SYBASE®

Component Integration Services User's Guide

Adaptive Server® Enterprise

12.5.1

DOCUMENT ID: DC32702-01-1251-01

LAST REVISED: August 2003

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

Contents

About This Book

Audience

This book is written for Sybase® Adaptive Server® Enterprise System Administrators, database administrators, and users.

How to use this book

This guide will assist you in configuring and using Component Integration Services. The book includes the following chapters:

- Chapter 1, "Introduction," provides an overview of Component Integration Services.
- Chapter 2, "Understanding Component Integration Services," provides a framework for understanding how Component Integration works. This chapter includes both basic concepts and in-depth topics.
- Chapter 3, "SQL Reference," describes Transact-SQL commands that utilize Component Integration Services.
- Chapter A, "Tutorial," includes a tutorial designed to help new users get Component Integration Services up and running.
- Chapter B, "Troubleshooting," provides troubleshooting tips if you encounter a problem with Component Integration Services.

Related documents

The Sybase Adaptive Server Enterprise documentation set consists of the following:

- The release bulletin for your platform – contains last-minute information that was too late to be included in the books.

  A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Technical Library.
- The *Installation Guide* for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.
- *What's New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server version 12.5.1, the system changes added to support those features, and the changes that may affect your existing applications.

- *ASE Replicator User's Guide* – describes how to use the ASE Replicator feature of Adaptive Server to implement basic replication from a primary server to one or more remote Adaptive Servers.
- *Component Integration Services User's Guide* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- *Configuring Adaptive Server Enterprise* for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.
- *EJB Server User's Guide* – explains how to use EJB Server to deploy and execute Enterprise JavaBeans in Adaptive Server.
- *Error Messages and Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
- *Full-Text Search Specialty Data Store User's Guide* – describes how to use the Full-Text Search feature with Verity to search Adaptive Server Enterprise data.
- *Glossary* – defines technical terms used in the Adaptive Server documentation.
- *Historical Server User's Guide* – describes how to use Historical Server to obtain performance information for SQL Server® and Adaptive Server.
- *Java in Adaptive Server Enterprise* – describes how to install and use Java classes as data types, functions, and stored procedures in the Adaptive Server database.
- *Job Scheduler User's Guide* – provides instructions on how to install and configure, and create and schedule jobs on a local or remote Adaptive Server using the command line or a graphical user interface (GUI).
- *Monitor Client Library Programmer's Guide* – describes how to write Monitor Client Library applications that access Adaptive Server performance data.
- *Monitor Server User's Guide* – describes how to use Monitor Server to obtain performance statistics from SQL Server and Adaptive Server.
- *Performance and Tuning Guide* – is a series of four books that explains how to tune Adaptive Server for maximum performance:
  - *Basics* – the basics for understanding and investigating performance questions in Adaptive Server.

    - *Locking* – describes how the various locking schemas can be used for improving performance in Adaptive Server.
    - *Optimizer and Abstract Plans* – describes how the optimizer processes queries and how abstract plans can be used to change some of the optimizer plans.
    - *Monitoring and Analyzing* – explains how statistics are obtained and used for monitoring and optimizing performance.
- *Quick Reference Guide* – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, datatypes, and utilities in a pocket-sized book.
- *Reference Manual* – is a series of four books that contains the following detailed Transact-SQL® information:
    - *Building Blocks* – Transact-SQL datatypes, functions, global variables, expressions, identifiers and wildcards, and reserved words.
    - *Commands* – Transact-SQL commands.
    - *Procedures* – Transact-SQL system procedures, catalog stored procedures, system extended stored procedures, and `dbcc` stored procedures.
    - *Tables* – Transact-SQL system tables and `dbcc` tables.
- *System Administration Guide* – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources, security, user and system databases, and specifying character conversion, international language, and sort order settings.
- *System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Available only in print version.
- *Transact-SQL User's Guide* – documents Transact-SQL, Sybase's enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the `pubs2` and `pubs3` sample databases.
- *Using Adaptive Server Distributed Transaction Management Features* – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.

- *Using Sybase Failover in a High Availability System* – provides instructions for using Sybase's Failover to configure an Adaptive Server as a companion server in a high availability system.
- *Utility Guide* – documents the Adaptive Server utility programs, such as isql and bcp, which are executed at the operating system level.
- *Web Services User's Guide* – explains how to configure, use, and troubleshoot Web Services for Adaptive Server.
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* – provides instructions for using the Sybase DTM XA interface with X/Open XA transaction managers.
- *XML Services in Adaptive Server Enterprise* – describes the Sybase native XML processor and the Sybase Java-based XML support, introduces XML in the database, and documents the query and mapping functions that comprise XML Services.

Other sources of information

Use the Sybase Getting Started CD, the Sybase Technical Library CD and the Technical Library Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).
- The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.
- The Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Updates, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Technical Library Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Select Products from the navigation bar on the left.

3 Select a product name from the product list and click Go.

4 Select the Certification Report filter, specify a time frame, and click Go.

5 Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software updates

❖ **Finding the latest information on EBFs and software updates**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Updates. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).

3 Select a product.

4 Specify a time frame and click Go.

5 Click the Info icon to display the EBF/Update report, or click the product description to download the software.

Conventions

What you type to the computer screen is shown as:

```
Enter text in an entry field
```

Computer output is shown as:

```
CIS returns results.
```

Command arguments you replace with a non-generic value are shown in italics:

machine_name

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1

Introduction

Component Integration Services extends Adaptive Server capabilities and provides enhanced interoperability.

It also provides location transparency and functional compensation.

Location transparency means that Component Integration Services allows Adaptive Server to present a uniform view of enterprise data to client applications. Enterprise-wide data from heterogeneous sources can be accessed as if it were local.

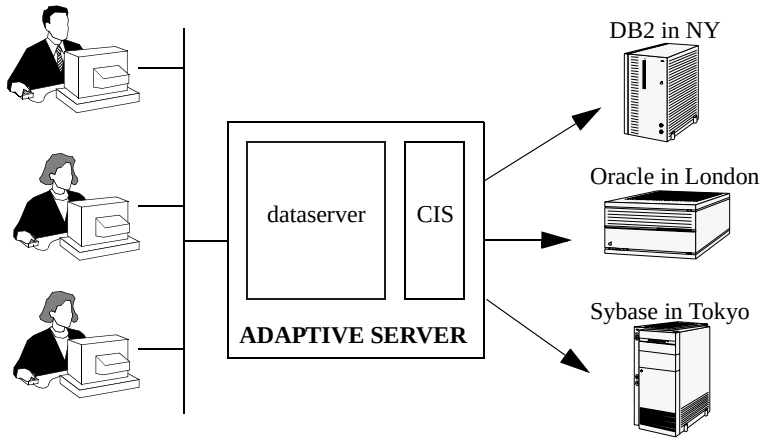
Functional compensation allows Component Integration Services to emulate all features of Transact-SQL, and interact with a data source only when actual data is needed. With this capability, the full range and power of Transact-SQL can be applied to any data source, whether or not the data source provides support for a particular feature of Transact-SQL. Examples of this capability are built-in functions and Java functions. Component Integration Services allows statements to use these functions even though the data on which these functions may operate is derived from external sources that cannot support the functions.

Component Integration Services, together with Adaptive Server Anywhere, Adaptive Server IQ and various DirectConnect interfaces, extends the reach of Adaptive Server by enabling transparent access to database management systems anywhere in the enterprise. This transparent, extended reach of Adaptive Server Enterprise makes it easy for Enterprise Portal components to:

- Access data from anywhere, and present it as dynamic content to Web pages
- Execute transactions that span heterogeneous boundaries
- View an entire enterprise through a single view provided by the global metadata stored in the Adaptive Server/Component Integration Services system catalogs

Component Integration Services allows users to access both Sybase and non-Sybase databases on different servers. These external data sources include host data files, tables, views, and RPCs (remote procedure calls) in database systems such as Adaptive Server, Oracle, and DB2.

Figure 1-1: Component Integration Services connects to multiple vendor databases



Using Component Integration Services, you can:

- Access tables in remote servers as if the tables were local.
- Perform joins between tables in multiple remote, heterogeneous servers. For example, it is possible to join tables between an Oracle database management system (DBMS) and an Adaptive Server, and between tables in multiple Adaptive Servers.
- Transfer the contents of one table into a new table on any supported remote server by means of a select into statement.
- Maintain referential integrity across heterogeneous data sources.
- Access native remote server capabilities using the Component Integration Services passthrough mode.

Component Integration Services can be used by anyone who needs to access multiple data sources or legacy data. It can also be used by anyone who needs to migrate data from one server to another.

A single server is often used to access data on multiple external servers. Component Integration Services manages the data regardless of the location of the external servers. Data management is transparent to the client application.

Component Integration Services, in combination with EnterpriseConnect™ and MainframeConnect™, provides transparent access to a wide variety of data sources, including:

- Oracle

- Informix
- IBM databases including:
  - DB2 for MVS
  - DB2/400
  - DB2/2
  - DB2 for VM (SQL/DS)
- Microsoft SQL Server
- Adaptive Server Enterprise
- Adaptive Server Anywhere
- Adaptive Server IQ
- Mainframe data, including:
  - ADABAS
  - IDMS
  - IMS
  - VSAM

To start Component Integration Services:

- Install DirectConnect server(s) or gateways for the external data sources you choose to access (for example, Oracle, DB2, Informix, Microsoft SQL Server).
- Configure the server to access remote objects as described in Chapter 2, “Understanding Component Integration Services.”

CHAPTER 2

Understanding Component Integration Services

This chapter explains how to use Component Integration Services. It is intended to help you understand how Adaptive Server works with the Component Integration Services option configured.

Basic concepts

The ability to access remote (or external) tables as if they were local is a hallmark of Component Integration Services. Component Integration Services presents tables to a client application as if all the data in the tables were stored locally. Remote tables are mapped to local proxy tables which hold metadata. Internally, when a query involving remote tables is executed, the storage location is determined, and the remote location is accessed so that data can be retrieved.

The access method used to retrieve remote data is determined by two attributes of the external object:

- The server class associated with the remote object
- The object type

To achieve location transparency, tables must first be mapped to their corresponding external locations.

Access methods

Access methods form the interface between the server and an external object. For each server class, there are separate access methods that handle all interaction between Adaptive Server and remote servers of the same class and object type.

Server classes

A server class must be assigned to each server when it is added using sp_addserver. Server classes determine the access method used to interact with the remote server. The server classes are:

- *ASEnterprise* – used if the server is Adaptive Server. This is the default server class.
- *ASAnywhere* – used if the server is Adaptive Server Anywhere version 6.0 or later. This server class should be used for Adaptive Server IQ versions earlier than Adaptive Server IQ 12.5.
- *ASIQ* – used if the server is Adaptive Server IQ version 12.5.
- *local* – the local server. There can be only one.
- *direct_connect* – indicates that the server is an Open Server™ application that conforms to the interface requirements of a DirectConnect™ server. For access to Microsoft SQL Server, you must use a DirectConnect server.
- *db2* – indicates that the server is a gateway to DB2 or DB2-compatible databases. This class is provided only for backward compatibility. The preferred class is *direct_connect*.
- *sds* – indicates that the server conforms to the interface requirements of a Specialty Data Store.

Object types

The server presents a number of object types to client applications as if they were local tables. Supported object types are:

- table – the object in a remote server of any class is a relational table. This is the default type.
- view – the object in a remote server of any class is a view. Component Integration Services treats views as if they were local tables without any indexes.
- remote procedure – the object in a remote server of any class is a remote procedure. Component Integration Services treats the result set from the remote procedure as a read-only table.
- file – the object is an individual file within a file system.
- directory – the object is a file system directory.

Interface to remote servers

The interface between the server and remote servers is handled by the Open Client software, Client-Library™. The Client-Library features that are used to implement the interface are dependent upon the class of server with which Component Integration Services is interacting.

For example, if the server class is *direct_connect*, a number of features such as cursor and dynamic requests are used. These features are not used by a server of class *db2*.

Before the server can interact with a remote server, you must configure the following:

- Remote server addition to directory services
- Remote server definition
- Remote server login information
- Remote object definition

Directory services

Before accessing remote tables with Component Integration Services, you must either have access to LDAP directory services, or an *interfaces* file (*sql.ini* file on Windows NT). For information on setting up directory services, see the configuration documentation for your platform. See Appendix A, “Tutorial,” which serves as a basic tutorial for Component Integration Services users.

Remote server definition

Remote servers are defined by means of the stored procedure sp_addserver. This procedure is documented in the *Reference Manual*.

Logging in to remote servers

Once you have configured the remote server, you must provide login information. By default, Component Integraiton Services uses the names and passwords of Adaptive Server clients whenever it connects to a remote server on behalf of those clients. However, this default can be overridden using sp_addexternlogin, which allows a System Administrator to define the name and password for each user who connects to a remote server.

Using connect to *server_name*, you can verify that the server configuration is correct. This command establishes a passthrough mode connection to the remote server. Passthrough mode allows clients to communicate with remote servers in native syntax. This passthrough mode remains in effect until you issue a disconnect command.

Defining remote objects

Once you have configured a remote server, you cannot access objects in that remote server as tables until a mapping between them and a local object (proxy table) has been established.

You can create new tables on remote servers, and you can define the schema for an existing object in a remote server. The procedures for both are similar.

Proxy tables

Proxy tables are the key to location transparency. A proxy table is a local table containing metadata which points to a remote or external table. The remote table is mapped to the proxy table to make it appear as if it were a local table.

The complete description of how to do this is in Chapter 3, “SQL Reference.”

Using the *create table* command

The create table command creates a proxy table and a remote table at the same time with the following syntax:

create table table_name (column_list) [[external {table | file}] at “pathname”]]

The remote location is specified with the at pathname clause. create table allows external object type table and file. The datatype of each column is passed to the remote server without conversion except with server class *db2*.

Using the *create existing table* command

The create existing table command allows the definition of existing tables (proxy tables). The syntax for this option is similar to the create table command:

```
create existing table table_name (column_list)
[[external {table | procedure | file}] at pathname]
```

The action taken by the server when it receives this command is quite different from the action it takes when it receives the create table command, however. A new table is not created at the remote location; instead, the table mapping is checked, and the existence of the underlying object is verified. If the object does not exist (either host data file or remote server object), the command is rejected with an error message.

If the object does exist, its attributes are obtained and used to update system tables sysobjects, syscolumns, and sysindexes.

- The nature of the existing object is determined.
- For remote server objects (other than RPCs), column attributes found for the table or view are compared with those defined in the *column_list*. Column names must match (case is ignored), column types and lengths must match, or at least be convertible, and the NULL attributes of the columns must match.
- Index information from the host data file or remote server table is extracted and used to create rows for the system table sysindexes. This defines indexes and keys in server terms and enables the query optimizer to consider any indexes that may exist on this table.
- The on *segment_name* clause is processed locally and is not passed to a remote server.

After successfully defining an existing table, issue an update statistics command for the table.This allows the query optimizer to make intelligent choices regarding index selection and join order.

Datatype conversions

When you use the create table or create existing table commands, you must specify all datatypes, using recognized Adaptive Server datatypes. If the remote server tables reside on a class of server that is heterogeneous, the datatypes of the remote table are converted into the specified Adaptive Server types automatically when the data is retrieved. If the conversion cannot be made, the create table or create existing table commands do not allow the table to be created or defined.

Example of remote table definition

The following example defines the remote Adaptive Server table authors, starting with the server definition:

1 Define a server named SYBASE. Its server class is *ASEnterprise*, and its name in the interfaces file is SYBASE:

```
exec sp_addserver SYBASE, ASEnterprise, SYBASE
```

2 Define a remote login alias. This step is optional. User “sa” is known to remote server SYBASE as user “sa,” password “timothy”:

```
exec sp_addexternlogin SYBASE, sa, sa, timothy
```

3 Define the remote authors table:

```
create existing table authors
 (
au_id      id              not null,
au_lname   varchar(40)     not null,
au_fname   varchar(20)     not null,
phone      char(12)        not null,
address    varchar(40)     null,
city       varchar(20)     null,
state      char(2)         null,
country    varchar(12)     null,
postalcode char(10)        null
 )
EXTERNAL TABLE at "SYBASE.pubs2.dbo.authors"
```

4 Update statistics in tables to ensure reasonable choices by the query optimizer:

```
update statistics authors
```

5 Execute a query to test the configuration:

```
select * from authors where au_lname = 'Carson'
```

Using the create proxy_table command

create proxy_table is a variant of create existing table. Use create proxy_table to create a proxy table, but do not specify a column list. Component Integration Services derives the column list from the metadata it obtains from the remote table.

create proxy_table creates proxy tables that automatically inherit all the columns, column names, and datatypes of the external table using the following syntax:

```
create proxy_table table_name
[ external type ] at pathname
```

create proxy table allows external object type table, file, and directory.The location information provided by the at keyword specifies the path name to the remote object.

External type can be one of the following:

- external table (the default) specifies that the object is a remote table or view.
- external directory specifies that the object is a directory with a path similar to the following: "*/tmp/directory_name* [;R]"."R" indicates "recursive."
- external file specifies that the object is a file with a path similar to */tmp/filename*.

Remote procedures as proxy tables

You can add an optional clause to the create existing table statement to indicate the remote object is actually a stored (or other) procedure instead of a table. Without this clause, the remote object is assumed to be a table or view:

```
create existing table t1
    (
                column_1    int,
                column_2    int
    )
    EXTERNAL PROCEDURE AT "SERVER_A.mydb.dbo.p1"
```

If the remote object is type procedure, several processing differences occur:

- No indexes are created for objects of this type.
- You must provide a column list that matches the description of the remote procedure's result set. No verification of the list's accuracy is provided.

- You can use column names beginning with underscore ('_') to specify columns that are not part of the remote procedure's result set. These columns are referred to as parameter columns. For example:

```
create existing table rpc1

    a       int,
    b       int,
    c       int,
    _p1     int null,
    _p2     int null
)
external procedure
at "SYBASE.sybsystemprocs.dbo.myproc"

select a, b, c from t1
where _p1 = 10 and _p2 = 20
```

- In this example, the parameter columns *_p1* and *_p2* are not expected in the result set, but can be referenced in the query. Component Integration Services passes the search arguments to the remote procedure via parameters, using the names *@p1* and *@p2*.
- If a parameter column is included in the select list, its value is equivalent to the values specified for it in the where clause, if it was passed to the remote procedure as a parameter. If the parameter column did not appear in the where clause, or was not able to be passed to the remote procedure as a parameter, but was included in the select list, its value would be NULL.
- A parameter column can be passed to the remote procedure as a parameter if it is what the Adaptive Server query processor considers to be a searchable argumen. It is generally a seachable arguement if it is not included in any "or" predicates. For example, the following query would prevents the parameter columns from being used as parameters:

```
select a, b, c from t1
where _p1 = 10 OR _p2 = 20
```

- Rules exist for the definition of parameter columns in the create existing table statement:
  - Parameter columns must allow NULL.
  - Parameter columns cannot precede normal, result columns (they must appear at the end of the column list).

Allowing the definition of remote procedures as local tables allows Component Integration Services to treat the result set of a remote procedure as a "virtual table," which can be sorted, joined with other tables, or inserted into another table via insert/select syntax. However, virtual tables are considered read-only:

- You cannot issue a delete, update, or insert command against a table of type *procedure*;
- You cannot issue a create index, truncate table, or alter table command against virtual tables.

If an object of the type procedure has been defined within the server, a query is not issued to the remote server on which the object resides. Instead, Component Integration Services issues an RPC and treats the results from the RPC as a read-only table.

Examples

```
create existing table rtable
 ( col1   int,
   col2   datetime,
   col3   varchar(30)
 )
external procedure at "RMTSERVER...myproc"

select * from rtable
```

When this query is issued, Component Integration Services sends the RPC named *myproc* to server RMTSERVER. Row results are treated like the results from any other table; they can be sorted, joined with other tables, grouped, inserted into another table, and so forth.

RPC parameters should represent arguments that restrict the result set. If the RPC is issued without parameters, the entire result set of the object is returned. If the RPC is issued with parameters, each parameter further limits the result set. For example:

```
select * from rtable where col1 = 10
```

results in a single parameter, named @col1, that is sent along with the RPC. Its value is 10.

Component Integration Services attempts to pass as many of the search arguments as possible to the remote server, but depending on the SQL statement being executed, Component Integration Services might perform the result set calculation itself. Each parameter represents a search for an exact match, for example, the = operator.

The following rules define the parameters sent to the RPC. If an RPC is used as a Component Integration Services object, you should keep these rules in mind during development.

- Component Integration Services sends = operators in the where clause as parameters. For example, this query results in Component Integration Services sending two parameters:

  ```
  select * from rpc1 where a = 3 and b = 2
  ```

  Parameter a has a value of 3 and parameter b has a value of 2. The RPC is expected to return only result rows in which column a has a value of 3 and column b has a value of 2.

- Component Integration Services does not send any parameters for a where clause, or portion of a where clause, if there is not an exact search condition. For example:

  ```
  select * from rpc1 where a = 3 or b = 2
  ```

  Component Integration Services does not send parameters for a or b because of the or clause.

  Another example:

  ```
  select * from rpc1 where a = 2 and b < 3
  ```

  Component Integration Services sends parameters for *a* and *b*, and filters rows containing *b* with values smaller than 3.

Server limits

Adaptive Server configuration allows page sizes of 2K, 4K, 8K, or 16K bytes. Also, the limit of 255 bytes for char/binary columns has been removed. Adaptive Server supports extended sizes of char, varchar, binary, and varbinary datatypes. The new limit depends on the page size of the server. For various page sizes, the new limits are as follows:

Table 2-1: New limits

Page size	Maximum column size
2048	2048
4096	4096
8192	8192
16384	16384

These sizes are approximate. The basic rule specifies that the limit is the maximum size that still allows a single row to fit on a page. These limits also vary depending on the locking scheme specified when the table is created. It is assumed that the bulk of proxy tables are created with the default locking scheme, which is allpages locking.

- Limits on length of Transact-SQL variables and parameters – the size of char, varchar, binary, and varbinary variables are extended to equal the maximum size of columns of the same datatype for a given server. This allows variables to be passed to stored procedures (or RPCs) whose length exceeds the current limit of 255 bytes.
- Limits on number of columns per table – as many as 1024 columns per table are allowed, as long as the columns can still fit on a page. There is a limit of 254 variable-length columns (null columns are also considered variable length).
- Limits on the width of an index – the total width of an index within Adaptive Server version 12.5 can be larger than in earlier version, depending on server page size. In Table 2-2, maximum index width is shown according to page size:

Table 2-2: Maximum index width

Page size	Index width
2048	600
4096	1250
8192	2600
16384	5300

- Limits on the number of columns per index – the current limit of 31 columns per index is unchanged in version 12.5.

create new proxy table

create table allows columns of datatype char, varchar, binary, and varbinary to be specified with extended lengths, as described above. These datatypes and lengths are forwarded to the remote server on which the table is to be created.

create existing proxy table

The create existing table command also allows columns to be specified with a length of greater than 255 bytes. This allows Component Integration Services to treat columns in remote databases as char, varchar, binary, or varbinary that previously had to be treated as text or image columns.

There is still an opportunity for column size mismatch errors. For example, in the case where the remote database contains a table with a column length of 5000 bytes, and the Adaptive Server processing create existing table supports columns only up to 1900 bytes, a size mismatch error occurs. In this case, it is necessary to respecify the column as a text or image column.

When the proxy table column size exceeds that of the corresponding column in the remote table, a size mismatch error is detected and the command is aborted.

create proxy_table

create proxy_table imports metadata from a remote server and converts column information into an internal create existing table command, with a column list derived from the imported metadata. When obtaining the column metadata, conversion from the remote DBMS type to internal Adaptive Server Enterprise types is required.

If the size of a remote char, varchar, binary, or varbinary column exceeds the maximum allowed by the local server, its length is truncated to the maximum size possible, which depends on page size. If the size exceeds 16K bytes, the type is converted from char or varchar to text, or from binary or varbinary to image.

alter table

If alter table operates on a proxy table, it is first processed locally, then forwarded to the remote server for execution. If the remote execution fails, the local changes are backed out and the command is aborted.

The remote server must process the command appropriately, or raise an error. If an error is produced, the Component Integration Services side of the command is aborted and rolled back.

select, insert, delete, update

Component Integration Services handles large column values when proxy tables are involved in data manipulation language (DML) operations. Component Integration Services handles DML using one of several strategies:

- Tabular data stream (TDS)™ language commands – if the entire SQL statement can be forwarded to a remote server, then Component Integration Services does so using TDS Language commands generated by CT-Library ct_command (CS_LANG_CMD).

  The text of the language buffer may contain data for long char or binary values that exceeds 255 bytes, and remote servers must handle parsing of these command buffers.

- TDS dynamic commands – if Component Integration Services cannot forward the entire SQL statement to a remote server (for example, Component Integration Services is forced to provide functional compensation for the statement), then an insert, update, or delete may be handled by using TDS dynamic commands, with parameters as needed, using the CT-Library function ct_dynamic (CS_PREPARE_CMD, CS_EXECUTE_CMD, CS_DEALLOC_CMD).

  The parameters for the dynamic command may be CS_LONGCHAR_TYPE or CS_LONGBINARY_TYPE.

- TDS cursor commands – CT-Library cursor operations can be used to handle proxy table operations for select, update, and delete if functional compensation has to be performed. For example, if you are updating a proxy table and there are multiple tables in the from clause, Component Integration Services may have to fetch rows from multiple data sources, and for each qualifying row, apply the update to the target table. In this case, Component Integration Services uses ct_cursor ({CS_DECLARE_CMD, CS_OPEN_CMD, CS_CURSOR_UPDATE_CMD, CS_CLOSE_CMD, CS_DEALLOC_CMD}).

  After a cursor is prepared, parameters are specified. These parameters may include those of type CS_LONGCHAR or CS_LONGBINARY.

- Bulk insert commands – when performing a select/into operation, if the target server supports the bulk interface (only true of remote Adaptive Servers and DirectConnect for Oracle), then the remote server must be prepared to handle char and binary values greater than 255 (via CS_LONGCHAR, CS_LONGBINARY values).

Columns from remote servers may be returned to Component Integration Services as type CS_LONGCHAR_TYPE or CS_LONGBINARY_TYPE.

RPC handling

RPCs sent to remote servers can contain parameters of types CS_LONGCHAR and CS_LONGBINARY. The Component Integration Services command cis_rpc_handling supports these types.

Sending long parameters toAdaptive Servers older than version 12.5 is not allowed, as earlier versions of Adaptive Server do not support CS_LONGCHAR or CS_LONGBINARY data. Component Integration Services examines TDS capabilities for the remote server prior to sending the RPC, and if the remote server cannot accept these datatypes, an error results.

Cascading proxy tables

Adaptive Server allows cascading proxy table configurations between any number of instances of Component Integration Services.

There are conditions where this can cause problems, such as circular references, or transactions in which the second proxy table references a local table on the same server as the first proxy table. In this case, application deadlocks can result that are not detected by Component Integration Services. You must configure your systems to avoid these potential pitfalls.

Proxy databases

There are two types of proxy databases: user and system.

User proxy databases

When a user proxy database is created, metadata for the proxy tables is imported automatically from the remote location that contains the actual tables. This metadata is then used to create proxy tables within the proxy database.

