



OmniConnect™
Component Integration Services User's Guide for
Sybase® Adaptive Server™ Enterprise and OmniConnect

Adaptive Server Enterprise Version 12

Document ID: 32702-01-1200-01

Last revised: October 1999

Copyright © 1989-1999 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase database management software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase, the Sybase logo, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-FORMS, APT-Translator, APT-Library, Backup Server, ClearConnect, Client-Library, Client Services, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, E-Anywhere, E-Whatever, Embedded SQL, EMS, Enterprise Application Server, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Gateway Manager, ImpactNow, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, MainframeConnect, Maintenance Express, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaWorks, MySupport, Net-Gateway, Net-Library, NetImpact, ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Client, Open ClientConnect, Open Client/Server, Open Client/Server Interfaces, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, PowerStage, PowerStudio, PowerTips, Powersoft Portfolio, Powersoft Professional, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, Report Workbench, Report-Execute, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Resource Manager, RW-DisplayLib, RW-Library, S Designer, S-Designer, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL SMART, SQL Toolset, SQL Server/CFT, SQL Server/DBM, SQL Server SNMP SubAgent, SQL Station, SQLJ, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Financial Server, Sybase Gateways, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase User Workbench, SybaseWare, Syber Financial, SyberAssist, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, Transact-SQL, Translation Toolkit, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Viewer, Visual Components, VisualSpeller, VisualWriter, VQL, WarehouseArchitect, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server and XP Server are trademarks of Sybase, Inc. 9/99

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., 6475 Christie Avenue, Emeryville, CA 94608.

Contents

About This Book	vii	
CHAPTER 1	Introduction	1
	New Features in Adaptive Serve Enterprise 12.x.....	3
	Who Can Use Component Integration Services.....	4
	Steps Needed to Use Component Integration Services.....	5
CHAPTER 2	Understanding Component Integration Services	7
	Basic Concepts	8
	Remote Table Access	8
	Access Methods	8
	Server Classes	8
	Object Types	9
	Interface to Remote Servers	10
	Topics and Issues	15
	Using the create existing table Command.....	15
	auto identity Option	17
	Passthrough Mode	17
	Transaction Management.....	22
	Remote Procedures as Proxy Tables.....	27
	text and image Datatypes.....	30
	Internal Operations.....	35
	Connection Management	35
	Query Processing	35
	Query Plan Execution.....	42
	Triggers	45
	Referential Integrity	46
	Security Issues	46
	Trusted Mode	46
	Adaptive Server Features and Remote Data Access.....	47
	ANSI Joins.....	47
	50-Table Join Limit	48
	Remote Server Logins.....	48

	Adaptive Server Features Not Supported by CIS	49
	Parallel Processing.....	49
	External Security	49
CHAPTER 3	Using Component Integration Services	51
	Getting Started with Component Integration Services	52
	Adding a Remote Server	52
	Mapping Remote Objects to Local Proxy Tables	54
	Join Between Two Remote Tables.....	56
	Configuration and Tuning.....	60
	Using sp_configure.....	60
	Global Variables for Status.....	64
	Java in the Database	65
	Quickpass.....	65
	@@textsize	65
	Constraints on Java Class Columns	65
	Error Messages	66
	RPC Handling and Component Integration Services	67
	Site Handler Handling Outbound RPCs	67
	Component Integration Services Handling Outbound RPCs...	68
	Sending Text as RPC Parameters	69
	Sending Large Data Chunks	70
	dbcc Commands	74
	dbcc Options	74
	Trace Flags	75
	Using update statistics	76
	Shared Memory Requirements	78
	Additional Component Integration Services Memory	
	Requirements	79
	Backing Up Your System	80
CHAPTER 4	Server Classes	81
	Defining Remote Servers	82
	Server Class ASEnterprise.....	82
	Server Class ASAnywhere	82
	Server Class ASIQ	82
	Server Class sql_server	83
	Server Class db2.....	83
	Server Class direct_connect	83
	Server Class sds	84
	Server Class generic	85
	Datatype Conversions	86
	Remote Server Capabilities	87

Transact-SQL Commands	88
alter database	90
alter table	92
begin transaction.....	97
case.....	100
close.....	102
commit transaction	103
connect to...disconnect	106
create database	108
create existing table	110
create index.....	119
create proxy_table.....	121
create table	122
create trigger	126
deallocate cursor.....	128
declare cursor	129
delete	130
drop database	134
drop index	135
drop table	137
execute.....	139
fetch	140
Functions.....	142
insert	147
open	149
prepare transaction	150
readtext	152
rollback transaction	154
select.....	156
set	161
setuser	163
truncate table	164
update	165
update statistics	170
writetext.....	172
APPENDIX A Troubleshooting.....	173
Problems Accessing Component Integration Services	174
Problems Using Component Integration Services	175
Unable to Access Remote Server	175
Unable to Access Remote Object	177
Problem Retrieving Data From Remote Objects	178
If You Need Help.....	182

About This Book

Audience

This book is written for Sybase® Adaptive Server™ Enterprise and OmniConnect™ 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, “Using Component Integration Services,” includes a tutorial designed to help new users get Component Integration Services up and running, and provides configuration and tuning information.
- Chapter 4, “Server Classes,” describes the Component Integration Services server classes required to access remote databases.
- Appendix A, “Troubleshooting,” provides troubleshooting tips if you encounter a problem with Component Integration Services.

Adaptive Server Enterprise Documents

The following documents comprise the Sybase Adaptive Server Enterprise documentation:

- 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 SyBooks™-on-the-Web.

- The Adaptive Server installation documentation for your platform – describes installation and upgrade procedures for all Adaptive Server and related Sybase products.
- The Adaptive Server configuration documentation for your platform – describes configuring a server, creating network connections, configuring for optional functionality, such as auditing, installing most optional system databases, and performing operating system administration tasks.
- *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.
- *System Administration Guide* – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources and user and system databases, and specifying character conversion, international language, and sort order settings.
- *Adaptive Server Reference Manual* – contains detailed information about all Transact-SQL commands, functions, procedures, and datatypes. This manual also contains a list of the Transact-SQL reserved words and definitions of system tables.
- *Performance and Tuning Guide* – explains how to tune Adaptive Server for maximum performance. This manual includes information about database design issues that affect performance, query optimization, how to tune Adaptive Server for very large databases, disk and cache issues, and the effects of locking and cursors on performance.
- The *Utility Programs* manual for your platform – documents the Adaptive Server utility programs, such as **isql** and **bcp**, which are executed at the operating system level.
- *Security Features User's Guide* – provides instructions and guidelines for using the security options provided in Adaptive Server from the perspective of the non-administrative user.

- *Error Messages and Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
- *Component Integration Services User's Guide for Adaptive Server Enterprise and OmniConnect* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- *Adaptive Server Glossary* – defines technical terms used in the Adaptive Server documentation.

Other Sources of Information

Use the SyBooks™ and SyBooks-on-the-Web online resources to learn more about your product:

- SyBooks documentation is on the CD that comes with your software. The DynaText browser, also included on the CD, allows you to access technical information about your product in an easy-to-use format.

Refer to *Installing SyBooks* in your documentation package for instructions on installing and starting SyBooks.

- SyBooks-on-the-Web is an HTML version of SyBooks that you can access using a standard Web browser.

To use SyBooks-on-the-Web, go to <http://www.sybase.com>, and choose Documentation.

Conventions

What you type to the computer screen is shown as:

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

Computer output is shown as:

```
OmniConnect 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, ask a designated person at your site to contact Sybase Technical Support.

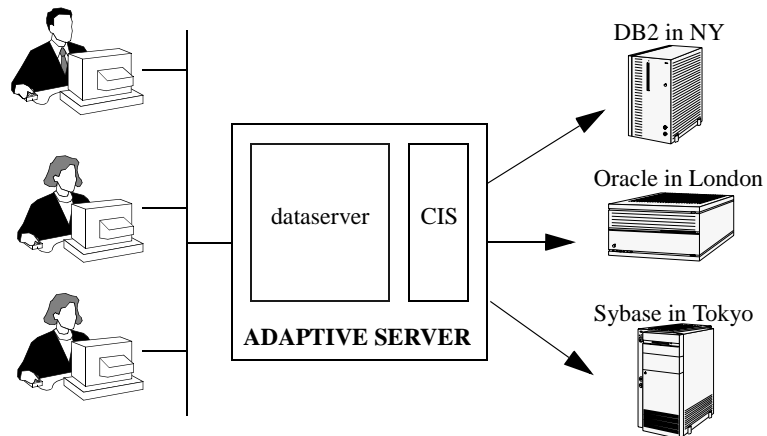
Introduction

Component Integration Services is a feature that extends Adaptive Server capabilities and provides enhanced interoperability. It is the core interoperability feature of OmniConnect.

Component Integration Services allows Adaptive Server and OmniConnect to present a uniform view of enterprise data to client applications and provides location transparency to enterprise-wide data sources.

Component Integration Services allows users to access both Sybase and non-Sybase databases on different servers. These external data sources include host data files and tables, views and RPCs (remote procedure calls) in database systems such as Adaptive Server, Oracle, and DB2, as shown in Figure 1-1.

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.
 - Provide applications, such as PowerBuilder®, Microsoft Access, and DataEase, with transparent access to heterogeneous data.
 - Maintain referential integrity across heterogeneous data sources.
 - Access native remote server capabilities using the Component Integration Services passthrough mode.

New Features in Adaptive Serve Enterprise 12.x

The Adaptive Server 12.x release includes these new Component Integration Services (CIS) features:

- New dbcc commands – **dbcc cis showcaps** and **dbcc cis setcaps** allow examination and modification of server capabilities.
- Proxy databases – A proxy database is created containing proxy tables. Metadata for proxy tables is imported automatically from the remote location which contains the actual tables. This metadata is then used to create proxy tables within the proxy database.
- Synchronizing proxy tables – Proxy tables in a proxy database can be resynchronized using the **alter database** command.
- Java in the database
- Support for ANSI joins
- 50-table join limit
- New global variables
- Support for remote server logins
- Two-phase commit transaction management
- Support for **execute immediate**
- New set commands
- System proxy databases in support of high availability clusters
- **create table** syntax
- **create existing table** syntax
- **create proxy_table**
- RPC's as proxy tables

Who Can Use Component Integration Services

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™ products, provides transparent access to a wide variety of data sources, including:

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

The list of certified and supported sources and front-end tools is increasing. For current information on all data sources, versions supported, and products required for support, please call the Sybase FAX on Demand at 1-800-423-8737. Request the “Partner Certification Report.”

Steps Needed to Use Component Integration Services

To get Component Integration Services running:

- Make sure you have installed the Component Integration Services or the OmniConnect option.
- Issue the command **sp_configure "enable cis", 1** (This is the default for Adaptive Server Enterprise 12.)
- Restart Adaptive Server
- Install DirectConnect server(s) or gateways for the external data sources you choose to access (Oracle, Ingres, DB2, Informix, Rdb).
- Configure the server to access remote objects as described in Chapter 3, "Using Component Integration Services."

Steps Needed to Use Component Integration Services

Understanding Component Integration Services

This chapter discusses some of the essential features of Component Integration Services. It is intended to help you understand how Adaptive Server works with the Component Integration Services option configured. The chapter includes the following topics:

Name	Page
Basic Concepts	8
Topics and Issues	15
Internal Operations	35
Adaptive Server Features Not Supported by CIS	49

Basic Concepts

Before accessing remote tables with Component Integration Services, you must have a valid *interfaces* file (or *sql.ini* file if you are using Windows NT). For information on setting up an interfaces file, see the configuration documentation for your platform.

Remote Table Access

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. 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, which means remote tables appear as local tables to the client, tables must first be mapped to their corresponding external locations. This mapping is performed by means of stored procedures. See the *Adaptive Server Reference Manual* for more information on stored procedures.

Access Methods

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

Server Classes

A server class must be assigned to each server when it is added by means of the system procedure **sp_addserver**. There are seven server classes, each of which specifies the access method used to interact with the remote server. The server classes are:

- *ASEnterprise* – Used if the server is an Adaptive Server version 11.5 or later. This is the default server class.
- *ASAnywhere* – Used if the server is an Adaptive Server Anywhere version 6.0 or later.
- *ASIQ* – Used if the server is an Adaptive Server version 12.0 or later.
- *sql_server* – indicates that the server is a Sybase SQL Server™ or an Adaptive Server or a Microsoft SQL Server. Component Integration Services determines whether the Sybase server is a release 10.0 or later server (supports cursors and dynamic SQL) or a pre-release 10.0 server (does not support cursors or dynamic SQL).
- *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.
- *access_server* – a synonym for server class *direct_connect* for compatibility with previous releases.
- *db2* – indicates that the server is a gateway to DB2 or DB2-compatible databases. Net-Gateway™ and Database Gateway for DB2, AS/400, InfoHub, and SQL/DS fall into this category.
- *generic* – indicates that the server is an Open Server application that conforms to the interface requirements of a Generic Access Module.
- *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.
- *rpc* – The object in a remote server of any class is an RPC. Component Integration Services treats the result set from the RPC as a read-only table.

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* (*access_server*), a number of features such as cursor and dynamic requests are used. These features are not used by a server of class *generic*.

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

- Remote server addition to the *interfaces* file (or *sql.ini* file if you are using Windows NT)
- Remote server definition
- Remote server login information
- Remote object definition

Remote Server Definition

Remote servers are defined by means of the stored procedure **sp_addserver**. This procedure is documented in the *Adaptive Server Reference Manual*. When using release 12.x you must name the local server.

Logging into Remote Servers

Once the remote server has been configured, login information must be provided. By default, the server uses the names and passwords of its clients whenever it connects to a remote server on behalf of those clients. However, this default can be overridden by the use of the stored procedure **sp_addexternlogin**. This procedure 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 a remote server has been properly configured, objects in that remote server cannot be accessed 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.

You can use one of two methods for defining the storage location of remote objects:

- 1 Define the storage location of individual objects
- 2 Define the default location of all objects in a database

Defining the Storage Location of Individual Objects

Defining individual object storage locations is done by means of the system procedure **sp_addobjectdef** or the use of the **at pathname** syntax. This procedure allows you to associate a remote object with a local proxy table name. The remote object may or may not exist before you do the mapping. Complete syntax for **sp_addobjectdef** is provided in the *Adaptive Server Reference Manual*.

Defining the Default Storage Location for Tables

Defining the default storage location for all tables in a given database is done by means of the stored procedure **sp_defaultloc**. This procedure establishes the location of tables that are to be created for a given database. The remote objects may or may not already exist. You can override **sp_defaultloc** with **sp_addobjectdef** for individual tables. **sp_defaultloc** can not be used when local tables are required. The syntax for **sp_defaultloc** is provided in the *Adaptive Server Reference Manual*.

Using the *create [existing] table* Command

Once you have defined the storage location, you can create the table as a new or existing object. If the table does not already exist at the remote location, use the **create table** syntax. If it already exists, use the **create existing table** syntax. If the object type is *rpc*, only the **create existing table** syntax is allowed.

When a **create existing table** statement is received, and the object type is either *table* or *view*, the existence of the remote object is checked by means of the catalog stored procedure **sp_tables**. If the object exists, then its column and index attributes are obtained. Column attributes are compared with those defined for the object in the **create existing table** statement. Column name, type, length, and null property are checked. Index attributes are added to the *sysindexes* system table.

Once the object has been created, either as a new or an existing object, the remote object can be queried by using its local name.

Using Create Proxy Database

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

To create a proxy database use the **create database** command:

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

Use the clause **with default_location** to specify the storage location of any new tables. Use the **for proxy_update** to specify the location from which metadata is imported for automatic proxy table creation. The **for proxy_update** clause establishes the database as a proxy database; the **with default_location** clause defines the location from which proxy tables are imported. Without the **for proxy_update** clause, the behavior of the **with default_location** clause is the same as that provided by the stored procedure **sp_defaultloc**. With **sp_defaultloc** a default storage location is established for new and existing table creation, but automatic import of proxy table definitions is not performed during the processing of the **create database** command.

The value of pathname is a string identifier in the following format:

