Standard Full-Text Search Specialty Data Store User's Guide

Adaptive Server Enterprise Version 11.5.x
Document ID: 36521-01-1150-02
Last Revised: January 7, 1998

Principal author: Martin Ash

Document ID: 36521-01-1150

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

Sybase, the Sybase logo, APT-FORMS, Certified SYBASE Professional, Column Design, Data Workbench, First Impression, InfoMaker, ObjectCycle, PowerBuilder, PowerDesigner, Powersoft, Replication Server, S-Designor, SQL Advantage, SQL Debug, SQL SMART, SQL Solutions, Transact-SQL, Visual Components, VisualWriter, and VQL are registered trademarks of Sybase, Inc. Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Monitor, Adaptive Warehouse, ADA Workbench, AnswerBase, Application Manager, AppModeler, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, APT Workbench, Backup Server, BayCam, Bit-Wise, ClearConnect, Client-Library, Client Services, CodeBank, Connection Manager, DataArchitect, Database Analyzer, DataExpress, Data Pipeline, DataWindow, DB-Library, dbQueue, Developers Workbench, DirectConnect, Distribution Agent, Distribution Director, Dynamo, Embedded SQL, EMS, Enterprise Client / Server, Enterprise Connect, Enterprise Manager, Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, EWA, Formula One, Gateway Manager, GeoPoint, ImpactNow, InformationConnect, InstaHelp, InternetBuilder, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Logical Memory Manager, MainframeConnect, Maintenance Express, MAP, MDI Access Server, MDI Database Gateway, media.splash, MetaBridge, MetaWorks, MethodSet, Net-Gateway, NetImpact, Net-Library, ObjectConnect, OmniConnect, OmniSQL Access Module, OmniSQL

Restricted Rights

Table of Contents

About This Book

1. Introduction

2. Installing the Full-Text Search Engine on UNIX Platforms

3. Installing the Full-Text Search Engine on Windows NT

4. Configuring the Full-Text Search Engine and Adaptive Server

5. Administration and Tuning

6. Full-Text Search Engine Commands

A. System Procedures

B. Sample Files

Index

List of Tables

List of Figures

About This Book

This book explains how to install and configure the Standard Full-Text Search Specialty Data Store product and how to use it with Sybase® Adaptive Server™ Enterprise.

The Standard Full-Text Search Specialty Data Store uses the Verity Search '97 to perform full-text searches on Adaptive Server. For more information about the Verity product and the Verity operators used to perform full-text searches, see the World Wide Web at:

http://www.verity.com

Audience

This book is intended for System Administrators who are installing and configuring the Standard Full-Text Search Specialty Data Store release 11.5.x, and for users who are performing full-text searches on Adaptive Server® Enterprise data.

How to Use This Book

This book will assist you in installing, configuring, and using the Standard Full-Text Search Specialty Data Store. It includes the following chapters:

- Chapter 1, "Introduction," provides an overview of Standard Full-Text Search Specialty Data Store.
- Chapter 2, "Installing the Full-Text Search Engine on UNIX Platforms," describes the process of unloading and configuring the product for UNIX platforms.
- Chapter 3, "Installing the Full-Text Search Engine on Windows NT," describes the process of unloading and configuring the product for Windows NT.
- Chapter 4, "Configuring the Full-Text Search Engine and Adaptive Server," describes the process of configuring Adaptive Server so that it can perform full-text searches on its databases.
- Chapter 5, "Administration and Tuning," provides information about system administration and performance and tuning issues.

- Chapter 6, “Full-Text Search Engine Commands,” describes the Verity operators (commands) you use to perform full-text searches.
- Appendix A, “System Procedures,” describes the system procedures used for configuring Standard Full-Text Search Specialty Data Store.
- Appendix B, “Sample Files,” contains the text of the *textsvr.cfg*, *style.vgw*, and *style.ufl* files and discusses issues regarding the *text_sample* script.

Adaptive Server Enterprise Documents

The following documents comprise the Sybase Adaptive Server Enterprise documentation:

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

  A more recent version of the *Release Bulletin* may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use SyBooks™-on-the-Web.
- The Adaptive Server installation documentation for your platform – describes installation and upgrade procedures for all Adaptive Server and related Sybase products.
- The Adaptive Server configuration documentation for your platform – describes configuring a server, creating network connections, configuring for optional functionality, such as auditing, installing most optional system databases, and performing operating system administration tasks.
- *What’s New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server release 11.5, the system changes added to support those features, and the changes that may affect your existing applications.
- *Navigating the Documentation for Adaptive Server* – an electronic interface for using Adaptive Server. This online document provides links to the concepts and syntax in the documentation that are relevant to each task.
- *Transact-SQL User’s Guide* – documents Transact-SQL®, Sybase’s enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database

management system. This manual also contains descriptions of the *pubs2* and *pubs3* sample databases.

- *System Administration Guide* – provides in-depth information about administering servers and databases. This manual includes instructions and guidelines for managing physical resources and user and system databases, and specifying character conversion, international language, and sort order settings.
- *Adaptive Server Reference Manual* – contains detailed information about all Transact-SQL commands, functions, procedures, and datatypes. This manual also contains a list of the Transact-SQL reserved words and definitions of system tables.
- *Performance and Tuning Guide* – explains how to tune Adaptive Server for maximum performance. This manual includes information about database design issues that affect performance, query optimization, how to tune Adaptive Server for very large databases, disk and cache issues, and the effects of locking and cursors on performance.
- The *Utility Programs* manual for your platform – documents the Adaptive Server utility programs, such as **isql** and **bcp**, which are executed at the operating system level.
- *Security Administration Guide* – explains how to use the security features provided by Adaptive Server to control user access to data. This manual includes information about how to add users to Adaptive Server, administer both system and user-defined roles, grant database access to users, and manage remote Adaptive Servers.
- *Security Features User's Guide* – provides instructions and guidelines for using the security options provided in Adaptive Server from the perspective of the non-administrative user.
- *Error Messages* and *Troubleshooting Guide* – explains how to resolve frequently occurring error messages and describes solutions to system problems frequently encountered by users.
- *Component Integration Services User's Guide for Adaptive Server Enterprise and OmniConnect* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- *Adaptive Server Glossary* – defines technical terms used in the Adaptive Server documentation.
- *Master Index for Adaptive Server Publications* – combines the indexes of the *Adaptive Server Reference Manual*, *Component*

Integration Services User's Guide, *Performance and Tuning Guide*, *Security Administration Guide*, *Security Features User's Guide*, *System Administration Guide*, and *Transact-SQL User's Guide*.

Conventions

Formatting SQL Statements

SQL is a free-form language: there are no rules about the number of words you can put on a line or where you must break a line. However, for readability, all examples and syntax statements in this manual are formatted so that each clause of a statement begins on a new line. Clauses that have more than one part extend to additional lines, which are indented.

SQL Syntax Conventions

The conventions for syntax statements in this manual are as follows:

Table 1: Syntax statement conventions

Key	Definition
command	Command names, command option names, utility names, utility flags, and other keywords are in **`bold Courier`** in syntax statements and in **bold Helvetica** in paragraph text.
variable	Variables, or words that stand for values that you fill in, are in *italics*.
`{ }`	Curly braces indicate that you choose at least one of the enclosed options. Do not include braces in your option.
`[ ]`	Brackets mean choosing one or more of the enclosed options is optional. Do not include brackets in your option.
`( )`	Parentheses are to be typed as part of the command.
`|`	The vertical bar means you may select only one of the options shown.
`,`	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.

- Syntax statements (displaying the syntax and all options for a command) are printed like this:

```
sp_dropdevice [device_name]
```

or, for a command with more options:

```
select column_name
      from table_name
      where search_conditions
```

In syntax statements, keywords (commands) are in normal font and identifiers are in lowercase: normal font for keywords, italics for user-supplied words.

- Examples showing the use of Transact-SQL commands are printed like this:

  ```
  select * from publishers
  ```

- Examples of output from the computer are printed like this:

  ```
  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)
  ```

Case

In this manual, most of the examples are in lowercase. However, you can disregard case when typing Transact-SQL keywords. For example, **SELECT**, **Select**, and **select** are the same.

Adaptive Server's sensitivity to the case of database objects, such as table names, depends on the sort order installed on Adaptive Server. You can change case sensitivity for single-byte character sets by reconfiguring the Adaptive Server sort order. See "Changing the Default Character Set, Sort Order, or Language" in Chapter 19 of the *System Administration Guide* for more information.

Obligatory Options {You Must Choose At Least One}

- **Curly Braces and Vertical Bars**: Choose **one and only one** option.

  ```
  {die_on_your_feet | live_on_your_knees |
  live_on_your_feet}
  ```

- **Curly Braces and Commas**: Choose one or more options. If you choose more than one, separate your choices with commas.

  ```
  {cash, check, credit}
  ```

Optional Options [You Don't Have to Choose Any]

- **One Item in Square Brackets:** You don't have to choose it.

  ```
  [anchovies]
  ```

- **Square Brackets and Vertical Bars:** Choose **none or only one**.

  ```
  [beans | rice | sweet_potatoes]
  ```

- **Square Brackets and Commas**: Choose **none, one, or more than one** option. If you choose more than one, separate your choices with commas.

  ```
  [extra_cheese, avocados, sour_cream]
  ```

Ellipsis: Do It Again (and Again)...