To create a proxy database, use:

```
create database <dbname>
    [create database options]
    [with default_location = 'pathname']
    [for proxy_update]]
```

The use of the clause with default_location allows you to specify the storage location of any new tables, and the location from which metadata may be imported for automatic proxy table creation if the for proxy_update clause is also specified. for proxy_update establishes the database as a proxy database; with default_location defines the location from which proxy tables are imported. Without for proxy_update, the behavior of with default_location is the same as that provided by sp_defaultloc — a default storage location is established for new and existing table creation, but automatic import of proxy table definitions does not take place during the processing of the create database command.

The value of path name is a string identifier in the following format: *servername.dbname.owner*.

- *servername* – required field; represents the name of the server that owns the objects to be referenced by proxy tables. Must exist in *master.dbo.sysservers.srvname*.
- *dbname* – optional. The name of the database within *servername* which contains objects to be referenced by proxy tables
- *owner* – optional. The name of the owner of objects to be referenced by proxy tables. This may be restrictive, so that if more than one user owns objects in *dbname*, specifying the owner selects only those objects owned by that user. Do not create proxy tables for objects owned by other users.

If for proxy_update is specified with no default_location, an error is reported.

When a proxy database is created (using the for proxy_update option), Component Integration Services functions are called upon to:

- Provide an estimate of the database size required to contain all proxy tables representing the actual tables/views found in the primary server's database. This estimate is provided in terms of the number of database pages needed to contain all proxy tables and indexes. This size is used if no size is specified, and no database devices are specified. This formula is described in create database on page 109.

  Note If the database is created with specific size specifications [on device_name = nn], or if a device name is specified with no size [on device_name], then the size requirements for the proxy database are not estimated; it is assumed in this case that the user or data base administrator wants to override the default size calculated for the proxy database.

  If you are importing metadata from another Adaptive Server, remote database users are imported before proxy tables are created. Each imported database user must have a corresponding system user name in syslogins.

- Create all proxy tables representing the actual tables/views found in the companion server's database. Proxy tables are not created for system tables.

  Note Before you create the proxy tables, turn on the quoted identifier state, and create each table with quotes surrounding the table name and column name. This allows the creation of tables containing names that may be Transact-SQL reserved words. When all proxy tables are created, the quoted identifier state is restored to its original setting.

- Grant all permissions on proxy tables to "public."
- Add the "guest" user to the proxy database.
- Import database users from remote site (if Adaptive Server).
- Grant create table permission to "public."
- Set the database status to indicate that this database is a user proxy database. This is done by setting a status field in *master.dbo.sysdatabases.status*3 (0x0001, DBT3_USER_PROXYDB).

After the database has been created, it contains a proxy table for each table or view found in the default location. The behavior for a user proxy database is identical to prior database behavior. Users can create additional objects, such as procedure, views, rules, defaults, and so on, and both DDL and DML statements that operate on proxy tables behave as documented in this book.

The only exception to this is the alter database command. The syntax and capabilities of this command are described in the next section.

User proxy database schema synchronization

At times, it may be necessary for a DBA to force resynchronization of the proxy tables contained within the proxy database. This can be done using the alter database command:

```
alter database <dbname>
        [alter database options]
        [for proxy_update]
```

If the for proxy_update clause is entered with no other options, the size of the database is not extended; instead, the proxy tables, if any, are dropped from the proxy database and re-created from the metadata obtained from the *pathname* specified during create database ... with default_location = 'pathname'.

If create database is used with other options to extend the size of the database, the proxy table synchronization is performed after the size extensions are made.

The purpose of this alter database extension is to provide a DBA with an easy-to-use, single-step operation with which to obtain an accurate and up-to-date proxy representation of all tables at a single remote site.

This resynchronization is supported for all external data sources, and not just the primary server in a HA-cluster environment. Also, a database need not have been created with the for proxy_update clause. If a default storage location has been specified, either through the create database command or using sp_defaultloc, the metadata within the database can be synchronized with the metadata at the remote storage location.

Certain behavior is implied by the use of create/alter database to specify a proxy database:

- Modification to the default location specified with the create database command is not allowed using alter database.
- Local tables cannot be created in the proxy database. create table commands result in the creation of proxy tables, and the actual table is created at the default location.
- The default location of the table may be specified in the create table command, using the at 'pathname' syntax. If the path name differs from the default location, then the alter database command will not synchronize the metadata for this table.

- To change the default location, drop the database, then re-create it with a new path name specified in the default_location = 'pathname' clause. If the location is changed using sp_defaultloc, then the new location is used to provide metadata synchronization, and proxy tables that were created with the prior location not be synchronized, and may be dropped and replaced if the name conflicts with that of tables at the new location.

System proxy databases

System proxy databases behave like user proxy databases, with some notable enhancements and exceptions. System proxy databases are only used in an HA configuration.

System proxy databases allow customer-written applications to run on either node in a high-availability cluster. This does not imply "single-system image" capability; rather, it suggests an environment in which most user-written applications can execute on either node in the cluster. This means that both databases and user-created objects should be visible to both nodes.

A system proxy database has the same name as the database in the primary node it references, and contains handling for the user-defined objects that are necessary to support the application. Proxy tables are created for each user table and view found in the primary database, and stored procedures are converted to RPCs and forwarded to the node referenced by the proxy database.

System proxy database creation

A system proxy database is created automatically under the following circumstances:

- The HA cluster is being configured through the use of the stored procedure sp_companion ServerName, 'configure', with_proxydb.

  In this case, a system proxy database is created for each user database found in server indicated by ServerName.

- A create database command is issued in a server whose HA state is one of MODE_APNC, MODE_SNC, or MODE_ASNC.

When the creation of the system proxy database is complete, Component Integration Services functions are called upon to:

- grant create table to public – this allows table creation on the primary server to result in proxy table creation in the system proxy database.

Schema synchronization when current database has a system proxy database

In an HA cluster, some of the changes to a primary server's database must be forwarded to the companion server to keep both servers synchronized.

Several DDL commands, when executed within a database that has a system proxy database, cause notification of the companion server and result in automatic synchronization of the resulting changes:

- create table and drop table – local operation executes, resulting in the local table being created or dropped. The command is then forwarded to the companion server, for execution in the system proxy database, so that a proxy table can be created or dropped
- create index and drop index – local operation executes, resulting in an index being created or dropped. The server owning the system proxy database is then notified, and the proxy table is dropped and re-created, allowing the change to the index to be represented within the proxy table.
- create view and drop view – the local operation succeeds, resulting in the local view being created or dropped. The server owning the system proxy database is then notified, and a proxy table is either created or dropped.

If these commands are executed within the system proxy database, similar behavior occurs:

- create table and drop table – local proxy table is created or dropped. The command is then forwarded to the primary server, so that a local table referenced by the proxy table can be created or dropped.
- create index and drop index – local operation on the proxy table executes, resulting in an index being created or dropped. The server owning the primary database is then notified, and an index is either created or dropped on the local table referenced by the proxy table
- create view and drop view – not allowed within a system proxy database.

Stored procedure execution within a system proxy database

If a system stored procedure request is encountered when the current database is a system proxy database, Component Integration Services attempts to locate the stored procedure first in the local sybsystemprocs database, and execute it. If it is not found in sybsystemprocs, Component Integration Services searches the master database. If the procedure is not a system stored procedure, or if it is but cannot be found locally, the stored procedure request is converted to an RPC and transmitted to the server referenced by the system proxy databases default location.

Additional behavior of the system proxy database

Certain commands, when executed within a system proxy database, are rejected with an error:

- create procedure and drop procedure
- create view and drop view
- create trigger and drop trigger
- create rule and drop rule
- create default and drop default

The error generated in these cases is: `Msg 12818, Severity 16: Cannot create an object of this type in system-created proxy database`.

File system access

Note Directories and files mapped to proxy tables now have a file path limit of 255 bytes.

Adaptive Server provides access to the file system through the SQL language. With file system access, you can create proxy tables that are mapped to file system directories, or to individual files.

A new class of proxy tables is allowed in Adaptive Server that enables SQL access to file system directories and their underlying files. To create proxy tables mapped to directories or files, you must have System Administrator or System Security Officer privileges.

File system access features must be purchased separately.

Security considerations

Only Adaptive Server Enterprise users with System Administrator (sa) or System Security Officer (sso) roles are allowed to create proxy tables that are mapped to files or directories. This requirement addresses the concerns over the security aspects of accessing file system data from within the Adaptive Server Enterprise server process (which may have root permission as it runs).

Directory access

Proxy tables can be created to reference file system directories. The supported syntax is:

```
create proxy_table <table_name>
external directory at "directory pathname[;R]"
```

The directory path name must reference a file system directory visible to and searchable by the Adaptive Server Enterprise process. A proxy table that maps column names to attributes of files that exist within the directory is created. If the ';R' (indicating "recursion") extension is added to the end of the path name, Component Integration Services includes entries in all subordinate directories. Table 2-3 contains a description of the proxy table columns that are created when this command successfully completes:

Table 2-3: Proxy table columns

Column name	Datatype	Description
id	numeric(24) – on 32-bit machines numeric(36) – on 64-bit machines	Identity value consisting of values from st_dev and st_ino. These two values are converted first to a single string (format: "%d%014ld"), and the string is then converted to a numeric value.

Column name	Datatype	Description
filename	varchar(n)	The name of the file within the directory specified in at 'pathname', or within directories subordinate to pathname. The total length (n) of filename is limited to 255 bytes.
size	int	For regular files – specifies the number of bytes in the file. For directories – block special or character special, this is not defined.
filetype	varchar(4)	The file type – legal values are: FIFO, for pipe files; DIR for directories; CHRS for character special files; BLKS for block special files; REG for ordinary files; UNKN for all other file types. Links are automatically expanded, and do not appear as a separate file type.
access	char(10)	Access permissions, presented in a more or less 'standard' UNIX format: "drwxrwxrwx"
uid	varchar(n)	The name of the file owner. The value of n is specified by the system definition L_cuserid, which is 9 on all UNIX systems except HP/Compaq Tru64, where it is 64. This value is 0 on Windows systems.
gid	varchar(n)	The name of the owning group. The value of n is specified by the system definition L_cuserid, which is 9 on all UNIX systems except HP/Compaq Tru64, where it is 64. This value is 0 on Windows systems.
atime	datetime	Date/time file data was last accessed.
mtime	datetime	Date/time when file was last modified.
ctime	datetime	Date/time when file status was last changed.
content	image	The actual physical content of the file (for regular files only). NULL if the file is not a regular file.

A proxy table that maps to a file system directory can support the following SQL commands:

- select – file attributes and content can be obtained from the proxy table using the select command. Built-in functions that are designed to handle text values are fully supported for the content column, (for example, textptr, textvalid, patindex, pattern).

- insert – new files or directories can be created using the insert command. The only columns that have meaning are filename, filetype, and content. The rest of the columns should be left out of the insert statement, and are ignored if they are located. The content column is ignored if file type is *DIR*, which indicates that a new directory is to be created.

  To create a new directory, enter:

  ```
  insert D1 (filename, filetype) values ("newdir",
  "DIR")
  ```

  To create a new file, enter:

  ```
  insert D1 (filename, filetype) values
  ("newdir/newfile", "This is an example.")
  ```

- delete – files or directories may be removed by the use of the delete command. A directory can be removed only if it is empty. For example:

  ```
  /* delete the files only */
  delete D1 where filename = 'newdir/newfile'
  /* deletes the directory (if empty) */
  delete D1 where filetype = 'DIR' and filename =
  'newdir'
  ```

- update – only the name of a file may be changed using the update command.
- readtext – the contents of a file may be retrieved using the readtext command.
- writetext – the contents of a file may be modified using the writetext command.

No other SQL commands operate on proxy tables.

Regular file content is available only if the Adaptive Server process has sufficient privileges to access and read the file, and if the file type indicates an "ordinary" file. In all other cases, the content column is NULL. For example:

```
select filename, size, content
    from directory_table
    where filename like '%.html'
```

returns the name, size and content of regular files with a suffix of ".html," if the Adaptive Server process has access privileges to the file. Otherwise, the content column will be NULL.

create proxy_table fails if the path name referenced by directory path name is not a directory, or is not searchable by the Adaptive Server Enterprise process.

If trace flag 11206 is turned on, messages are written to the error log that contain information about the contents of the directories and the query processing steps needed to obtain that information.

Recursion through subordinate directories

If the path name specified in the create proxy_table statement contains the *;R* extension, Component Integration Services traverses all directories subordinate to the path name, and returns information for the contents of each subordinate directory. When this is done, the file name returned by a query contains the complete name of the file relative to the path name. In other words, all subordinate directory names appear in the file name. For example, if path name specifies *"/work;R"*:

```
create proxy_table d1 external directory at "/work;R"
select filename, filetype from d1
```

values for files in subordinate directories are returned as outlined in Table 2-4:

Table 2-4: Values for files

File name	File type
dir1	DIR
dir1/file1.c	REG
dir1/file2.c	REG
dir2	DIR
dir2/file1.c	REG

File access

Another class of proxy tables allowed in Adaptive Server enables SQL access to individual files within a file system. The supported syntax is:

```
create proxy_table <table_name>
    external file at " pathname" [column delimiter "<string>"]
```

When this command is used, a proxy table with one column (named "record", type varchar(255)) is created. It is assumed in this case that the contents of the file are readable characters, and that individual records within the file are separated by the newline (\n) character.

You can also specify your own column names and datatypes, using the create [existing] table command:

```
create existing table fname (
    column1 int  null,
    column2 datetime null,
    column3 varchar(1024)  null
    etc. etc.
) external file at "pathname" [column delimiter “<string>”]
```

Columns may be any datatype except text, image, or a Java ADT. The use of the existing keyword is optional, and has no effect on the processing of the statement. If the file referenced by path name does not exist, it is created. If it does exist, its contents are not overwritten. There is no difference in behavior between the create table and create existing table commands.

When a proxy table is mapped to a file, these assumptions about the file and its contents are made:

- The file is not a directory, block special, or character special file.
- The Adaptive Server process has at least read access to the file. If the file is to be created, the server process must have write access to the directory in which the file is to be created.
- The contents of an existing file are in human-readable form.
- Records within the file are delimited by a newline character.
- The maximum supported record size is 32767 bytes.
- Individual columns, except for the last one, are delimited by the column delimiter string, which can be up to 16 bytes long; the default is a single tab character.
- There is a correspondence between delimited values within each record of the file and the columns within the proxy table.

With proxy tables mapped to files, you can:

1 Back up database tables to the file system using either select/into or insert/select. When an insert statement is processed, each column is converted to characters in the default character set of the server. The results of the conversion are buffered, and all columns (except the last) are delimited by a single tab. The last column is terminated by a newline character. The buffer is then written to the file, representing a single row of data.

2 Provide a SQL alternative to using bcp in and bcp out. The use of a select/into statement can easily back up a table to a file, or copy a file’s contents into a table.

3 Query file content with the select statement, qualifying rows as needed with search arguments or functions. For example, you can read the individual records within the Adaptive Server error log file:

```
create proxy_table errorlog
    external file at "/usr/sybase/ase12_5/install/errorlog"
select record from errorlog where record like "%server%"
```

This query returns all rows from the file that match the like pattern. If the rows are longer than 255 bytes, they are truncated. You can specify longer rows by entering:

```
create existing table errorlog
(
    record    varchar(512) null
)
external file at "/usr/Sybase/ase12_5/install/errorlog"
```

In this case, records up to 512 bytes in length are returned. Again, since the proxy table contains only one column, the actual length of each column is determined by the presence of a newline character.

Only the select, insert, and truncate table statements are supported for file access. update and delete result in errors if the file proxy is the target of these commands.

When inserting values into a file, all datatypes are first converted to char values and then delimited by the column delimiter.

Warning! truncate table sets the file size to 0.

Trace flag 11206 is used to log messages to the error log. These messages contain information about the stages of query processing that are involved with file access.

Remote servers

Use sp_addserver to add entries to the sysservers table for the local server and for each remote server that is to be called. The sp_addserver syntax is:

sp_addserver *server_name [,server_class [,network_name]]*

where:

- *server_name* is a unique name used to identify the server.

- *server_class* is the type of server. The supported server classes with the types of servers that are in each class are described in the following sections. The default is server class ASEnterprise.
- *network_name* is the server name in the interfaces file. This name may be the same as *server_name*, or it may differ. The *network_name* is sometimes referred to as the *physical name*. The default is the same name as *server_name*.

Note You can specify the *network_name* in the following additional formats:

```
"hostname:portnumber"
```

or

```
"ipadd:portnumber"
```

For more information, see "Connecting to remote servers without the interfaces file" on page 34, and sp_addserver in "Chapter 1, System Procedures" in *Reference Manual: Procedures*.

Server class ASEnterprise

Adaptive Server uses server class ASEnterprise. When Component Integration Services first establishes a connection to a server in this class, Component Integration Services determines the Adaptive Server version and establishes server capabilities based on the version found. For example, version 12.0 supports ANSI syntax for outer joins, while versions earlier than 12.0 do not.

Server class ASAnywhere

A server with server class ASAnywhere is an instance of Adaptive Server Anywhere or Adaptive Server IQ:

- Adaptive Server Anywhere 6.0 or later

Server class ASIQ

A server with server class ASIQ is Adaptive Server IQ version 12.0 or later.

Server class *db2*

A server with server class db2 is an IBM DB2 database accessed through:

- DirectConnect for MVS / TRS (can also be configured as server class direct_connect)
- Direct (gateway-less) access to Mainframe Connect

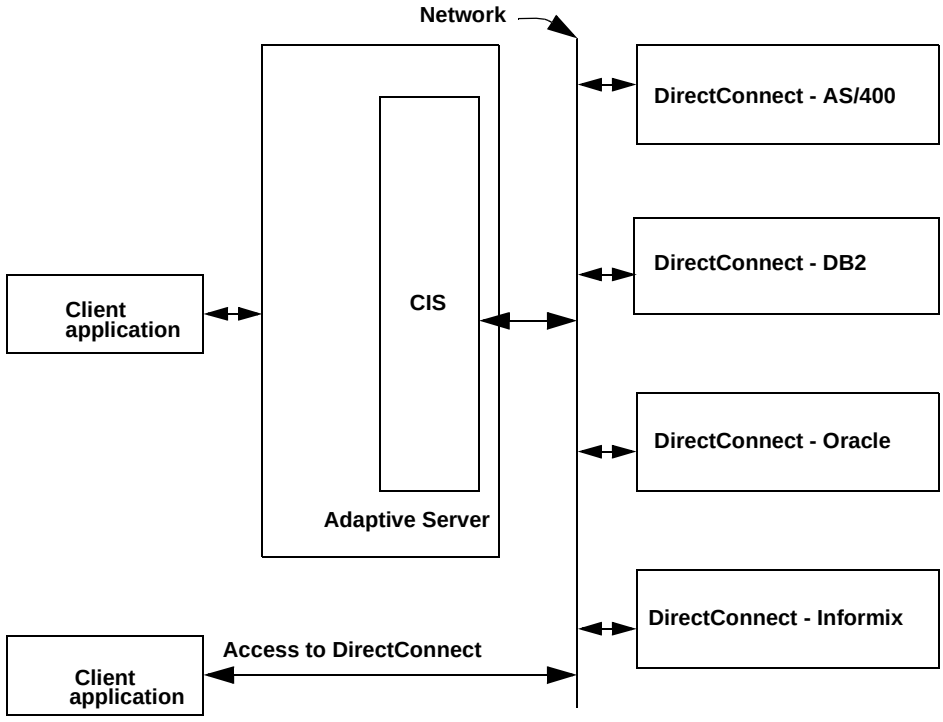
Server class *direct_connect*

A server with server class direct_connect is an Open Server-based application that conforms to the direct_connect interface specification. Server class access_server is synonymous with server class direct_connect. It is used for compatibility.

Open Server-based applications using server class direct_connect are the preferred means of accessing all external, non-Sybase data sources.

Figure 2-1 illustrates how Adaptive Server with Component Integration Services enabled interacts with clients and Open Server-based applications. The data sources are not limited to those in this diagram:

Figure 2-1: Adaptive Server with Component Integration Services interacting with clients and other servers



Server class *sds*

A server with server class sds conforms to the interface requirements of a Specialty Data Store™ as described in the *Adaptive Server Specialty Data Store Developer's Kit* manual. A Specialty Data Store is an Open Server application you design to interface with Adaptive Server.

Connection management

When connecting to a remote server on behalf of a client, Component Integration Services uses Client-Library functions. Once the first connection to a remote server is established for a given client, that connection remains open until the client disconnects from Component Integration Services.

For servers of class ASA, ASE, ASIQ, direct_connect (access_server) and sql_server (version 10.0 and later), only one connection is established to that server for each client that requires access to that server. All interaction with these servers is done with this single connection context.

However, for SQL Server versions earlier than 10.0, and servers of class db2, you may need to establish more than one connection to that server to process a single client request. In this case, multiple connections are established as needed, and all but one are closed when the Transact-SQL command requiring them has completed.

Connecting to remote servers without the interfaces file

You can establish a connection to remote servers without using corresponding entries in directory services *ldap* or *interfaces* files. This is accomplished through Component Integration Services's use of the CT-Library connection property CS_SERVERADDR, which allows a server to be specified in the form:

```
"hostname.domain.com:99999"
"hostname:99999"
"255.255.255.255:99999"
```

where 99999 is the port number, and *hostname* is expressed as a simple name, an IP address, or a complete domain name.

Enter names in this format using sp_addserver with the net name argument:

```
sp_addserver S1, ASEnterprise,
"myhost.sybase.com:11222"
```

or:

```
sp_addserver S1, ASEnterprise, "192.123.321.101:11222"
```

There are some limitations to this usage of net names:

- Adaptive Server site handler does not recognize this syntax.
- Replication Agent threads do not recognize this syntax.

If this syntax is used, CT-Library does not attempt to look up connection information from directory services, whether an interfaces file or LDAP server is configured.

LDAP directory services

The LDAP directory services means that it is no longer necessary to use an interfaces file in both the client and the server. Adaptive Server supports LDAP services for obtaining server information, and so does Component Integration Services. When a connection to a remote server is attempted, Component Integration Services instructs Open Client software to reference either the interfaces file or an LDAP server unless the net name argument to sp_addserver contains a colon (:).

Component Integration Services uses LDAP services only when the configuration file (*libtcl.cfg*) specifies it. *libtcl.cfg* can be found at *$SYBASE/$SYBASE_OCS/config/libtcl.cfg*.

Note When an LDAP server is specified in *libtcl.cfg*, server information becomes accessible from the LDAP server only and Adaptive Server and Component Integration Services ignore any (traditional) interfaces file.

Secure communication with SSL

Using SSL, you can establish secure connections from Component Integration Services to any number of remote servers that support the SSL protocol (Adaptive Server and some DirectConnects).

Component Integration Services handles SSL connections as follows:

- The location of the trusted roots file is established. If the current server is SSL-enabled, then all outbound Component Integration Services connections will use the same trusted roots file as Adaptive Server Enterprise.
- If the current server is SSL-enabled, then a connection property is established to define the Open Client callback that will be used to respond to a challenge from a remote SSL-enabled server. If the current server is not SSL-enabled, then the callback used fails any connection to a remote SSL-enabled server.

Trusted roots files

The trusted roots file contains certificates for other servers that the local server treats as trusted when properly added to the system. If $SYBASE_CERT is defined, a trusted roots file is accessible by the local server (Adaptive Server) in:

```
$SYBASE_CERT/servername.txt
```

Otherwise it is in:

```
$SYBASE/$SYBASE_ASE/certificates/servername.txt
```

(for UNIX)

```
%SYBASE%\%SYBASE_ASE%\certificates\servername.txt
```

(for NT)

where *servername* is the name of the current Adaptive Server.

Security issues

When establishing a connection to a remote Adaptive Server, Client-Library functions are used instead of a site handler when either cis_rpc_handling or set transactional_rpc is on. This method of establishing connections prevents the remote server from distinguishing these connections from those of other clients. Thus, any remote server security configured on the remote server to allow or disallow connections from a given server does not take effect.

Another Adaptive Server with Component Integration Services enabled cannot use *trusted mode* for remote server connections. This forces the Adaptive Server to be configured with all possible user accounts if it is going to be used with Component Integration Services.

Passwords are stored internally in encrypted form.

Remote server logins

To fully support remote logins, Client-Library provides connection properties that enable Component Integration Services to request a *server connection*. This connection is recognized at the receiving server as a server connection (as opposed to an ordinary client connection), allowing the remote server to validate the connection through the use of sysremotelogins as if the connection were made by a site handler.

Server connections are not enabled automatically. Instead, the SSO or DBA must request it by executing sp_serveroption:

```
exec sp_serveroption <server_name>,
    'server login', true | false
```

You cannot change the server login property if the current server's @@servername global variable is NULL.

If the server login option is true, then Component Integration Services uses Client-Library connection properties to establish connections to the specified server.

Remote passwords specified by the client application are passed unchanged to the remote server. The use of and rules associated with remote passwords in server logins are identical to those associated with site handler connections.

These connection properties are only established if:

- The server option server logi is set to true.
- The remote server is configured with server class ASEnterprise.
- There is a local server name defined for the Component Integration Services-enabled server (in other words the query select @@servername returns something other than NULL).

Trusted mode

Trusted mode can be used with Component Integration Services connections if "server logins" is set for a remote server.

Connecting to Backup Server and XP Server

Beginning with Adaptive Server 12.5.1, Component Integration Services can send RPCs to Backup Server or XP Server. Before doing so, the server option negotiated logins must be enabled:

```
exec sp_serveroption server_name, "negotiated logins",
true
```

This allows Component Integration Services to respond to the login challenge initiated by either of these Sybase-provided servers.

Mapping external logins

Adaptive Server users who invoke Component Integration Services require login names and passwords to remote servers. By default, the user name and password pair used by Component Integration Services to connect to a remote server is the same as is used by the client to connect to Adaptive Server.

Component Integration Services supports a one-to-one mapping of Adaptive Server login names and passwords to remote server login names and passwords. For example, using the stored procedure sp_addexternlogin, it is possible to map Adaptive Server user steve, password sybase to DB2 login name login1, password password1:

```
sp_addexternlogin DB2, steve, login1, password1
```

In Adaptive Server version 12.5 and later, you can provide a many-to-one mapping so that all Adaptive Server users who need a connection to DB2 can be assigned the same name and password:

```
sp_addexternlogin DB2, NULL, login2, password2
```

One-to-one mapping has precedence, so that if user steve has an external login for DB2, that would be used rather than the many-to-one mapping.

In addition, you can assign external logins to Adaptive Server roles. With this capability, anyone with a particular role can be assigned a corresponding login name/password for any given remote server:

```
sp_addexternlogin DB2, null, login3, password3, rolename
```

The role name identifies the name of a role, rather than the name of a user. When a user with this role active requires a connection to DB2, the appropriate login name/password for the role is used to establish the connection. When establishing a connection to a remote server for a user who has more than one role active, each role is searched for an external login mapping, and the first mapping found is used to establish the login. This is the same order as displayed by sp_activeroles.

The general syntax for sp_addexternlogin is:

```
sp_addexternlogin
    <servername>,
    <loginname>,
    <external_loginname>,
    <external_password>
    [, <rolename>]
```

<rolename> is optional; if specified then *loginname* is ignored.

Precedence for these capabilities are as follows:

- If one-to-one mapping is defined, it is used.
- If no one-to-one mapping is defined, and a role is active and a mapping for it can be found, the role mapping is used to establish a remote connection.
- If neither of the above are true, then many-to-one mapping is used if defined.
- If none of the above is true, then the Adaptive Server login name and password are used to make the connection.

