT2 CS_INT4 CS_REAL CS_FLOAT CS_MONEY CS_MONEY4 CS_DECIMAL	`?N`	n/a	n/a	n/a	n/a
CS_DATETIME CS_DATETIME4	`?D`	'yy/mm/dd'	'mm/dd/yyyy'	'hh:mm:ss'	n/a
CS_BINARY CS_VARBINARY CS_IMAGE	`?X` or `?x`	n/a	n/a	n/a	Convert to hex; leading X' trailing '. For example, X'ab70'

For CS_CHAR, CS_VARCHAR, and CS_TEXT used with the `?y`qualifier, if the source is an ISO `DATE`, `TIME`, or `TIMESTAMP`, it is converted to '`yy/mm/dd`'.

For CS_CHAR, CS_VARCHAR, and CS_TEXT used with the `?d` qualifier, if the source is an ISO `DATE`, `TIME`, or `TIMESTAMP`, it is converted to '`mm/dd/yy`'.

For CS_CHAR, CS_VARCHAR, and CS_TEXT used with the `?t` qualifier, if the source is an ISO `DATE`, `TIME`, or `TIMESTAMP`, it is converted to '`hh:mm:ss`'.

For all datatypes used with the `?x` qualifier, if the target database is ODBC, `?x` converts the data to the standard ODBC hexadecimal format (a quoted hexadecimal number with a leading X).

See the effects of the `?x` and `?O` qualifiers.

Table 9. Effects of Qualifiers on Datatypes (3)

Open Server Datatype	Default	Effects (by Qualifier)
		?X
CS_CHAR CS_VARCHAR CS_TEXT		Convert to hex; leading 0x, no quote
CS_BIT CS_ INT1 CS_INT2 CS_INT4 CS_REAL CS_FLOAT CS_MONEY CS_MONEY4 CS_DECIMAL	`?N`	n/a
CS_DATETIME CS_DATETIME4	`?D`	n/a
CS_BINARY CS_VARBINARY CS_IMAGE	`?X` or `?x`	Convert to hex; leading 0x, no quote

For CS_BINARY, CS_VARBINARY, and CS_IMAGE datatypes used with the `?X` qualifier, if the target database is ODBC, `?x` converts the data to the standard ODBC hexadecimal format (a quoted hexadecimal number with a leading X).

Special Date and Time Qualifiers

You can combine special date and time qualifiers and construct the date or time format that the target database requires.

Rules:

- Enter special characters in either uppercase or lowercase.
- Separate special characters by any arbitrary character, such as a hyphen, slash, or space. Any unrecognized character is copied to the target as is.

- Enclose special characters in single or double quotes, because the resulting value is passed to the target as a character string.
- Allow qualifiers to contain a null terminated string. The string is limited only by the buffer size (1KB).

Table 10. Qualifier Definitions

Qualifier	Definition
`yy`	Last two digits of year.
`yyyy`	All four digits of year.
`mm`	Month or minute (recognized by context).
`mon`	Three-character month abbreviation: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec.
`dd`	Day.
`hh`	Hour.
ss	Seconds.
`nnn`	Milliseconds.
`nnnnnn` or `uuuuuu`	Microseconds.
`am` or `pm`	Indicates that you want an AM or PM designator included. The actual designator is inserted appropriate to the value.

Destination-Template Processing

A service library processes destination-template processes using **transfer from** and **transfer to** statements.

Transfer from Statements

Use a service library to process a destination-template **transfer from** statement from Adaptive Server to ODBC.

1. The access service executes the **sourceselectstatement** against the primary database and retrieves the schema of the result set.
2. The access service inserts the first `n` rows of data (where `n` is the setting of the `TransferBatch` property) resulting from the **sourceselectstatement** into the destination-template. The access service formats the data according to the specified qualifiers and groups the statements into a request.
3. The access service executes the request against the primary database.

4. The access service substitutes the next *n* rows and executes another request. It repeats this process until all the rows finish processing.
5. If conversion errors occur as rows are inserted, the invalid rows are handled according to the values set in the `CharConvertError`, `NumConvertError`, `DatetimeConvertError`, `DefaultDate`, `DefaultNum`, and `DefaultTime` properties, and the transfer continues. If `SendWarningMessages` is set to yes, a warning message is sent to the client.

Transfer to Statements

Use a service library to process a destination-template **transfer to** statement to a target from ODBC.

1. The access service issues the **sourceselectstatement** against the primary database and receives the schema of the result set.
2. The access service receives the results of the **sourceselectstatement** and converts them to the predefined Adaptive Server datatypes. These datatypes do not necessarily match the datatypes of the columns in the destination table.
3. The access service substitutes the first `n` rows of the result set (where `n` is the number of rows specified in the `TransferBatch` configuration property). The access service formats the data according to the specified datatype qualifiers and batches the statements into a request.
4. The access service issues the request against the secondary database.
5. The access service substitutes the next `n` rows into the **destinationtemplatestatement** and issues another request against the secondary database. It repeats this process until all the rows finish processing.
6. If conversion errors occur as rows are inserted, the invalid rows are handled according to the values set in the `CharConvertError`, `NumConvertError`, `DatetimeConvertError`, `DefaultDate`, `DefaultNum`, and `DefaultTime` properties, and the transfer continues. If `SendWarningMessages` is set to yes, a warning message is sent to the client.

Datatype Conversion for Transfer to Statements

The incoming target ODBC datatypes are initially converted to Open Server datatypes.These datatypes are converted into the datatypes specified by the **destinationtemplatestatement** datatype qualifiers.

ODBC Datatype	Open Server Datatype
CHAR	CS_CHAR
VARCHAR	CS_CHAR
LONGVARCHAR	CS_TEXT
SMALLINT	CS_SMALLINT

ODBC Datatype	Open Server Datatype
INT	CS_INT
DECIMAL	CS_DECIMAL
DOUBLE	CS_FLOAT
REAL	CS_REAL
FLOAT	CS_FLOAT
DATE	CS_CHAR
TIME	CS_CHAR
TIMESTAMP	CS_CHAR
BINARY	CS_BINARY
VARBINARY	CS_VARBINARY
LONGVARBINARY	CS_IMAGE
TINYINT	CS_TINYINT
BIT	CS_BIT
BIGINT	CS_FLOAT
NUMERIC	CS_NUMERIC

Destination-Template Transfer Errors

Value errors occur during transfer processing when the value being inserted is out of range for the column’s datatype.

The access service uses these properties to handle value errors:

- `CharConvertError`
- `NumConvertError`
- `DateTimeConvertError`
- `DefaultDate`
- `DefaultTime`
- `DefaultNum`

If the `SendWarningMessages` property is set to yes, the access service sends a message to the client application when it encounters value errors. You can also use these properties to fill in default values when datatype conversion fails during both types of transfer.

Obtaining Error Information

Obtain destination-template transfer errors.

Use one of these methods:

1. If you include **with report** in the **transfer** statement, you receive a result set containing one `VARCHAR` column and one row indicating the number of rows transferred, rejected, and modified during processing.

   Execute **@@RejectedRowCount** or **@@DefaultedRowCount** immediately after a successful transfer. These global variables return the number of rejected or defaulted rows.
2. If you set `SendWarningMessages` to yes, the access service returns data conversion errors to the client application.

   During processing, the access service sets the `StopCondition` property to none. It uses the value in the `TransferErrorCount` property to determine the number of error rows it allows before it stops processing.

   Set this value with this statement:

   ```
   set TransferErrorCount nnn
   ```

   The default setting is `0`, which causes the access service to ignore errors.

Transfer RPC

The client application can create a stored procedure within Adaptive Server that invokes the access service library **transfer** function.

The access service library receives the **transfer** command as an RPC event with these arguments:

- Name of the RPC – “transfer”
- Argument 1 – The secondary connection information ({to | from} “server userid pw”)
- Argument 2 – Either the bulk copy target command or the destination-template **sourceselectstatement**.
- Argument 3 – Either the bulk copy **sourceselectstatement** or the destination-template **sourceselectstatement**.

Example 1: Outlines how a stored procedure shows a bulk copy transfer statement to be received as an RPC:

```
create procedure replauth as
begin
execute servername. . .transfer
‟to ‘servername2 userid password’;”,
 ‟with replace into authors;”,
```

```
 "select * from authors;"
end
```

where:

- *servername* specifies the access service to handle the transfer. In addition:
  - The double quotes ("...") after the *servername* are required.
  - The access service library recognizes anything other than "transfer" in the next position as the name of an ODBC stored procedure.
- *servername2* specifies the secondary database for the **transfer** command.

Executing a Transfer RPC

A client can log in to the Adaptive Server on which a procedure is defined and run the transfer RPC created.

1. Log in to Adaptive Server.
2. Run:

   ```
   execute replauth
   ```

   When Adaptive Server executes **replauth**, it passes an RPC to the access service. The access service returns any result rows or messages to the client application, not to Adaptive Server.

Catalog Stored Procedures

Catalog stored procedure (CSPs) are specially recognized commands that return catalog information.

Client applications use CSPs instead of SQL to access information contained in the system catalog of the target database. The access service library implements CSPs by executing stored procedures against the target catalog.

When you invoke a CSP, the access service executes a stored procedure that returns a result set.

CSP attempts to match the results an Adaptive Server would return under the same circumstances. Because the ODBC catalog is significantly different from the Adaptive Server catalog, an exact match is impossible. To make the results match, the access service performs additional computations as the rows are returned.

Because some information is static, **sp_datatype_info** uses memory tables to improve operation speed.

You can execute CSPs using a language command or an RPC event.

When the access service processes a CSP as an RPC event, it retrieves the name and parameters from the client application using standard RPC processing techniques.

Syntax

CSP syntactical rules:

- Arguments can be delimited with commas and identified by position:
  ```
  sp_columns parm1,parm2
  ```
- Arguments can be identified using the keyword `NULL`:
  ```
  sp_columns NULL, NULL, parm3
  ```
- Empty arguments can be identified using empty strings:
  ```
  sp_columns ’ ’,’ ’,smith
  ```
- Arguments can be named using the syntax `@name=parm`:
  ```
  sp_columns @table_owner=smith
  ```

The positional forms (first three bullets) cannot be mixed with the named form (fourth bullet).

The access service library does not support the `TABLE_QUALIFIER` or `PROCEDURE_QUALIFIER` parameters. For all CSPs, leave the parameter empty or set it to `NULL`.

See the *Adaptive Server Enterprise Reference Manual* for details.

Treatment of Special Characters

The access service supports only the “%” (percent) wildcard character, which can be used in parameters that allow wildcard-character search patterns.

The character represents any string of zero or more characters.

The access service treats all underscore characters as literals.

ODBC Information

The ODBC definition of a CSP is the model, when the access service cannot support a particular procedure, it returns the expected column form descriptors, with no data rows.

See the *Microsoft ODBC 3.5 Programmer’s Reference and SDK Guide* for the format and content of results returned by most CSPs.

Compatibility

To maintain compatibility with Adaptive Server and previous versions of the MDI Database Gateway, all CSPs accomplish the same tasks as their counterparts in the other systems. However, in ECDA Option for ODBC:

- The result sets returned conform to ODBC requirements.
- The result sets returned conform to Adaptive Server/CIS requirements.

When a CSP requirements conflict arises, these rules are used, in this order:

1. Make the results conform to ODBC specifications.
2. Make the results conform to Adaptive Server/CIS specifications.
3. Make the behavior of the procedure conform to its counterpart in Adaptive Server.

ODBC Conformance Levels

The conformance levels for ODBC drivers are core, level one, and level two.

CSPs that support the functionality in levels one and two are:

- Level one
  - **sp_columns**
  - **sp_datatype_info**
  - **sp_special_columns**
  - **sp_statistics**
  - **sp_tables**
- Level two
  - **sp_column_privileges**
  - **sp_fkeys**

- **sp_pkeys**
- **sp_sproc_columns**
- **sp_stored_procedures**
- **sp_table_privileges**

sp_column_privileges

Returns column privilege information for one or more columns in a table or view.

Note: This stored procedure is not supported in ECDA Option for ODBC to DB2 UDB data sources.

Syntax

```
sp_column_privileges table_name [, table_owner
[, table_qualifier [, column_name]]]
```

Parameters

- **table_name –** is the name of the table. Wildcard-character search patterns are not supported.
- **table_owner –** is the name of the table owner. Wildcard-character search patterns are not supported.
- **table_qualifier –** is ignored. Leave blank or set to `NULL`.
- **column_name –** is the name of the column for which you want privilege information. Use wildcard-character search patterns to request information about more than one column. Leave blank or set to `NULL` to request information about all columns in the table or tables.

Usage

This procedure corresponds to the ODBC function called **SQLColumnPrivileges**.

sp_columns

Returns information about the type of data that can be stored in one or more columns.

Syntax

```
sp_columns table_name [, table_owner]
 [, table_qualifier] [, column_name]
```

Parameters

- **table_name –** is the table name or view. Use the wildcard character to request information about more than one table.
- **table_owner –** is the owner of the table or view. Use the wildcard character to request information about tables owned by more than one user.
- **table_qualifier –** is ignored. Leave blank or set to `NULL`.
- **column_name –** is the name of the column for which you want information. Use the wildcard character to request information about more than one column.

Usage

- This procedure returns the Adaptive Server datatype that most clearly matches the native datatype of the target, regardless of the current datatype properties.
- This procedure corresponds to the ODBC function **SQLColumns**.
- This procedure returns one row containing a description of each column in a table.
  - There are three columns in the result set that describe each datatype; `TYPE_NAME`, `DATA_TYPE`, and `REMOTE_DATA_TYPE`.
  - `TYPE_NAME` contains the ODBC datatype name and `DATA_TYPE` contains the ODBC integer identifier.
  - The `REMOTE_DATA_TYPE` column contains a 32-bit or 4-byte composite user datatype UDT specifically identifying the remote datatype.

  This procedure allows transmission of column datatypes using a target-specific-type ID. The `REMOTE_DATA_TYPE` column contains a 32-bit composite datatype defined by the access service.

ODBC Datatypes

The identifiers returned in the `TYPE_NAME` and `DATA_TYPE` columns in the result set for **sp_special_columns**.

ODBC Datatype (TYPE_NAME)	Target Data-type Length	Datatype	ODBC Type	Sybase Type
BINARY	254	(-2)	SQL_BINARY	CS_BINARY
VARBINARY	254	(-3)	SQL_VARBINARY	CS_VARBINARY
LONGVARBINARY	2^31	(-4)	SQL_LONGVARBINA-RY	CS_LONGBINARY
CHAR()	254	(1)	SQL_CHAR	CS_CHAR
VARCHAR	254	(12)	SQL_VARCHAR	CS_VARCHAR

ODBC Datatype (TYPE_NAME)	Target Data-type Length	Datatype	ODBC Type	Sybase Type
LONGVARCHAR	2^31	(-1)	SQL_LONGVARCHAR	CS_LONGCHAR
SMALLINT	2	(5)	SQL_SMALLINT	CS_SMALLINT
INTEGER	4	(4)	SQL_INTEGER	CS_INT
DOUBLE	8	(8)	SQL_DOUBLE	CS_FLOAT
FLOAT()	8	(6)	SQL_FLOAT	CS_FLOAT
REAL	4	(7)	SQL_REL	CS_REAL
DECIMAL()	17	(3)	SQL_DECIMAL	CS_DECIMAL
NUMERIC	17	(2)	SQL_NUMERIC	CS_NUMERIC
DATE	4	(9)	SQL_DATE	CS_DATE
TIME		(10)	SQL_TIME	CS_TIME
TIMESTAMP	10	(11)	SQL_TIMESTAMP	CS_DATETIME
TINYINT	1	(-6)	SQL_TINYINT	CS_TINYINT
BIGINT	19	(-5)	SQL_BIGINT	CS_FLOAT
BIT	1	(-7)	SQL_BIT	CS_BIT
GUID	36	-11	SQL_GUID	CS_CHAR
WCHAR	254	-8	SQL _WCHAR	CS_UNICHAR
WVARCHAR	254	-9	SQL _WVARCHAR	CS_UNICHAR
WLONGVARCHAR	230	-10	SQL _WLONGVARCH-AR	CS_UNITEXT

REMOTE_DATA_TYPE Return Value

The values returned in the `REMOTE_DATA_TYPE` columns of the result set.

Bits	Value Returned
Bits 0–7	ODBC datatype (can be extended for types not defined in ODBC).
Bit 8	`1` if nullable, `0` if not nullable.

Bits	Value Returned
Bit 9	`1` if case-sensitive, `0` if not case-sensitive.
Bits 10, 11	`10` (binary); ability to be updated unknown.
Bits 12, 13	Reserved; always returns `00` (binary). The access service bulk-copy feature uses this.
Bits 14, 15	`01` (binary); NEWODBCDATATYPE (used for all except REAL) `10` (binary); NEWUSERTYPE (used for REAL).
Numeric types: Bits 17–23 Bits 24–31	Precision. Scale.
Non-numeric types: Bits 16–31	Length.

sp_databases

Returns a list of databases on a target DBMS.

