Sybase SQL Server™
Utility Programs for UNIX

Sybase SQL Server Release 11.0.x
Document ID: 30475-01-1100-02
Last Revised: December 15, 1995

Principal author: Server Publications Group

Contributing authors: Anneli Meyer

Document ID: 30475-01-1100

This publication pertains to Sybase SQL Server Release 11.0.x of the 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.

Document Orders

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.

Sybase Trademarks

APT-FORMS, Data Workbench, DBA Companion, Deft, GainExposure, Gain *Momentum*, Navigation Server, PowerBuilder, Powersoft, Replication Server, SA Companion, SQL Advantage, SQL Debug, SQL Monitor, SQL SMART, SQL Solutions, SQR, SYBASE, the Sybase logo, Transact-SQL, and VQL are registered trademarks of Sybase, Inc. Adaptable Windowing Environment, ADA Workbench, AnswerBase, Application Manager, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, APT Workbench, Backup Server, Bit-Wise, Client-Library, Client / Server Architecture for the Online Enterprise, Client / Server for the Real World, Client Services, Configurator, Connection Manager, Database Analyzer, DBA Companion Application Manager, DBA Companion Resource Manager, DB-Library, Deft Analyst, Deft Designer, Deft Educational, Deft Professional, Deft Trial, Developers Workbench, DirectCONNECT, Easy SQR, Embedded SQL, EMS, Enterprise Builder, Enterprise Client / Server, Enterprise CONNECT, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Gain Interplay, Gateway Manager, InfoMaker, Interactive Quality Accelerator, Intermedia Server, IQ Accelerator, Maintenance Express, MAP, MDI, MDI Access Server, MDI Database Gateway, MethodSet, Movedb, Navigation Server Manager, Net-Gateway, Net-Library, New Media Studio,

Restricted Rights

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

Table of Contents

3. Using the *isql* Utility

Index

List of Tables

Preface

SQL Server Utility Programs for UNIX is a guide to the Sybase SQL Server™ utility programs available on all UNIX platforms. Utility programs are commands that you invoke directly from the operating system.

Audience

This manual is for anyone using Transact-SQL® and SQL Server release 11.0. It is a reference manual, and assumes that you have basic knowledge of how to use the UNIX operating system and SQL Server.

How to Use This Book

This manual includes the following chapters:

- Chapter 1, "Utility Programs Reference," consists of alphabetized reference pages for all of the available utility commands.
- Chapter 2, "Using bcp to Transfer Data to and from SQL Server," discusses **bcp** in detail.
- Chapter 3, "Using the isql Utility," discusses using the interactive SQL utility.

The examples in this book are based on the *pubs2* sample database. Ask your System Administrator how to get a clean copy of *pubs2*.

Related Documents

Other manuals that you may find useful are:

- SQL Server installation and configuration guide, which describes the installation procedures for SQL Server and the operating system-specific system administration, security administration, and tuning tasks.
- *SQL Server Performance and Tuning Guide*, which explains how to tune SQL Server for maximum performance. The book includes information about database design issues that affect performance, query optimization, how to tune SQL Server for

very large databases, disk and cache issues, and the effects of locking and cursors on performance.

- *SQL Server Reference Manual*, which contains detailed information on all of the commands and system procedures discussed in this manual.
- *SQL Server Reference Supplement*, which contains a list of the Transact-SQL reserved words, definitions of system tables, a description of the *pubs2* sample database, a list of SQL Server error messages, and other reference information that is common to all the manuals.
- *SQL Server Security Administration Guide*, which explains how to use the security features provided by SQL Server to control user access to data. The manual includes information about how to add users to SQL Server, how to give them controlled access to database objects and procedures, and how to manage remote SQL Servers.
- *SQL Server Security Features User's Guide*, which explains how to use the security features of SQL Server.
- *SQL Server System Administration Guide*, which provides in-depth information about administering servers and databases. The manual includes instructions and guidelines for managing physical resources and user and system databases, and specifying character conversion, international language, and sort order settings.
- *Transact-SQL User's Guide*, which documents Transact-SQL, the Sybase enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system.
- *What's New in Sybase SQL Server Release 11.0?*, which describes the new features in SQL Server release 11.0.
- *Open Client DB-Library/C Reference Manual*, a collection of manual pages and code samples for the SQL Server interface library, Open Client DB-Library™.

Conventions Used in This Manual

The following paragraphs detail the typographic conventions used in this manual.

Font and Syntax Conventions

The font and syntax conventions in this reference are as follows:

Element	Example
Command names, command option names, utility names, utility flags, and other keywords are **bold**.	**select**
Database names, datatypes, file names and path names are in *italics*.	*master* database
Variables, or words that stand for values that you fill in, are in *italics*.	`select` `from` *`table_name`* `where` *`search_conditions`*
Parentheses are to be typed as part of the command.	`compute` *`row_aggregate`* `(`*`column_name`*`)`
Curly braces indicate that you must choose at least one of the enclosed options. Do not type the braces.	`{cash, check, credit}`
Brackets mean choosing one or more of the enclosed options is optional. Do not type the brackets.	`[anchovies]`
The vertical bar means you may select only one of the options shown.	`{die_on_your_feet | live_on_your_knees | live_on_your_feet}`
The comma means you may choose as many of the options shown as you like, separating your choices with commas to be typed as part of the command.	`[extra_cheese, avocados, sour_cream]`
An ellipsis (...) means that you can **repeat** the last unit as many times as you like.	`buy thing = price [cash | check | credit] [, thing = price [cash | check | credit]]...` You must buy at least one thing and give its price. You may choose a method of payment: one of the items enclosed in square brackets. You may also choose to buy additional things: as many of them as you like. For each thing you buy, give its name, its price, and (optionally) a method of payment.

- Syntax statements (displaying the syntax and all options for a command) appear as follows:

  `showserver` ***`parameter`***

or, for a command with more options:

```
buildmaster [-d physicalname] [-c cntrltype]
     [-s ] [-m]
```

In syntax statements, commands are in normal font and options are in lowercase: normal font for flags, italics for user-supplied values.

- Examples showing utility commands appear in bold, as follows:

  ```
  bcp -v
  ```

- Examples of output from the computer appear as follows:

```
pub_id  pub_name                city         state
------- ------------------      -----------  -----
0736    New Age Books           Boston       MA
0877    Binnet & Hardley        Washington    DC
1389    Algodata Infosystems    Berkeley      CA

(3 rows affected)
```

If You Need Help

Help with your Sybase software is available in the form of documentation and Technical Support.

Each Sybase installation has a designated person who may contact Technical Support. If you cannot resolve your problem using the manuals, ask the designated person at your site to contact Sybase Technical Support.

1 Utility Programs Reference

This chapter consists of reference pages for the utility programs listed below.

The UNIX system shell interprets the utility program commands. Place characters with special meaning to the shell, such as the backslash (\), asterisk (*), slash (/), and spaces, in quotes. You can precede some special characters with the backslash (\) to "escape" them. This prevents the shell from interpreting the special characters.

Utility	Description
backupserver	Executable form of the Backup Server™ program
bcp	Copies rows in a database table to or from an operating system file in a user-specified format
buildmaster	Builds the master device and creates the *master*, *model*, and *tempdb* databases on the device
dataserver	Executable form of the SQL Server program
defncopy	Copies definitions for specified views, rules, defaults, triggers, procedures, or reports from a database to an operating system file or from an operating system file to a database
isql	Interactive SQL parser to SQL Server
langinstall	Installs one new language on the SQL Server
showserver	Shows SQL Servers and Backup Servers that are currently running on the local machine
startserver	Starts a SQL Server or a Backup Server

backupserver

Function

The executable form of the Backup Server program.

Syntax

backupserver [-C*server_connections*] [-S*b_servername*]
[-I*interfaces_file*] [-e*error_log_file*]
[-M*sybmultbuf_binary*] [-N*network_connections*]
[-T*trace_value*] [-L*Sybase_language_name*]
[-J*Sybase_character_set_name*] [-c*tape_config_file*]

or

backupserver [-v]

Parameters

-C*server_connections* – specifies the number of server connections for the Backup Server. The Backup Server requires:

- Two connections for each dump session
- One connection for each load session
- One connection for volume change messages

Allow a maximum of three times the number of expected concurrent dump and load sessions. The default value is 20 server connections.

-S*b_servername* – specifies the name of the Backup Server to start. The default is SYB_BACKUP. This entry must specify the name of a Backup Server in the interfaces file.

-I*interfaces_file* – specifies the name and location of the interfaces file to search when connecting to Backup Server. If **-I** is omitted, **backupserver** looks for a file named *interfaces* in the directory pointed to by your SYBASE environment variable.

-e*error_log_file* – specifies the name and location of the Backup Server error log file used to report Open Server™ internal errors, **sybmultbuf** errors, errors that halt the Backup Server, and errors for disconnected sessions. All other errors are sent to the **notify** destination specified in the **dump database**, **dump transaction**, **load database**, and **load transaction** commands.

-M*sybmultbuf_binary* – specifies the full path name of the **sybmultbuf** executable. Use this option only when starting Backup Server from a directory other than the */bin* directory of the SYBASE installation tree, or when using a diagnostic version of **sybmultbuf**.

-N*network_connections* – specifies the number of total network connections (DBPROCESSes) that the master Backup Server can originate. The default value is 25.

-T*trace_value*– interprets *trace_value* as a bit mask (base-2 number). The 1 bits in *trace_value* correspond to Open Server trace flags to turn on. If you specify more than one **-T** option on the command line, the final **-T** value overrides the values from earlier **-T** options. The *trace_value* must be a positive integer.

-L*Sybase_language_name* – specifies the default language for Backup Server. If not specified, Backup Server uses the locale specified by the LC_ALL or LANG environment variables. If these variables are not set, Backup Server searches for the "default" entry in *locales.dat*.

-J*Sybase_character_set_name* – specifies the default character set for Backup Server.

-c*tape_config_file* – specifies the name and location of the tape configuration file to search for tape device configuration information before doing a **dump database** or a **dump transaction**. If you do not specify **-c**, the default path name for the tape configuration file is *$SYBASE/backup_tape.cfg*.

-v – prints the version number and copyright message of the **backupserver** software, then exits.

Comments

- Start Backup Server with the **startserver** command rather than by directly executing the **backupserver** program. If you need to change any of the default values, edit the *RUN_servername* file in your *SYBASE* installation directory. See the **startserver** manual page for details.
- If you do not specify a Backup Server name with the **-S** option, and have not set the environment variable DSLISTEN, **backupserver** uses the default Backup Server name SYB_BACKUP. The value of the DSLISTEN environment variable overrides this default value, and the **-S** option overrides both the default and the value in DSLISTEN.

- Whenever possible, the Backup Server and any SQL Servers that dump or load directly through the Backup Server should share the same interfaces file. The interfaces file that Backup Server uses must contain entries for:
  - Backup Server
  - Any other Backup Servers with which this Backup Server communicates
- Trace flags cause the Backup Server to print information regarding its operation while it is running, for debugging problems in the Backup Server. See the *Open Server Server-Library/C Reference Manual* for more details on trace flags. The UNIX Backup Server does not support use of the Open Server-defined SRV__TR symbols for **-T**.
- If Backup Server cannot find the *locales* and *charsets* directories specified by the **-L** and **-J** options, or if these options specify an incorrect language and character set combination, Backup Server issues an error message and uses the default language and character set.

See Also

Utilities	**startserver**

bcp

Function

Copies a database table to or from an operating system file in a user-specified format.

Syntax

```
bcp [[database_name.]owner.]table_name {in | out}
   datafile
   [-m maxerrors] [-f formatfile] [-e errfile]
   [-F firstrow] [-L lastrow] [-b batchsize] [-n]
   [-c [-N]] [-t field_terminator] [-r row_terminator]
   [-U username] [-P password] [-I interfaces_file]
   [-S server] [-a display_charset]
   [-q datafile_charset] [-z language] [-v]
   [-A packet_size] [-J client_charset]
   [-T text_or_image_size] [-E ] [-X] [-y sybase_dir]
```

Parameters

database_name – is optional if the table being copied is in your default database. Otherwise, you must specify a database name.

owner – is optional if you or the Database Owner own the table being copied. If you do not specify an owner, **bcp** first looks for a table of that name that you own, and then looks for one owned by the Database Owner. If another user owns the table, you must specify the owner's name or the command fails.

table_name – is the name of the database table to copy. The table name cannot be a Transact-SQL reserved word.

in | **out** – is the direction of the copy. **in** indicates a copy from a file into the database table; **out** indicates a copy to a file from the database table.

datafile – is the full path name of an operating system file. The path name can be from 1–255 characters in length.

-m *max_errors* – is the maximum number of nonfatal errors permitted before **bcp** aborts the copy. **bcp** discards each row that it cannot insert (due to a data conversion error, or an attempt to insert a null value into a column that does not allow them), counting each rejected row as one error. If you do not include this option, **bcp** uses a default value of 10.

-f *format_file* – is the full path name of a file with stored responses from a previous use of **bcp** on the same table. After you answer **bcp**'s format questions, it asks you if you want to save your answers in a format file; creation of the format file is optional. The default file name is *bcp.fmt*. The **bcp** program can refer to a format file when copying data, so that you do not have to duplicate your previous format responses interactively. Use this option only when you previously created a format file that you want to use now for a copy in or out. If this option is not used, **bcp** queries you for format information interactively.

-e *errfile* – is the full path name of an error file where **bcp** stores any rows that it was unable to transfer from the file to the database. Error messages from the **bcp** program appear on your terminal. **bcp** creates an error file only when you specify this option. If you specify this option, and **bcp** does not encounter any nonfatal errors, it does not create the error file.

-F *firstrow* – is the number of the first row to copy (default is the first row).

-L *lastrow* – is the number of the last row to copy (default is the last row).