If role mapping is done, and a user's role is changed (via set role), any connections made to remote servers that used role mapping are disconnected.

sp_helpexternlogin has been updated to allow viewing the various types of external logins that have been added using sp_addexternlogin. The syntax for sp_helpexternlogin is:

```
sp_helpexternlogin [<servername> [,<loginname> [,<rolename>]]]
```

All three parameters are optional, and any of the parameters can be NULL.

The stored procedure sp_dropexternlogin also accepts the *<rolename>* argument. If *<role name>* is specified then the second argument, *<login name>*, is ignored.

Remote server connection failover

If the interfaces file (or LDAP directory service) is set up to define a failover configuration, then Component Integration Services takes advantage of it by automatically failing over connections to the failover server if a connection to the primary server fails.

You can set up remove servers for failover after performing these configuration steps:

1 Enable new server option cis hafailover:

```
exec sp_serveroption server_name, 'cis hafailover',
true
```

2 Modify directory services (interfaces file or server entries in the LDAP server) to specify a failover server

For example, you can configure server S2 to serve as a failover server for S1, and vice-versa, by additions to the interfaces file, as shown in this example:

```
S1
    master tcp ether host1 8000
    query tcp ether host1 8000
    hafailover S2
S2
    master ether host2 9000
    query ether host2 9000
    hafailover S1
```

See *Using Sybase Failover in a High Availability System*, Appendix C, for more discussion of the CS_HAFAILOVER connection property. Component Integration Services uses the ct_con_props() API to set this property, if the cis hafailover server option is true.

Remote server capabilities

The first time Adaptive Server establishes a connection to a remote server of class sds or direct_connect, it issues an RPC named sp_capabilities and expects a result set in return. This result set describes functional capabilities of the remote server so that Component Integration Services can adjust its interaction with that remote server to take advantage of available features. Component Integration Services forwards as much syntax as possible to a remote server, according to its capabilities.

For servers in other classes, Component Integration Services sets remote server capabilities for the remote server based on a set of assumptions. For example, server class db2 inherits a set of assumptions based on known capabilities of IBM's DB2 database management system. For server class ASEnterprise, capabilities are established based on the version of Adaptive Server represented by the remote server.

Query processing

This section describes query processing within Component Integration Services.

Processing steps

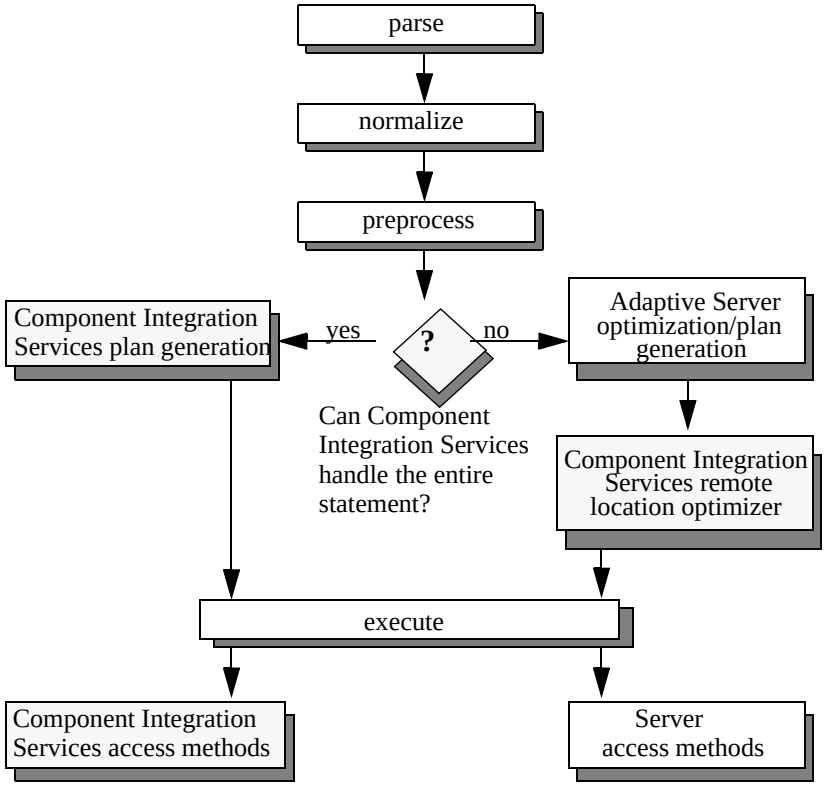
The query processing steps taken when Component Integration Services is enabled are similar to the steps taken by Adaptive Server, except for the following:

- If a client connection is made in passthrough mode, the Adaptive Server query processing is bypassed and the SQL text is forwarded to the remote server for execution.
- When select, insert, delete, or update statements are submitted to the server for execution, additional steps may be taken by Component Integration Services to improve the query's performance, if local proxy tables are referenced.

The query processing steps are shown in Figure 2-2.

An overview of these steps follows.

Figure 2-2: Query processing steps



☐ Shaded boxes indicate steps taken by Component Integration Services.

Query parsing

The SQL parser checks the syntax of incoming SQL statements, and raises an error if the SQL being submitted for execution is not recognized by the Transact-SQL parser.

Query normalization

During query normalization, each object referenced in the SQL statement is validated. Query normalization verifies the objects referenced in the statement exist, and the datatypes are compatible with values in the statement.

Example

```
select * from t1 where c1 = 10
```

The query normalization stage verifies that table t1 with a column named c1 exists in the system catalogs. It also verifies that the datatype of column c1 is compatible with the value 10. If the column's datatype is datetime, for example, this statement is rejected.

Query preprocessing

Query preprocessing prepares the query for optimization. It may change the representation of a statement such that the SQL statement Component Integration Services generates is syntactically different from the original statement.

Preprocessing performs view expansion, so that a query can operate on tables referenced by the view. It also takes steps such as reordering expressions and transforming subqueries to improve processing efficiency. For example, subquery transformation may convert some subqueries into joins.

Decision point

After preprocessing, a decision is made as to whether Component Integration Services or the standard Adaptive Server query optimizer handles optimization.

Component Integration Services handles optimization (using a feature known as quickpass mode) when:

- Every table represented in the SQL statement resides within a single remote server.
- The remote server is capable of processing all the syntax represented by the statement.

  Component Integration Services determines the query processing capabilities of the remote server by its server class. Servers with server class sql_server or db2 have implied capabilities. For example, Component Integration Services assumes that any server configured as server class sql_server is capable of processing all Transact-SQL syntax.

  For remote servers with server class access_server or direct_connect, Component Integration Services issues an RPC to ask the remote server for its capabilities the first time a connection is made to the server. Based on the server's response to the RPC, Component Integration Services determines the syntax of the SQL it forwards to the remote server.
- The following is true of the SQL statement:

- It is a select, insert, delete, or update statement.
- If it is an insert, update, or delete statement, there are no identity or timestamp columns, or referential constraints.
- It contains no text or image columns.
- It contains no compute by clauses.
- It contains no for browse clauses.
- It is not a select...into statement.
- It is not a cursor-related statement (for example, fetch, declare, open, close, deallocate, update, or delete statements that include where current of cursor).

If the above conditions are not met, quickpass mode cannot be used, and the standard Adaptive Server query optimizer handles optimization.

Component Integration Services plan generation

If quickpass mode can be used, Component Integration Services produces a simplified query plan. When statements contain proxy tables, they are executed more quickly when processed by the remote server than when processed through the Adaptive Server plan generation phase.

Adaptive Server optimization and plan generation

Adaptive Server optimization and plan generation evaluates the optimal path for executing a query and produces a query plan that tells the Adaptive Server how to execute the query.

If the update statistics command has been run for the tables in the query, the optimizer has sufficient data on which to base decisions regarding join order. If update statisticshas not been run, the Adaptive Server defaults apply.

For more information on Adaptive Server optimization, see Chapter 7, “The Adaptive Server Query Optimizer,” in the *Performance and Tuning Guide*.

Component Integration Services remote location optimizer

Adaptive Server generates a query plan containing the optimal join order for a multitable query *without* regard to the storage location of each table. If remote tables are represented in the query, Component Integration Services, which takes the storage location into account, performs additional optimization:

- Join processing
- Aggregate processing

To make intelligent evaluations of a query to improve performance in the above areas, statistics are required. These are obtained by executing update statistics for a specific table.

update statistics

When updating statistics on a remote table, Component Integration Services intercepts the request and provides meaningful statistics for the remote table and all of its indexes (if any). The result of executing update statistics is a distribution statistics page stored in the database, for each index.

In Adaptive Server, data used to create this distribution page comes from local index pages. When you are updating statistics on a remote table, the data used to create the distribution statistics page comes from the keys used to make up the index on the remote table.

The Adaptive Server issues a query to the remote server to obtain all columns making up the index, sorted according to position within the index. For example, if table1 has an index made up of two columns, col1 and col2, then the query to that server is sent as follows when update statistics is executed:

```
select col1, col2 from table1 order by col1, col2
```

The results are then used to construct a distribution page in the format needed by the optimizer.

The detailed distribution statistics are used to determine optimal join order. This allows the server to generate optimal queries against remote databases that may not support cost-based query optimization.

On large tables, update statistics can take a long time. To speed up the process, turn on trace flag 11209 before executing update statistics. This trace flag instructs update statistics to obtain only row counts on remote tables. The Adaptive Server query optimizer uses the row count information to make assumptions about the selectivity of a particular index. While these assumptions are not as complete as the full distribution statistics, they provide the minimal information needed to handle query optimization.

Join processing

The Component Integration Services remote location optimizer isolates join conditions represented in the query plan. For each remote server that is represented by two or more tables in the join, Component Integration Services modifies the query plan to appear as though a single virtual table is being processed for that server. Component Integration Services then forwards the join conditions to the remote server during query execution.

For example, if a query involves four tables, two that are located on the remote server SERVERA and two that are located on the remote server SERVERB, Component Integration Services processes the query as though it were a two-way join. This query:

```
select * from A1, A2, B1, B2
 where A1.id = A2.id and A2.id = B1.id
 and B1.id = B2 id
```

gets converted to:

```
select * from V1, V2 where V1.id = V2.id
```

V1 is the virtual table representing the results of the join between A1 and A2 (processed by SERVERA), and V2 is the virtual table representing the results of the join between B1 and B2 (processed by SERVERB). Since the Adaptive Server uses nested iteration (looping) to process inner tables of a join, the query is processed as follows:

```
open cursor on V1
 fetch V1 row
 for each row in V1
    open a cursor on V2
    fetch V2
    route results V1, V2 to client
    close cursor on V2
```

Aggregate processing

Component Integration Services optimizes queries containing ungrouped aggregate functions (min, max, sum, and count) by passing the aggregate to the remote server if the remote server is capable of performing the function.

For example, consider the following query on the remote table A1:

```
select count(*) from A1 where id > 100
```

The count(*) aggregate is forwarded to the remote server that owns A1.

Query execution

The query execution stage receives a query plan, generated either as a result of an ad hoc query or a stored procedure, and executes each step of the plan according to the information stored in the plan. Query plan structures are tagged with information that indicates which access method is to be invoked. If a table is local, then normal Adaptive Server access methods used to process a query are activated as required by the plan execution logic. If the table is remote, then Component Integration Services access methods are invoked to process each table (or virtual table) represented in the query.

Distributed query optimization

The performance of queries involving proxy tables that reference two or more remote servers is critical to the success of the Component Integration Services features incorporated into Adaptive Server. Several optimization strategies are provided to make distributed query processing as optimal as possible within the constraints of the current Adaptive Server query processor.

Optimizer cost model for proxy tables

In versions earlier than 12.5, the Adaptive Server optimizer was modified to incorporate the cost of network access to remote servers. The network cost was hard-coded into the Adaptive Server optimizer as an algorithm that assumes network exchanges are required to:

- Open a cursor
- Fetch 50 rows
- Close a cursor

An exchange is required for each 50 rows. The cost of an exchange in versions was hard-coded at 100 milliseconds. With version 12.5, the cost of a single exchange is under the user's control, and is specified on a per-server basis, defaulting to 1000 milliseconds, by sp_serveroption:

```
sp_serveroption <servername>,  "server cost", "nnnn"
```

where *nnnn* is a string of numeric digits representing the number of milliseconds to be used per exchange during the optimizer's calculation of network cost. "server cost" represents a server option introduced in Adaptive Server version 12.5.

Note The server cost limit is 32767. If you exceed that limit, an arithmetic overflow error occurs.

When a new server is added to sysservers using sp_addserver, the default cost, 1000ms, is stored in sysattributes for that server. sp_serveroption can be used to specify a greater or lesser cost for a given server. sp_helpserver shows the current network cost associated with the server.

Sort/merge joins

Sort/merge joins are enabled as a possible join strategy for joining local tables. However, they are disabled if any table in a query is a proxy table. Joins between proxy tables are not managed by the sort/merge algorithm.

Semijoins (reformatting)

Reformatting allows the contents of the inner table of a nested loop join to be transferred to a work table. A clustered index is created on the join column of the work table, and subsequent join operations use this work table rather than the original.

When a proxy table is chosen to be the inner table of a nested loop join, the reformatting strategy can result in significant performance improvements, as the network is only accessed once, rather than for each row obtained by the outer tables.

Component Integration Services access methods

The Component Integration Services access methods interact with the remote servers that contain objects represented in a query. In Adaptive Server all interaction is done through Client-Library.

When an entire statement can be forwarded to the remote server, the statement is taken from the query plan. After any parameters have been substituted into the text of the statement, the entire statement is forwarded to the appropriate remote server.

When the Adaptive Server optimizer and plan generator are involved, the statement or fragment of a statement that is to be executed remotely is constructed from data structures contained within the query plan. The statement or fragment of a statement is then forwarded to the appropriate remote server.

The results from the remote servers are then converted into the necessary internal data types, and processed as if they were derived from local tables.

When an order by is processed by the remote server, the results may be different from what Adaptive Server would return for the same query, because the sort order is determined by the remote server, not by Adaptive Server.

Query plan execution

Any command that can affect a table is checked by the server to determine whether the object has a local or remote storage location. If the storage location is remote, then the appropriate access method is invoked when the query plan is executed in order to apply the requested operation to the remote objects. The following commands are affected if they operate on objects that are mapped to a remote storage location:

- alter table
- begin transaction
- commit
- create index
- create table
- create existing table
- deallocate table
- declare cursor
- delete
- drop table
- drop index
- execute
- fetch
- insert
- open
- prepare transaction
- readtext
- rollback

- select
- set
- setuser
- truncate table
- update
- update statistics
- writetext

create table

When the server receives a create table command, the command is interpreted as a request for new table creation. The server invokes the access method appropriate for the server class of the table that is to be created. If it is remote, the table is created. If this command is successful, system catalogs are updated, and the object appears to clients as a local table in the database in which it was created.

create table is reconstructed in a syntax that is appropriate for the server class. For example, if the server class is db2, the command is reconstructed using DB2 syntax before being passed to the remote server. Datatype conversions are made for datatypes that are unique to the Adaptive Server environment.

Some server classes have restrictions on what datatypes can and cannot be supported.

create table is passed to remote servers as a language request.

create existing table

When a create existing table command is received, it is interpreted as a request to import metadata from the remote or external location of the object for updating system catalogs. Importing this metadata is performed by means of three RPCs sent to the remote server with which the object has been associated:

- sp_tables – verifies that the remote object actually exists.
- sp_columns – obtains column attributes of the remote object for comparison with those defined in create existing table.
- sp_statistics – obtains index information to update the local system table, sysindexes.

alter table

When the server receives the alter table command, it passes the command to an appropriate access method if:

- The object on which the command is to operate has been associated with a remote or external storage location.
- The command consists of an add column request. Requests to add or drop constraints are not passed to the access methods; instead, they are handled locally.

alter table is passed to remote servers as a language request.

create index command

When the server receives the create index command, it passes the command to an appropriate access method, if the object on which the command is to operate has been associated with a remote or external storage location.

The command is reconstructed using a syntax appropriate for the class and is passed to the remote server for execution.

create index is passed to remote servers as a language request.

drop table

When the server receives the drop table command for a remote table, a check is made to determine whether the table to be dropped has been created with the existing option. If so, references to the object within the system tables are removed, and the operation is complete.

If the table was not created with the existing option, the command is passed to an appropriate access method, if the object on which the command is to operate has been associated with a remote or external storage location.

drop table is reconstructed using a syntax appropriate for the class and is passed to the remote server for execution.

This command is passed to remote servers as a language request.

In all cases, references to the object from within the system catalogs are removed.

drop index

When the server receives the drop index command, it passes the command to an appropriate access method, if the object on which the command is to operate has been associated with a remote or external storage location.

drop index is reconstructed using a syntax appropriate for the class and is passed to the remote server for execution.

This command is passed to remote servers as a language request.

truncate table

When the server receives the truncate table command, it passes the command to an appropriate access method, if the object on which the command is to operate has been associated with a remote or external storage location.

The command is reconstructed using a syntax appropriate for the class and is passed to the remote server for execution. Since this syntax is unique to the Adaptive Server environment, a server of class db2 would receive a delete command with no qualifying where clause:

```
delete from t1
```

truncate table is passed to remote servers as a language request.

Passthrough mode

Passthrough mode is provided within Component Integration Services as a means of enabling a user to perform native operations on the server to which the user is being "passed through."

For example, requesting passthrough mode for an Oracle server allows you to send native Oracle SQL statements to the Oracle DBMS. Results are converted into a form that is usable by the Open Client application and passed back to the user.

The Transact-SQL parser and compiler are bypassed in this mode, and each language batch received from the user is passed directly to the server to which the user is connected in passthrough mode. Results from each batch are returned to the client.

There are several ways to use passthrough mode:

- connect to
- sp_autoconnect

- sp_passthru
- sp_remotesql

connect to

The connect to command enables users to specify the server to which a passthrough connection is required. The syntax of the command is:

```
connect to server_name
```

where *server_name* is the name of a server added to the *sysservers* table, with its server class and network name defined. See sp_addserver in the *Reference Manual*.

When establishing a connection to *server_name* on behalf of the user, the server uses:

- A remote login alias set using sp_addexternlogin, or
- The name and password used to communicate with the Adaptive Server.

In either case, if the connection cannot be made to the server specified, the reason is contained in a message returned to the user.

Once a passthrough connection has been made, the Transact-SQL parser and compiler are bypassed when subsequent language text is received. Any statements received by the server are passed directly to the specified remote server.

Note Some database management systems do not recognize more than one statement at a time and produce syntax errors if, for example, multiple select statements were received as part of a single language text buffer.

After statements are passed to the requested server, any results are converted into a form that can be recognized by the Open Client interface and sent back to the client program.

To exit from passthrough mode, issue the disconnect, or disc, command. Subsequent language text from this client is then processed using the Transact-SQL parser and compiler.

Permission to use connect to must be explicitly granted by the System Administrator. The syntax is:

```
grant connect to user_name
```

To revoke permission to use connect to, the syntax is:

revoke connect from *user_name*

The connect to permissions are stored in the master database. To globally grant or revoke permissions to "public", the System Administrator sets the permissions in the master database; the effect is server-wide, regardless of what database is being used. The System Administrator can only grant or revoke permissions to or from a user if the user is a valid user of the master database.

The System Administrator can grant or revoke "all" permissions to or from "public" within any database. If the permissions are in the master database, "all" includes the connect to command. If they are in another database, "all" does not include the connect to command.

Example

The System Administrator wants to revoke permission from "public" and wants only the user "fred" to be able to execute the connect to command. "fred" must be made a valid user of master. To do this, the System Administrator issues the following commands in master:

```
revoke connect from public
 sp_adduser fred
 grant connect to fred
```

sp_autoconnect

Some users may always require a passthrough connection to a given server. If this is the case, Component Integration Services can be configured so that it automatically connects these users to a specified remote server in passthrough mode when the users connect to the server. This feature is enabled and disabled by sp_autoconnect using this syntax:

sp_autoconnect *server_name*, true|false [,*loginname*]

Before using sp_autoconnect, add the *server_name* to sysservers using sp_addserver.

A user can request automatic connection to a server using sp_autoconnect, but only the System Administrator can enable or disable automatic passthrough connection for another user. Thus, only the System Administrator can specify a third argument to this procedure.

If the second argument is true, the autoconnect feature is enabled for the current user (or the user specified in the third argument). If the second argument is false, autoconnect is disabled.

When a user connects to the server, that user's autoconnect status in syslogins is checked. If enabled, the *server_name*, also found in syslogins (placed there by sp_autoconnect), is checked for validity. If the server is valid, the user is automatically connected to that server, and a passthrough status is established. Subsequent language statements received by the server from this user are handled exactly as if the user explicitly entered the connect command. This user then views the server similar to a passthrough gateway to the remote server.

When an "autoconnected" user executes a disconnect, she or he is returned normally to the server.

If the remote server cannot be reached, the user (unless the user is assigned the "sa" role) will not be connected to the local Adaptive Server. A "login failed" error message is returned.

sp_passthru

sp_passthru allows the user to pass a SQL command buffer to a remote server. The syntax of the SQL statements being passed is assumed to be the syntax native to the class of server receiving the buffer; no translation or interpretation is performed. Results from the remote server are optionally placed in output parameters. The syntax for sp_passthru follows:

```
sp_passthru server, command, errcode, errmsg, rowcount
 [, arg1, arg2, ... argn]
```

where:

- *server* is the name of the server that is to receive the SQL command buffer; the datatype is varchar(30).
- *command* is the SQL command buffer; the datatype is varchar(255).
- *errcode* is the error code returned by the remote server; the datatype is int output.
- *errmsg* is the error message returned by the remote server; the datatype is varchar(255) output.
- *rowcount* is the number of rows affected by the last command in the command buffer; the datatype is int output.
- *arg1 – argn* are optional parameters. If provided, these output parameters will receive the results from the last row returned by the last command in the command buffer. The datatypes may vary. All must be output parameters.

Example

```
sp_passthru ORACLE, "select date from dual", @errcode
output, @errmsg output, @rowcount output, @oradate
output
```

This example returns the date from the Oracle server in the output parameter @oradate. If an Oracle error occurs, the error code is placed in @errcode and the corresponding message is placed in @errmsg. The @rowcount parameter is set to 1.

For more information on sp_passthru and its return status, see the *Reference Manual*.

sp_remotesql

sp_remotesql allows you to pass native syntax to a remote server. The procedure establishes a connection to a remote server, passes a query buffer, and relays the results back to the client. The syntax for sp_remotesql is as follows:

sp_remotesql server_name, *query_buf1*
[, *query_buf2*, ... , *query_buf254*]

where:

- *server_name* is the name of a server that has been defined using sp_addserver.
- *server_name* is a varchar(30) field. If *server_name* is not defined or is not available, the connection fails, and the procedure is aborted. This parameter is required.
- *query_buf1* is a query buffer of type char or varchar with a maximum length of 255 bytes. This parameter is required.

Each additional buffer is char or varchar with a maximum length of 255 bytes. If supplied, these optional arguments are concatenated with the contents of *query_buf1* into a single query buffer.

Example

```
sp_remotesql freds_server, "select @@version"
```

In this example, the server passes the query buffer to *freds_server*, which interprets the select *@@version* syntax and returns version information to the client. The returned information is not interpreted by the server.

For more information on sp_remotesql and its return codes, see the *Reference Manual*.

Quoted identifier support

Quoted identifiers are forwarded to remote servers that support them. This is triggered by a set command:

```
set quoted_identifier on
```

If this thread property is enabled, Component Integration Services quotes identifiers before sending SQL statements to remote servers.

Remote servers must have the ability to support quoted identifiers. There is a capability in the sp_capabilities result set reserved for this purpose:

- Capability ID: 135
- Capability name: quoted identifier
- Capability value: 0 = no support; 1 = supported

The capability defaults to 0 for DirectConnects that do not provide a value for this capability.

auto identity option

When the Adaptive Server auto identity database option is enabled, an IDENTITY column is added to any tables that are created in the database. The column name is CIS_IDENTITY_COL, for proxy tables, or SYB_IDENTITY_COL, for local tables. In either case, the column can be referenced using the syb_identity keyword.

Triggers

Component Integration Services allows triggers on proxy tables; however, their usefulness is limited. You can create a trigger on a proxy table and the trigger is invoked just as it would be for a normal Adaptive Server table. However, before and after image data is not written to the log for proxy tables because the insert, update, and delete commands are passed to the remote server. The inserted or deleted tables, which are actually views into the log, contain no data for proxy tables. Users cannot examine the rows being inserted, deleted, or updated, so a trigger with a proxy table has limited value.

RPC handling and Component Integration Services

When Component Integration Services is enabled, you can choose between the site handler or Component Integration Services to handle outbound remote procedure calls (RPCs). Each of these mechanisms is described in the following sections.

Site handler and outbound RPCs

Within an Adaptive Server, outgoing RPCs are transmitted by means of a site handler, which multiplexes multiple requests through a single physical connection to a remote server. The RPC is handled as part of a multistep operation:

1 Establish connection – the Adaptive Server site handler establishes a single physical connection to the remote server. Each RPC requires that a logical connection be established over this physical connection. The logical connection is routed through the site handler of the intended remote server.

   The connection validation process for these connect requests is different from that of normal client connections. First, the remote server must determine if the server from which the connect request originated is configured in its sysservers table. If so, then the system table sysremotelogins is checked to determine how the connect request should be handled. If trusted mode is configured, password checking is not performed. (For more information about trusted mode, see “Trusted mode” on page 37.)

2 Transmit the RPC – the RPC request is transmitted over the logical connection.

3 Process results – all results from the RPC are relayed from the logical connection to the client.

4 Disconnect – the logical connection is terminated.

Because of the logical connect and disconnect steps, site handler RPCs can be slow.

Component Integration Services and outbound RPCs

If Component Integration Services has been enabled, a client can use one of two methods to request that Component Integration Services handle outbound RPC requests:

- Configure Component Integration Services to handle outbound RPCs as the default for all clients by issuing:

  ```
  sp_configure "cis rpc handling", 1
  ```

  If you use this method to set the cis rpc handling configuration parameter, all new client connections inherit this behavior, and outbound RPC requests are handled by Component Integration Services. This is a server property inherited by all future connections. The client can, if necessary, revert back to the default Adaptive Server behavior by issuing the command:

  ```
  set cis_rpc_handling off
  ```

- Configure Component Integration Services to handle outbound RPCs for the current connection only by issuing:

  ```
  set cis_rpc_handling on
  ```

  This command enables cis rpc handling for the current thread only, and does not affect the behavior of other threads.

When cis rpc handling is enabled, outbound RPC requests are not routed through the Adaptive Servers site handler. Instead, they are routed through Component Integration Services, which uses persistent Client-Library connections to handle the RPC request. Using this mechanism, Component Integration Services handles outbound RPCs as follows:

1 Determines whether the client already has a Client-Library connection to the server in which the RPC is intended. If not, establish one.

2 Sends the RPC to the remote server using Client-Library functions.

3 Relays the results from the remote server back to the client program that issued the RPC using Client-Library functions.

RPCs can be included within a user-defined transaction. In fact, all work performed by Component Integration Services on behalf of its client can be performed within a single connection context. This allows RPCs to be included in a transaction's unit of work, and the work performed by the RPC can be committed or rolled back with the other work performed within the transaction.