```
servername.dbname.owner.
```

Note: the dots are significant, and all three must be present! Each field in this string is described as follows:

- *servername* - required field; represents the name of the server that owns the objects to be referenced by proxy tables. Must exist in *master.dbo.sys.servers.srvname*.
- *dbname* - may be omitted. The name of the database within *servername* which contains objects to be referenced by proxy tables
- *owner* - may be omitted. The name of the owner of objects to be referenced by proxy tables. This may be restrictive, such that if more than one user owns objects in *dbname*, specifying the owner will select only those objects owned by that user. Proxy tables will not be created for objects owned by other users.

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

When a proxy database is created (using the **for proxy_update** option), CIS functions will be 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 will be provided in terms of the number of database pages needed to contain all proxy tables and indexes. This size will be used if no size is specified, and no database devices are specified.

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 will not be estimated; it is assumed in this case that the user or dba wants to override the default size calculated for the proxy database.

- 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 the proxy tables are created, the **quoted identifier** state is turned on, and each table is created with quotes surrounding the table name and column name. This allows the creation of tables containing names that may be Sybase 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
- **grant create table** permission to *public*
- The database status will be set to indicate that this database is a user proxy database. This is done by setting a status field in *master.dbo.sysdatabases.status3* (0x0001, DBT3_USER_PROXYDB).

After the database has been created, it will contain a proxy table for each table or view found in the **default_location**. Users can create additional objects, such as procedure, views, rules, defaults, etc., and both DDL and DML statements that operate on proxy tables will behave as documented in the *Sybase Component Integration Services User Guide*.

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

Meta Table Synchronization

At times, it may be necessary for a DBA to force re-synchronization of the proxy tables contained within the proxy database. Do this with 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 will not be extended; instead, the proxy tables, if any, will be dropped within the proxy database and re-created from the metadata obtained from the pathname specified during **create database ... with default_location = 'pathname'**

If this command 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 re-synchronization is supported for all external data sources, and not just the primary server in an 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 with **sp_defaultloc**, the metadata contained 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 primary location specified with the **create database** command will not be allowed using **alter database**.
- local tables cannot be created in the proxy database. **create table** commands will result in the creation of proxy tables, and the actual table will be created at the remote location.
- The primary location of the table may be specified in the **create table** command, using the **at "pathname"** syntax. If the pathname differs from the primary location, then the **alter database** command will not synchronize the metadata for this table.
- To change the primary location, first drop the database then re-create it with a new pathname specified in the **with default_location = "pathname"** clause. If the location is changed using **sp_defaultloc**, then the new location will be used to provide metadata synchronization, and proxy tables that were created with the prior location will not be synchronized, and in fact may be dropped and replaced if the name conflicts with that of tables at the new location.

Topics and Issues

The following topics, commands, and processes are key features of Component Integration Services and are described in the following sections:

- Using the create existing table Command
- auto identity Option
- Passthrough Mode
- Transaction Management
- Remote Procedures as Proxy Tables
- text and image Datatypes

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 and reads as follows:

```
create existing table table_name (column_list)  
  [ on segment_name ]  
  [[external {table | procedure}] 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. In this case, 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 identically (although case is ignored), column types and lengths must match, or at least be convertible, and the NULL attributes of the columns must match. (See the data type conversion tables in Chapter 4, “Server Classes.”)

- 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.
- Referential constraints are passed to the remote location when appropriate. See Chapter 4, “Server Classes.”

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.

Chapter 4, “Server Classes,” contains a section for each supported server class that describes all possible datatype conversions that are implicitly performed by the server.

Example of Remote Table Definition

The following example illustrates the steps necessary to define the remote Adaptive Server table, *authors*, starting with the server definition:

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

```
exec sp_addserver SYBASE, sql_server, 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 Add an object definition for the remote *authors* table, to be known locally as *authors*:

```
exec sp_addobjectdef authors,  
"SYBASE.pubs2.dbo.authors", "table"
```

- 4 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,  
  postalcodechar(10)  null  
)  
at "SYBASE.pubs2.dbo.authors", "table"
```

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

```
update statistics authors
```

- 6 Execute a query to test the configuration:

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

***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.

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:

- The **connect to** command
- The **sp_autoconnect** stored procedure
- The **sp_passthru** stored procedure
- The **sp_remotesql** procedure

The *connect to* Command

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

```
connect to server_name
```

where *server_name* is the name of a server added to the *sys.servers* table, with its server class and network name defined. See **sp_addserver** in the *Adaptive Server 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 the **connect to** command must be explicitly granted by the System Administrator. The syntax is:

```
grant connect to user_name
```

To revoke permission to use the **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 the system procedure **sp_autoconnect** using the following syntax:

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

Before using **sp_autoconnect**, add the *server_name* to *sys.servers* by 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**, the **autoconnect** feature is disabled.

Anytime 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 very much like 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

The **sp_passthru** procedure allows the user to pass a SQL command buffer to a remote server. The syntax of the SQL statement(s) 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, refer to the *Adaptive Server 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 fred_s_server, "select @@version"
```

In this example, the server passes the query buffer to *fred_s_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, refer to the *Adaptive Server Reference Manual*.

Transaction Management

Transactions provide a way to group Transact-SQL statements so that they are treated as a unit—either all work performed by the statements is committed to the database, or none of it is.

For the most part, transaction management with Component Integration Services is the same as transaction management in Adaptive Server, but there are some differences. They are discussed in the following section.

Two-phase Commit

Two-phase commit transaction management is now available for remote data. It is transparent to user-written applications.

This service tracks the state of a transaction in the local CIS-enabled server, 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 the Adaptive Server Transaction Coordinator (ASTC).

The management of multi-site transactions is handled by the ASTC; CIS registers new participating servers for each transaction, then turns over control of the transaction coordination to ASTC.

Two-phase commit works only with other Adaptive Server Enterprise 12.x servers. For all other servers, both Sybase and non-Sybase, transaction management will be handled as in the earlier releases: CIS makes a ‘best effort’ at managing the transaction, but cannot guarantee recovery.

Configure *sybssystemdb* for at least 10MB.

DDL is not supported within a distributed transaction. An attempt to do so results in an exception.

Server Classes and ASTC

Internally, ASTC views a server as one of three types: DTM-enabled, Pre-DTM, or No-DTM. These types map to the three sets of callbacks used, and map to server classes as indicated in the following table:

Table 2-1: ASTC and CIS Server Classes

ASTC Server Type	CIS Server Class
DTM-enabled	ASEnterprise (12.x or greater)
Pre-DTM	ASEnterprise(pre-12.x ASAnywhere sql_server(10.x or greater) sds
No-DTM	ASIQ sql_server (pre-system 10) generic db2

Transaction processing through an Adaptive Server Enterprise release 12.x server to pre-DTM and No-DTM remote servers should produce the same output as transaction processing through a pre-12.x server to the same remote servers.

Before starting distributed transactions, the local server must be named.

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 CIS before **xact coordination** is enabled. While **xact coordination** is enabled, CIS cannot be disabled. When **xact coordination** is enabled, **transactional_rpc**'s are implicitly enabled.

Enable CIS

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

CIS Set Commands

The behavior of the **cis rpc handling** configuration property and the **set transactional_rpc** commands has changed with the introduction of ASTC. In earlier releases, enabling **cis rpc handling** caused all RPCs to be routed through CIS 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 Enterprise release 12.x, this behavior has changed. If **cis rpc handling** is enabled and **transactional_rpc** is **off**, RPCs within a transaction are routed through the site handler. RPCs executed outside a transaction are sent via the CIS Client-Library connection. The following table illustrates this change in functionality. As with previous releases, **cis rpc handling** is disabled by default.

Table 2-2: 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

Attach and Detach

ASTC provides the capability to attach and detach from a transaction. This allows a user to detach from a transaction that will later be attached to a TP monitor for completion.

An exception results if you attempt to detach from a transaction that includes pre-DTM and no-DTM servers.

Execute Immediate

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

Pre-12.x Servers

Component Integration Services makes every effort to manage user transactions for pre-12.x servers reliably. However, the 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 *direct_connect (access_server)*, *sql_server* (when the server involved is release 10.0 or later), 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 11.5 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. The purpose of **prepare transaction** is to “ping” the remote server to determine that 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.

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** or **set cis_rpc_handling 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 being released by Sybase. However, some database management systems may not support this capability.

Restrictions on Transaction Management

Restrictions on transaction management are as follows:

- Savepoints are not propagated to remote servers.
- 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.

- The transaction model described in “Pre-12.x Servers” on page 25 is not supported in server class *generic* or server class *db2*. It is also not supported in server class *sql_server* when the remote server is a pre-release 10.0 SQL Server or a Microsoft SQL Server. In these cases, the transactions are committed after each statement is completed.

Remote Procedures as Proxy Tables

An optional clause may be added 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.t1"
```

In the case where the remote object is type procedure, several processing differences will occur:

- No indexes will be created for objects of this type.
- A column list must be provided which matches the description of the remote procedure’s result set. This column list is the responsibility of the user, and no verification of its accuracy is provided.
- Column names beginning with underscore (‘_’) can be used to specify parameters, which 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.sybssystemprocs.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. CIS will pass 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 will be 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 ASE query processor considers to be a searchable argument, or SARG. It is generally a SARG if it is not included in any **or** predicates. For example, the following query would prevent 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 (i.e. they must appear at the end of the column list).

Allowing the definition of remote procedures as local tables gives CIS the ability 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, tables of this type 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 tables of this type.

Component Integration Services users can map remote or external objects of the type *rpc* to local proxy tables. If a table is created in this way, it can be referenced only by the **select** and **drop** commands. The commands **insert**, **delete**, and **update** generate error messages, since the table is assumed to be read-only. Proxy definitions should only be created for procedures which return data.

If an object of the type *rpc* has been defined within the server, a query is not issued to the remote server on which the object resides. Instead, the server 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, the server 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, the following query:

```
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 are rules which define the parameters sent to the RPC. If an RPC will be used as a Component Integration Services object, these rules should be kept in mind during development.

- Component Integration Services sends = operators in the **where** clause as parameters. For example, the query:

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

results in Component Integration Services sending two parameters. 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 will 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 rpcl where a = 3 or b = 2
```

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

Another example:

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

Component Integration Services will not send parameters because there is nothing in the **where** clause representing an exact search condition. Component Integration Services will perform the result set calculation locally.

***text* and *image* Datatypes**

The text datatype is used to store printable character data which can be more than 255 bytes. The image datatype is used to store more than 255 bytes of hexadecimal-encoded binary data. The maximum length for text and image data is defined by the server class of the remote server to which the column is mapped:

- For servers of class *sql_server*, the maximum is 2147MB.
- For Open Server applications of class *direct_connect* (*access_server*) the maximum byte count is defined by the functionality of the DirectConnect server.

Restrictions on *text* and *image* Columns

text and *image* columns cannot be used:

- As parameters to stored procedures. text or image values cannot be passed to stored procedures.
- 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, 255 bytes. 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. The following 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 length of the data is limited to 450 bytes.

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, then 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. Users must be aware of this for the following reasons:

- The overhead of allocating and releasing data pages impacts performance.
- The 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 `sql_server`

- 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. See **writetext** for more information.

- 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 450 bytes.
- **insert select** cannot be used to insert text or image values.
- **readtext** is the most efficient way to access text and image data.

text and image Data with Server Class `direct_connect (access_server)`

- Specific DirectConnect servers support text and image data to varying degrees. Refer to 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 the **set textsize** command.
- 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 the @@*textsize* global variable. 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.
- **writetext**, **select into**, or **insert...select** must be used to enter text or image values that exceed 450 bytes.
- **select into** and **insert...select** can be used 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.

Internal Operations

This section describes the underlying operations on remote servers performed by Component Integration Services on behalf of client applications.

Connection Management

When connecting to a remote server on behalf of a client, the server 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 *direct_connect* (*access_server*) and *sql_server* (release 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 within this single connection context.

However, for pre-release 10.0 SQL Server, and servers of class *db2* and *generic*, it may be necessary to establish more than one connection to that server in order 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.

Query Processing

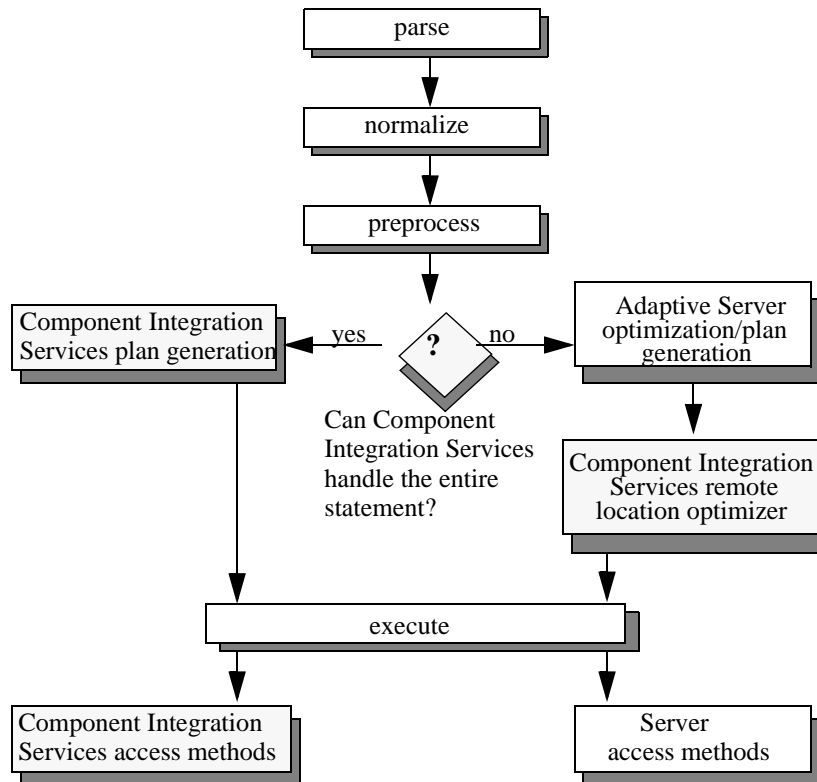
The query processing steps taken when Component Integration Services is enabled are similar to the steps taken by Adaptive Server, except for the following:

- 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 2-1.

An overview of these steps follows.

Figure 2-1: 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 will be 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 will handle optimization.

Component Integration Services will handle 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*, *db2*, or *generic* 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 will forward 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 the **update statistics** command has not been run, the Adaptive Server defaults apply.

For more information on Adaptive Server optimization, refer to 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 for the following conditions:

- Join processing
- Aggregate processing

In order to make intelligent evaluations of a query to improve performance in the above areas, statistics are required. These are obtained by executing the command **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 an **update statistics** command 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 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 gives the server the ability 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

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. The following 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 *AI*:

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

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

Query Execution

The query execution stage receives a query plan, generated either as a result of an adhoc 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.

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 11.5, 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 could 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* Command**

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, and then creates the table. 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.

The **create table** command is reconstructed in a syntax that is appropriate for the server class. For example, if the server class is *db2*, then 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.

Datatype conversion charts for each server class are provided in Chapter 4. Some server classes have restrictions on what datatypes can and cannot be supported. These are also described in Chapter 4, “Server Classes.”

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

***create existing table* Command**

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 the **create existing table** command.

- **sp_statistics** – obtains index information in order to update the local system table, *sysindexes*.

alter table Command

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.

The **alter table** command 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.

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

drop table Command

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.

The **drop table** command 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* Command**

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.