Syntax

```
sp_databases
```

Parameters

This procedure does not allow parameters.

sp_datatype_info

Returns information about a particular datatype or all supported datatypes.

Syntax

```
sp_datatype_info [data_type]
```

Parameters

- **data_type –** is the ODBC code number for the specified datatype for which **sp_datatype_info** returns information.

Usage

- If *data_type* is not provided, **sp_datatype_info** returns information about all supported datatypes.
- This procedure corresponds to the ODBC function **SQLGetTypeInfo**.
- The `DatatypeInfo` property specifies whether information is returned for Transact-SQL datatypes or target database datatypes.

  If the value for *data_type* equals:
  - **target**, the **sp_datatype_info** returns all target datatypes and their associated ODBC datatypes. A specific ODBC datatype may be used to represent multiple target datatypes.
  - **transact**, the **sp_datatype_info** returns the Transact-SQL datatype that best matches each ODBC datatype that the target represents.

See also

- *DatatypeInfo* on page 21

Results

sp_datatype_info returns a list of datatypes with information about each column.

Results are ordered by these columns:

- `DATA_TYPE`
- `TYPE_NAME`

The lengths for `varchar` columns shown in the result set tables are maximums; the actual lengths depend on the target database.

Table 11. Result Set for sp_datatype_info

Column	Datatype	Description
`TYPE_NAME`	`varchar`(128) NOT NULL	Name of the Transact-SQL datatype or the target database datatype that corresponds to the ODBC datatype in the `DATA_TYPE` column.
`DATA_TYPE`	`smallint` NOT NULL	ODBC datatype to which all columns of this type are mapped.
`PRECISION`	`int`	Maximum precision allowed for this datatype; NULL is returned for datatypes where precision is not applicable.
`LITERAL_PREFIX`	`varchar`(128)	Characters used to prefix a literal; NULL is returned for datatypes where a literal prefix is not applicable.

Column	Datatype	Description
`LITERAL_SUFFIX`	`varchar`(128)	Characters used to mark the end of a literal; NULL is returned for datatypes where a literal suffix is not applicable.
`CREATE_PARAMS`	`varchar`(128)	Description of the creation parameters required for this datatype (for example: precision and scale); NULL is returned if the datatype does not have creation parameters.
`NULLABLE`	`smallint` NOT NULL	Indicates whether the datatype accepts NULL values: • 0 means the column does not accept NULL values. • 1 means the column accepts NULL values.
`CASE_SENSITIVE`	`smallint` NOT NULL	Indicates whether the datatype distinguishes between uppercase and lowercase characters: • 0 means the datatype is not a character type or is not case-sensitive. • 1 means the datatype is a character type and is case-sensitive.
`SEARCHABLE`	`smallint` NOT NULL	Indicates how this datatype is used in **where** clauses: • 0 means the datatype cannot be used in a **where** clause. • 1 means the datatype can be used in a **where** clause.
`UNSIGNED_ATTRIBUTE`	`smallint`	Indicates whether this attribute is unsigned: • 0 means the datatype is signed. • 1 means the datatype is unsigned. • NULL means the datatype is not numeric.
`MONEY`	`smallint` NOT NULL	Indicates whether this is a money datatype: • 0 means it is not a money datatype. • 1 means it is a money datatype.

Column	Datatype	Description
AUTO_INCREMENT	smallint	Indicates whether this datatype automatically increments: • 0 means columns of this datatype do not automatically increment. • 1 means columns of this datatype automatically increment. • NULL means the column is not numeric and does not have a sign.
LOCAL_TYPE_NAME	varchar(128)	The database name or the Transact-SQL name for the datatype.
MINIMUM_SCALE	smallint	Minimum scale for the datatype; NULL if scale is not applicable.
MAXIMUM_SCALE	smallint	Maximum scale for the datatype; NULL if scale is not applicable.

sp_fkeys

Returns primary and foreign key information for the specified table or tables.

Foreign keys must be declared through the ANSI integrity constraint mechanism.

Syntax

```
sp_fkeys pktable_name [, pktable_owner]
 [, pktable_qualifier] [, fktable_name]
 [, fktable_owner] [, fktable_qualifier]
```

Parameters

- **pktable_name –** is the name of the table containing the primary key. Wildcard-character search patterns are not supported. You must specify this parameter, the **fktable_name** parameter, or both.
- **pktable_owner –** is the owner of the table containing the primary key. Wildcard-character search patterns are not supported.
- **pktable_qualifier –** is ignored. Leave blank or set to NULL.
- **fktable_name –** is the name of the table containing the foreign key. Wildcard-character search patterns are not supported. You must specify this parameter, the **pktable_name** parameter, or both.

- **fktable_owner –** is the owner of the table containing the foreign key. Wildcard-character search patterns are not supported.
- **fktable_qualifier –** is ignored. Leave blank or set to NULL.

Usage

This procedure corresponds to the ODBC function **SQLForeignKeys**.

sp_pkeys

Returns primary key information for a single table.

Primary keys must be declared through the ANSI integrity constraint mechanism.

Syntax

```
sp_pkeys table_name [, table_owner]
 [, table_qualifier]
```

Parameters

- **table_name –** is the name of the table. Wildcard-character search patterns are not supported.
- **table_owner –** is the owner of the table. Wildcard-character search patterns are not supported.
- **table_qualifier –** is ignored. Leave blank or set to NULL.

Usage

This procedure corresponds to the ODBC function **SQLPrimaryKeys**.

sp_server_info

Returns target server metadata containing a list of attribute names and matching values for the target.

Syntax

```
sp_server_info [attribute_id]
```

Parameters

- **attribute_id –** is the integer ID of the attribute.

Usage

This procedure generates an extensible result set. It can be expanded, depending upon the needs of the specific access service library.

sp_special_columns

Retrieves the information about columns within a specified table or view.

- The optimal set of columns that uniquely identifies a row in the table or view
- The columns that are automatically updated when any value in the row is updated by a transaction

Syntax

```
sp_special_columns table_name [, table_owner]
 [, table_qualifier] [, col_type]
```

Parameters

- **table_name –** is the name of the table. Wildcard-character search patterns are not supported.
- **table_owner –** is the owner of the table. Wildcard-character search patterns are not supported.
- **table_qualifier –** is ignored. Leave blank or set to NULL.
- **col_type –** is a value that requests information about columns of a specific type:
  - "R" returns information about columns with values that uniquely identify any row in the table.
  - "V" returns information about columns with values that are automatically generated by a target each time a row is inserted or updated.

Usage

- This procedure corresponds to the ODBC function **SQLSpecialColumns**.

See *ODBC Datatypes* on page 130 for matching ODBC integer identifiers returned in the `TYPE_NAME` and `DATA_TYPE` columns of the result set.

sp_sproc_columns

Returns information about stored procedure input and return parameters.

Syntax

```
sp_sproc_columns sp_name [, sp_owner]
 [, sp_qualifier] [, column_name]
```

```
or
```

```
sp_sproc_columns procedure_name [, procedure_owner]
 [, procedure_qualifier] [, column_name]
```

Parameters

- **sp_name or procedure_name –** is the name of the stored procedure. Wildcard-character search patterns are not supported.
- **sp_owner or procedure_owner –** is the owner of the stored procedure. Wildcard-character search patterns are not supported.
- **sp_qualifier or procedure_qualifier –** is ignored. Leave blank or set to NULL.
- **column_name –** is the name of the parameter about which you want information. If you do not supply a parameter name, this procedure returns information about all input parameters.

Usage

This procedure corresponds to the ODBC function **SQLProcedureColumns**.

Note: In ECDA Option for ODBC, **sp_sproc_columns** returns extra, unsolicited columns.

sp_statistics

Returns a list of indexes in a single table.

Syntax

```
sp_statistics table_name [, table_owner]
 [, table_qualifier] [, index_name] [, is_unique]
```

Parameters

- **table_name –** is name of the table. Wildcard-character search patterns are not supported.
- **table_owner –** is the owner of the table.
- **table_qualifier –** is ignored. Leave blank or set to NULL.

- **is_unique –** is one of :
    - “Y” if unique indexes are to be returned.
    - “N” if unique indexes are not to be returned.
- **index_name –** is the name of the index. Wildcard-character search patterns are not supported.

Usage

This procedure corresponds to the ODBC function **SQLStatistics**.

Note: With all platforms in ECDA Option for ODBC, the *index_name* parameter is ignored, regardless of the value. This applies even if you set the value to a nonexistent name or to NULL.

sp_stored_procedures

Returns a list of available procedures.

Syntax

```
sp_stored_procedures [sp_name] [, sp_owner]
 [, sp_qualifier]
```

Parameters

- **sp_name –** is the stored procedure name. Use the wildcard character to request information about more than one stored procedure.
- **sp_owner –** is the owner of the stored procedure. Use the wildcard character to request information about procedures owned by more than one user.
- **sp_qualifier –** is the name of the database. Acceptable values are the current database or NULL.

Usage

This procedure corresponds to the ODBC function **SQLProcedures**.

sp_table_privileges

Returns privilege information for all columns in a table or view.

Note: This stored procedure is not supported in ECDA Option for ODBC to DB2 UDB data sources.

Syntax

```
sp_table_privileges table_name [, table_owner
[, table_qualifier]]
```

Parameters

- **table_name –** is the name of the table. Wildcard-character search patterns are not supported.
- **table_owner –** is the name of the table owner. Wildcard-character search patterns are not supported.
- **table_qualifier –** For DB2 UDB targets, this is ignored. Leave blank or set to NULL.

  For non DB2 targets, this is the name of the database. Acceptable values are the current database or NULL.

Usage

This procedure corresponds to the ODBC function **SQLTablePrivileges**.

sp_tables

Returns a list of objects that can appear in a **from** clause.

Syntax

```
sp_tables [table_name] [, table_owner]
 [, table_qualifier] [, table_type]
```

Parameters

- **table_name –** is the name of the table. Use the wildcard character to request information about more than one table.
- **table_owner –** is the owner of the table. Use the wildcard character to request information about tables owned by more than one user.
- **table_qualifier –** is ignored for DB2 UDB targets. Leave blank or set to NULL.

  For non DB2 targets, is the name of the database. Acceptable values are the current database or NULL.
- **table_type –** is a list of values, separated by commas, that gives information about the table types specified.

Usage

- Enclose each table type with single quotation marks and enclose the entire parameter with double quotation marks. Enter table types in uppercase. For example: “‘TABLE’, ‘SYSTEM TABLE', ‘VIEW’, ‘SYNONYM’”
- This procedure corresponds to the ODBC function **SQLTables**.

ECDA Option for ODBC and Adaptive Server System Procedures

System procedures are Sybase-supplied stored procedures that return information about the access service and the target database.

If the access service cannot support one of these procedures, the procedure returns a formatted result set containing zero rows.

Table 12. Support Between ECDA System Procedures and Other Sybase Products

ECDA System Procedures	Support Relationship
sp_groups	Supported in ECDA Option for ODBC but not in Adaptive Server
sp_who	Supported only in ECDA Option for ODBC to Microsoft SQL Server datasources and ECDA Option for Oracle
• **sp_configure** (read-only in ECDA Option for ODBC) • **sp_helpserver** • **sp_sqlgetingo**	Defined differently in ECDA Option for ODBC and Adaptive Server
• **sp_capabilities** • **sp_thread_props**	Defined in ECDA Option for Oracle to support Adaptive Server/CIS products
• **sp_char_length** • **sp_datalength** • **sp_password** • **sp_patindex** • **sp_textvalid**	Not supported in ECDA Option for Oracle

sp_capabilities

Returns the SQL capabilities of an access service.

Syntax

```
sp_capabilities
```

Parameters

This procedure does not allow parameters.

Usage

- The result set contains information that allows applications to successfully interact with an access service during normal query processing.
- See the *Adaptive Server Enterprise Component Integration Services Users Guide* for information about the requirements for **sp_capabilities**.

Result Set for sp_capabilities

The result set includes column name, datatype, and a description.

Column	Datatype	Description
ID	`int`	Capability ID
CAPABILITY_NAME	`char`(30)	Capability name
VALUE	`int`	Capability value
DESCRIPTION	`char`(128)	Capability description

Values for sp_capabilities

sp_capabilities values.

Capability Number	Capability	Description
101	SQL syntax	• 1 – Sybase supported • 2 – DB2 UDB supported
102	Join handling	• 0 – unsupported • 1 – all but outer join supported • 2 – full join supported
103	Aggregate handling	• 0 – unsupported • 1 – count not supported • 2 – all functions
104	AND predicates	• 0 – unsupported • 1 – supported

Capability Number	Capability	Description
105	OR predicates	• 0 – unsupported • 1 – supported
106	LIKE predicates	• 0 – unsupported • 1 – ANSI supported • 2 – Sybase supported
107	Bulk insert handling	• 0 – unsupported • 1 – supported
108	Text/image handling	• 0 – unsupported • 1 – text without textptr supported • 2 – text with textptr supported
109	Transaction handling	• 0 – unsupported • 1 – local supported • 2 – two-phase-commit supported
110	Text pattern handling	• 0 – unsupported • 1 – supported
111	Order by	• 0 – unsupported • 1 – supported
112	Group by	• 0 – unsupported • 1 – supported
113	Net password encryption	• 0 – unsupported • 1 – supported
114	Object case sensitivity	• 0 – not case-sensitive • 1 – case-sensitive
115	Distinct	• 0 – unsupported • 1 – supported

Capability Number	Capability	Description
116	Wildcard escape	• 0 – unsupported • Anything else is the escape character
117	Union handling	• 0 – unsupported • 1 – supported
118	String functions	• 0 – unsupported • 1 – substring supported • 2 – Oracle subset supported • 3 – all Transact-SQL supported
119	Expression handling	• 0 – unsupported • 1 – supported • 2 – full Transact-SQL supported
120	Truncate trailing spaces on `varchar/char` CSP parameters	• 0 – no • 1 – yes
121	Language events	• 0 – no support for DML • 1 – DML support without datetime in **where** clause • 2 – no restrictions on DML
122	Date functions	• 0 – unsupported • 1 – all Transact-SQL supported
123	Math functions	• 0 – unsupported • 1 – Oracle subset supported • 2 – all Transact-SQL supported
124	Transact-SQL convert function	• 0 – unsupported • 1 – supported
125	Transact-SQL delete/update	• 0 – unsupported • 1 – Transact-SQL extensions supported

Capability Number	Capability	Description
126	Insert/update handling	• 0 – unsupported • 1 – supported
127	Subquery handling	• 0 – unsupported • 1 – supported
128	**In/not in** clause	• 0 – unsupported • 1 – supported
129	Case expression in a SQL statement	• 0 – unsupported • 1 – supported
132	Tables per statement	• 0 – unsupported • 1 – supported
133	Java UDF support	• 0 – unsupported • 1 – supported
134	Java ADT support	• 0 – unsupported • 1 – supported
135	Quoted Identifier support	• 0 – unsupported • 1 – supported
137	SELECT-NULL support	• 0 – unsupported • 1 – supported
138	Identity column support	• 0 – unsupported • 1 – supported

sp_configure

Provides a complete list of configuration names, minimum and maximum values, configured values, and current run values for each item.

Syntax

```
sp_configure
```

Parameters

Does not allow parameters.

Usage

Returns an empty result set because none of the configuration information is supported.

sp_groups

Returns the current user name as the sole user group.

Syntax

```
sp_groups
```

Parameters

- **None. –** This procedure does not allow parameters.

Usage

This procedure has been created for ECDA Option for ODBC. It is not documented in any Adaptive Server or ODBC manuals.

Shows the result set for:

- **Column –** GROUP_NAME.
- **Datatype –** `char`(8).
- **Description –** group name (authorization ID).

sp_helpserver

Returns help server information with the access service library details.

- The name of the access service in use
- The version of Open Server in use
- The version of the DirectConnect server in use
- The version of the ECDA Option for ODBC access service library in use
- The version of the DBMS with which the access service is associated

Syntax

```
sp_helpserver
```

Parameters

This procedure does not allow parameters.

sp_sqlgetinfo

Provides information about SQL grammar, syntax, and capabilities that the target DBMS supports.

Syntax

```
sp_sqlgetinfo [attribute_name]
```

Parameters

- **attribute_name –** is the name of a particular SQL option.

Usage

- This function corresponds to the ODBC function **SQLGetInfo**.
- If this procedure is called but no option is specified, the result set includes all SQL options.
- If the attribute is not found in the internal table, the access service returns an error.
- If the parameter is not provided, the access service returns a result set of all supported SQL options.