The side effects of using Component Integration Services to handle outbound RPC requests are as follows:

- Client-Library connections are persistent so that subsequent RPC requests can use the same connection to the remote server. This can result in substantial RPC performance improvements, since the connect and disconnect logic is bypassed for all but the first RPC.
- Work performed by an RPC can be included in a transaction, and is committed or rolled back with the rest of the work performed by the transaction. This transactional RPC behavior is currently supported only when the server receiving the RPC is another Adaptive Server or a DirectConnect which supports transactional RPCs.
- Connect requests appear to a remote server as ordinary client connections. The remote server cannot distinguish the connection from a normal application's connection, unless server logins are enabled. This affects the remote server management capabilities of an Adaptive Server, since no verification is performed against sysremotelogins, and all connections must have valid Adaptive Server login accounts established prior to the connect request (trusted mode cannot be used in this case).

Text parameters for RPCs

Adaptive Server can send large chunks of data in a single remote procedure call. This is done by treating certain parameters as text pointers, then dereferencing these text pointers to obtain the text values associated with them. The text data is then packaged into 16K chunks for Adaptive Server and 32K chunks for all other servers, and handed to Client-Library as parameters to the RPC.

A text pointer is identified as a parameter of type binary(16) or varbinary(16). The text value referenced by each text pointer parameter is obtained when the RPC is executed, and expanded into 16K chunks for Adaptive Server and 32K chunks for all other servers, each of which is passed to Client-Library as a parameter of type CS_LONGCHAR_TYPE.

This behavior is triggered by this set command:

```
set textptr_parameters ON
```

When an RPC is requested (cis_rpc_handling must be on), text pointers are dereferenced in the Component Integration Services layer, and the text value obtained is used to construct one or more parameters for Client-Library.

For this to work, the text pointers must be preceded by a path name argument, which is used to identify the table from which the text pointers have been derived. For example:

```
declare @pathname varchar(90)
declare @textptr1 binary(16)
declare @textptr2 binary(16)
select @pathname = "mydatabase.dbo.t1",
            @textptr1 = textptr(c1),
            @textptr2 = textptr(c2)
      from mydatabase.dbo.t1
      where ... (whatever)
set textptr_parameters ON
exec NETGW...myrpc @pathname, @textptr1, @textptr2
set textptr_parameters OFF
```

When the RPC named 'myrpc' gets sent to server NETGW, the *@pathname* parameter is not actually sent, but is used to help locate the text values referenced by the textptr's *@textptr1* and *@textptr2*.

The varchar parameter *@pathname* must immediately precede the binary(16) parameter, otherwise *@textptr1* is considered an ordinary parameter and is transmitted to the server NETGW as a normal binary(16) value.

The text will be broken into 16K or 32K chunks, each of which is a separate parameter of type CS_LONGCHAR_TYPE.

The current value of *@@textsize* is ignored.

This scheme is also designed to work with proxy tables mapped to remote procedures. For example:

```
create existing table myrpctable
(
      id int,     -- result column
      crdate datetime, -- result column
      name varchar(30), -- result column
      _pathname varchar(90), -- parameter column
      _textptr1 binary(16), -- parameter column
      _textptr2 binary(16), -- parameter column
) external procedure at 'NETGW...myrpc'
go
declare @textptr1 binary(16)
declare @textptr2 binary(16)
select @textptr1 = textptr(c1), @textptr2 = textptr(c2)
from mydatabase.dbo.t1 where <whatever>
set textptr_parameters ON
select id, crdate, name
from myrpctable
where_pathname = "mydatabase.dbo.t1" and
      _textptr1 = @textptr1 and
      _textptr2 = @textptr2
```

When the query against the proxy table myrpctable is processed, Component Integration Services sends an RPC named ‘myrpc’ to the server ‘NETGW’. The parameters will be derived from the search arguments contained in the where clause of the query. Since the textptr_parameter option has been set ON, the textptrs are expanded to CS_LONGCHAR_TYPE, as in the case of the RPC example shown previously.

Text parameter support for XJS/390

Because of the ability to forward large blocks of text as RPC parameters, it is now possible for Component Integration Services to interact with IBM mainframes using XJS/390. XJS/390 scripts (JavaScript-like syntax) can be stored within Adaptive Server tables (or files accessible via proxy tables), and forwarded to the mainframe using an RPC. The syntax of the script is analyzed and executed by XJS/390 facilities, and result sets are generated according to the procedural logic of the script.

Several features are enabled:

- Database events within Adaptive Server can result in the generation of an MQ Series message. Since XJS/390 Mscript supports the generation of messages, an RPC can be sent to the mainframe to request that such a message be generated in response to a triggered event within the database.
- Component Integration Services users have access to VSAM, IMS, and MQSeries data without the need to install third-party middleware such as InfoHub.

Version 2.0 or later of XJS/390 is required for handling scripts as RPC parameters. See the XJS/390 specification for details.

Distributed Transaction Management

Distributed Transaction Management within Adaptive Server tracks the state of a transaction in the local Adaptive Server/Component Integration Services, as well as in all remote servers participating in transactions. When a user application commits a transaction, the commit is propagated to all participating remote servers using Adaptive Server Transaction Coordinator (ASTC). The management of multisite transactions is handled by ASTC in cooperation with Component Integration Services. Component Integration Services registers new participating servers for each transaction, then turns over control of the transaction coordination to ASTC, which calls back into Component Integration Services to execute various commands for transaction management.

Component Integration Services version 12.5 introduces a powerful mechanism for supporting distributed transaction management transparently using only Adaptive Server, which supports transparent two-phase commit services between local and remote Adaptive Server 12.0 or later, involving both RPC and DML (select, insert, delete, update) operations. In version 12.0, transparent two-phase commit was limited to DTM-enabled Adaptive Server (12.0 and later). However, with Adaptive Server 12.5 and later, this limit has been removed and support for DTM-enabled DirectConnects has been provided. A DirectConnect indicates its ability to handle two-phase commit transactions in the capability response resulting from the sp_capabilities RPC; currently, only Direct Connect for Oracle supports the full DTM-enabled behavior that enables two-phase commits.

Server classes and ASTC

Internally, ASTC views a server as one of three types:

- DTM-enabled
- Pre-DTM
- No-DTM

These types map to the three sets of callbacks used, and map to server classes as indicated in Table 2-5:

Table 2-5: Transaction capabilities

ASTC server type	Component Integration Services server class
DTM-enabled	ASEnterprise (12.x or greater) DC/Oracle 12.5
Pre-DTM	ASEnterprise (pre-12.0) ASAnywhere ASIQ other Direct Connect sds
No-DTM	db2

Note Before starting a distributed transaction, the local server must be named. *@@servername* cannot be null.

DTM-enabled servers

Remote servers that are "DTM-enabled" support the full two-phase commit service enabled by ASTC. Servers that support this must allow a separate connection (or session) to either commit or roll back a transaction that was begun by another session. This capability is necessary if the commit coordinator (ASTC) is required to connect to a remote site and commit or roll back in-doubt transactions. Adaptive Server 12.0 and later provide this support, as does DirectConnect for Oracle 12.5.

Pre-DTM servers

Remote servers that are classified as "pre-DTM" are those that support transaction management statements such as begin tran, commit tran, rollback tran, but do not support for one session's ability to commit or rollback a transaction started by another session.

Component Integration Services makes every effort to manage user transactions for pre-DTM servers reliably. However, different access methods incorporated into the server allow varying degrees of support for this capability. The general logic described below is employed by server classes ASEnterprise (prior to 12.0), ASAnywhere, ASIQ, direct_connect, and sds if the Specialty Data Store supports transaction management. The method for managing transactions involving remote servers uses a two-phase commit protocol. Adaptive Server implements a strategy that ensures transaction integrity for most scenarios. However, there is still a chance that a distributed unit of work will be left in an undetermined state. Even though two-phase commit protocol is used, no recovery process is included. The general logic for managing a user transaction is as follows:

Component Integration Services prefaces work to a remote server with a begin transaction notification. When the transaction is ready to be committed, Component Integration Services sends a prepare transaction notification to each remote server that has been part of the transaction. prepare transaction pings the remote server to determine whether the connection is still viable. If a prepare transaction request fails, all remote servers are told to roll back the current transaction. If all prepare transaction requests are successful, the server sends a commit transaction request to each remote server involved with the transaction. Any command preceded by begin transaction can begin a transaction. Other commands are sent to a remote server to be executed as a single, remote unit of work.

No-DTM servers

Remote servers that are classified as “no-DTM” do not properly support any transaction management. This is not necessarily due to inherent limitations of the remote DBMS, but may be due to the limitations of the gateway or direct_connect used to interface between Adaptive Server and Component Integration Services and the remote DBMS. For servers in this category, Adaptive Server and Component Integration Services does not send any transaction control statements, and does not attempt to manage transactions in any way.

strict DTM enforcement

To ensure complete two-phase commit capability, ASTC uses the concept of strict dtm enforcement. When enabled, strict dtm enforcement causes a transaction to abort if an attempt is made to include a pre-DTM or no-DTM server in the transaction.

enable xact coordination

ASTC uses the configuration option enable xact coordination. This option, enabled by default, allows ASTC to manage all transactions involving remote servers. You must enable Component Integration Services before xact coordination is enabled. While xact coordination is enabled, Component Integration Services cannot be disabled. When xact coordination is enabled, transactional_rpcs are implicitly enabled.

Enable Component Integration Services

ASTC relies on Component Integration Services to handle all communication with remote servers. Since ASTC is enabled by default (enable xact coordination), Component Integration Services is also enabled by default.

Component Integration Services set commands

The behavior of the cis rpc handling configuration property and the set transactional_rpc commands changed with the introduction of ASTC. In earlier versions, enabling cis rpc handling caused all RPCs to be routed through the Component Integration Services Client-Library connection. As a result, whenever cis rpc handling was enabled, transactional_rpc behavior occurred whether or not it had been specifically set.

With Adaptive Server version 12.x, this behavior has changed. If cis rpc handling is enabled and transactional_rpcs is off, RPCs within a transaction are routed through the site handler. RPCs executed outside a transaction are sent via the Component Integration Services Client-Library connection. Table 2-6 illustrates this change in functionality. As with earlier versions, cis rpc handling is disabled by default.

Table 2-6: CIS RPC handling and transactional RPCs

	12.x	Pre-12.x
CIS RPC handling off; transactional RPCs off	Non-transactional	Non-transactional
CIS RPC handling on; transactional RPCs off	Non-transactional	Transactional
CIS RPC handling on; Transactional RPCs on	Transactional	Transactional

Transactional RPCs

The server allows RPCs to be included within the unit of work initiated by the current transaction.

Before using transactional RPCs, issue the set transactional_rpc on command.

Assuming that the remote server can support the inclusion of RPCs within transactions, the following syntax shows how this capability might be used:

```
begin transaction
 insert into t1 values (1)
 update t2 set c1 = 10
 execute @status = RMTSERVER.pubs2.dbo.myproc
 if @status = 1
      commit transaction
 else
      rollback transaction
```

In this example, the work performed by the procedure *myproc* in server RMTSERVER is included in the unit of work that began with the begin transaction command. This example requires that the remote procedure *myproc* return a status of "1" for success. The application controls whether the work is committed or rolled back as a complete unit.

The server that is to receive the RPC must allow RPCs to be included in the same transactional context as Data Manipulation Language (DML) commands (select, insert, delete, update). This is true for Adaptive Server and is expected to be true for most DirectConnect products released by Sybase. However, some database management systems may not support this capability.

Restrictions on transaction management

If nested begin transaction and commit transaction statements are included in a transaction that involves remote servers, only the outermost set of statements is processed. The innermost set, containing the begin transaction and commit transaction statements, is not transmitted to remote servers.

Using *update statistics*

The update statistics command helps the server make the best decisions about which indexes to use when it processes a query, by providing information about the distribution of the key values in the indexes. update statistics does not automatically run when you create or re-create an index on a table that already contains data. It can be used when a large amount of data in an indexed column has been added, changed, or deleted. The crucial element in query optimization is the accuracy of the distribution steps. If there are significant changes in the key values in the index, re-run update statistics on that index.

Only the table owner or the System Administrator can issue the update statistics command.

The syntax is:

update statistics *table_name* [*index_name*]

Try to run update statistics at a time when the tables you specify are not heavily used. update statistics acquires locks on the remote tables and indexes as it reads the data. If you use trace flag 11209, tables are not locked.

The server performs a table scan for each index specified in the update statistics command.

Since Transact-SQL does not require index names to be unique in a database, you must give the name of the table with which the index is associated.

After running update statistics, run sp_recompile so triggers and procedures that use the indexes use the new distribution:

```
sp_recompile authors
```

Finding index names

You can find the names of indexes by using sp_helpindex. This procedure takes a table name as a parameter.

To list the indexes for the authors table, enter:

```
sp_helpindex authors
```

To update the statistics for all of the indexes in the table, enter:

```
update statistics authors
```

To update the statistics only for the index on the au_id column, enter:

```
update statistics authors auidind
```

Java in the database

Java in the database is supported for remote data access with Component Integration Services.

The following restrictions apply:

- Java is supported for remote Adaptive Server 12.x and later only.
- Java is supported for language events only (no dynamic SQL can be used with remote tables.)

Before using Java for remote data access, read "Java class definitions" on page 73. Then, after installing your Java class files on the local server, install the required Java class files on the remote server.

@@textsize

Data is returned as a serialized Java object using the image datatype format and then deserialized on the local server. @@textsize must be set large enough to hold the serialized object. If @@textsize is set too small, the object is truncated, and the deserialization fails.

@@stringsize

@@stringsize indicates the amount of character data to be returned from a toString() method. It is similar in behavior to @@textsize, except it applies only the char data returned by the Java Object.toString() method. The default value is 50. The maximum value is 16384. A value of zero means "use the default." This value can be modified by a set command:

```
set stringsize n
```

where *n* is an integer value between 0 and 16384. The value immediately displays in the global variable @@stringsize.

Constraints on Java class columns

Constraints defined on Java columns of remote tables must be checked on the remote server. If the constraint checking is attempted on the local server, it fails. Therefore, you must enable trace flag 11220 when you insert, update, or delete data for which constraint checking is done on Java datatypes. See "Trace flags" on page 90.

Error messages

There are two error messages that are specific to Java use with remote data access:

- Error 11275 – a statement referencing an extended datatype contained syntax that prevented it from being sent to the remote server. Rewrite the statement or remove the extended datatype reference.
- Error 11276 – an object in column '<colname>' could not be deserialized, possibly because the object was truncated. Check that the value of @@textsize is large enough to accommodate the serialized object.

SQLJ in Adaptive Server Enterprise

The SQLJ effort in Adaptive Server consists of enhancements to the 12.0 Java classes to bring the implementation into compliance with the current SQLJ standard proposal. In particular, the SQLJ proposal addresses Part 1 of the SQLJ standard – which specifies conventions for calling static Java methods as stored procedures and user-defined functions (UDFs).

Changes to Component Integration Services

The create function statement allows keyword, exportable, that determines whether or not a function can be exported to a remote site. It does not specify that the function must be exported, it specifies only that the function may be exported. The syntax is:

```
create function
      sql_function_name
      sql_function_signature
      sql_properties
      external_java_reference

sql_function_name ::= [[identifier1.]identifier2.]identifier3
sql_function_signature ::= ( [sql_parameters]) returns sql_datatype
sql_parameters ::= sql_parameter [{, sql_parameter}...]
sql_parameter ::= [parameter_mode] [sql_identifier] sql_datatype
parameter_mode ::= in | out | inout
sql_properties ::=
      [modifies sql data]
      | [dynamic result sets integer]
      | [deterministic | not deterministic ]
      | [returns null on null input | called on null input ]
      | [exportable]
      | language java
      | parameter style java
external_java_reference ::=
      external name 'java_method_name [java_method_signature]'
java_method_name ::= java_class_name.method_identifier
java_class_name ::= [packages.]class_identifier
packages ::= package_identifier[.package_identifier]...
package_identifier ::= java_identifier
class_identifier ::= java_identifier
method_identifier ::= java_identifier
java_method_signature ::= ( [ java_parameters ])
java_parameters ::= java_datatype [{, java_datatype}...
```

When Component Integration Services encounters a SQL function, a check for the presence of the exportable keyword is made during the Component Integration Services query decomposition phase. If the function is exportable, the statement is a candidate for quickpass mode. If the function is not exportable, it causes the statement to be thrown out of quickpass mode.

In quickpass mode, the function name and argument list are forwarded to a remote server, even if the capability for Java UDF indicates that Java functions are not supported. This allows a Component Integration Services administrator to create packages of functions that, for example, emulate behavior of foreign database systems (Oracle, DB2, and so on). In this way, SQLJ functions can be created within Component Integration Services that allow Component Integration Services to forward statements in quickpass mode even though the function named in the query is not standard Transact-SQL.

If the statement containing the function cannot be forwarded to the remote server in quickpass mode, Component Integration Services still provides for the correct execution of the function by retrieving data from remote sites, then invoking the function locally to operate on the recently-fetched remote data.

Java abstract datatypes (ADTs)

Java Classes in SQL (JCS) is the method of storing and using Java objects within the Adaptive Server. Component Integration Services interaction in this implementation is needed to support Java objects and Java functions on remote servers.

Component Integration Services supports JCS on remote Adaptive Server version 12.0 or later.

Objects are passed between the local and remote servers in a serialized format that is a binary representation used to reinstantiate the object. Component Integration Services treats a serialized object as an image blob, using text and image handling functions to pass objects between servers. The object is reinstantiated on the destination server before processing continues.

When handling queries containing references to Java objects and functions on remote servers, Component Integration Services attempts to forward as much syntax as possible to the remote server. Any portion of the query that cannot be passed to the remote server is handled on the local server, requiring the serialization and deserialization of all necessary remote objects. Due to the overhead associated with serializing and deserializing Java objects, performance of such queries is significantly less than comparable local access.

To facilitate the interchange of Java objects between servers, Component Integration Services issues:

```
set raw_object_serialization ON
```

to each ASEnterprise server that is Java-enabled. This allows Component Integration Services to easily deserialize the object obtained from the remote site.

Java class definitions

The Java class definitions on the local and remote servers must be compatible to facilitate passing objects between servers. For this reason, Component Integration Services assumes that compatibility exists, and any errors in object definition are detected during deserialization efforts. Objects are considered compatible if the serialized form of the object on the remote server can be used to successfully instantiate an object on the local server, or vise versa. Also, any Java method referenced in the local server in conjunction with a remotely mapped object must be defined on the remote object as well.

It is the responsibility of the database administrator to ensure that class definitions on local and remote servers are compatible. Incompatible objects and invalid method references result in deserialization errors or Java exceptions that cancel the requesting query.

To improve overall performance, increase the cis packet size configuration variable to better facilitate passing serialized objects between servers. Serialized objects are passed between servers with an image datatype, and can vary in size from a few bytes to 2GB.

Datatypes

This section discusses how Component Integration Services deals with various datatype issues.

Unicode support

Adaptive Server contains formal support for the Unicode character set. The datatypes provided are unichar and univarchar. They comprise 2-byte characters expressed in Unicode. Adaptive Server provides conversion functions between Unicode data and all other datatypes, consistent with current handling of char and varchar datatypes. By supporting these datatypes, Component Integration Services is able to present a view of all enterprise character data expressed in Unicode. Character data from mainframes and all other foreign or legacy systems is converted to Unicode when columns of type unichar or univarchar are used to defined columns in proxy tables.

The Component Integration Services features below are affected by these new datatypes:

create table

create table may contain columns described using the new Unicode datatypes. If the table to be created is a proxy table, Component Integration Services forwards the entire command, including the Unicode datatype names (unichar, univarchar) to the remote server where the new table is to be created. If the remote server cannot handle the datatypes, it raises an error.

create existing table

When comparing Adaptive Server column types and lengths with the metadata obtained from a remote server, Unicode datatypes in the proxy table are allowed under the following circumstances:

- The remote server datatype for a column is either unichar or univarchar with equal length (expressed in characters, not bytes).
- The remote server datatype for a given column is char or varchar. In this case, Component Integration Services performs conversions to Unicode on data fetched from the remote server, and conversions from Unicode to the default Adaptive Server character set (UTF8) on data transmitted as part of DML commands (select, insert, delete, update).
- The remote server datatype for a Unicode column is binary or varbinary. The length of the remote server column must be twice the length of the Unicode column. Component Integration Services performs conversions as required when transmitting data to or from the remote server.

No other datatype mapping for Unicode datatypes is allowed when mapping a proxy table to a remote table. Other types result in a type mismatch error. You can convert data from legacy systems into Unicode simply by creating a proxy table that maps a Unicode column to an existing char or varchar column.

create proxy_table

By using create proxy_table, an Adaptive Server user does not have to specify the column list associated with the proxy table. Instead, the column list is derived from column metadata imported from the remote server on which the actual table resides. Unicode columns from the remote server are mapped to Unicode columns in the proxy table only when the remote column is datatype unichar or univarchar.

alter table

alter table allows column types to be modified. With Adaptive Server version 12.5 and later, a column's type can be modified to and from Unicode datatypes. If the command operates on a proxy table, the command is reconstructed and forwarded to the remote server that owns the actual table. If the remote server (or DirectConnect) cannot process the command, an error is expected, and the Adaptive Server command is aborted.

If trace flag 11221 is on, alter table does not get forwarded to a remote server; adding, deleting, or modifying columns is done locally on the proxy table only.

select, *insert*, *update*, and *delete* statements

Unicode datatypes impact the processing of select statements in two ways when proxy tables are involved. The first involves the construction of SQL statements and parameters that are passed to remote servers; the second involves the conversion of data to Unicode when Component Integration Services fetches non-Unicode data.

A DML command involving a proxy table is handled using either TDS language requests or TDS cursor requests when interacting with the remote server. If a select statement contains predicates in the where clause that involve Unicode columns and constants, the Unicode constants musts be handled in one of two ways, depending on whether language or cursor commands are used to process the statement:

1 TDS language – generate clear-text values that can be included in the language text buffer. This involves converting a constant Unicode value to clear text values that can be transmitted as part of a language request.

2 TDS cursor – generate Unicode parameters for CT-Library cursor requests. Parameter values may be Unicode data, requiring Component Integration Services to use parameter types of CS_UNICHAR_TYPE.

Component Integration Services handles an insert command involving a proxy table using either TDS language requests or TDS dynamic requests.

If the insert command can be processed in quickpass mode, then TDS language requests are used. If the command cannot be handled in quickpass mode, the insert is processed using TDS Dynamic requests.

In language requests, the issues are the same as with select — Unicode values must be converted to clear-text form so they can be transmitted with the rest of the SQL statement. In dynamic requests, Unicode data (along with all other data values) is transmitted as parameters to the dynamic command. The receiving server is expected to process parameters of type CS_UNICHAR_TYPE.

The issues with update and delete commands are the same as for select and insert. Unicode values must be converted either to clear-text characters for transmission with the rest of the SQL statement, or they must be converted into parameters of type CS_UNICHAR_TYPE.

Datatype conversions

Datatype conversion can take place whenever the server receives data from a remote source, be it DB2, Adaptive Server, or an Open Server-based application.

Depending on the remote datatype of each column, data is converted from the native datatype on the remote server to a form that the local server supports.

Datatype conversions are made when the create table, alter table and create existing table commands are processed. The datatype conversions are dependent on the remote server's server class. See the create table, alter table, and create existing table commands Chapter 3, "SQL Reference," for tables that illustrate the datatype conversions that take place for each server class when the commands are processed.

text and *image* datatypes

The text datatype is used to store printable character data, the column size of which depends on the logical page size of the Adaptive Server. The image datatype is used to store a number of bytes of hexadecimal-encoded binary data that, again, depends on the logical page size of the Adaptive Server. The maximum length for text and image data is defined by the server class of the remote server to which the column is mapped.

Restrictions on *text* and *image* columns

text and image columns cannot be used:

- As parameters to stored procedures, except when set textptr_parameters is on
- As local variables
- In order by, compute, or group by clauses
- In indexes
- In subqueries
- In where clauses, except with the keyword like
- In joins

Limits of @@*textsize*

select statements return text and image data up to the limit specified in the global variable @@*textsize.* The set textsize command is used to change this limit. The initial value of @@*textsize* is 32K; the maximum value for @@*textsize* is 2147MB.

Odd bytes padded

image values of less than 255 bytes that have an odd number of bytes are padded with a leading zero (an insert of "0xaaabb" becomes "0x0aaabb"). It is an error to insert an image value of more than 255 bytes if the value has an odd number of bytes.

Converting *text* and *image* datatypes

You can explicitly convert text values to char or varchar and image values to binary or varbinary with the convert function, but you are limited to the maximum length of the character and binary datatypes, which depends on the logical page size of the Adaptive Server. If you do not specify the length, the converted value has a default length of 30 bytes. Implicit conversion is not supported.

Pattern matching with *text* data

Use the patindex function to search for the starting position of the first occurrence of a specified pattern in a text, varchar, or char column. The % wildcard character must precede and follow the pattern (except when you are searching for the first or last character).

You can use the like keyword to search for a particular pattern. This example selects each text data value from the blurb column of the texttest table that contains the pattern “Straight Talk%”:

```
select blurb from texttest
 where blurb like "Straight Talk%"
```

Entering *text* and *image* values

The DB-Library™ functions dbwritetext and dbmoretext and the Client-Library function ct_send_data are the most efficient ways to enter text and image values.

When inserting text or image values using the insert command, the maximum length of the data is 16KB.

readtext using bytes

If you use the readtext using bytes command on a text column, and the combination of size and offset result in the transmission of a partial character, errors result.

text and *image* with *bulk copy*

When you use bulk copy to copy text and image values to a remote server, the server must store the values in data pages before sending them to the remote server. Once the values have been issued to the remote server, the data pages are released. Data pages are allocated and released row by row. This is important because:

- The overhead of allocating and releasing data pages impacts performance.
- Data pages are allocated in the database where the table resides, so the database must be large enough to accommodate enough data pages for the largest text and image values that exist for any given row.

Error logging

Processing of text and image data (with remote servers only) can be logged by using trace flag 11207.

text and *image* data with server class *ASEnterprise*

- A pointer in a text or image column is assigned when the column is initialized. Before you can enter text or image data into a column, the column must be initialized. This causes a 2K page to be allocated on the remote or Adaptive Server. To initialize text or image columns, use the update with a NULL or a non-null insert command.
- Before you use writetext to enter text data or readtext to read it, the text column must be initialized. Use update or insert non-null data to initialize the text column, and then use writetext and readtext.
- Using update to replace existing text and image data with NULL reclaims all of the allocated data pages, except the first page, in the remote server.
- writetext, select into, DB-Library functions, or Client-Library functions must be used to enter text or image values that are larger than 16KB.
- readtext is the most efficient way to access text and image data.
- insert select and select into can be used to insert text and image data to proxy tables, but a unique index is required.

text and *image* data with server class *direct_connect (access_server)*

- Specific DirectConnect servers support text and image data to varying degrees. See the DirectConnect documentation for information on text and image support.
- The server uses the length defined in the global variable *@@textsize* for the column length. Before issuing create table, the client application should set *@@textsize* to the required length by invoking set textsize.
- For DirectConnect servers that support text and image datatypes but do not support text pointers, the following restrictions apply:
  - The writetext command is not supported.
  - The readtext command is not supported.
  - Client-Library functions that use text pointers are not supported.
  - DB-Library functions that use text pointers are not supported.
- For DirectConnect servers that support text and image datatypes but do not support text pointers, some additional processing is performed to allow the following functions to be used:
  - patindex

- char_length
- datalength

If text pointers are supported, the server performs these functions by issuing an RPC to the DirectConnect server.