The **drop index** command 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* Command**

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
```

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

Triggers

Component Integration Services allows triggers on proxy tables; however their usefulness is limited. It is possible to **create** a trigger on a proxy table and the trigger will be 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.

Referential Integrity

You can use **Component Integration Services** to maintain referential integrity between remote tables. See the section on constraints in the *Transact-SQL User's Guide*. During **update**, **insert**, and **delete** operations, **Component Integration Services** checks the referenced table. If the check fails, the transaction is rolled back.

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.

Trusted Mode

Trusted mode can be used only between two servers with site handlers. When **Component Integration Services** establishes a **Client-Library** connection to a remote server, trusted mode cannot be used. If a trusted mode connection is needed, use **set cis_rpc_handling off**.

For more information about trusted mode, see Chapter 8, "Managing Remote Servers," in the *Security Administration Guide*.

Adaptive Server Features and Remote Data Access

The following sections present compatibility considerations when using Adaptive Server for remote data access with Component Integration Services.

ANSI Joins

ANSI joins are fully supported for remote data access. If a query cannot be translated, CIS will compensate functionally. The following rules apply to ANSI joins with remote data:

When the remote server supports only ANSI joins

All queries containing outer joins are converted to ANSI joins.

When the remote server supports both ANSI joins and T-SQL joins

Queries containing ANSI join syntax are sent using ANSI join syntax. Queries containing T-SQL outer join syntax are sent with T-SQL syntax.

When the remote server supports only DB2-ANSI

The query to the remote server will be sent in ANSI if all tables in the from clause participate in an ANSI join.

- Not sent:

```
select * from T1 left join T2 on  
T1.a = T2.a, T3
```
- Sent

```
select * from T1 Left Join T2 on  
T1.a = T2.a Left Join T3 on  
T3.a = T2.a
```

When an ANSI query is received for a server that does not support ANSI syntax

The query will be converted to T-SQL if possible.

50-Table Join Limit

The 50-table join limit is fully functional with remote servers that support it.

CIS first checks the capabilities of remote servers. If the 50-table join is supported by the remote server, the requested query is passed onto the remote server.

If the remote server does not support a 50-table join, a query is sent that references 16 tables or less at a time.

Remote Server Logins

To fully support remote logins, Client-Library provides new connection properties which enable CIS 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.

This feature is not enabled automatically. Instead, the SSO or DBA will have to request it through the use of the stored procedure *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 currently NULL.

If the *server login* option is enabled (set to TRUE), then CIS 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 login*' is set to TRUE
- The remote server is configured with server class *ASEnterprise*
- There is a local server name defined for the CIS-enabled server (i.e. the query **select @@servername** returns something other than NULL)

Adaptive Server Features Not Supported by CIS

Parallel Processing

Parallel processing is disabled while Component Integration Services accesses remote tables.

External Security

Clients connecting to Adaptive Server can use external security features when Component Integration Services is enabled. However, Component Integration Services does not incorporate security features when communicating with remote servers. Clients using external security features should use **sp_addexternlogin** to access remote servers.

Adaptive Server Features Not Supported by CIS

Using Component Integration Services

This chapter provides information on defining objects, configuring, tuning, and using Component Integration Services.

- Getting Started with Component Integration Services
- Configuration and Tuning

Name	Page
Getting Started with Component Integration Services	52
Configuration and Tuning	60
RPC Handling and Component Integration Services	67
dbcc Commands	74
Shared Memory Requirements	78
Backing Up Your System	80

Getting Started with Component Integration Services

This section is intended to help first-time users get Component Integration Services running quickly. It 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.

Follow these steps to configure the server to access remote data:

Overview of the Procedure

- 1 Add the remote server to the interfaces file, using the **dsedit** or **dscp** utility.
- 2 Add the name, server class, and network name of the remote server to system tables, using the system procedure **sp_addserver**.
- 3 Assign an alternate login name and password, using the system procedure **sp_addexternlogin**. This step is optional.

Step 1: Add the Remote Server to the Interfaces File

Use the **dsedit** or **dscp** utility to edit the interfaces file located in the *\$SYBASE* directory on the UNIX platform:

- 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.

Step 2: Create Server Entries in System Tables

Use the system procedure **sp_addserver** to add entries to the *sys.servers* 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. Server classes are defined in Chapter 4, “Server Classes.” The default value is **sql_server**. 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 *sql_server*.

```
sp_addserver DOCS, local
sp_addserver CTOSDEMO, sql_server, CTOSDEMO
```

Step 3: Add an Alternate Login and Password

Use the system procedure **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.sys.servers* 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 into 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.

Mapping Remote Objects to Local Proxy Tables

Location transparency of remote data is enabled through remote object mapping.

Once a remote server has been properly configured, users can reference the remote objects that have been defined. Users can create new tables on remote servers and can define the schema for an existing table on a remote server.

Overview of the Procedure

- 1 Use the stored procedure **sp_addobjectdef** to define the storage location of a remote object.
- 2 Use the **create table** or the **create existing table** command to map the remote table schema to the server.

Step 1: Define the Storage Location of a Remote Object

The **at pathname** syntax used with **create existing table** command is the preferred method for defining the storage location of remote objects. The following method using **sp_addobjectdef** is also supported.

The stored procedure **sp_addobjectdef** defines the storage location of a remote object. This procedure allows the user to associate a remote object name with a local table name. The remote object may or may not exist before the storage location is defined. The syntax for **sp_addobjectdef** is:

```
sp_addobjectdef object_name, "object_loc" [,"object_type"]
```

where:

- *object_name* is the local proxy table name to be used by subsequent statements. *object_name* takes the form:

```
dbname.owner.object
```

where *dbname* and *owner* are optional and represent the local database and owner name. If not present, the object is defined in the current database owned by the current owner. If either *dbname* or *owner* is specified, the entire *object_name* must be enclosed in quotes. If only *dbname* is present, a placeholder is required for *owner*.

- *object_loc* is the storage location of the remote object. It takes the form:

```
server_name.dbname.owner.object;aux1.aux2
```

where:

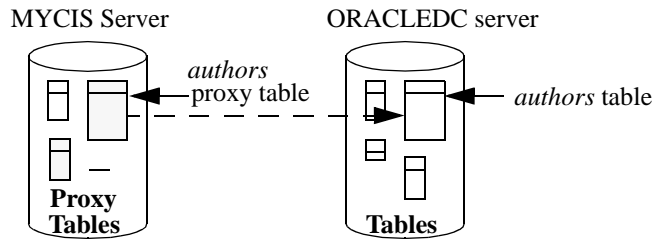
- *server_name* is the name of the server that contains this remote object (required.)
- *dbname* is the name of the database managed by the remote server that contains this object (optional). If the server is class *db2*, this is the *location_name* portion of a DB2 table name.
- *owner* is the name of the remote server user that owns the remote object (optional). If the server is class *db2*, this is the DB2 authorization ID.
- *object* is the name of the remote table, view, or rpc.
- *aux1.aux2* is a string of characters that is passed to the remote server during a **create table** or **create index** command as the segment name; the meaning of this string is dependent upon the class of the server that receives it. If the server is class *db2*, *aux1* is the DB2 database in which to place the table, and *aux2* is the DB2 tablespace in which to place the table. *aux1.aux2* is optional.
- *object_type* is the type of remote object. It can be a table, view, or rpc. This parameter is optional; the default is **table**.

When present, the *object_type* option must be enclosed in quotes.

Example To map the proxy table *authors* to the remote *authors* table, use the following syntax for the database shown in Figure 3-1:

```
sp_addobjectdef authors, "ORACLEDC...authors", "table"
```

Figure 3-1: Using *sp_addobjectdef* to map a remote table to a proxy table



Step 2: Map Remote Table Schema to Adaptive Server

Once you have defined the storage location, you can create the table as a new object or as an existing object. If the table does not exist at the remote storage location, use the **create table** syntax. If it already exists, use the **create existing table** syntax. If the object type is **rpc**, only the **create existing table** syntax is allowed.

When a **create existing table** statement is received and the object type is either **table** or **view**, the existence of the remote object is checked using the catalog stored procedure **sp_tables**.

If the object exists, column and index attributes are obtained and compared with those defined for the object in the **create existing table** command. The server checks the column name, type, length and null property and adds index attributes to the *sysindexes* system table.

Once the object has been created, either as a new or existing object, users can query the remote object by using the local proxy name.

See **create table** and **create index in** the *Adaptive Server Reference Manual*.

Join Between Two Remote Tables

With Component Integration Services, you can perform joins across remote tables. The following steps show how to join two Adaptive Server tables:

Overview of the Procedure

- 1 Add the remote servers to the interfaces file.
- 2 Define each remote server using **sp_addserver**.
- 3 Map the remote tables to the server using **create existing table**.
- 4 Perform the join using **select**.

Step 1: Add the Remote Servers to the Interfaces File

Edit the interfaces file using the **dsedit** utility.

Step 2: Define the Remote Servers

Use the system procedure **sp_addserver** to add entries to the *sys.servers* 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, defined in Chapter 4, “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 *sql_server*.

```
sp_addserver DOCS, local
sp_addserver CTOSDEMO, sql_server, SYBASE
```

Step 4: Map the Remote Tables to Adaptive Server

The **create existing table** command enables the definition of existing (proxy) tables. The syntax for this option is similar to the **create table** command and reads as follows:

```
create existing table table_name (column_list)
[ on segment_name ]
at "pathname"
```

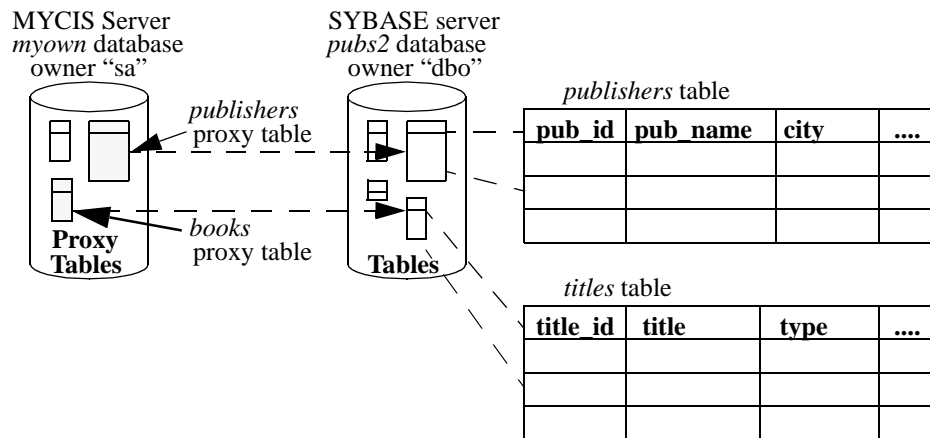
When the server processes this command, it does not create a new table. Instead, it checks the table mapping and verifies the existence of the underlying object. If the object does not exist (either host data file or remote server object), the server rejects the command and returns an error message to the client.

After you define an existing table, it is good practice to issue an **update statistics** command for that table. This helps the query optimizer make intelligent choices regarding index selection and join order.

Example

Figure 3-2 illustrates the remote Adaptive Server tables *publishers* and *titles* in the sample *pubs2* database mapped to a local server.

Figure 3-2: 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 *sql_server*, and its name in the interfaces file is SYBASE:

```
exec sp_addserver SYBASE, sql_server, 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 Add an object definition for the remote *publishers* table:

```
exec sp_addobjectdef publishers,
"SYBASE.pubs2.dbo.publishers", "table"
```

- 4 Define the remote *publishers* table:

```
create existing table publishers
```

```
(
  pub_idchar(4) not null,
  pub_namevarchar(40)null,
  city varchar(20)null,
  state char(2) null
)
at "SYBASE.pubs2.dbo.titles"
```

- 5 Define the remote *titles* table:

```
create existing table books
(
  title_idtid not null,
  title varchar(80)not null,
  type char(12) not null,
  pub_idchar(4) null,
  price money null,
  advancemoney null,
  total_salesint null,
  notes varchar(200)null,
  pubdatedatetime not null,
  contractbit not null
)
```

- 6 Update statistics in both tables to ensure reasonable choices by the query optimizer:

```
update statistics publishers
update statistics books
```

Step 5: Perform the Join

Use the **select** statement to perform the join.

```
select Publisher = p.pubname, Title = b.title
from publishers p, books b
where p.pub_id = b.pub_id
order by p.pubname
```

Configuration and Tuning

This section is intended for System Administrators. It 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 in such a way as 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 the **sp_configure** system procedure control resource allocation and performance. The System Administrator can reset these configuration parameters in order 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 the system procedure **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 connect timeout

cis bulk insert batch 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 the system procedure **sp_configure** without giving it any parameters.

Changing the Configuration Parameters

The stored procedure **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**
- **max cis remote connections**

	<ul style="list-style-type: none">• max cis remote servers• cis bulk insert batch size• cis connect timeout• cis cursor rows• cis packet size• cis rpc handling
enable cis	<p>Use this parameter with sp_configure to enable Component Integration Services as follows:</p> <ol style="list-style-type: none">1 Log into Adaptive Server as the System Administrator and issue the following command: <pre>sp_configure "enable cis", 1</pre>2 Restart Adaptive Server. <p>Issuing the command sp_configure "enable cis", 0 disables Component Integration Services after restarting the server.</p>
max cis remote connections	<p>The server establishes Client-Library connections to remote servers on behalf of clients. More than one connection per client may be required if multiple servers are being accessed by that client. By default, Component Integration Services allows up to 4 connections per user to be made simultaneously to remote servers. For example, if you set the maximum number of users to 25, up to 100 simultaneous Client-Library connections are allowed by Component Integration Services.</p> <p>If this number does not meet the needs of your installation, you can override the setting by specifying how many outgoing Client-Library connections you want the server to be able to make at one time.</p>
max cis remote servers	<p>This configuration parameter allows you to specify how many concurrent servers can be accessed from within the server using Client-Library connections. The default is 25.</p>
cis bulk insert batch size	<p>This configuration parameter determines how many rows from the source table(s) 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.</p> <p>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, the server issues a bulk commit to the target server, causing the batch to be committed.</p>

	<p>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 the server ignores the value of this configuration parameter.</p>
cis connect timeout	<p>This configuration parameter determines the wait time in seconds for a successful Client-Library connection. By default, no timeout is provided.</p>
cis cursor rows	<p>This configuration parameter allows users to specify the cursor row count for cursor open and cursor fetch operations. Increasing this value means more rows will be fetched in one operation. This increases speed but requires more memory. The default is 50.</p>
cis packet size	<p>This configuration parameter allows you to specify the size of Tabular Data Stream™ (TDS) packets that are exchanged between the server and a remote server when connection is initiated.</p> <p>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.</p> <p>If a packet size larger than the default is specified, and the requested server is release 10.0 or later, then the target server must be configured to allow variable-length packet sizes. Adaptive Server configuration parameters of interest in this case are:</p> <ul style="list-style-type: none"> • additional netmem • maximum network packet size <p>Refer to the <i>System Administration Guide</i> for a complete explanation of these configuration parameters.</p>
cis rpc handling	<p>This global configuration parameter determines whether Component Integration Services will handle 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.</p> <p>For more information on using the Adaptive Server site handler vs. using Component Integration Services to handle outbound RPCs, see “RPC Handling and Component Integration Services” on page 67.</p>

Global Variables for Status

The following global variables have been added for CIS users:

- **@@cis_rpc_handling**
- **@@transactional_rpc**
- **@@textptr_parameters**

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).

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 Enterprise 12.x servers 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 the section elsewhere in this book entitled “*Java Classes in SQL*.” Then, after installing your Java class files on the local server, install the required Java class files on the remote server.

Quickpass

Inserts, updates and deletes must employ the Component Integration Services **quickpass mode**. The **quickpass mode** is defined as follows:

- 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.

@ @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 will be truncated, and the deserialization will fail.

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 will fail. Therefore, you must enable trace flag 11220 when you **insert**, **update** or **delete** data for which constraint checking will be done on Java data types. See “Trace Flags” on page 75.

Error Messages

There are two new 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.

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 Handling 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 than that of normal client connections. First, the remote server must determine if the server from which the connect request originated is configured in its *syservers* 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 2-32.)

- 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 Handling 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 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 will not affect the behavior of other threads.

When **cis rpc handling** is enabled, outbound RPC requests are not routed through the Adaptive Server's 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. This transactional RPC capability is supported only when release 10.0 or later Servers or DirectConnect servers are involved.

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. 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). For more information about trusted mode, see "Trusted Mode" on page 2-32.