An ellipsis (...) means that you can **repeat** the last unit as many times as you like. In this syntax statement, **buy** is a required keyword:

```
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.

If You Need Help

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

1 Introduction

What Is the Standard Full-Text Search Specialty Data Store?

Using the Standard Full-Text Search Specialty Data Store product (referred to in this book as the Full-Text Search engine), you can perform powerful, full-text searches on Adaptive Server data. By default, Sybase Adaptive Server without the Full-Text Search engine allows you to search text columns only for data that matches what you specify in a **select** statement. For example, if a table contains documents about dog breeds, and you perform a search on the words "Saint Bernard," the query produces only the rows that include "Saint Bernard" in the text column. However, with the Full-Text Search engine, you can expand queries on text columns to do the following:

- Rank the results by order of how often a searched item appears in the selected document. For example, you can obtain a list of document titles that reference the words "Saint Bernard" five or more times.
- Select documents in which the words you search for appear within *N* number of words of each other. For example, you can search only for the documents that include the words "Saint Bernard" and "Swiss Alps" and that appear within 10 words of each other.
- Select documents that include all the search elements you specify within a single paragraph or sentence. For example, you can query the documents that include the words "Saint Bernard" in the same paragraph or sentence as the words "Swiss Alps."
- Select documents that contain one or more synonyms of the word you specify. For example, you can select documents that discuss "working dogs", "large dogs", "European Breeds", and so on.
- As you use the Full-Text Search engine, you will find that it offers many other search capabilities.

Product Components

The Standard Full-Text Search Specialty Data Store product consists of four key components:

- Full-Text Search engine

- Text database
- Source and index tables
- Verity collections

The relationships between these components are shown in Figure 1-1.

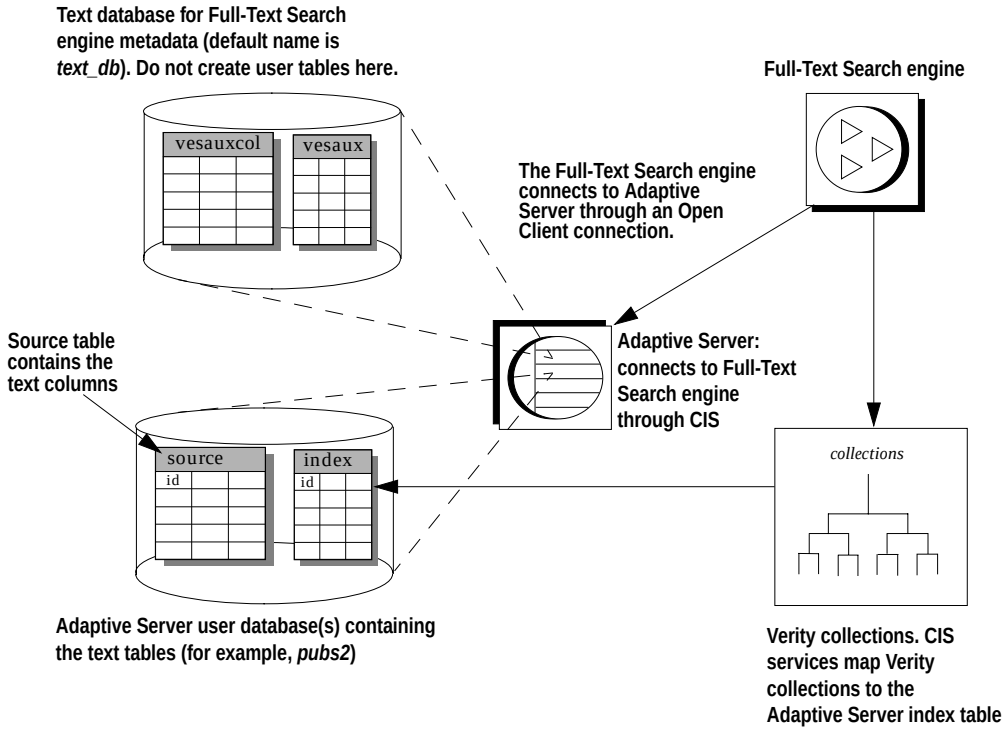


Figure 1-1: Components of the Full-Text Search engine

The Full-Text Search Engine

The Full-Text Search engine is an Open Server™ application built on the Verity Search '97 product. Adaptive Server connects to the Full-Text Search engine, allowing queries written in the Verity query language to perform full-text searches on Adaptive Server data, as described in Chapter 6, "Full-Text Search Engine Commands."

The special command operators that you use to perform full-text searches are part of the Verity Search '97 search engine. You enter the

search operators as part of a **select** statement; then, the Full-Text Search engine processes the full-text searches.

Filters Included with Full-Text Search Engine

The text documents in a database can be stored in a variety of document types (MicroSoft Word, SGML, HTML, FrameMaker, and so on). Verity includes a series of filters that allow you to index these document types.

You do not have to configure Adaptive Server or Full-Text Search engine to use these filters; they automatically detect the document type and apply the correct filter.

The Text Database

During the installation of the Full-Text Search engine, a database named *text_db* is added to Adaptive Server using the installation scripts **installtextserver** and **installevent.** The database contains two support tables: *vesaux* and *vesauxcol*. These tables contain the metadata used by the Full-Text Search engine.

- The *vesaux* table contains:
  - The name of the table that contains the text being searched
  - The location of the table that contains the text you are searching for
  - The name of the Verity collection
- The *vesauxcol* table contains the names of the columns in the text index.

After an insert, update, or delete is made to an indexed column, the Full-Text Search engine queries the *vesaux* and *vesauxcol* tables. These tables determine which collections contain the modified columns so that all affected collections can be updated. The Full-Text Search engine also uses these tables when it is brought online to make sure that all necessary collections exist.

The Source and Index Tables

Two tables enable Adaptive Server to work with the Full-Text Search engine and perform searches on text data: the **source table** and the **index table**. These tables provide a means of locating and searching documents stored in text columns.

The Source Table

The **source table** is a standard table maintained by Adaptive Server. It contains a column using the *text*, *image*, *char*, *varchar*, *datetime*, or *small datetime* datatype, which holds the data to be searched. The source table must also have an IDENTITY column, which is used to join with the IDENTITY column of an index table during text searches.

The Index Table

The **index table** is maintained by the Full-Text Search engine and has an IDENTITY column that maps to the IDENTITY column of the corresponding source table. The IDENTITY value from the row in the source table is stored with the data in the Verity collections, which allows the source and index tables to be joined. Although the index table is stored and maintained by the Full-Text Search engine, it functions as a local table to Adaptive Server through the Adaptive Server Component Integration Services (CIS) feature.

Index Table Pseudo Columns

The index table contains special columns, called **pseudo columns**, that are used by the Full-Text Search engine to determine the parameters of the search and the location of the text data in the source table. Pseudo columns have no associated physical storage—the values of a pseudo column are valid only for the duration of the query and are removed immediately after the query finishes running.

For example, when using the *score* pseudo column (which ranks each document according to how well the document matches a query) in a query, you may have to use a *score* of 15 to find references to the phrase “small Saint Bernards” in the text database. This phrase probably won’t occur very often, and a low *score* value broadens the search to include documents that have a small number of occurrences of the search criteria. However, if you are searching for a phrase that is common, like “large Saint Bernards,” you could use a *score* of 90, which would limit the search to those documents that have many occurrences of the search criteria.

The score column and other pseudo columns *id*, *index_any*, *score*, *sort_by*, *summary*, and *max_docs* determine the specific parameters you can include in your search. For more information about pseudo columns, see “Pseudo Columns in the Index Table” on page 4-6.

The Verity Collections

The Full-Text Search engine includes the Verity collections, which are located in *$SYBASE/sds/text/collections*. When you create the text indexes, as described in "The Text Database" on page 1-3, Verity creates a **collection**, which is a directory that implements a text index. This collection is queried by the Full-Text Search engine. For more information about Verity collections, see the Web site at:

http://www.verity.com

How a Full-Text Search Works

To perform a full-text search, you enter a **select** statement that joins the IDENTITY column from the source table with the IDENTITY column of the index table, using pseudo columns as needed to define the search. For example, the following query searches for documents in the *blurbs* table of the *pubs2* database in which the word "Greek" appears near the word "Gustibus":

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and t1.score > 20
and t1.index_any = "<near>(Greek, Gustibus)"
```

Adaptive Server and the Full-Text Search engine split the query processing, as follows:

- The Full-Text Search engine processes the query:

  ```
  select t1.score, t1.id
  from p_blurbs t1, blurbs t2
  where t1.score > 20
  and t1.index_any = "<near>(Greek, Gustibus)"
  ```

  which includes the Verity operators **index_any** and **near.** These operators provide the parameters for the search on the Verity collections, which narrows the result set from the entire *copy* column to only the documents that contain the search criteria.

- Adaptive Server processes the **select** statement:

  ```
  select t1.score, t2.copy
  from p_blurbs t1, blurbs t2
  where t1.id=t2.id
  ```

  to join the *blurbs* and the *p_blurbs* tables (the source and index tables, respectively) on their IDENTITY columns. If you run only this **select** statement, the result set is all the rows in the *copy* column of the *blurbs* table.

Figure 1-1 describes how Adaptive Server and the Full-Text Search engine process the query:

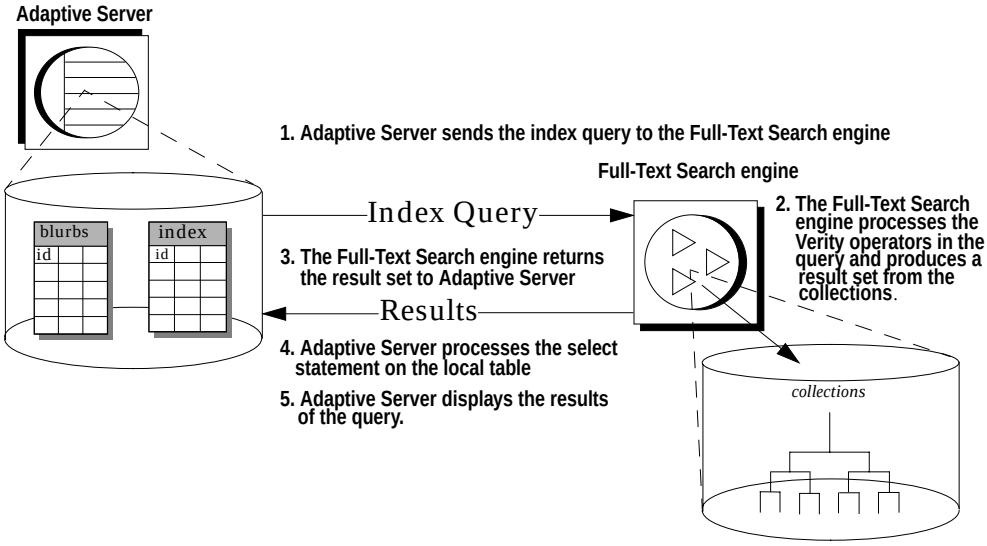


Figure 1-2: Steps for processing a full-text search query

Hardware and Software Requirements

To install and use the Standard Full-Text Search Specialty Data Store product, you must already have Adaptive Server release 11.5 installed. The Full-Text Search engine requires the same operating system and patches that are required by Adaptive Server release 11.5.

Requirements for Windows NT

- Windows NT 4.0; service pack number 2 or later
- A minimum of 64MB RAM – Sybase recommends 128MB
- 100MB disk space for the Full-Text Search engine files.
- Twice the amount of space required by the text being indexed (for example, if you have 100MB of text, you need 200MB of space)

Requirements for Sun Solaris

- Sun Solaris 2.5.1

- Minimum of 64MB of RAM
- 90MB of disk space for the Full-Text Search engine and Verity collections
- Twice the amount of space required by the text being indexed (for example, if you have 100MB of text, you need 200MB of space).

2 Installing the Full-Text Search Engine on UNIX Platforms

This chapter contains information about unloading the Full-Text Search engine from the media and configuring it to work with Adaptive Server.

Installing the Full-Text Search Engine

The Full-Text Search engine is installed in your Sybase installation directory (*$SYBASE*) and adds an *sds* directory to that directory. The *sds* directory contains the Full-Text Search engine installation directory. After installation, the structure of the Sybase installation directory is similar to the structure shown in Figure 2-1.