- For DirectConnect servers that do not support text pointers, the server stores data in the sysattributes system table. Data pages are preallocated on a per column per row basis. The column size is determined by @@*textsize*. If this value is not sufficient an error is returned.
- Specific DirectConnect servers may or may not support pattern matching against the text datatype. If a DirectConnect server does not support this pattern matching, the server copies the text value to internal data pages and performs the pattern matching internally. The best performance is seen when pattern matching is performed by the DirectConnect server.
- You must usewritetext, select into, or insert...select to enter text or image values that exceed 450 bytes.
- You can use select into and insert...select to insert text or image values, but the table must have a unique index.

db2 server issues

text and image datatypes for a server of class db2 are not supported. If you need text and image datatypes, you must use a DirectConnect server.

Row-level access control

Component Integration Services users can employ the features of row-level access control because access rules can be bound to columns on proxy tables.

When queries against proxy tables are processed, the access rule is added to the query tree during query normalization, thus making its existence transparent to downstream query processing. Therefore, Component Integration Services users can forward additional predicates to remote servers to restrict the amount of data transferred, according to the expression defined by the access rule.

Component Integration Services can function as a row-level access control hub to the entire enterprise through the use of access rules bound to columns on proxy tables.

select into

An Adaptive Server with a 2K page configuration cannot be automatically upgraded to a 4K, 8K, or 16K configuration, nor can the dump and load backup facilities provide this upgrade through the Backup Server. Instead, data (and metadata) must be transferred, or migrated, from one server to another. To accomplish this task, use the ddlgen feature of Sybase Central™ for Adaptive Server. Adaptive Server fully supports DDL, and enables the transfer of server schema and configuration data from one server to another. In addition, a migration tool serves as a driver for the data transfer.

Once metadata has been transferred from one server to another, the migration tool is used to coordinate the data transfer. Create proxy tables at the source server for each table on the target server, and then execute a select into statement to effect the transfer from the source table, which is on local disk, to the target, which is a proxy table referencing the target server.

To facilitate this process, three significant changes to the manner in which the select into command is executed have been made:

- Parallel data transfer – if the source table is partitioned and local, then the data transfer is achieved through worker threads, one per partition.
- Allow bulk transfer to existing tables – since the remote tables are already in place, enable data transfer via select into even if the target table already exists. Using this syntax:

  ```
  select <column_list> into existing table <table_name> from ...
  ```

  existing table allows the command to operate on tables that already exitst. A check is made to ensure that the datatypes of the <column_list> match, in type and length, the data types of the target table.
- Enable bulk insert arrays – when performing a bulk transfer of data from one Adaptive Server to another, Component Integration Services buffers rows internally, and asks the Open Client bulk library to transfer them as a block. The size of the array is controlled by the configuration parameter cis bulk insert array size. The default is 50 rows, and the property is dynamic, allowing it to be changed without server restart.

To achieve performance levels required, the bulk interface currently in use by Component Integration Services has been modified to support bulk insert array binding. This allows Component Integration Services to buffer a specified number of rows in local memory, and transfer them all with a single bulk transfer command. An Adaptive Server configuration property has been implemented to specify the size of the bulk insert array:

```
sp_configure “cis bulk insert array size”, no
```

where *n* is an integer value greater than 0. The default is 50.

Set commands can override this behavior:

- set bulk array size n – similar to the configuration property cis bulk insert array size. set bulk array size n applies to the current session only. New sessions inherit the value specified in the confiugration file, but can override using this set command. *n* is an integer value greater than 0.
- set bulk batch size n – similar to the configuration property cis bulk insert batch size. set bulk batch size n applies to the current session only. New sessions inherit the value specified in the configuration file, but can be overriden using this set command. *n* is an integer value greater than 0.

select into

There is syntax that allows the specification of proxy table attributes when you want to create a proxy table through the select into statement. The syntax is:

```
select <column_list> into <newtablename>
[[external <type>] at "location_string"
[column delimiter "<string>"]]
from <source_table(s)  [where...]
```

Using this syntax, you do not need to use sp_addobjectdef to specify the location of a proxy table. You can specify all attributes of a proxy table with this new select into syntax.

The new table created is a proxy table if the clause at *location_string* is provided. The external <type> qualifier is used to indicate that the new proxy table is mapped to a remote table, a directory, or a file. The column delimiter is valid only if the new type is file, and only if the delimiter is used to specify the string used to delimit separate fields within the file. The default is a tab character.

There is also syntax that enables select into when tables already exist:

```
select <column_list> into existing table <newtablename>
    from <source_table(s) [where...]
```

A check is made to ensure that the datatypes of the *column_list* match, in type and length, the datatypes of the target table. If the source columns can be NULL, the corresponding column in the target table must also allow NULL. Additional restrictions on the use of existing table syntax include:

- The statement containing existing table cannot be used in a procedure, trigger, or view.
- The target table (the "existing table") must be a proxy table; you cannot use this syntax for local tables. Use insert select instead.

Execute immediate

The execute immediate feature is fully supported when using Component Integration Services.

Configuration and tuning

This section provides information about configuration, tuning, trace flags, backup and recovery, and security issues.

The System Administrator or Database Owner may elect to use the server to optimize performance or to allow use by a required number of clients. Configuration choices might involve being able to review total numbers of reads and writes for a given SQL command.

Once an application is up and running, the System Administrator should monitor performance and may choose to customize and fine-tune the system. The server provides tools for these purposes. This section explains:

- Changing system parameters with the sp_configure procedure
- Using update statistics to ensure that Component Integration Services makes the best use of existing indexes
- Monitoring server activity with the dbcc command
- Setting trace flags
- Executing ddlgen and related backup and recovery issues
- Determining database size requirements

Using *sp_configure*

The configuration parameters in sp_configure control resource allocation and performance. The System Administrator can reset these configuration parameters to tune performance and redefine storage allocation. In the absence of intervention by the System Administrator, the server supplies default values for all the parameters.

The procedure for resetting configuration parameters is:

- Execute sp_configure, which updates the values field of the system table master..sysconfigures.
- Restart the server if you have reset any of the static configuration parameters. The parameters listed below are dynamic; all others are static:
    - cis rpc handling
    - cis cursor rows
    - cis bulk insert batch size
    - cis bulk insert array size
    - cis packet size

sysconfigures table

The master..sysconfigures system table stores all configuration options. It contains columns identifying the minimum and maximum values possible for each configuration parameter, as well as the configured value and run value for each parameter.

The status column in sysconfigures cannot be updated by the user. Status 1 means dynamic, indicating that new values for these configuration parameters take effect immediately. The rest of the configuration parameters (those with status 0) take effect only after the reconfigure command has been issued and the server restarted.

You can display the configuration parameters currently in use (run values) by executing sp_configure without giving it any parameters.

Changing the configuration parameters

sp_configure displays all the configuration values when it is used without an argument. When used with an option name and a value, the server resets the configuration value of that option in the system tables.

See the *System Administration Guide* for a complete discussion of sp_configure with syntax options.

To see the Component Integration Services options, enter:

```
sp_configure "Component Integration Services"
```

To change the current value of a configuration parameter, execute sp_configure as follows:

```
sp_configure "parameter", value
```

Component Integration Services configuration parameters

The following configuration parameters are unique to Component Integration Services:

- enable cis
- enable file access
- enable full-text search
- max cis remote connections
- cis bulk insert batch size
- cis bulk insert array size
- cis cursor rows
- cis packet size
- cis rpc handling

enable cis

Use this parameter with sp_configure to enable Component Integration Services as follows:

1 Log in to Adaptive Server as the System Administrator and issue the following command:

```
sp_configure "enable cis", 1
```

2 Restart Adaptive Server.

Issuing sp_configure "enable cis", 0 disables Component Integration Services after restarting the server.

enable file access

This configuration parameter enables access through proxy tables to eXternal File System. Requires a license for ASE_XFS.

enable full-text search	This configuration parameter enables Enhanced Full-Text Search services. Requires a license for ASE_EFTS.
max cis remote connections	This configuration property is no longer used.
cis bulk insert batch size	This configuration parameter determines how many rows from the source tables are to be bulk copied into the target table as a single batch using select into, when the target table resides in an Adaptive Server or in a DirectConnect server that supports a bulk copy interface. If left at zero (the default), all rows are copied as a single batch. Otherwise, after the count of rows specified by this parameter has been copied to the target table, Component Integration Services issues a bulk commit to the target server, causing the batch to be committed. If a normal client-generated bulk copy operation (such as that produced by the bcp utility) is received, the client is expected to control the size of the bulk batch, and Component Integration Services ignores the value of this configuration parameter.
cis bulk insert array size	When performing a bulk transfer of data from one Adaptive Server to another, Component Integration Services buffers rows internally, and asks the Open Client bulk library to transfer them as a block. The size of the array is controlled by the configuration parameter cis bulk insert array size. The default is 50 rows, and the property is dynamic, allowing it to be changed without server reboot.
cis cursor rows	This configuration parameter allows users to specify the cursor row count for cursor open and cursor fetch operations. Increasing this value means more rows are fetched in one operation. This increases speed but requires more memory. The default is 50.
cis packet size	This configuration parameter allows you to specify the size of Tabular Data Stream™ (TDS) packets that are exchanged between Component Integration Services and a remote server when connection is initiated. The default packet size on most systems is 512 bytes, which is adequate for most applications. However, larger packet sizes may result in significantly improved query performance, especially when text and image or bulk data is involved. If a packet size larger than the default is specified, then the target server must be configured to allow variable-length packet sizes. Adaptive Server configuration parameters of interest in this case are: • additional netmem

- maximum network packet size

See the *System Administration Guide* for a complete explanation of these configuration parameters.

cis rpc handling

This global configuration parameter determines whether Component Integration Services handles outbound RPC requests by default. When this is enabled using sp_configure "cis rpc handling" 1, all outbound RPCs are handled by Component Integration Services. When you use sp_configure "cis rpc handling" 0, the Adaptive Server site handler is used. The thread cannot override it with set cis_rpc_handling on. If the global property is disabled, a thread can enable or disable the capability, as required.

For more information on using the Adaptive Server site handler versus using Component Integration Services to handle outbound RPCs, see "RPC handling and Component Integration Services" on page 58.

Global variables for status

The following global variables have been added for Component Integration Services users:

- @@cis_rpc_handling
- @@transactional_rpc
- @@textptr_parameters
- @@stringsize
- @@bulkbatchsize – contains the value of the current cis bulk insert batch size configured via sp_configure, or set through the set bulk batch size command.
- @@bulkarraysize – contains the value of the current cis bulk insert array size configured via sp_configure or set through the set bulk array size command.

These global variables show the current status of the corresponding configuration parameters. For instance, to see the status of cis_rpc_handling, issue the following command:

```
select @@cis_rpc_handling
```

This returns either 0 (off) or 1 (on).

CHAPTER 3

SQL Reference

This chapter provides reference material on the server classes supported by Component Integration Services.

Each server class has a set of unique characteristics that System Administrators and programmers need to know about in order to configure the server for remote data access. These properties are:

- Types of servers that each server class supports
- Datatype conversions specific to the server class
- Restrictions on Transact-SQL statements that apply to the server class

dbcc commands

All dbcc commands used by Component Integration Services are available with a single dbcc entry point.

The syntax for dbcc cis is:

```
dbcc cis ("subcommand"[, vararg1, vararg2...])
```

If Component Integration Services is not configured or loaded, the command results in a runtime error.

The use of the dbcc cis command is unrestricted.

dbcc options

The following dbcc options are unique to Component Integration Services.

remcon

remcon displays a list of all remote connections made by all Component Integration Services clients. It takes no arguments.

srvdes

srvdes returns a formatted list of all in-memory SRVDES structures, if no argument is provided. If an argument is provided, this command syncs the in-memory version of a SRVDES with information found in sysservers. The command takes an optional argument as follows:

```
srvdes, [ srvid ]
```

showcaps

showcaps displays a list of all capabilities for servername by capability name, ID, and value as follows:

```
showcaps, servername
```

Example:

dbcc cis("showcaps", "servername")

Trace flags

The dbcc traceon option allows the System Administrator to turn on trace flags within Component Integration Services. Trace flags enable the logging of certain events when they occur within Component Integration Services. Each trace flag is uniquely identified by a number. Some are global to Component Integration Services, while others are spid-based and affect only the user who enabled the trace flag. dbcc traceoff turns off trace flags.

The syntax is:

dbcc traceon (*traceflag* [, *traceflag*...])

Trace flags and their meanings are shown in Table 3-1:

Table 3-1: Component Integration Services trace flags

Trace flag	Description
11201	Logs client connect events, disconnect events, and attention events. (global)
11202	Logs client language, cursor declare, dynamic prepare, and dynamic execute-immediate text. (global)
11203	Logs client RPC events. (global)
11204	Logs all messages routed to client. (global)
11205	Logs all interaction with remote server. (global)

Trace flag	Description
11206	Logs file/directory processing steps. (global)
11207	Logs text and image processing. (global)
11208	Prevents the create index and drop table statements from being transmitted to a remote server. sysindexes is updated anyway. (spid)
11209	Instructs update statistics to obtain only row counts rather than complete distribution statistics, from a remote table. (spid)
11210	Disables the Component Integration Services enhanced remote query optimization. (spid)
11211	Not used.
11212	Prevents escape on underscores ("_") in table names. (spid)
11213	Prevents generation of column and table constraints. (spid)
11214	Disables Component Integration Services recovery at start-up. (global)
11215	Sets enhanced remote optimization for servers of class db2. (global)
11216	Disables enhanced remote optimization. (spid)
11217	Disables enhanced remote optimization. (global)
11220	Disables constraint checking of remote tables on the local server. This avoids duplicate checking. Setting this trace flag on ensures that queries are not rejected by the quickpass mode because of constraints. (spid)
11221	Disables alter table commands to the remote server when ON. This allows users to modify *type*, *length*, and *nullability* of columns in a local table without changing columns in the remote table. Use trace flag 11221 with caution. It may lead to tables that are "out of sync." (spid)
11223	Disables proxy table index creation during create existing table or create proxy_table command execution. If this flag is set on, no index metadata is imported from the remote site referenced by the proxy table, and no indexes for the proxy table are created. This trace flag should be used with care and turned off when no longer necessary. (global)

Functions

This section defines the compatibility of the Component Integration Services server classes with the built-in Adaptive Server functions.

Support for functions within Component Integration Services

When a SQL statement such as a select, insert, delete, or update contains a built-in function, Component Integration Services has to determine whether or not the function can be forwarded to the remote server, or if it must be evaluated within the local server using remote data.

Functions are only sent to a remote server if the statement containing them can be handled by quickpass mode.

In the tables shown below, support for function by server class is indicated by a 'Y'; 'N' indicates no support is provided, and 'C' indicates support for it is determined by capabilities of the underlying DBMS (often the case for DirectConnects).

Aggregate functions

The aggregate functions generate summary values that appear as new columns in the query results.

Table 3-2: Server class support for aggregate functions

Function	ASE	ASA	ASIQ	dir_con	db2
avg	Y	Y	Y	C	Y
count	Y	Y	Y	C	Y
max	Y	Y	Y	C	Y
min	Y	Y	Y	C	Y
sum	Y	Y	Y	C	Y

Datatype conversion functions

Datatype conversion functions change expressions from one datatype to another and specify new display formats for date/time information.

Table 3-3: Server class support for datatype conversion functions

Function	ASE	ASA	ASIQ	dir_con	db2
convert()	Y	Y	Y	C	N
inttohex()	Y	Y	N	C	N
hextoint()	Y	Y	N	C	N

Date functions

The date functions manipulate values of the datatype datetime or smalldatetime. The getdate() function is always expanded by the local server; the presence of this builtin function does not cause a query to be eliminated from quickpass mode optimizations, however.

Table 3-4: Server class support for date functions

Function	ASE	ASA	ASIQ	dir_con	db2
dateadd	Y	Y	Y	C	N
datediff	Y	Y	Y	C	N
datename	Y	Y	N	C	N
datepart	Y	Y	Y	C	N

Mathematical functions

Mathematical functions return values commonly needed for operations on mathematical data. Mathematical function names are not keywords.

Each function also accepts arguments that can be implicitly converted to the specified type. For example, functions that accept approximate numeric types also accept integer types. Adaptive Server automatically converts the argument to the desired type

Table 3-5: Server class support for mathematical functions

Function	ASE	ASA	ASIQ	dir_con	db2
abs	Y	Y	Y	C	N
acos	Y	Y	N	C	N
asin	Y	Y	N	C	N
atan	Y	Y	N	C	N
atn2	Y	Y	N	C	N
ceiling	Y	Y	Y	C	N
cos	Y	Y	N	C	N
cot	Y	Y	N	C	N
degrees	Y	Y	N	C	N
exp	Y	Y	N	C	N
floor	Y	Y	Y	C	N
log	Y	Y	N	C	N
log10	Y	Y	N	C	N
pi	Y	Y	N	C	N
power	Y	Y	N	C	N
radians	Y	Y	N	C	N
rand	Y	Y	Y	C	N
round	Y	Y	N	C	N
sign	Y	Y	N	C	N
sin	Y	Y	N	C	N
sqrt	Y	Y	Y	C	N
tan	Y	Y	N	C	N

Security functions

Security functions return security-related information.

Table 3-6: Server class support for security functions

Function	ASE	ASA	ASIQ	dir_con	db2
ic_sec_service_on()	N	N	N	N	N
show_sec_services)	N	N	N	N	N

String functions

String function operate on binary data, character strings, and expressions. The string functions are:

Table 3-7: Server class support for string functions

Function	ASE	ASA	ASIQ	dir_con	db2
ascii	Y	Y	N	C	N
char	Y	Y	N	C	N
charindex	Y	Y	N	C	N
char_lengt	Y	Y	N	C	N
difference	Y	Y	Y	C	N
lower	Y	Y	Y	C	N
ltrim	Y	Y	Y	C	N
patindex	N	N	N	N	N
replicate	Y	Y	N	C	N
reverse	Y	N	N	Y	N
right	Y	Y	Y	C	N
rtrim	Y	Y	Y	C	N
soundex	Y	N	Y	C	N
space	Y	Y	N	C	N
str	Y	Y	N	C	N
stuff	Y	Y	N	C	N
substring	Y	Y	Y	C	N
upper	Y	Y	Y	C	N

System functions

System functions return special information from the database.

Table 3-8: Server class support for system functions

Function	ASE	ASA	ASIQ	dir_con	db2
col_length	Y	Y	N	C	N
col_name	Y	Y	N	C	N
curunreservedpgs	N	N	N	N	N
data_pgs	N	N	N	N	N
datalength	Y	Y	N	C	N
db_id	N	N	N	N	N
db_name	N	N	N	N	N
host_id	N	N	N	N	N
host_name	N	N	N	N	N
index_col	N	N	N	N	N
isnull	Y	Y	N	N	N
lct_admin	N	N	N	N	N
mut_excl_roles	N	N	N	N	N
object_id	N	N	N	N	N
object_name	N	N	N	N	N
proc_role	N	N	N	N	N
ptn_data_pgs	N	N	N	N	N
reserved_pgs	N	N	N	N	N
role_contain	N	N	N	N	N
role_id	N	N	N	N	N
role_name	N	N	N	N	N
rowcnt	N	N	N	N	N
show_role	N	N	N	N	N
suser_id	N	Y	Y	N	N
suser_name	N	Y	Y	N	N
tsequal	Y	Y	N	N	N
used_pgs	N	N	N	N	N
user	Y	Y	Y	N	N
user_id	Y	Y	Y	N	N
user_name	Y	Y	Y	N	N
valid_name	N	N	N	N	N
valid_user	N	N	N	N	N

Text and image functions

Text and image functions operate on text and image data.

Table 3-9: Server class support for text and image functions

Function	ASE	ASA	ASIQ	dir_con	db2
textptr()	Y	Y	N	C	N
textvalid)	Y	Y	N	C	N

Transact-SQL commands

The following pages discuss, in alphabetical order, Transact-SQL commands that directly or indirectly affect external tables, and, as a result, Component Integration Services. For each command, a description of its effect on Component Integration Services, and the manner in which Component Integration Services processes the command is provided. For a complete description of each command, see the *Reference Manual*.

If Component Integration Services does not pass all of a command's syntax to a remote server (such as all clauses of a select statement), the syntax that is passed along is described for each server class.

Each command has several sections that describe it:

- Description – contains a brief description of the command.
- Syntax – contains a description of the full Transact-SQL syntax of the command.
- Usage – contains a general, server class-independent description of handling by Component Integration Services.
- Server class ASEnterprise – contains a description of handling specific to server class ASEnterprise. This includes syntax that is forwarded to a remote server of class ASEnterprise.
- Server class ASAnywhere – contains a description of handling specific to server class ASAnywhere. This includes syntax that is forwarded to a remote server of class ASAnywhere.
- Server class ASIQ – contains a description of handling specific to server class ASIQ. This includes syntax that is forwarded to a remote server of class ASIQ.

- Server class direct_connect – contains a description of handling specific to server class direct_connect (access_server). This includes syntax that is forwarded to a remote server of class direct_connect (access_server). In this release, all comments that apply to server class direct_connect, also apply to server class sds.
- Server class db2 – contains a description of handling specific to server class db2. This includes syntax that is forwarded to a remote server of class db2.

alter database

Description

Increases the amount of space allocated to a database. Synchronizes proxy table metadata with tables at remote location.

Syntax

alter database *database_name*
[on {default | *database_device* } [= size]
[, database_device [= size]]...]
[log on { default | *database_device* } [= *size*]
[, *database_device* [= *size*]]...]
[with override]
[for load]
[for proxy_update]]

Usage

- If a database has been created with the optional clause with default_location = pathname, then alter database, with the for proxy_update clause, resynchronizes the proxy tables in the named database with tables and views found in the path name to the remote location.
- The default location may also have been specified with sp_defaultloc. The for proxy_update clause of alter database works the same way in this case.
- This is a convenient, one-step procedure for keeping the proxy table definition in sync with the definition of actual tables and views in a remote database.
- If for proxy_update is specified with no size or device name, then the size is not altered; only proxy table synchronization is performed.
- In some cases, a database may not be large enough to contain all proxy table definitions; therefore, it may be necessary to change the size as well when the for proxy_update clause is used.

- When for proxy_update is used, the names of remote tables and views are obtained from the server specified in the default location for the database (master.dbo.sysdatabases.default_loc) using the RPC named sp_tables. For each user table and view, column attributes are then obtained, using the RPC named sp_columns. Once all metadata has been obtained for a table (or view), an internal command is executed which is equivalent to create existing table, causing the proxy table to be created within the named database.
- If the proxy table already exists, it is automatically dropped before the internal create existing table command is executed.
- After the proxy table is created, index metadata is obtained from the remote location so that indexes on the proxy table can also be created. Index metadata is obtained from the remote server using the RPC sp_statistics.
- This command behaves the same way for all server classes; interaction with the remote server associated with the database default location is limited to the RPCs sp_tables, sp_columns and sp_statistics (to import index information).

Note alter database for proxy update drops all proxy tables in the proxy database.

See also

create database in the *Reference Manual*.

alter table

Description

Adds, changes, or drops columns; adds, changes, or drops constraints; partitions or unpartitions an existing table; changes the locking scheme for an existing table; specifies ascending or descending index order when alter table is used to create referential integrity constraints that are based on indexes; specifies the ratio of filled pages to empty pages, to reduce storage fragmentation.

Syntax

```
alter table [database.[owner].]table_name
 {add column_name datatype
    [default {constant_expression | user | null}]
     {identity | null | not null}
     | [[constraint constraint_name]
       {{unique | primary key}
```

```
        [clustered | nonclustered] [asc | desc]
        [with { { fillfactor = pct
                | max_rows_per_page = num_rows }
                , reservepagegap = num_pages }]
        [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 [asc | desc]
            [, column_name [asc | desc]...])
      [with { { fillfactor = pct
            | max_rows_per_page = num_rows}
            , reservepagegap = num_pages}]
      [on segment_name]
  | foreign key (column_name [{, column_name}...])
      references [[database.]owner.]ref_table
        [(ref_column [{, ref_column}...])]
  | check (search_condition)}

| drop {[column_name [, column_name]] |
        [constraint constraint_name]}

| modify column_name {[data_type] [null] |
        [not null]] [, column_name]

| replace column_name
   default {constant_expression | user | null}

| partition number_of_partitions

| unpartition

| lock {allpages | datarows | datapages } }

| with exp_row_size = num_bytes
```

Usage

- Component Integration Services processes alter table when the table on which it operates has been created as a proxy table. Component Integration Services forwards the request (or part of it) to the server that owns the actual object.
- When Component Integration Services forwards alter table to a remote server, it is assumed that the column names on the proxy table and on the remote server are the same.

- The only portions of the alter table command that are forwarded to a remote server are add, modify, drop column, partition, and unpartition. The rest of the syntax is processed internally, and not forwarded to a remote server. The only exception to this is the lock clause, and then only for ASEnterprise-class servers.

Server class *ASEnterprise*

Component Integration Services forwards the following syntax to a server configured as class ASEnterprise:

```
alter table [database.[owner].]table_name
    {add column_name datatype [{identity | null}]
        {[, next_column]}...}
    | [drop column_name [, column_name]}
    | modify column_name [data_type] [NULL] |
        [not null]] [, column_name]}
```

- When a user adds a column with the alter table command, Component Integration Services passes the datatype of each column to the remote server without type name conversions.
- For ASEnterprise class servers only, the lock clause is also forwarded, if contained in the original query, if the version of Adaptive Server is 11.9.2 or later.

Server class *ASAnywhere*

Handling of alter table by servers in this class is the same as for ASEnterprise servers.