Sending Text as RPC Parameters

You can add an optional clause to the create existing table statement to indicate that a remote object is 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.t1"
```

In the case where the remote object is type procedure, processing differences will occur:

No indexes will be created for objects of this type.

Provide a column list that matches the description of the remote procedure's result set. This column list is the responsibility of the user, and no verification of its accuracy is provided. You can use column names beginning with underscore ('_') to specify parameters, which 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.sybssystemprocs.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.

Sending Large Data Chunks

You can send large chunks of data in a single remote procedure call. This is done by treating certain parameters as `textptrs`, then de-referencing these text pointers to obtain the text values associated with them. The text data is then packaged into 32K chunks and handed to Client Library as parameters to the RPC.

A text pointer is identified as a parameter of type either `binary(16)` or `varbinary(16)`. The text value referenced by each text pointer parameter is obtained when an RPC is executed, and expanded into 32K chunks, each of which is passed to Client Library as a parameter of type `CS_LONGCHAR_TYPE`.

To send text as RPC parameters, take the following steps:

- 1 Use the following set commands:

```
set cis_rpc_handling ON
set textptr_parameters ON
```

The defaults are OFF
- 2 Declare a pathname as `varchar`.
- 3 Declare `textptrs` as `binary(16)` or `varbinary(16)`.
- 4 Assign actual values to the `textptrs` from actual text pointers.
- 5 Send the RPC using the *pathname* argument.

When an RPC is requested (assume **omni_rpc_handling ON**), `textptrs` are de-referenced in the CIS layer, and the text value is used to construct the parameter for Client Library.

For this to work, `textptrs` must be preceded by a path name argument, which is used to identify the source table from which the `textptrs` were 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 ...

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 `textptrs` `@textptr1` and `@textptr2`.

The varchar parameter @pathname must immediately precede the binary(16) parameter, otherwise it will be considered an ordinary parameter and will be transmitted to the server NETGW as such.

If the text values of a textptr exceed 32K in size, the text will be broken into 32K chunks, each of which will be a separate parameter of type CS_LONGCHAR_TYPE.

The current value of @@textsize will be ignored.

This scheme also works with proxy tables mapped to remote procedures. For example:

```
select id,
       crdate,
       name
from myrpctable
where _pathname = "mydatabase.dbo.t1" and
      _textptr1 = @textptr1 and
      _textptr2 = @textptr2

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 will send 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_parameters** option has been set ON, the textptrs will be expanded to CS_LONGCHAR_TYPE, just as in the case of the RPC example shown previously.

This feature only works with servers that support CS_LONGCHAR_TYPE.

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 will result in a run-time 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.

rusage

rusage returns a report describing the total memory used by each Component Integration Services resource utilizing shared memory. The report describes total configured items, number of items used, number of items available, and total memory used for each resource.

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 **sys.servers**. The command takes an optional argument as follows:

```
srvdes, [ srvid ]
```

showcaps

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

```
showcaps, servername
```

Example:

dbcc cis("showcaps", "servername")

setcaps

setcaps sets a specific capability (identified by *cap_id*) to a new value. The command takes an optional argument as follows:

```
setcaps, servername, [cap_id, new_value]
```

In the following format, **setcaps** resets all capabilities for the server to original values:

```
dbcc cis("setcaps", "servername", -1)
```

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
3703	Disables proxy table index creation during create existing table or create proxy_table command execution. If this flag is set on, then no index metadata will be imported from the remote site referenced by the proxy table, and no indexes for the proxy table will be created. This trace flag is global and should be used with care and turned off when no longer necessary. (global)
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)
11206	Not used.
11207	Logs text and image processing. (global)
11208	Prevents the create index and drop table statements from being transmitted to a remote server. <i>sysindexes</i> is updated anyway. (<i>spid</i>)

Trace Flag	Description
11209	Instructs update statistics to obtain just row counts rather than complete distribution statistics, from a remote table. (<i>spid</i>)
11210	Disables Component Integration Services enhanced remote query optimization. (<i>spid</i>)
11211	Not used.
11212	Prevents escape on underscores (“_”) in table names. (<i>spid</i>)
11213	Prevents generation of column and table constraints. (<i>spid</i>)
11214	Disables Component Integration Services recovery at start-up. (global)
11215	Sets enhanced remote optimization for servers of class <i>db2</i> . (global)
11216	Disables enhanced remote optimization. (<i>spid</i>)
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 a query won’t be rejected by the quickpass mode because of constraints. (<i>spid</i>)
11221	Disables alter table commands to the remote server when ON. This allows users to modify <i>type</i> , <i>length</i> and <i>nullability</i> of columns in a local table without changing columns in the remote table. Use trace flag 11221 with caution. It may lead to tables which are “out of sync.” (<i>spid</i>)

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** is *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 the optimization of your queries is the accuracy of the distribution steps. Therefore, if there are significant changes in the key values in your index, rerun **update statistics** on that index.

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

Only the table owner or System Administrator can issue the **update statistics** command. In addition, your **curwrite** level must dominate the hurdle of the table.

The syntax is:

```
update statistics table_name [index_name]
```

If you do not specify an index name, the command updates the distribution statistics for all the indexes in the specified table. Giving an index name updates statistics for that index only.

Try to run **update statistics** at a time when the tables you need to specify are not heavily used. **update statistics** acquires locks on the remote tables and indexes as it reads the data. If trace flag 11209 is used, tables will not be 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 that triggers and procedures that use the indexes will use the new distribution:

```
sp_recompile authors
```

Finding Index Names

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

To list the indexes for the *authors* table, type:

```
sp_helpindex authors
```

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

```
update statistics authors
```

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

```
update statistics authors auidind
```

Shared Memory Requirements

When configured, the Component Integration Services shared library draws memory from the shared memory pool initialized by the server during start-up. The amount of memory required by the shared library varies, depending on configuration values. Five resources are managed by the shared library. To view the resources, execute the following **dbcc** command:

```
dbcc cis("rusage")
```

With 25 users, the output might look like this:

```
ResourceConfiguredAvailableMemory(Bytes)
-----CIS
SRVDES 25          24          4900
CIS DES 500        498          134000
CIS PSS 49         48           2940
CIS RDES 100       100          20800
CIS CURSOR 400    400          49600
```

With 50 users, the output might look like this:

```
ResourceConfiguredAvailableMemory(Bytes)
----- CIS
SRVDES 25          25          4900
CIS DES 500        500          134000
CIS PSS 74         73           4440
CIS RDES 100       100          20800
CIS CURSOR 800    800          99200
```

The resources are configured as follows:

- CIS SRVDES – configured via the **max cis remote servers** configuration parameter. The default is 25. Each additional SRVDES requires approximately 196 bytes of memory.
- CIS DES – configured indirectly via the **open objects** configuration parameter. For each open object, a CIS DES is allocated. Each CIS DES requires 268 bytes of memory.
- CIS PSS – configured indirectly via the **user connections** configuration parameter. For each user connection, a CIS PSS is allocated. Each CIS PSS requires 60 bytes of memory.
- CIS RDES – configured indirectly via the **max cis remote connections** configuration parameter. There is one CIS RDES for each remote connection.

- CIS CURSOR – configured indirectly via the **user connections** configuration parameter. The number of CIS CURSOR resources is calculated as:

$$4 * \text{user connections}$$

Each CIS CURSOR requires 124 bytes of memory.

Additional Component Integration Services Memory Requirements

In addition to the shared memory used by **Component Integration Services**, dynamic memory which is not accounted for by any configuration value is also used. Dynamic memory is used for:

- The shared library – When **Component Integration Services** is configured and loaded, the shared library adds approximately 550K to the size of the server executable. This additional memory usage does not appear anywhere, except in operating system commands (for example, the UNIX **ps** command).
- Dynamic Client-Library memory – When connections to remote servers are necessary, **Component Integration Services** uses Client-Library to establish them. During query and results processing, Client-Library dynamically allocates additional memory and then frees it when the connection ends or statement completes.

Backing Up Your System

Use the **backup** utility. See the *System Administration Guide* for details.

Server Classes

This chapter provides reference material on the server classes supported by Component Integration Services. The topics include:

Name	Page
Defining Remote Servers	82
Datatype Conversions	86
Remote Server Capabilities	87
Transact-SQL Commands	88

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

Defining Remote Servers

Use the system procedure **sp_addserver** to add entries to the *sys.servers* 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 the name used to identify the server. It must be unique.
- *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*.

Server Class ASEnterprise

A server with server class *ASEnterprise* is Adaptive Server Enterprise version 11.5 or later. When CIS first establishes a connection to a server in this class, CIS will determine the version (i.e. 11.5, 11.9.2, 12.0, etc.) and establish server capabilities based on the version found. For example, version 12.0 supports ANSI syntax for outer joins, while prior versions 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
- Adaptive Server IQ 12.0 or later

Server Class ASIQ

A server with server class *ASIQ* is any version of Adaptive Server IQ prior to version 12.

Server Class *sql_server*

A server with server class *sql_server* is:

- SQL Server release 4.9 or later
- SQL Anywhere release 5.5.01 or later
- Microsoft SQL Server version 4.2 or later
- OmniConnect release 10.5 or later
- OmniSQL Server 10.1.2

Server Class *db2*

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

- Net-Gateway™ release 3.0 or later (Net-Gateway release 3.01 can also be configured as server class *direct_connect*)
- DirectConnect for MVS / TRS (can also be configured as server class *direct_connect*)
- InfoHub

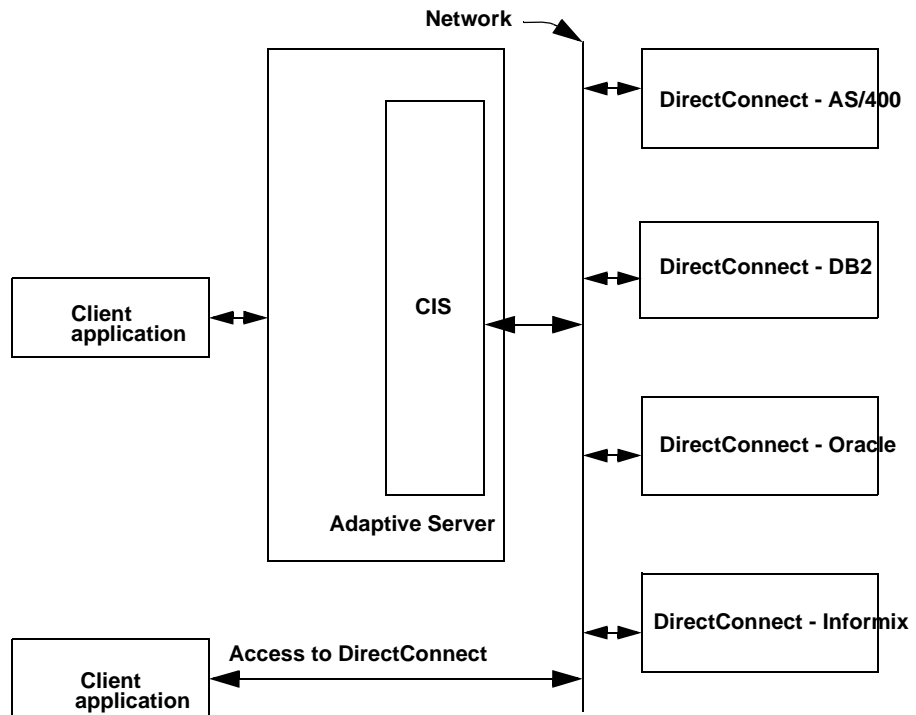
Server Class *direct_connect*

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 with previous releases.

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

Figure 4-1 illustrates the manner in which 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 4-1: Adaptive Server with CIS interacts 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.

Server Class *generic*

The server class *generic* allows customers to build their own Open Server applications to communicate with the server. The customer-built application must conform to the interface described in the *OmniSQL Server Generic Access Module Reference Manual*.

The server class *generic* is supported for compatibility with existing Open Server applications. Server class *sds* replaces server class *generic*. New Open Server applications that are compatible with Component Integration Services should follow the interface specification in the *Adaptive Server Specialty Data Store Developer's Kit* manual.

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 server's server class. See the **create table**, **alter table** and **create existing table** commands in the following reference pages for tables that illustrate the datatype conversions that take place for each server class when the commands are processed.

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 set of results 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, CIS will set remote server capabilities for the remote server based on a set of assumptions. For example, server class *db2* will inherit 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 ASE represented by the remote server.

Transact-SQL Commands

The following pages are reference pages, presented in alphabetical order, which discuss Transact-SQL commands that either 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 described. For a complete description of each command, see the *Adaptive Server 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:

Function - contains a brief description of the command.

Syntax - contains a description of the full Transact-SQL syntax of the command.

Comments - 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 sql_server - contains a description of handling specific to server class *sql_server*. This includes syntax that is forwarded to a remote server of class *sql_server*.

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*.

Server Class generic - contains a description of handling specific to server class *generic*. This includes syntax that is forwarded to a remote server of class *generic*.

alter database

Description	Increases the amount of space allocated to a database. Synchronizes proxy table metadata with tables at remote location.
Syntax	<pre>alter database <i>database_name</i> [on {default <i>database_device</i> } [= size] [, <i>database_device</i> [= size]]...] [log on { default <i>database_device</i> } [= size] [, <i>database_device</i> [= size]]...] [with override] [for load] [for proxy_update]]</pre>
Usage	<p>Usage</p> <ul style="list-style-type: none">• If a database has been created with the optional clause with default_location = pathname, then the alter database command, with the for proxy_update clause, will re-synchronize the proxy tables in the named database with tables and views found in the pathname to the remote location.• The default location may also have been specified with the system stored procedure 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 (<i>master.dbo.sysdatabases.default_loc</i>) 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).

See also

See Also

create database in the *Adaptive Server Reference Manual* and later in this chapter.

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	<pre> alter table [<i>database</i>.<i>owner</i>].<i>table_name</i> {add <i>column_name datatype</i> [default {<i>constant_expression</i> user null}] {identity null not null} [[constraint <i>constraint_name</i>] {{unique primary key} [clustered nonclustered] [asc desc] [with { { fillfactor = pct max_rows_per_page = num_rows } , reservepagegap = num_pages }}] [on <i>segment_name</i>] references [[<i>database</i>.<i>owner</i>].<i>ref_table</i>] [(<i>ref_column</i>)] check (<i>search_condition</i>)] ... } [, <i>next_column</i>]... add { [constraint <i>constraint_name</i>] { {unique primary key} [clustered nonclustered] (<i>column_name</i> [asc desc] [, <i>column_name</i> [asc desc]...]) [with { { fillfactor = pct max_rows_per_page = num_rows } , reservepagegap = num_pages}] [on <i>segment_name</i>] foreign key (<i>column_name</i> [{, <i>column_name</i>}...]) references [[<i>database</i>.<i>owner</i>].<i>ref_table</i>] [(<i>ref_column</i> [{, <i>ref_column</i>}...])] check (<i>search_condition</i>)} drop {[<i>column_name</i> [, <i>column_name</i>]} [<i>constraint constraint_name</i>]} modify <i>column_name</i> {[<i>data_type</i>] [null] [not null]} [, <i>column_name</i>] replace <i>column_name</i> default {<i>constant_expression</i> user null} partition <i>number_of_partitions</i> unpartition </pre>