-b *batchsize* – is the number of rows per serial batch of data copied (the default is to copy all the rows in one batch). Each batch is a transaction that is committed at the end of the batch. Batching applies only when bulk copying in; it has no effect on bulk copying out.

-n – performs the copy operation using native (operating system) formats. This option does not prompt for each field. Files in native data format are not human-readable.

-c – performs the copy operation with *char* datatype as the default. Use this format if you are sharing data between platforms. This option does not prompt for each field; it uses *char* as the default storage type, no prefixes, \t (tab) as the default field terminator, and \n (newline) as the default row terminator.

-N – skips the IDENTITY column. Use this option when copying in if your host data file does not include a placeholder for the IDENTITY column values, or when copying data out and you do not want to include the IDENTITY column information in the host file.

-t *field_terminator* – specifies the default field terminator.

-r *row_terminator* – specifies the default row terminator.

➤ ***Note***

When specifying terminators from the command line with the **-t** or **-r** options, you must escape characters that have special significance to the UNIX operating system (see example 1 on page 1-10). Either place a backslash in front of the special character or enclose it in quotes. This is not necessary when **bcp** prompts you (interactive mode).

-U *username* – specifies a SQL Server login name. If you do not specify *username*, **bcp** uses the current user's operating system login name.

-P *password* – specifies a SQL Server password. If you do not specify **-P** *password*, **bcp** prompts for a password. If your password is null, place the **-P** flag at the end of the command line by itself.

-I *interfaces_file* – specifies the name and location of the interfaces file to search when connecting to SQL Server. If you do not specify **-I**, **bcp** looks for a file named *interfaces* in the directory that your SYBASE environment variable specifies.

-S *server* – specifies the name of the SQL Server to connect to. If you specify **-S** with no argument, **bcp** uses the server that your DSQUERY environment variable specifies.

-a *display_charset* – allows you to run **bcp** from a terminal where the character set differs from that of the machine on which **bcp** is running. (See "Converting Character Sets Between SQL Server and Clients" in the *System Administration Guide* for more information about changing character sets.) **-a** in conjunction with **-J** specifies the character set translation file (*.xlt* file) required for the conversion. Use **-a** without **-J** only if the client character set is the same as the default character set.

-q *datafile_charset* – allows you to run **bcp** to copy character data to or from a file system that uses a character set different from the client character set. **-q** in conjunction with **-J** specifies the character set translation file (*.xlt* file) required for the conversion.

In Japanese language environments, the **-q** flag translates Hankaku Katakana (half-width characters) into Zenkaku Katakana (full-width characters). Use with the argument "zenkaku" and with the **-J** flag to indicate the client's Japanese

character set (**sjis** or **eucjis**). The *zenkaku.xlt* file was designed to translate **only** from terminal display to SQL Server, **not** from SQL Server to the terminal.

➤ ***Note***

The ascii_7 character set is compatible with all character sets. If either the SQL Server or client character set is set to ascii_7, any 7-bit ASCII character is allowed to pass between client and server unaltered. Other characters produce conversion errors. Character set conversion issues are covered more thoroughly in "Converting Character Sets Between SQL Server and Clients" in the *System Administration Guide*.

-z *language* – is the official name of an alternate language that the server uses for date formats and to display **bcp** prompts and messages. Without the **-z** flag, **bcp** uses the server's default language. Add languages to a SQL Server at installation, or afterwards with either the utility **langinstall** or the stored procedure **sp_addlanguage**.

-v – displays the version number of **bcp** and a copyright message and returns to the operating system.

-A *packet_size* – specifies the network packet size to use for this **bcp** session. For example:

```
bcp -A 2048
```

sets the packet size to 2,048 bytes for this **bcp** session. *size* must be between the values of the **default network packet size** and **max network packet size** configuration parameters, one-third the size of the **additional network memory** configuration parameter, and it must be a multiple of 512.

Use larger-than-default network packet sizes to improve the performance of large bulk copy operations.

-J *client_charset* – specifies the character set to use on the client. **bcp** uses a filter to convert input between *client_charset* and the SQL Server character set.

-J *client_charset* requests that SQL Server convert to and from *client_charset*, the character set used on the client.

-J with no argument sets character set conversion to NULL. No conversion takes place. Use **-J** with no argument if the client and server use the same character set.

Omitting **-J** sets the character set to a default for the platform. Table 1-1 lists platform defaults.

Table 1-1: Default character sets for different platforms

Platform	Default Character Set
Sun, Digital, Pyramid, AT&T, RS6000 / AIX, others	iso_1
HP	roman8
OS / 2	cp850
Macintosh	mac

The default may not necessarily be the character set that the client is using. See "Converting Character Sets Between SQL Server and Clients" in the *System Administration Guide* and the SQL Server installation and configuration guide for more information about character sets and the associated flags.

-T *text_or_image_size* – allows you to specify, in bytes, the maximum length of *text* or *image* data that SQL Server sends. The default is 32K. If a *text* or *image* field is larger than the value of **-T** or the default, **bcp** does not send the overflow.

-E – explicitly specifies values for a table's IDENTITY column when copying data in. To use an explicit IDENTITY column value from the host file for each row, specify the **-E** flag when copying data into a table.

By default, when you bulk copy data into a table with an IDENTITY column, the host file must contain a placeholder for the IDENTITY column (a value of 0 is recommended). The server assigns the row a unique, sequential IDENTITY column value as **bcp** inserts each row into the table. If the number of inserted rows exceeds the maximum possible IDENTITY column value, SQL Server returns an error.

The **-E** option has no effect on bulk copying out.

-X – when connecting to the server, **bcp** initiates the login with client-side password encryption. **bcp** (the client) specifies to the server that password encryption is desired. The server sends back an encryption key, which **bcp** uses to encrypt your password, and the server uses the key to authenticate your password when it arrives.

If **bcp** crashes, the system creates a core file that contains your password. If you did not use the encryption option, the password appears in plain text in the file. If you used the encryption option, your password is not readable.

-y *sybase_dir* – specifies a *SYBASE* directory other than the *$SYBASE* directory.

Examples

1. In the following example, the **-c** option copies data out of the *publishers* table in character format (using *char* for all fields). The **-t** *field_terminator* option ends each field with a comma, and the **-r** *row_terminator* option ends each line with a Return. **bcp** prompts only for a password. The first backslash before the final "r" escapes the second so that one backslash prints.

   ```
   bcp pubs2..publishers out pub_out -c -t , -r \\r
   ```

2. In the following example, **bcp** copies data from the *publishers* table to a file named *pub_out* for later reloading into SQL Server. Pressing Return accepts the defaults that the prompts specify. The same prompts appear when copying data into the *publishers* table.

   ```
   bcp pubs2..publishers out pub_out
   Password:

   Enter the file storage type of field pub_id [char]:
   Enter prefix length of field pub_id [0]:
   Enter length of field pub_id [4]:
   Enter field terminator [none]:

   Enter the file storage type of field pub_name
   [char]:
   Enter prefix length of field pub_name [1]:
   Enter length of field pub_name [40]:
   Enter field terminator [none]:

   Enter the file storage type of field city [char]:
   Enter prefix length of field city [1]:
   Enter length of field city [20]:
   Enter field terminator [none]:

   Enter the file storage type of field state [char]:
   Enter prefix length of field state [1]:
   Enter length of field state [2]:
   Enter field terminator [none]:
   ```

```
Do you want to save this format information in a
file? [Y-n] y
Host filename [bcp.fmt]: pub_form

Starting copy...

3 rows copied.
Clock Time (ms.): total = 300   Avg = 100  (10.00
rows per sec.)
```

3. To copy this data back into SQL Server using the saved format file, *pub_form*, use the following command:

   ```
   bcp pubs2..publishers in pub_out -f pub_form
   ```

4. To see examples of datatypes, enter "?" at the prompt:

   ```
   Enter the file storage type of field 'pub_id'
   ['char']:?
   Invalid column type. Valid types are:
   <cr>: same type as SQL Server column.
       c : char
       T : text
       i : int
       s : smallint
       t : tinyint
       f : float
       m : money
       b : bit
       d : datetime
       x : binary
       I : image
       D : smalldatetime
       r : real
       M : smallmoney
       n : numeric
       e : decimal
   ```

   Enter the single letter exactly as it appears above.

5. The following example copies a data file created with a character set used on a VT200 terminal into the *pubs2..publishers* table. The **-q** flag translates it. The **-z** flag displays **bcp** messages in French.

   ```
   bcp pubs2..publishers in vt200_data -J iso_1 -q
   vt200 -z french
   ```

6. The following example specifies that you are using a Macintosh, running **bcp** on a workstation that is using roman8, with the file system on another machine that uses iso_1:

   ```
   bcp pubs2..publishers in -a mac -J roman8 -q iso_1
   ```

7. The following example specifies that SQL Server send 40K of *text* or *image* data using a packet size of 4096 bytes:

   ```
   bcp pubs2..publishers out -T 40960 -A 4096
   ```

Comments

- See Chapter 2, "Using bcp to Transfer Data to and from SQL Server" for a more in-depth discussion of **bcp**.
- **bcp** provides a convenient and high-speed method for transferring data between a database table and an operating system file. It is capable of reading or writing files in a wide variety of formats. When copying in from a file, **bcp** appends data to an existing database table; when copying out to a file, **bcp** overwrites any previous contents of the file.
- Upon completion, **bcp** informs you of the number of rows of data successfully copied, the number of rows (if any) that it could not copy, the total time the copy took, the average amount of time that it took to copy one row (in milliseconds), and the number of rows copied per second.
- **bcp** copies in each batch in a single **insert** transaction. If SQL Server rejects any row in the batch, the entire **insert** rolls back. The default is to copy all rows in a single batch; change this with the **-b** option. SQL Server then considers each batch to be a single **bcp** operation, writes each batch to a separate data page, and continues to the next batch whether or not the **insert** succeeded.
- When copying data into a table, **bcp** observes any defaults defined for columns and user-defined datatypes. However, **bcp** ignores rules in order to load data at the fastest possible speed.
- When you send host data files to sites that use terminals different from your own, inform them of the *datafile_charset* that you used to create the files.
- You cannot use named pipes to **bcp** files in or out.

Fast and Slow bcp

- The **bcp** program is optimized to load data into tables that do not have indexes or triggers associated with them. It loads data into tables without indexes or triggers at the fastest possible speed,

logging only page allocations and without logging the insertion of rows.

- To use the fast version of **bcp**, the System Administrator or Database Owner must first set the **select into/bulkcopy** option on with the system procedure **sp_dboption**. If the option is not set on, and you try to copy data into a table that has no indexes or triggers, SQL Server generates an error message. You do not need to set this option in order to copy data out to a file, or to copy data into a table with indexes or triggers.

  By default, the **select into/bulkcopy** option is off in newly created databases. To change the default situation, turn this option on in the *model* database.
- Fast **bcp** runs more slowly while a **dump database** is taking place.
- When you copy data into a table that has one or more indexes (including indexes that are implicitly created using the unique integrity constraint of **create table**) or triggers, a slower version of **bcp**, which logs the row inserts as well as the page allocations, is automatically used. **bcp** does not enforce rules or other integrity constraints defined for a table.

  The log can grow very large during slow **bcp**. After the bulk copy completes, back up your database with **dump database**, then truncate the log with **dump transaction**.

➤ ***Note***

The performance penalty for copying data into a table that has indexes or triggers in place can be severe. If you are copying in a very large number of rows, it may be faster to drop all the indexes and triggers first with **drop index** (or **alter table** for indexes created as a unique constraint) and **drop trigger**, set the database option, copy the data into the table, re-create the indexes and triggers, and then dump the database. Remember to allocate disk space for the construction of indexes and triggers—for a clustered index, about 1.2 times the amount of space needed for the data, in addition to the space needed for the data.

Responding to bcp Prompts

When you copy data in or out using the **-n** (native format) or **-c** (character format) option, **bcp** prompts you only for your password, unless you supplied it with the **-P** option. If you do not supply either

the **-n** or **-c** options, **bcp** prompts you for information for each field (column) in the table.

- Each prompt displays a default value, in brackets, which you can accept by pressing Return. The prompts include:
  - The file storage type, which can be character or any valid SQL Server datatype
  - The prefix length, which is an integer indicating the length in bytes of the following data
  - The storage length of the data in the file
  - The field terminator, which can be any character string

  The row terminator is the field terminator of the last field in the table or file.
- The bracketed defaults represent reasonable values for the datatypes of the field in question. For the most efficient use of space when copying out to a file:
  - Use the default prompts
  - Copy all data in their table datatypes
  - Use prefixes as indicated
  - Do not use terminators
  - Accept the default lengths

  The following table shows the defaults and possible alternate responses:

Table 1-2: bcp prompts, their defaults and responses

Prompt	Default Provided	Possible Responses
File Storage Type	Use database storage type for most fields except: *char* for *varchar* *binary* for *varbinary*	*char* to create or read a human-readable file; any SQL Server datatype where implicit conversion is supported.
Prefix Length	0 for fields defined with *char* datatype (not storage type) and all fixed-length datatypes 1 for most other datatypes 2 for *binary* and *varbinary* saved as *char* 4 for *text* and *image*	0 if no prefix is desired; defaults are recommended in all other cases.

Table 1-2: bcp prompts, their defaults and responses (continued)

Prompt	Default Provided	Possible Responses
Storage Length	For *char* and *varchar*, use defined length. For *binary* and *varbinary* saved as *char*, use double the defined length. For all other datatypes, use maximum length needed to avoid truncation or data overflow.	Default values, or greater, are recommended.
Field or Row Terminator	none	Up to 30 characters

➤ ***Note***

Be careful copying data in native format from different versions of SQL Server because they do not always have the same datatypes.