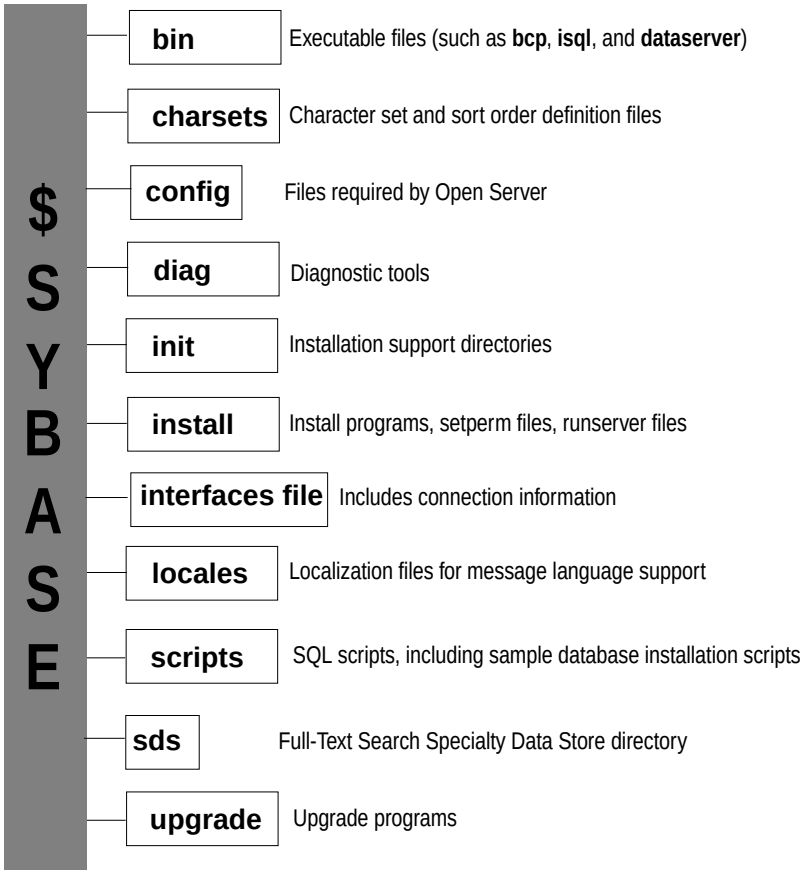


Figure 2-1: Sybase installation directory

The Full-Text Search engine installation directory is in *$SYBASE/sds/text* and has the structure shown in Figure 2-2.

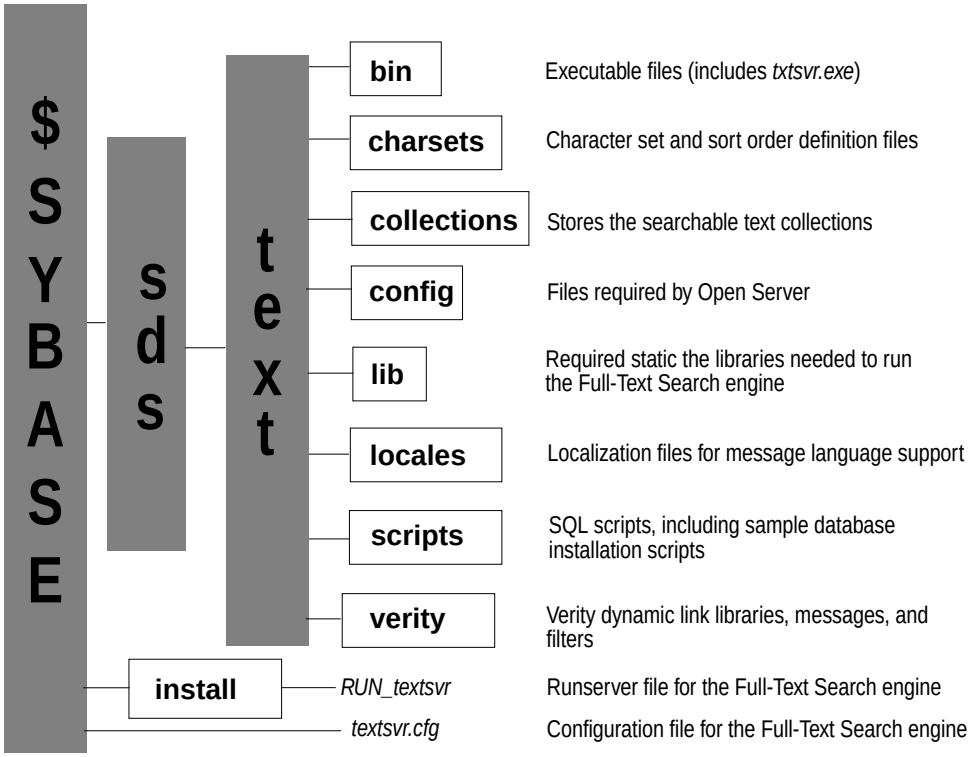


Figure 2-2: Full-Text Search engine installation directory for UNIX platforms

Note that the *RUN_textsvr* file is copied to the *$SYBASE/install* directory and the *textsvr.cfg* file is copied directly to the *$SYBASE* directory.

Unloading the Files from Media

You must install the Full-Text Search engine with Adaptive Server release 11.5. You can install Full-Text Search engine on a machine other than the machine on which Adaptive Server is installed if the two machines have a shared drive.

By default, the **sybsetup** utility is run from the CD. Optionally, you can copy the **sybsetup** executable to your *$SYBASE/bin* directory. Doing so allows you to run **sybsetup** and invoke the configuration utilities from within the **sybsetup** graphical interface. At the end of the unload

process, **sybsetup** lets you copy the **sybsetup** executable to the *$SYBASE/bin* directory.

The Full-Text Search engine is released on CD. Complete the following steps to unload the files from the media:

1. Check your operating system kernel to make sure the ISO 9660 option is on.
2. Make sure the SYBASE environment variable is set to the Sybase installation directory.
3. Place the Sybase products CD in the CD-ROM drive. Solaris automatically mounts the CD.

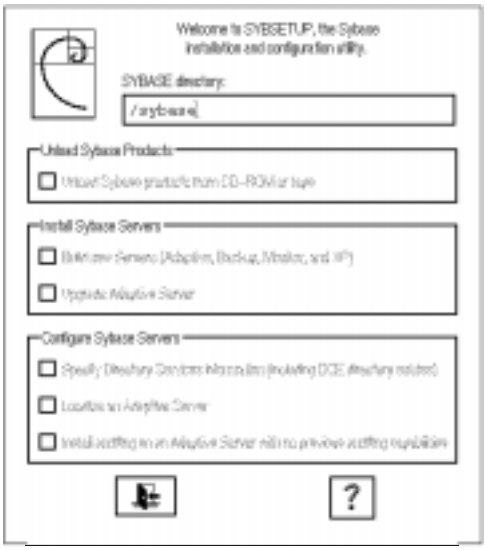
➤ ***Note***

If you are unable to mount the CD-ROM drive, consult your operating system documentation or contact your System Administrator.

4. At the command prompt, enter:

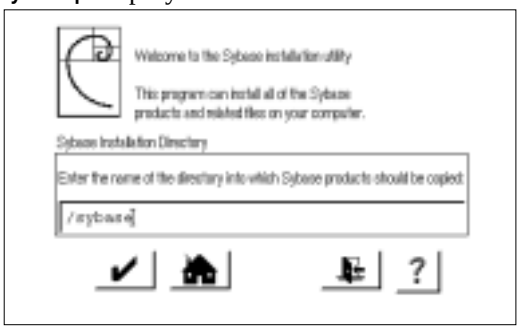
   `/cdrom/cdrom0/sybsetup`

   The **sybsetup** utility appears:



5. Make sure the Sybase Directory window lists the directory into which you want the Full-Text Search engine software unloaded (This is the same as your *$SYBASE* directory). If it is not, enter the correct directory.

6. Select Unload Sybase Products from the **sybsetup** dialog box. **sybsetup** displays the Installation Destination dialog box



   The Sybase Installation Directory window lists the directory into which the software is unloaded. If it is not correct, enter the correct directory.

7. Select the check button. **sybsetup** displays the Installation Source dialog box:

   sybsetup displays the path to the image on the CD that contains the compressed software image. The information is in the format:

   ***CD-ROM_device*/sybimage**

   If **sybsetup** does not display the information, enter this value, specifying the device location followed by "/*sybimage*" (the compressed file containing the Sybase product files).

➤ ***Note***

If you get CD-reading errors, check your operating system kernel to make sure the ISO 9660 option is on.

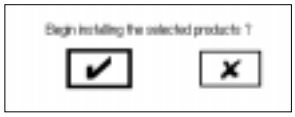
8. Choose the check button. **sybsetup** displays the Products Selection dialog box, which lists the products available for installation:

9. Select Standard Full Text Search SDS from the Available Products list.

   After you select the product, **sybsetup** lists the space required in the Total Space Required window. If there is not enough space available, sybsetup issues a message similar to the following:

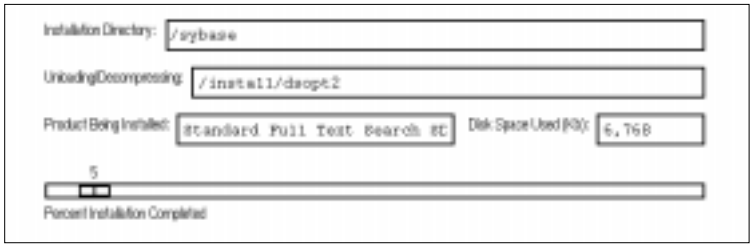
   See your System Administrator about increasing the space necessary to install the Full-Text Search engine.

10. Choose the check button. **sybsetup** displays the Install Products? dialog box:



11. Choose the checkbox to install the Full-Text Search engine.

12. **sybsetup** displays the Installation Status dialog box, which describes the progress of the Full-Text Search engine installation:



13. **sybsetup** displays a message box stating that the installation is complete.
14. Log out, then log in as "root" and enter the following command to remove the CD from the drive:

    ```
    /usr/bin/eject cd
    ```

15. Proceed to "Configuring the Full-Text Search Engine," below, to configure the Full-Text Search engine.

Configuring the Full-Text Search Engine

This sections describes how to configure Adaptive Server to connect to the Full-Text Search engine. The configuration process includes:

- Setting the SYBASE environment variable
- Using the **srvbuild** utility to:
  - Add an entry to the interfaces file
  - Edit the configuration file
  - Create a runserver file

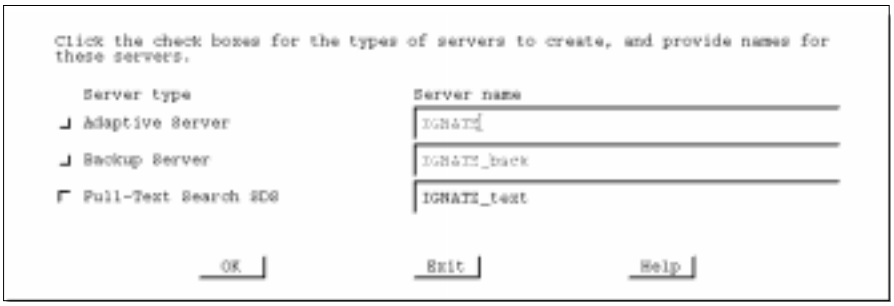
Run *srvbuild*

Follow these steps to configure and start a Full-Text Search engine.