Server class *ASIQ*

- Handling of alter tableby servers in this class is the same as for ASEnterprise servers.
- text and image datatypes are not supported by server class ASIQ. If text and image datatypes are used, Component Integration Services raises Error 11205:

```
Datatype <typename> is unsupported for server
<servername>.
```

Server class *direct_connect*

- Component Integration Services forwards the following syntax to a remote server configured as class direct_connect:

```
alter table [database.[owner].]table_name
 add column_name datatype [{identity | null}]
        {[, next_column]}...
```

- Although Component Integration Services requests a capabilities response from a server with class direct_connect, support for alter table is not optional. Component Integration Services forwards alter table to the remote server regardless of the capabilities response.
- The behavior of the server with class direct_connect is database dependent. The Transact-SQL syntax is forwarded, and errors may or may not be raised, depending on the ability of the remote database to handle this syntax.
- If the syntax capability of the remote server indicates Sybase Transact-SQL, Adaptive Server datatypes are sent to the remote server. If the syntax capability indicates DB2 SQL, DB2 datatypes are sent. The mapping for these datatypes is shown in Table 3-10:

Table 3-10: DirectConnect datatype conversions for alter table

Adaptive Server datatype	DirectConnect default datatype	DirectConnect DB2 syntax mode datatype
binary(*n*)	binary(*n*)	char(*n*) for bit data
bit	bit	char(1)
char	char	char
datetime	datetime	timestamp
decimal(*p*, *s*)	decimal(*p*, *s*)	decimal(*p*, *s*)
float	float	float
image	image	varchar(*n*) for bit data; the value of *n* is determined by the global variable @@textsize
int	int	int
money	money	float
numeric(*p*, *s*)	numeric(*p*, *s*)	decimal(*p*, *s*)
nchar(*n*)	nchar(*n*)	graphic(*n*)
nvarchar(*n*)	nvarchar(*n*)	vargraphic(*n*)
real	real	real
smalldatetime	smalldatetime	timestamp
smallint	smallint	smallint
smallmoney	smallmoney	float
timestamp	timestamp	varbinary(8)
tinyint	tinyint	smallint

Adaptive Server datatype	DirectConnect default datatype	DirectConnect DB2 syntax mode datatype
text	text	varchar(*n*); the value of *n* is determined by the global variable @@textsize
unichar	unichar	varchar(*n*) for bit data
univarchar	univarchar	varchar(*n*) for bit data
varbinary(*n*)	varbinary(*n*)	varchar(*n*) for bit data
varchar(*n*)	varchar(*n*)	varchar(*n*)

Server class *db2*

- Component Integration Services forwards the following syntax to a remote server configured as class db2:

```
alter table [database.[owner].]table_name
 add column_name datatype [null]
        {[, next_column]}...
```

- text and image datatypes are not supported by server class db2. If text and image datatypes are used, Component Integration Services raises error 11205:

```
Datatype <typename> is unsupported for server
<servername>
```

The datatype specification contains db2 datatypes that are mapped from Adaptive Server datatypes. The datatype conversions are shown in Table 3-11:

Table 3-11: DB2 datatype conversions for alter table

Adaptive Server datatype	DB2 datatype
binary(n)	char(n) for bit data, where *n* <= 254
bit	char(1)
char(n)	char(n), where *n* <= 254
datetime	timestamp
decimal*(p, s)*	decimal*(p, s)*
float	float
image	Not supported
int	int
money	float
nchar	char(n)
nvarchar	varchar(n)

Adaptive Server datatype	DB2 datatype
numeric*(p, s)*	decimal*(p, s)*
real	real
smalldatetime	timestamp
smallint	smallint
smallmoney	float
tinyint	smallint
text	Not supported
unichar	varchar(n) for bit data
univarchar	varchar(n) for bit data
varbinary(n)	varchar(n) for bit data, where *n* <=254
varchar(n)	varchar(n), where *n* <= 254

See also — alter table in the *Reference Manual*

begin transaction

Description — Marks the starting point of a user-defined transaction.

Syntax — begin tran[saction] [*transaction_name*]

Usage

- When a begin transaction command is received by Adaptive Server, it is not known which remote servers are involved in the transaction. When subsequent statements require Component Integration Services involvement, Component Integration Services determines whether a transaction is active or not, and if so, registers any remote servers as participants in the transaction during the execution of the statement. The first time Component Integration Services registers a new participant, ASTC ensures that a transaction is begun on the participant, unless that participant has no-DTM capability.
- For all server classes, when Adaptive Server receives a begin transaction command, an internal state is set that marks the beginning of a transaction. At this point, Component Integration Services is not involved, and the command is not immediately forwarded to remote locations.
- *transaction_name* is not currently used by Component Integration Services.

See also — begin transaction in the *Reference Manual*

case

Description

Supports conditional SQL expressions; can be used anywhere a value expression can be used.

Syntax

```
case
        when search_condition then expression
        [when search_condition then expression]...
        [else expression]
end
```

Usage

- case expression simplifies standard SQL expressions by allowing you to express a search condition using a when...then construct instead of an if statement.
- case expressions can be used anywhere an expression can be used in SQL.
- If your query produces a variety of datatypes, the datatype of a case expression result is determined by datatype hierarchy. If you specify two datatypes that Adaptive Server cannot implicitly convert (for example, char and int), the query fails.

Server class *ASEnterprise*

The ability to handle case expressions is set for Adaptive Server 11.5 and later. The presence of a case expression in the original query syntax does not cause the query optimizer to reject quickpass mode.

Server class *ASAnywhere*

The ability to handle case expressions is set for ASA 6.0, ASIQ 12.0, and all later versions. The presence of a case expression in the original query syntax will not cause the query optimizer to reject quickpass mode.

Server class *ASIQ*

The ability to handle case expressions is not set for servers in this class. When a SQL statement containing a case expression is optimized, the presence of the case expression causes the Component Integration Services quickpass optimization to reject the statement. When this happens, the case expression must be evaluated by the local Adaptive Server after retrieving data from the remote server.

Server class *direct_connect*

The ability to handle case expressions is determined by the result set from the RPC sp_capabilities. If direct_connect indicates that it can handle case expressions, then Component Integration Services forwards them to the direct_connect when quickpass mode is used to handle the query.

Server class *db2*

The ability to handle case expressions is, by default, not set for servers in this class. When a SQL statement containing a case expression is optimized, the presence of the case expression causes Component Integration Services quickpass optimization to reject the statement. When this happens, the case expression must be evaluated by the local Adaptive Server after retrieving data from the remote server.

If trace flag 11215 is turned on, the default capabilities for server class db2 are modified to enable more capabilities, and case expressions are enabled by default. DB2 does recognize case expression syntax. Also, the trace flag must be turned on before Component Integration Services makes its first connection to the remote server, or else the capabilities set by the first connection remain in effect until the server is restarted.

close

Description

Deactivates a cursor.

Syntax

close *cursor_name*

Usage

- If the cursor specified by *cursor_name* contains references to proxy tables, Adaptive Server notifies Component Integration Services to close and deallocate its remote cursors for those tables.
- Component Integration Services uses Client-Library to manage cursor operations to a remote server. When Component Integration Services receives a close command, it uses the following Client-Library functions to interact with the remote server:

  ct_cursor(command, CS_CURSOR_CLOSE, NULL, CS_UNUSED, NULL, CS_UNUSED, CS_UNUSED)

  ct_cursor(command, CS_CURSOR_DEALLOC, NULL, CS_UNUSED, NULL, CS_UNUSED, CS_UNUSED)

- If the cursor contains references to more than one proxy table, Component Integration Services must close a remote cursor for each server represented by the proxy tables.

See also

deallocate cursor, declare cursor, fetch, open in this chapter

close in the *Reference Manual*

commit transaction

Description — Marks the successful ending point of a user-defined transaction.

Syntax — commit [tran[saction] | work] [*transaction_name*]

Usage

- When Adaptive Server receives the commit transaction command, it notifies Component Integration Services, and Component Integration Services attempts to commit work associated with remote servers involved in the current transaction.
- Before committing work on remote servers, each server is sent a prepare command, according to its DTM capabilities (DTM-enabled, pre-DTM, no-DTM). If all participants respond with no error, a commit is sent to all participants, again according to their DTM.
- Multiple remote servers can be involved in a single transaction, each with its own unit of work, which is associated with the Adaptive Server unit of work.
- Remote work is committed before local work. If the remote servers do not respond, or respond with errors, the transaction is aborted, including any local work.
- Work performed by transactional RPCs must be part of an explicit transaction.
- *transaction_name* is currently not used by Component Integration Services.

Object type = file

- files are not part of transaction management. They cannot be committed or rolled back.

Object type = directory

- directories are not part of transaction management. They cannot be committed or rolled back.

See also — commit in the *Reference Manual*

connect to...disconnect

Description — Connects to the specified server to establish a passthrough-mode connection; takes the connection out of passthrough mode.

Syntax

```
connect to server_name
disconnect
```

Usage

- connect to specifies the server to which a passthrough connection is required. Passthrough mode enables you to perform native operations on a remote server.
- server_name must be the name of a server in the sysservers table, with its server class and network name defined.
- When establishing a connection to *server_name* on behalf of the user, Component Integration Services uses one of the following identifiers:
  - A remote login alias described in sysattributes, if present
  - The user's name and password

  In either case, if the connection cannot be made to the specified server, Adaptive Server returns an error message.
- For more information about adding remote servers, see sp_addserver.
- After making a passthrough connection, Component Integration Services bypasses the Transact-SQL parser and compiler when subsequent language text is received. It passes statements directly to the specified server, and converts the results into a form that can be recognized by the Open Client interface and returned to the client program.
- To take the connection created by the connect to command out of passthrough mode, use disconnect. You can use this command only after the connection has been made using connect to.
- disconnect does not actually cause the termination of the connection to the remote server; instead, it simply takes the connection out of passthrough mode, leaving the connection available for subsequent DDL or DML statements that are processed normally by the Adaptive Server query processor.
- disconnect can be abbreviated to disc.
- disconnect returns an error unless connect to has already been previously issued and the server is connected to a remote server.

Server class ASEnterprise

When disconnect is issued, Component Integration Services forwards disconnect to the remote server, to take it out of passthrough mode. If not in passthrough mode, syntax errors may occur, but they are ignored by Component Integration Services and not forwarded to the client.

Server class *ASAnywhere*

No interaction occurs with ASAnywhere when connect or disconnect are issued.

Server class *ASIQ*

No interaction occurs with ASIQ when connect or disconnect are issued.

Server class direct_connect

When connect is issued using a server in class direct_connect, the direct_connect is sent an RPC:

```
sp_thread_props “passthru mode”, 1
```

When disconnect is issued, and the server for which a passthrough-mode connection has been established is a direct_connect, the direct_connect is sent an RPC:

```
sp_thread_props “passthru mode”, 0
```

Server class *db2*

No interaction occurs with db2 when connect or disconnect are issued.

See also commit in the *Reference Manual*

create database

Description Creates a new database.

Syntax

```
create database database_name
[on {default | database_device} [= size]
    [, database_device [= size]]...]
[log on database_device [= size]
    [, database_device [= size]]...]
[with override]
[with default_location = “pathname”]
[for proxy_update]
[for load]
```

Usage

- This command creates a new database within Adapter Server. The syntax with default_location = pathname and for proxy_update allows automatic creation of proxy tables representing tables and views found in a remote location.
- If the clause with default_location = pathname is used, the *pathname* is stored in master.dbo.sysdatabases.default_loc, and serves the same purpose as the default location added using sp_defaultloc.

- If the clause for proxy_update is used, the with default_location = pathname clause must also be used. This clause indicates that the database is to be a proxy database, and all tables created in it will become proxy tables, referencing objects contained at the default location.
- When a database is created as a proxy database, and no device or size specification is included in the syntax, the default size is calculated based on the number of proxy tables that it is expected to contain. The formula for calculating the number of 2K, 4K, 8K, and 16K pages for the database is as follows:

  1MB for every 100 proxy tables plus size of model database plus 1MB
- The new database is placed on the default device, if no device name is specified.
- After the database is created, but before the command is complete, the presence of the for proxy_update clause instructs Component Integration Services to create a proxy table in the new database for each remote table or view. When the create database command is finished, the newly created database is populated with proxy tables representing all user tables and views found at the remote location.

See also

alter database in the *Reference Manual*

create existing table

Description

Creates a new proxy table representing an existing object in a remote server.

Syntax

```
create existing 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]
[external {table | procedure}]
[at "pathname"]
```

Usage

- Adaptive Server processes create existing table as if the table being created is a new local table.
- After creating the local table, Adaptive Server passes create existing table to Component Integration Services, with the external location for the existing remote object.

  Component Integration Services verifies that the table exists by issuing the sp_tables RPC to the remote server that owns the existing object.
- Component Integration Services verifies the column list by sending the sp_columns RPC to the remote server. Column names, datatypes, lengths, identity property, and null properties are checked for the following:
  - Datatypes in create existing table must match or be convertible to the datatypes of the column on the remote location. For example, a local column datatype might be defined as money, while the remote column datatype might be numeric. This is a legal conversion, therefore, no errors are reported.
  - Each column's null property is checked. If the local column's null property is not identical to the remote column's null property, a warning message is issued, but the command is not aborted.
  - Each column's length is checked. If the length of char, varchar, binary, varbinary, decimal, and numeric columns do not match, a warning message is issued, but the command is not aborted.
- The column names used in the syntax must match those at the remote location.
- The proxy table need not contain the exact number of columns found in the remote table. However, all columns referenced by the proxy table must exist in the remote table. If the count of columns in the proxy table is less than the actual number of columns in the remote server, a warning is issued, but the command is not aborted.

- The remote column name is stored in syscolumns.remote_name and is used during query processing when a statement is forwarded to the remote server. This name is not affected by sp_rename, so after the proxy table is created, if any column name is changed, it does not affect processing of subsequent SQL commands.
- Column datatypes do not need to be identical, but they must be convertible in both directions, or a column datatype mismatch error is raised and the command is aborted.
- The column length defined for columns of type char, varchar, binary, and varbinary must match the length of the corresponding columns in the remote table.
- Scale and precision of columns of type numeric or decimal must match the scale and precision of the corresponding columns in the remote table.
- If the null property is not identical to the remote column's null property, a warning message is issued, but the command is not aborted.

Server class *ASEnterprise*

- Table 3-12 describes the allowable datatypes that can be used when mapping remote Adaptive Server columns to local proxy table columns:

Table 3-12: Adaptive Server datatype conversions for create existing table

Remote Adaptive Server datatype	Allowable Adaptive Server datatypes
binary(*n*)	image, binary(*n*), and varbinary(*n*); if not image, the length must match
bit	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
char(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*), unichar, univarchar; if not text, the length must match
datetime	datetime and smalldatetime
decimal(*p*, *s*)	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
float	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
image	image
int	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
money	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint

Remote Adaptive Server datatype	Allowable Adaptive Server datatypes
nchar(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*); if not text, the length must match
numeric(*p*, *s*)	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
nvarchar(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*), unichar, univarchar; if not text, the length must match
real	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
smalldatetime	datetime and smalldatetime
smallint	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
smallmoney	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
text	text
timestamp	timestamp
tinyint	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
unichar	char, varchar, unichar, univarchar, text, datetime, and smalldatetime
univarchar	char, varchar, unichar, univarchar, text, datetime, and smalldatetime
varbinary(*n*)	image, binary(*n*), and varbinary(*n*); if not image, the length must match
varchar(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*) unichar, univarchar; if not text, the length must match

Server class *ASAnywhere*

- Table 3-13 describes the allowable datatypes that can be used when mapping remote Adaptive Server columns to local proxy table columns:

Table 3-13: Adaptive Server Anywhere datatype conversions for create existing table

Remote Adaptive Server datatype	Allowable Adaptive Server datatypes
binary(*n*)	image, binary(*n*), and varbinary(*n*); if not image, the length must match
bit	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint

Remote Adaptive Server datatype	Allowable Adaptive Server datatypes
char(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*), unichar, univarchar; if not text, the length must match
datetime	datetime and smalldatetime
decimal(*p*, *s*)	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
float	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
image	image
int	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
money	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
nchar(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*), unichar, univarchar; if not text, the length must match
numeric(*p*, *s*)	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
nvarchar(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*), unichar, univarchar; if not text, the length must match
real	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
smalldatetime	datetime and smalldatetime
smallint	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
smallmoney	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
text	text
timestamp	timestamp
tinyint	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
varbinary(*n*)	image, binary(*n*), and varbinary(*n*), unichar, univarchar; if not image, the length must match
varchar(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*), unichar, univarchar; if not text, the length must match

Server class *ASIQ*

- text and image datatypes are not supported by ASIQ.
- Other than text and image datatypes, behavior is the same as for server class ASAnywhere.

Server class *direct_connect*

- The RPC sp_columns queries the datatypes of the columns in the existing table.
- Local column datatypes do not need to be identical to remote column datatypes, but they must be convertible as shown in Table 3-14. If not, a column type error is raised and the command is aborted.

Table 3-14: DirectConnect datatype conversions for create existing table

DirectConnect datatype	Allowable Adaptive Server datatypes
binary(*n*)	image, binary(*n*), varbinary(*n*); if the length does not match, the command is aborted
binary(16)	timestamp
bit	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
char(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*) and varchar(*n*), unichar, univarchar; if the length does not match, the command is aborted
datetime	datetime, smalldatetime
decimal(*p*, *s*)	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
float	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
image	image
int	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
money	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
nchar(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*) and varchar(*n*), unichar, univarchar; if the length does not match, the command is aborted
numeric(p, s)	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
nvarchar(*n*)	text, nchar(*n*), nvarchar(*n*), char(*n*) and varchar(*n*), unichar, univarchar; if the length does not match, the command is aborted

DirectConnect datatype	Allowable Adaptive Server datatypes
real	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
smalldatetime	datetime, smalldatetime
smallint	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
smallmoney	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
text	text
timestamp	timestamp, binary(8), varbinary(8)
unichar	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*), unichar, univarchar; if not text, the length must match
univarchar	text, nchar(*n*), nvarchar(*n*), char(*n*), varchar(*n*), unichar, univarchar; if not text, the length must match

- Datatype information is passed in the CS_DATAFMT structure associated with the parameter. The following fields of the structure contain datatype information:
  - *datatype* – the CS_Library datatype representing the Adaptive Server datatype. For example, CS_INT_TYPE.
  - *usertype* – the native DBMS datatype. sp_columns passes this datatype back to Component Integration Services during a create existing table command as part of its result set (see sp_columns in the *Reference Manual*). Adaptive Server returns this datatype in the *usertype* field of parameters to assist the DirectConnect in datatype conversions.

Server class *db2*

- Column names are checked in a case-insensitive manner. If there is no match, a column name error is raised and the command is aborted.

  Note The Adaptive Server table can contain fewer columns than the remote table, but each column in the Adaptive Server table must have a matching column in the remote table.

- text and image datatypes are not supported by server class db2.

- When a create existing table command is processed, the datatype for each column specifies the type of conversion to perform between the DB2 and Adaptive Server datatypes during query processing. Table 3-15 describes the allowable Adaptive Server datatypes that can be used for existing DB2 datatypes:

Table 3-15: DB2 datatype conversions for create existing table

DB2 datatype	Allowable Adaptive Server datatypes
int	int
smallint	int, smallint, and tinyint; if length does not match, a warning message is issued
tinyint	int, smallint, and tinyint; if length does not match, a warning message is issued
float	real, float, and money
double precision	real, float, and money
real	real, float, and money
decimal(scale > 0)	float, money, decimal, and numeric; for decimal and numeric, scale and precision must match
decimal (scale = 0)	float, money, decimal, and numeric; for decimal and numeric, scale and precision must match
numeric (scale > 0)	float, money, decimal, and numeric; for decimal and numeric, scale and precision must match
numeric (scale = 0)	float, money, decimal, and numeric; for decimal and numeric, scale and precision must match
char	char, varchar, bit, binary, varbinary, text and image, unichar, univarchar; if not text or image, length must match
char(*n*) for bit data	binary(*n)*, varbinary(*n)*, unichar, univarchar, and image; if not image, length must match
varchar	char, varchar, bit, binary, varbinary,unichar, univarchar, text and image; if not text or image, length must match
varchar(*n*) for bit data	binary(*n)*, varbinary(*n)*,unichar, univarchar, and image; if not image, length must match
long varchar (length of maximum column size depends on the logical page size of the Adaptive Server)	char, varchar, bit, binary, varbinary, unichar, univarchar, text and image; if not text or image, length must match
date	char(10), varchar(10), and datetime (time set to 12:00 a.m.)

DB2 datatype	Allowable Adaptive Server datatypes
time	char(8), varchar(8), and datetime (date set to 1/1/1900)
timestamp	char(26), varchar(26), datetime, and smalldatetime
graphic	Not supported
vargraphic	Not supported
long vargraphic	Not supported

- If the data contained in a long varchar column exceeds the length allowed by the logical page size of the Adapted Server, it is truncated, or, if the gateway is so configured, an error is returned.
- DB2 table names are limited to 18 characters.
- DB2 authorization IDs (owner names) are limited to 8 characters.
- The maximum string length for columns returned by DB2 is 254 characters for char and varchar datatypes. For long varchar, the length is 32,704 bytes.
- DB2 can return date values that are not within the range of the Adaptive Server datetime datatype. DB2s range is 0001-01-01 to 9999-12-31. The Adaptive Server's range is 1753-01-01 to 9999-12-31. When a date earlier than 1753-01-01 is retrieved from DB2, it is converted to 1753-01-01.
- Check DB2 documentation for the maximum number of columns per DB2 table. This varies with the DB2 version.

See also create existing table in the *Reference Manual*

create index

Description Creates an index on one or more columns in a table.

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]
```

Usage

- Component Integration Services processes create index when the table involved has been created as a proxy table. The actual table resides on a remote server, and Component Integration Services forwards the request to the remote server after Adaptive Server catalogs are updated to represent the new index.
- Trace flag 11208 changes the behavior of create index. If trace flag 11208 is turned on, Component Integration Services does not send create index to the remote server. Instead, Adaptive Server processes the command locally, as if the table on which it operates is local. This is useful for creating an index on a proxy table that maps to a remote view.
- Adaptive Server performs all system catalog updates to identify the index. However, just as there are no data pages in the server for proxy tables, there are no index pages.
- When Component Integration Services forwards create index to a remote server, the table name used is the remote table name, and the column names used are the remote column names. These names may not be the same as the local proxy table names.

Server class *ASEnterprise*

Component Integration Services forwards everything except the on *segment_name* clause to the remote server.

Server class *ASAnywhere*

Component Integration Services forwards everything except the on *segment_name* clause to the remote server.

Server class *ASIQ*

Component Integration Services forwards everything except the on *segment_name* clause to the remote server.

Server class *direct_connect*

- When the language capability is set to “Transact-SQL”, Component Integration Services forwards all syntax except the max_rows_per_page and on *segment_name* clauses to the remote server.
- When the language capability is set to “DB2”, the behavior is the same as for server class db2.
- The DirectConnect must either translate the Sybase extensions to equivalent native syntax or ignore them.

Server class *db2*

- Component Integration Services does not forward the following clauses to the remote server:
  - on segment_name
  - max_rows_per_page
  - ignore_dup_key
  - ignore_dup_row
  - allow_dup_row
- Component Integration Services converts the fillfactor option to pctfree and then forwards it to the remote server.

See also

create index in the *Reference Manual*

create proxy_table

Description

Creates a proxy table without specifying a column list. Component Integration Services derives the column list from the metadata it obtains from the remote table.

Syntax

```
create proxy_table table_name
[ external type ] at pathname
```

Usage

- create proxy_table is a variant of the create existing table command. You use create proxy_table to create a proxy table, but (unlike create existing table) you do not specify a column list. Component Integration Services derives the column list from the metadata it obtains from the remote table.
- The location information provided by the at keyword specifies the path name to the remote object.
- If the remote server object does not exist, the command is rejected with an error message.
- If the remote object exists, column metadata is obtained from the remote server, and an internal create existing table command is processed.
- If the remote server is case insensitive (as is the case with DB2, Oracle, perhaps others), then the case of the generated table and column names is determined by the case of the table_name used with the create proxy_table statement:

- If table_name is lowercase, the generated proxy table name is also lower case, as are all of its columns
- If table_name is uppercase, the generated proxy table name is also upper case, as are all of its columns

- The external type can be one of the following:
  - external table specifies that the object is a remote table or view. external table is the default.
  - external directory specifies that the object is a directory with a path in the following format: "*/tmp/directory_name* [;R]". The option "R" indicates recursive.
  - external file specifies that the object is a file with a path in the following format: "*/tmp/filename*"

See also — create table and create existing table in the *Reference Manual*

create table

Description — Creates new tables and optional integrity constraints; specifies a locking scheme for the table being created; specifies ascending or descending index order when creating referential integrity constraints that depend on indexes; specifies the expected row size, to reduce row forwarding; specifies a ratio of empty pages to be left for each filled page; allows you to map the table to a table, view, or procedure at a remote location.

Syntax

```
create table [database.[owner].]table_name (column_name datatype
   [default {constant_expression | user | null}]
   {[{identity | null | not null}]
                  [off row | in row]
   | [[constraint constraint_name]
      {{unique | primary key}
      [clustered | nonclustered] [asc | desc]
      [with { { fillfactor = pct
                | max_rows_per_page = num_rows }
                , reservepagegap = num_pages }]
       [on segment_name]
       | references [[database.]owner.]ref_table
          [(ref_column)]
       | check (search_condition)}]}...
            | [constraint constraint_name]
   {{unique | primary key}
      [clustered | nonclustered]
```

```
      (column_name [asc | desc]
                    [{, column_name [asc | desc]}...])
    [with { {fillfactor = pct
                     | max_rows_per_page = num_rows },
                    reservepagegap = num_pages } ]
     [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}}...])
[lock {datarows | datapages | allpages }]
[with { max_rows_per_page = num_rows ,
                exp_row_size = num_bytes ,
                reservepagegap = num_pages } ]
[on segment_name]
[ [ external {table | file}] at “pathname” ]]
```

Usage

- If the table being created is mapped to a remote location, a proxy table is created. A proxy table is identical to a local table, except that the sysobjects.sysstat2 column contains a status flag that indicates the table is mapped to an external location.
- The external location must be previously defined using the at pathname clause.
- After the Adaptive Server processes create table, it notifies Component Integration Services of the need to forward the command to the remote location (if a location has been previously specified).

  Component Integration Services reconstructs the SQL necessary to create the table, and forwards the SQL to the remote server. It does not forward all the original syntax to the remote server. The following clauses are processed by Adaptive Server:

  - on *segment name*
  - check constraints
  - default
  - with max_rows_per_page
- Trace flag 11213 changes the behavior of create table. Referential constraints and unique or primary key constraints are forwarded to the remote server unless trace flag 11213 is turned on, in which case they are processed locally.
- For each column, the column name, datatype, length, identity property, and null property are reconstructed from the original statement.

- Component Integration Services passes a NULL char column as a NULL varchar column.
- Component Integration Services passes a NULL binary column as a NULL varbinary column.

Server class *ASEnterprise*

Component Integration Services passes the datatype of each column to the remote server without conversion.

Server class *ASAnywhere*

Component Integration Services passes the datatype of each column to the remote server without conversion.

Server class *ASIQ*

Component Integration Services passes the datatype of each column to the remote server without conversion.

Server class *direct_connec*t

- Component Integration Services reconstructs create table and passes commands to the targeted DirectConnect. The gateway transforms the commands into a form that the underlying DBMS recognizes.
- Some DirectConnects support DB2 syntax mode, which is described in the DirectConnect documentation. When the DirectConnect enables DB2 syntax mode, Component Integration Services constructs DB2 SQL syntax and converts the column to a datatype DB2 supports.
- Adaptive Server datatypes are converted to either the DirectConnect or DB2 syntax mode datatypes shown in Table 3-16, depending on whether the DirectConnect supports DB2 syntax mode.

Table 3-16: DirectConnect datatype conversions for create table

Adaptive Server datatype	DirectConnect default datatype	DirectConnect DB2 syntax mode datatype
binary*(n)*	binary*(n)*	char*(n)* for bit data
bit	bit	char(1)
char	char	char
datetime	datetime	timestamp
decimal*(p, s)*	decimal*(p, s)*	decimal*(p, s)*
float	float	float
image	image	varchar*(n)* for bit data; the value of *n* is determined by the global variable @@textsize

Adaptive Server datatype	DirectConnect default datatype	DirectConnect DB2 syntax mode datatype
int	int	int
money	money	float
numeric*(p, s)*	numeric*(p, s)*	decimal*(p, s)*
nchar(n)	nchar(n)	graphic(n)
nvarchar(n)	nvarchar(n)	vargraphic(n)
real	real	real
smalldatetime	smalldatetime	timestamp
smallint	smallint	smallint
smallmoney	smallmoney	float
timestamp	timestamp	varbinary(8)
tinyint	tinyint	smallint
text	text	varchar*(n)*; the value of *n* is determined by the global variable @@textsize
unichar(n)	unichar	char*(n)* for bit data
univarchar(n)	char*(n)* for bit data	varchar*(n)* for bit data
varbinary*(n)*	varbinary*(n)*	varchar*(n)* for bit data
varchar*(n)*	varchar*(n)*	varchar*(n)*

Server class *db2*

Table 3-17 shows the datatype conversions that are performed when a create table command is processed. Adaptive Server datatypes are converted to the DB2 datatypes shown.

Table 3-17: DB2 datatype conversions for create table

Adaptive Server datatype	DB2 datatype
binary(n)	char(n) for bit data, where *n* <= 254
bit	char(1)
char(n)	char(n), where *n* <= 254
datetime	timestamp
decimal(p, s)	decimal*(p, s)*
float	float
image	Not supported
int	int
money	float
nchar	char(n)

Adaptive Server datatype	DB2 datatype
nvarchar	varchar(n)
numeric*(p, s)*	decimal*(p, s)*
real	real
smalldatetime	timestamp
smallint	smallint
smallmoney	float
tinyint	smallint
text	Not supported
varbinary(n)	varchar(n) for bit data, where *n* <=254
varchar(n)	varchar(n), where *n* <= 254

See also

create table in the *Reference Manual*

create trigger

Description

Creates a trigger, which is a type of stored procedure that is often used for enforcing integrity constraints. A trigger executes automatically when a user attempts a specified data modification statement on a specified table.

Syntax