Table 13. Format for sp_sqlgetinfo

sql_option	varchar(30)	not null
sql_value	varchar(255)	null

Note: If the **sql_value** column is NULL, this option is not supported for the target DBMS.

SQL Options for sp_sqlgetinfo

Options for the ECDA Option for ODBC and SQL.

SQL Option	Description
SQL_Accessible_Procedures	User can execute all procedures returned by **sp_stored_procedures**.
SQL_Accessible_Tables	User is guaranteed **SELECT** privileges to tables returned by **sp_tables**.
SQL_Active_Connections	No known limit to the number of connections.
SQL_Active_Statements	No known limit to the number of statements for a connection.
SQL_Alter_Table	Bitmask indicating which clauses in **ALTER TABLE** are supported.
SQL_Bookmark_Persistence	Bitmask enumerating through which bookmarks persist. None supported.
SQL_Column_Alias	Support for column alias.
SQL_Concat_Null_Behavior	Bitmask indicating how the DBMS handles concatenations with NULL.

SQL Option	Description
SQL_Convert_Bigint SQL_Convert_Binary SQL_Convert_Bit SQL_Convert_Char SQL_Convert_Date SQL_Convert_Decimal SQL_Convert_Double SQL_Convert_Float SQL_Convert_Integer SQL_Convert_Longvarbinary SQL_Convert_Longvarchar SQL_Convert_Numeric SQL_Convert_Real SQL_Convert_Smallint SQL_Convert_Time SQL_Convert_Timestamp SQL_Convert_Tinyint SQL_Convert_Varbinary SQL_Convert_Varchar	Bitmask indicating conversions “to type” supported.
SQL_Convert_Functions	Bitmask indicating conversion functions supported.
SQL_Correlation_Name	Table correlation names supported.
SQL_CSP_Support	Sybase/Intersolv extension for supporting CSPs. Value = 16383.
SQL_Cursor_Commit_Behavior	Bitmask indicating how a **commit** operation affects a cursor.
SQL_Cursor_Rollback_Behavior	Bitmask indicating how a **rollback** operation affects a cursor.
SQL_Cursor_Sensitivity	A value indicating support for cursor sensitivity.
SQL_Database_Name	Value provided by the DirectConnect server.

SQL Option	Description
SQL_Date_Source_Read_Only	The datasource is read/write.
SQL_DBMS_Name	The target DBMS name. A maximum of 30 characters is returned.
SQL_DBMS_Ver	The target DBMS version in the form ##.##.####. A maximum of 30 characters is returned. The version string may have target-specific information that follows.
SQL_Default_TXN_Isolation	Bitmask indicating the default transaction level supported by the DBMS.
SQL_Dynamic_Cursor_Attributes1	Bitmask that describes the attributes of a dynamic cursor that are supported by the driver (1st subset of attributes.)
SQL_Dynamic_Cursor_Attributes2	Bitmask that diatribes the attributes of a dynamic cursor that are supported by the driver (2nd subset of attributes.)
SQL_Expressions_In_Orderby	Support for expressions in **order by** clause.
SQL_Fetch_Direction	Bitmask enumerating supported options.
SQL_File_Usage	Files treated in datasource.
SQL_Forward_Only_Cursor_Attributes1	A bitmask that describes the attributes of a forward-only cursor that are supported by the driver (1st subset of attributes.)
SQL_Forward_Only_Cursor_Attributes2	A bitmask that describes the attributes of a forward-only cursor that are supported by the driver (2nd subset of attributes.)
SQL_Getdata_Extensions	Bitmask enumerating extensions to **SQLGetData**.
SQL_Group_By	Bitmask indicating the relationship between **GROUP BY** columns supported in the DBMS.
ICD_Cursor_Support	Bitmask indicating cursor support.
ICD_Dynamic_Support	Bitmask indicating dynamic statement support.
ICD_Execdirect	Bitmask indicating how dynamic **execdirect** statement is supported.

SQL Option	Description
ICD_Language_Support	Bitmask indicating language statement support. No parameter marker support.
ICD_Longtypes_Supported	Support for long types as parameters.
ICD_Modify_Groupby	Intersolv driver insures **GROUP BY** clause when aggregate functions are used as part of the select list.
SQL_Identifier_Case	Defines whether identifiers are case-sensitive.
SQL_Identifier_Quote_Char	Character used to delimit quoted identifiers.
SQL_Keywords	See the *Microsoft ODBC 3.5 Programmer's Reference and SDK Guide* for information.
SQL_Like_Escape_Clause	Support of "%" character and "_" character as escape characters in **like** clause.
SQL_Lock_Types	Bitmask enumerating supported lock types.
SQL_Max_Binary_Literal_Len	Maximum length of binary literal is either unknown or unlimited.
SQL_Max_Char_Literal_Len	Maximum length of character literal is either unknown or unlimited.
SQL_Max_Column_Name_Len	Maximum length for a column name. Convert this string to an integer. A value of 0 means not supported.
SQL_Max_Columns_In_Group_By	Maximum number of columns allowed in **SQL GROUP BY** clause. Convert this string to an integer. A value of 0 means that the limit is unknown or unlimited.
SQL_Max_Columns_In_Index	Maximum number of columns allowed in a **SQL CREATE INDEX**. Convert this string to an integer. A value of 0 means that the limit is unknown.
SQL_Max_Columns_In_Order_By	Maximum number of columns allowed in a **SQL ORDER BY** clause. Convert this string to an integer. A value of 0 means that the limit is unknown.
SQL_Max_Columns_In_Select	Maximum number of columns allowed in a **SQL SELECT** column list. Convert this string to an integer. A value of 0 means that the limit is unknown.

SQL Option	Description
SQL_Max_Columns_In_Table	Maximum number of columns allowed in a **SQL CREATE TABLEE**. Convert this string to an integer. A value of 0 means that the limit is unknown.
SQL_Max_Cursor_Name_Len	Maximum length for a cursor name. Convert this string to an integer. A value of 0 means not supported.
SQL_Max_Index_Size	Maximum number of characters allowed in the combined column length of an index. Convert this string to an integer. A value of 0 indicates that the limit is unknown.
SQL_Max_Identifier_Len	Maximum size in characters that the datasource supports for user-defined names.
SQL_Max_Owner_Name_Len	Maximum length for an owner name. Convert this string to an integer. A value of 0 means not supported.
SQL_Max_Procedure_Name_Len	Maximum length for a procedure name. Convert this string to an integer. A value of 0 means not supported
SQL_Max_Qualifier_Name_Len	Maximum length for a qualifier name. Convert this string to an integer. A value of 0 means not supported.
SQL_Max_Row_Size	Maximum number of characters allowed in the combined column length of a row in a table. Convert this string to an integer. A value of 0 means that the limit is unknown.
SQL_Max_Row_Size_Includes_Long	Includes the length of all long datatypes.
SQL_Max_Statement_Len	Maximum length allowed for a SQL statement. Convert this string to an integer. A value of `0` means that the limit is unknown.
SQL_Max_Table_Name_Len	Maximum length allowed for a table name. Convert this string to an integer. A value of `0` means not supported.
SQL_Max_Tables_In_Select	Maximum number of columns allowed in a **SQL SELECT FROM** clause. Convert this string to an integer. A value of 0 means that the limit is unknown.
SQL_Max_User_Name_Len	Maximum length for the user name. Convert this string to an integer. A value of 0 means not supported.

SQL Option	Description
SQL_Mult_Result_Sets	Driver does not support multiple result sets in a given language event.
SQL_Multiple_Active_TXN	Only one connection can have an active transaction.
SQL_Need_Long_Data_Len	Need the length of the long datatypes.
SQL_Non_Nullable_Columns	Bitmask indicating whether the DBMS supports non-nullable columns.
SQL_Null_Collation	Bitmask indicating how the DBMS collates NULL values.
SQL_Numeric_Functions	Bitmask indicating the supported scalar numeric functions.
SQL_ODBC_API_Conformance	Bitmask enumerating ODBC level.
SQL_ODBC_SAG_CLI_Conformance	Bitmask enumerating compliance to functions of the SAG specification.
SQL_ODBC_SQL_Conformance	Bitmask indicating supported SQL grammar.
SQL_ODBC_SQL_Opt_IEF	Support for Integrity Enhancement Facility (IEF).
SQL_Order_By_Columns_In_Select	Columns in **ORDER BY** clause must be in select list.
SQL_OJ_Capabilities	A bitmask enumerating the types of outer joins supported by the driver and datasource.
SQL_Outer_Joins	Support for outer joins.
SQL_Owner_Term	The DBMS term for an owner name. A maximum of 30 characters is returned. A null value means not supported.
SQL_Owner_Usage	Bitmask indicating statements in which owners can be used.
SQL_Pos_Operations	Bitmask enumerating the operations in **SQLSetPos**.
SQL_Positioned_Statements	Bitmask indicating supported positioned SQL statements.
SQL_Procedure_Term	DBMS term for a procedure name. A maximum of 30 characters is returned. A null value means not supported.
SQL_Procedures	Support for procedures.

SQL Option	Description
SQL_Qualifier_Location	Bitmask indicating the position of the qualifier in a qualified table name.
SQL_Qualifier_Name_Separator	Character or string separator between the qualifier and the name element. A maximum of five characters is returned.
SQL_Qualifier_Term	DBMS term for a qualifier name. A maximum of 30 characters is returned.
SQL_Qualifier_Usage	Bitmask indicating in which statements a qualifier can be used.
SQL_Quoted_Identifier_Case	Bitmask describing SQL identifier case and storage in system tables when used in SQL statements.
SQL_Row_Updates	See the *Microsoft ODBC 3.5 Programmer's Reference and SDK Guide* for information.
SQL_Scroll_Concurrency	Bitmask identifying concurrency control options for scrollable cursors.
SQL_Scroll_Options	Bitmask indicating scroll options for scrollable cursors.
SQL_Search_Pattern_Escape	See the *Microsoft ODBC 3.5 Programmer's Reference and SDK Guide* for information.
SQL_Set_Database_Context	Sybase/Intersolv extension for supporting CSPs. If value = Y, the driver issues **use_database_name** to the configured database name and is sensitive to three-part names.
SQL_Special_Characters	Special characters used in object names. All characters except a–z, A–Z, 0–9, and the underscore character.
SQL_SQL_Conformance	A value indicating the level of SQL-92 supported by the driver.
SQL_String_Functions	Bitmask indicating supported scalar string functions.
SQL_Subqueries	Bitmask indicating predicates that support subqueries.
SQL_System_Functions	Bitmask indicating supported scalar system functions.
SQL_TimeDate_Add_Intervals	Bitmask indicating supported timestamp intervals associated with `TIMESTAMPADD` function.

SQL Option	Description
SQL_TimeDate_Diff_Intervals	Bitmask indicating supported timestamp intervals associated with `TIMESTAMPDIFF` function.
SQL_TimeDate_Functions	Bitmask indicating supported timestamp intervals.
SQL_TXN_Capable	Indicates the transaction support in the DBMS.
SQL_TXN_Isolation_Option	Bitmask indicating transaction isolation levels.
SQL_Union	Bitmask indicating support for **union** clause.
SQL_User_Name	Current user name. A maximum of **SQL_Max_User_Name_Len** characters are returned. A null value means not supported.

sp_thread_props

Allows the client to retrieve and set various thread properties.

Syntax

```
sp_thread_props [ property_name [, property_value ]]
```

Parameters

- **property_name –** is the name of the property to be set or shown.
- **property_value –** is the value to which the property is set.

Usage

If you do not provide any parameters, or if you provide only *property_name*, the access service returns a single result set consisting of every instance of *property_name* and the value for each.

sp_who

Reports information on current users and processes on Microsoft SQL Server.

Note: This property is supported by ECDA Option for ODBC and ECDA Option for Oracle to Microsoft SQL data sources only.

Syntax

```
sp_who [ login_name | "spid" ]
```

Parameters

- **login_name –** is the SQL Server login name for the user. If you provide a login name, **sp_who** reports information on processes being run by the specified user.
- **“spid” –** is the number of a specific process. You must enclose it in quotes because a character-type argument is expected.

Usage

For each process being run, **sp_who** reports the server process ID, its status, the login name, the name of the host computer, the name of the database, and the command being run.

If no name is provided, **sp_who** reports on processes being run by all users.

Configuration Properties Quick Reference

A list of configuration properties for ECDA Option for ODBC, which accesses DB2 UDB, Microsoft SQL Server, and ODBC-accessible datasources.

The property category represents the subsection heading in the access service library configuration file for ECDA Option for ODBC targets.

The far right column identifies:

- GV – global variable, which allows an application to issue a **select** statement to query a global variable for the values of the property and view those values.
- SS – **set** statement, which allows an application to change the values of the property, but only for the current connection.
- None – the values of a configuration property cannot be accessed by either global variable or a **set** statement. You can use DirectConnect Manager to do so, or you can access the file directly.

Table 14. Configuration Properties for ECDA Option for ODBC

Property Name	Property Values	Property Category {subsection name}	Global Variable (GV) or Set Statement (SS)
`Allocate`	`[connect|request]`	{Target Interaction}	GV and SS
`BinaryResults`	`[binary|char]`	{Datatype Conversion}	GV and SS
`BulkCommitCount`	`integer`	{Transfer}	None
`CharConvertError`	`[reject|truncate]`	{Data Conversion Errors}	GV and SS
`ClientDecimalSeparator`	`char`	{Client Interaction}	GV and SS
`ClientIdleTimeout`	`integer`	{Client Interaction}	None
`ConnectionSpec1`	`char`	{ACS Required}	None

Property Name	Property Values	Property Category {subsection name}	Global Variable (GV) or Set Statement (SS)
`CSPExclusions`	`[none|user|non-auth|nonauth-public]`	{Catalog Stored Procedures}	GV and SS
`CSPIncludeAlias`	[no | yes]	{Catalog Stored Procedures}	GV and SS
`CSPIncludeSyno-nym`	[no | yes]	{Catalog Stored Procedures}	GV and SS
`CSPIncludeSystem`	[no | yes]	{Catalog Stored Procedures}	GV and SS
`CSPIncludeTable`	[no | yes]	{Catalog Stored Procedures}	GV and SS
`CSPIncludeView`	[no | yes]	{Catalog Stored Procedures}	GV and SS
`DatatypeInfo`	`[transact|tar-get]`	{Catalog Stored Procedures}	GV and SS
`DateResults`	`[datetime|date-time4|char_iso|char_usa|char_eur|char_jis|char_odbc]`	{Datatype Conversion}	GV and SS
`DateTimeConver-tError`	`[reject|null|default]`	{Data Conversion Errors}	GV and SS
`DateTimeResults`	`[datetime|date-time4|char_iso|char_usa|char_eur |char_jis|char_odbc]`	{Datatype Conversion}	GV and SS
`DecimalResults`	`[autoconvert|int|float|real|char|money|mon-ey4|bcd]`	{Datatype Conversion}	GV and SS
`DefaultDate`	`yyyy-mm-dd`	{Data Conversion Errors}	None

Property Name	Property Values	Property Category {subsection name}	Global Variable (GV) or Set Statement (SS)
DefaultNum	integer	{Data Conversion Errors}	None
DefaultTime	hh.mm.ss	{Data Conversion Errors}	None
DelimitSqlRequests	[no | yes]	{Target Interaction}	GV and SS
EnableAtStartup	[no | yes]	{Client Interaction}	None
FloatResults	[float|real|char]	{Datatype Conversion}	GV and SS
Int2Results	[smallint|char]	{Datatype Conversion}	GV and SS
Int4Results	[int|char]	{Datatype Conversion}	GV and SS
IsolationLevel	[ur|cr|rr|sr|vr|no]	{Target Interaction}	GV
LogConnectionStatistics	[no | yes]	{Logging}	None
LogReceivedSQL	[no | yes]	{Logging}	None
LogRequestStatistics	[no | yes]	{Logging}	None
LogServiceStatistics	integer	{Logging}	None
LogSvcLibStatistics	integer	[Service Library] {Logging}	None
LogTargetActivity	[no | yes]	{Logging}	None
LogTransferStatistics	[no | yes]	{Logging}	None
LogTransformedSQL	[no | yes]	{Logging}	None
MaxResultSize	integer	{Client Interaction}	GV and SS

Property Name	Property Values	Property Category {subsection name}	Global Variable (GV) or Set Statement (SS)
MaxRowsReturned	integer	{Client Interaction}	GV and SS
MaxSvcConnections	integer	{Client Interaction}	GV
NumConvertError	[reject|null|default]	{Data Conversion Errors}	GV and SS
ODBCDriverManager	ODBC Driver Manager library name	[Service Library] {Client Interaction}	None
quoted_identifier	[on | off]	{Client Interaction}	GV and SS
QuotedStringDelimiter	char	{Target Interaction}	GV and SS
RealResults	[float|real|char]	{Datatype Conversion}	GV and SS
ReturnNativeError	[no | yes]	{Target Interaction}	GV
SendWarningMessages	[no | yes]	{Client Interaction}	GV and SS
ServiceDescription	char	{Client Interaction}	GV
SQLOdbcCursors	[if_needed|odbc|driver|default]	{Target Interaction}	SS
SQLTransformation	[passthrough|sybase|tsql0|tsql1|tsql2]	{Target Interaction}	GV and SS
StripBinaryZero	[no | yes]	{Client Interaction}	GV and SS
StopCondition	[error|none|warning]	{Target Interaction}	GV and SS

Property Name	Property Values	Property Category {subsection name}	Global Variable (GV) or Set Statement (SS)
SvclibDescription	char	[Service Library] {Client Interaction}	GV
TargetDBMS	[notused|UDBLAN|UDBOS390|UDBAS400]	{Target Interaction}	GV and SS
TargetDecimalSeparator	char (default is a period)	{Target Interaction}	GV
TextSize	integer	{Client Interaction}	GV and SS
TimeResults	[datetime|datetime4|char_iso|char_usa|char_eur|char_jis|char_odbc]	{Datatype Conversion}	GV and SS
TinyIntResults	[smallint|tinyint]	{Datatype Conversion}	GV and SS
TraceEvents	[no | yes]	{Tracing}	none
TraceInterface	[no | yes]	{Tracing}	none
TraceTarget	[no | yes]	{Tracing}	none
TransactionMode	[short|long]	{Client Interaction}	GV and SS
TransferBatch	integer	{Transfer}	GV and SS
TransferBatchSeparator	integer	{Transfer}	GV and SS
TransferErrorAction	[noaction|rollback]	{Transfer}	GV and SS
TransferErrorCount	integer	{Transfer}	GV and SS

Property Name	Property Values	Property Category {subsection name}	Global Variable (GV) or Set Statement (SS)
`TransferPacket-Size`	`integer`	{Transfer}	GV and SS
`Version`	`versionstring`	{Client Interaction}	GV
`XNLChar`	`integer`	{Datatype Conversion}	GV and SS
`XNLVarChar`	`integer`	{Datatype Conversion}	GV and SS

Datatype Conversion

An access service converts data as ODBC-to-Open Server and Open Server-to-ODBC datatypes.