Usage

| lock {allpages | datarows | datapages } }

| with exp_row_size = num_bytes

Usage

- Component Integration Services processes the **alter table** command 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 the **alter table** command 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** and **drop** column. 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 `sql_server`:

```
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 ASE is 11.9.2 or later.

Server Class *ASAnywhere* •Handling of the **alter table** command by servers in this class is the same as for *ASEnterprise* servers.

Server Class *ASIQ* Handling of the **alter table** command by servers in this class is the same as for *ASEnterprise* servers.

- text and 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 *sql_server* • Handling of the **alter table** command by servers in this class is the same as for *ASEnterprise* servers.

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 the **alter table** command to the remote server regardless of the capabilities response.
- The behavior of the server with class *direct_connect* is database dependent. *alter table [database .[owner].]table_name {add column_name datatype [{identity | null}] {[, next_column]}...* 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 4-1

Table 4-1: DirectConnect datatype conversions for alter table

Adaptive Server Datatype	DirectConnect Default Datatype	DirectConnect DB2 Syntax Mode Datatype
binary(<i>n</i>)	binary(<i>n</i>)	char(<i>n</i>) for bit data
bit	bit	char(1)
char	char	char
datetime	datetime	timestamp
decimal(<i>p</i> , <i>s</i>)	decimal(<i>p</i> , <i>s</i>)	decimal(<i>p</i> , <i>s</i>)
float	float	float
image	image	varchar(<i>n</i>) for bit data; the value of <i>n</i> is determined by the global variable @@textsize
int	int	int
money	money	float
numeric(<i>p</i> , <i>s</i>)	numeric(<i>p</i> , <i>s</i>)	decimal(<i>p</i> , <i>s</i>)

Adaptive Server Datatype	DirectConnect Default Datatype	DirectConnect DB2 Syntax Mode Datatype
nchar(<i>n</i>)	nchar(<i>n</i>)	graphic(<i>n</i>)
nvarchar(<i>n</i>)	nvarchar(<i>n</i>)	vargraphic(<i>n</i>)
real	real	real
smalldatetime	smalldatetime	timestamp
smallint	smallint	smallint
smallmoney	smallmoney	float
timestamp	timestamp	varbinary(8)
tinyint	tinyint	smallint
text	text	varchar(<i>n</i>); the value of <i>n</i> is determined by the global variable @@textsize
varbinary(<i>n</i>)	varbinary(<i>n</i>)	varchar(<i>n</i>) for bit data
varchar(<i>n</i>)	varchar(<i>n</i>)	varchar(<i>n</i>)

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 4-2.

Table 4-2: DB2 datatype conversions for alter table

Adaptive Server Datatype	DB2 Datatype
binary(<i>n</i>)	char(<i>n</i>) for bit data, where $n \leq 254$
bit	char(1)
char(<i>n</i>)	char(<i>n</i>), where $n \leq 254$
datetime	timestamp
decimal(<i>p</i> , <i>s</i>)	decimal(<i>p</i> , <i>s</i>)
float	float
image	Not supported

Adaptive Server Datatype	DB2 Datatype
int	int
money	float
nchar	char(n)
nvarchar	varchar(n)
numeric(<i>p</i> , <i>s</i>)	decimal(<i>p</i> , <i>s</i>)
real	real
smalldatetime	timestamp
smallint	smallint
smallmoney	float
tinyint	smallint
text	Not supported
varbinary(n)	varchar(n) for bit data, where <i>n</i> <=254
varchar(n)	varchar(n), where <i>n</i> <= 254

Server Class *generic* •Component Integration Services forwards the following syntax to a server with server class *generic*, unless a text, image, decimal, or numeric datatype is specified:

```
alter database [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]
```

If a text, image, decimal, or numeric datatype is used, Component Integration Services raises Error 11205:

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

- When a user defines a column with the **alter table** command, a datatype must be provided. The server passes the datatype name of each column to the Generic Access Module without conversion.

See also

See Also

alter table in the *Adaptive Server Reference Manual*.

begin transaction

Description	Marks the starting point of a user-defined transaction.
Syntax	begin tran[saction] [<i>transaction_name</i>]
Usage	<p>Usage</p> <ul style="list-style-type: none"> • When the Distributed Transaction Manager (DTM) is enabled, DTM handles all transaction processing for servers of server class <i>ASEnterprise</i> with a version of 12.0 or later. • If the Adaptive Server is configured with strict dtm enforcement = 1, any attempt to include a remote server in the transaction that has a server class other than <i>ASEnterprise</i> will cause the transaction to be aborted. • For all server classes, when Adaptive Server receives a begin transaction command, an internal state is set which 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. • <i>transaction_name</i> is not used by Component Integration Services in this release. <p>Server Class ASEnterprise •Transaction process for servers in class <i>ASEnterprise</i> with a version prior to 12.0 is identical to that of server class <i>sql_server</i> (release 10.0 or later).</p> <ul style="list-style-type: none"> • When DTM is not enabled, transaction processing for all servers in class <i>ASEnterprise</i> is identical to that of server class <i>sql_server</i> (release 10.0 or later). • Component Integration Services checks the transaction state of the connection to a server of class <i>ASEnterprise</i>. If the internal transaction state indicates that a transaction is in progress, and the state of the connection to the remote participant indicates that no transaction is in progress, Component Integration Services informs the Distributed Transaction Manager that the server has become a participant in the transaction. The Distributed Transaction Manager then issues a BeginXact RPC to the remote server. <p>Server Class ASAnywhere •Transaction processing for servers in class <i>ASAnywhere</i> is identical to that of server class <i>sql_server</i> (release 10.0 or later).</p> <p>Server Class ASIQ •Transaction processing for servers in class <i>ASIQ</i> is identical to that of server class <i>sql_server</i> (release 10.0 or later).</p>

Server Class *sql_server* •Component Integration Services checks the transaction state of the connection to a server of class *sql_server*. If the internal transaction state indicates that a transaction is in progress, and the state of the connection to the server indicates that no transaction is in progress, Component Integration Services forwards the **begin transaction** command to the server prior to forwarding the first command to that server. In the example below, assume tables *t1* and *t2* are both located on the same remote SQL Server:

```
begin transaction

insert into t1 values (...)
update t2 ...

commit transaction
```

At the time the **begin transaction** command is processed, no interaction with the remote SQL Server occurs.

When the **insert** command is processed, the transaction state of the connection to the server that owns *t1* is checked. Since this is the first command within the transaction, the connection is in a NO TRANSACTION ACTIVE state, and the **begin transaction** command is forwarded to the server. The **insert** command is then forwarded to the remote location, and the transaction state for the connection is marked as TRANSACTION ACTIVE.

When processing the **update** command, the transaction state of the server that owns table *t2* is checked. Since it is the same server that owns table *t1*, it is in the TRANSACTION ACTIVE state, and the **begin transaction** command is not forwarded.

Note These comments apply only to release 10.0 or later, which supports cursors. For pre-release 10.0 SQL Server and Microsoft SQL Server, transaction handling is similar to server class *db2*, described below.

Server Class *direct_connect* •Transaction processing for servers in class *direct_connect* is identical to that of server class *sql_server* (release 10.0 or later).

Server Class *db2* •Transactions are supported only at the statement level for servers in class *db2*. When the internal state of a client connection indicates that there is an active transaction, Component Integration Services precedes each statement forwarded to the server with a **begin transaction** command. Component Integration Services then issues a **commit** or **rollback transaction** (depending on the success or failure of the statement) immediately after the statement is complete.

Server Class *generic* •Transactions are supported only at the statement level for servers in class *generic*. When the internal state of a client connection indicates that there is an active transaction, Component Integration Services precedes each statement forwarded to the server with the RPC **gen_begin_xact**. It then issues a **gen_commit_xact** or **gen_rollback_xact** RPC (depending on the success or failure of the statement) immediately after the statement is complete. Each statement executes completely or not at all.

See also

See Also

begin transaction in the *Adaptive Server Reference Manual*.

case

Description	Supports conditional SQL expressions; can be used anywhere a value expression can be used.
Syntax	<pre> case when search_condition then expression [when search_condition then expression]... [else expression] end </pre>
Usage	<p>Usage</p> <ul style="list-style-type: none"> • 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. <p>Server Class ASEnterprise The capability for handling case expressions is set for ASE 11.5 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.</p> <p>Server Class ASAnywhere The capability for handling 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.</p> <p>Server Class ASIQ •The capability for handling 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 will cause CIS quickpass optimization to reject the statement. When this happens, the case expression must be evaluated by the local ASE after retrieving data from the remote server.</p> <p>Server Class sql_server •The capability for handling 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 will cause CIS quickpass optimization to reject the statement. When this happens, the case expression must be evaluated by the local ASE after retrieving data from the remote server.</p>

Server Class *direct_connect* The capability for handling **case** expressions is determined by the result set from the RPC **sp_capabilities**. If the *direct_connect* indicates that it can handle case expressions, then CIS will forward them to the *direct_connect* when quickpass mode is used to handle the query.

Server Class *generic* •The capability for handling **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 will cause CIS quickpass optimization to reject the statement. When this happens, the case expression must be evaluated by the local ASE after retrieving data from the remote server.

Server Class *db2* The capability for handling 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 will cause CIS quickpass optimization to reject the statement. When this happens, the **case** expression must be evaluated by the local ASE after retrieving data from the remote server.

- If traceflag 11215 is turned **on**, the default capabilities for server class *db2* are modified to enable more capabilities, and case expressions are enabled by default. Note that DB2 does recognize case expression syntax. Also, the traceflag must be turned on before CIS makes its first connection to the remote server, or else the capabilities set by the first connection will remain in effect until the server is rebooted or until modified by the command:

```
dbcc cis("setcaps", "server", id, value)
```

close

Description Deactivates a cursor.

Syntax `close cursor_name`

Usage 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 See Also

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

close in the *Adaptive Server Reference Manual*.

commit transaction

Description	Marks the successful ending point of a user-defined transaction.
Syntax	commit [tran[saction] work] [<i>transaction_name</i>]
Usage	<p>Usage</p> <ul style="list-style-type: none"> • 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. • For all other server classes, 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. • Multiple remote servers can be involved in a single transaction, each with their 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 RPC's must be part of an explicit transaction. • <i>transaction_name</i> is not used by Component Integration Services in this release. <p>Server Class ASEnterprise •Transaction process for servers in class <i>ASEnterprise</i> with a version prior to 12.0 is identical to that of server class <i>sql_server</i> (release 10.0 or later)</p> <ul style="list-style-type: none"> • When DTM is not enabled, transaction processing for all servers in class <i>ASEnterprise</i> is identical to that of server class <i>sql_server</i> (release 10.0 or later). • In all other cases, when the Adaptive Server receives notification to commit a transaction, the Distributed Transaction Manager issues a CommitXact RPC to all remote participants having a server class of <i>ASEnterprise</i>. <p>Server Class ASAnywhere •Transaction processing for servers in class <i>ASAnywhere</i> is identical to that of server class <i>sql_server</i> (release 10.0 or later).</p>

Server Class *ASIQ* •Transaction processing for servers in class *ASIQ* is identical to that of server class *sql_server* (release 10.0 or later).

Server Class *sql_server* •When Component Integration Services receives notification to commit a transaction, it checks the TRANSACTION ACTIVE state of all remote participants associated with the client application. If there is more than one remote server involved in a transaction, Component Integration Services first sends a **prepare transaction** command to each connection with an active transaction. If all remote servers respond with no error, Component Integration Services sends a **commit transaction** command to each server involved in the transaction. If all remote servers again respond with no error, Component Integration Services notifies the Adaptive Server that it can commit local work.

This process applies to release 10.0 or later. Transaction handling is the same as server class *db2*, described below, if the server represented by server class *sql_server* is:

- Pre-release 10.0 SQL Server
- Microsoft SQL Server (any version)
- Sybase IQ
- OmniConnect 10.1.2

Server Class *direct_connect* •Transaction processing for servers in class *direct_connect* is identical to that of server class *sql_server* (release 10.0 or later).

Server Class *db2* •Transactions are supported only at the statement level for servers in class *db2*. When the internal state of a client connection indicates that there is an active transaction, a **begin transaction** command precedes all **insert**, **update** and **delete** commands. Component Integration Services issues a **commit** or **rollback transaction** (depending on the success or failure of the statement) immediately after the statement is complete.

Server Class *generic* •Transactions are supported only at the statement level for servers in class *generic*. When the internal state of a client connection indicates that there is an active transaction, the RPC **gen_begin_xact** precedes all **insert**, **update** and **delete** commands. Component Integration Services issues a **gen_commit_xact** or **gen_rollback_xact** RPC (depending on the success or failure of the statement) immediately after the statement is complete.

See also

See Also

commit in the *Adaptive Server 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	<p>Usage</p> <ul style="list-style-type: none">• connect to specifies the server to which a passthrough connection is required. Passthrough mode enables you to perform native operations on a remote server.• <i>server_name</i> must be the name of a server in the sys.servers table, with its server class and network name defined.• When establishing a connection to <i>server_name</i> on behalf of the user, Component Integration Services uses one of the following identifiers:<ul style="list-style-type: none">- A remote login alias described in <i>sysattributes</i>, if present- The user's name and password <p>In either case, if the connection cannot be made to the specified server, Adaptive Server returns an error message.</p> <ul style="list-style-type: none">• 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 the disconnect command. 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 ASE query processor.• The disconnect command can be abbreviated to disc.• The disconnect command returns an error unless connect to has been previously issued and the server is connected to a remote server.

Server Class ASEnterprise •When the **disconnect** command is issued, CIS will forward the **disconnect** command 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 CIS and not forwarded to the client.

Server Class ASAnywhere •No interaction occurs with *ASAnywhere* when the **connect** or **disconnect** commands are issued.

Server Class ASIQ •No interaction occurs with *ASIQ* when the **connect** or **disconnect** commands are issued.

Server Class sql_server No interaction occurs with *sql_server* when the **connect** or **disconnect** commands are issued.

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

sp_thread_props "passthru mode", 1

- When the **disconnect** command 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 the **connect** or **disconnect** commands are issued.

Server Class generic No interaction occurs with *generic* when the **connect** or **disconnect** commands are issued.

See also

See Also

commit in the *Adaptive Server Reference Manual*.

create database

Description	Creates a new database
Syntax	<pre>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]</pre>
Usage	<p>Usage</p> <ul style="list-style-type: none"> • This command creates a new database within Adapter Server Enterprise. The new syntax with default_location = <i>pathname</i> and for proxy_update have been added to allow automatic creation of proxy tables representing tables and views found in a remote location. • If the clause with default_location = <i>pathname</i> is used, the <i>pathname</i> is stored in <i>master.dbo.sysdatabases.default_loc</i>, and serves the same purpose as the default location added via the system stored procedure sp_defaultloc • If the clause for proxy_update is used, the with default_location = <i>pathname</i> 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 will be calculated based on the number of proxy tables that it will be expected to contain. The formula for calculating the number of 2k pages for the database is as follows: <pre>pages = #rmt_tbls * 32/* 4 extents per table */ overhead = (pages * 1.1) /* add 10% */ if overhead < 500 pages then overhead = 500 pages total_pages = pages + overhead</pre> • 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 will instruct CIS 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 will be populated with proxy tables representing all user tables and views found at the remote location.

See also

See Also

alter database in the *Adaptive Server 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 the **create existing table** command as if the table being created is a new local table.
- After creating the local table, Adaptive Server passes the **create existing table** command 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 the **create existing table** command 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 with those found at the remote location.
- The proxy table need not contain the exact number of columns as found in the remote table. However, all columns referenced in by the proxy table must be found 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, then 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 won't 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 4-3 describes the allowable datatypes that can be used when mapping remote Adaptive Server columns to local proxy table columns:

Table 4-3: Adaptive Server datatype conversions for create existing table

Remote Adaptive Server Datatype	Allowable Adaptive Server Datatypes
binary(<i>n</i>)	image, binary(<i>n</i>), and varbinary(<i>n</i>); if not image, the length must match
bit	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
char(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>), varchar(<i>n</i>); if not text, the length must match
datetime	datetime and smalldatetime
decimal(<i>p</i> , <i>s</i>)	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(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>), varchar(<i>n</i>); if not text, the length must match
numeric(<i>p</i> , <i>s</i>)	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
nvarchar(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>), varchar(<i>n</i>); 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(<i>n</i>)	image, binary(<i>n</i>), and varbinary(<i>n</i>); if not image, the length must match
varchar(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>), varchar(<i>n</i>); if not text, the length must match

Server Class ASAnywhere • Table 4-4 describes the allowable datatypes that can be used when mapping remote Adaptive Server columns to local proxy table columns:

Table 4-4: Adaptive Server Anywhere datatype conversions for create existing table

Remote Adaptive Server Datatype	Allowable Adaptive Server Datatypes
binary(<i>n</i>)	image, binary(<i>n</i>), and varbinary(<i>n</i>); if not image, the length must match
bit	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
char(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>), varchar(<i>n</i>); if not text, the length must match
datetime	datetime and smalldatetime
decimal(<i>p</i> , <i>s</i>)	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(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>), varchar(<i>n</i>); if not text, the length must match
numeric(<i>p</i> , <i>s</i>)	bit, decimal, float, int, money, numeric, real, smallint, smallmoney, and tinyint
nvarchar(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>), varchar(<i>n</i>); 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

Remote Adaptive Server Datatype	Allowable Adaptive Server Datatypes
varbinary(<i>n</i>)	image, binary(<i>n</i>), and varbinary(<i>n</i>); if not image, the length must match
varchar(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>), varchar(<i>n</i>); 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 *ASEnterprise*

Server Class *sql_server* •Allowable datatype conversions are the same as for server class *ASEnterprise*.

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 4-5. If not, a column type error is raised, and the command is aborted.

Table 4-5: DirectConnect datatype conversions for create existing table

DirectConnect Datatype	Allowable Adaptive Server Datatypes
binary(<i>n</i>)	image, binary(<i>n</i>), varbinary(<i>n</i>); 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(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>) and varchar(<i>n</i>); if the length does not match, the command is aborted
datetime	datetime, smalldatetime
decimal(<i>p</i> , <i>s</i>)	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(<i>n</i>)	text, nchar(<i>n</i>), nvarchar(<i>n</i>), char(<i>n</i>) and varchar(<i>n</i>); if the length does not match, the command is aborted

DirectConnect	
Datatype	Allowable Adaptive Server Datatypes
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); if the length does not match, the command is aborted
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)

- 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 *Adaptive Server 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 4-6 describes the allowable Adaptive Server datatypes that can be used for existing DB2 datatypes:

Table 4-6: 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; if not text or image, length must match
char(n) for bit data	binary(n), varbinary(n), and image; if not image, length must match
varchar	char, varchar, bit, binary, varbinary, text and image; if not text or image, length must match
varchar(n) for bit data	binary(n), varbinary(n), and image; if not image, length must match
long varchar (length truncated to 255)	char, varchar, bit, binary, varbinary, text and image; if not text or image, length must match
date	char(10), varchar(10), and datetime (time set to 12:00AM)
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

DB2 Datatype	Allowable Adaptive Server Datatypes
long vargraphic	Not supported

- If the data contained in a long varchar column exceeds 255 bytes, 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. DB2's 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.

Server Class *generic* •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, image, decimal, and numeric datatypes are not supported by the server class *generic*.
- Table 4-7 illustrates datatype compatibility when the **create existing table** command is processed. When the server encounters a datatype shown in the "ODBC Datatype" column, it allows any of the datatypes shown in the "Allowable Adaptive Server Datatypes" column. When a datatype other than an allowable datatype is encountered, Adaptive Server returns an error message and the **create existing table** command is aborted.

Table 4-7: ODBC datatype conversions for create existing table

ODBC Datatype	Allowable Adaptive Server Datatypes
int	int
smallint	smallint
tinyint	tinyint

create existing table

ODBC Datatype	Allowable Adaptive Server Datatypes
float	float, money, and smallmoney
double precision	float, money, and smallmoney
real	real, money, and smallmoney
decimal(p,s) (scale less than 0 or precision greater than or equal to 10)	float, money
decimal(p,s) (scale equal to 0 or precision greater than or equal to 10)	int, float, money
numeric(p,s) (scale less than 0 or precision greater than or equal to 10)	float, money
numeric(p,s) (scale equal to 0 or precision greater than or equal to 10)	int, float, money
char(n)	char(n), varchar(n) (n truncated to 255 bytes)
long varchar(n), varchar(n)	char(n), varchar(n) (n truncated to 255 bytes)
date	datetime (time set to 12:00AM)
time	datetime (date set to 1/1/1900)
timestamp	datetime
bit	bit
binary(n)	binary(n), varbinary(n); length must match
varbinary(n)	binary(n), varbinary(n); length must match
long varbinary(n)	binary(255), varbinary(255)

See also

See Also

create existing table in the *Adaptive Server Reference Manual*.

create index

Description	Creates an index on one or more columns in a table.
Syntax	<pre>create [unique] [clustered nonclustered] index <i>index_name</i> on [[<i>database.</i>]<i>owner.</i>]<i>table_name</i> (<i>column_name</i> [, <i>column_name</i>]...) [with {{fillfactor max_rows_per_page} = x, ignore_dup_key, sorted_data, [ignore_dup_row allow_dup_row]]] [on <i>segment_name</i>]</pre>
Usage	<p>Usage</p> <ul style="list-style-type: none"> • Component Integration Services processes the create index command 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 the create index command. If trace flag 11208 is turned on, Component Integration Services does not send the create index command 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 in order 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 the create index 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. <p>Server Class ASEnterprise •Component Integration Services forwards everything except the on segment_name clause to the remote server.</p> <p>Server Class ASAnywhere •Component Integration Services forwards everything except the on segment_name clause to the remote server.</p> <p>Server Class ASIQ •Component Integration Services forwards everything except the on segment_name clause to the remote server.</p> <p>Server Class sql_server •For pre-release 10.0 SQL Server or Microsoft SQL Server 6.5, neither the max_rows_per_page or on segment_name clause is forwarded to the remote server.</p>

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.

Server Class *generic* •Component Integration Services forwards all syntax except the **max_rows_per_page** and **on segment_name** clauses to the remote server.

- The Generic Access Module must either translate the Sybase extensions to equivalent native syntax or ignore them.

See also

See Also

create index in the *Adaptive Server 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	<code>create proxy_table table_name [external table] at "pathname"</code>
Usage	<p>Usage</p> <ul style="list-style-type: none">• 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. CIS derives the column list from the metadata it obtains from the remote table.• The location information provided by the at keyword specifies the pathname to the remote object (table or view).• 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 <code>table_name</code> used with the <code>create proxy_table</code> statement:<ul style="list-style-type: none">• if <code>table_name</code> is lower case, the generated proxy table name is also lower case, as are all of its columns• if <code>table_name</code> is uppercase, the generated proxy table name is also upper case, as are all of its columns
See also	See Also create table and create existing table in the <i>Adaptive Server Reference Manual</i> .

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	<pre> 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 procedure}] at "pathname"]] </pre>
Usage	<p>Usage</p> <ul style="list-style-type: none"> • 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 <i>sysobjects.sysstat2</i> 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 the **create table** command, 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 the **create table** command. 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 sql_server •Component Integration Services passes the datatype of each column to the remote server without conversion.

Server Class direct_connect •Component Integration Services reconstructs the **create table** command 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 4-8, depending on whether the DirectConnect supports DB2 syntax mode

Table 4-8: DirectConnect datatype conversions for create table

Adaptive Server Datatype	DirectConnect Default Datatype	DirectConnect DB2 Syntax Mode Datatype
binary(<i>n</i>)	binary(<i>n</i>)	char(<i>n</i>) for bit data
bit	bit	char(1)
char	char	char
datetime	datetime	timestamp
decimal(<i>p, s</i>)	decimal(<i>p, s</i>)	decimal(<i>p, s</i>)
float	float	float
image	image	varchar(<i>n</i>) for bit data; the value of <i>n</i> is determined by the global variable @@textsize
int	int	int
money	money	float
numeric(<i>p, s</i>)	numeric(<i>p, s</i>)	decimal(<i>p, s</i>)
nchar(<i>n</i>)	nchar(<i>n</i>)	graphic(<i>n</i>)
nvarchar(<i>n</i>)	nvarchar(<i>n</i>)	vargraphic(<i>n</i>)
real	real	real
smalldatetime	smalldatetime	timestamp
smallint	smallint	smallint
smallmoney	smallmoney	float
timestamp	timestamp	varbinary(8)
tinyint	tinyint	smallint
text	text	varchar(<i>n</i>); the value of <i>n</i> is determined by the global variable @@textsize
varbinary(<i>n</i>)	varbinary(<i>n</i>)	varchar(<i>n</i>) for bit data
varchar(<i>n</i>)	varchar(<i>n</i>)	varchar(<i>n</i>)

Server Class *db2* Table 4-9 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 4-9: DB2 datatype conversions for create table

Adaptive Server Datatype	DB2 Datatype
binary(n)	char(n) for bit data, where $n \leq 254$
bit	char(1)
char(n)	char(n), where $n \leq 254$
datetime	timestamp
decimal(p, s)	decimal(p, s)
float	float
image	Not supported
int	int
money	float
nchar	char(n)
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 $n \leq 254$
varchar(n)	varchar(n), where $n \leq 254$

Server Class *generic* • CIS passes the datatype name of each column to the Generic Access Module without conversion.

- The *generic* server class does not allow text, image, decimal or numeric datatypes. Use of these datatypes results in an error.

See also

See Also

create table in the *Adaptive Server Reference Manual*.

create trigger

Description	Creates a trigger, 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	<pre>create trigger [owner.]trigger_name on [owner.]table_name for {insert, update, delete} as SQL_statements</pre> <p>Or, using the if update clause:</p> <pre>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]...</pre>
Usage	<p>Usage</p> <ul style="list-style-type: none">• 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.• Some direct_connects have the ability to support the special tables inserted and deleted. If this is the case, CIS will forward references to these tables when found within a trigger. The reference will be constructed as in this example:<pre>select ... from dbname.owner.tablename inserted, dbname.owner.tablename deleted where inserted.id = deleted.id</pre>• The names for the inserted and deleted tables are passed to the direct_connect as alias names, and the table name in the from clause is the actual name of the table in the DBMS accessed by the DirectConnect. <p>Server Class ASEnterprise •Servers in this class do not support access to remote <i>inserted</i> and <i>deleted</i> tables.</p> <p>Server Class ASAnywhere •Servers in this class do not support access to remote <i>inserted</i> and <i>deleted</i> tables.</p>

Server Class *ASIQ* •Servers in this class do not support access to remote *inserted* and *deleted* tables.

Server Class *sql_server* •Servers in this class do not support access to remote *inserted* and *deleted* tables.

Server Class *direct_connect* •The ability to support *inserted* and *deleted* tables is determined by a capability. If enabled, CIS will forward syntax referencing these tables to the DirectConnect.

- With version 12.0, the only DirectConnect supporting this capability is the DirectConnect for Oracle.

Server Class *db2* •Servers in this class do not support access to remote *inserted* and *deleted* tables.

Server Class *generic* •Servers in this class do not support access to remote *inserted* and *deleted* tables.

See also

See Also

deallocate cursor

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

Syntax `deallocate cursor cursor_name`

Usage 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 See Also

close, **declare cursor**, **fetch**, **open** in this chapter.

deallocate cursor in the *Adaptive Server Reference Manual*.

declare cursor

Description	Defines a cursor.
Syntax	<pre>declare <i>cursor_name</i> cursor for <i>select_statement</i> [for {read only update [of <i>column_name_list</i>]}]</pre>
Usage	<p>Usage</p> <ul style="list-style-type: none">• If the cursor specified by <i>cursor_name</i> contains references to proxy tables, Adaptive Server notifies Component Integration Services to establish a connection to the remote servers referenced by the proxy tables. <p>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.</p>
See also	<p>See Also</p> <p>close, deallocate cursor, fetch, open in this chapter.</p> <p>declare cursor in the <i>Adaptive Server Reference Manual</i>.</p>

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

Usage

- Component Integration Services processes the **delete** 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.
- Component Integration Services executes the **delete** 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 delete is executed.
- When Component Integration Services forwards the **delete** 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 **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 *sql_server* •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 the **delete** command 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* •Server's 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 **delete** command using method 2, described above, when the statement is complex.

- If the server is a DB2 system, use traceflag 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

Server Class *generic* •Server's of class *generic* 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 the *generic* server class
- Complex statements cause Component Integration Services to perform a **select** statement followed by the **delete** statement when qualifying rows are found.

See also

See Also

delete in the *Adaptive Server Reference Manual*.

drop database

Description	Removes one or more databases from Adaptive Server.
Syntax	drop database <i>database_name</i> [, <i>database_name</i>]...
Usage	<p>Usage</p> <ul style="list-style-type: none">For each database being dropped, Component Integration Services scans <i>sysobjects</i> to check for proxy tables in the database. Each proxy table that was not created with the existing keyword is dropped in the remote server that owns the object. <p>Server Class ASEnterprise •Component Integration Services issues a drop table command for each table that was not created with the existing keyword.</p> <p>Server Class ASAnywhere •Component Integration Services issues a drop table command for each table that was not created with the existing keyword.</p> <p>Server Class ASIQ •Component Integration Services issues a drop table command for each table that was not created with the existing keyword.</p> <p>Server Class sql_server •Component Integration Services issues a drop table command for each table that was not created with the existing keyword.</p> <p>Server Class direct_connect •Component Integration Services issues a drop table command for each table that was not created with the existing keyword.</p> <p>Server Class db2 •Component Integration Services issues a drop table command for each table that was not created with the existing keyword.</p> <p>Server Class generic •Component Integration Services issues the following RPC for each proxy table that was not created with the existing keyword:</p> <pre>gen_drop_table <i>table_name</i>, <i>owner_name</i>, <i>database_name</i></pre>
See also	<p>See Also</p> <p>drop database in the <i>Adaptive Server Reference Manual</i>.</p>

drop index

Description	Removes an index from a table in the current database.
Syntax	drop index <i>table_name.index_name</i> [, <i>table_name.index_name</i>]...
Usage	<p>Usage</p> <ul style="list-style-type: none"> • Component Integration Services processes the drop index command 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 the drop index command 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 the drop index command. If trace flag 11208 is turned on, the drop index command 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. <p>Server Class <i>ASEnterprise</i> •Component Integration Services forwards the following drop index syntax to a remote server configured as class <i>ASEnterprise</i>:</p> <pre>drop index <i>table_name.index_name</i></pre> <p>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.</p> <p>Server Class <i>ASAnywhere</i> •Component Integration Services forwards the following drop index syntax to a remote server configured as class <i>ASAnywhere</i>:</p> <pre>drop index <i>table_name.index_name</i></pre> <p>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.</p>

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 *sql_server* •Component Integration Services forwards the following **drop index** syntax to a remote server configured as class *sql_server*:

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*

Server Class *generic* •Component Integration Services forwards the following RPC to a remote server configured as class *generic*:

gen_drop_index *index_name, table_name, owner_name, database_name*

See also

See Also

drop index in the *Adaptive Server Reference Manual*.

drop table

Description	Removes a table definition and all of its data, indexes, triggers, and permissions from the database.
Syntax	drop table [[<i>database.</i>] <i>owner.</i>] <i>table_name</i> [, [[<i>database.</i>] <i>owner.</i>] <i>table_name</i>]..
Usage	<p>Usage</p> <ul style="list-style-type: none"> • 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. • When Component Integration Services forwards the drop 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. • If multiple tables are dropped in a single command, each table is sent as an individual drop table command. • A table in use by another user or process cannot be dropped and an error stating that the table is in use is returned. <p>Server Class <i>ASEnterprise</i> •Component Integration Services forwards the following drop table syntax to a remote server configured as class <i>ASEnterprise</i>:</p> <pre>drop table <i>database.owner.table_name</i></pre> <p>Server Class <i>ASAnywhere</i> •Component Integration Services forwards the following drop table syntax to a remote server configured as class <i>ASAnywhere</i>:</p> <pre>drop table <i>database.owner.table_name</i></pre> <p>Server Class <i>ASIQ</i> •Component Integration Services forwards the following drop table syntax to a remote server configured as class <i>ASIQ</i>:</p> <pre>drop table <i>database.owner.table_name</i></pre> <p>Server Class <i>sql_server</i> •Component Integration Services forwards the following drop table syntax to a remote server configured as class <i>sql_server</i>:</p> <pre>drop table <i>database.owner.table_name</i></pre>

Server Class *direct_connect* •Component Integration Services requests a capabilities response from a remote server with server class *direct_connect*, but support for **drop table** is not optional. The behavior of the DirectConnect is database dependent.

Server Class *db2* •Component Integration Services forwards the following **drop table** syntax to a remote server configured as class *db2*:

drop table owner.table_name

Server Class *generic* •Component Integration Services forwards the following RPC to a remote server configured as class *generic*:

gen_drop_table table_name, owner, database

See also

See Also

drop table in the *Adaptive Server Reference Manual*.

execute

Description	Runs a system procedure or a user-defined stored procedure.
Syntax	<pre>[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]</pre>
Usage	<p>Usage</p> <ul style="list-style-type: none"> • When the execute command 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 on-going transaction. The two methods are as follows: <ol style="list-style-type: none"> a 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. b 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 the set command or the sp_configure system procedure.
See also	<p>See Also</p> <p>“RPC Handling and Component Integration Services” on page 67.</p> <p>set in this chapter.</p> <p>execute in the <i>Adaptive Server Reference Manual</i>.</p>

fetch

Description

Returns a row or a set of rows from a cursor result set.

Syntax