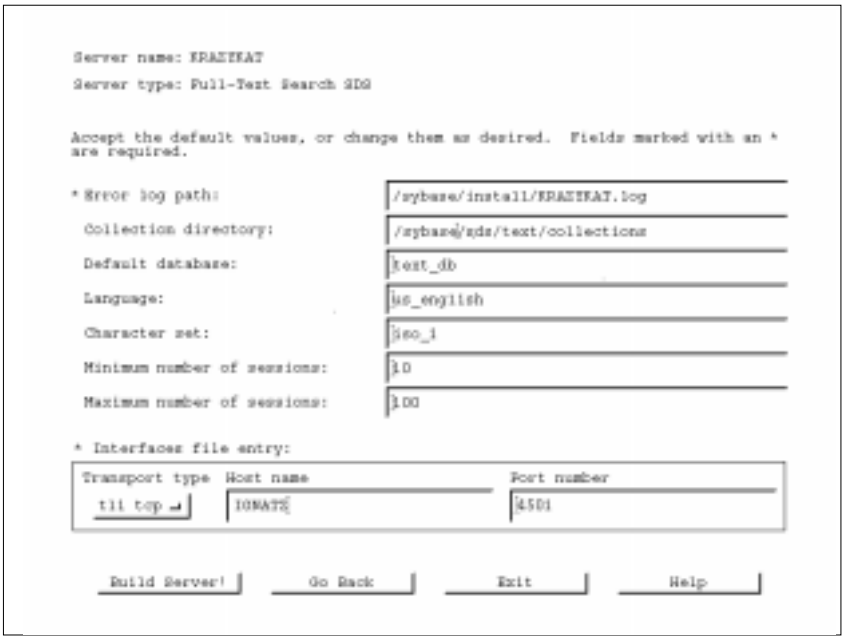
1. Log in as the user "sybase" before you start the Full-Text Search engine.
2. Enter the following command to start the **srvbuild** utility:

   ```
   $SYBASE/bin/srvbuild
   ```

srvbuild displays the Select Servers to Build menu:



3. Select Full-Text Search SDS. **srvbuild** activates the Server Name box and enters the default name "*machine_name*_text" for the name of your Full-Text Search engine, where *machine_name* is the name of the machine on which you installed the Full-Text Search engine.
4. Either accept the default name or enter a new name in the Server Name box. The name you enter here also determines the name of the configuration file, *server_name.cfg*. For example, if you select KRAZYKAT as the name of your Full-Text Search engine, your configuration file is named *KRAZYKAT.cfg*.
5. Choose OK. **srvbuild** displays the Server Attribute Editor dialog box, where you select options for the configuration file and the Full-Text Search engine entries for the interfaces file:



Following are the configuration file options – the only option you **must** select is the errorlog path:

- Error log path – the full path name to the Full-Text Search engine error log file. The default entry for the error log path is *$SYBASE/install/server_name_text.log*.
- Collection directory – the location of the Verity collections. The default location of the collections is *$SYBASE/sds/text/collections*.
- Default database – the name of the Full-Text Search engine database. The default name is *text_db*. This database is created in Adaptive Server and contains the *vesaux* and *vesauxcol* tables. You are only naming the default database at this time; you will create it following the instructions in “Running the installtextserver Script” on page 4-4.
- Language – the language used by the Full-Text Search engine. The default is us_english. The language parameter should be set to the same value as Adaptive Server.
- Character set – the character set used by the Full-Text Search engine. The default is iso_1. The character set parameter should be set to the same value as Adaptive Server.

- Minimum number of sessions – defines the **min_sessions** parameter, which specifies the minimum number of user sessions for the Full-Text Search engine. The default is 10. For more information about **min_sessions**, see "Setting min_sessions and max_sessions" on page 5-6.
- Maximum number of sessions – defines the **max_sessions** parameter, which specifies the maximum number of user sessions for the Full-Text Search engine. The default is 100. For more information about **max_sessions**, see "Setting min_sessions and max_sessions" on page 5-6.

You can adjust any of these options for a configuration that best suits your site.

6. Edit the interfaces file. Select Transport Type, Host Name, and Port Number. The following list describes each option:
   - Transport Type – The network interface used by your system. Select either tli tcp or tcp spx.
   - Host Name – the name of the machine on which you installed the Full-Text Search engine.
   - Port Number – the port number Adaptive Server uses to connect to the Full-Text Search engine.
7. Select Build Server! to build the Full-Text Search engine. **srvbuild** displays the Status Output window, which describes the process of building the Full-Text Search engine. While it builds the Full-Text Search engine, **srvbuild** also creates the runserver and configuration files. **srvbuild** sets both the SYBASE and the LD_LIBRARY_PATH environment variables.

   After **srvbuild** completes, the Full-Text Search engine is now running, but is not yet connected to the Adaptive Server.
8. Exit **srvbuild**.

Continue the Configuration for UNIX

Go to "Configuring the Full-Text Search Engine and Adaptive Server" on page 4-1 to continue configuring the Full-Text Search engine to connect to Adaptive Server.

3 Installing the Full-Text Search Engine on Windows NT

This chapter contains information about unloading the Full-Text Search engine from the media.

Installing the Full-Text Search Engine

The Full-Text Search engine is installed in your Sybase installation directory (*%SYBASE%*) and adds an *sds* directory to that directory. The *sds* directory contains the Full-Text Search engine installation directory. After installation, the structure of the installation directory is similar to the structure shown in Figure 3-1.

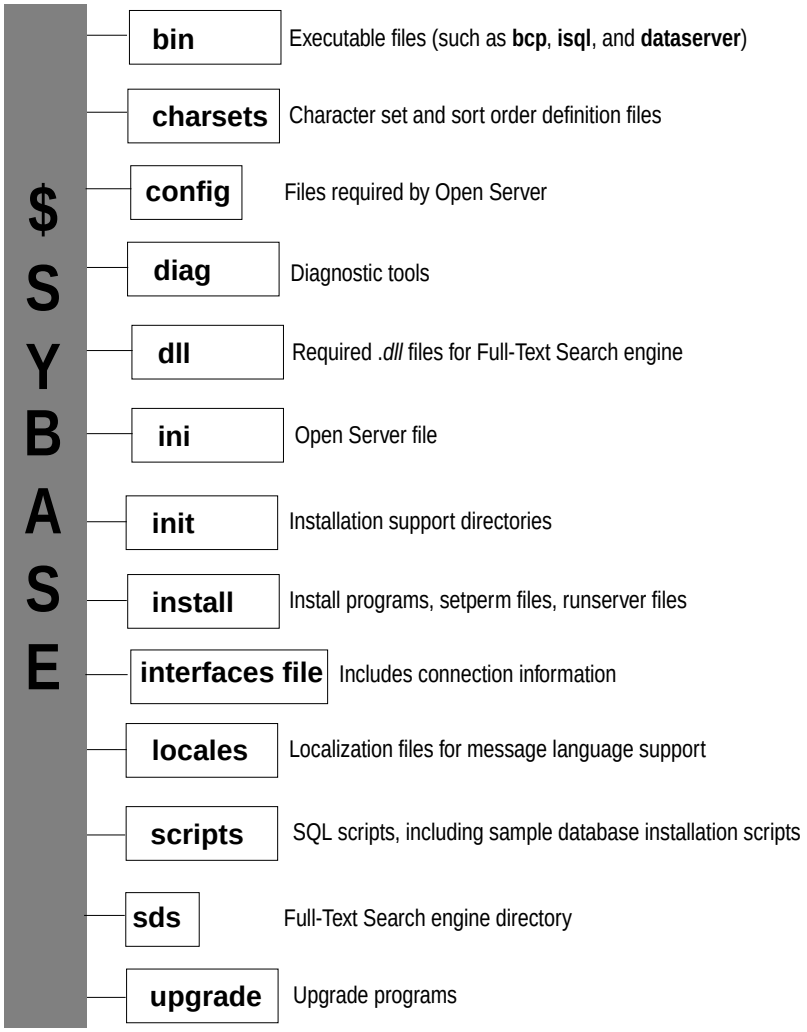


Figure 3-1: Sybase installation directory

The Full-Text Search engine installation directory is in *%SYBASE% \ sds \ text* and has the structure shown in Figure 3-2

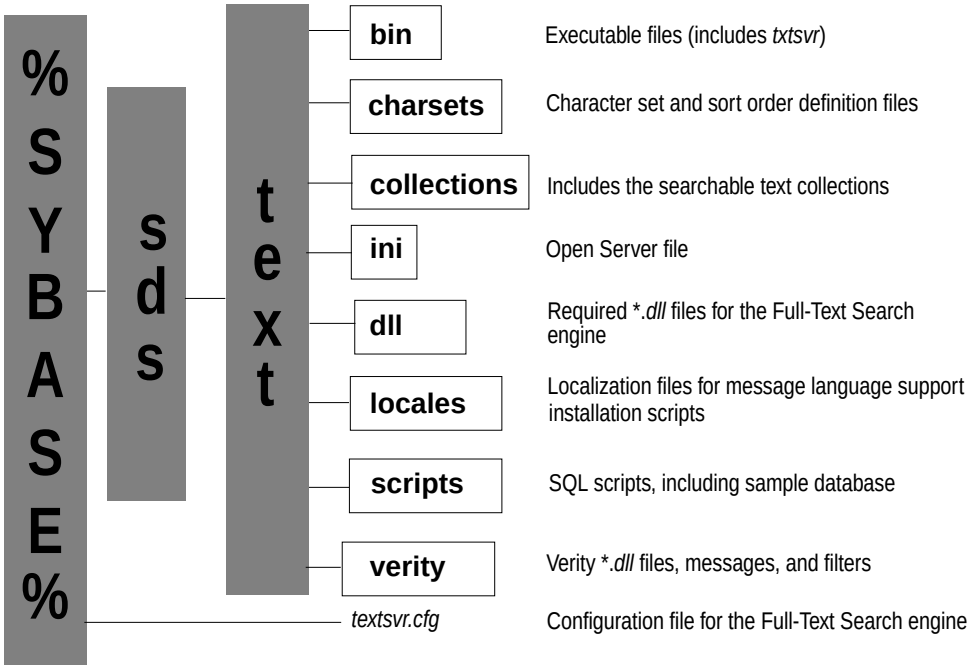


Figure 3-2: Full-Text Search engine installation directory for Windows NT

Note that the *textsvr.cfg* file is copied to the *%SYBASE%* directory.

Unloading Files from the Media

You must install the Full-Text Search engine with Adaptive Server release 11.5. You can install Full-Text Search engine on a machine other than the machine on which Adaptive Server is installed, if the two machines have a shared drive.

Complete the following steps to unload the files from the media.

1. Make sure that the SYBASE environment variable is set to the Sybase installation directory. For example, if Adaptive Server is installed in *C:\sybase*, you would set *%SYBASE%* as follows:

   ```
   set SYBASE=C:\sybase
   ```

2. Insert the Full-Text Search engine product CD into the drive.
3. To start the installer, choose Run from the File menu, and type:

```
n:\setup.exe
```

where *n* is the letter for your CD-ROM drive.

Alternatively, you can start the installer by opening the Windows NT Explorer and double-clicking *setup.exe* in the CD directory.

4. The installation wizard guides you through the installation process.
5. At the end of the installation, the installer asks if you want to read the README file. Select Yes or No. This completes the installation of the Full-Text Search engine. Continue the configuration of the Full-Text Search engine by following the procedures outlined in "Configuring the Full-Text Search Engine," below.

Configuring the Full-Text Search Engine

Perform the steps in this section to configure Adaptive Server to connect to the Full-Text Search engine. This process includes:

- Setting the SYBASE environment variable
- Setting the PATH environment variable
- Adding entries to the interfaces file
- Editing the parameters in the configuration file
- Enabling CIS
- Adding the Full-Text Search engine to *sysservers*

Set the SYBASE Environment Variable

If it is not already set, set the SYBASE environment variable to point to the Adaptive Server installation directory.