Limitations

The maximum size of data through ECDA Option for ODBC products is 32,767 bytes for all datatypes, including `text` and `image`.

A data size larger than 32,767 bytes is truncated.

Warning! ECDA Option for ODBC cannot correctly represent or transport `varchar` values containing empty strings (zero length non null strings). Empty string `varchar` values are represented as `NULL` values.

ODBC-to-Open Server Datatypes

When you retrieve data from the target database, the access service converts the target data to default Open Server datatypes for delivery to the client application.

The ODBC datatypes and the resulting Open Server datatypes.

Table 15. ODBC to Open Server Datatype Mapping

ODBC Datatype	Open Server Datatype
SQL_CHAR	CS_CHAR
SQL_VARCHAR	CS_VARCHAR
SQL_LONGVARCHAR	CS_TEXT
SQL_DECIMAL	CS_DECIMAL
SQL_NUMERIC	CS_NUMERIC
SQL_SMALLINT	CS_SMALLINT
SQL_INTEGER	CS_INTEGER
SQL_REAL	CS_REAL
SQL_FLOAT	CS_FLOAT
SQL_DOUBLE	CS_FLOAT
SQL_BIT	CS_BIT

ODBC Datatype	Open Server Datatype
SQL_TINYINT	CS_TINYINT
SQL_BIGINT	CS_NUMERIC
SQL_BINARY	CS_BINARY
SQL_VARBINARY	CS_VARBINARY
SQL_LONGVARBINARY	CS_IMAGE
SQL_DATE	CS_CHAR
SQL_TIME	CS_CHAR
SQL_TIMESTAMP	CS_CHAR
SQL_INTERVAL	CS_CHAR

Data Value Conversion Result Set

Data values returned from the target DBMS to a client application are converted into a format that Open Client and Open Server can handle.

This conversion may encounter inconsistencies, particularly in supported ranges. The access service must convert the value from the target into a matching Open Client and Open Server datatype before it sends the value back to the client.

To do this, the access service uses configuration properties. Each target datatype has a default Open Client and Open Server mapping, but these may be overridden either by the configuration property (thus affecting the entire service) or through a **set** statement (thus affecting only the client connection).

See also

- *Data Conversion Error Properties* on page 30

Data Values Sent to the Client Application

CHAR and VARCHAR datatype values shorter than the XNLCHAR and XNLVARCHAR values are returned to the client application as CS_CHAR.

Datatype values longer than the XNLCHAR and XNLVARCHAR values, are returned as CS_TEXT.

A DECIMAL datatype is returned to the client application as CS_DECIMAL. Otherwise, the configuration settings shown in the `DecimalResults` configuration property applies.

For clients with System 10™ and earlier versions, DECIMAL data is returned as CS_FLOAT.

Open Server-to-ODBC Datatypes

An access service converts or performs SQL transformation on incoming Open Server data it receives in a client request.

The data includes:

- Data values embedded as strings within the text of **select**, **insert**, **delete**, **update**, and **execute** language commands
- Data values as parameters of RPC, cursor, transfer, or dynamic SQL commands
- Datatype names as part of **create table** or **alter table** commands

The access service does not perform automatic incoming datatype conversions on data values embedded in strings or received as parameters. Instead, the client application receives a string template from the target datatypes so that it can format the strings correctly before sending them to the target DBMS. The formatting is set up through the **sp_columns** catalog stored procedure.

Data Values Embedded as Strings

When the access service receives a SQL command with embedded data values, the SQL transformation mode determines whether any transformation is applied to these values.

Transformation rules:

- If the access service is in **passthrough** mode, it does not perform transformation.
- If the access service is in **sybase** mode, it:
  - Removes the currency symbol from `money` datatypes
  - Transforms quoted strings to quoting conventions specific to the target DBMS

Datatype constants are not transformed in any way except as described. When passing datatype constants, the client must verify that the constants are in the proper format required by the target DBMS.

See also

- *SQL Statements* on page 69

Data Values Received as Parameters

When an access service receives data values as parameters to RPC commands, cursor commands, or dynamic SQL commands, it converts Open Client and Open Server datatypes to default target DBMS datatypes.

In most cases, Open Client and Open Server datatypes directly map to target datatypes, and the service library defines default mapping rules. However, if the defaults are not valid, the CT-

Library client specifies the intended target datatype through the `usertype` field of the CS_DATAFMT structure.

An Open Server datatype without an associated user-defined datatype is transformed to an ODBC datatype.

The client application can obtain the actual target DBMS datatype for a particular column through the **sp_columns** CSPs.

See also

- *ODBC-to-Open Server Datatypes* on page 165
- *Microsoft SQL Server ODBC-Supported Datatypes* on page 169

CS_DATAFMT usertype Field Values

The `usertype` field of the CS_DATAFMT structure is a 32-bit integer.

The client application can specify a target DBMS datatype for a given column. The client application obtains the column datatype indicator from the `REMOTE_DATA_TYPE` column of the **sp_columns** result set.

The client application must place the value from **sp_columns** in the least-significant byte of the `usertype` field. The remainder of the value is ignored. If a value of `0` is used, the default conversion applies.

The user-defined datatype is used in the child process to transform the Open Server datatype to the ODBC datatype.

Datatype Names

An access service receives datatype names as part of **create table** or **alter table** commands.

- If the access service is in **passthrough** mode, the datatype names are not modified.
- If the access service is in **sybase** mode, Sybase names are assumed and are converted to corresponding target-specific datatype names.

A given target may not be able to support all Open Client and Open Server datatypes, but it permits conversion to an equivalent or compatible datatype. For example, the CS_MONEY datatype can be converted to a numeric (19,4) or equivalent datatype.

Microsoft SQL Server ODBC-Supported Datatypes

Supported Microsoft SQL Server datatypes and their corresponding ODBC datatypes.

Table 16. Microsoft SQL Datatypes and Related ODBC Datatypes

Microsoft SQL Server Datatypes	ODBC Datatypes
DATE	SQL_TYPE_DATE(91)
CHAR	SQL_WCHAR(1)
NUMERIC	SQL_WNUMERIC(2)
DECIMAL	SQL_WDECIMAL(3)
MONEY	SQL_WDECIMAL(3)
SMALLMONEY	SQL_WDECIMAL(3)
INT	SQL_WINTEGER(4)
SMALLINT	SQL_WSMALLINT(5)
FLOAT	SQL_WFLOAT(6)
REAL	SQL_WREAL(7)
VARCHAR	SQL_WVARCHAR(12)
TEXT	SQL_WLONGVARCHAR(-1)
TIMESTAMP	SQL_WBINARY(-2)
BINARY	SQL_WBINARY(-2)
VARBINARY	SQL_WVARBINARY(-3)
IMAGE	SQL_WLONGVARBINARY(-4)
TINYINT	SQL_WTINYINT(-6)
BIT	SQL_WBIT(-7)
*NCHAR	SQL_WCHAR(-8)
* NVARCHAR	SQL_WVARCHAR(-9)
* NTEXT	SQL_WLONGVARBINARY(-10)
TIMESTAMP	SQL_TYPE_TIMESTAMP(93)
DATETIME	SQL_TYPE_WTIMESTAMP(93)
SMALL DATETIME	SQL_TYPE_WTIMESTAMP(93)

* Datatypes are currently converted to single-byte ASCII.

DB2 UDB / ODBC-Supported Datatypes

Supported DB2 UDB datatypes and their related ODBC datatypes.

Table 17. Supported DB2 UDB Datatypes and Related ODBC Datatypes

DB2 UDB Datatypes	ODBC Datatypes
CHAR	SQL_CHAR(1)
CHAR () (for bit data)	SQL_BINARY(-2)
DATE	SQL_TYPE_DATE(91)
DECIMAL	SQL_DECIMAL(3)
DOUBLE	SQL_DOUBLE(8)
FLOAT	SQL_FLOAT(6)
FLOAT (4)	SQL_REAL(7)
INTEGER	SQL_INTEGER(4)
*CLOB	SQL_LONGVARCHAR(-1)
LONG VARCHAR	SQL_LONGVARCHAR(-1)
*BLOB	SQL_LONGVARBINARY(-4)
LONG VARCHAR () (for bit data)	SQL_LONGVARBINARY(-4)
NUMERIC	SQL_NUMERIC(2)
SMALLINT	SQL_SMALLINT(5)
TIME	SQL_TYPE_TIME(92)
TIMESTAMP	SQL_TYPE_TIMESTAMP(93)
VARCHAR	SQL_VARCHAR(12)
VARCHAR () (for bit data)	SQL_VARBINARY(-3)

* Results sets containing these datatypes are truncated to 32,768 bytes.

SQL Stored Procedures

SQL stored procedures are single SQL statements that are statically bound to the database and can be used by any client.

The access service does not support creating SQL stored procedures, because the SQL transformation process does not provide support to handle the translation. You must create SQL stored procedure source code outside of ECDA Option for ODBC.

The access service does not support either dropping SQL stored procedures or granting authorization for them. Both of these functions are target-dependent.

Running SQL Stored Procedures

Clients execute SQL stored procedures in different ways, depending upon the SQL transformation mode in effect.

- In passthrough mode, execute:
  `{call procname(parm1, parm2, ... parmn )}`
  or:
  `{call procname( ?, ?, ?, ..., ? )}`
  where *procname* is the name of the procedure.
- In sybase mode, execute:
  `EXEC procname argvalues`
  or:
  `EXECUTE procname argvalues`
  where:
  - *procname* is the name of the procedure.
  - *argvalues* is a list of argument values separated by commas.

  The values must be specified in the exact order specified in the **create procedure** statement contained in the SQL stored procedure. Input parameters are positional.

Datatype for Argument Values

The datatype for all argument values must be consistent with the datatypes of the relevant columns.

- Any argument declared as `numeric` can be used with any numeric column.
- Any argument declared as `character` can be used with any character column. The access service library treats `TEXT` as `CHAR` or `VARCHAR`.

- Any argument declared as `character` may be used with any `DATE`, `TIME`, or `TIMESTAMP` column. The value must be in the proper format for the specified type.

Character Arguments

Character arguments in **sybase** mode must be delimited.

- The value can be enclosed in single or double quotes.
- The value can be enclosed with these special delimiters (**passthrough** or **tsql0** modes only):
  - !
  - %
  - (
  -)
  - *
  - /
  - :
  - <
  - >
  - ?
  - \
  - |
  - ‘
  - {
  - }
  - ~
- The same character must be used before and after the value.

Note: Do not enclose numeric arguments in quotes or any special delimiter.

You can specify NULL values in **sybase** mode either with:

- “NULL” or “null” used as the value
- The argument left out of the list (if the procedure contains more than one argument)

SQL Stored Procedures Rules

When a **select** is issued as a SQL stored procedure, column names are not available.

If the client requests column names, the access service library returns dummy names.

To run SQL stored procedures:

- You can use binary values in character arguments. The argument cannot contain the `0` value, because the access service library sends all character arguments to ODBC as null terminated strings.

- Do not use escape sequences, because the access service library does not support them. For example, "ABCD\n" is sent as a 6-character string.
- Character values sent in the **use procedure** statement can be longer than declared in the **create procedure** statement. Any extra characters are truncated, and no error message is sent.
- If a numeric value has a larger scale than that in the target column, the argument is truncated, and no error is recorded.

DB2 Stored Procedures

DB2 stored procedures (external stored procedures) are customer-written programs that reside on the mainframe.

The programs can be written in assembler, COBOL, PL/1 or C, and execute within the DB2 stored procedure *ADDRESS* space.

Note: ECDA Option for ODBC for DB2 UDB targets does not support RSPs or host-resident requests.

Running DB2 Stored Procedures

Clients execute DB2 stored procedures in different ways, depending upon the SQL transformation mode in effect.

- In passthrough mode, execute:
  `{call procname(parm1, parm2, ... parmn )}`
  or:
  `{call procname( ?, ?, ?, ..., ? )}`
  where *procname* is the name of the stored procedure.
- In sybase mode, execute:
  `EXEC procname argvalues`
  or
  `EXECUTE procname argvalues`
  where:
  - *procname* is the name of the stored procedure.
  - *argvalues* is a list of argument values separated by commas.

Datatype for Argument Values

The datatype for all argument values must be consistent with the datatypes of the relevant columns.

- Any argument declared as `numeric` can be used with any numeric column.
- Any argument declared as `character` can be used with any character column. The access service library treats `TEXT` as `CHAR` or `VARCHAR`.
- Any argument declared as `character` may be used with any `DATE`, `TIME`, or `TIMESTAMP` column. The value must be in the proper format for the specified type.

Transact-SQL Commands

Many Transact-SQL commands use table names of up to three parts.

ODBC supports three-part naming convention:

- *location_name* (DBMS name, current server)
- *authorization_ID* (owner)
- *table_name or view_name*

If the first or second parts are not present, they are omitted. The access service supports three-part objects in all cases in which a SQL object is required in a SQL statement. In doing so, the SQL transformation code either drops the qualifier or performs the correct action with the qualifier.

When the access service receives a **use database_name** in **sybase** mode, it captures the name and sets a member of `smConnectionConcrete` with the database name so that SQL_GETINFO and SQL_DATABASE_NAME pass back the current database name.

alter table (core)

Adds new columns and constraints, changes or drops constraints, and partitions or unpartitions an existing table.

Syntax

Transact-SQL Syntax

```
alter table [database.[owner].]table_name
```

```
{add column_name datatype
[default {constant_expression | user | null}]
 {[{identity | null}]
 | [[constraint constraint_name]
 {{unique | primary key}
 [clustered | nonclustered]
 [with {fillfactor | max_rows_per_page} = x]
 [on segment_name]
 [references [[database.]owner.]ref_table
[(ref_column)]
 | check (search_condition)}]}...
 {[, next_column]}...
```

```
| add {[constraint constraint_name]
 {unique | primary key}
 [clustered | nonclustered]
 (column_name [{, column_name}...])
 [with {fillfactor | max_rows_per_page} = x]
 [on segment_name]
 | foreign key (column_name [{, column_name}...])
 references [[database.]owner.]ref_table
[(ref_column [{, ref_column}...])]
 | check (search_condition)}
```

```
| drop constraint constraint_name
```

```
| replace column_name
default {constant_expression | user | null}
```

```
| partition number_of_partitions
```

```
| unpartition}
```

ODBC Syntax

```
ALTER TABLE base_table_name
```

```
{ADD column_identifier datatype
```

```
|ADD(column_identifier datatype[,column_identifier datatype]...)
```

```
|DROP[COLUMN]column_identifier[CASCADE|RESTRICT]}
```

Parameters

- **null –** specifies that you should assign a NULL value when a value is not provided during an insertion.
- **table_name –** is the name of the table to be changed.
- **column_name –** is the name of a column to be added.
- **datatype –** is any of the system datatypes except `Bit`. If the transformation mode is **passthrough**, the datatype is expressed as an ODBC datatype.
- **next_column –** indicates that you can include additional column definitions separated by commas, using the same syntax described for a column definition.