```
fetch cursor_name [ into fetch_target_list ]
```

Usage

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 takes the following steps:

- 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 the **declare cursor** command 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 *sql_server* •For pre-version 10.0 SQL Server, Component Integration Services sends a language request to the remote server when the first **fetch** is received after the cursor is opened.

- For version 10.0 or later servers, Component Integration Services declares a cursor to the remote server by means of Client-Library.

Server Class *direct_connect* •Component Integration Services treats servers in class *direct_connect* as if they were version 10.0 or later of class *sql_server*.

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.

Server Class *generic* •Cursors are not supported by server class *generic*. Component Integration Services sends a language request to the remote server when the first **fetch** is requested.

- The configuration parameter **cis cursor rows** determines how many rows are returned from a single fetch sent to the remote server. If this number is greater than 1, the rows are buffered by Client-Library. Subsequent **fetch** requests retrieves rows from the buffer until it is empty, at which time Component Integration Services issues another **fetch** to the remote server.

See also

See Also

close, **deallocate cursor**, **declare cursor**, **open** in this chapter.

fetch in the *Adaptive Server Reference Manual*.

Functions

Description The following section defines the compatibility of the CIS server classes with the built-in ASE 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, CIS 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* (see the **select** command).

In the tables shown below, support for function by server class is indicated by a 'Y'; an '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. The aggregate functions are:

Table 4-10: Server Class Support for Aggregate Functions

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

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

Table 4-11: Server Class Support for Datatype Conversion Functions

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

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

Table 4-12: Server Class Support for Date Functions

Function	ASE	ASA	ASIQ	sql_serv	dir_con	db2	generic
dateadd	Y	Y	Y	Y	C	N	N
datediff	Y	Y	Y	Y	C	N	N
datename	Y	Y	N	Y	C	N	N
datepart	Y	Y	Y	Y	C	N	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 4-13: Server Class Support for Mathematical Functions

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

Functions

Security Functions Security functions return security-related information. The security functions are:

Table 4-14: Server Class Support for Security Functions

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

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

Table 4-15: Server Class Support for String Functions

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

System Functions System functions return special information from the database. The system functions are:

Table 4-16: Server Class Support for System Functions

Function	ASE	ASA	ASIQ	sql_serv	dir_con	db2	generic
col_length	Y	Y	N	Y	C	N	N
col_name	Y	Y	N	Y	C	N	N

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

Text and Image Functions

Text and image functions operate on text and image data. The text and image functions are:

Table 4-17: Server Class Support for Text and Image Functions

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

Functions

Function	ASE	ASA	ASIQ	sql_serv	dir_con	db2	generic
textvalid)	Y	Y	N	Y	C	N	N

insert

Description	Adds new rows to a table or view.
Syntax	<pre>insert [into] [database.[owner.]]{table_name view_name} [(column_list)] {values (expression [, expression]...) select_statement }</pre>
Usage	<p>Usage</p> <ul style="list-style-type: none"> • Component Integration Services processes the insert 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 insert 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. <p>Server Class ASEnterprise •insert commands using the values keyword are fully supported.</p> <ul style="list-style-type: none"> • 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 <i>quickpass mode</i>. Quickpass mode is not used if the select statement does not conform to all the quickpass rules for a select command (see “select” on page 156). • 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. <p>Server Class ASAnywhere •Handling of the insert statement is the same as for <i>ASEnterprise</i>.</p> <p>Server Class ASIQ •Handling of the insert statement is the same as for <i>ASEnterprise</i>.</p> <p>Server Class sql_server •Handling of the insert statement is the same as for <i>ASEnterprise</i>.</p> <p>Server Class direct_connect •insert commands using the values keyword are fully supported.</p>

- **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 capabilities 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.

Server Class generic • **insert** commands using the **values** keyword are supported for all valid datatypes in server class generic.

- **insert** commands using a **select** command are supported for all valid datatypes in server class generic. When using **insert** with a **select** command, Component Integration Services selects each row from the source tables, and then inserts the row into the target table.

See also

See Also

insert in the *Adaptive Server Reference Manual*.

open

Description	Opens a cursor for processing.
Syntax	<code>open cursor_name</code>
Usage	<p>Usage</p> <ul style="list-style-type: none">• 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 <i>Transact-SQL User's Guide</i>.• Adaptive Server returns an error message if the cursor is already open or if the cursor has not been created with the declare cursor statement.• 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	<p>See Also</p> <p>close, deallocate cursor, declare cursor, fetch in this chapter.</p> <p>open in the <i>Adaptive Server Reference Manual</i>.</p>

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	<p>Usage</p> <ul style="list-style-type: none">• When the Distributed Transaction Manager (DTM) is enabled, DTM handles all transaction processing for servers of server class <i>ASEnterprise</i> with a version of 12.0 or later.• prepare transaction is ignored for servers with a server class of <i>db2</i> or <i>generic</i>.• For all other server classes, Adaptive Server notifies Component Integration Services when it receives a <i>prepare transaction</i> 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 transaction command 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. <p>Server Class <i>ASEnterprise</i> •Transaction process for servers in class <i>ASEnterprise</i> with a version prior to 12.0 is identical to that of server class <i>sql_server</i> (release 10.0 or later).</p> <ul style="list-style-type: none">• When DTM is not enabled, transaction processing for all servers in class <i>ASEnterprise</i> is identical to that of server class <i>sql_server</i> (release 10.0 or later).• When the Adaptive Server receives notification to prepare a transaction, the Distributed Transaction Manager issues a PrepareXact RPC to all remote participants having a server class of <i>ASEnterprise</i>. When all remote participants have acknowledged the PrepareXact RPC, any local data changes are written to the database. <p>Server Class <i>ASAnywhere</i> •Transaction processing for servers in class <i>ASAnywhere</i> is identical to that of server class <i>sql_server</i> (release 10.0 or later).</p> <p>Server Class <i>ASIQ</i> •Transaction processing for servers in class <i>ASAnywhere</i> is identical to that of server class <i>sql_server</i> (release 10.0 or later).</p>

Server Class *sql_server* •Component Integration Services sends a **prepare transaction** command to each server in class *sql_server* that is version 10.0 or later.

- The **prepare transaction** command is not sent to the following types of servers:
 - Sybase IQ 11.x
 - Microsoft SQL Server
 - Pre-version 10.0 SQL Server
 - OmniSQL Server 10.1.2

Server Class *direct_connect* •Handling of the **prepare transaction** command for servers in class *direct_connect* is identical to that of server class *sql_server* (version 10.0 or later).

Server Class *db2* •Component Integration Services does not send the **prepare transaction** command to servers in class *db2*.

Server Class *generic* •Component Integration Services does not send the **prepare transaction** command to servers in class *generic*

See also

See Also

prepare transaction in the *Adaptive Server Reference Manual*.

readtext

Description	Reads text and image values, starting from a specified offset and reading a specified number of bytes or characters.
Syntax	<pre>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] }]]</pre>
Usage	<p>Usage</p> <ul style="list-style-type: none"> • 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. <p>Server Class ASEnterprise •Component Integration Services forwards the following syntax to the remote server when the underlying table is a proxy table:</p> <pre>readtext [[database.]owner.]table_name.column_name text_pointer offset size [using {chars characters}]</pre> <p>Server Class ASAnywhere •Handling of the readtext statement is the same as for <i>ASEnterprise</i>.</p> <p>Server Class ASIQ •Handling of the readtext statement is the same as for <i>ASEnterprise</i>.</p> <p>Server Class sql_server •Handling of the readtext statement is the same as for <i>ASEnterprise</i>.</p> <p>Server Class direct_connect •If the DirectConnect does not support text pointers, readtext cannot be sent and its use results in errors.</p>

- 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*.

Server Class *generic* •**readtext** is not supported since text and image datatypes are not supported for servers in class *generic*.

See also

See Also

readtext in the *Adaptive Server 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} [<i>transaction_name</i> <i>savepoint_name</i>]
Usage	<p>Usage</p> <ul style="list-style-type: none">• When the Distributed Transaction Manager (DTM) is enabled, DTM handles all transaction processing for servers of server class <i>ASEnterprise</i> with a version of 12.0 or later.• For all other server classes, Adaptive Server notifies Component Integration Services when it receives a rollback transaction command and Component Integration Services attempts to rollback work associated with remote servers in the current transaction.• Multiple remote servers can be involved in a single transaction, each with their 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 RPC's is included in the local transaction and can be rolled back if the remote server supports RPC's within transactions.• <i>transaction_name</i> and <i>savepoint_name</i> is not used by Component Integration Services in this release. <p>Server Class ASEnterprise •Transaction process for servers in class <i>ASEnterprise</i> with a version prior to 12.0 is identical to that of server class <i>sql_server</i> (release 10.0 or later).</p> <ul style="list-style-type: none">• When DTM is not enabled, transaction processing for all servers in class <i>ASEnterprise</i> is identical to that of server class <i>sql_server</i> (release 10.0 or later).• When the Adaptive Server receives notification to rollback a transaction, the Distributed Transaction Manager issues a RollbackXact RPC to all remote participants having a server class of <i>ASEnterprise</i>. <p>Server Class ASAnywhere •Transaction processing for servers in class <i>ASAnywhere</i> is identical to that of server class <i>sql_server</i> (version 10.0 or later).</p> <p>Server Class ASIQ •Transaction processing for servers in class <i>ASIQ</i> is identical to that of server class <i>sql_server</i> (version 10.0 or later).</p>

Server Class *sql_server* •When Component Integration Services receives notification that a transaction is to be rolled back, it checks the TRANSACTION ACTIVE state of all remote connections associated with the client application. For each connection with an active transaction, Component Integration Services sends a **rollback transaction**. If all remote servers respond with no error, Component Integration Services notifies the Adaptive Server that it can begin to roll back local work.

This process applies to version 10.0 or later, but not to the following servers represented by server class *sql_server* is:

- Pre-version 10.0 SQL Server
- Microsoft SQL Server (any version)
- Sybase IQ
- OmniConnect 10.1.2

For these types of servers, transaction handling is similar to server class *db2*, described below.

Server Class *direct_connect* •Transaction processing for servers in class *direct_connect* is identical to that of server class *sql_server* (version 10.0 or later).

Server Class *db2* •Transactions are supported only at the statement level for servers in class *db2*. When the internal state of a client connection indicates that there is an active transaction, Component Integration Services precedes each **insert**, **update** and **delete** command with a **begin transaction** command. It then issues a **commit** or **rollback transaction** (depending on the success or failure of the statement) immediately after the statement is complete.

Server Class *generic* •Transactions are supported only at the statement level for servers in class *db2*. When the internal state of a client connection indicates that there is an active transaction, Component Integration Services precedes each **insert**, **update** and **delete** command with a **gen_begin_xact** RPC. It then issues a **gen_commit_xact** or **gen_rollback_xact** RPC (depending on the success or failure of the statement) immediately after the statement is complete.

See also

See Also

rollback in the *Adaptive Server 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

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 Sybase System 10 and later versions of Adaptive Server Enterprise, 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 the **select** command 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 rerun the query with traceflag 11216 turned on to receive the expected sort order. This traceflag 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 CIS-generated **insert** command.

Server Class sql_server •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 *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 parameterized 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 traceflag 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.

Server Class *generic* •Component Integration Services only forwards the syntax to servers in class *generic* as is documented in the *Generic Access Module Reference Manual*. Quickpass mode is not used when servers in class *generic* are involved in a query.

- If the **select...into** format is used and the **into** table is accessed through the *generic* interface, bulk inserts are not used. Instead, a separate connection is used to handle the text of a CIS-generated **insert** command.

select

See also

See Also

select in the *Adaptive Server 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 <i>Adaptive Server Reference Manual</i> .
Syntax	<pre>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 <i>value</i></pre>
Usage	<p>Usage</p> <ul style="list-style-type: none"> • Normally, all outbound RPCs are routed through Adaptive Server's 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's 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 ON, then transactions that involve participants that are not DTM-enabled servers (i.e. non-ASE 12.0 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 prior release 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 will affect the behavior of parameters to RPCs that are managed by CIS.

- Any RPC parameter that is of type binary(16) or varbinary(16) is assumed to be a **textptr** if the setting for **textptr_parameters** 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** (Please refer to the discussion of **textptr_parameter** handling in Chapter 3 of this book).
- The **textptr** is expanded into as many 32k chunks of text 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 data type. If this type is not supported, then the expansion is disabled, and the setting of **textptr_parameters** is assumed to be **off**.

See also

See Also

set in the *Adaptive Server Reference Manual*.

setuser

Description	Allows a Database Owner to impersonate another user.
Syntax	<code>setuser ["user_name"]</code>
Usage	<p>Usage</p> <ul style="list-style-type: none">• 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 <i>not</i> 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.• Prior to 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 on sp_addexternlogin, see the <i>Adaptive Server Reference Manual</i>).
See also	<p>See Also</p> <p>setuser in the <i>Adaptive Server Reference Manual</i>.</p>

truncate table

Description	Removes all rows from a table.
Syntax	<code>truncate table [[<i>database.</i>]owner.]table_name</code>
Usage	<p>Comments</p> <ul style="list-style-type: none">• 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. <p>Server Class <i>ASEnterprise</i> •Component Integration Services forwards the truncate table command to servers of class <i>ASEnterprise</i>.</p> <p>Server Class <i>ASAnywhere</i> •Component Integration Services forwards the truncate table command to servers of class <i>ASAnywhere</i>.</p> <p>Server Class <i>ASIQ</i> •Component Integration Services forwards the truncate table command to servers of class <i>ASIQ</i>.</p> <p>Server Class <i>sql_server</i> •Component Integration Services forwards the truncate table command to servers of class <i>sql_server</i>.</p> <p>Server Class <i>direct_connect</i> and <i>sds</i> •If the remote server has requested DB2 syntax, the following statement is forwarded:</p> <pre>delete from [owner.]table_name</pre> <p>Otherwise, Transact-SQL syntax is sent:</p> <pre>truncate table [[database.]owner.]table_name</pre> <p>Server Class <i>db2</i> •The following syntax is forwarded to the remote server:</p> <pre>delete from [owner.]table_name</pre> <p>Server Class <i>generic</i> •Component Integration Services processes the truncate table command using an RPC call to the procedure gen_truncate_table.</p>
See also	truncate table in the <i>Adaptive Server Reference Manual</i> .

update

Description	Changes data in existing rows, either by adding data or by modifying existing data.
Syntax	<pre> 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 </pre>
Usage	<ul style="list-style-type: none"> • 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: <ol style="list-style-type: none"> 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 <i>quickpass mode</i>. 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 *sql_server* • 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]
```

- Server's 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 traceflag 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.

Server Class *generic* The following syntax is supported by servers of class *generic*:

```
update [[database.]owner.]{table_name | view_name}
set [[[database.]owner.]{table_name.|view_name.}
    column_name1 =
        {expression1|NULL}
    [, column_name2 =
        {expression2|NULL}]...
[where search_conditions]
```

- Server's of class *generic* 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 the *generic* server class

Component Integration Services processes the **update** command using method 2, described above, when the statement is complex.

- When an **update** statement contains a complex search condition, a **select** statement, or 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 *Adaptive Server 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 *sql_server* •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 it 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.

Server Class *generic* •When an **update statistics** command is issued, Component Integration Services issues the following command:

```
select count(*) from table_name
```

- No distribution statistics are calculated for tables owned by servers of this class.