- A prefix length is a 1-, 2-, or 4-byte integer that represents the length of each data value in bytes. It immediately precedes the data value in the host file.
- A field terminator string can be up to 30 characters long. The most common terminators are a tab (entered as "\t" and used for all columns except the last one), and a newline (entered as "\n" and used for the last field in a row). Other terminators are: "\0" (the null terminator), "\" (backslash), and "\r" (Return). When choosing a terminator, be sure that its pattern does not appear in any of your character data, because **bcp** always looks for the first possible terminator.

  A terminator or prefix affects the actual length of data transferred:

  - If the length of an entry being copied out to a file is less than the storage length (other than *char*, *nchar*, and *binary* data), it is followed immediately by the terminator or the prefix for the next field, and not padded to the full storage length.
  - When copying in, **bcp** copies data until either it reaches the number of bytes indicated in the "Length" prompt or it encounters the terminator. If the data exceeds the specified length, **bcp** flushes the data until it finds the terminator. Without terminators, the table storage length is strictly observed.
- Fields stored as *char* (except *char*, *nchar*, and *binary* fields) instead of their database datatypes take less file storage space with the

default length and prefix or a terminator. **bcp** can use either a terminator or a prefix to determine the most efficient use of storage space. **bcp** suggests the maximum amount of storage space required for each field as the default. For *char* or *varchar* data, **bcp** accepts any length.

- Fields defined in the database as *char*, *nchar*, and *binary* are always padded with spaces (null bytes for binary) to the full length defined in the database. *timestamp* data is treated as *binary(8)*.

  If data in *varchar* and *varbinary* fields is longer than the length you specify for copy out, **bcp** silently truncates the data in the file at the specified length.

- The following tables show the interaction of prefix lengths, terminators, and field length on the information in the file. "P" indicates the prefix in the stored table. "T" indicates the terminator, and dashes (-) show appended spaces. An ellipsis (...) indicates that the pattern repeats for each field. The field length is 8 for each column, and "string" represents the 6-character field each time.

Table 1-3: SQL Server char data

	Prefix length = 0	**Prefix length 1, 2 or 4**
No terminator	string--string--...	Pstring--Pstring--...
Terminator	string--Tstring--T...	Pstring--TPstring--T...

Table 1-4: Other datatypes converted to char storage

	Prefix length = 0	**Prefix length 1, 2 or 4**
No terminator	string--string--...	PstringPstring...
Terminator	stringTstringT...	PstringTPstringT...

- File storage length is generally the maximum amount of data to be transferred for the column, plus terminators and prefixes.
- Because **bcp** considers any data column that can contain null values to be variable length, use either a length prefix or terminator to denote the length of each row of data.

Datatypes and Storage Types

- **bcp** can copy data out to a file either as its database datatype or as any datatype for which implicit conversion is supported for the datatype in question. **bcp** copies user-defined datatypes as their base datatype or as any datatype for which implicit conversion is supported. For more information on datatype conversions, see **dbconvert** in the *Open Client DB-Library/C Reference Manual*.
- Note that the file storage type and length of a column do not have to be the same as the type and length of the column in the database table. (If types and formats copied in are incompatible with the structure of the database table, the copy fails.)
- Data written to a host file in its native format preserves all of its precision. *datetime* and *float* values preserve all of their precision even when they are converted to character format. SQL Server stores *money* values to a precision of one ten-thousandth of a monetary unit. However, when *money* values are converted to character format, their character format values are recorded only to the nearest two places.
- Before copying data that is in character format from a file into a database table, check the datatype entry rules in the "Datatypes" section of the *SQL Server Reference Manual*. Character data that is being copied into the database with **bcp** must conform to those rules. Note especially that dates in the undelimited *(yy)yymmdd* format may result in overflow errors if the year is not specified first.

Messages

- `Error in attempting to determine the size of a pair of translation tables.:'stat' utility failed.`

  The character translation file(s) named with the **-a** or **-q** parameter is missing, or you mistyped the name(s).
- `Slow bulk copy is being used because the table has triggers or indexes.`
- `You cannot use the -g and -E flags together.`
- `partition number` *`partition_number`* `does not exist in table` *`table_name`*.

See Also

Chapter 2, "Using bcp to Transfer Data to and from SQL Server."

buildmaster

Function

Builds the master device and creates the *master*, *model*, and *tempdb* databases on the device.

Syntax

```
buildmaster [-d disk] [-c cno] [-s size] [-m]
   [-q] [-x]
```

or

```
buildmaster -v
```

Parameters

-d *disk*– is the physical name of the raw disk partition or operating system file where the master device resides.

-c *cno*– is the controller number for the master device. Together, *cno* and *disk* specify the device. The default value for *cno* is 0. Do not change this value unless instructed to do so by Sybase Technical Support.

-s *size* – is the size of the master device in 2K blocks. For example, a *size* of "5120" creates a 10MB master device. **buildmaster** verifies that the value you specify for this parameter does not exceed the space available to the master device, unless you use the **-m** option.

-m – rewrites only the *master* database, without changing the configuration block or initializing the master device. Use this option when the *master* database is corrupted but the other databases on the master device are undamaged.

-q – does not clear unallocated pages in *master* and *model* databases ("quick" version). This option has no effect when used with **-x**.

-x – rewrites only the *model* database, without changing the configuration block or initializing the master device. Use this option when the *model* database is corrupted and you cannot load it successfully from a backup. If you modified *model*, you must restore it from a backup after reinitializing it with this option.

-v – prints the version number and copyright message for **buildmaster**, and exits.

Examples

1. **`buildmaster -d /dev/rsd1f -s8704`**

   Initializes the raw device */dev/rsd1f* as a 17MB master device, and creates the system databases *master*, *model*, and *tempdb* on the device.

Comments

- Use **buildmaster** only when SQL Server is shut down.
- The **buildmaster** program initializes the specified device (database device) as a SQL Server master device, and builds the *master* and *model* databases on it.
- The SQL Server installation program runs **buildmaster** and builds an initial *master* database on the database device you specify in answer to the program's prompts.
- If you run **buildmaster** with no options, it prompts for the information listed below. **You must enter a response for each prompt**.

  ```
  master disk name?
  master disk controller number?
  master disk size?
  configuration only? (y or n)
  databases only? (y or n)
  ```

  Answer "n" to "configuration only". "databases only" is the same as **-m.**
- If you rebuild the *model* database, you must run the *installmodel* script, located in *$SYBASE/scripts. installmodel* sets up the necessary permissions for the *model* database. To run the script enter the following series of commands:

  ```
  cd $SYBASE/scripts
  setenv DSQUERY server_name
  isql -Usa -P -Sserver_name < installmodel
  ```
- If you rebuild the *master* database, and you do not have a database dump or cannot load the dump, you must:

  1. Start SQL Server using **startserver.**
  2. If your *sybsystemprocs* database is undamaged, run **disk reinit** and **disk refit** to restore the system tables entries. Otherwise, create a *sybsystemprocs* database on any device or on an operating system file. This database should be at least 16 MB.

3. Run the **installmaster** script. To run the script, use the following commands:

```
cd $SYBASE/scripts
setenv DSQUERY server_name
isql -Usa -P -Sserver_name < installmaster
```

The **installmaster** script installs the system stored procedure tables such as *spt_values* in the *master* database and creates the system procedures in *sybsystemprocs*. If *sybsystemprocs* does not exist, **installmaster** tries to create the database.

◆ ***WARNING!***

Do not run the installmaster script repeatedly without dropping the *sybsystemprocs* database. Running installmaster repeatedly can change the distribution of index values in such a way that the *sysprocedures* table will require much more disk space to store the same amount of data. To avoid this problem, drop and re-create the *sybsystemprocs* database before running installmaster.

- The password to the default "sa" account reverts to null after you run **buildmaster -m**, and the account is unlocked. Loading a backup of the *master* database restores the "sa" password and account lock state to what they were when the dump was taken.

See Also

Utilities	**startserver**

dataserver

Function

The executable form of the SQL Server program.

Syntax

```
dataserver -ddevicename [-cconfigurationfile]
   [-eerrorlogfile] [-m]
   [-rmastermirror] [-Msharedmem_directory]
   [-iinterfaces_file_directory] [-sservername]
   [-psso_login_name]
```

Parameters

-d*devicename* – is the full path name of the device for the *master* database. The *master* database device must be writable by the user who starts SQL Server. The default *master* database device name is *d_master*.

-c*configurationfile* – specifies the full path name of a SQL Server configuration file. Use this option to start SQL Server with the configuration values in the specified configuration file. See Chapter 11, "Starting SQL Server with a Configuration File" in the *System Administration Guide* for further information on using this option.

-e*errorlogfile* – is the full path name of the error log file for SQL Server system-level error messages.

-m – starts SQL Server in single user mode.

-r*mastermirror* – starts the mirror of the *master* database device. Use this option to start SQL Server if the *master* database device has been damaged.

-M*sharedmem_directory* – places shared memory files in the specified directory instead of in the default location, *$SYBASE*. If *sharedmem_directory* starts with "/", the directory name is assumed to be absolute. Otherwise, the directory name is interpreted relative to *$SYBASE*.

-s*servername* – specifies the name of the SQL Server to start. If **-s** is omitted, a server named SYBASE is started.

-i*interfaces_file_directory* – specifies the directory location of the interfaces file to search when connecting SQL Server. If **-i** is omitted, **dataserver** looks for a file named *interfaces* in the directory pointed to by your SYBASE environment variable.

-p*sso_login_name* – specifies the login name of a System Security Officer when starting SQL Server, for the purposes of getting a new password for that account. SQL Server generates a random password, displays it, encrypts it, and saves it in *master..syslogins* as that account's new password.

Comments

- Start SQL Server with the **startserver** command rather than by directly executing the **dataserver** program.
- SQL Server derives its running environment from values in the *sysconfigures* system table. Use the system procedure **sp_configure** to see or change configuration values.
- Because SQL Server passwords are encrypted, you cannot recover forgotten passwords. If all System Security Officers lose their passwords, the **-p** option generates a new password for a System Security Officer account. Start SQL Server with **-p**, immediately log into SQL Server with the new random password and execute **sp_password** to reset your password to a more secure one.
- After you have finished running the SQL Server installation program, be sure to set the file permissions on the **dataserver** executable to limit who can execute it.

defncopy

Function

Copies definitions for specified views, rules, defaults, triggers, or procedures from a database to an operating system file or from an operating system file to a database.

➤ ***Note***

The **defncopy** utility cannot copy table definitions or reports created with Report Workbench.

Syntax

```
defncopy [-U username][-P password] [-S server ]
   [-I interfaces_file] [-a display_charset]
   [-J client_charset] [-z language] [-X]
   {in filename dbname | out filename dbname
       [owner.]objectname [[owner.]objectname]...}
defncopy [-v]
```

Parameters

-U *username* – allows you to specify a login name. Login names are case-sensitive. If you do not specify *username*, **defncopy** uses the current user's operating system login name.

-P *password* – allows you to specify your password. If you do not specify **-P**, **defncopy** prompts for your password.

-S *server* – allows you to specify the name of the SQL Server to connect to. If you specify **-S** with no argument, **defncopy** looks for a server named SYBASE. If you do not specify **-S**, **defncopy** uses the server specified by your DSQUERY environment variable.

-I *interfaces_file* – allows you to specify the name and location of the interfaces file to search when connecting to SQL Server. If you do not specify **-I**, **defncopy** looks for a file named *interfaces* in the directory that your SYBASE environment variable specifies.

-a *display_charset* – allows you to run **defncopy** from a terminal whose character set differs from that of the machine on which **defncopy** is running. **-a** in conjunction with **-J** specifies the character set translation file (*.xlt* file) required for the conversion. Use **-a**

without **-J** only if the client character set is the same as the default character set.

In Japanese language environments, the **-q** flag is required to translate Hankaku Katakana (half-width characters) into Zenkaku Katakana (full-width characters). Use with the argument "zenkaku" and with the **-J** flag to indicate the client's Japanese character set (**sjis** or **eucjis**). The *zenkaku.xlt* file was designed to translate **only** from terminal display to SQL Server, **not** from SQL Server to the terminal.

➤ ***Note***

The ascii_7 character set is compatible with all character sets. If either the SQL Server or client character set is set to ascii_7, any 7-bit ASCII character is allowed to pass between client and server unaltered. Other characters produce conversion errors. (Character set conversion issues are covered more thoroughly in "Converting Character Sets Between SQL Server and Clients" in the *System Administration Guide*.)

-J *client_charset* – specifies the character set to use on the client. A filter converts input between *client_charset* and the SQL Server character set.

-J *client_charset* requests that SQL Server convert to and from *client_charset*, the client's character set.

-J with no argument sets character set conversion to null. No conversion takes place. Use this if the client and server are using the same character set.

Omitting **-J** sets the character set to a default for the platform. The default may not necessarily be the character set that the client is using. (See "Converting Character Sets Between SQL Server and Clients" in the *System Administration Guide* for more information about character sets and the associated flags.)

-z *language* – is the official name of an alternate language that the server uses to display **defncopy** prompts and messages. Without the **-z** flag, **defncopy** uses the server's default language. Add languages to a SQL Server at installation, or afterwards with the utility **langinstall** or the stored procedure **sp_addlanguage**.

-X – initiates the login with client-side password encryption in this connection to the server. **defncopy** (the client) specifies to the server that password encryption is desired. The server sends back an

encryption key, which **defncopy** uses to encrypt your password, and the server uses the key to authenticate your password when it arrives.

If **defncopy** crashes, the system creates a core file which contains your password. If you did not use the encryption option, the password appears in plain text in the file. If you used the encryption option, your password is not readable.

in | out – specifies the direction of definition copy.

filename – specifies the name of the operating system file destination or source for the definition copy. The copy out overwrites any existing file.

dbname – specifies the name of the database to copy the definitions from or to.

objectname – specifies name(s) of database object(s) for **defncopy** to copy out. Do not use *objectname* when copying definitions in.