Examples

- **Example 1 –** adds the *manager_name* column to the `publishers` table. For each existing table row, a NULL value is assigned to the new column.

  ```
  alter table publishers
  ```

  ```
  add manager_name varchar(40) null
  ```

Usage

- Adaptive Server/CIS sends the **alter table** command to the DirectConnect server as a language event.
- These are not supported:
  - `add constraint`
  - `drop constraint`

- `replace column name`
- `partition|unpartition`
- Transformation adds parentheses when the `add column` option includes more than one column.

begin transaction (Transact-SQL only)

Marks the starting point of a user-defined transaction.

Syntax

```
begin [transaction][transaction_name]
```

Parameters

- **transaction_name –** is the name assigned to the transaction. It must conform to the rules for identifiers. Use transaction names only on the outermost pair of nested **begin transaction**/**commit** or **begin transaction**/**rollback** statements.

Usage

- The access service library accepts the transaction name parameter, then strips it before passing it to the target.
- The **begin transaction** command is not recognized by ODBC, so the DirectConnect server traps this statement and monitors the transaction state internally. When a **commit** is issued after a **begin transaction**, **SQL Transact()** is called.
- Only one transaction can be open for each connection.

commit transaction (Transact-SQL only)

Marks the ending point of a user-defined transaction.

Syntax

```
commit [tran[saction] | work][transaction_name]
```

Parameters

- **transaction_name –** is the name assigned to the transaction. It must conform to the rules for identifiers. Use transaction names only on the outermost pair of nested begin **transaction**/**commit** or **begin transaction**/**rollback** statements.

Usage

- The access service strips the *transaction_name* from the statement before passing it to the target.
- *transaction_name* is not used with version 10.5 of Adaptive Server/CIS.
- The **commit** command is not recognized by ODBC, so the DirectConnect Server traps this statement and monitors the transaction state internally. When a **commit** is issued after a **begin transaction**, **SQL Transact()** is called.
- Only one transaction can be open for each connection.

create index (core)

Creates an index on one or more columns in a table.

Syntax

Transact-SQL Syntax

```
create [unique][clustered | nonclustered]
 index index_name
```

```
on [[database.]owner.]table_name(column_name
[, column_name]...)
```

```
[with {{fillfactor | max_rows_per_page} = x, ignore_dup_key,
sorted_data,
 [ignore_dup_row | allow_dup_row]}]
```

```
[on segment_name]
```

ODBC Syntax

```
CREATE [UNIQUE] INDEX index_name
```

```
ON base_table_name
```

```
(column_identifier[ASC|DESC]
 [,column_identifier[ASC|DESC]]...)
```

Parameters

- **unique –** prohibits duplicate index values (key values). The system checks for duplicate key values when an index is created and checks each time data is added with an **insert** or **update**. If a duplicate key value exists, or if more than one row contains a NULL value, the command aborts and an error message shows the duplicate value prints.
- **clustered –** indicates that the physical order of rows on this table is the same as the indexed order of rows. Only one clustered index per table is permitted.
- **nonclustered –** indicates that a level of indirection exists between the index structure and the data. Up to 249 nonclustered indexes per table are permitted.

- **fillfactor –** specifies how full the DBMS makes each page when it creates a new index on existing data. This percentage is relevant only at the time the index is created. As the data changes, the pages are not maintained at any level of fullness. The default is `0`. If the *fillfactor* is set to `100`, the DBMS creates indexes with pages 100% full.
- **ignore_dup_key –** responds to a duplicate key entry into any table with a unique index. Any attempted **insert** of a duplicate key is ignored, and the **insert** is cancelled with an informational message.
- **on segment_name –** specifies that the index is to be created on the named segment.
- **index_name –** is the name of the index. Index names must be unique within a table but need not be unique within a database.
- **table_name –** is the name of the table that contains the indexed column or columns.
- **column_name –** is the column or columns to be included in the index. Composite indexes are based on the combined values of up to 16 columns. The sum of the maximum lengths of all the columns used in a composite index cannot exceed 256 bytes.

Examples

- **Example 1 –** creates an index of `aud_id`.

```
create index au_id_ind
on authors (au_id)
```

- **Example 2 –** creates an index of `title_id`.

```
create index ind1
 on titleauthor (au_id, title_id)
```

- **Example 3 –** creates a nonclustered index of `zip_ind` with fillfactor.

```
create nonclustered index zip_ind
on authors (zip) with fillfactor = 25
```

Usage

- Adaptive Server/CIS sends the **create index** command to the DirectConnect server as a language event.
- Columns of type `bit`, `text`, and `image` cannot be indexed.
- You cannot create an index on a view.
- These parts of the **create index** command are not recognized by transformation:
  - **clustered** | **nonclustered**
  - **with fillfactor**
  - **max_rows_per_page**
  - **ignore_dup_key**
  - **sorted_data**
  - **ignore_dup_row** | **all_dup_row**
- **on segment**

- The ODBC command **ASC|DESC** cannot be generated by transformation. Only ascending indexes can be created.

create table (minimum)

Creates new tables and optional integrity constants.

Syntax

Transact-SQL Syntax

```
create table [database.[owner.]table_name (column_name datatype
[default {constant_expression | user | null}]
 {[{identity | null | not null}]
 | [[constraint constraint_name]
 {{unique | primary key}
 [clustered | nonclustered]
 [with {fillfactor | max_rows_per_page} = x]
 [on segment_name]
 | references [[database.]owner.]ref_table
[(ref_column)]
 | check (search_condition)}]}...
```

```
| [constraint constraint_name]
 {{unique | primary key}
 [clustered | nonclustered]
 (column_name [{, column_name}...])
 [with {fillfactor | max_rows_per_page} = x]
 [on segment_name]
 | foreign key (column_name [{, column_name}...])
 references [[database.]owner.]ref_table
[(ref_column [{, ref_column}...])]
 | [check (search_condition)}
```

```
[{, {next_column | next_constraint}}...])
```

```
[with max_rows_per_page = x][on segment_name]
```

ODBC Syntax

```
CREATE TABLE base_table_name
(column_element[,column_element]...)
```

```
column_element::=column_definition|
 table_constraint_definition
```

```
column_definition:=
 column_identifier datatype
[DEFAULT default_value]
 [column_constraint_definition
[column_constraint_definition]...]
```

```
default_value::=literal|NULL|USER
```

```
column_constraint_definition::=
 NOT NULL
 |UNIQUE|PRIMARY KEY
 |REFERENCES ref_table_name referenced_columns
```

```
table_constraint_definition::=
 UNIQUE(column_identifier[,column_identifier]...)
 |PRIMARY KEY(column_identifier[,column_identifier]...]
 |CHECK(search_condition)
 |FOREIGN KEY referencing_columns REFERENCES
 ref_table_name referenced_columns
```

Parameters

- **null | not null –** specifies a NULL value if you do not provide a value during an insertion and no default exists (for null), or that you must provide a non NULL value if no default exists (for not null).
- **next_column –** indicates that you can include additional column definitions (separated by commas) using the same syntax described for a column definition.
- **on segment_name –** specifies the name of the segment on which to place the table.
- **table_name –** is the name of the new table. It conforms to the rules for identifiers and is unique within the database and to the owner.
- **column_name –** is the name of the column in the table. It conforms to the rules for identifiers and is unique in the table.
- **datatype –** is the datatype of the column. Only system datatypes are used. See *Datatype Conversions of Transact-SQL to ODBC* on page 184 for several datatypes expect a length, n, in parentheses: *datatype* (n).

Examples

- **Example 1 –** creates the `titles` table.

  ```
  create table titles
       (title_id     tid     not null,
       title     varchar(80)     not null,
       type     char(12)     not null,
       pub_id     char(4)     null,
       price     money     null,
       advance     money     null,
       total_sales     int     null,
       notes     varchar(200)    null,
       pubdate     datetime     not null,
       contract     bit     not null)
  ```

Usage

- These Transact-SQL parts of the **create table** command are not recognized by **sybase** transformation mode:
  - **with fillfactor**
  - **clustered** | **nonclustered**

   - **with max_rows_per_page**
   - **on segment name**
- Transact-SQL allows you to specify null or notnull, with a default of not null. ODBC allows only not null to be specified. The default is null.

Table 18. Null Transformations During Transact-SQL to ODBC CREATE TABLE

Transact-SQL Specification	Transformed to
null	null
not null	not null
<nothing>	not null

The selection order is:

1. The access service attempts to change the Transact-SQL datatype to the primary ODBC datatype.
2. If the ODBC driver does not support the ODBC datatype, the access service uses the secondary ODBC datatype.
3. If the secondary ODBC datatype is also unsupported, the access service uses the final ODBC datatype.

Because the ODBC driver may not support extended datatypes such as `Tinyint` and `Bit`, a core ODBC type is associated with those datatypes as the second and third choices.

The size of the `char` value for `Nchar` and `Nvarchar` conversions should be doubled to accommodate the double-byte characters possible.

For `Text` secondary and final ODBC datatype values, and `Image` final ODBC datatype values, the original value can be truncated to fit the “transformed to” value.

`Binary`, `Varbinary`, and `Image` final ODBC datatype values can go through code set translation, which is not desirable for binary datatypes.

Datatype Conversion of Transact-SQL to ODBC

An access service transforms Transact-SQL datatypes to ODBC datatypes.

Transact-SQL Datatype	Primary ODBC Datatype	Secondary ODBC Datatype	Final ODBC Datatype
`Tinyint`	SQL_TINYINT	SQL_SMALLINT	SQL_INTEGER
`Smallint`	SQL_SMALLINT	SQL_INTEGER	SQL_INTEGER
`Int`	SQL_INTEGER	SQL_INTEGER	SQL_INTEGER

Transact-SQL Datatype	Primary ODBC Datatype	Secondary ODBC Datatype	Final ODBC Datatype
Numeric	SQL_NUMERIC	SQL_DECIMAL	SQL_FLOAT
Decimal	SQL_DECIMAL	SQL_NUMERIC	SQL_FLOAT
Float	SQL_FLOAT	SQL_DOUBLE	SQL_CHAR
Double Precision	SQL_DOUBLE	SQL_FLOAT	SQL_FLOAT
Real	SQL_REAL	SQL_FLOAT	SQL_FLOAT
Smallmoney	SQL_DECIMAL	SQL_NUMERIC	SQL_FLOAT
Money	SQL_DECIMAL	SQL_NUMERIC	SQL_FLOAT
Smalldatetime	TIMESTAMP	SQL_CHAR	SQL_CHAR
Datetime	TIMESTAMP	SQL_CHAR	SQL_CHAR
Char	SQL_CHAR	SQL_CHAR	SQL_CHAR
Varchar	SQL_VARCHAR	SQL_VARCHAR	SQL_VARCHAR
Nchar	1 (SQL_CHAR(2n))	SQL_CHAR	SQL_CHAR
Nvarchar	12 (SQL_VARCHAR(2n))	SQL_VARCHAR	SQL_VARCHAR
Text	SQL_LONGVARCHAR	SQL_VARCHAR	SQL_VARCHAR
Binary	SQL_BINARY	SQL_VARBINARY	SQL_CHAR
Varbinary	SQL_VARBINARY	SQL_LONGVARBINARY	SQL_VARCHAR
Image	SQL_LONGVARBINARY	SQL_LONGVARCHAR	SQL_VARCHAR
Bit	SQL_BIT	SQL_CHAR	SQL_CHAR

create view (core)

Creates a new view.

Syntax

Transact-SQL Syntax

```
create view [owner.]view_name
[(column_name [, column_name]...)]
```

```
 as select [distinct] select_statement
[with check option]
```

ODBC Syntax

```
CREATE VIEW viewed_table_name
```

```
[(column_identifier[,column_identifier]...)]
```

```
AS query_specification
```

Parameters

- **select –** begins the **select** statement that defines the view.
- **distinct –** specifies that the view cannot contain duplicate rows (optional).
- **with check option –** indicates that all data modification statements are validated against the view selection criteria. All rows inserted or updated through the view must remain visible through the view.
- **view_name –** is the name of the view. The view name cannot include the database name. It must conform to the rules for identifiers.
- **column_name –** is the name of the column in the view. It must conform to the rules for identifiers.
- **select_statement –** completes the **select** statement that defines the view. It can include more than one table and other views.

Examples

- **Example 1 –** creates the *new view* from the *old view*.

  ```
  create view “new view” (“column 1”, “column 2”)
   as select col1, col2 from “old view”
  ```

  Both columns are renamed in the new view. All view and column names that include embedded blanks are enclosed in double quotation marks. Before creating the view, you must set the `quoted_identifier` property to on.

Usage

- You can use views as security mechanisms by granting authorization on a view but not on its underlying tables.
- The **with check** option is removed from the transformed SQL text.

delete (minimum)

Removes rows from a table.

Syntax

Transact-SQL Syntax

```
delete [from][[database.]owner.]{view_name | table_name}
 [where search_conditions]
```

```
delete [[database.]owner.]{table_name | view_name}
 [from [[database.]owner.]{view_name | table_name
[(index index_name [prefetch size][lru | mru])]}
 [, [[database.]owner.]{view_name | table_name
(index index_name [prefetch size][lru | mru])]} ]...]
 [where search_conditions]
```

ODBC Syntax

```
DELETE FROM table_name [WHERE search_condition]
```

Parameters

- **from –** (after **delete**) allows you to name more than one table or view to use with a **where** clause when specifying the rows to delete. The **from** clause allows you to delete rows from one table based on data stored in other tables, giving you much of the power of an embedded **select** statement.
- **from –** (after *table_name* or *view_name*) is an optional keyword used for compatibility with other versions of SQL. Follow it with the name of the table or view from which you want to remove rows.
- **where –** is a standard **where** clause.
- **where current of –** *cursor_name* causes Adaptive Server to delete the table row or view indicated by the current cursor position for *cursor_name*.

Examples

- **Example 1 –** deletes author `McBadden`.

  ```
  delete from authors
   where au_lname = “McBadden”
  ```

Usage

- You cannot use **delete** with a multitable view.
- If you do not use a **where** clause, all rows are deleted from the table or view that is named after the **delete [from]** Transact-SQL keyword parameter.

- Upon completion of the **delete** command, the number of rows affected must be indicated.

delete (core)

Removes a row from a table (cursor event). The affected row must be made current by a read cursor.

Syntax

Transact-SQL Syntax

```
delete [from][[database.]owner.]{table_name | view_name}
 where current of cursor_name
```

ODBC Syntax

```
DELETE FROM table_name WHERE CURRENT OF cursor_name
```

Examples

- **Example 1 –** deletes title.

```

delete titles where current of title_crsr
```

Usage

- Adaptive Server/CIS issues a **cursor delete** request if it examines any column data to fulfill the client request. This is true when:
  - More than one table is involved in the **delete** statement.
  - The statement contains built-in functions in the **where** clause.
- Adaptive Server/CIS passes the **delete** command to the DirectConnect server as this series of cursor commands:
  - **declare**
  - **open**
  - **close**
  - **deallocate**

  The **where** clause is not constructed. Open Server appends the equivalent of **where current of cursor** `read_cursor_name`.
- The cursor can be reused multiple times before it is deallocated.
- When the DirectConnect server calls **srv_senddone** to mark the completion of the **delete** command, the number of affected rows must be indicated. Normally, the row count is `1`.
- Any valid object in the catalog can be substituted for *table_name* or *view_name*.

delete (minimum)

Removes rows from a table (dynamic event).

Syntax

Transact-SQL Syntax

```
delete [[database.]owner.]{table_name | view_name}
 [where column_name relop ? {AND | OR} column_name relop ? ...]]
```

Parameters

- **relop –** is a relational operation.
- **? –** is the parameter marker.

Usage

- Adaptive Server/CIS issues a dynamic **delete** request if it does not have to examine any column data to fulfill the client request. This is true when:
  - Only one table is involved in the **delete** statement.
  - The statement contains no built-in functions in its **where** clause.
- Supported relational operators are:
  =, <>, <, >, <=, >=, LIKE.
- Adaptive Server/CIS passes the dynamic **delete** command to DirectConnect as a series of dynamic requests:
  - **prepare**
  - **define** (parameter formats)
  - **execute** (with parameter data)
  - **deallocate**
- The **where** clause is optional. It is provided by Adaptive Server/CIS if the original **delete** command contained one.
- The **prepare** statement can execute multiple times before it is deallocated.
- Upon completion of the **delete** command, the number of affected rows must be indicated.
- Any valid object in the catalog can be substituted for *table_name*, *view_name*, and other variables.

drop index (core)

Removes an index from a table in the current database.