See also

update statistics in the *Adaptive Server Reference Manual*.

writetext

Description	Permits non-logged, interactive updating of an existing text or image column.
Syntax	<code>writetext [[database.]owner.]table_name.column_name text_pointer [with log] data</code>
Usage	<ul style="list-style-type: none">• 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 Client Library commands using the connection established to the remote server:<pre>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)</pre> <p>Server Class <i>ASEnterprise</i> •The writetext command is processed using a separate connection to the remote server.</p> <p>Server Class <i>ASAnywhere</i> •The writetext command is processed using a separate connection to the remote server.</p> <p>Server Class <i>ASIQ</i> •The writetext command is processed using a separate connection to the remote server.</p> <p>Server Class <i>sql_server</i> •The writetext command is processed using a separate connection to the remote server.</p> <p>Server Class <i>direct_connect</i> •If the DirectConnect supports text pointers, Component Integration Services treats the DirectConnect as if it were a server in class <i>sql_server</i>.</p> <p>Server Class <i>db2</i> •writetext is not supported for tables owned by servers in this class.</p> <p>Server Class <i>generic</i> •writetext is not supported for tables owned by servers in this class.</p>
See also	writetext in the <i>Adaptive Server Reference Manual</i> .

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
- To provide lists of information that you can gather before calling Technical Support, which will help resolve your problem quickly
- To provide you with a greater understanding of Component Integration Services

Error Messages and the *Troubleshooting Guide* should also be used for troubleshooting. While this appendix provides troubleshooting tips for most frequently asked Component Integration Services questions, *Error Messages* lists all error messages with a one-line recovery procedure; the *Troubleshooting Guide* provides tips on SQL Server problems that are not specific to Component Integration Services.

For the most up-to-date information on troubleshooting and technical tips, refer to Sybase's electronic services. See "Other Sources of Information" on page ix.

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:

```
4050 cis extension not enabled or installed
```

Do the following:

- 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 will not take effect until the server is restarted.

- Check the error log. If Component Integration Services loaded correctly, you will see the following line at the start of the error log:

```
Distributed services option loaded.
```

If there was a problem loading Component Integration Services, the message stating the problem is displayed instead. Contact Sybase Technical Support to correct the problem. (See “If You Need Help” on page 182.)

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 will be 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 the **sp_addserver** stored procedure, 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.
```

Check the following:

- Is your environment set up correctly?

To test this, try to connect directly to the remote server using **isql** or a similar tool. Do this by following these steps:

- Log into the machine where Component Integration Services is running.
- 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.

- 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.

- Is the remote server up and running?

Log into the machine where the remote server is located to verify the server is running. If the server is running, continue through this list. If the server is hung, restart the server and try your query again.

- Is the entry for the remote server in the interfaces file correct:
 - Is the machine name the correct name for the machine the software is loaded on?
 - If the interfaces file is a text file, do the query and master lines start with a tab and not spaces?
 - Is the port number available? Check the *services* file in the */etc* directory to ensure that the port number is not reserved for another process.

If the port is available, is it already in use? To determine this on UNIX, run the command:

```
netstat -a
```

Login Failed

If the remote server is accessed, but the login name and password are not correct, the following messages display:

```
Login failed
11206 Unable to connect to server server_name.
```

Check to 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. It is not possible to 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 the following:

- Has the object 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 (see “Mapping Remote Objects to Local Proxy Tables” on page 3-4).

- If the object has been defined in Component Integration Services, is the definition 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 correctly using **sp_addobjectdef**.

- If the local object definition is correct, check the table on the remote server:
 - Are permissions set to allow access to both the database and table?
 - Has the database been marked suspect?
 - Is the database available?
 - Can you access the remote table using a native tool (for example, SQL on Rdb or 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 the statement:

```
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 the **create table** command.

If **create existing table** was used to create the table in Component Integration Services:

- 1 Use the **drop table** command 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 prior to 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 do a **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 CIS

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 the **describe** command 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 the statement:

```
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 the **create table** command.

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 table** 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.

An alternative method if **create table** was used to define the object is to turn on trace flag 11208. This trace flag prevents the **create index** statement 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 will look 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 will be 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.

If You Need Help

Index

A

Access methods 8
access_server server class 83
 connection management 35
 datatype conversions 115
 with text and image datatypes 33
Adding
 columns to a table 90, 92
 rows to a table or view 147
 space to a database 90
Aliases, user
 remote logins 53
Allocating resources with sp_configure 60
alter database command 90
alter table command 90, 92
@@textsize global variable 31
auto identity database option 17
Automatic connections 20

B

Backups 79
bcp (bulk copy utility)
 for text and image datatypes 32
begin transaction command 25
 proxy tables and 97

C

Changes, canceling. See rollback command 154
Changing
 database size 90
 remote tables 90, 92
Checkpoint process
 See also Recovery\ 154
Savepoints 154
cis bulk insert batch size configuration parameter 62

cis connect timeout configuration parameter 63
cis cursor rows configuration parameter 63
cis packet size configuration parameter 63
cis rpc handling configuration parameter 63
Client-Library functions 10
 connection management 35
 ct_send_data 32
close command 99
Clustered indexes
 See also Indexes 119
Columns
 adding to table 90, 92
 creating indexes on proxy table 118
commit command 102
 remote servers and 25
commit work command. See commit command 103
Component Integration Services
 configuring and tuning 59
 enabling 5
 running 5
 setting up 5, 52
 users 4
Configuration (Server)
 Component Integration Services 4, 52, 59
Configuration parameters
 Component Integration Services 61, 63
connect to command 18, 54
connect to option, grant 19
Connections
 listing of remote 74
 management of 35
 maximum of CIS 62
 permission 19
 physical and logical 67
 timeouts 63
 verification 25, 54
Constraints
 preventing 76
Conventions
 used in manuals ix

- Converting remote server datatypes 16
 - server class db2 117
 - Copying
 - text and image datatypes 32
 - create existing table command 11, 15
 - datatype conversions and 16
 - example 16
 - proxy tables 105, 107
 - create index command 118
 - query plan for remote tables 44
 - create proxy_table command
 - mapping proxy tables to remote tables 121
 - create table command
 - proxy tables 120
 - query plan 43
 - remote tables 11, 15
 - Creating
 - indexes on proxy tables 118
 - proxy tables 105, 107, 120
 - ct_send_data Client-Library function 32
 - Cursor result set
 - returning rows 140
 - Cursors
 - deallocating 128
 - fetching remotely 140
 - opening 149
 - row count, setting 63
- D**
- Data modification
 - text and image with writetext 172
 - update 165
 - Database syntax, using native. See Passthrough mode 17
 - Databases
 - increasing size of 90
 - Datatype conversions 86
 - remote servers 16
 - server class db2 117
 - server class direct_connect or access_server 115
 - db2 server class
 - connection management 35
 - datatype conversions 117
 - with text and image datatypes 34
 - db2 syntax mode, Open Server applications that support 124
 - dbcc (Database Consistency Checker) 69, 76
 - DB-Library programs
 - prepare transaction 150
 - dbmoretext DB-Library function 32
 - dbwritetext DB-Library function 32
 - deallocate cursor command
 - remote servers and 125
 - Deallocating cursors 128
 - declare cursor command 128
 - Defining
 - default storage locations 11
 - indexes 15
 - remote objects 10, 11, 54, 56
 - remote servers 10, 52, 54
 - storage locations of remote objects 11, 54
 - tables 11, 15, 17
 - delete command
 - remote tables 129
 - Deleting
 - See also Dropping 130
 - direct_connect server class 83
 - connection management 35
 - with text and image datatypes 33
 - DirectCONNECT servers 5
 - disconnect command 19
 - drop database command
 - remote servers 133
 - drop index command
 - proxy tables 134
 - query plan for remote tables 45
 - drop table command
 - proxy tables 136
 - query plan for remote tables 44
 - Dropping
 - databases from remote servers 134
 - indexes on proxy tables 135
 - proxy tables 137
 - rows from a table 130
- E**
- enable cis configuration parameter 62
 - Enabling Component Integration Services 5

Error logging of text and image datatypes 32
 Event logging 75
 execute command
 RPCs 139
 Extending database storage 90
 External logins 53

F

fetch command
 proxy tables 139
 Fetching cursors
 proxy tables 139
 Files
 interfaces 52
 sql.ini file 52

G

generic server class 85
 connection management 35
 grant command
 passthrough connections 19
 grant connect to command 19

I

IDENTITY columns 17
 image datatype 30
 bulk copy to remote servers 32
 converting 31
 entering values 32
 error logging 32
 padding 31
 pattern matching 31
 pointer values in readtext 152
 restrictions 30
 with server class sql_server 32
 with server class db2 34
 with server class direct_connect or access_server 33
 with server class sql_server 32
 writetext to 172

Impersonating a user. See setuser command 163

Indexes

 defining 15
 dropping from proxy tables 134
 update statistics on 170
 updating 75
 insert command
 proxy tables 141
 Integrity of data
 remote tables and 45
 Interface to remote servers 9
 Interfaces file
 adding remote servers 52

J

Joins
 between remote tables 56, 58

L

like keyword 31
 Local tables. See Proxy tables 45
 Location for remote tables 11
 lock timeout interval configuration parameter 69
 Logging
 events 75
 text or image data 172
 Logging in
 to remote servers 10
 Logical connections 67
 Logins
 external 53
 See also Remote logins\ 163
 Users 163

M

Mapping
 remote objects 54, 56
 Markers, user-defined. See Placeholders\ 154
 max cis remote connections configuration parameter 62

max cis remote servers configuration parameter 62

Memory

releasing with deallocate cursor 128

Memory usage report 74

Modes, trusted/untrusted 46

Modifying

databases 90

N

Names

local 11

setuser 163

Native database syntax, using. See Passthrough mode 17

Nested select statements. See select command\ 156

Non-logged operations 172

O

Object types 9

rpc as read only tables 28

open command 148

Opening cursors 149

Optimization

defining existing tables and 16

quickpass mode 37, 130, 147, 156, 165

remote tables 39, 76

update statistics 39

Original identity, resuming an. See setuser command 163

Outbound remote procedure calls 63

P

Packets, network

size for remote servers 63

Pages, data

See also Index pages\ 119

Table pages 119

Passthrough connection permission 19

Passthrough mode 17

connect to command 18, 54

connect to command 106

sp_autoconnect system procedure 19

sp_passthru system procedure 20

sp_remotesql system procedure 21

patindex string function 31

Pattern matching

remote tables 31

with text datatype 31

Performance

configuration parameters 60

query optimization 35

remote tables 39, 76

Permissions

passthrough connections 19

Physical connections 67

prepare transaction command 25

proxy tables and 150

Processing remote procedure calls 67

Proxy tables

mapping 54, 56

mapping to remote tables with create proxy_table

121

triggers 45

Q

Queries

execution settings 161

Query optimization 35, 42

disabling 76

Query plans 42

create table 43

remote tables and 42

Quickpass mode 37, 130, 147, 156, 165

R

readtext command

errors from 32

remote tables and 151

Recovery 79

disabling CIS at start-up 76

Reference information

Transact-SQL commands for CIS 88

Referential integrity 45

remcon option, dbcc 74

- Remote connection listing 74
 - Remote logins. See External logins 53
 - Remote objects
 - default storage location 11
 - defining 10, 11
 - individual storage location 54
 - mapping 54, 56
 - Remote procedure calls 27
 - handling outbound 63
 - transactional 25
 - transmitting 67
 - Remote servers
 - adding 52, 54
 - connection verification 54
 - definition 10
 - interface to 9
 - interfaces file entries 52
 - joins 56, 58
 - logging in 10
 - security issues 46
 - setting up external logins 53
 - transaction management 25
 - Remote tables
 - accessing 8
 - joins 56, 58
 - Removing. See Dropping 130
 - Reports
 - in-memory SRVDES structures 74
 - memory usage 74
 - remote connections 74
 - Resource allocation (sp_configure) 60
 - Results
 - cursor result set 140
 - rollback command
 - remote servers and 153
 - rollback command\ 154
 - rollback transaction command. See rollback command 154
 - rollback work command. See rollback command 154
 - Rows, table
 - See also select command 156
 - update 165
 - RPCs. See Remote procedure calls 25
 - Running a procedure with execute
 - remote servers 139
 - Running Component Integration Services 4, 52
 - rusage option, dbcc 74
- ## S
- Savepoints 154
 - sds server class 84
 - Search conditions
 - remote tables 31
 - Security
 - issues for remote servers 46
 - Security issues 46
 - select command
 - remote tables 155, 159
 - Server class access_server 83
 - connection management 35
 - datatype conversions 115
 - with text and image datatypes 33
 - Server class db2 83
 - connection management 35
 - datatype conversions 117
 - with text and image datatypes 34
 - Server class direct_connect 83
 - connection management 35
 - with text and image datatypes 33
 - Server class generic 85
 - connection management 35
 - Server class sds 84
 - Server class sql_server 83
 - connection management 35
 - with text and image datatypes 32
 - Server classes 8
 - See also individual server class names 8
 - access_server 83
 - db2 83
 - direct_connect 83
 - generic 85
 - sds 84
 - sql_server 83
 - set command
 - See also individual set options 160
 - remote queries 160
 - Setting up Component Integration Services 4, 52
 - setuser command
 - remote objects and 162
 - sp_addexternlogin system procedure 53

- sp_addobjectdef system procedure 11, 55
 - sp_addserver system procedure 53, 57
 - sp_autoconnect system procedure 20
 - sp_capabilities system procedure 87
 - sp_configure system procedure 5, 60
 - sp_defaultloc system procedure 11
 - sp_passthru system procedure 20
 - sp_remotelogin system procedure 46
 - sp_remotesql system procedure 21
 - Space
 - adding to database 90
 - sql.ini file 52
 - sql_server server class
 - connection management 35
 - srvdes option, dbcc 74
 - Start-up recovery, disabling 76
 - Statistics
 - update statistics 169
 - Storage location for remote tables 11
 - Stored procedures
 - executing remote 139
 - Subqueries 156
 - Syntax, using native database. See Passthrough mode 17
 - sysconfigures system table
 - updating values in 61
 - syssservers system table
 - remote servers for Component Integration Services 53, 82
 - System activities
 - setting query processing option for 161
- T**
- Tables
 - changing remote 90, 92
 - creating proxy 110
 - creating remote 122
 - dropping proxy 137
 - read-only 29
 - remote access 8
 - remote, joins 56, 58
 - Tables, proxy
 - defining 11, 15, 17
 - triggers 45
 - text datatype 30
 - bulk copy to remote servers 32
 - converting 31
 - entering values 32
 - error logging 32
 - padding 31
 - pattern matching 31
 - restrictions 30
 - with server class db2 34
 - with server class direct_connect or access_server 33
 - with server class sql_server 32
 - @@textsize global variable 31
 - textsize option, set 31
 - Timeout, connect 63
 - Trace flags 75
 - traceon/traceoff option, dbcc 75
 - Transaction canceling. See rollback command 154
 - Transaction management 22, 26
 - Transactional remote procedure calls 25
 - transactional_rpc on option, set command 26
 - Transactions
 - ending with commit 103
 - preparing 149
 - See also Batch processing\ 154
 - User-defined transactions 154
 - Transmitting remote procedure calls 67
 - Triggers 45
 - truncate table command
 - query plan for remote tables 45
 - remote tables 163
 - Trusted mode 46
 - Tuning
 - Component Integration Services 59
- U**
- Undoing changes. See rollback command 154
 - update command
 - remote tables 164
 - update statistics command
 - defining existing tables and 16
 - obtaining complete distribution statistics 76
 - remote tables 39, 76, 170
 - Updating
 - image datatype 33

- indexes 75
- text datatype 33
- writetext 172
- User-defined stored procedures, executing
 - RPCs 139
- User-defined transactions
 - See also Transactions 97
 - begin transaction 97
 - ending with commit 103
- Users 163
- Users of Component Integration Services 4
- using option, readtext
 - errors from 32

V

- Variables, configuration. See Configuration parameters
 - 60
- Verifying connectivity 54

W

- Wildcard characters 31
- Work session, set options for 161
- Write operations
 - logging text or image 172
- writetext command
 - remote tables 171