-v – displays the version and copyright message of **defncopy** and returns to the operating system.

Comments

- Invoke the **defncopy** program directly from the operating system. **defncopy** provides a non-interactive way of copying out definitions (**create** statements) for views, rules, defaults, triggers, or procedures from a database to an operating system file. Alternatively, it copies in all the definitions from a specified file.
- You must have **select** permission on the *sysobjects* and *syscomments* tables to copy out definitions; you do not need permission on the object itself.

  You may not have **select** permission on the *text* column of the *syscomments* table if the System Security Officer has reset the **allow select on syscomments.text column** parameter with the system procedure **sp_configure**. This reset restricts **select** permission to the object owner and the System Administrator. This restriction is required in order to run SQL Server in the evaluated configuration. In this case, the object owner or a System Administrator must execute **defncopy** to copy out definitions.
- You must have the appropriate **create** permission for the type of object you are copying in. Objects copied in belong to the copier. A System Administrator copying in definitions on behalf of a

user must log in as that user to give the user proper access to the reconstructed database objects.

- The **in** *filename* or **out** *filename* and the database name are required and must be unambiguously stated. For copying out, use file names that reflect both the object's name and its owner.
- **defncopy** ends each definition that it copies out with the comment

  ```
  /* ### DEFNCOPY: END OF DEFINITION  */
  ```

  Definitions you create as text must end with this comment, or **defncopy** cannot copy them in successfully.
- Enclose values specified to **defncopy** in quotation marks if they contain characters that could be significant to the shell.

isql

Function

Interactive SQL parser to SQL Server.

Syntax

```
isql [-e] [-F] [-p] [-n] [-v] [-X] [-Y]
   [-a display_charset] [-c cmdend] [-E editor]
   [-h headers] [-H hostname] [-i inputfile]
   [-I interfaces_file] [-J client_charset]
   [-l login_timeout] [-m errorlevel]
   [-o outputfile] [-P password] [-s colseparator]
   [-S server] [-t timeout] [-U username]
   [-w columnwidth] [-y sybase_directory]
   [-z language] [-A size]
```

To terminate a command: **go**

To clear the query buffer: **reset**

To call the default editor: **vi**

To execute an operating system command: **!! *command***

To exit from **isql**: **quit** or **exit**

Parameters

-e – echoes input.

-F – enables the FIPS flagger. With this option, the server flags any non-SQL standard commands sent.

-p – prints out performance statistics.

-n – removes numbering and the prompt symbol (>) from input lines.

-v – prints the version and copyright of the **isql** software that you are using, and exits.

-X – initiates the login connection to the server with client-side password encryption. **isql** (the client) specifies to the server that password encryption is desired. The server sends back an encryption key, which **isql** uses to encrypt your password, and the server uses the key to authenticate your password when it arrives.

If **isql** crashes, the system creates a core file which contains your password. If you did not use the encryption option, the

password appears in plain text in the file. If you used the encryption option, your password is not readable.

-Y – tells the SQL Server to use chained transactions.

-a *display_charset* – allows you to run **isql** from a terminal whose character set differs from that of the machine on which **isql** is running. **-a** in conjunction with **-J** specifies the character set translation file (*.xlt* file) required for the conversion. Use **-a** without **-J** only if the client character set is the same as the default character set.

In Japanese language environments, the **-q** flag is required to translate Hankaku Katakana (half-width characters) into Zenkaku Katakana (full-width characters). Use with the argument "zenkaku" and with the **-J** flag to indicate the client's Japanese character set (**sjis** or **eucjis**). The *zenkaku.xlt* file was designed to translate **only** from terminal display to SQL Server, **not** from SQL Server to the terminal.

➤ ***Note***

The ascii_7 character set is compatible with all character sets. If either the SQL Server's or client's character set is set to ascii_7, any 7-bit ASCII character is allowed to pass between client and server unaltered. Other characters produce conversion errors. See "Converting Character Sets Between SQL Server and Clients" in the *System Administration Guide* for more information on character set conversion.

-c *cmdend* – changes the command terminator. By default, terminate commands and send them to SQL Server by typing "go" on a line by itself. When you change the command terminator, do not use SQL reserved words or control characters.

-E *editor* – specifies an editor other than the default editor **vi**.

-h *headers* – specifies how many rows to print between column headings. The default prints headings only once for each set of query results.

-H *hostname* – sets the client hostname.

-i *inputfile* – specifies the name of an operating system file to use for input to **isql**. The file must contain command terminators ("go" by default).

Specifying the parameter as follows:

```
-i inputfile
```

is equivalent to:

```
< inputfile
```

If you use **-i** and do not specify your password on the command line, **isql** prompts you for it.

If you use **<** *inputfile* and do not specify your password on the command line, you must specify your password as the first line of the input file.

-I *interfaces_file* – specifies the name and location of the interfaces file to search when connecting to SQL Server. Without **-I**, **isql** looks for a file named *interfaces* in the directory specified by your SYBASE environment variable.

-J *client_charset* – specifies the character set to use on the client. **-J** *client_charset* requests that SQL Server convert to and from *client_charset*, the character set used on the client. A filter converts input between *client_charset* and the SQL Server character set.

-J with no argument sets character set conversion to null. No conversion takes place. Use this if the client and server use the same character set.

Omitting **-J** sets the character set to a default for the platform. The default may not necessarily be the character set that the client is using. (See "Converting Character Sets Between SQL Server and Clients" in the *System Administration Guide* for more information about character sets and the associated flags.)

-l *login_timeout* – specifies the maximum timeout value allowed when connecting to SQL Server.

-m *errorlevel* – customizes the error message display. For errors of the severity level specified or higher only the message number, state, and error level display; no error text appears. For error levels lower than the specified level, nothing appears.

-o *outputfile* – specifies the name of an operating system file to store the output from **isql**. Specifying the parameter as follows:

```
-o outputfile
```

is similar to:

```
> outputfile
```

-P *password* – specifies your current SQL Server password. If you do not specify the **-P** flag, **isql** prompts for a password. If your password is NULL, use the **-P** flag at the end of the command line without any password.

-s *colseparator* – resets the column separator character, which is blank by default. To use characters that have special meaning to the operating system (for example, " | ", ";", "&", "<", ">"), enclose them in quotes or precede them with a backslash.

-S *server* – specifies the name of the SQL Server to connect to. **isql** looks this name up in the interfaces file. If you specify **-S** with no argument, **isql** looks for a server named SYBASE. Without **-S**, **isql** looks for the server specified by your DSQUERY environment variable.

-t *timeout* – specifies the number of seconds before a SQL command times out. If you do not specify a timeout, a command runs indefinitely. This affects commands issued from within **isql**, not the connection time. The default timeout for logging into **isql** is 60 seconds.

-U *username* – specifies a login name. Logins are case-sensitive.

-w *columnwidth* – sets the screen width for output. The default is 80 characters. When an output line reaches its maximum screen width, it breaks into multiple lines.

-y *sybase_directory* – specifies a *SYBASE* directory other than the default *$SYBASE* directory.

-z *language* – is the official name of an alternate language to display **isql** prompts and messages. Without **-z**, **isql** uses the server's default language. You can add languages to a SQL Server at installation, or add them afterwards with the utility **langinstall** or the stored procedure **sp_addlanguage**.

-A *size* - specifies the network packet size to use for this **isql** session For example:

```
isql -A 2048
```

sets the packet size to 2,048 bytes for this **isql** session. *size* must be between the values of the **default network packet size** and **max network packet size** configuration parameters, one-third the size of the **additional network memory** configuration parameter, and it must be a multiple of 512.

Examples

1. **isql -Ujoe -Pabracadabra**

   ```
   1> select *
   2> from authors
   3> where city = "Oakland"
   4> vi
   ```

   Puts you in a text file where you can edit the query. When you write and save the file, you are returned to **isql**. The query appears; type **go** on a line by itself to execute it.

2. **isql -U alma**

   ```
   Password:

   1> select *
   2> from authors
   3> where city = "Oakland"
   4> reset
   1> quit
   ```

   reset clears the query buffer. **quit** returns you to the operating system.

3. **isql -a mac -J roman8**

   Specifies that you are running **isql** from a Macintosh against a server that is using the roman8 character set.

Comments

- To use **isql** interactively, give the command **isql** (and any of the optional flags) at your operating system prompt. The **isql** program accepts SQL commands and sends them to SQL Server. The results are formatted and printed on standard output. Exit **isql** with **quit** or **exit.**
- Send a command to SQL Server by typing the command terminator (**go** by default) at the beginning of a line. You may follow the command terminator with an integer to specify how many times to run the command. For example, to execute this command 100 times, type:

  ```
  select x = 1
  go 100
  ```

  The results appear once at the end of execution.

- If you enter an option more than once on the command line, **isql** uses the last value. For example, if you enter the following command:

  ```
  isql -c"." -csend
  ```

  "send", the second value for **-c**, overrides ".", the first value. This enables you to override any aliases you set up.

- To call an editor on the current query buffer, enter its name as the first word on a line. Define your preferred callable editor by specifying it with the EDITOR environment variable. If EDITOR is not defined, the default is **vi**.

  Execute operating system commands by starting a line with "!!" followed by the command. Call alternate editors this way, without defining EDITOR.

- To clear the existing query buffer, type **reset** on a line by itself. **isql** discards any pending input. You can also press Ctrl-c anywhere on a line; this cancels the current query and returns the user to the **isql** prompt.

- Read in an operating system file containing a query for execution by **isql** as follows:

  ```
  isql -U alma -P****** < input_file
  ```

  The file must include command terminator(s). The results appear on your terminal. Read in an operating system file containing a query and direct the results to another file as follows:

  ```
  isql -U alma -P****** < input_file > output_file
  ```

- Case is significant for the **isql** flags.

- **isql** displays only six digits of *float* or *real* data after the decimal point, rounding off the remainder.

- When using **isql** interactively, read an operating system file into the command buffer with the command:

  ```
  :r filename
  ```

  Do not include a command terminator in the file; once you have finished editing, enter the terminator interactively on a line by itself.

- You can include comments in a Transact-SQL statement submitted to SQL Server by **isql**. For more information, see "Comments" in the *SQL Server Reference Manual*.

See Also

Chapter 3, “Using the isql Utility.”

langinstall

Function

Installs one new language on SQL Server.

Syntax

```
langinstall  [-S server] [-I interfaces_file]
   [-P password] [-R release_number] language
   character_set
langinstall [-v]
```

Parameters

-S *server* – specifies the name of the SQL Server to connect to. If you do not specify **-S**, **langinstall** uses the server specified by your DSQUERY environment variable. If DSQUERY is not set, **langinstall** attempts to connect to a server named SYBASE.

-I *interfaces_file* – specifies the name and location of the interfaces file that **langinstall** searches when connecting to SQL Server. If you do not specify **-I**, **langinstall** uses the interfaces file in the directory that your SYBASE environment variable specifies. If SYBASE is not set, **langinstall** looks for the default *SYBASE* home directory.

-P *password* – specifies the "sa" account password. If you omit **-P**, **langinstall** prompts for the "sa" account password.

-R *release_number* – specifies the release number, in the format *n.n.n*., to use to upgrade messages in *master..sysmessages*. Use **-R** only in failure conditions, such as if **langinstall** fails or in case of user error, when you think that messages in *sysmessages* are out of date.

language – is the official name of the language to install. You must specify a language.

character_set – is the name of SQL Server's default character set. *character_set* indicates the directory name of the localization files for the language. The *common.loc* and *server.loc* localization files for an official language reside in the character set directory *$SYBASE/locales/language/character_set*. You must specify a character set.

-v – prints the version number and copyright message for **langinstall**, and exits.

Comments

- The SQL Server installation program runs **langinstall** automatically for a new installation as well as for customers who are upgrading from a previous release.
- **langinstall** does the following:
  - Adds the specified language-specific information to *master..syslanguages* using the **sp_addlanguage** stored procedure. If the language already exists, **langinstall** updates the appropriate row in *syslanguages*.
  - Adds to, updates, and deletes error messages as necessary from *master..sysmessages*.
  - Updates *syslanguages.update*, inserting the new release number.
- **langinstall** validates the entries found in the localization file sections that it uses. If anything is missing, **langinstall** prints an error message and does not add the language to *syslanguages*.
- **langinstall** compares the version numbers of each of the localization files that it uses, *common.loc* and *server.loc*. If they are not the same, it prints a warning message. *syslanguages.upgrade* is always set according to the version number in *server.loc*.
- The **-R** option forces **langinstall** to collect messages from a release previous to the most current one. **langinstall** compares the existing messages with the ones to be installed and replaces any that have changed.

  For example, if the most current release is 11.0, but you think that *sysmessages* may not be correct, specify the messages added before the release in the *syslanguages.upgrade* column (11.0 in this case) with **-R 10.0**. **langinstall** then installs all messages from release 10.0 and earlier.

Permissions

Only the "sa" account can run **langinstall**.

Tables Used

master.dbo.syslanguages, master.dbo.sysmessages

See Also

System procedures	**sp_addlanguage, sp_addlogin, sp_configure, sp_defaultlanguage, sp_droplanguage, sp_helplanguage**

showserver

Function

Shows SQL Servers and Backup Servers currently running on the local machine.

Syntax

showserver

Examples

1. showserver

```
USER       PID %CPU %MEM   SZ  RSS TT STAT START  TIME COMMAND
user114276  0.0  1.7  712 1000 ?  S    Apr  5514:05 dataserver
-dstop.dat -sstop -einstall/stop_errorlog
sybase    1071  0.0  1.4  408  820 ?  S    Mar 28895:38
/usr/local/sybase4.9.1/bin/dataserver -d/dev/rsd1f
-e/install/errorlog
user128493  0.0  0.0 3692    0 ?  IW   Apr  1  0:10 backupserver
-SSYB_BACKUP -e/install/backup.log -Iinterfaces -Mbin/sybmultbuf
-Lus_english -Jiso_1
```

Comments

- **showserver** prints process information about SQL Server or Backup Server. If no servers are running, only the header appears.

startserver

Function

Starts a SQL Server and / or a Backup Server.