Syntax

Transact-SQL Syntax

```
drop index table_name.index_name
[, table_name.index_name]...
```

ODBC Syntax

```
DROP INDEX index_name
```

Parameters

- **table_name –** is the table in which the indexed column is located. The table must be in the current database.
- **index_name –** is the name of the index to be dropped.

Examples

- **Example 1 –** drops authors index.

  ```
  drop index authors.au_id_ind
  ```

Usage

- You can specify multiple index names in Transact-SQL, but ODBC supports only a single name per statement. If multiple names are present, multiple ODBC **DROP** statements are generated.
- Adaptive Server/CIS passes the **drop index** command to the DirectConnect server as a language event.
- To get information about existing indexes on a table:

  ```

  sp_helpindex table_name
  ```

drop table (minimum)

Removes a table definition and all of its data, indexes, triggers, and permissions from the database.

Syntax

Transact-SQL Syntax

```
drop table [[database.]owner.]table_name
[, [[database.]owner.]table_name]...
```

ODBC Syntax

```
DROP TABLE base_table_name
[CASCADE|RESTRICT]
```

Parameters

- **table_name –** is the name of the table to be dropped.

Usage

- Transact-SQL allows you to drop multiple tables in one statement, but ODBC does not. If multiple tables are encountered, transformation generates multiple **DROP** statements.
- ODBC allows the keywords *CASCADE* and *RESTRICT* in the statement. Since Transact-SQL does not support this, these keywords are not generated during transformation.
- Adaptive Server/CIS passes the **drop table** command to the DirectConnect server as a language event.

drop view (core)

Removes one or more views from the current database.

Syntax

Transact-SQL Syntax

```
drop view [owner.]view_name[,[owner.]view_name]...
```

ODBC Syntax

```
DROP VIEW viewed_table_name
[CASCADE|RESTRICT]
```

Parameters

- **view_name –** is the name of the view to be dropped. The name must be a legal identifier and cannot include a database name.

Examples

- **Example 1 –** removes the view *new_price* from the current database.

  ```
  drop view new_price
  ```

Usage

- Transact-SQL allows you to drop multiple views in one statement, but ODBC does not. If multiple views are encountered, transformation generates multiple **DROP** statements.
- ODBC allows the keywords *CASCADE* and *RESTRICT* in the statement. Because Transact-SQL does not support this, these keywords are not generated during transformation.
- Each time a view is referenced, another view or stored procedure checks the existence of the view.

execute

Runs a system procedure or a user-defined stored procedure.

Syntax

Transact-SQL Syntax

```
[execute][@return_status = ]
```

```
[[[server.]database.]owner.]procedure_name[; number]
 [[@parameter_name =] value |
 [@parameter_name =] @variable [output]
 [,[@parameter_name =] value |
 [@parameter_name =] @variable [output]...]]
```

```
[with recompile]
```

ODBC Syntax

```
{CALL PROCEDURE(parm1_value, parm2_value, ..., parmN_value)}
```

Usage

- This transformation uses only the shorthand notation ODBC syntax.
- Procedure return values are not supported.
- Procedures that return multiple result sets are not supported.

grant (core)

Assigns authorization to users.

Syntax

Transact-SQL Syntax

Grant authorization to access database objects:

```
grant {all [privileges] | permission_list}
 on {table_name [(column_list)]
 | view_name[(column_list)]
 | stored_procedure_name}
 to {public | name_list | role_name}
 [with grant option]
```

Grant authorization to create database objects:

```
grant {all [privileges] | command_list}
 to {public | name_list | role_name}
```

ODBC Syntax

```
GRANT {ALL|grant_privilege[,grant_privilege]...}
```

```
ON table_name
```

```
TO {PUBLIC|user_name[,user_name]...}
```

```
grant privilege::=
 DELETE
 | INSERT
 | SELECT
 | UPDATE[(column_identifier[,column_identifier]...)]
 | REFERENCES[(column_identifier[,column_identifier]
 ...)]
```

Parameters

- **all –** when used to assign authorization to access database objects (first syntax format), specifies that all privileges applicable to the specified object are granted or revoked.
- **public –** is all users of the "public" group, which includes all users of the system.
- **with grant option –** allows the users specified in *name_list* to grant the privileges specified by *permission_list* to other users.
- **permission_list –** is a list of authorizations granted.
- **command_list –** is a list of commands granted.
- **table_name –** is the name of a table in the database.
- **column_list –** is a list of columns, separated by commas, to which the privileges apply.
- **view_name –** is the name of a view in the current database. Only one view can be listed for each **grant** command.
- **stored_procedure –** is the name of a stored procedure in the database.
- **name_list –** is a list of user database names or group names or both, separated by commas.
- **role_name –** is the name of an Adaptive Server role. Use it to grant authorizations to all users who have been granted a specific role.

Examples

- **Example 1 –** grants insert and delete permissions on `titles` table to username Mary.

  ```
  grant insert, delete
   on titles
   to mary, sales
  ```

- **Example 2 –** grants update permission on `titles` table to all users.

```
grant update
 on titles (price, advance)
 to public
```

- **Example 3 –** grants create database and create table permission to usernames Mary and John.

```
grant create database, create table
 to mary, john
```

- **Example 4 –** grants update permission on `authors` table to username Mary.

```
grant update on authors
 to mary
 with grant option
```

Usage

- Any valid object in the catalog can be substituted for *table_name* or *view_name*.
- **with grant** optionis not available. Transformation removes this phrase.
- ODBC does not allow you to grant authorization to a stored procedure.
- You can substitute *from* for *to* in the **grant** syntax.
- You can **grant** or **revoke** authorizations only on objects in the current database.

insert (minimum)

Adds new rows to a table or view.

Syntax

Transact-SQL Syntax

```
insert [into][database.[owner.]]{table_name | view_name}
 [(column_list)]
 {values (expression [, expression]...)
 | select_statement}
```

ODBC Syntax

```
INSERT INTO table_name[(column_identifier[,column_identifier]
 ...)]
```

```
{query_specification|VALUES(insert_value
[,insert_value]...)
```

Parameters

- **values –** is a keyword that introduces a list of expressions.

- **? –** specifies parameters passed by Adaptive Server/CIS at the time the **insert** command is executed.
- **column_list –** is a list of one or more columns to which data is to be added. The columns can be in any order, but the incoming data (whether in a **values** clause or **select** clause) is in the same order.

Examples

- **Example 1 –** inserts new titles with `?` expression values.

```
insert titles
 (title_id, title, type, pub_id, notes, pubdate,contract)
 values (?, ?, ?, ?, ?, ?, ?)
```

Usage

- Any valid object in the catalog can be substituted for *table_name*, *view_name*, and so on.
- Adaptive Server/CIS passes the **insert** command to DirectConnect as this series of dynamic SQL commands:

  prepare

  execute

  close

  deallocate
- The values in the values list are passed as dynamic SQL parameters.

prepare transaction

Used by a DB-Library™ in a two-phase **commit** application to determine whether a server is prepared to commit a transaction.

Syntax

Transact-SQL Syntax

```
prepare tran[saction]
```

ODBC Syntax

Not supported.

revoke (core)

Revokes permissions from users.

Syntax

Transact-SQL Syntax

Revoke permission to access database objects:

```
revoke [grant option for]
 {all [privileges] | permission_list}
 on {table_name [column_list)]
 | view_name [(column_list)]
 | stored_procedure_name}
 from {public | name_list | role_name}
 [cascade]
```

Revoke permission to create database objects:

```
revoke {all [privileges] | command_list}
 from {public | name_list | role_name}
```

ODBC Syntax

```
REVOKE {ALL|revoke_privilege[,revoke_privilege]...}
```

```
ON table_name
```

```
FROM {PUBLIC|user_name[,user_name]...}
```

```
[CASCADE|RESTRICT]
```

```
revoke_privilege::=
 DELETE
 |INSERT
 |SELECT
 |UPDATE
 |REFERENCES
```

This statement revokes authorization from users.

Parameters

- **all –** specifies that all privileges applicable to the specified object are revoked when used to revoke authorizations to access database objects (first syntax format).
- **public –** is all users of the “public” group, which includes all system users.
- **grant option for –** prohibits the users specified in *name_list* from granting the privileges specified by *permission_list* to other users.
- **cascade –** revokes **grant** authorization for the privileges specified in *permission_list* from the users specified in *name_list* and from all users to whom they granted privileges.

  The cascading effect occurs even if it is not specified by the user. For example, suppose User A has granted User B privileges, and in turn, User B granted privileges to User C. If

User A is revoked, all privileges that User A granted to User B and User B indirectly granted to User C are revoked.

- **permission_list –** is a list of authorizations to be revoked.
- **command_list –** is a list of commands for which authorizations are to be revoked.
- **table_name –** is the name of a table in the database.
- **column_list –** is a list of columns, separated by commas, to which the privileges apply. If columns are specified, only **select** and **update** authorizations can be revoked.
- **view_name –** is the name of a view in the current database. Only one view can be listed for each **revoke** statement.
- **stored_procedure –** is the name of a stored procedure in the database. Only one object can be listed for each **revoke** statement.
- **name_list –** is a list of user database names and group names, separated by commas.
- **role_name –** is the name of an Adaptive Server role. This allows you to **revoke** from all users who have been granted a specific role.

Examples

- **Example 1 –** revokes insert and delete permission on `titles` table for username Mary.

```
revoke insert, delete
 on titles
 from mary, sales
```

- **Example 2 –** revokes update permission on `titles` table from all users.

```
revoke update
 on titles (price, advance)
 from public
```

- **Example 3 –** revokes create database and create table permissions.

```
revoke create database, create table
 from mary, john
```

- **Example 4 –** revokes execute permission from `oper_role`.

```
revoke execute on new_sproc
from oper_role
```

Usage

- Valid permissions for Transact-SQL are:
  - **select**
  - **insert**
  - **delete**
  - **update**
  - **references**
- ODBC does not support revoking a stored procedure.

- Authorizations can be revoked only on objects in the current database.
- **grant** and **revoke** commands are order-sensitive. When a conflict occurs, the most recently issued command takes effect.
- *to* can be substituted for *from* in the **revoke** syntax.

rollback transaction

Rolls back a user-specified transaction to the last savepoint inside the transaction or to the beginning of the transaction.

Syntax

Transact-SQL Syntax

```
rollback {transaction | work}
 [transaction_name | savepoint_name]
```

ODBC Syntax

ODBC does not support **rollback** as a SQL command. ECDA Option for ODBC traps the command and monitors the transaction state with the child process. If a transaction is opened, a call to **SQL Transact()** is generated.

Parameters

- **transaction_name –** is the name assigned to the transaction. It must conform to the rules for identifiers.

Usage

- The *transaction name* and *savepoint name* are ignored. Only one pending transaction is allowed for each connection.
- *transaction_name* is not used with Adaptive Server/CIS 10.5.
- The access service strips the *transaction_name* from the statement before passing it to the target.

select (minimum)

Retrieves rows from database objects.

Syntax

Transact-SQL Syntax

```
select [all | distinct] select_list
 [into [[database.]owner.]table_name]
 [from [[database.]owner.]{view_name | table_name
[(index index_name [prefetch size][lru | mru])]}
 [holdlock | noholdlock] [shared]
 [,[[database.]owner.]{view_name | table_name
[(index index_name [prefetch size][lru | mru])]}
 [holdlock | noholdlock] [shared]]...]
```

```
[where search_conditions]
```

```
[group by [all]aggregate_free_expression
[, aggregate_free_expression]...]
 [having search_conditions]
```

```
[order by
 {[[[database.]owner.]{table_name. | view_name.}]
 column_name | select_list_number | expression}
 [asc | desc]
 [,{[[[database.]owner.]{table_name. | view_name.}]
 column_name | select_list_number | expression}
 [asc | desc]]...]
```

```
[compute row_aggregate(column_name)
 [, row_aggregate(column_name)]...
 [by column_name [, column_name]...]]
```

```
[for {{read only | update [of column_name_list]}]
```

```
[at isolation {read uncommitted | read committed | serializable}]
```

```
[for browse]
```

ODBC Syntax

```
SELECT [ALL|DISTINCT]select_list
```

```
FROM table_reference_list
```

```
[WHERE search_condition]
```

```
[GROUP BY column_name[,column_name]...]
```

```
[HAVING search_condition]
```

```
[UNION [ALL]select_statement]...
 [order_by_clause]
```

An alternate syntax for updating tables if the driver supports core or extended functionality:

```
SELECT [ALL|DISTINCT]select_list
```

```
FROM table_reference_list
```

```
[WHERE search_condition]
```

```
FOR UPDATE OF [column_name[,column_name]...]
```

Parameters

- **all –** includes all rows in the result.
- **from –** indicates a comma-separated list of tables or views to use in the **select** statement.

- **group by –** finds a value for each group. These values appear as new columns in the results, rather than as new rows.
- **order by –** sorts the results by columns.
- **having –** sets conditions for the **group by** clause, similar to the way that **where** sets conditions for the **select** clause. No limit exists for the number of conditions that can be included.
- **union –** returns a single result set that combines the results of two or more queries. Duplicate rows are eliminated from the result set unless the **all** keyword is specified.
- **read only –** indicates that the cursor is a read-only cursor and that updates cannot be applied to rows made current by it.
- **update –** indicates that the cursor is an updatable cursor, and that the rows it makes current can be deleted or updated.
- **select_list –** is one or more of these items:
  - A list of column names in the order in which you want them returned
  - An aggregate function
  - Any combination of these items
- **table_name –** and *view_name* list tables and views used in the **select** statement.

  If more than one table or view is in the list, the names are separated by commas. Table names and view names are given correlating names. You can do this by providing the table or view name, then a space, then the correlation name, for example:

  ```
  select * from publishers t1, authors t2
  ```
- **search_conditions –** sets the conditions for the rows that are retrieved. A search condition can include column names, constants, joins, the keywords **is null**, **is not null**, **or**, **like**, **and**, or any combination of these items.

Examples

- **Example 1 –** counts publishers with read-only status.

  ```
  select count(*) from publishers for read only
  ```
- **Example 2 –** selects `pub_id`, `pub_name`, `city`, and `name` from publishers.

  ```
  select pub_id, pub_name, city, state from publishers for read only
  ```
- **Example 3 –** selects type and price from `titles` table.

  ```
  select type, price from titles
   where price > @p1 for update of price
  ```
- **Example 4 –** selects `stor_id` and `stor_name` from `sales_east` table.

  ```
  select stor_id, stor_name from sales union
   select stor_id, stor_name from sales_east
  ```

Usage

- You can issue this command as a language command or a client-based cursor request.
- This statement is accepted and sent to ODBC without change, subject to the qualifications listed in this section.
- The **TEXTPTR()** function cannot appear in the select list.
- These Microsoft SQL Server 10.x aggregate functions are supported:
  - **sum ([all | distinct])**
  - **avg ([all | distinct])**
  - **count ([all | distinct])**
  - **count (*)**
  - **max** *(expression)*)
  - **min** *(expression)*)
- If the command is issued as a cursor, the access service library supports these cursor commands:
  - **declare**
  - **open**
  - **fetch**
  - **close**
  - **deallocate**
- If a cursor is passed a new set of parameters before it is opened, it can be reused multiple times.
- The data values used in the **where** clause search conditions are passed as cursor parameters, using the datatype associated with the column.
- Cursor parameters are indicated with the "@" character when Transact-SQL syntax is used, and with a question mark when **passthrough** mode is used.
- These are not supported:
  - Transact-SQL **select into** syntax
  - The use of **index** in a **from** clause
  - The use of **prefetch size** in a **from** clause
  - The use of **holdlock|noholdlock|shared** in a **from** clause
  - The **compute** phrase
  - The **at isolation** phrase
  - The **for browse** phrase
- The availability of the **GROUP BY**, **HAVING**, and **UNION** clauses depends upon the ODBC driver level of conformance.

truncate table (extension using where 1=1)

Removes all rows from a table.

Syntax

Transact-SQL Syntax

```
truncate table [[database.]owner.]table_name
```

ODBC Syntax

Not supported.

The command can be transformed into an equivalent **delete** command as:

```
DELETE FROM table_name
```

Parameters

- **table_name –** is the name of the table to be truncated.

Usage

Adaptive Server/CIS passes the command to the DirectConnect server as a language event.

update (core)

Changes existing rows by adding data or modifying existing data.