```
create trigger [owner.]trigger_name
on [owner.]table_name
for {insert, update, delete}
as SQL_statements
```

Or, using the if update clause:

```
create trigger [owner.]trigger_name
on [owner.]table_name
for {insert, update}
as
   [if update (column_name)
      [{and | or} update (column_name)]...]
      SQL_statements
   [if update (column_name)
      [{and | or} update (column_name)]...
      SQL_statements]...
```

Usage

When a trigger is created on a proxy table, it will execute after an insert, delete or update statement on that proxy table completes. However, the special tables inserted and deleted, which normally are views into the local transaction log, will not contain any data, since changes to remote data are not logged locally.

deallocate cursor

Description

Makes a cursor inaccessible and releases all memory resources committed to that cursor.

Syntax

deallocate cursor *cursor_name*

Usage

- If the cursor specified by *cursor_name* contains references to proxy tables, Adaptive Server notifies Component Integration Services to deallocate its remote cursors for those tables.
- If the remote cursor is not closed, Component Integration Services closes and deallocates it. If the remote cursor is already closed, no additional actions are taken.
- Component Integration Services uses Client-Library to manage cursor operations to a remote server. When Component Integration Services receives a deallocate cursor command and the cursor has not been explicitly closed with a close command, Component Integration Services uses the following Client-Library functions to interact with the remote server:

  ct_cursor(*command*, CS_CURSOR_CLOSE, NULL, CS_UNUSED, NULL, CS_UNUSED, CS_UNUSED)

  ct_cursor(*command*, CS_CURSOR_DEALLOC, NULL, CS_UNUSED, NULL, CS_UNUSED, CS_UNUSED)
- If the cursor contains references to more than one proxy table, Component Integration Services must deallocate a remote cursor for each server represented by the proxy tables.

See also

close, declare cursor, fetch, open in this chapter

deallocate cursor in the *Reference Manual*

declare cursor

Description — Defines a cursor.

Syntax

```
declare cursor_name cursor
 for select_statement
[for {read only | update [of column_name_list]}]
```

Usage — If the cursor specified by *cursor_name* contains references to proxy tables, Adaptive Server notifies Component Integration Services to establish a connection to the remote servers referenced by the proxy tables.

A separate connection is required for each server represented by all proxy tables. For example, if all proxy tables in the cursor reference the same remote server, only one connection is required while the declare cursor command is processed. However, if two or more servers are referenced by the proxy tables, a separate connection to each server is required.

See also — close, deallocate cursor, fetch, open in this chapter

declare cursor in the *Reference Manual*

delete

Description — Removes rows from a table.

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]
```

```
delete [from] [[database.]owner.]{table_name|view_name}
 where current of cursor_name
```

Usage

- Component Integration Services processes delete when the table on which it operates has been created as a proxy table. Component Integration Services forwards the entire request (or part of it) to the server that owns the actual object.
- Component Integration Services executes delete using one of two methods:

  - Method 1 – The entire command is forwarded to the remote server as a single statement in close to its original syntax. If the syntax and remote capabilities match, the entire statement is forwarded and processed remotely. This is referred to as *quickpass mode*.
  - Method 2 – If the entire command cannot be forwarded to a remote server, Component Integration Services declares and opens one or more cursors in update mode, and begins a scan on the remote table. Each cursor forwards as much of the original statement's predicates to the remote server as possible. For each row fetched that meets the search criteria, a positioned delete is executed.
- When Component Integration Services forwards delete to a remote server, the table name used is the remote table name, and the column names used are the remote column names. These names may not be the same as the local proxy table names.
- Component Integration Services generally passes the original delete syntax to remote servers as a single statement, but the following conditions will likely cause the statement to be executed using method 2, above:
  - The statement contains multiple tables that are not located in the same remote server
  - The statement contains local tables (including temporary tables)
  - The statement contains case expressions
  - The statement contains text or image columns
  - The statement contains certain referential integrity checks
  - The statement contains system functions in the predicate list
  - The statement contains syntax that the remote server does not support
- The format involving where current of is never forwarded to a remote server and causes the statement to be executed using method 2 above.
- If Component Integration Services cannot pass the entire statement to a remote server, a unique index must exist on the table.

Server class *ASEnterprise*

If Component Integration Services cannot forward the original query without alteration, it performs the delete using method 2.

Server class *ASAnywhere*

If Component Integration Services cannot forward the original query without alteration, it performs the delete using method 2.

Server class *ASIQ*

If Component Integration Services cannot forward the original query without alteration, it performs the delete using method 2.

Server class *direct_connect*

- The syntax forwarded to servers of class direct_connect is dependent on the capabilities negotiation, which occurs when Component Integration Services first connects to the remote DirectConnect. Examples of negotiable capabilities include: subquery support, group by support, and built-in support.
- A DirectConnect can request that delete be generated in DB2 syntax.
- Component Integration Services passes data values as parameters to either a cursor or a dynamic SQL statement. Language statements can also be used if the DirectConnect supports it. The parameters are in the datatype native to Adaptive Server and must be converted by the DirectConnect into formats appropriate for the target DBMS.

Server class *db2*

- Servers of class db2 do not contain the capabilities negotiation features of server class direct_connect, so the syntax passed to the remote server is simpler than that allowed by Transact-SQL. The syntax does not contain the following:
  - Search conditions containing subqueries, group by, or order by clauses
  - Transact-SQL built-in functions
  - Transact-SQL operators (such as bitwise operators)
  - Syntax not allowed by DB2

  Component Integration Services processes delete using method 2, described above, when the statement is complex.
- If the server is a DB2 system, use trace flag 11215 to instruct Component Integration Services that the remote server can handle all DB2 syntax. This assumption is not made automatically because not all gateways using the db2 server class are actually connected to DB2 systems. When trace flag 11215 is turned on, quickpass mode is used unless the following conditions exist:
  - The statement cannot be expressed in DB2 syntax.
  - The statement contains outer joins.
  - The statement contains like clauses with Sybase extensions.

- The statement contains built-in functions that are not supported by DB2.

See also — delete in the *Reference Manual*

drop database

Description — Removes one or more databases from Adaptive Server.

Syntax — drop database *database_name* [, *database_name*]...

Usage — For each database being dropped, Component Integration Services scans sysobjects to check for proxy tables in the database. Each proxy table that was not created with the existing keyword, with create_proxy_table, or with create proxy database is dropped in the remote server that owns the object.

See also — drop database in the *Reference Manual*

drop index

Description — Removes an index from a table in the current database.

Syntax — drop index *table_name.index_name*
[, *table_name.index_name*]...

Usage

- Component Integration Services processes drop index when the table involved has been created as a proxy table. The actual table and index reside on a remote server. Component Integration Services forwards the request to the remote server, and removes the index from the proxy table.
- When Component Integration Services forwards drop index to a remote server, the table name used is the remote table name, and the index names used are the remote index names. These names may not be the same as the local proxy table names.
- If multiple indexes are dropped in a single command, each index is sent as an individual drop index command.

- Trace flag 11208 changes the behavior of drop index. If trace flag 11208 is turned on, drop index is not sent to the remote server. Instead, Adaptive Server processes the command locally, as if the table on which it operates is local. This is useful for synchronizing the local Adaptive Server schema with the schema of the remote database.

Server class *ASEnterprise*

Component Integration Services forwards the following drop index syntax to a remote server configured as class ASEnterprise:

```
drop index table_name.index_name
```

Component Integration Services precedes this statement with a use database command since the drop index syntax does not allow you to specify the database name.

Server class *ASAnywhere*

- Component Integration Services forwards the following drop index syntax to a remote server configured as class ASAnywhere:

  ```
  drop index table_name.index_name
  ```

  Component Integration Services precedes this statement with a use database command since the drop index syntax does not allow you to specify the database name.

Server class *ASIQ*

Component Integration Services forwards the following drop index syntax to a remote server configured as class ASIQ:

```
drop index table_name.index_name
```

Component Integration Services precedes this statement with a use database command since the drop index syntax does not allow you to specify the database name.

Server class *direct_connect*

Component Integration Services forwards the following drop index syntax to a remote server configured as class direct_connect:

```
drop index table_name.index_name
```

Server class *db2*

Component Integration Services forwards the following drop index syntax to a remote server configured as class db2:

```
drop index index_name
```

See also drop index in the *Reference Manual*

drop table

Description

Removes a table definition and all of its data, indexes, triggers, and permissions from the database.

Syntax

drop table [[*database*.]*owner*.]*table_name*
[, [[*database*.]*owner*.]*table_name*]...

Usage

Component Integration Services processes the drop table command when the table on which it operates has been created as a proxy table. Component Integration Services forwards the entire request (or part of it) to the server that owns the actual object if the table was not created with the existing keyword, create_proxy_table, or create proxy database.

See also

drop table in the *Reference Manual*

execute

Description

Runs a system procedure or a user-defined stored procedure.

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]
```

Usage

- When execute is used to issue an RPC to a remote server, Adaptive Server issues the RPC via one of two methods. The method used to issue the RPC determines whether the work performed by the RPC can be part of an ongoing transaction. The two methods are as follows:
  - The RPC is issued via the Adaptive Server's site handler. This is the Adaptive Server's default method of issuing RPCs. In this case, the RPC cannot be part of an on-going transaction.
  - The RPC is issued via Component Integration Services. In this case, the RPC can be part of an on-going transaction. To issue RPCs using this method, cis rpc handling must be turned on. This is done via set or sp_configure.

See also

"RPC handling and Component Integration Services" on page 58

set in this chapter

execute in the *Reference Manual*

fetch

Description

Returns a row or a set of rows from a cursor result set.

Syntax

fetch *cursor_name* [into *fetch_target_list*]

Usage

- When the first fetch is received, Component Integration Services constructs the query defined by the declare cursor command and sends it to the remote server.

  If the remote server supports Client-Library cursors, Component Integration Services:

  a Declares a cursor:

  ```
  ct_cursor(command, CS_CURSOR_DECLARE...)
  ```

  b Establishes the cursor row count:

  ```
  ct_cursor(command, CS_CURSOR_ROWS,...
  cursor_row_count)
  ```

  c Opens a Client-Library client cursor to the remote server:

  ```
  ct_cursor(command, CS_CURSOR_OPEN...)
  ```

  If the remote server does not support Client-Library cursors, Component Integration Services sends a language request to the server. This may require an additional connection to that server.

- If declare cursor included a for update clause, the cursor row count is set to 1; otherwise, it is set to the value of the configuration parameter cis_cursor_rows.

- After the cursor is opened or the language request is sent, Component Integration Services issues a Client-Library ct_fetch command to obtain the first row. Client-Library array binding is used to establish the buffer in which to place the fetched results, whether Client-Library cursors or language requests are used to generate the fetchable results. The number of rows that are buffered by a single fetch is determined by the cursor row count discussed above.

  Subsequent fetch requests retrieve rows from the buffered results, until the end of the buffer is reached. At that time, Component Integration Services issues another Client-Library ct_fetch command to the remote server.

- A fetch against a cursor that has no remaining rows in its result set causes Component Integration Services to close the remote cursor.

Server class *ASEnterprise*

If the cursor is read only, Component Integration Services sends a language request to the remote server when the first fetch is received after the cursor is opened. Otherwise, Component Integration Services declares a cursor to the remote server by means of Client-Library.

Server class *ASAnywhere*

Handling of the fetch statement is the same as for ASEnterprise.

Server class *ASIQ*

Component Integration Services sends a language request to the remote server when the first fetch is requested after the cursor is opened.

Server class *direct_connect*

Component Integration Services treats servers in this class the same as servers in ASEnterprise.

Server class *db2*

Component Integration Services sends a language request to the remote server when the first fetch is requested after the cursor is opened.

See also

close, deallocate cursor, declare cursor, open in this chapter

fetch in the *Reference Manual*

insert

Description

Adds new rows to a table or view.

Syntax

```
insert [into] [database.[owner.]]{table_name|view_name}
 [(column_list)]
 {values (expression [, expression]...)
    |select_statement }
```

Usage

- Component Integration Services processes insert when the table on which it operates has been created as a proxy table. Component Integration Services forwards the entire request (or part of it) to the server that owns the actual object.

- When Component Integration Services forwards insert to a remote server, the table name used is the remote table name, and the column names used are the remote column names. These names may not be the same as the local proxy table names.

Server class *ASEnterprise*

- insert commands using the values keyword are fully supported.
- insert commands using a select command are supported for all datatypes except text and image. text and image columns are only supported when they contain null values.
- If all insert and select tables reside on the same remote server, the entire statement is forwarded to the remote server for execution. This is referred to as quickpass mode. Quickpass mode is not used if select does not conform to all the quickpass rules for a select command (see select on page 140).
- If the select tables reside on one remote server, and the insert table resides on a different server, Component Integration Services selects each row from the source tables, and inserts the row into the target table.

Server class *ASAnywhere*

Handling of the insert statement is the same as for ASEnterprise.

Server class *ASIQ*

Handling of the insert statement is the same as for ASEnterprise.

Server class *direct_connect*

- insert commands using the values keyword are fully supported.
- insert commands using a select command are fully supported, but the table must have a unique index if the table has text or image columns. When using insert with a select command, the entire command is sent to the remote server if:
  - All tables referenced in the command reside on the remote server.
  - The capability's response from the DirectConnect indicates that insert-select commands are supported.

  If both conditions are not met, Component Integration Services selects each row from the source tables, and inserts the row into the target table.

- Component Integration Services passes data values as parameters to either a cursor or a dynamic SQL statement. Language statements can also be used if the DirectConnect supports it. The parameters are in the datatype native to Adaptive Server and must be converted by the DirectConnect into formats appropriate for the target DBMS.

Server class *db2*

- insert commands using the values keyword are fully supported for all valid DB2 datatypes.
- insert commands using a select command are fully supported for all valid DB2 datatypes.
- When using insert with a select command, the entire statement is sent to the remote server if:
  - All tables referenced in the statement reside on the remote server.
  - Trace flag 11215 is enabled.

  If both conditions are not met, Component Integration Services selects each row from the source tables, and inserts the rows into the target table.

See also

insert in the *Reference Manual*

open

Description

Opens a cursor for processing.

Syntax

open *cursor_name*

Usage

- open opens a cursor. Cursors allow you to modify or delete rows on an individual basis. You must first open a cursor to use the fetch, update, and delete statements. For more information about cursors, see the *Transact-SQL User's Guide.*
- Adaptive Server returns an error message if the cursor is already open or if the cursor has not been created with declare cursor.
- Opening the cursor causes Adaptive Server to evaluate the select statement that defines the cursor (specified in the declare cursor statement) and makes the cursor result set available for processing.
- When the cursor is first opened, it is positioned before the first row of the cursor result set.

- When you set the chained transaction mode, Adaptive Server implicitly begins a transaction with the open statement if no transaction is currently active.

See also

close, deallocate cursor, declare cursor, fetch in this chapter

open in the *Reference Manual*

prepare transaction

Description

Used by two-phase commit applications to see if a server is prepared to commit a transaction.

Syntax

prepare tran[saction]

Usage

- Adaptive Server notifies Component Integration Services when it receives a prepare transaction command so that remote servers involved in the current transaction can enter the prepared state.
- For each server that is involved in the current transaction, a prepare command (according to the DTM capabilities of the server) is sent to the server and the results are monitored. If there are no errors reported, each remote server is assumed to be in a prepared state and Component Integration Services returns control to the Adaptive Server. Adaptive Server then enters a prepared state for local work performed by the transaction.

See also

prepare transaction in the *Reference Manual*

readtext

Description

Reads text and image values, starting from a specified offset and reading a specified number of bytes or characters.

Syntax

```
readtext [[database.]owner.]table_name.column_name
text_pointer offset size
[holdlock | noholdlock] [readpast]
[using {bytes | chars | characters}]
[at isolation {
        [ read uncommitted | 0 ] |
        [ read committed | 1 ] |
```

```
[ repeatable read | 2 ]|
[ serializable | 3 ] } ]]
```

Usage

- Component Integration Services processes the readtext command when the table on which it operates has been created as a proxy table. Component Integration Services forwards the entire request (or part of it) to the server that owns the actual object.
- When Component Integration Services forwards the readtext command to a remote server, the table name used is the remote table name, and the column names used are the remote column names. These names may not be the same as the local proxy table names.
- The using bytes and at isolation clauses are ignored.
- The holdlock, noholdlock, and readpast options are ignored.

Server class *ASEnterprise*

Component Integration Services forwards the following syntax to the remote server when the underlying table is a proxy table:

```
readtext [[database.]owner.]table_name.column_name
 text_pointer offset size
[using {chars | characters}]
```

Server class *ASAnywhere*

Handling of the readtext statement is the same as for ASEnterprise.

Server class *ASIQ*

Handling of the readtext statement is the same as for ASEnterprise.

Server class *direct_connect*

- If the DirectConnect does not support text pointers, readtext cannot be sent and its use results in errors.
- If the DirectConnect does support text pointers, Component Integration Services forwards the following syntax to the remote server:

  ```
  readtext
   [[database.]owner.]table_name.column_name
  text_pointer offset size
   [using {chars | characters}]
  ```

- readtext is issued anytime text or image data must be read. readtext is called when a select command refers to a text or image column in the select list, or when a where clause refers to a text or image column.

  For example, you have a proxy table books that is mapped to the books table on the remote server foo. The columns are id, name, and the text column blurb. When the following statement is issued:

```
select * from books
```

Component Integration Services sends the following syntax to the remote server:

```
select id, name, textptr(blurb) from foo_books
readtext foo_books.blurb @p1 0 0 using chars
```

Server class *db2*

readtext is not supported since text and image datatypes are not supported for servers in class db2.

See also

readtext in the *Reference Manual*

rollback transaction

Description

Rolls a user-defined transaction back to the last savepoint inside the transaction or to the beginning of the transaction.

Syntax

```
rollback {transaction | tran | work}
 [transaction_name | savepoint_name ]
```

Usage

- Adaptive Server notifies Component Integration Services when it receives a rollback transaction command and Component Integration Services attempts to roll back work associated with remote servers in the current transaction.
- Multiple remote servers can be involved in a single transaction, each with its own unit of work, which is associated with the Adaptive Server unit of work.
- Remote work is rolled back before local work.
- Work performed by transactional RPCs is included in the local transaction and can be rolled back if the remote server supports RPC's within transactions.
- *transaction_name* and *savepoint_name* are not currently used by Component Integration Services.

Object type = file

- files are not part of transaction management. They cannot be committed or rolled back.

Object type = directory

- directories are not part of transaction management. They cannot be committed or rolled back.

See also rollback in the *Reference Manual*

select

Description Retrieves rows from database objects.

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]
[plan "abstract plan"]
```

Usage

- Component Integration Services processes the select command when any table on which it operates has been created as a proxy table. When possible, Component Integration Services forwards the entire syntax of a select command to a single remote server. This is referred to as quickpass mode.
- When Component Integration Services forwards the select command to a remote server, the table name used is the remote table name, and the column names used are the remote column names.
- The following keywords are ignored for all servers except Adaptive Server, but they do not prevent Component Integration Services from using quickpass mode:
  - lock
  - index
  - parallel
  - prefetch size
  - holdlock
  - noholdlock
  - readpast
  - shared
  - at isolation
- The following keywords are never forwarded to a remote server and they do prevent Component Integration Services from using quickpass mode:
  - compute by
  - for browse
  - into
  - plan “abstract plan”
- Quickpass mode is not used if any of the following conditions exist:
  - All tables referenced in the from clause do not reside on the same remote server.
  - Any tables are local (including temporary tables).
  - The query contains syntax that the remote server does not support.

- select commands in a union operation can all be forwarded to a remote server, including the union operator, if all tables in the select commands reside on the same remote server.
- If select returns a sorted result set involving a character column from a remote server (for example, in a union operation, a group by clause, or an order by clause), the rows may be returned in an unexpected sort order if the remote server is configured with a different sort order than Adaptive Server. You can re-run the query with trace flag 11216 turned on to receive the expected sort order. This trace flag is global and should be turned off as soon as the query is executed.

Server class *ASEnterprise*

- All syntax is supported. Since the remote server is assumed to have all capabilities necessary to process Transact-SQL syntax, all elements of a select command, except those mentioned above, are forwarded to a remote server, using quickpass mode.
- A bulk copy transfer is used to copy data into the new table when a select...into command is issued and the into table resides on a remote Adaptive Server. Both the local and remote databases must be configured with dboption set to select into / bulkcopy.

Server class *ASAnywhere*

- All syntax is supported. Since the remote server is assumed to have all capabilities necessary to process Transact-SQL syntax, all elements of a select command, except those mentioned above, are forwarded to a remote server, using quickpass mode.
- If the select...into format is used and the into table is accessed through the ASAnywhere interface, bulk inserts are not used. Instead, Component Integration Services uses Client-Library to prepare a parameterized dynamic insert command, and executes it for each row returned by the select portion of the command.

Server class *ASIQ*

- All syntax is supported. Since the remote server is assumed to have all capabilities necessary to process Transact-SQL syntax, all elements of a select command, except those mentioned above, are forwarded to a remote server, using quickpass mode.
- If the select...into format is used and the into table is accessed through the db2 interface, bulk inserts are not used. Instead, a separate connection is used to handle the text of a Component Integration Services-generated insert command.

Server class *direct_connect*

- The first time Component Integration Services requires a connection to a server in class direct_connect, a request for capabilities is made of the DirectConnect. Based on the response, Component Integration Services determines the parts of a select command to forward to the DirectConnect. In most cases, this is determined by the capabilities of the DBMS with which the DirectConnect is interfacing.
- If the entire statement cannot be forwarded to the DirectConnect using quickpass mode, Component Integration Services compensates for the functionality that cannot be forwarded. For example, if the remote server cannot handle the order by clause, quickpass is not used and Component Integration Services performs a sort on the result set.
- Component Integration Services passes data values as parameters to either a cursor or a dynamic SQL statement. Language statements can also be used if the DirectConnect supports it. The parameters are in the datatype native to Adaptive Server and must be converted by the DirectConnect into formats appropriate for the target DBMS.
- The select...into command is supported, but the table must have a unique index if the table has text or image columns.
- If the select...into format is used and the into table is accessed through a DirectConnect, bulk inserts are not used. Instead, Component Integration Services uses Client-Library to prepare a dynamic insert command, and executes it for each row returned by the select portion of the command.

Server class *db2*

- By default, Component Integration Services does not forward syntax involving order by, group by, union, distinct, all, and expressions that involve more than column names.
- When you turn trace flag 11215 on, the full capabilities of a DB2 database are assumed, and Component Integration Services forwards as much syntax to the remote server (gateway) as DB2 can process, including order by, group by, union, and so forth.

See also

select in the *Reference Manual*

set

Description

Sets Adaptive Server query processing options for the duration of the user's work session. The subset of options listed below affects behavior unique to Component Integration Services. For a complete list of options, see the *Reference Manual*.

Syntax

set cis_rpc_handling {on | off}

set strict_dtm_enforcement {on | off}

set transaction_isolation_level {on | off}

set transactional_rpc {on | off}

set textptr_parameters {on | off}

set textsize *value*

set bulk batch size n

set bulk array size n

Usage

- Normally, all outbound RPCs are routed through the Adaptive Server site handler. These RPCs cannot participate in any transactions, and the performance characteristics of routing many RPCs through the site handler may necessitate the use of an alternate method for RPC handling.
- Component Integration Services provides an alternate means of handling outbound RPCs. If cis_rpc_handling is on, outbound RPCs are routed through a Client-Library connection that is persistent through the life of the client's connection to the Adaptive Server. This means that any number of RPCs can be routed through the same connection, without a connect and disconnect between each RPC. This connection is the same connection used by Component Integration Services to handle all interaction with the remote server, including processing of select, insert, delete, and update commands.
- The client application issues set cis_rpc_handling on or off to control whether an outbound RPC is to be routed through the Adaptive Server site handler or through a Component Integration Services connection. If cis_rpc_handling is on, Component Integration Services processes the RPC request; if cis_rpc_handling is off, the site handler processes the RPC.
- When a client application makes a new connection to Adaptive Server, the connection inherits the setting for the configuration parameter cis rpc handling (default is off). This determines the default handling for outbound RPCs.

strict_dtm_enforcement

- If this property is turned ON, then transactions that involve participants that are not DTM-enabled servers are aborted.
- The default is OFF, in which case the behavior of transaction management for remote servers is "best-effort," which is compatible with earlier version behavior.

transaction_isolation_level

- The isolation level state between the local thread and the thread created by the connection to the remote server is maintained. If the isolation level changes, then the state is synchronized between the local and remote servers by sending the appropriate set transaction_isolation_level command.
- This state synchronization is only performed for connections to servers in class ASEnterprise.

transactional_rpc

- Setting transactional_rpc on results in the same behavior as setting cis_rpc_handling on, except that RPCs that are issued outside of a transaction will continue to be routed through the site handler if cis_rpc_handling is off.

textptr_parameters

- This command affects the behavior of parameters to RPCs that are managed by Component Integration Services.
- Any RPC parameter that is of type binary(16) or varbinary(16) is assumed to be a textptr if the setting for textptr_parametrs is on — only if the binary parameter is preceded by a char or varchar parameter that contains a string reference to a table that was the source of the textptr.
- The textptr is expanded into as many chunks of text (chunk size is 16K for remote Adaptive Servers; 32K for other servers) as are needed to contain the full text value, and sent to the remote server as CS_LONGCHAR parameters if the remote server supports the TDS LONGCHAR datatype. If this type is not supported, then the expansion is disabled, and the setting of textptr_parameters is assumed to be off.

See also

set in the *Reference Manual*

setuser

Description	Allows a Database Owner to impersonate another user.
Syntax	setuser ["*user_name*"]
Usage	• The Database Owner uses the setuser command to adopt the identity of another user in order to use another user's database objects. When using Component Integration Services, these objects can be either local or remote. • Component Integration Services processes the setuser command—it does *not* forward the command to the remote server. Component Integration Services drops all current connections that have been made on behalf of the current user. • The setuser command cannot be executed when a transaction is current. • Permissions that are set on a remote server override permissions set by Component Integration Services. Component Integration Services cannot change permissions of a user on a remote server. • Before using the setuser command, the user to be impersonated must have an external login mapped to the remote server. This is set by the sp_addexternlogin system procedure (for more information, see the *Reference Manual*).
See also	setuser in the *Reference Manual*

truncate table

Description	Removes all rows from a table.
Syntax	truncate table [[*database*.]*owner*.]*table_name*
Usage	• Component Integration Services processes the truncate table command when the table on which it operates has been created as a proxy table. • When Component Integration Services forwards the truncate table command to a remote server, the table name used is the remote table name. This name may not be the same as the local proxy table name. Server class *ASEnterprise* Component Integration Services forwards the truncate table command to servers of class ASEnterprise.