Syntax

```
startserver [[-f runserverfile] [-m]] ...
```

Parameters

-f *runserverfile* – specifies the relative path name of a runserver file used as a reference each time you restart a SQL Server or Backup Server. By default, the runserver file is in the current directory and is named *RUN_servername*. If you start a second SQL Server on the same machine, **startserver** creates a new runserver file named *RUN_servername*.

-m – starts SQL Server in single-user mode, allowing only one System Administrator to log in, and turns the **allow updates to system tables** configuration parameter on. Use this mode to restore the *master* database. The System Administrator can use the **dbo use only** option of **sp_dboption** for system administration activities that require more than one process, such as bulk copying or using the data dictionary. **startserver** normally starts up only one server per node.

The **-m** option creates an *m_RUNSERVER* file and overwrites any existing *m_RUNSERVER* file.

Examples

1. `startserver`

   Starts a SQL Server named SYBASE from a runserver file named *RUN_SYBASE* in the current directory.

2. `startserver -f RUN_MYSERVER -f RUN_SYB_BACKUP`

   Starts a SQL Server named MYSERVER and a Backup Server named SYB_BACKUP.

3. `startserver -f RUN_SYB_BACKUP`

   Starts only the Backup Server SYB_BACKUP.

Comments

- **startserver** uses the information in the runserver file to start a SQL Server or Backup Server. The master device must be writable by the user who starts SQL Server.

  The **startserver** command creates the SQL Server error log file (named *errorlog*) in the directory where you start the server, and adds this information as part of the **-e** option on the SQL Server executable line in the runserver file. If a second SQL Server is started on the same machine, a new error log named *errorlog_servername* is created; this information is added to that server's runserver file. The user must have execute permission on the specified runserver file.
- You can start multiple servers by specifying more than one runserver file, as shown in example 2. You can specify **-m** after each **-f** *runserverfile*.
- If the master database device has been damaged, use the **dataserver -r** option to start the mirror of the device. See **dataserver** for information.
- SQL Server derives its running environment from values in the *sysconfigures* system table. Run the system procedure **sp_configure** to see or change configuration.
- To ensure the integrity of your SQL Server, it is important that you apply appropriate operating system protections to the **startserver** executable file.

See Also

Utilities	**backupserver, dataserver**

2 Using *bcp* to Transfer Data to and from SQL Server

This chapter explains how to use **bcp** to move data between SQL Server and an operating system file. The three ways to move data are as follows:

- Using the bulk copy utility (**bcp**) as a standalone program from the operating system.
- Using Client-Library, which calls bulk library routines. See the *Open Client and Open Server Common Libraries Reference Manual* for details.
- Using DB-Library™ applications, which can call DB-Library routines. See the *Open Client DB-Library/C Reference Manual* for details.

This chapter discusses **bcp** in detail. See "bcp" on page 1-5 for the full command syntax and descriptions of the parameters.

Importing and Exporting Data

There are no Transact-SQL commands for the bulk transfer of data. Use **bcp**, the bulk copy utility, from the operating system command line.

bcp is most frequently used to import data that was previously associated with another program (such as another database management system). Use the dump facilities from the other program to put the data to be transferred into an operating system file.

You can also use **bcp** to move tables between SQL Servers or between SQL server and other data sources that can produce an operating system file.

SQL Server's bulk copy utility can transfer data for use with other programs as well—for example, with spreadsheet programs. **bcp** moves the data from SQL Server into an operating system file; from there the other program can import the data. When you finish using your data with the other program, transfer it back into an operating system file, and then use **bcp** to copy it back to SQL Server.

SQL Server can accept data in any character or binary format, as long as you can describe the **terminators** (the characters used to separate columns) or the length of the fields in the data file. The table structures need not be identical. When importing from a file, **bcp**

appends data to an existing database table; when exporting to a file, **bcp** overwrites any previous contents of the file.

Requirements for Using *bcp*

In general, you must supply the following information for transferring data to and from SQL Server:

- Name of the database and table
- Name of the operating system file
- Direction of the transfer (in or out)

In addition, you can optionally modify the storage type, storage length, and terminator for each column.

When the transfer is complete, **bcp** reports the number of rows successfully copied and some performance information.

Permissions Needed for Copying Data

To use **bcp**, you must have a SQL Server account and the appropriate permissions on the database tables and operating system files that you will use.

To copy data into a table, you must have **insert** permission on the table.

To copy a table out to an operating system file, you must have **select** permission on the following tables:

- The table being copied
- *sysobjects*
- *syscolumns*
- *sysindexes*

bcp Performance Issues

bcp in works in two modes: fast and slow. **bcp** is optimized to load data into tables that do not have indexes or triggers.

Depending on the size of the table into which you are copying data, the amount of data you are copying in, the number of indexes on the table, and the amount of spare database device space that you have

for re-creating indexes, you can make some choices between performance and recoverability.

You can remove any indexes and triggers on the target table and use **fast bcp**, where the individual insert operations are not logged and cannot be recovered from a log backup created with **dump transaction**. Or, you can retain any indexes and triggers on the table and use **slow bcp**, where every **insert** is logged, which can cause the transaction log to fill very quickly. If you are copying a large number of rows, the performance penalty and log space requirements for using slow **bcp** can be severe.

The performance of fast **bcp** can be improved even further with partitioned tables. By using several **bcp** sessions with a partitioned table, you can dramatically reduce the time required to copy the data.

The following sections discuss these issues in detail.

Bulk Copying Data with Indexes and Triggers

For copying data in, **bcp** is fastest if your database table has no indexes or triggers, because "fast" **bcp** does not log data inserts in the transaction log. "fast" **bcp** logs only the page allocations.

When you copy into a table that has indexes or triggers, **bcp** automatically uses a slower version, which logs data inserts in the transaction log. This can cause the transaction log to become very large, but dumping the log to a backup device with **dump transaction** assures that the database is fully recoverable in the event of a failure.

bcp does not fire the triggers, if any exist, on the target table.

➤ ***Note***

To allow any user to copy in data using the fast version of **bcp**, a System Administrator or the Database Owner must first use the **sp_dboption** system procedure to set the **select into/bulkcopy** option to **on** for the database containing the target table(s). If the option is not set to **on** and a user tries to copy data into a table that does not have indexes or triggers, SQL Server generates an error message.

You do not need to set **select into/bulkcopy on** to copy data out of or to copy data into a table that has indexes or triggers. Tables with indexes or triggers are always copied with slow **bcp**, and all inserts are logged.

If you have made unlogged data inserts with fast **bcp**, you cannot dump the transaction log to a device, because changes are not in the log and, therefore, are not recoverable from such a dump. In this situation, issuing **dump transaction** to a device produces an error message instructing you to use **dump database** instead. This restriction remains in force until a **dump database** successfully completes.

The following table shows which version of **bcp** is used when copying in, the necessary settings for the **select into/bulkcopy** option, and whether the transaction log can be dumped.

Table 2-1: Fast and slow bcp with select into/bulkcopy

	select into/bulkcopy on	**select into/bulkcopy off**
fast **bcp** (no indexes or triggers on target table)	OK **dump transaction** to a device prohibited	**bcp** prohibited
slow **bcp** (one or more indexes or triggers)	OK **dump transaction** OK	OK **dump transaction** OK

By default, the **select into/bulkcopy** option is **off** in newly created databases. To change the default setting, turn this option on in the *model* database.

If you are copying a very large number of rows, it may be faster to drop all the indexes and triggers beforehand with **drop index** and **drop trigger**, set the **select into/bulkcopy** database option on, copy the data into the table, re-create the indexes and triggers, and then dump the database. Remember to allocate 1.2 times the amount of space needed for the data, in addition to the data space, to reconstruct a clustered index. If you don't have enough space for the server to sort the data and build the index(es), use slow **bcp.**

Steps for Copying Data

Table 2-2 summarizes the steps for copying data into SQL Server.

Table 2-2: Steps for copying data

Step	Who Can Do It
Use **sp_dboption** to set **select into/bulkcopy** to **true**, and then run **checkpoint** in the database that was changed	System Administrator or Database Owner
Drop the indexes and triggers on the table (Make sure you have enough space to re-create them)	Table owner
Be sure that you have **insert** permission on the table	Granted by the table owner
Perform the copy with **bcp**	Any user with **insert** permission
Re-create the indexes and triggers	Table owner
Reset **sp_dboption**, if desired, and run **checkpoint** in the database that was changed	System Administrator or Database Owner
Use **dump database** to back up the newly inserted data	System Administrator, Operator, or Database Owner
Run stored procedures or queries to see if any of the newly loaded data violates rules	Table owner or stored procedure owner

Bulk Copying Data into Partitioned Tables

Under certain circumstances, you can improve **bcp** performance dramatically by executing several **bcp** sessions with a partitioned table. Partitioned tables improve insert performance by reducing lock contention (in online transaction processing systems) and by distributing I/O over multiple devices. **bcp** performance with partitioned tables is improved primarily because of distributed I/O.

However, not all partitioned tables will benefit equally from the use of multiple **bcp** sessions. Considering the following:

- Partitioned tables can only improve the performance of bulk copying **into** the table
- The table's partitions must be distributed over separate physical devices. The actual number of separate devices and partitions will vary depending on the performance you require.
- The performance of slow **bcp** will not improve dramatically with partitioned tables. Drop all indexes and triggers and use fast **bcp**, as described under "Steps for Copying Data" on page 2-5.

- If possible, use a local connection to the SQL Server. Network traffic can quickly become a bottleneck when executing multiple **bcp** sessions.

The following steps outline the general process of using multiple **bcp** sessions into a partitioned table:

1. Configure the table with as many partitions and physical devices as you require for your system. See "Improving Insert Performance with Partitions" on page 13-11 in the *Performance and Tuning Guide* for more details.
2. Make sure SQL Server is configured with enough locks to support the partitioned table. See "number of locks" on page 11-41 in the *System Administration Guide* for information on configuring locks.
3. Follow the instructions under "Steps for Copying Data" on page 2-5 to remove the triggers and indexes on the table and enable fast **bcp**. If you use slow **bcp**, performance may not improve significantly. Also, if the table contains indexes, you may experience deadlocks on the index pages; this is caused by the **bcp** sessions competing for the index.
4. Determine how many simultaneous **bcp** sessions you will use. Note that you cannot assign a particular **bcp** session to a partition—SQL Server randomly assigns each session to a partition. For this reason, make sure you use **fewer bcp** sessions than there are partitions in the table.

   A good rule of thumb is to start by using half as many **bcp** sessions as you have partitions. You can increase or decrease this number depending on the amount of contention for free partitions.
5. Divide the **bcp** input file into as many equal files as you will have simultaneous **bcp** sessions, or use the *firstrow* and *lastrow* options.
6. Execute the **bcp** sessions with separate files in parallel, preferably on the local SQL Server machine. For example, on UNIX platforms you can execute different sessions in different shell windows, or start individual **bcp** sessions in the background.

SQL Server randomly assigns the multiple **bcp** sessions to the table's available partitions. You may want to monitor the active partitions to determine whether or not the inserts were distributed evenly. Use the **dbcc checktable** command periodically to check the total page counts for each partition. If only a few partitions are active, you may want to cancel and restart the **bcp** sessions to better distribute I/O

activity. Optionally, you can use an operating system utility to monitor I/O activity directly. If you use such a utility, make sure that I/O is distributed evenly over the table's physical devices. See Chapter 17, "Checking Database Consistency" in the *System Administration Guide* for more information about **dbcc checktable**.

For more information about table partitions, see "Improving Insert Performance with Partitions" on page 13-11 in the *Performance and Tuning Guide*.

Reducing Logging by increasing Page Allocations

Each **bcp in** batch requires a trip to the page manager to allocate one or more extents, and each trip to the page manager generates a single log record. The **number of preallocatied extents** configuration parameter specifies how many extents SQL Server allocates in a single trip to the page manager. Increase this number when doing large **bcp** operations to prevent the page allocations from filling the log.

Because an object may allocate more pages than actually needed, keep the value small when space is limited. Valid values for the parameter are from 1 to 31. You must reboot SQL Server to change the value.

For more information, see "number of pre-allocated extents" on page 11-87 in the *System Administration Guide*.

Using the *bcp* Options

See "bcp" on page 1-6 for bcp syntax and a full discussion of the available options. The following sections describe some of the more complex options.

Using the Default Formats

bcp provides two command line options that create files with frequently used default formats. These options provide the easiest way to copy data in and out of SQL Server. The **-n** option uses "native" (or operating system) formats. The **-c** option uses "character" (*char* datatype) for all columns, providing tabs between fields on a row, and a newline terminator (such as a carriage return) at the end of each row.

If you are using the native or character options, **bcp** operates non-interactively and does not ask you for any information except your SQL Server password.

Native Format

The **-n** option creates files using **native** (operating system-specific) formats. Native format usually creates a more compact operating system file. For example, the following commands copy the *publishers* table to the file called *pub_out*, using native data format:

```
bcp pubs2..publishers out pub_out -n
```

Here are the contents of *pub_out*:

```
0736^MNew Age Books^FBoston^BMA0877^PBinnet & Hardley^J
Washington^BDC1389^TAlgodata Infosystems^HBerkeley^BCA
```

bcp prefixed each field (except the *pub_id*, which is a *char*(4) datatype) with an ASCII character equivalent to the length of the data in the field. For example, "New Age Books" is 13 characters, and ^M (Ctrl-M) is ASCII 13. All of the data in this table is *char* or *varchar* data, so it is human-readable. In a table with numeric data, **bcp** writes the information to the file in the operating system's data representation format and it may not be human-readable.