Syntax

Transact-SQL Syntax

```
update [[database.]owner.]{table_name | view_name}
```

```
set [[[database.]owner.]{table_name. | view_name.}]
 column_name1 =
 {expression1 | NULL | (select_statement)}
 [, column_name2 =
 {expression2 | NULL | (select_statement)}]...
```

```
[from [[database.]owner.]{view_name | table_name
```

```
[(index index_name [prefetch size][lru | mru])]}
 [, [[database.]owner.]{view_name | table_name
```

```
[(index index_name [prefetch size][lru | mru])]}]...]
 [where search_conditions]
```

ODBC Syntax

```
UPDATE table_name
```

```
SET column_identifier={expression|NULL}
 [,column_idenfifier={expression|NULL}]...
```

```
[WHERE search_condition]
```

Parameters

- **set –** specifies the column name and assigns the new value. The value can be an expression or a NULL. More than one column name or value pair must be separated by commas.
- **where –** is a standard **where** clause.

Examples

- **Example 1 –** updates `au_lname` value in the `authors` table.

  ```
  update authors
   set au_lname = "MacBadden"
   where au_lname = "McBadden"
  ```

Usage

- Use **update** to change values in rows that have already been inserted. Use **insert** to add new rows.
- You cannot update views defined with the **distinct** clause.
- ODBC does not support a **from** clause in the **update** statement. The access service ignores the **from** clause.
- **select** statements are not supported.
- The "index" phrase is not supported.

update (core)

Changes data in a row made current by a read cursor by adding data or modifying existing data (cursor event).

Syntax

Transact-SQL Syntax

```
update [[database.]owner.]{table_name | view_name}
```

```
set [[[database.]owner.]{table_name. | view_name.}]
 column_name1 =
 {expression1 | NULL | (select_statement)}
 [, column_name2 =
 {expression2 | NULL | (select_statement)}
 where current of cursor_name
```

ODBC Syntax

```
UPDATE table_name
```

```
SET column_idenfifier={expression|NULL}
 [,column_identifier={expression|NULL}]...
```

```
WHERE CURRENT OF cursor_name
```

Parameters

- **set –** specifies the column name and assigns the new value. The value is passed as a cursor parameter.
- **where current of –** causes Adaptive Server to update the row of the table or view indicated by the current cursor position for *cursor_name*.

Examples

- **Example 1 –** The row made current by the cursor *authors_cursor* is modified. The column *au_lname* is set to the value of the parameter `@p1`.

  ```
  update authors
  ```

  ```
  set au_lname = @p1
  ```

Usage

- Any valid object in the catalog can be substituted for *table_name, view_name*, and so forth.
- Adaptive Server/CIS issues a cursor **update** request if it must examine any column data to fulfill the client request, subject to these conditions:
  - More than one table is involved in the **update** statement.
  - The statement contains built-in functions in its **where** clause.
  - A column name is referenced to the right of any **set** expression.
- Adaptive Server/CIS passes the **update** command to the DirectConnect server as this series of cursor commands:
  - **declare** (define parameter formats)
  - **open** (define parameter data)
  - **close**
  - **deallocate**
- The cursor can be reused multiple times before it is deallocated.
- Upon completion of the **update** command, the number of rows affected must be indicated.
- ODBC does not support a **from** clause in the **update** statement. The access service ignores the **from** clause.

update (core)

Changes data in existing rows of the referenced table (dynamic event).

Syntax

Transact-SQL Syntax

```
update [[database.]owner.]{table_name | view_name}
 set column_name1 = ?
 [, column_name2 = ?]...
 [ where column_name relop ?
 [ {AND | OR} column_name relop ? ...]]
```

Parameters

- **set –** specifies the column name and assigns the new value. The value is passed as a parameter.
- **relop –** is a relational operation.

Examples

- **Example 1 –** sets the *au_lname* column to the value of *<parameter 1>,* where the value of *au_id* is equal to the value of *<parameter 2>*.

  ```
  update authors
   set au_lname = ?
   where au_id = ?
  ```

Usage

- You can substitute *table_name* and *view_name* with any valid object in the catalog.
- Adaptive Server/CIS issues a dynamic **update** request if it does not need to examine any column data to fulfill the client request, subject to these conditions:
  - Only one table is involved in the **update** statement.
  - The statement does not contain built-in functions in its **where** clause.
  - A column name is not referenced to the right of any **set** expression.
- These relational operators (relops) are supported in search conditions:

  =, <>, <, >, <=, >=, LIKE
- Adaptive Server/CIS passes the **update** command to DirectConnect as this series of dynamic requests:
  - **prepare** (define parameter formats)
  - **execute** (define parameter data)
  - **deallocate**

-
- Upon completion of the **update** command, the number of rows affected must be indicated.
- ODBC does not support a **from** clause in the **update** statement. The access service ignores the **from** clause.

use

Specifies the database with which you want to work.

Syntax

Transact-SQL Syntax

```
use database_name
```

ODBC Syntax

No ODBC SQL equivalent.

Parameters

- **database_name –** is the name of the database you want to access.

Usage

Supported only for drivers and targets that support the ODBC SQL_CURRENT_QUALIFIER connection option. For these targets, the current connection is dropped, and a new connection is opened with SQL_CURRENT_QUALIFIER set to *database_name*.

Obtaining Help and Additional Information

Use the Sybase Getting Started CD, Product Documentation site, and online help to learn more about this product release.

- The Getting Started CD (or download) – contains release bulletins and installation guides in PDF format, and may contain other documents or updated information.
- Product Documentation at *http://sybooks.sybase.com/* – is an online version of Sybase documentation that you can access using a standard Web browser. You can browse documents online, or download them as PDFs. In addition to product documentation, the Web site also has links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, Community Forums/Newsgroups, and other resources.
- Online help in the product, if available.

To read or print PDF documents, you need Adobe Acrobat Reader, which is available as a free download from the *Adobe* Web site.

Note: A more recent release bulletin, with critical product or document information added after the product release, may be available from the Product Documentation Web site.

Technical Support

Get support for Sybase products.

If your organization has purchased a support contract for this product, then one or more of your colleagues is designated as an authorized support contact. If you have any questions, or if you need assistance during the installation process, ask a designated person to contact Sybase Technical Support or the Sybase subsidiary in your area.

Downloading Sybase EBFs and Maintenance Reports

Get EBFs and maintenance reports from the Sybase Web site or the SAP® Service Marketplace (SMP). The location you use depends on how you purchased the product.

- If you purchased the product directly from Sybase or from an authorized Sybase reseller:
  a) Point your Web browser to *http://www.sybase.com/support*.
  b) Select **Support > EBFs/Maintenance**.
  c) If prompted, enter your MySybase user name and password.
  d) (Optional) Select a filter, a time frame, or both, and click **Go**.
  e) Select a product.

Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as an authorized support contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click **My Account** to add the "Technical Support Contact" role to your MySybase profile.

f) Click the **Info** icon to display the EBF/Maintenance report, or click the product description to download the software.

- If you ordered your Sybase product under an SAP contract:
  a) Point your browser to *http://service.sap.com/swdc* and log in if prompted.
  b) Select **Search for Software Downloads** and enter the name of your product. Click **Search**.

Sybase Product and Component Certifications

Certification reports verify Sybase product performance on a particular platform.

To find the latest information about certifications:

- For partner product certifications, go to *http://www.sybase.com/detail_list?id=9784*
- For platform certifications, go to *http://certification.sybase.com/ucr/search.do*

Creating a MySybase Profile

MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Go to *http://www.sybase.com/mysybase*.
2. Click **Register Now**.

Accessibility Features

Accessibility ensures access to electronic information for all users, including those with disabilities.

Documentation for Sybase products is available in an HTML version that is designed for accessibility.

Vision impaired users can navigate through the online document with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Sybase HTML documentation has been tested for compliance with accessibility requirements of Section 508 of the U.S Rehabilitation Act. 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.

Note: You may 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 the Sybase Accessibility site: *http://www.sybase.com/products/accessibility*. The site includes links to information about Section 508 and W3C standards.

You may find additional information about accessibility features in the product documentation.

Glossary

Glossary of terms used in Enterprise Connect™ Data Access.

- **accept –** establishment of a SNA or TCP/IP connection between Mainframe Connect™ Server Option and Mainframe Connect DirectConnect for z/OS Option.
- **access service –** the named set of properties, used with an access service library, to which clients connect. Each DirectConnect server can have multiple services.
- **access code –** a number or binary code assigned to programs, documents, or folders that allows authorized users to access them.
- **access service library –** a service library that provides access to non-Sybase data contained in a database management system or other type of repository. Each such repository is called a "target." Each access service library interacts with exactly one target and is named accordingly. See also *service library*.
- **ACSLIB –** see *access service library*.
- **Adaptive Server Enterprise –** the server in the Sybase client/server architecture. Adaptive Server manages multiple databases and multiple users, tracks the actual location of data on disks, maintains mapping of logical data description to physical data storage, and maintains data and procedure caches in memory.
- **Adaptive Server Enterprise/Component Integration Services –** includes a variation of Adaptive Server that provides a Transact-SQL interface to various sources of external data. Component Integration Services allows Adaptive Server to present a uniform view of enterprise data to client applications.
- **administrative service library –** a service library that provides remote management capabilities and server-side support. It supports a number of remote procedures, invoked as RPC requests, that enable remote DirectConnect server management. See also *remote procedure call*, *service library*.
- **ADMLIB –** see *administrative service library*.
- **Advanced Interactive Executive –** the IBM implementation of the UNIX operating system. The RISC System/6000, among other workstations, runs the AIX operating system.
- **advanced program-to-program communication –** hardware and software that characterizes the LU 6.2 architecture and its implementations in products. See also *logical unit 6.2*.
- **AIX –** see *Advanced Interactive Executive*.
- **AMD2 –** the component of the Mainframe Connect DB2 UDB Option that allows clients to submit SQL statements to DB2 UDB. It is a CICS transaction that receives SQL statements sent from Mainframe Connect DirectConnect for z/OS Option and submits them to DB2 UDB, using the DB2 UDB dynamic SQL facility. It also receives the results

and messages from DB2 UDB and returns them to Mainframe Connect DirectConnect for z/OS Option.

- **American Standard Code for Information Interchange –** the standard code used for information interchange among data processing systems, data communication systems, and associated equipment. The code uses a coded character set consisting of 7-bit coded characters (including a parity check, 8 bits).
- **API –** see *application program interface*.
- **APPC –** see *advanced program-to-program communication*.
- **application program interface –** the programming language interface between the user and Mainframe Connect Client Option or Mainframe Connect Server Option. The API for Mainframe Connect Client Option is Client-Library. The API for Mainframe Connect Server Option is Gateway-Library.
- **ASCII –** see *American Standard Code for Information Interchange*.
- **Adaptive Server Enterprise –** see *Adaptive Server Enterprise*.
- **Adaptive Server Enterprise /CIS –** see *Adaptive Server Enterprise/Component Integration Services*.
- **batch –** a group of records or data processing jobs brought together for processing or transmission.
- **bind –** in the Sybase environment, this term has different meanings depending on the context:
  - In CICS, bind is an SNA command that establishes a connection between LUs, or a TCP/IP call that connects an application to a port on its system.
  - In DB2 UDB, bind compiles the Database Request Module, the precompiler product that contains SQL statements in the incoming request, and produces an access plan, a machine code version of the SQL statements that specifies the optimal access strategy for each statement.
  - In the mainframe access product set, bind establishes a connection between a TRS port and a CICS or IMS region.
- **bulk copy transfer –** a transfer method in which multiple rows of data are inserted into a table in the target database. Compare with *destination-template transfer* and *express transfer*.
- **call level interface –** a programming style that calls database functions directly from the top level of the code. Contrast with *embedded SQL.*
- **catalog –** a system table that contains information about objects in a database, such as tables, views, columns, and authorizations.
- **catalog RPC –** a component of the Mainframe Connect DB2 UDB Option that allows clients to access DB2 UDB system catalogs. It uses an interface compatible with the catalog interface for the ODBC API.
- **catalog stored procedure –** a procedure used in SQL generation and application development that provides information about tables, columns, and authorizations.

- **character set –** a set of specific (usually standardized) characters with an encoding scheme that uniquely defines each character. ASCII is a common character set.
- **CICS –** see *Customer Information Control System*.
- **CICS region –** the instance of CICS.
- **client –** in client/server systems, the part of the system that sends requests to servers and processes the results of those requests. See also *client/server*. Compare with *server*.
- **client application –** software responsible for the user interface that sends requests to applications acting as servers. See also *client/server*.
- **Client-Library –** a library of routines that is part of Mainframe Connect Client Option.
- **client request –** an RPC or language request sent by a client to a server.
- **client/server –** an architecture in which the client is an application that handles the user interface and local data manipulation functions, and the server is an application providing data processing access and management. See also *client application*.
- **Client Services Application –** a customer-written CICS program initiated on the host that uses the API to invoke the Mainframe Connect Client Option as a client to the ECDA Option for Oracle server or to Adaptive Server. See also *application program interface*, *Client Services for CICS*.
- **Client Services for CICS –** a Sybase host API that invokes the Mainframe Connect Server Option as a client to an access service for DB2 UDB or Adaptive Server. See also *application program interface*, *Customer Information Control System*, *Client Services Application*, *Mainframe Connect Server Option*.
- **clustered index –** an index in which the physical order and the logical (indexed) order is the same. Compare with *nonclustered index*.
- **code page –** an assignment of graphic characters and control function meanings to all code points.
- **commit –** a process that makes permanent all changes made to one or more database files since the initiation of the application program, the start of an interactive session, or the last **commit** or **rollback** operation. Compare with *rollback*.
- **Common Programming Interface –** specifies the languages and services used to develop applications across SAA environments. The elements of the CPI specification are divided into two parts: processing logic and services.
- **configuration file –** a file that specifies the characteristics of a system or subsystem.
- **configuration set –** a section into which service library configuration files are divided.
- **conversion –** the transformation between values that represent the same data item but which belong to different datatypes. Information can be lost due to conversion, because accuracy of data representation varies among different datatypes.
- **connection –** a network path between two systems. For SNA, the path connects a logical unit (LU) on one machine to an LU on a separate machine. For TCP/IP, the path connects TCP modules on separate machines.
- **connection router –** a program provided with Mainframe Connect Client Option that directs requests to particular remote servers. Mainframe system programmers use the

connection router to define remote servers and server connections to Mainframe Connect Client Option.

- **Connection Router Table –** a memory-resident table maintained by a Mainframe Connect Client Option system programmer that lists servers and the connections that a Client-Library transaction can use to access them.
- **control section –** the part of a program specified by the programmer to be a relocatable unit, all elements of which are to be loaded into adjoining main storage locations.
- **control statement –** in programming languages, a statement that is used to alter the continuous sequential execution of statements. A control statement can be a conditional statement or an imperative statement.
- **conversation-level security –** the passing of client login information to the mainframe by TRS when it allocates a conversation.
- **CSA –** see *Client Services Application*.
- **CSP –** see *catalog stored procedure*.
- **cursor –** in SQL, a named control structure used by an application program to point to a row of data.
- **Customer Information Control System –** an IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs.
- **DASD –** see *direct access storage device*.
- **data definition statement –** an IBM mainframe statement that relates a name to a file.
- **data definition language –** a language for describing data and data relationships in a database.
- **data set name –** the term or phrase used to identify a data set.
- **database management system –** a computer-based system for defining, creating, manipulating, controlling, managing, and using databases.
- **database operation –** a single action against the database. For Mainframe Connect DirectConnect for z/OS Option, a database operation is usually a single SQL statement. One or more database actions can be grouped together to form a request. See also *request*.
- **Database 2 –** an IBM relational database management system.
- **datatype –** a keyword that identifies the characteristics of stored information on a computer.
- **DB-Library –** a Sybase and Microsoft API that allows client applications to interact with ODS applications. See also *application program interface*.
- **DBMS –** see *database management system*.
- **DB2 UDB –** see *Database 2*.
- **DDL –** See *data definition language*.
- **DD statement –** see *data definition statement*.
- **default language –** the language that displays a user’s prompts and messages.
- **destination-template transfer –** a transfer method in which source data is briefly put into a template where the user can specify that some action be performed on it before execution

against a target database. See also *transfer*. Compare with *bulk copy transfer* and *express transfer*.