Set the Full-Text Search Engine to the PATH Environment Variable

The Full-Text Search engine includes libraries that enable it to connect to Adaptive Server. Include the following in your PATH environment variable:

```
%SYBASE%\sds\text\dll
```

Editing the Interfaces File

The **dsedit** utility adds entries to the interfaces file. Adaptive Server uses the interfaces file to define connection information for servers and clients. Follow the steps below:

1. Start **dsedit** in one of the following ways:
   - Start the Windows NT Explorer and move to the *%SYBASE% \ bin* directory. Double-click on *dsedit.exe*.
   - Move to the *%SYBASE% \ bin* directory. At the command line, enter:

     ```
     dsedit
     ```

     dsedit displays the Select Directory Services dialog box.
2. Make sure **dsedit** is pointing to the correct configuration file. The default is *%SYBASE% \ ini \ libtcl.cfg*. If your configuration file is in another directory, change this setting.
3. Highlight the directory service listed in the DS Name list box that you want to open, and choose OK. **dsedit** displays the Interfaces Driver dialog box. Table 3-1 describes the attributes:

Table 3-1: Default attributes in dsedit

Attribute	Value	Description
Server Entry Version	1	Can be any positive integer value and can be set to indicate the software version of Adaptive Server (this attribute works only if you use the NT Registry Directory Services).
Server Name	*server_name*	The name of the Full-Text Search engine.
Server Status	4 Unknown	Indicates the state of the Full-Text Search engine (active, stopped, failed, or unknown, including a corresponding number).
Server Address	User configures	Connection information for the Full-Text Search engine.

4. Choose Add from the Server Object drop-down list. **dsedit** displays the Input Server Name dialog box.

5. Enter the name of the Full-Text Search engine in the Server Name box, and choose OK. **dsedit** returns to the Interfaces Driver dialog box, which displays the default values for Server Entry Version, Server Service, and Server Status.
6. In the Interfaces Driver dialog box, double-click Server Address. **dsedit** displays the Network Address Attribute dialog box.
7. Choose Add. **dsedit** displays the Input Network Address for Protocol dialog box.
8. Select the network protocol from the Protocol drop-down list. dsedit displays the Network Address dialog box.
9. Enter the network address information.
10. Choose OK.
11. **dsedit** returns to the Network Address Attribute dialog box and displays the new network connection information. Choose OK to return to the Interfaces Driver dialog box.

Edit the Configuration File

The Full-Text Search engine is shipped with a default configuration file named *textsvr.cfg*. This file is copied to %SYBASE% when the software is unloaded from the CD. You must manually edit the configuration file for Windows NT, as described in the following sections.

Rename the *textsvr.cfg* File

The syntax for naming the configuration file is *server_name.cfg*, where *server_name* is the name of the Full-Text Search engine you are installing. By default, the Full-Text Search engine is named "textsvr." If your Full-Text Search engine is not named "textsvr," make a copy of the sample configuration file using the name of the Full-Text Search engine as the prefix. For example, a Full-Text Search engine named KRAZYKAT would have a configuration file named *KRAZYKAT.cfg*.

Change the Name of the Full-Text Search Engine

The server name in the default configuration file is "textsvr." If your Full-Text Search engine is named something other than "textsvr", edit the configuration file to reflect this. The configuration file names

the Full-Text Search engine as the first entry in values section of the file:

```
[textsvr]
```

For example, if you named your Full-Text Search engine KRAZYKAT, you would edit the configuration file to read:

```
[KRAZYKAT]
```

➤ ***Note***

The *textsvr.cfg* file is shipped with default parameters listed but commented out. If necessary, you can alter these defaults in the configuration file for your Full-Text Search engine to better suit the needs of your site. Table 5-3 on page 5-12 lists all the parameters in the configuration file.

Starting the Full-Text Search Engine on Windows NT

You can start the Full-Text Search engine either from Sybase Central™, as a service, or from the command line:

- From Sybase Central – see your Sybase Central documentation for information about using this utility to start servers.
- For information about starting the Full-Text Search engine as a service, see "Starting the Full-Text Search Engine As a Service" on page 3-8.
- From the command line – use the following syntax:

%SYBASE%\sds\text\bin\txtsvr.exe -S*server_name* -t -i%SYBASE%*path_to_sql.ini_ file* -l %SYBASE%*path_to_errorlog*

where:

- **-S***server_name* is the name of the Full-Text Search engine you are starting.
- **-t** directs start-up messages to standard error.
- **-i** indicates the path to the *sql.ini* file.
- **-l** indicates the path to the error log.

For example, to start a Full-Text Search engine named KRAZYKAT using the default *sql.ini* and error log files, enter:

```
%SYBASE%\sds\text\bin\txtsvr -SKRAZYKAT -t
-i%SYBASE%\sds\text\bin\sql.ini -l%SYBASE%\sds\text\bin\KRAZYKAT.log
```

The Full-Text Search engine is now up and running.

Starting the Full-Text Search Engine As a Service

Using the **instsvr** utility, you can add the Full-Text Search engine to the list of items you can start and stop using the Services utility. **instsvr** is located in the *%SYBASE%\sds\text\bin* directory.

The **instsvr** utility uses the following syntax:

```
instsvr.exe service_name executable_location "startup_parameters"
```

where:

- *service_name* is the name of the Full-Text Search engine you are adding as a service.
- *executable_location* is the location of the *txtsvr.exe* file.
- *startup_parameters* are any parameters you want used at start-up.

For example, to install a Full-Text Search engine named KRAZYKAT as a service, enter:

```
instsvr.exe KRAZYKAT %SYBASE%\sds\text\bin\txtsvr.exe
"-SKRAZYKAT"
```

➤ *Note*

Note that if you need to include more than one parameter (for example, **-i**), you must include all the parameters in a quote set.

To configure Sybase Central to start and stop your Full-Text Search engine, you must provide a service name beginning with **SYBTXT**_*server_name*, where *server_name* is the name of the Full-Text Search engine listed in the interfaces file. For example, if the name listed in the interfaces file for your Full-Text Search engine is KRAZYKAT, run the following **instsvr** command to create a service manageable by Sybase Central:

```
instsvr SYBTXT_KRAZYKAT %SYBASE%\sds\text\bin\txtsvr.exe "-SKRAZYKAT"
```

Go to "Chapter 4, "Configuring the Full-Text Search Engine and Adaptive Server," to continue the configuration of the Full-Text Search engine.

4 Configuring the Full-Text Search Engine and Adaptive Server

This chapter describes the process of configuring the Full-Text Search engine and Adaptive Server to perform full-text searches. The process of configuring the Full-Text Search engine and Full-Text Search engine includes the following:

- Enabling Component Integration Services (CIS)
- Creating the text database
- Configuring the text database to support text indexing
- Configuring the source tables for indexing
- Installing the Full-Text Search engine messages for stored procedures
- Creating the text index
- Bringing the text database online

Enabling CIS

You must enable CIS on Adaptive Server before Adaptive Server can connect to the Full-Text Search engine. To enable CIS, connect to Adaptive Server with **isql** and enter:

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

Adaptive Server displays a series of messages stating that you have altered a configuration parameter and that Adaptive Server must be rebooted for the new configuration parameter to take effect.

Configuring the Text Database

The text database contains the *vesaux* and *vesauxcol* tables, which provide the metadata for the Full-Text Search engine. However, the text database does not contain any user data. The Full-Text Search engine refers to these tables

- When it is brought online, to verify that the text collections exist.
- When a table is updated, to determine which collections are affected.

The default name of the database is *text_db*, although you can specify a different name in the Full-Text Search engine configuration file.

However, if you change the name of the text database, you must also change the name used in the installation scripts **installtextserver** and **installevent**.

- See "Editing the installtextserver Script" on page 4-3 and "Editing the installevent Script" on page 4-4 for information about changing names in the installation scripts.
- See "Running srvbuild" on page 3-2 and"Manually Editing the Configuration File" on page 5-9 for more information about the configuration file.

The *vesaux* table contains the columns shown in Table 4-1:

Table 4-1: Columns in the vesaux table

Column Name	Description
id	IDENTITY column.
object_name	Name of the table on which the external index is being created.
collection_id	Name of the Verity collection.
key_column	Name of the IDENTITY column in the source table.
svr-id	Server ID of the Full-Text Search engine maintaining the collection.

The columns in the *vesauxcol* table are shown in Table 4-2:

Table 4-2: Columns in the vesauxcol table

Column Name	Description
id	ID of the referenced row in the *vesaux* table.
col_name	Name of the column for which you are searching.
col_type	Column type (*text*, *image*, *char*, *varchar*, *datetime*, *smalldatetime*).

Creating the Text Database

The **installtextserver** script:

- Defines the Full-Text Search engine to the Component Integration Services running on Adaptive Server.

- Creates the database you specify in the **installtextserver** script (*text_db* by default).
- Installs the system procedures required by the Full-Text Search engine.

➤ ***Note***

If it already exists, the **installtextserver** script drops the database you specify in the **installtextserver** script (*text_db* by default).

You should only have to run the **installtextserver** script once. To add another Full-Text Search engine use **sp_addserver**. See "Adding the Full-Text Search Engine to sysservers" on page 3-8 for more information about **sp_addserver**.

For a list and description of the system procedures added with the **installtextserver** script, see Appendix A, "System Procedures."

Editing the *installtextserver* Script

By default, the **installtextserver** script installs a text database called *text_db* for a Full-Text Search engine named *textsvr*. If your Full-Text Search engine or text database are named differently, you must edit this script so that it refers to the appropriate Full-Text Search engine and text database.

Note that if you specify the name of an existing database in the **installtextserver** script, it will be dropped. For example, if you specified the name *pubs2* in the script, the existing *pubs2* database would be dropped and the database defined by the **installtextserver** script would be created under that name.

The **installtextserver** script is located in the *$SYBASE/sds/text/scripts* directory. Use a text editor (such as **vi** or **emacs**) to open the script, and replace the name of the Full-Text Search engine and the text database with the names of the Full-Text Search engine and the text database on your system.

➤ ***Note***

The **installtextserver** script will fail if you do not edit this script so that it uses the correct names.

Running the installtextserver Script

Using **isql**, run the **installtextserver** script to install the text database and the associated system procedures in Adaptive Server. For example, to run the **installtextserver** script on an Adaptive Server named IGNATZ, enter:

```
isql -Ulogin -Ppassword -SIGNATZ -i
$SYBASE/sds/text/scripts/installtextserver
```

This command consists of a single line; it is in two lines here for formatting purposes.

Preparing Adaptive Server to Support Text Indexing

You must enable the text indexes in user databases before you can bring them online and begin performing full-text searches.

The *events* Table

Each database containing tables referenced by a text index must contain an *events* table, which logs **inserts**, **updates**, and **deletes** to indexed columns. Run the **installevent** script, as described below, to create the *events* table in Adaptive Server. You should run the **installevent** script in any existing database that includes tables that are referenced by the text index.

Editing the installevent Script

By default, the **installevent** script creates an events table named *text_events* in the *model* database. If it is run as it is shipped, all newly created databases will have an *events* table. To install the *events* table in an existing user database, edit the script and replace all references to *model* with the user database name.

The **installevent** script is located in the *$SYBASE/sds/text/scripts* directory. Use a text editor (such as **vi** or **emacs**) to open the script, and replace the name *text_db* with the name of the text database on Adaptive Server.