➤ ***Note***

Be careful when you copy native format data from different releases of SQL Server. Not all releases have the same datatypes.

Character Format

Character format uses "character" (the *char* datatype) for all columns. It inserts tabs between fields on each row and a newline terminator at the end of each row.

For example, the following commands copy the *publishers* file out in character format:

```
bcp pubs2..publishers out pub_out -c
```

This is the output:

```
0736    New Age Books           Boston       MA
0877    Binnet & Hardley        Washington   DC
1389    Algodata Infosystems    Berkeley     CA
```

Changing Terminators

Terminators are the characters used to separate data fields. The row terminator is the field terminator of the last field in the table or file.

Use the **-t** *field_terminator* and **-r** *row_terminator* command line options to change the terminators. The following example uses the comma as the field terminator and \r (Return) as the row terminator (remember to "escape" the backslash if necessary for your operating system command shell):

```
bcp pubs2..publishers out pub_out -c -t , -r \\r
```

This produces:

```
0736,New Age Books,Boston,MA
0877,Binnet & Hardley,Washington,DC
1389,Algodata Infosystems,Berkeley,CA
```

The **-t** and **-r** options can change the default terminators without the character option.

Changing the Defaults: Interactive *bcp*

If you do not use the native or character options, **bcp** prompts interactively for the storage type, prefix length, and terminator for each column of data to copy. For fields that are to be stored as *char* or *binary*, **bcp** also prompts for a field length.

The default values for these four prompts produce the same results as using the native option, and provide a simple means for copying data out of a database for later reloading into SQL Server. If you are copying data to or from SQL Server for use with other programs, base your answers to the prompts on the format that the other software requires.

Your responses to these four prompts provide an extremely flexible system that allows you to read files from other software or to create a file that requires little or no editing to conform to many other data formats. The following sections discuss these prompts and the way they interact to affect the data.

File Storage Type

The file storage type describes how to store the data in the file. You can copy data into a file either as its database table type, as a

character string, or as any datatype for which implicit conversion is supported. User-defined datatypes are copied as their base types.

Table 2-3 shows the default storage type for each SQL Server datatype, and the legal abbreviations. For the most compact storage, use the default value; for character files, use *char*. In Table 2-3, brackets [] indicate that you can use the initial character or the beginning characters of the word; for example, for "bit" you can use "b," "bi," or "bit." *timestamp* data is treated as *binary*(8). The *date* storage type is the SQL Server internal storage format of *datetime*, not the host operating system format of the date.

Table 2-3: File storage datatypes for bcp

Table Datatype	Storage Type
char, varchar	c[har]
text	T[ext]
int	i[nt]
smallint	s[mallint]
tinyint	t[inyint]
float	f[loat]
money	m[oney]
bit	b[it]
datetime	d[atetime]
binary, varbinary, timestamp	x
image	I[mage]
smalldatetime	D
real	r
smallmoney	M
numeric	n
decimal	e

To see this list while using **bcp** interactively, type a question mark in response to the "Enter the file storage type" prompt.

The suggested values that appear in the prompts are the defaults. Remember that your response determines how the data is stored in the output file; you need not indicate the column's type in the database table.

bcp fails if you enter a type that is not either implicitly convertible or *char*. For example, you may not be able to use *smallint* for *int* data (you may get overflow errors), but you can use *int* for *smallint*.

When storing noncharacter datatypes as their database types, **bcp** writes the data to the file in SQL Server's internal data representation format for the host operating system, rather than in human-readable form.

Prefix Length

By default, **bcp** precedes each field that has a variable storage length with a string of one or more bytes indicating the length of the field. This provides the most compact file storage. The default values in the prompts indicate the most efficient prefix length.

For fixed-length fields, the prefix length should be 0.

For fields of 255 bytes or less, the default prefix length is 1. For *text* or *image* datatypes, the default prefix length is 4. When *binary* and *varbinary* datatypes are being converted to *char* storage types, the default prefix length is 2, since each byte of table data requires 2 bytes of file storage.

SQL Server stores *binary*, *varbinary* and *image* data as an even number of hexadecimal digits. For these types, use even numbers for the prefix and length.

➤ ***Note***

bcp considers any data column that can contain null values to be variable-length. This includes columns with integer datatypes that might ordinarily be considered fixed-length. Use a length prefix (other than 0) or a terminator to denote the length of each row's data.

To store data with no prefix before its column, use a prefix length of 0. **bcp** pads each stored field with spaces to the full length specified at the next prompt, "length," unless you supply a terminator.

Because length prefixes consist of **native** format integers, the resulting host file contains nonprintable characters. You may not be able to print the host file or to transmit it using a communications program that cannot handle non-human-readable characters.

Storage Length

"Length" and "storage length" in this discussion always refer to the operating system file, not to SQL Server field lengths.

In almost all cases, accept the **bcp** default value for the storage length while copying data out. If you are making a file to reload into SQL Server, the default prefixes and length keep the storage space needed to a minimum. If you are creating a human-readable file, use the default length so that you do not truncate the data or create overflow errors that cause **bcp** to fail.

It is possible to change the default length by supplying another value. If you are copying character data in from other software, carefully examine the source file before choosing length values.

If the storage type is noncharacter, **bcp** stores the data in the operating system's native data representation and does not prompt for a length.

When **bcp** converts noncharacter data to character storage, it suggests a default field length large enough to store the data without truncating *datetime* data or causing overflow of numeric data. The default lengths are the number of bytes needed to display the longest value for the SQL Server datatype. Table 2-4 lists the default field lengths.

Table 2-4: Default field lengths for datatypes

Datatype	Default Size
int	12 bytes
smallint	6 bytes
tinyint	3 bytes
float	25 bytes
money	24 bytes
bit	1 byte
datetime	26 bytes
smalldatetime	26 bytes
real	25 bytes
smallmoney	24 bytes

If you specify a field length that is too short for numeric data when copying data out, **bcp** prints an overflow message and does not copy the data.

The default length for *binary* and *varbinary* fields is twice the length defined for the column, since each byte of the field requires 2 bytes of file storage.

If you accept the default storage length, the actual amount of storage space allocated depends on whether or not you specify a prefix length and terminators.

- If you specify a prefix length of 1, 2, or 4, **bcp** uses a storage space of the actual length of the data plus the length of the prefix plus any terminators.
- If you specify a prefix length of 0 and no terminator, **bcp** allocates the maximum amount of space shown in the prompt, which is the maximum space that may be needed for the datatype in question. In other words, **bcp** treats the field as if it were fixed length to

determine where one field ends and the next begins. For example, if the field is defined as *varchar*(30), **bcp** uses 30 bytes for each value, even if some of the values are only one character long. **bcp** does not know how large any one data value will be before copying all the data, so it always pads *char* datatypes to their full specified length.

Field and Row Terminators

A terminator can be used to mark the end of a column or row, separating one from the next. The default is no terminator. Field terminators separate table columns; the row terminator is the field terminator of the last field in the row of the table or file.

Terminators are very useful for dealing with character data because you can choose human-readable terminators. The **bcp** character option, which uses tabs between each column with a newline terminator at the end of each row, is an example of using terminators that enhance the readability of a data file.

When you prepare data for use with other programs, and when you want to use **bcp** to prepare tabular data, supply your own terminators. The available terminators are:

- Tabs, indicated by \t
- New lines, indicated by \n
- Carriage returns, indicated by \r
- Backslash, indicated by \
- Null terminators (no visible terminator), indicated by \0
- Any printable character (*, A, t, |, and so forth)
- Strings of up to 10 printable characters, including some or all of the terminators listed earlier (for example, **\t**, end, !!!!!!!!!!, or \t--\n).

➤ ***Note***

Control characters (ASCII 0–25) cannot be printed.

Choose terminators with patterns that do not appear in any of the data. For example, assume that using a tab terminator with a string of data that contains a tab creates an ambiguity: Which tab represents the end of the string? **bcp** always looks for the first possible

terminator, which in this case would be incorrect, since the first tab it would encounter would be the one that is part of the data string.

Data in native format can also conflict with terminators. Given a column that contains a 4-byte integer in native format, if the values of these integers are not strictly limited, it will be impossible to choose a terminator that is guaranteed not to appear inside the data. Use **bcp**'s native format option for data in native format.

Note that "no terminator" is different from a "null terminator," which is an invisible but real character.

Using Format Files

After gathering information about each field in the table, **bcp** asks if you want to save a format file and prompts for the file name. Use this format file to copy the data back into SQL Server, or to copy data out from the table at another time. When you copy data in or out using an existing format file, **bcp** does not prompt for information; the format file provides the information needed.

Figure 2-1 illustrates the format of the **bcp** format files. It shows the *publishers* table from the *pubs2* database, with all the host file columns in character format, no prefix, the default data length, a newline terminator at the end of the final column of a row, and tabs as terminators for all other columns.

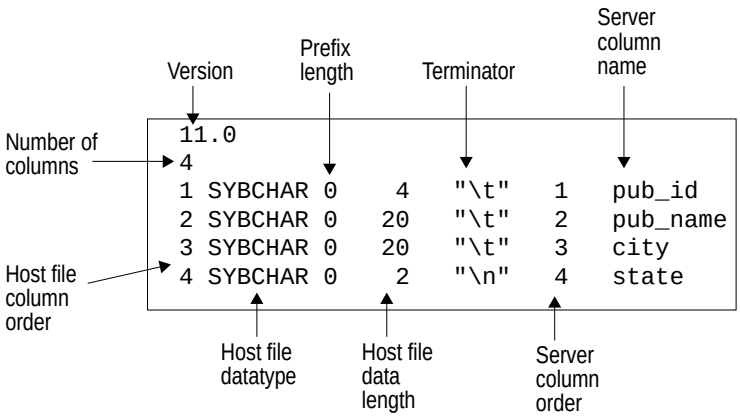

Figure 2-1: A bcp format file

Elements of the Format File

The **bcp** version is always the first line of the file. This is a literal string without quotation marks. In Figure 2-1, the version is 11.0.

The second line of a **bcp** format file is the **number of columns**, which refers to the number of records in the format file, not including lines 1 and 2. Each column in the host table has one line.

The first and second lines are followed by one line for each column in the database table, consisting of elements usually separated by tabs, except that the host file datatype and the prefix length are usually separated by a space. The following sections describe the elements in the format file.

Host File Column Order

The **host file column order** is the sequential number of the field in the host data file, starting with 1.

Host File Datatype

The **host file datatype** refers to the storage format of the field in the host data file, **not** the datatype of the database table column. Table 2-5 lists the valid storage formats.

Table 2-5: Host file datatype storage format

Storage Format	SQL Server Datatype
SYBCHAR	*char/varchar* (ASCII)
SYBTEXT	*text*
SYBBINARY	*binary*
SYBIMAGE	*image*
SYBINT1	*tinyint*
SYBINT2	*smallint*
SYBINT4	*int*
SYBFLT8	*float*
SYBREAL	*real*
SYBBIT	*bit*
SYBNUMERIC	*numeric*
SYBDECIMAL	*decimal*
SYBMONEY	*money*
SYBMONEY4	*smallmoney*
SYBDATETIME	*datetime*
SYBDATETIME4	*smalldatetime*

Prefix Length

Prefix length indicates the number of bytes in the field length prefix. The length prefix is a 0-, 1-, 2-, or 4-byte unsigned integer value embedded in the host data file that specfies the actual length of data contained in the field. Some fields may have a length prefix while others do not.

Table 2-6 shows the allowable prefix length values.

Table 2-6: Allowable prefix length values

No. of Bytes	Range
0	NO PREFIX
1	2^8-1; 0-255
2	2^{16}-1; 0-65535
4	2^{32} -1; 0-4,294,967,295

Host File Data Length

Host file data length refers to the maximum number of bytes to copy for the field. **bcp** uses either the maximum field length, the length prefix (if any), or the field terminator string (if any) to decide how much data to copy in or out. If more than one method of field length specification is given, **bcp** chooses the one that copies the least amount of data.

Terminator

The **terminator** can be up to 30 bytes of characters enclosed in quotation marks (" "). The terminator designates the end of data for the host data file field.

Server Column Order

The **host file column order** and the **server column order** together map host data file fields to the database table columns. The value in this field represents the *colid* of the table column into which to load the host data file column.

Server Column Name

The **server column name** is the name of the database table column into which this field is loaded.

Column Precision

The **precision** is the precision of the database table column into which this field is loaded. This element is present only if the storage format is *numeric* or *decimal*.

Column Scale

The **scale** is the scale of the database table column into which this field is loaded. This element is present only if the storage format is *numeric* or *decimal*.

Examples: Copying Out Data Interactively

By changing the default values of the prompts to **bcp**, you can prepare data for use with other software. To create a human-readable file, respond to the **bcp** prompts as follows:

- *char* storage type
- 0 prefix length
- Default field length
- Terminator depends on the software you plan to use. Choose between delimited fields or fixed length fields. Always use "\n", the newline terminator, to terminate the last field.

  For fixed-length fields, do not use a terminator. Each field has a fixed length, with spaces to pad the fields. Adjacent fields where the data completely fills the first field seem to run together, since there are no field separators on each line of output. See the example below.

  For comma-delimited output, use a comma as the terminator for each field. To create tabular output, use the tab character, "\t".