- **direct access storage device –** a device in which access time is effectively independent of the location of the data.
- **direct request –** a request sent directly from a client workstation through Transaction Router Service to the DirectConnect server without going through Adaptive Server. Contract with *indirect request*.
- **direct resolution –** a type of service name resolution that relies upon a client application specifying the exact name of the service to be used. See also *service name resolution*. Compare with *service name redirection*.
- **DirectConnect Manager –** a Java application from Sybase that can be used in Windows and UNIX environments. It provides remote management capabilities for DirectConnect products, including starting, stopping, creating, and copying services.
- **ECDA Option for Oracle server –** the component of Mainframe Connect DirectConnect for z/OS Option that provides general management and support functions to service libraries.
- **dll –** see *dynamic link library*.
- **DSN –** see *data set name*.
- **dynamic link library –** a file containing executable code and data bound to a program at load time or runtime, rather than during linking.
- **dynamic SQL –** the preparation and processing of SQL source statements within a program while the program runs. The SQL source statements are contained in host-language variables rather than being coded directly into the application program. Contrast with *static SQL*.
- **ECDA –** see *Enterprise Connect Data Access*.
- **ECDA Option for ODBC –** a Sybase solution that allows client applications to access ODBC data. It combines the functionality of the ECDA Option for ODBC architecture with ODBC to provide dynamic SQL access to target data, as well as the ability to support stored procedures and text and image pointers.
- **ECDA Option for Oracle –** a Sybase solution that provides Open Client access to Oracle databases. When used in combination with Adaptive Server, it provides many of the features of a distributed database system, such as location transparency, copy transparency, and distributed joins.
- **Embedded SQL™ –** SQL statements that are embedded within a program and are prepared in the process before the program runs. After it is prepared, the statement itself does not change, although values of host variables specified within the statement might change.
- **end user –** a person who connects to a DirectConnect server using an application to access databases and perform transfers. See also *transfer*.
- **Enterprise Connect Data Access –** an integrated set of software applications and connectivity tools that allow access to data within a heterogeneous database environment,

such as a variety of LAN-based, non-Sybase datasources, as well as mainframe data sources.

- **environment variable –** a variable that describes how an operating system runs and the devices it recognizes.
- **exit routine –** a user-written routine that receives control at predefined user exit points.
- **express transfer –** a form of bulk copy transfer that uses ODBC bulk APIs to improve performance when transferring bulk data between datasources. Because it uses the same syntax as bulk copy transfer, no modification of applications is required.
- **external call interface –** a CICS client facility that allows a program to call a CICS application as if the calling program had been linked synchronously from a previous program instead of started from a terminal.
- **External Security Manager –** an add-on security package for the z/OS mainframe, licensed by Computer Associates.
- **FCT –** see *forms control table*.
- **forms control table –** an object that contains the special processing requirements for output data streams received from a host system by a remote session.
- **gateway –** connectivity software that allows two or more computer systems with different network architectures to communicate.
- **Gateway-Library –** a library of communication, conversion, tracing, and accounting functions supplied with Mainframe Connect Server Option.
- **globalization –** the combination of internationalization and localization. See *internationalization*, *localization*.
- **global variable –** a variable defined in one portion of a computer program and used in at least one other portion of the computer program. Contrast with *local variable*.
- **handler –** a routine that controls a program’s reaction to specific external events, for example, an interrupt handler.
- **host –** the mainframe or other machine on which a database, an application, or a program resides. In TCP/IP, this is any system that is associated with at least one Internet address. See also *Transmission Control Protocol/Internet Protocol*.
- **host ID –** in Mainframe Connect Server Option, the ID that the TRS passes to the mainframe with a client request. The host ID is part of the client login definition at the TRS.
- **host password –** in Mainframe Connect Server Option, the password that the client passes to the mainframe with a client request.
- **host request library –** a DB2 UDB table that contains host-resident SQL statements that can be executed dynamically. See also *host-resident request*.
- **host-resident request –** a SQL request that resides in a DB2 UDB table called the host request library. See also *host request library*.
- **IMS –** see *Information Management System*.
- **indirect request –** a client request that is routed through a stored procedure on a SQL Server, which forwards the request to TRS as an RPC. Compare with *direct request*.

- **Information Management System –** a database/data communication system that can manage complex databases and networks.
- **interfaces file –** an operating system file that determines how the host client software connects to a Sybase product. An `interfaces` file entry contains the name of any ECDA Option for Oracle server and a list of services provided by that server.
- **internationalization –** the process of extracting locale-specific components from the source code and moving them into one or more separate modules, making the code culturally neutral so it can be localized for a specific culture. See also *globalization*. Compare with *localization*.
- **keyword –** a word or phrase reserved for exclusive use by Transact-SQL.
- **language RPC –** the name TRS uses to represent a client's language request. TRS treats a language request as a remote procedure call (RPC) and maps it to a language transaction at the remote server.
- **language transaction –** the server transaction that processes client language requests. The Mainframe Connect DB2 UDB Option language transaction for CICS is **AMD2**, which uses the DB2 UDB dynamic SQL facilities to process incoming SQL strings. The Mainframe Connect DB2 UDB Option for IMS uses **SYRT** by default.
- **linkage –** in computer security, combining data or information from one information system with data or information from another system with the intention to derive additional information; for example, the combination of computer files from two or more sources.
- **linkage editor –** a computer program that creates load modules from one or more object modules or creates load modules by resolving cross references among the modules, and if necessary, adjusts those addresses.
- **link-edit –** to create a loadable computer program by using a linkage editor. See also *linkage editor*.
- **localization –** the process of preparing an extracted module for a target environment, in which messages are displayed and logged in the user's language. Numbers, money, dates, and time are represented using the user's cultural convention, and documents are displayed in the user's language. See also *globalization*.
- **local variable –** a variable that is defined and used only in one specified portion of a computer program. Contrast with *global variable*.
- **logical unit –** a type of network addressable unit that enables a network user to gain access to network facilities and communicate remotely. A connection between a TRS and a CICS region is a connection between logical units.
- **logical unit 6.2 –** a type of logical unit that supports general communication between programs in a distributed processing environment. See also *advanced program-to-program communication*.
- **login ID –** in Mainframe Connect Server Option, the ID that a client user uses to log in to the system.

- **login packet –** client information made available to Mainframe Connect Server Option. The client program sets this information in a login packet and sends it to TRS, which forwards it to the mainframe.
- **long-running transaction –** a transaction that accepts more than one client request. Whereas short transactions end the communication after returning results to a client, a long-running transaction can await and process another request. Compare with *short transaction*.
- **LU 6.2 –** see *logical unit 6.2*.
- **mainframe access products –** Sybase products that enable client applications to communicate with mainframes in a client/server environment. See *client/server*.
- **Mainframe Connect –** the Sybase product set that provides access to mainframe data.
- **Mainframe Connect Client Option –** a Sybase product that, using Client-Library, allows mainframe clients to send requests to SQL Server, Open Server, the Mainframe Connect DB2 UDB Option and Mainframe Connect Server Option. Mainframe Connect Client Option provides capability for the mainframe to act as a client to LAN-based resources in the CICS or the IMS and MVS environment.
- **Mainframe Connect DB2 UDB Option –** a Sybase mainframe solution that provides dynamic access to DB2 UDB data. It is available in the CICS or IMS environment. See also *Customer Information Control System*, *Database 2*, *Multiple Virtual Storage*.
- **Mainframe Connect ECDA Option for Oracle for z/OS Option –** a Sybase Open Server application that provides access management for non-Sybase databases, copy management (transfer), and remote systems management.
- **Mainframe Connect Server Option –** a Sybase product that provides capability for programmatic access to mainframe data. It allows workstation-based clients to execute customer-written mainframe transactions remotely. It is available for the CICS and the IMS and MVS environments
- **Multiple Virtual Storage –** an IBM operating system that runs on most System/370 and System/390 mainframes. It supports 24-bit addressing up to 16 megabytes.
- **network protocol –** a set of rules governing the way computers communicate on a network.
- **nonclustered index –** an index that stores key values and pointers to data. Compare with *clustered index*.
- **null –** having no explicitly assigned value. NULL is not equivalent to 0 or to blank.
- **ODBC –** see *Open Database Connectivity*.
- **ODS –** see *Open Data Services*.
- **Open Client –** a Sybase product that provides customer applications, third-party products, and other Sybase products with the interfaces required to communicate with Open Client and Open Server applications.
- **Open Data Services –** a product that provides a framework for creating server applications that respond to DB-Library clients.
- **Open Database Connectivity –** a Microsoft API that allows access to both relational and non-relational databases. See also *application program interface*.

- **Open Server –** a Sybase product that provides the tools and interfaces required to create a custom server. Clients can route requests to the ECDA Option for Oracle server through an Open Server configured to meet specific needs, such as the preprocessing of SQL statements.
- **parameter –** a variable that is given a constant value for a specified application and can denote the application. Compare with *property*.
- **Partner Certification Reports –** Sybase publications that certify third-party or Sybase products to work with other Sybase products.
- **Password Expiration Management –** an IBM password management program with CICS Version 3.3 through an optional program temporary fix, and as an integral part of CICS with version 4.1 and higher.
- **PEM –** see *Password Expiration Management*.
- **PL/1 –** see *Programming Language /1*.
- **primary database –** the database management system that the DirectConnect server is always connected to. It is implied in the **transfer** statement.
- **Programming Language/1 –** a programming language designed for use in a wide range of commercial and scientific computer applications.
- **property –** a setting for a server or service that defines the characteristics of the service, such as how events are logged. Compare with *parameter*.
- **protocol –** the rules for requests and responses used to manage a network, transfer data, and synchronize the states of network components.
- **query –** a request for data from a database, based upon specified conditions.
- **Registry –** the part of the Windows operating system that holds configuration information for a particular machine.
- **relational database –** a database in which data is viewed as being stored in tables consisting of columns (data items) and rows (units of information).
- **relational operators –** operators supported in search conditions.
- **relops –** see *relational operators*.
- **remote procedure call –** a call to execute a stored procedure on a remote server. For Mainframe Connect Server Option, an RPC is a direct request from a client to TRS. For Mainframe Connect Client Option, a Client-Library transaction that calls a procedure on a remote server acts like an RPC.
- **remote stored procedure –** a customer-written CICS program using an API that resides on the mainframe and communicates with Mainframe Connect DB2 UDB Option. See also *Customer Information Control System*, *stored procedure*. Compare with *Client Services Application*.
- **remote systems management –** a feature that allows a system administrator to manage multiple DirectConnect servers and multiple services from a client.
- **Replication Server –** a Sybase Adaptive Server application that maintains replicated data and processes data transactions received from a datasource.

- **request –** one or more database operations an application sends as a unit to the database. Depending upon the response, the application commits or rolls back the request. See also *commit*, *rollback*, *unit of work*.
- **resource table –** a main storage table that associates each resource identifier with an external logical unit (LU) or application program.
- **rollback –** an instruction to a database to back out of changes requested in a unit of work. Compare with *commit*.
- **router –** an attaching device that connects two LAN segments, which use similar or different architectures, at the Open System Interconnection (OSI) reference model network layer. Contrast with *gateway*.
- **RPC –** see *remote procedure call.*
- **RSP –** see *remote stored procedure*.
- **SAA –** see *System Application Architecture*.
- **secondary connection –** The connection specified in the **transfer** statement. It represents anything that can be accessed using Mainframe Connect Client Option, such as Adaptive Server or another access service.
- **secondary database –** in transfer processing, the supported database that is specified in the **transfer** statement. Compare with *primary database*.
- **server –** a functional unit that provides shared services to workstations over a network. See also *client/server*. Compare with *client*.
- **server process ID –** a positive integer that uniquely identifies a client connection to the server.
- **service –** a functionality available in Mainframe Connect DirectConnect for z/OS Option. It is the pairing of a service library and a set of specific configuration properties.
- **service library –** in Mainframe Connect DirectConnect for z/OS Option, a set of configuration properties that determine service functionality. See also *access service library*, *administrative service library*, *Transaction Router Service library*, *transfer service library*.
- **service name redirection –** a type of service name resolution that allows a system administrator to create an alternative mechanism to map connections with services. See also *service name resolution*. Compare with *direct resolution*.
- **service name redirection file –** the default name of the file used for the service name redirection feature. See *service name redirection*.
- **service name resolution –** the DirectConnect server mapping of an incoming service name to an actual service. See also *direct resolution*, *service name redirection*.
- **session –** a connection between two programs or processes. In APPC communications, sessions allow transaction programs to have conversations between the partner LUs. See also *advanced program-to-program communication*.
- **short transaction –** a mainframe transaction that ends the communication when it finishes returning results to the client. Compare with *long-running transaction*.
- **SNA –** see *Systems Network Architecture*.

- **SNRF –** see *service name redirection file*.
- **SPID –** see *server process ID*.
- **SQL –** see *structured query language*.
- **SQLDA –** see *SQL descriptor area*.
- **sqledit –** a utility for creating and editing `sql.ini` files and file entries.
- **sql.ini –** the interfaces file containing definitions for each ECDA Option for Oracle server to which a workstation can connect. The file must reside on every client machine that connects to Adaptive Server.
- **SQL descriptor area –** a set of variables used in the processing of SQL statements.
- **SQL stored procedure –** a single SQL statement that is statically bound to the database. See also *stored procedure*.
- **static SQL –** SQL statements that are embedded within a program and prepared during the program preparation process before the program runs. Compare with *dynamic SQL*.
- **stored procedure –** a collection of SQL statements and optional control-of-flow statements stored under a particular name. Adaptive Server stored procedures are called “system procedures.” See also *remote stored procedure*, *system procedures*.
- **structured query language –** an IBM industry-standard language for processing data in a relational database.
- **stub –** A program module that transfers remote procedure calls (RPCs) and responses between a client and a server.
- **SYRT –** the component of Mainframe Connect DB2 UDB for IMS that allows clients to submit SQL language requests to DB2 through IMS.
- **system administrator –** the person in charge of server system administration, including installing and maintaining DirectConnect servers and service libraries.
- **System Application Architecture –** an IBM proprietary plan for the logical structure, formats, protocols, and operational sequences for transmitting information units through networks and controlling network configuration and operation. See also *advanced program-to-program communication*.
- **system procedures –** a stored procedure that Adaptive Server supplies for use in system administration. System procedures serve as shortcuts for retrieving information from system tables, or a mechanism for accomplishing database administration. See also *stored procedure*.
- **Systems Network Architecture –** an IBM proprietary plan for the structure, formats, protocols, and operational sequences for transmitting information units through networks. See also *advanced program-to-program communication*.
- **table –** an array of data or a named data object that contains a specific number of unordered rows. Each item in a row can be unambiguously identified by means of one or more arguments.
- **Tabular Data Stream™ –** a Sybase application-level protocol that defines the form and content of relational database requests and replies.

- **target –** a system, program, or device that interprets, rejects, satisfies, or replies to requests received from a source.
- **target database –** the database to which the DirectConnect server transfers data or performs operations on specific data.
- **TCP/IP –** see *Transmission Control Protocol/Internet Protocol*.
- **TDS –** see *Tabular Data Stream*.
- **transaction –** a unit of processing initiated by a single request. A transaction consists of one or more application programs that, when executed, accomplish a particular action. In Mainframe Connect Server Option, a client request (RPC or language request) invokes a mainframe transaction. In Mainframe Connect Client Option, a mainframe transaction executes a stored procedure on a remote server.
- **transaction processing –** a sequence of operations on a database that is viewed by the user as a single, individual operation.
- **Transaction Router Service –** a Mainframe Connect DirectConnect for z/OS Option program used when the mainframe acts as a transaction server to route requests from remote clients to the Mainframe Connect Server Option and return results to the clients.
- **Transaction Router Service library –** a service library that facilitates access to remote transactions, allowing customers to execute transactions from virtually any mainframe datasource. See also *service library*.
- **Transact-SQL –** a Sybase-enhanced version of the SQL database language used to communicate with Adaptive Server.
- **transfer –** a Mainframe Connect DirectConnect for z/OS Option feature that allows users to move data or copies of data from one database to another.
- **transfer service library –** a service library that provides copy management functionality. See also *service library*.
- **Transmission Control Protocol/Internet Protocol –** a set of communication protocols that supports peer-to-peer connectivity functions for both local and wide area networks.
- **trigger –** A form of stored procedure that automatically executes when a user issues a change statement to a specified table.
- **TRS –** see *Transaction Router Service*.
- **TRS library –** see *Transaction Router Service library*.
- **unit of work –** one or more database operations grouped under a commit or rollback. A unit of work ends when the application commits or rolls back a series of requests, or when the application terminates. See also *commit*, *rollback*, *transaction*.
- **user ID –** user identification. The ID number by which a user is known in a specific database or system.
- **variable –** an entity that is assigned a value. Mainframe Connect ECDA Option for Oracle for z/OS Option has two kinds of variables: *local* and *global.*
- **view –** an alternate representation of data from one or more tables. A view can include all or some of the columns contained the table or tables on which it is defined.

- **Virtual Storage Access Method –** an IBM-licensed program that controls communication and the flow of data in an SNA network.
- **Virtual Telecommunications Access Method –** IBM mainframe software that allows communication on an SNA network between mainframes and allows the mainframe to have multiple sessions per connection.
- **VSAM –** see *Virtual Storage Access Method.*
- **VTAM –** see *Virtual Telecommunications Access Method*.
- **wildcard –** a special character that represents a range of characters in a search pattern.

Index