➤ ***Note***

If you do not edit this script to use the correct name, changes to the source table will not be propagated to the Verity collections.

Running the installevent Script

➤ ***Note***

You must run the **installtextserver** script before you run the **installevent** script.

Using **isql**, run the **installevent** script to install the *events* table in Adaptive Server. For example, to run the **installevent** script in a server named IGNATZ, enter:

```
isql -Usa -P -SIGNATZ -i $SYBASE/sds/text/scripts/installevent
```

Adding the Full-Text Search Engine Messages

The Full-Text Search engine has its own set of messages for stored procedures that must be installed in Adaptive Server. The messages are installed using the **installmessages** script. You should only have to run the **installmessages** script once.

For example, to run the **installmessages** script in a server named IGNATZ, enter:

```
isql -Usa -P -SIGNATZ -i $SYBASE/sds/text/scripts/installmessages
```

Preparing Tables for Text Indexing

The tables in the text database need to be indexed so that you can perform full-text searches. Use **sp_create_text_index** to create this text index. The text index can contain up to 16 columns. Columns of the following datatypes can be indexed:

char, varchar, nchar, nvarchar, text, image, datetime, smalldatetime

Note that the text database does not have to be brought online to create or drop indexes, only to issue queries.

The Source and Index Tables

The source and index tables allow you to use the Full-Text Search engine to search the Verity collections. Full-text search queries join these two tables on their IDENTITY columns to produce the result set.

The Source Table

The source table contains the data on which you perform searches (for example, the *blurbs* table in the *pubs2* database). Other than the columns needed to hold your text or character set data, you must also create an IDENTITY column for the source table, if it does not already exist. See "Adding an IDENTITY Column" on page 4-7 for information about creating an IDENTITY column.

The *index* Table

The index table is created when the text index is created. It contains pseudo columns and a column named *id* that is used to join it with the source table. See "Creating the Text Index and Index Table", below, for information about creating the *id* column.

Pseudo Columns in the Index Table

pseudo columns are columns in the index table that define the parameters of the search and provide access to the results data. These columns are valid only in the context of the query; that is, the information in the columns is valid only for the duration of the query. If the query that follows contains a different set of parameters, the pseudo columns will contain a different set of values. Each pseudo column in an index table describes a different search attribute. For example, if you indicate the *score* column, the query will display only the result set that falls within the parameters you define. The following query will display only the results that have a score value greater than 90:

```
table_name.score > 90
```

You could perform the same search, but indicate:

```
table_name.score > 50
```

The query will search for the same data, but the result set will potentially be larger

Other pseudo columns (like *highlight*) are used to retrieve data generated by Verity for a particular document.

Table 4-3 describes the pseudo columns that are maintained by the Full-Text Search engine:

Table 4-3: Pseudo columns created by Full-Text Search engine

Pseudo Column Name	Description	Datatype	Length (in bytes)
highlight	Used in the target list of the query to return the Extended Markup Langauge (XML) of matched words. You can use *highlight* only in the **select** clause of a query.	*text*	16
id	Uniquely identifies a document within a collection.	*numeric*	6
index_any	Used to provide a Verity language query to the Full-Text Search engine.	*varchar*	255
max_docs	Limits results to the first *n* documents, based on the default sort order.	*int*	4
score	The normalized measure of correlation between search strings and indexed columns. The *score* associated with a specific document has meaning only in reference to the query used to retrieve the document.	*int*	4
sort_by	Specifies the sort order in which to return the result set.	*varchar*	35
summary	Selects summarization data.You can use *summary* only in the **select** clause of a query.	*varchar*	255

Adding an IDENTITY Column

Every source table must contain an IDENTITY column, which uniquely identifies each row and provides a means of joining the index table and the source table. When you create a text index, the IDENTITY column is passed with the indexed columns to the Full-Text Search engine. The IDENTITY value is stored in the text index and is mapped to the id column in the index table.

The IDENTITY column needs to have sufficient precision and scale to guarantee a unique IDENTITY for each row. Sybase recommends a precision of 10 and a scale of 0. You can use an existing IDENTITY column, if it is defined with sufficient precision and scale to identify each row uniquely.

For example, to create an IDENTITY column in a table named *composers*, define the table as follows:

```
create table composers (
    id          numeric(10,0)   identity,
    comp_fname  char(30)        not null,
    comp_lname  char(30)        not null,
    text_col    text
)
```

To add an IDENTITY column to an existing table, enter:

```
alter table table_name add id numeric(10,0) identity
```

The text index is populated automatically during index creation. See "Creating the Text Index and Index Table", below, for information about creating the index.

Adding a Unique Index to an IDENTITY Column

The IDENTITY column must have a unique index, which would contain only the IDENTITY column. For example, to create a unique index named *comp_id* on the IDENTITY column created above, issue the command:

```
create unique index comp_id
on composers(id)
```

For more information about creating a unique index, see Chapter 11, "Creating Indexes on Tables," of the *Transact-SQL User's Guide*.

Creating the Text Index and Index Table

Use **sp_create_text_index** to create the text index. This stored procedure:

- Updates *vesaux* and *vesauxcol*
- Creates the Verity collections
- Populates the Verity collections
- Creates the index table

➤ ***Note***

Make sure the text database name in the configuration file (listed after the *defaultDb* parameter) matches the database name in Adaptive Server. If they do not match, the text index cannot be created. The default name of the text database is *text_db*. If your text database is named something other than *text_db*, edit your configuration file and change *text_db* to the correct name.

Note that populating the Verity collections can take a long time, depending on the amount of data you are indexing. You may want to perform this step at a time when the server is not being heavily used. Increasing the **batch_size** configuration parameter will also expedite the process. See "batch_size Configuration" on page 5-4 for more information about increasing the batch size.

For example, to create an index table named *p_plurbs* for the *blurbs* table in *pubs2* on KRAZYKAT, enter:

```
sp_create_text_index "KRAZYKAT", "p_blurbs", "blurbs", " ",
"copy"
```

where:

- "KRAZYKAT" is the name of the Full-Text Search engine
- "p_blurbs" is the index table you are creating
- "blurbs" is the table for which you are creating the text index
- " " is a placeholder for future character–format definitions.
- "copy" is the column that you are indexing in the index table.

See "sp_create_text_index" on page A-4 for more information about this stored procedure.

Running *sp_refresh_text_index* to Propagate Changes to the Index

The **sp_refresh_text_index** stored procedure notifies the Full-Text Search engine that changes have been made to a source table in Adaptive Server. This stored procedure logs changes to the *events* table. After you have logged the changes, run **sp_text_notify** to notify the Full-Text Search engine that changes exist that need to be processed. The Full-Text Search engine then connects to Adaptive Server, reads the entries in the events table, and determines which indexes, tables, and rows are affected.

Sample Configuration: Bringing the *pubs2* Database Online

The process of configuring a database so that you can perform full-text searches includes three steps. The steps below describe the process of bringing the *blurbs* table of the *pubs2* database online. This process assumes that Adaptive Server and the Full-Text Search engine have been configured to connect to each other.

1. Bring the Database Online

You must bring the database online first so that the Full-Text Search engine can initialize the internal Verity structures and confirm that the Verity collections exist. Use the **sp_text_online** stored procedure to bring databases online. For example, to bring the *pubs2* database online for a Full-Text Search engine named KRAZYKAT, enter:

```
sp_text_online KRAZYKAT, pubs2
```

This message appears:

```
Database 'pubs2' is now online
```

The *pubs2* database is now available for performing full-text searches.

See "sp_text_online" on page A-14 for more information.

2. Create an IDENTITY Column

Every source table must contain an IDENTITY column, which provides a means of joining the index table and the source table. The *blurbs* table includes the *au_id* column, which serves as an index and IDENTITY column.

3. Create the Index Table

The index table contains the pseudo columns that determine the parameters of the search on the Verity collections. It also contains an *id* column that is joined with the source table IDENTITY columns to determine the result set during a search. The **sp_create_text_index** stored procedure creates both the text index (the Verity collection under *$SYBASE/sds/text/collection*) and the index table. To create an index table named *p_blurbs* and a text index for the *blurbs* table, use the following command:

```
sp_create_text_index "KRAZYKAT", "p_blurbs", "blurbs", "format
character", "copy"
```

The *blurbs* table is now online and available for running full-text searches.

5 Administration and Tuning

This chapter describes system administration and performance and tuning issues for the Full-Text Search engine.

Backup and Recovery

The Adaptive Server text database and the Verity collections are physically separate. Backing up your text database does **not** back up the Verity collections, and restoring your database from a backup does **not** restore your Verity collections. The backup and recovery procedures described in Chapter 21, "Backing Up and Restoring User Databases," of the *System Administration Guide* apply only to the text database in Adaptive Server.

Make sure you follow the recommended schedule for backing up your databases that is described in the backup and recovery section of the *System Administration Guide*. A regular backup schedule ensures the integrity of the *events* table, which is integral to recovering your text index without having to drop and re-create it.

Backing Up the Verity Collections

Perform the following procedures to back up your Verity collections:

1. Shut down the Full-Text Search engine. See "Shutting Down the Full-Text Search Engine" on page 5-9 for information about shutting down the Full-Text Search engine.
2. Back up the files. By default the collections are located in:

   $SYBASE/sds/text/collections/db.owner.index

   where each collection name consists of the database name, owner name, and index name, in the format *db.owner.index*. For example, if you created a text index called *p_blurbs* on the *pubs2* database, the full path to those files would be similar to:

   $SYBASE/sds/text/collections/pubs2.dbo.p_blurbs

   For UNIX, back up the files using the **tar** or **cpio** utility. For Windows NT, use a compression utility such as PKZIP to back up the files.
3. Note the time of the backup in a permanent location for future reference.

4. Restart the Full-Text Search engine. See “Starting the Full-Text Search Engine on UNIX” on page 5-9 and “Starting the Full-Text Search Engine on Windows NT” on page 3-7 for information about starting the Full-Text Search engine.

Restoring Your Collections and Text Indexes from Backup

As Database Administrator, perform the following procedures to restore your Verity collections:

1. Restore your Adaptive Server text databases. This returns the databases and the *events* table to a consistent and predictable state. Follow the procedures described in Chapter 21, “Backing Up and Restoring User Databases,” in the *System Administration Guide* to restore your text database.
2. Shut down the Full-Text Search engine. See “Shutting Down the Full-Text Search Engine” on page 5-9 for information about shutting down the Full-Text Search engine.
3. Restore your collections from the backup file created following the instructions in step 2 in “Backing Up the Verity Collections,” above.
4. Restart the Full-Text Search engine. See “Starting the Full-Text Search Engine on UNIX” on page 5-9 and “Starting the Full-Text Search Engine on Windows NT” on page 3-7 for information.
5. Log into Adaptive Server and run the following stored procedure in the recovered database. For example, if you are restoring the *pubs2* database, you have to be in that database to run the stored procedure, **sp_redo_text_events**, as follows:

   ```
   sp_redo_text_events "Verity_backup_date"
   ```

   where *Verity_backup_date* is the date and time associated with the backup used to recover the collections.

   For example:

   ```
   sp_redo_text_events "10/31/97"
   ```

   restores the collections up to October 31, 1997. For more information about **sp_redo_text_events**, see “sp_redo_text_events” on page A-8.
6. Execute **sp_text_notify.** This procedure notifies the Full-Text Search engine that changes need to be propagated. The Full-Text Search engine connects to Adaptive Server, reads all the unprocessed entries in the *events* table, and applies them to the text index. For

more information about **sp_text_notify**, see "sp_text_notify" on page A-13.