Copying Out Data with Field Lengths

The following example uses fixed-length fields to create output in the personal computer format called SDF (system data format). This format can be easily read or produced by other software.

```
bcp pubs2..sales out sal_out
Password:

Enter the file storage type of field stor_id [char]:
Enter prefix-length of field stor_id [0]:
Enter length of field stor_id [4]:
Enter field terminator [none]:

Enter the file storage type of field ord_num [char]:
Enter prefix-length of field ord_num [1]: 0
Enter length of field ord_num [20]:
Enter field terminator [none]:

Enter the file storage type of field date [datetime]: char
Enter prefix-length of field date [1]: 0
Enter length of field date [26]:
Enter field terminator [none]: \n

Do you want to save this format information in a file? [Y/n] y
Host filename [bcp.fmt]: sal_fmt

Starting copy...

30 rows copied.
Clock Time (ms.): total = 211 Avg = 7 (142.18 rows per sec.)
```

The contents of *sal_out* are as follows:

```
5023AB-123-DEF-425-1Z3  Oct 31 1985 12:00:00:000AM
5023AB-872-DEF-732-2Z1  Nov  6 1985 12:00:00:000AM
5023AX-532-FED-452-2Z7  Dec  1 1990 12:00:00:000AM
5023BS-345-DSE-860-1F2  Dec 12 1986 12:00:00:000AM
5023GH-542-NAD-713-9F9  Mar 15 1987 12:00:00:000AM
5023NF-123-ADS-642-9G3  Jul 18 1987 12:00:00:000AM
5023XS-135-DER-432-8J2  Mar 21 1991 12:00:00:000AM
5023ZA-000-ASD-324-4D1  Jul 27 1988 12:00:00:000AM
5023ZD-123-DFG-752-9G8  Mar 21 1991 12:00:00:000AM
5023ZS-645-CAT-415-1B2  Mar 21 1991 12:00:00:000AM
5023ZZ-999-ZZZ-999-0A0  Mar 21 1991 12:00:00:000AM
6380234518              Sep 30 1987 12:00:00:000AM
6380342157              Dec 13 1985 12:00:00:000AM
6380356921              Feb 17 1991 12:00:00:000AM
7066BA27618             Oct 12 1985 12:00:00:000AM
7066BA52498             Oct 27 1987 12:00:00:000AM
7066BA71224             Aug  5 1988 12:00:00:000AM
7067NB-1.142            Jan  2 1987 12:00:00:000AM
7067NB-3.142            Jun 13 1990 12:00:00:000AM
7131Asoap132            Nov 16 1986 12:00:00:000AM
```

```
7131Asoap432            Dec 20 1990 12:00:00:000AM
7131Fsoap867            Sep  8 1987 12:00:00:000AM
7896124152              Aug 14 1986 12:00:00:000AM
7896234518              Feb 14 1991 12:00:00:000AM
804212-F-9              Jul 13 1986 12:00:00:000AM
804213-E-7              May 23 1989 12:00:00:000AM
804213-J-9              Jan 13 1988 12:00:00:000AM
804255-V-7              Mar 20 1991 12:00:00:000AM
804291-A-7              Mar 20 1991 12:00:00:000AM
804291-V-7              Mar 20 1991 12:00:00:000AM
```

The contents of the format file *sal_fmt* are as follows:

```
11.0
3
1   SYBCHAR 04  "" 1  stor_id
2   SYBCHAR 020 "" 2  ord_num
3   SYBCHAR 026 "" 3  date
```

Copying Out Data with Delimiters

In the following examples, **bcp** interactively copies data from the *publishers* table to a file.

The first example creates an output file with commas between all fields in a row and a newline terminator at the end of each row. This example creates a format file (*pub_fmt*) which you can later use to copy the same or similar data back into SQL Server.

```
bcp pubs2..publishers out pub_out
Password:
Enter the file storage type of field pub_id [char]:
Enter prefix length of field pub_id [0]:
Enter length of field pub_id [4]:
Enter field terminator [none]:,
Enter the file storage type of field pub_name [char]:
Enter prefix length of field pub_name [1]: 0
Enter length of field pub_name [40]:
Enter field terminator [none]:,
Enter the file storage type of field city [char]:
Enter prefix length of field city [1]:0
Enter length of field city [20]:
Enter field terminator [none]:,
```

```
Enter the file storage type of field state [char]:
Enter prefix length of field state [1]:0
Enter length of field state [2]:
Enter field terminator [none]:\n

Do you want to save this format information in a file? [Y/n] y
Host filename [bcp.fmt]: pub_fmt

Starting copy...

3 rows copied.
Clock Time (ms.): total = 0   Avg = 0  (3.00 rows per sec.)
```

These are the results in *pub_out*:

```
0736,New Age Books,Boston,MA
0877,Binnet & Hardley,Washington,DC
1389,Algodata Infosystems,Berkeley,CA
```

The contents of *pub_fmt* are as follows:

```
11.0
4
1   SYBCHAR 0 4  ","  1  pub_id
2   SYBCHAR 0 40 ","  2  pub_name
3   SYBCHAR 0 20 ","  3  city
4   SYBCHAR 0 2  "\n" 4  state
```

Similarly, the following example creates tab-delimited output from the table *pubs2..publishers* in the file *pub_out*.

```
bcp pubs2..publishers out pub_out

Password:

Enter the file storage type of field pub_id [char]:
Enter prefix-length of field pub_id [0]: 0
Enter length of field pub_id [4]:
Enter field terminator [none]: \t

Enter the file storage type of field pub_name [char]:
Enter prefix-length of field pub_name [1]: 0
Enter length of field pub_name [40]:
Enter field terminator [none]: \t

Enter the file storage type of field city [char]:
Enter prefix-length of field city [1]: 0
Enter length of field city [20]:
Enter field terminator [none]: \t

Enter the file storage type of field state [char]:
Enter prefix-length of field state [1]: 0
```

```
Enter length of field state [2]:
Enter field terminator [none]: \n

Do you want to save this format information in a file? [Y/n] y
Host filename [bcp.fmt]: pub_fmt

Starting copy...

3 rows copied.
Clock Time (ms.): total = 120 Avg = 40 (25.00 rows per sec.)
```

The contents of *pub_out* are as follows:

```
0736  New Age Books        Boston     MA
0877  Binnet & Hardley     Washington DC
1389  Algodata Infosystems Berkeley   CA
```

The contents of the format file *pub_fmt* are as follows:

```
11.0
4
1   SYBCHAR 04   "\t"  1 pub_id
2   SYBCHAR 040  "\t"  2 pub_name
3   SYBCHAR 020  "\t"  3 city
4   SYBCHAR 02   "\n"  4 state
```

Examples: Copying In Data Interactively

To copy data successfully into a table from a file, you must know what the terminators in the file are, or what the field lengths are, and specify them when you use **bcp**. The following examples show how to copy data (either with fixed field lengths or with delimiters) in using **bcp**, with or without a format file.

Copying In Data with Field Lengths

In the following example, **bcp** copies data from the file *salesnew* into the table *pubs2..sales*. In the *salesnew* file are three fields: the first is 4 characters long, the second is 20, and the third is 26 characters long. Each row ends with a newline terminator (\n), as follows:

```
5023ZS-731-AAB-780-2B9  May 24 1993 12:00:00:000AM
5023XC-362-CFB-387-3Z5  May 24 1993 12:00:00:000AM
6380837206              May 24 1993 12:00:00:000AM
6380838441              May 24 1993 12:00:00:000AM
```

Use the following command to copy in the data from *salesnew* interactively:

```
bcp pubs2..sales in salesnew
```

The system responds as follows:

```
Password:

Enter the file storage type of field stor_id [char]:
Enter prefix-length of field stor_id [0]:
Enter length of field stor_id [4]:
Enter field terminator [none]:

Enter the file storage type of field ord_num [char]:
Enter prefix-length of field ord_num [1]: 0
Enter length of field ord_num [20]:
Enter field terminator [none]:

Enter the file storage type of field date [datetime]: char
Enter prefix-length of field date [1]: 0
Enter length of field date [26]:
Enter field terminator [none]: \n

Do you want to save this format information in a file? [Y/n] y
Host filename [bcp.fmt]: salesin_fmt

Starting copy...

4 rows copied.
Clock Time (ms.): total = 45    Avg = 11    (88.89 rows per sec.)
```

When you log into SQL Server and access *sales*, you will see the following data from *salesnew* appended to the table:

```
select * from sales

 stor_id ord_num              date
 ------- -------------------- --------------------------
 5023    AB-123-DEF-425-1Z3          Oct 31 1985 12:00AM
 5023    AB-872-DEF-732-2Z1          Nov  6 1985 12:00AM
 5023    AX-532-FED-452-2Z7          Dec  1 1990 12:00AM
 5023    BS-345-DSE-860-1F2          Dec 12 1986 12:00AM
 5023    GH-542-NAD-713-9F9          Mar 15 1987 12:00AM
 5023    NF-123-ADS-642-9G3          Jul 18 1987 12:00AM
 5023    XC-362-CFB-387-3Z5          May 24 1993 12:00AM
 5023    XS-135-DER-432-8J2          Mar 21 1991 12:00AM
 5023    ZA-000-ASD-324-4D1          Jul 27 1988 12:00AM
 5023    ZD-123-DFG-752-9G8          Mar 21 1991 12:00AM
 5023    ZS-645-CAT-415-1B2          Mar 21 1991 12:00AM
 5023    ZS-731-AAB-780-2B9          May 24 1993 12:00AM
 5023    ZZ-999-ZZZ-999-0A0          Mar 21 1991 12:00AM
 6380    234518                      Sep 30 1987 12:00AM
```

```
6380    342157                  Dec 13 1985 12:00AM
6380    356921                  Feb 17 1991 12:00AM
6380    837206                  May 24 1993 12:00AM
6380    838441                  May 24 1993 12:00AM
7066    BA27618                 Oct 12 1985 12:00AM
7066    BA52498                 Oct 27 1987 12:00AM
7066    BA71224                 Aug  5 1988 12:00AM
7067    NB-1.142                Jan  2 1987 12:00AM
7067    NB-3.142                Jun 13 1990 12:00AM
7131    Asoap132                Nov 16 1986 12:00AM
7131    Asoap432                Dec 20 1990 12:00AM
7131    Fsoap867                Sep  8 1987 12:00AM
7896    124152                  Aug 14 1986 12:00AM
7896    234518                  Feb 14 1991 12:00AM
8042    12-F-9                  Jul 13 1986 12:00AM
8042    13-E-7                  May 23 1989 12:00AM
8042    13-J-9                  Jan 13 1988 12:00AM
8042    55-V-7                  Mar 20 1991 12:00AM
8042    91-A-7                  Mar 20 1991 12:00AM
8042    91-V-7                  Mar 20 1991 12:00AM

(34 rows affected)
```

Since there is a unique clustered index on the *stor_id* and *ord_num* columns of *sales*, the new rows were sorted into order. Had there been any violations of the unique index on the columns in the data being copied from the file, **bcp** would have discarded the entire batch that contained a violating row. (A batch size of 1 evaluates each row individually, but loads more slowly and creates a separate data page for each row.) If the types copied in are incompatible with the database types, the entire copy fails.

Copying In Data with Delimiters

In the following example, **bcp** copies data from the file *newpubs* into the table *pubs2..publishers*. In the *newpubs* file, each field in a row ends with a tab character (\t) and each row ends with a newline terminator (\n), as follows:

```
1111    Stone Age Books         Boston      MA
2222    Harley & Davidson       Washington  DC
3333    Infodata Algosystems    Berkeley    CA
```

Since *newpubs* contains all character data, you can use the character command line flag and specify the terminators with command line options as follows:

```
bcp pubs2..publishers in newpubs -c -t \\t -r \\n
```

Copying In Data with a Format File

To copy data back into SQL Server using the saved format file *pub_fmt*, use the following command:

```
bcp pubs2..publishers in pub_out -f pub_fmt
```

You can use the *pub_fmt* file to copy any data with the same format into SQL Server. If you have a similar data file with different delimiters, you can change the delimiters in the format file.

Similarly, you can edit the format file to reflect any changes to the field lengths, so long as all fields have the same length. For example, the file *moresales* contains the following:

```
804213-L-9 Jan 21 1993 12:00AM
804255-N-8 Mar 12 1993 12:00AM
804291-T-4 Mar 23 1993 12:00AM
804291-W-9 Mar 23 1993 12:00AM
```

Edit the format file *sal_fmt* to read as follows:

```
11.0
3
1   SYBCHAR 0  4   ""   1   stor_id
2   SYBCHAR 0  7   ""   2   ord_num
3   SYBCHAR 0  21  "\n" 3   date
```

Then enter the following command:

```
bcp pubs2..sales in moresales -f sal_fmt
```

The system responds as follows:

```
Starting copy...

4 rows copied.
Clock Time (ms.): total = 28 Avg = 7 (142.86 rows per sec.)
```

Using *bcp* with Alternate Languages

SQL Server stores data using its default character set, which is configured during installation. If your terminal does not support that default character set, it may send confusing characters to **bcp** when you respond to prompts by typing or by using host file scripts.

Omitting all character set options sets the character set to a default for the platform. (This is not necessarily the same one that was configured for SQL Server.) Note also that this default may not necessarily be the character set that the client is using. (See "Converting Character Sets Between SQL Server and Clients" in the

System Administration Guide for more information about character sets and the associated flags.)

Batches and Copy In

Batching applies only to bulk copying in; it has no effect when copying out. By default, SQL Server copies all the rows specified in one batch. Use the command line option to specify a batch size.

SQL Server treats each batch as a separate **bcp** operation; **bcp** copies each batch in a single **insert** transaction to a separate data page, and if the server rejects any row in the batch, the entire **insert** rolls back. **bcp** then continues to the next batch. Only fatal errors roll back the **insert**.

bcp cannot save copies of rows that SQL Server rejects in an error file (for example, when SQL Server encounters a duplicate row for a table that has a unique index). SQL Server generates error messages on a batch-by-batch basis, instead of row-by-row, and rejects each batch in which it finds an error. Error messages appear on your terminal.