Server class *ASAnywhere*

Component Integration Services forwards the truncate table command to servers of class ASAnywhere.

Server class *ASIQ*

Component Integration Services forwards the truncate table command to servers of class ASIQ.

Server class *direct_connect* and *sds*

If the remote server has requested DB2 syntax, the following statement is forwarded:

```
delete from [owner.]table_name
```

Otherwise, Transact-SQL syntax is sent:

```
truncate table [[database.]owner.]table_name
```

Server class *db2*

The following syntax is forwarded to the remote server:

```
delete from [owner.]table_name
```

See also truncate table in the *Reference Manual*

update

Description Changes data in existing rows, either by adding data or by modifying existing data.

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]
```

```
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*

Usage

- Component Integration Services processes the update command when the table on which it operates has been created as a proxy table. Component Integration Services forwards the entire request (or part of it) to the server that owns the actual object.
- The update command specifies the row or rows you want to change, and the new data. The new data can be a constant, an expression, or data pulled from other tables.
- Component Integration Services executes the update command using one of two methods:
  a The entire command is forwarded to the remote server as a single statement in close to its original syntax. If the syntax and remote capabilities match, the entire statement is forwarded and processed remotely. This is referred to as quickpass mode.
  b If the entire command cannot be forwarded to a remote server, Component Integration Services declares and opens one or more cursors in update mode, and begins a scan on the remote table. Each cursor forwards as much of the original statement's predicates to the remote server as possible. For each row fetched that meets the search criteria, a positioned update is executed.
- When Component Integration Services forwards the update command to a remote server, the table name used is the remote table name, and the column names used are the remote column names. These names may not be the same as the local proxy table names.
- Component Integration Services generally passes the original update syntax to remote servers as a single statement, but the following conditions will likely cause the statement to be executed using method 2, above:
  - The statement contains multiple tables that are not located in the same remote server.
  - The statement contains local tables (including temporary tables).
  - The statement contains certain referential integrity checks.
  - The statement contains system functions in the predicate list.
  - The statement contains syntax that the remote server does not support.

- The following keywords are ignored and do not prevent Component Integration Services from using quickpass mode:
  - prefetch
  - index
  - lru | mru
- The format involving where current of is never forwarded to a remote server and causes the statement to be executed using method 2 above.

Server class *ASEnterprise*

- If Component Integration Services cannot pass the entire statement to a remote server, a unique index must exist on the table.
- The update command is fully supported for all datatypes except text and image. text and image data cannot be changed with the update command, except when setting the text or image value to null. Use the writetext command instead.
- If quickpass mode is not used, data is retrieved from the source tables, and the values in the target table are updated using a separate cursor designed for handling a positioned update.

Server class *ASAnywhere*

Handling of the update statement is the same as for ASEnterprise.

Server class *ASIQ*

Handling of the update statement is the same as for ASEnterprise.

Server class *direct_connect*

- The following syntax is supported by servers of class direct_connect:

```
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 search_conditions]
```

  update commands that conform to this syntax use quickpass mode, if the capabilities response from the remote server indicates that all elements of the command are supported. Examples of negotiable capabilities include: subquery support, group by support, and built-in support.

- If the remote server does not support all elements of the command, or the command contains a from clause, Component Integration Services issues a query to obtain the values for the set clause, and then issues an update command to the remote server.
- Component Integration Services passes data values as parameters to either a cursor or a dynamic SQL statement. Language statements can also be used if the DirectConnect supports it. The parameters are in the datatype native to Adaptive Server and must be converted by the DirectConnect into formats appropriate for the target DBMS.

Server class *db2*

- The following syntax is supported by servers of class db2:

```
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 search_conditions]
```

- Servers of class db2 do not contain the capabilities negotiation features of server class direct_connect, so the syntax passed to the remote server is simpler than that allowed by Transact-SQL. The syntax does not contain the following:
  - Search conditions containing subqueries, group by, or order by clauses
  - Transact-SQL built-in functions
  - Transact-SQL operators (such as bitwise operators)
  - Syntax not allowed by DB2

  Component Integration Services processes the update command using method 2, described above, when the statement is complex.
- If the server is a DB2 system, use trace flag 11215 to instruct Component Integration Services that the remote server is capable of handling all DB2 syntax. This assumption is not made automatically because not all gateways using the db2 server class are actually connected to DB2 systems. When trace flag 11215 is turned on, quickpass mode is used unless the following conditions exist:
  - The statement cannot be expressed in DB2 syntax.
  - The statement contains outer joins.
  - The statement contains like clauses with Sybase extensions.

- The statement contains built-in functions that are not supported by DB2.

- When an update statement contains a select statement, Component Integration Services issues a query to obtain the values for the set clause, and then issues an update command to the remote server, unless trace flag 11215 is enabled.
- When an update statement contains a from clause, Component Integration Services issues a query to obtain the values for the set clause, and then issues an update command to the remote server.

See also update in the *Reference Manual*

update statistics

Description Updates information about the distribution of key values in specified indexes. Also updates row count information.

Syntax update statistics *table_name* [*index_name*]

Usage

- When the update statistics command is issued against a proxy table, Component Integration Services provides meaningful statistics on the remote table and the given index or on all indexes if no index is specified. The results are used to construct a distribution page for each index. This distribution page is stored in the database. When a new distribution page is created for an index, any previous distribution page for that index is freed.
- Using update statistics, Component Integration Services creates extremely accurate distribution statistics for remote tables. This information is used to determine the optimal join order, giving Component Integration Services the ability to generate optimal queries against remote databases which may not support cost-based query optimization.
- When Component Integration Services forwards the command to a remote server, the table name used is the remote table name, and the column names used are the remote column names. These names may not be the same as the local proxy table names.
- Obtaining information on an index, and especially on a number of indexes, can be time-consuming on large tables. Trace flag 11209 can be used to indicate that update statistics is to obtain row count only. When this flag is on, previous distribution pages for indexes are not replaced.

- Component Integration Services retrieves row count information even if no indexes exist.

Server class *ASEnterprise*

- If the table on which the statistics are requested has no indexes, Component Integration Services issues the following command:

  ```
  select count(*) from table_name
  ```

  It is also the only command issued when trace flag 11209 is on.

- If the table has an index and the index is specified in the command, Component Integration Services issues the following commands:

  ```
  select count(*) from table_name

  select count(*) column_name [,column_name, ...]
   from table_name
   group by column_name [,column_name, ..]
  ```

  The column name(s) represent the column or columns that make up the index.

  For example, when the following command is issued:

  ```
  update statistics customers ind_name
  ```

  Component Integration Services issues:

  ```
  select count(*) from customers

  select count(*) last_name, first_name
   from customers
   group by last_name, first_name
  ```

- If the table has one or more indexes but no index is specified in the statement, Component Integration Services issues the select count (*) once, and the select/order by commands for each index.

Server class *ASAnywhere*

The processing of update statistics in this server class is identical to that of server class ASEnterprise described above.

Server class *ASIQ*

The processing of update statistics in this server class is identical to that of server class ASEnterprise described above.

Server class *direct_connect*

- The processing of update statistics in this server class is identical to that of server class ASEnterprise described above.

- If the direct_connect indicates that is cannot handle the group by or the count(*) syntax, statistics are not collected for the direct_connect.

Server class *db2*

The processing of update statistics in server class db2 is identical to that of server class ASEnterprise described above.

See also update statistics in the *Reference Manual*

writetext

Description Permits non-logged, interactive updating of an existing text or image column.

Syntax writetext [[*database*.]*owner*.]*table_name*.*column_name* *text_pointer* [with log] *data*

Usage

- Component Integration Services processes the writetext command when the table on which it operates has been created as a proxy table.
- If the remote server referenced by the proxy table does not support text pointers, writetext is not supported.
- To process the writetext command, Component Integration Services issues the following ClientLibrary commands using the connection established to the remote server:

```
ct_command(command, CS_SEND_DATA_CMD, NULL,
 CS_UNUSED, CS_COLUMN_DATA);
ct_data_info(command, CS_SET, CS_UNUSED, iodesc)
ct_send_data(command, (CS_VOID *) start, length)
```

Server class *ASEnterprise*

The writetext command is processed using a separate connection to the remote server.

Server class *ASAnywhere*

The writetext command is processed using a separate connection to the remote server.

Server class *ASIQ*

The writetext command is processed using a separate connection to the remote server.

Server class *direct_connect*

If the DirectConnect supports text pointers, Component Integration Services treats the DirectConnect as if it were a server in class ASEnterprise.

Server class *db2*

- writetext is not supported for tables owned by servers in this class.

See also

writetext in the *Reference Manual*

APPENDIX A

Tutorial

This chapter provides a tutorial for setting up Component Integration Services and accessing a remote server.

Getting started with Component Integration Services

This section provides a step-by-step guide to configuring the server to access remote data sources. It includes instructions for:

- Adding a remote server
- Mapping remote objects to local proxy tables
- Performing joins between remote tables

Routine system administration tasks such as starting and stopping Adaptive Server, creating logins, creating groups, adding users, granting permissions, and password administration are explained in the Adaptive Server documentation.

Adding a remote server

You can use the server to access data on remote servers. Before you can do this, you must configure Component Integration Services.

Overview

1 Add the remote server to the interfaces file.

2 Add the name, server class, and network name of the remote server to system tables.

3 Optionally, assign an alternate login name and password.

Adding the remote server to the interfaces file

Use the dsedit or dscp utility to edit the interfaces file located in the *$SYBASE* directory:

- In UNIX, the interfaces file is called *interfaces.*
- In Windows NT, the interfaces file is called *sql.ini*.

For a complete discussion of the interfaces file, see the *Adaptive Server Configuration Guide* for your platform.

Creating server entries in system tables

Use sp_addserver to add entries to the sysservers table. sp_addserver creates entries for the local server and an entry for each remote server that is to be called. The sp_addserver syntax is:

sp_addserver *server_name [,server_class [,network_name]]*

where:

- *server_name* is the name used to identify the server. It must be unique.
- *server_class* is one of the supported server classes. The default value is ASEnterprise. If server_class is set to local, *network_name* is ignored.
- *network_name* is the server name in the interfaces file. This name may be the same as *server_name*, or it may differ. The *network_name* is sometimes referred to as the *physical name*.

Example

The following examples create entries for the local server named DOCS and for the remote server CTOSDEMO with server class ASEnterprise.

```
sp_addserver DOCS, local
sp_addserver CTOSDEMO, ASEnterprise, CTOSDEMO
```

Adding an alternate login and password

Use sp_addexternlogin to assign an alternate login name and password to be used when communicating with a remote server. This step is optional. The syntax for sp_addexternlogin is:

sp_addexternlogin *remote_server*, *login_name*, *remote_name* [, *remote_password*]

where:

- *remote_server* is the name of the remote server. The *remote_server* must be known to the local server by an entry in the master.dbo.sysservers table.

- *login_name* is an account known to the local server. *login_name* must be represented by an entry in the master.dbo.syslogins table. The "sa" account, the "sso" account, and the *login_name* account are the only users authorized to modify remote access for a given local user.
- *remote_name* is an account known to the *remote_server* and must be a valid account on the node where the *remote_server* runs. This is the account used for logging in to the *remote_server*.
- *remote_password* is the password for *remote_name.*

Examples

```
sp_addexternlogin FRED, sa, system, sys_pass
```

Allows the local server to gain access to remote server FRED using the remote name "system" and the remote password "sys_pass" on behalf of user "sa".

```
sp_addexternlogin OMNI1012, bobj, jordan, hitchpost
```

Tells the local server that when the login name "bobj" logs in, access to the remote server OMNI1012 is by the remote name "jordan" and the remote password "hitchpost". Only the "bobj" account, the "sa" account, and the "sso" account have the authority to add or modify a remote login for the login name "bobj".

Verifying connectivity

Use the connect to *server_name* command to verify that the configuration is correct. connect to requires that "sa" explicitly grant connect authority to users other than "sa." The connect to command establishes a passthrough mode connection to the remote server. This passthrough mode remains in effect until you issue a disconnect command.

Join between two remote tables

With Component Integration Services, you can perform joins across remote tables.

Overview

1 Add the remote servers.

2 Define each remote server.

3 Map the remote tables to the server.

4 Perform the join.

Adding the remote servers to the interfaces file

Edit the interfaces file using dsedit.

Defining the remote servers

Use sp_addserver to add entries to the sysservers system table. On the server originating the call, there must be an entry for each remote server that is to be called. The sp_addserver syntax is:

sp_addserver *server_name [,server_class] [,network_name]*

where:

- *server_name* is the name used to identify the server. It must be unique.
- *server_class* is one of the supported server classes. The default value is sql_server. If the value is local, *network_name* is ignored.
- *network_name* is the server name in the interfaces file. This name may be the same as the *server_name* specification, or it may be different. If *network_name* is not provided, the default value is the *server_name*.

Example

The following examples create entries for the local server named DOCS and for the remote server SYBASE of class ASEnterprise.

```
sp_addserver DOCS, local
sp_addserver CTOSDEMO, ASEnterprise, SYBASE
```

Mapping the remote tables to Adaptive Server

create existing table enables the definition of existing (proxy) tables. The syntax for this option is similar to the create table command and reads as follows:

create proxy_table
    table_name
    at “pathname”

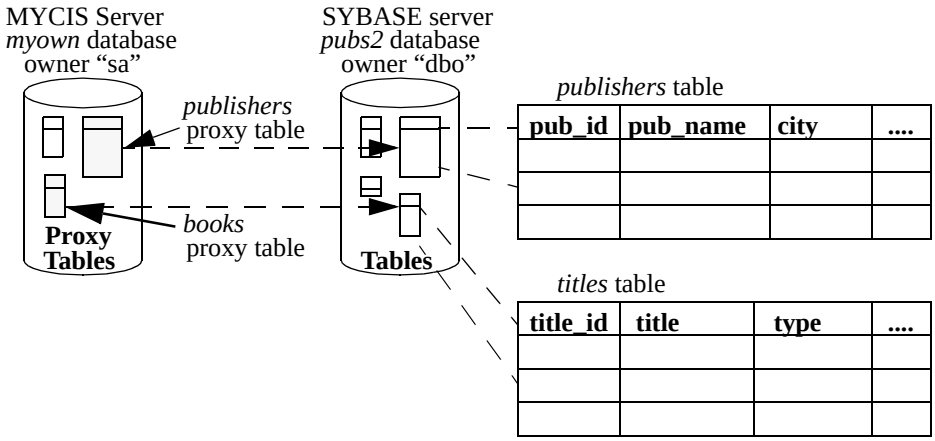
When the server processes this command, it does not create a new table. Instead, it checks the table mapping and verifies the existence of the underlying object. If the object does not exist (either host data file or remote server object), the server rejects the command and returns an error message to the client.

After you define an existing table, issue an update statistics command for that table. This helps the query optimizer make intelligent choices regarding index selection and join order.

Example

Figure A-1 illustrates the remote Adaptive Server tables publishers and titles in the sample pubs2 database mapped to a local server.

Figure A-1: Defining remote tables in a local server



Mapping the remote tables

The steps required to produce the mapping illustrated above are as follows:

1 Define a server named SYBASE. Its server class is ASEnterprise, and its name in the interfaces file is SYBASE:

```
exec sp_addserver SYBASE, ASEnterprise, SYBASE
```

2 Define a remote login alias. This step is optional. User "sa" is known to remote server SYBASE as user "sa," password "timothy":

```
exec sp_addexternlogin SYBASE, sa, sa, timothy
```

3 Define the remote publishers table:

```
create proxy_table publishers
at "SYBASE.pubs2.dbo.publishers"
```

4 Define the remote titles table:

```
create Proxy_table titles
```

```
        at "SYBASE.pubs2.dbo.titles"
```

Performing the join

Use the select statement to perform the join.

```
select Publisher = p.pubname, Title = t.title
 from publishers p, titles t
 where p.pub_id = t.pub_id
 order by p.pub_name
```

APPENDIX B

Troubleshooting

This appendix provides troubleshooting tips for problems that you may encounter when using Component Integration Services. The purpose of this chapter is to provide:

- Enough information about certain error conditions so that you can resolve problems without help from Technical Support
- Lists of information that you can gather before calling Technical Support, which may help resolve your problem more quickly
- You with a greater understanding of Component Integration Services

The *Troubleshooting and Error Messages Guide* should also be used for troubleshooting. While this appendix provides troubleshooting tips for most frequently asked Component Integration Services questions, lists all error messages with a one-line recovery procedure; the *Troubleshooting and Error Messages Guide* provides tips on Adaptive Server problems that are not specific to Component Integration Services.

Problems accessing Component Integration Services

If you issue a command that accesses a remote object and Component Integration Services is not found, the following error message appears:

```
cis extension not enabled or installed
```

- Verify that the enable cis configuration parameter is set to 1 by running:

  ```
  sp_configure "enable cis"
  ```

  sp_configure returns the following row for the enable cis parameter:

```
name           min  max   config value   run value
enable cis     0    1     1              1
```

Both “config value” and “run value” should be 1. If both values are 0, set the enable cis configuration parameter to 1, and restart the server. Use the syntax:

```
sp_configure "enable cis" 1
```

If “config value” is 1 and “run value” is 0, the enable cis configuration parameter is set, but does not take effect until the server is restarted.

Note Component Integration Services is enabled by default beginning with Adaptive Server version 12.0.

Problems using Component Integration Services

This section provides tips on how to correct problems you may encounter when using Component Integration Services.

Unable to access remote server

When you cannot access a remote server, the following error message is returned:

```
11206 Unable to connect to server server_name.
```

The message is preceded by one of the following Client-Library messages:

```
Requested server name not found
Driver call to connect two endpoints failed
Login failed
```

The Client-Library message indicates why you cannot access the remote server as described in the following sections.

Requested server name not found

The server is not defined in the interfaces file when the following messages display:

```
Requested server name not found
11206 Unable to connect to server server_name.
```

When a remote server is added using sp_addserver, the interfaces file is not checked. It is checked the first time you try to make a connection to the remote server. To correct this problem, add the remote server to the interfaces file that is being used by Component Integration Services.

Driver call to connect two endpoints failed

If the remote server is defined in the interfaces file, but no response was received from the connect request, the following messages are displayed:

```
Driver call to connect two endpoints failed
11206 Unable to connect to server server_name.
```

- Verify that your environment is set up correctly.

  To test this, try to connect directly to the remote server using isql or a similar tool:

  a Log in to the machine where Component Integration Services is running.

  b Set the SYBASE environment variable to the same location that was used when Component Integration Services was started. Component Integration Services uses the interfaces file in the directory specified by the SYBASE environment variable, unless it is overridden in the *runserver* file by the -i argument.

  Note These first two steps are important to ensure that the test environment is the same environment that Component Integration Services was using when you could not connect to the remote server.

  c Use isql or a similar tool to connect directly to the remote server.

  If the environment is set up correctly and the connection fails, continue through this list. If the connection is made, there is a problem with the environment being used by Component Integration Services.

- Verify that the remote server is up and running.

  Log in to the machine where the remote server is located to verify that the server is running. If the server is running, continue through this list. If the server is not running, restart the server and try your query again.

- Verify that the entry for the remote server in the interfaces file is correct:

- Verify that the machine name is the correct name for the machine the software is loaded on.
- Verify that if the interfaces file is a text file, the query and master lines start with a tab and not spaces.
- Verify that the port number is available. Check the *services* file in the */etc* directory to ensure that the port number is not reserved for another process.

Login failed

If you can access the remote server, but the login name and password are correct, the following messages display:

```
Login failed
11206 Unable to connect to server server_name.
```

See if there is an external login established for the remote server by executing:

```
exec sp_helpexternlogin server_name
```

If no external login is defined, Component Integration Services uses the user login name and password that was used to connect to Adaptive Server. For example, if the user connected to Adaptive Server using the "sa" account, Component Integration Services uses the login name "sa" when making a remote connection. Unless the remote server is another Adaptive Server, the "sa" account probably does not exist, and an external login must be added using sp_addexternlogin.

If an external login is defined, verify that the user's login name is correct. Remote server logins are case sensitive; for example, DB2 logins are all uppercase. Is the case correct for the user login name you are using and the entry in *externlogins*?

If the login name is correct, the password might be incorrect. You cannot display the password. If the user login name is incorrect or if the password might be incorrect, drop the existing external login and redefine it by executing the commands:

```
exec sp_dropexternlogin server_name, login_name
go
 exec sp_addexternlogin server_name, login_name,
remote_login, remote_password
go
```

Unable to access remote object

When you are unable to access a remote object, the following error message appears:

```
Error 11214  Remote object object does not exist.
```

The problem may be in the local proxy table definition or in the table itself on the remote server.

Verify that:

- The object has been defined in Component Integration Services.

  To confirm, run:

  ```
  sp_help object_name
  ```

  If the object does not exist, create the object in Component Integration Services.

- If the object has been defined in Component Integration Services, the definition is correct.

  Table names can have four parts with the format *server.dbname.owner.tablename*. The *dbname* part is not valid for DB2, Oracle, or InfoHUB servers.

  If the object definition is incorrect, delete it using sp_dropobjectdef, and define it correctly using sp_addobjectdef.

- If the local object definition is correct, check the table on the remote server, to verify that:
  - Permissions are set to allow access to both the database and table.
  - The database has been marked suspect.
  - The database is available.
  - You can access the remote table using a native tool (for example, SQL*Plus on Oracle).

Problem retrieving data from remote objects

When you receive error messages pertaining to mismatches in remote objects, the Component Integration Services object definition does not match the remote object definition. This happens if:

- The object definition was altered outside of Component Integration Services.
- An index was added or dropped outside of Component Integration Services.

Object is altered outside Component Integration Services

Once an object is defined in Component Integration Services, alterations made to an object at the remote server are not made to the local proxy object definition. If an object is altered outside of Component Integration Services, the steps to correct the problem differ, depending on whether create existing table or create table was used to define the object.

To determine which method was used to define the object, run:

```
sp_help object_name
```

If the object was defined via create existing table, the following message is returned in the result set:

```
Object existed prior to CIS.
```

If this message is not displayed, the object was defined via create table.

If create existing table was used to create the table in Component Integration Services:

1 Use drop table in Component Integration Services.

2 Create the table again in Component Integration Services using create existing table. This creates the table using the new version of the table on the remote server.

If the table was created in Component Integration Services using create table, you will drop the remote object when you use drop table. To prevent this, follow these steps:

1 Rename the table on the remote server so the table is not deleted when you use drop table.

2 Create a table on the remote server using the original name.

3 Use drop table in Component Integration Services to drop the table in Component Integration Services and on the remote server.

4 Rename the saved table in step 1 with its original name on the remote server.

5 Create the table again in Component Integration Services using create existing table.

Warning! Do not use drop table in Component Integration Services before renaming the table on the remote server, or you will delete the table on the remote server.

A good rule to follow is to create the object on the remote server, and then execute create existing table to create the object in Component Integration Services. This enables you to correct mismatch problems with fewer steps and with no chance of deleting objects on the remote server.

Index is added or dropped outside Component Integration Services

Component Integration Services is unaware of indexes that are added or dropped outside Component Integration Services. Verify that the indexes used by Component Integration Services are the same as the indexes used on the remote server. Use sp_help to see the indexes used by Component Integration Services. Use the appropriate command on your remote server to verify the indexes used by the remote server. For example, you can use describe with an Oracle server or select * from syscolumns, sysindexes for a DB2 server.

If the indexes are not the same, the steps to correct the problem differ, depending on whether create existing table or create table was used to define the object.

To determine which method was used to define the object, run:

```
sp_help object_name
```

If the object was defined via the create existing table command, the following message is returned in the result set:

```
Object existed prior to CIS.
```

If this message is not displayed, the object was defined via create table.

If create existing table was used to create the object:

1 Use drop table in Component Integration Services.

2 Re-create the table in Component Integration Services using create existing table. This will update the indexes to match the indexes on the remote table.

If create table was used to create the object:

1 Use drop index to drop the index from the remote table.

2 Re-create the index in Component Integration Services using create index. This creates the index in Component Integration Services and the remote server.

If create table was used to define the object, an alternative method is to turn on trace flag 11208. This trace flag prevents create index from transmitting to the remote server. To use trace flag 11208, follow these steps:

1 Turn on trace flag 11208:

```
dbcc traceon(11208)
```

2 Create the index using create index.

3 Turn off trace flag 11208:

```
dbcc traceoff(11208)
```

If you need help

If you encounter a problem that you cannot resolve using the manuals, ask the designated person at your site to contact Sybase Technical Support. Gather the following information prior to calling Technical Support to help resolve your problem more quickly.

- If a problem occurs while you are trying to access remote data, execute the same script against a local table. If the problem does not exist on the local table, it is specific to Component Integration Services and you should continue through this list.
- Find out what version of Component Integration Services you are using:

```
select @@cis_version
```

- Note the SQL script that reproduces the problem. Include the script that was used to create the tables.
- Find the processing plan for your query. This is generated using set showplan. An example of this is:

```
set showplan, noexec on
 go
 select au_lname, au_fname from authors
 where au_id = 'A1374065371'
 go
```

The output for this query looks like this:

```
STEP1
 The type of query is SELECT.
 FROM TABLE
 authors
 Nested iteration
 Using Clustered Index
```

The noexec option compiles the query, but does not execute it. No subsequent commands are executed until noexec is turned off.

- Obtain the event logging when executing the query by turning on trace flags 11201 – 11205. These trace flags log the following:
  - 11201 – client connect, disconnect, and attention events.
  - 11202 – client language, cursor declare, dynamic prepare, and dynamic execute-immediate text.
  - 11203 – client RPC events.
  - 11204 – messages routed to client.
  - 11205 – interaction with remote servers.

  After executing the script with the trace flags turned on, the logging is found in the error log in the *$SYBASE/install* directory. For example:

```
dbcc traceon (11201,11202,11203,11204,11205)
 go
 select au_lname, au_fname from authors
 where au_id = 'A1374065371'
 go
 dbcc traceoff (11201,11202,11203,11204,11205)
 go
```

  The error log output is as follows (the timestamps printed at the beginning of each entry have been removed to improve legibility):

```
server  LANGUAGE, spid 1: command text:
 select au_lname, au_fname from authors where au_id
= 'A1374065371'
 server  SIGDISABLE, spid 1: signals disabled on
endpoint 10
 server  RMT_CONNECT, spid 1: connected to server
'SYBASE', using language/charset
'us_english.iso_1', packet size 512
 server  SYB_TSCN, spid 1, server SYBASE:
 SELECT au_id, au_lname, au_fname FROM
```

```
pubs2.dbo.authors WHERE au_id = "A1374065371"
 server  OMNIENDS, spid 1: closing cursor 'O1_16'
 server  OMNICLOS, spid 1: deallocating cursor
'O1_16', type CONNECTION.
```

This tracing is global, so once the trace flags are turned on, any query that is executed is logged; therefore, turn tracing off once you have your log. Also, clean out the error log periodically by bringing the server down, renaming the error log, and restarting the server. This creates a new error log.

Index

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

V

W