Your text indexes and collections are now fully recovered.

Performance and Tuning

The Full-Text Search engine is shipped with a default configuration. You can optimize the performance of the Full-Text Search engine by altering the default configuration so that it better reflects the needs of your site. The following sections describe some ways in which you can enhance performance.

Updating Existing Indexes

The amount of time it takes to update records in a text index can be reduced by enabling (turning on) trace flags 11 or 12, or both.

Enabling trace flag 11 disables Verity collection optimization. That is, if trace flag 11 is enabled, Verity does not optimize the text index after you issue **sp_text_notify**, which is a performance gain. If trace flag 11 is turned off (the default), the Full-Text Search engine calls Verity to optimize the text index at the end of **sp_text_notify** processing, which can delay the the completion of sp_text_notify.

Turning on trace flag 12 disables the Full-Text Search engine from returning **sp_statistics** information. If trace flag 12 is turned off (the default), an **update statistics** command is issued to Full-Text Search engine, which can delay the completion of **sp_text_notify**.

If updates to the text index occur as often as every few seconds, you may improve performance by disabling the **update statistics** processing and the Verity optimization, or both, for the majority of **updates**.

Both trace flags 11 and 12 can be enabled and disabled interactively using **sp_traceon** and **sp_traceoff**.

Increasing Query Performance

Two issues can significantly improve query performance:

- Limiting the number of rows returned by the Full-Text Search engine.
- Ensuring the correct join order for queries.

Limiting the Number of Rows

Use the **max_docs** psuedo column to limit the number of rows returned by the Full-Text Search engine. The fewer the number of rows returned by the Full-Text Search engine, the faster Adaptive Server can process the join between the text index and the indexed table will be (the indexed table is the table you created using **sp_create_text_index**).

Ensuring the Correct Join Order for Queries

The more tables and text indexes that are listed in a join, the greater the chance that the query will run slowly because of incorrect join order. Queries run fastest if the text index is queried first during a join between the text index and one or more tables.

The following helps ensure correct join order:

- Make sure that a unique clustered or nonclustered index is created on the IDENTITY column of the table being indexed.
- Limit joins to one base table and one text index.

If a query is running slowly, use **showplan** to examine the join order. The fastest queries contain an **index_any** search condition in the **where** clause and query the text index first.

The slowest queries contain the *id* column in the text index **where** clause, and query the indexed table first. In this case, rewrite the query or use **forceplan** to force the join order that is listed in your query. For more information about **forceplan**, see Chapter 10, "Advanced Optimizing Techniques," in the *Performance and Tuning Guide*.

batch_size Configuration

The **batch_size** configuration parameter determines the number of rows per batch the Full-Text Search engine indexes. *batch_size* is set in the configuration file (*server_name.cfg*) and has a default of 500 (that is, 500 rows of data indexed per batch). Performance improves if you increase the size of the batches that are indexed. However, the larger the batch size, the more memory the Full-Text Search engine allocates to this parameter.

When considering how large to set **batch_size**, consider the size of the data on which you are creating a text index. When creating the text

index, the Full-Text Search engine allocates memory equal to (in bytes):

(*amount of space needed for data*) x (*batch_size*) = memory used

For example, if the data you are indexing is 10,000 bytes per row, and **batch_size** is set to 500, then the text server will need to allocate almost 5MB of memory when creating the text index.

The batch size you choose is based on the typical size of your data and the amount of memory available on your machine.

Improving Performance by Reconfiguring Adaptive Server

You can improve the performance of the Full-Text Search engine by reconfiguring the following Adaptive Server **sp_configure** parameters:

cis cursor rows

The **cis cursor rows** parameter specifies the number of rows received by Adaptive Server during a single fetch operation. The default number for **cis cursor rows** is 50. Increasing this number increases the number of rows received by Adaptive Server from the Full-Text Search engine during a fetch operation. However, keep in mind that the larger the number you set for **cis cursor rows**, the more memory Adaptive Server allocates to that parameter.

See Chapter 11,"Setting Configuration Parameters," in the *System Administration Guide* for information about setting **sp_configure** parameters.

cis packet size

The **cis packet size** parameter determines the number of bytes contained in a single network packet. The default for **cis packet size** is 512. You must specify values for this parameter in multiples of 512. Increasing this parameter improves the performance of Full-Text Search engine because, with a larger packet size, it returns fewer packets for each query. However, keep in mind that the larger the number you set for **cis packet size**, the more memory Adaptive Server allocates to that parameter.

See Chapter 11,"Setting Configuration Parameters," in the *System Administration Guide* for information about setting **sp_configure** parameters.

The **cis packet size** parameter is dynamic; you do not need to reboot Adaptive Server for this parameter to take effect.

➤ ***Note***

If you change the **cis packet size**, you must also change the **max_packetsize** parameter in the Full-Text Search engine configuration file to the same value.

You need to reboot the Full-Text Search engine for **max_packetsize** parameter size to take effect.

Setting *min_sessions* and *max_sessions*

min_sessions and **max_sessions** determine the minimum and maximum number of user connections allowed for the Full-Text Search engine. Each user connection requires about 5MB of memory. Do not to set **max_sessions** to an amount that exceeds your available memory. Also, because the memory for **min_sessions** is allocated at start-up, if you set the number for **min_sessions** extremely high (to allow for a large number of user connections), a large percentage of your memory will be dedicated to user connections for the Full-Text Search engine.

You may improve the performance of the Full-Text Search engine by setting **min_sessions** equal to the average number of user sessions that will be used. Doing so prevents the Full-Text Search engine from having to allocate memory at the start of the user session.

How Often to Issue *sp_text_notify*

Review the needs of your site before you decide how often to issue **sp_text_notify.**

Issuing **sp_text_notify** produces a load on the Full-Text Search engine as the system procedure reads the data and updates the text collections. Depending on the size of this load, the performance hit for issuing **sp_text_notify** can be substantial. Because of the performance implications, you must determine how up to date your indexes need to be. If they need to be current (close to real-time), then you will have to issue **sp_text_notify** frequently (as often as every 5 seconds). However, if your indexes do not need to be that current, it may be prudent to wait until the system is not active before you issue **sp_text_notify.**

➤ ***Note***

You cannot issue **sp_text_notify** from within a transaction.

Configuring Multiple Full-Text Search Engines for Adaptive Server

If you have tables that are used frequently, you can improve performance by placing the indexes for these tables on separate Full-Text Search engines. Performance improves because users can spread their queries over a number of Full-Text Search engines, instead of sending all queries to a single engine.

To Create Additional Full-Text Search Engines

Follow the steps described in Chapters 3 and 4 to configure additional Full-Text Search engines. Note that each Full-Text Search engine requires its own interfaces file entry and its own entry in *sysservers*. All the Full-Text Search engines use the same text database (named *text_db* by default) and the same *vesaux* and *vesauxcol* tables.

If You Are Creating Multiple Full-Text Search Engines at Start-Up

If you are initially creating many Full-Text Search engines, you can edit the **installtextserver** script so that it includes all the Full-Text Search engines you are creating. By doing so, you can include the **sp_addserver** commands for each of the Full-Text Search engine you are configuring to run with Adaptive Server. **installtextserver** includes the following section for naming the Full-Text Search engine you are configuring:

```
/*
** Add the text server
*/
exec sp_addserver textsvr,sds,textsvr
go
```

Edit this script so that it includes entries for all the Full-Text Search engines you are configuring. For example, if you are configuring three Full-Text Search engines named KRAZYKAT, OFFICAPUP, and MOUSE, the lines would be similar to:

```
/*
** Add the text server
*/
exec sp_addserver KRAZYKAT, sds, KRAZYKAT
exec sp_addserver OFFICAPUP, sds, OFFICAPUP
exec sp_addserver MOUSE, sds, MOUSE
go
```

Do not run **installtextserver** more than once because **installtextserver** drops and re-creates *text_db* each time it is run. All metadata is lost when you drop the *text_db*. Any indexes that existed prior to dropping the *text_db* will have to be manually deleted before they can be re-created.

Adding Additional Full-Text Search Engines

You can add additional Full-Text Search engines at a later date by issuing the **sp_addserver** command from **isql**. The **sp_addserver** command has the following syntax:

sp_addserver *lname* [, {*server_class*} [, *pname*]]

where:

- *lname* is the name used to address the server on your system (in this case, the Full-Text Search engine).
- *server_class* identifies the category of server being added. For the Full-Text Search engine, the value is "sds". See **sp_addserver** in the *Adaptive Server Reference Manual* for a list of other categories).
- *pname* is the name in the interfaces file used by the server *lname*.

For more information, see **sp_addserver** in the *Adaptive Server Reference Manual*.

For example, to add a Full-Text Search engine named BLUE, enter:

```
sp_addserver BLUE, sds, BLUE
```

To see if you can connect to the Full-Text Search engine, enter:

connect to *server_name*

For example, to connect to a server named BLUE, enter:

```
connect to BLUE
```

Adaptive Server displays the following message:

```
Entered passthru mode to server 'BLUE'
```

Starting and Stopping the Full-Text Search Engine

The methods for starting the Full-Text Search engine from the command line are different for UNIX and Windows NT. Note that at the end of the **srvbuild** session (used only on UNIX platforms), the Full-Text Search engine is running.

Starting the Full-Text Search Engine on UNIX

Use the **startserver** utility to start the Full-Text Search engine. The **startserver** utility is included in the *bin* directory of Adaptive Server. For example, to start a Full-Text Search engine named KRAZYKAT, enter:

```
startserver -f /$SYBASE/install/RUN_KRAZYKAT
```

where the **-f** flag specifies the relative path to the runserver file. After you issue the command, the Full-Text Search engine issues a series of messages describing the settings in the configuration file:

Shutting Down the Full-Text Search Engine

Use the following command to shut down the Full-Text Search engine from Adaptive Server or OmniConnect™:

```
server_name...sp_shutdown
```

where *server_name* is the name of the Full-Text Search engine you are shutting down.

For example, to shut-down a Full-Text Search engine named KRAZYKAT, enter:

```
KRAZYKAT...sp_shutdown
```

Manually Editing the Configuration File

For UNIX, the **srvbuild** utility creates a configuration file when it builds the Full-Text Search engine; however, it lists only the required configuration file parameters. For Windows NT, you must manually edit the default configuration file. See “Editing the Configuration File” on page 3-6 for information about editing the configuration file for Windows NT.

Table 5-3 on page 5-12 lists all the available configuration file parameters. A sample configuration file that includes all of these

parameters is copied to your installation directory during installation. The sample configuration file is named *textsvr.cfg*. If the configuration file requires more extensive editing than that offered by **srvbuild**, you can make a copy of *textsvr.cfg* and edit it as described in the following sections to better suit your site's environment.

➤ ***Note***

The entire sample configuration file is listed in Appendix B, "Sample Files."

Renaming the *textsvr.cfg* File

The Full-Text Search engine is shipped with a configuration file named *textsvr.cfg*. The syntax for naming the configuration file is *server_name.cfg*, where *server_name* is the name of the Full-Text Search engine you are installing. If your Full-Text Search engine is not named "textsvr," make a copy of the sample configuration file using the name of the Full-Text Search engine as the prefix. For example, a Full-Text Search engine named KRAZYKAT would have a configuration file named *KRAZYKAT.cfg*.