You can break large input files into smaller units for better recoverability. For example, if 300,000 rows are bulk copied in with a batch size of 100,000 rows, and there is a fatal error after row 200,000, the first two batches—200,000 rows—will have been successfully copied into SQL Server. If batching had not been used, no rows would have been copied into SQL Server.

The log entry for the transaction is available for truncation after the batch completes. If you copy into a database that has **trunc log on chkpt** set on, the next automatic checkpoint removes the log entries for completed batches. This breaks up large **bcp** operations and keeps the log from filling.

You can even set the *batchsize* to 1, which causes only the defective row to be rejected. This allows you to identify exactly which row failed. However, this loads slowly and takes up storage space. Because **bcp** creates one data page per batch, setting *batchsize* to 1 creates data pages with 1 row on each page.

Batches and Partitioned Tables

When you bulk copy data into a partitioned table, SQL Server randomly assigns each **insert** transaction (each batch) to an available partition. Copying rows in a single batch places all those rows in a single partition, which can quickly fill a partition's device(s) and lead

to **page stealing**. To help keep partitioned tables balanced, use a small batch size when bulk copying data. See "Partitioning and Unpartitioning Tables" on page 13-13 in the *Performance and Tuning Guide* for information about partitioning tables.

Specifying a Network Packet Size

You may want to use larger network packet sizes than the defaults to improve the performance of large bulk copy operations. The **-A** *size* option specifies the network packet size to use for this **bcp** session. *size* must be between the values of the **default network packet size** and **max network packet size** configuration parameters, one-third the size of the **additional network memory** configuration parameter, and it must be a multiple of 512. The new packet size is in effect for that **bcp** session only.

For example:

```
bcp pubs2..authors out -A 2048 -T 40960
```

specifies that SQL Server send 40K of *text* or *image* data using a packet size of 2,048 bytes for this **bcp** session.

Copying Out *text* and *image* Data

When you copy out *text* or *image* data, by default SQL Server copies the first 32K of data in a *text* or *image* field. The **-T** flag allows you to specify a different value. For example, if the *text* field to copy out contains 40K of data, copy out all 40K with the following command:

```
bcp pubs2..publishers out -T 40960
```

If a *text* or *image* field is larger than the given value or the default, the overflow is not sent.

Copy In and Error Files

When you specify the **-e** flag when copying data in, **bcp** stores certain rows that it cannot copy into SQL Server in the specified error file.

The error file stores a line indicating which row failed and what error occurred, and a line that is an exact copy of the row from the host file. **bcp** overwrites any file of the same name. If **bcp** does not encounter any errors, it does not create the file.

bcp in detects two types of errors:

- Data conversion errors
- Errors in building the row, such as attempts to insert a NULL into columns that don't accept them, or illegal data formats (such as a 3-byte integer)

Error messages appear on your terminal. The following example loads the *newpubs* file into the *publishers* database, storing any error rows in the file *pub_err*:

```
bcp pubs2..publishers in newpubs -e pub_err
```

bcp stores rows in an error file only when the **bcp** program itself detects the error, and continues to copy rows until **bcp** encounters the maximum number of error rows, at which point **bcp** stops the copy.

Because **bcp** sends rows to SQL Server in batches, it cannot save copies of rows that SQL Server rejects (for example, when SQL Server encounters a duplicate row for a table that has a unique index). SQL Server generates error messages on a batch-by-batch basis, instead of row-by-row, and rejects the entire batch if it finds an error.

It is not considered an error for SQL Server to reject duplicate rows if **allow_dup_row** or **ignore_dup_key** was set when a table's index was created. The copy proceeds normally, and the duplicate rows are not stored in the table, nor in the **bcp** error file.

Copy Out and Error Files

When you use the **-e** flag when copying data out, **bcp** stores the rows that it cannot copy out in the specified error file. As with copy in, **bcp** overwrites any file of the same name, and does not create the file if no errors occurred.

There are two situations that cause rows to be logged in the error file during a copy out:

- A data conversion error in one of the row's columns
- An I/O error in writing to the host file

bcp logs rows in the error file in the default character format. All data values print as characters, with tabs between the columns, and a newline terminator at the end of each row.

Data Integrity: Defaults, Rules, and Triggers

When copying data into a table, **bcp** observes any defaults defined for the columns and datatypes. That is, if there is a null field in the data in a file, **bcp** loads the default value instead during the copy. For example, here are two rows in a file to load into *authors*:

```
409-56-7008,Bennet,Abraham,415 658-9932,6223 Bateman
St.,Berkeley,CA,USA,94705
213-46-8915,Green,Marjorie,,309 63rd St. #411,Oakland,CA,USA,94618
```

Commas separate the fields; a newline terminator separates the rows. Note that there is no phone number for Marjorie Green. Because the *phone* column of the *authors* table has a default of "unknown," the rows in the loaded table look like this:

```
409-56-7008  Bennet  Abraham   415 658-9932  6223 Bateman St.
        Berkeley  CA  USA  94705
213-46-8915  Green   Marjorie  unknown       309 63rd St. #411
        Oakland   CA  USA  94618
```

In order to load data at the maximum speed, **bcp** does not fire rules and triggers. To find any rows that violate rules and triggers, copy the data into the table and run queries or stored procedures that test the rule or trigger conditions.

How Bulk Copy Differs from Other SQL Server Facilities

The bulk copy facilities, which copy entire tables or portions of a single table, are distinct from several other commands and options that also move data from one place to another. These commands are:

- The SQL commands **dump database**, **load database**, **dump transaction**, and **load transaction**. These are used for backup purposes only. Unlike the bulk copy facilities, the dump facilities create a physical image of the entire database. Data dumped with **dump database** or **dump transaction** can be read only by using **load database** or **load transaction**. (See Chapter 18, "Developing a Backup and Recovery Plan" in the *System Administration Guide* or Chapter 3, "Transact-SQL Commands," in the *SQL Server Reference Manual* for details.)
- The data modification commands **insert**, **update**, and **delete**. Use these to add new rows to, change existing rows in, or remove rows from a table or view. The **insert** command can also be used with a **select** statement in order to move data from one table to another. The **select** statement with an **into** clause can create a new table, based on the columns in the **select** statement and the tables

in the **from** clause, and copy the rows specified in the **where** clause. (See Chapter 3, "Transact-SQL Commands," in the *SQL Server Reference Manual* or Chapter 8, "Adding, Changing, and Deleting Data" in the *Transact-SQL User's Guide,* for details.)

3 Using the *isql* Utility

This chapter introduces the interactive SQL utility **isql** and discusses some **isql** topics: changing the command terminator, the interaction of the performance option and command terminator values, setting the network packet size, and input and output files.

How to Use Transact-SQL with the *isql* Utility

You can use SQL directly from the operating system with the standalone utility program **isql**. You must have an account, or login, on SQL Server. To access the account, enter this command at your operating system prompt:

```
isql
```

The following prompt appears:

```
Password:
```

Type your password at the prompt and press the Return key. The password does not appear on the screen as you type. The **isql** prompt appears, as follows:

```
1>
```

You can now start issuing Transact-SQL commands. The **isql** program sends the commands to SQL Server, formatting the results and printing them to standard output. There is no maximum size for an **isql** statement.

Terminate a command by typing the default command terminator **go** on a new line. For example:

```
isql
Password:

1> select *
2> from authors
3> where city = "Oakland"
4> go
```

To exit **isql**, type “quit” or “exit” on a line by itself.

Formatting *isql* Output

Table 3-1 describes the command line options that change the format of **isql** output:

Table 3-1: Format options for isql

Option	Default	Meaning
-h *headers*	1	Number of rows to print between column headings
-s *colseparator*	Single space	Changes the column separator character
-w *linewidth*	80 characters	Changes the line width

To include each command issued to **isql** in the output, use the **-e** option; use the **-n** option to remove numbering and prompt symbols. For example:

```
isql -e -n -o output
Password:
select *
from authors
where city = "Oakland"
go
quit

cat output
select *
from authors
where city = "Oakland"
 au_id       au_lname                              au_fname
      phone        address
      city               state  country      postalcode
 ----------- ------------------ --------------------------
     ------------ ---------------------------------------
     -------------------- ----- ------------ ----------
 213-46-8915 Green                                  Marjorie
      415 986-7020 309 63rd St. #411
      Oakland              CA    USA          94618
 274-80-9391 Straight                               Dick
      415 834-2919 5420 College Av.
      Oakland              CA    USA          94609
```

```
 724-08-9931 Stringer                                Dirk
      415 843-2991 5420 Telegraph Av.
      Oakland              CA    USA          94609
 724-80-9391 MacFeather                              Stearns
      415 354-7128 44 Upland Hts.
      Oakland              CA    USA          94612
 756-30-7391 Karsen                                  Livia
      415 534-9219 5720 McAuley St.
      Oakland              CA    USA          94609
(5 rows affected)
```

Note that the output file does not include the command terminator.

Correcting Input

If you make an error when typing a Transact-SQL command, you can:

- Press Ctrl-c or type the word “reset” on a line by itself.

  This clears the query buffer and returns the **isql** prompt.

- Type the name of your text editor on a line by itself.

  This puts you in a text file where you can edit the query. When you write and save the file, you are returned to **isql**. The query appears; type “go” to execute it.

set Options That Affect Output

Table 3-2 lists the **set** options that affect Transact-SQL output. For more information, refer to the **set** command in the *SQL Server Reference Manual*.

Table 3-2: set options that affect Transact-SQL output

set Option	Default	Meaning
char_convert	Off	Turns character set conversion off and on between SQL Server and a client; it also starts a conversion between the server character set and a different client character set
fipsflagger	Off	Warns when any Transact-SQL extensions to entry level SQL92 are used
flushmessage	Off	Sends messages as they are generated
language	us_english	Sets the language for system messages

Table 3-2: set options that affect Transact-SQL output (continued)

set Option	Default	Meaning
nocount	Off	Turns off report of number of rows affected
noexec	Off	Compiles each query but does not execute it; often used with **showplan**
parseonly	Off	Checks the syntax of queries and returns error messages without compiling or executing the queries
showplan	Off	Generates a description of the processing plan for a query; does not print results when used inside a stored procedure or trigger
statistics io **statistics time**	Off	Displays performance statistics after each execution
statistics subquerycache	Off	Displays the number of cache hits, misses, and rows in the subquery cache for each subquery
textsize	32KB	Controls the number of bytes of *text* or *image* data returned

Changing the Command Terminator

If you include the command terminator argument (**-c**), you can choose your own terminator symbol; "go" is the default value for this option. You must always enter the command terminator without blanks or tabs in front of it.

For example, to use a period as the command terminator, invoke **isql** as follows:

```
isql -c.
```

A sample **isql** session with this command terminator looks like this:

```
1> select name from sysusers
2> .
name
-----------
sandy
kim
leslie
(3 rows affected)
```

Using the **isql** command terminator option with scripts requires advance planning:

- All SQL Server-supplied scripts, such as **installmaster**, use "go". Do not change the command terminator for any session that uses these scripts.
- Your own scripts may already have "go" in them. Remember to update any of your own scripts to include the terminator you plan to use.

Performance Statistics Interaction with Command Terminator Values

isql provides a performance statistics option (**-p**). For example:

```
isql -p

1> select * from sysobjects
2> go
```

returns the following statistics:

```
1 xact:
Clock Time (ms.): total = 2000 avg = 2000 (0.50 xacts per sec.)
```

This means that a single transaction took 2,000 milliseconds, so the average is one transaction per 2,000 milliseconds. The clock time value reflects the entire transaction, which starts when Client-Library builds the query and ends when Client-Library returns the information from SQL Server.

You can gather performance statistics based on executing one or more transactions. To gather statistics on more than one transaction, specify a number after the command terminator (**go**, by default). For example:

```
isql -p

1> select * from sysobjects
2> go 3
```

instructs SQL Server to execute three **select *** transactions and report the performance statistics. SQL Server returns:

```
3 xacts:
Clock Time (ms.): total = 1000 avg = 333 (3.00 xacts per sec.)
```

Setting the Network Packet Size

The **-A** *size* option specifies the network packet size to use for this **isql** session For example:

```
isql -A 2048
```

sets the packet size to 2,048 bytes for this **isql** session. To check, type:

```
select * from sysprocesses
```

The value appears under the *network_pktsz* heading.

size must be between the values of the **default network packet size** and **max network packet size** configuration parameters, one-third the size of the **additional network memory** configuration parameter, and must be a multiple of 512. SQL Server uses the closest available packet size that is a multiple of 512 if there is not enough memory available.

Use packet sizes larger than the defaults to perform I/O-intensive operations, such as **readtext** or **writetext** operations.

Setting or changing SQL Server's packet size does not affect the remote procedure call's packet size.

Input and Output Files

You can specify input and output files on the command line with the **-i** and **-o** options.

isql does not provide formatting options for the output. However, you can use the **-n** option to eliminate the **isql** prompts, and use other tools to reformat the output.

If you use the **-e** option, **isql** echoes the input to output. The resulting output file contains both the queries and their results.

UNIX Command Line Redirection

The UNIX redirection symbols, "<" and ">", provide a similar mechanism to the **-i** and **-o** options, as follows:

```
isql -Usa -Ppassword < input > output
```

You can direct **isql** to take input from the terminal, as in the following example:

```
isql -Usa -Ppassword << EOF > output
select * from table
go
EOF
```

"<<EOF" instructs **isql** to take input from the terminal until the string "EOF." You can replace "EOF" with any character string.

Similarly, the following example signals the end of input with Ctrl-d:

```
isql -Usa -Ppassword << > output
```

Index

The index is divided into two sections:

- Symbols

  Indexes each of the symbols used in Sybase SQL Server documentation.

- Subjects

  Indexes subjects alphabetically.

Page numbers in **bold** are primary references.

Symbols

A

B

C

D

E

F

L

M

N

O

P

Q

R

S

T

U

V

W