Changing the Name of the Search Engine in the Configuration File

The default configuration file is for a Full-Text Search engine named "textsvr". If your Full-Text Search engine is named something other than "textsvr", you will have to edit the configuration file to reflect this. The configuration file names the Full-Text Search engine as the first entry in the values section of the file:

```
[textsvr]
```

For example, if you named your Full-Text Search engine KRAZYKAT, you would edit the configuration file to read:

```
[KRAZYKAT]
```

Setting the Locales

The following sections describe how to set the default language and character set in the configuration file.

Setting the Default Language

The default language for Verity is set with the **vdkLanguage** parameter in the configuration file. By default, **vdkLanguage** is set to "english0". You can configure Verity to use a different default language. The following table lists the default locales available for the Verity languages:

Table 5-1: vdkLanguage configuration parameters

Language	Default Locale Name
English	english0
German	german0
French	french0
Dutch	dutch0
Italian	italian0
Spanish	spanish0
Swedish	swedish0

The files used for the Verity languages are in $*SYBASE/sds/text/verity/common*. To change the Verity language, include the following line in the configuration file:

```
vdkLanguage = verity_language
```

where *verity_language* is the new default Verity language. For more information about the Verity languages, see the Web site at:

http://www.verity.com

Setting the Default Character Set

The default character set for Verity is set with the **vdkCharset** parameter in the configuration file. By default, the **vdkCharset** parameter is set to cp 850. The files used for the Verity character sets are in *$SYBASE/sds/text/verity/common*. Table 5-2 describes the character sets you can use with Verity:

Table 5-2: Verity character sets

Character Set	Description
cp 850	Default

Table 5-2: Verity character sets

Character Set	Description
cp 437	IBM PC character set
1252	Windows code page for Western European languages
mac1	Macintosh roman

To change the Verity character set, include the following line in the configuration file:

```
vdkCharset = verity_character_set
```

were verity_character_set is the new default Verity character set. For more information about Verity character sets, see

http://www.verity.com

Editing Configuration File Parameters

Table 5-3 describes the parameters included in the configuration file for both UNIX and Windows NT. Use a text editor to edit individual values, or use the default values provided.

Table 5-3: Configuration file parameters

Parameter	Description	Default Value
batch_size	Determines the size of the batches sent to Full-Text Search engine	500
max_indexes	The maximum number of text indexes that will be created in the Full-Text Search engine.	126
max_stacksize	Size of the stack allocated for client threads (in kilobytes)	34816
max_threads	Maximum number of threads available for the Full-Text Search engine	50
max_packetsize	Packet size sent between the Full-Text Search engine and the Adaptive Server	2048
max_sessions	Maximum number of sessions for the Full-Text Search engine	100

Table 5-3: Configuration file parameters (continued)

Parameter	Description	Default Value
min_sessions	Minimum number of sessions for the Full-Text Search engine	10
language	Language used by the Full-Text Search engine	us_english
charset	Character set used by the Full-Text Search engine	iso_1
vdkCharset	Character set used by Verity Search '97	850
vdkLanguage	Language used by Verity Search '97	us_english
vdkHome	Verity directory	UNIX: *$SYBASE/sds/text/verity* Windows NT: *%SYBASE% \ sds \ text \ verity*
collDir	Storage location of the Full-Text Search engine's collection	UNIX: *$SYBASE/sds/text/collections* Windows NT: *%SYBASE% \ sds \ text \ collections*
default_Db	Name of the Full-Text Search engine database	*text_db*
interfaces	Full path to the directory in which the interfaces file used by the Full-Text Search engine is located	UNIX: *$SYBASE/interfaces* Windows NT: *%SYBASE% \ ini \ sql.ini*
errorLog	Full path name to the error log file	The directory in which you start Full-Text Search engine
traceflags	String containing numeric identifiers used to generate diagnostic information	0
svr_traceflags	String containing numeric flag identifiers used to generate Open Server diagnostic information	0

Trace Flags

Table 5-4 describes the function of each Full-Text Search engine trace flag in the configuration file:

Table 5-4: Configuration file trace flags

Trace Flag	Description
1	Traces connects and disconnects and attention events from Adaptive Server.
2	Traces language events. Traces the SQL statement that ASE sent to the text server.
3	Traces RPC events.
4	Traces cursor events. Traces the SQL statement sent to the text server by Adaptive Server.
5	Writes the error that is sent to the user to the log.
6	Traces information about indexes. Writes the search string being passed to Verity to the log, and writes the number of records that the search returns to the log.
7	Traces done packets.
8	Traces calls to the interface between the Full-Text Search engine and the Verity API.
9	Traces SQL parsing.
10	Traces Verity processing.
11	Disables Verity collection optimization.
12	Disables **sp_statistics** from returning information.

Editing the Runserver File

This section describes the components of the runserver file and how to edit it to suit the needs of your site.

The runserver file contains start-up commands for the Full-Text Search engine. The runserver file can include the flags: shown in Table 5-5.

Table 5-5: Definition of flags in the runserver file

Flag	Definition
-S*server_name*	Specifies the name of the Full-Text Search engine and is used to locate the configuration file and the network connection information in the interfaces file.
-t	Causes the Full-Text Search engine to write start-up messages to standard error.
-l*errorlog_path*	Specifies the path to the error log file.
-i*interfaces_file_path*	Specifies the path to the interfaces file.

A sample runserver file is copied to the *$SYBASE/install* directory during installation. Make a copy of this file, renaming it *RUN_server_name*, where *server_name* is the name of the Full-Text Search engine. You must include the LD_LIBRARY_PATH environment variable in the runserver file. For example, the runserver file for a Full-Text Search engine named KRAZYKAT would be RUN_KRAZYKAT, and would be similar to:

```
#!/bin/sh
#
# Verity Text Server name:      KRAZYKAT
# Error log path:       /work/work/Gryphons/install/KRAZYKAT.log
# Interfaces file path: /work/work/Gryphons/interfaces
#
SYBASE=$SYBASE/sds/text; export SYBASE
LD_LIBRARY_PATH="$SYBASE/lib:$LD_LIBRARY_PATH"; export LD_LIBRARY_PATH
/work/work/Gryphons/sds/text/bin/txtsvr \
-SKRAZYKAT \
-t \
-l/work/work/Gryphons/install/KRAZYKAT.log \
```

The start-up command in the runserver file must consist of a single line and cannot include a return. If you have to carry the contents of the file over to a second or third line, include a backslash (\) for a line break (as in the example above).

Specifying a Sort Order

The sort order specifies the collating sequence used to order the data in the result set. You can set the default sort order, so that all queries

return a result set with the same sort order, or you can specify the sort order at the time you issue the query, by using the *sort_by* pseudo column.

Setting the Default Sort Order

By default, the Full-Text Search engine sorts the result set by the *score* pseudo column in descending order (the higher scores appear first). To change the default sort order, set the **sort_order** parameter in your configuration file to one of the values described in Table 5-6:

Table 5-6: Sort order values for the configuration file

Value	Description
0	Returns result sets sorted by the *score* pseudo column in descending order. The default value.
1	Returns result sets sorted by the *score* pseudo column in ascending order.
2	Returns result sets sorted by a timestamp in descending order.
3	Returns result sets sorted by a timestamp in ascending order.

When you sort a result set by descending timestamp (value 2 in Table 5-6), the Full-Text Search engine returns the newest documents first. The newest documents are those that were inserted or updated most recently. When results are sorted by ascending timestamp (value 3 in Table 5-6), the Full-Text Search engine returns the oldest documents first.

Setting the default sort order is especially important if your query uses the *max_docs* pseudo column. The *max_docs* pseudo column limits the number of rows of the result set to the first *n* rows, ordered by the sort order. If you set *max_docs* to a number smaller than the size of the result set, the sort order you select could exclude the rows that contain the information for which you are searching.

For example, if you sort by ascending timestamp, the latest document added to the table appears last in the result set. If the entire result set consists of 11 documents, and you set *max_docs* to 10, the latest document does not appear in the result set. However, if you sort by descending timestamp, it appears first in the result set.

Setting the Sort Order for Individual Queries

The *sort_by* pseudo column allows a query to return a result set with a sort order other than the default. Like all pseudo column settings, the *sort_by* pseudo column values are valid only for the duration of the query. Table 5-7 lists the values available for the *sort_by* pseudo column:

Table 5-7: Values for the sort_by pseudo column

Value	Description
fts_score desc	Returns a result set sorted in descending order.
fts_score asc	Returns a result set sorted in ascending order.
fts_timestamp desc	Returns a result set sorted by a timestamp in descending order.
fts_timestamp asc	Returns a result set sorted by a timestamp in ascending order.

Sorting by Defined Columns

The following values allow you to use the *sort_by* pseudo column to sort the result set by the data columns in the text index:

Table 5-8: Values of sort_by for sorting by columns

Value	Description
***column_name* desc**	Returns a result set sorted according to the descending order of a column.
***column_name* asc**	Returns a result set sorted according to the ascending order of a column.

Where *column_name* is the name of the data column.

Before you can sort by specific columns, you must modify the *style.vgw* and *style.ufl* files. Both files are in the *$SYBASE/sds/text/collections/db.owner.index/style* directory, where *db.owner.index* is the database, the database owner, and the index created using **sp_create_text_index**. For example, if you created a text index called *p_blurbs* on the *pubs2* database, the full path to those files would be similar to:

$SYBASE/sds/text/collections/pubs2.dbo.p_blurbs/style

Editing the style.vgw and style.ufl Files

To edit the *style.vgw* and *style.ufl* files, follow these steps:

1. Drop the text index that contains the columns for which you are adding definitions.

   For example, to add definitions for the *copy* column of the *blurbs* table, and create a text index named *p_blurbs*, use the following command:

   ```
   sp_drop_text_index p_blurbs
   ```

2. Edit the *style.vgw* file. After the following entry:

   ```
   dda "SybaseTextServer"
   ```

   add an entry for the column you are defining. The syntax for that definition is:

```
table: DOCUMENTS
   {
           copy: fcolumn_number copy_column_number
   }
```

   where *column_number* is the number of the column you are defining. Column numbers start with 0; if you want the first column to be sorted, specify “f0”; to sort the second column, specify “f1”; to sort the third column, specify “f2”, and so on.

   For example, to define the first column in a table, the syntax is:

```
table: DOCUMENTS
{
        copy: f0 copy_f0
}
```

   Then, your *style.vgw* file will be similar to this:

```
#
#       Sybase Text Server Gateway
#
$control: 1
gateway:
{
    dda:    "SybaseTextServer"
{
    copy: f0 copy_f0
}
}
```

3. Edit the *style.ufl* file, and add the column definition for a data table named *fts*. The syntax for the definition is:

```
data-table:    fts
   {
      fixwidth: copy_fcolumn_number precision datatype
   }
```

Column numbers start with 0; if you want the first column to be sorted, specify “f0”; to sort the second column, specify “f1”; to sort the third column, specify “f2”, and so on. For example, to add a definition for the first column of a table, with a precision of 4, and a datatype of *date*, enter:

```
data-table: fts
{
        fixwidth:   copy_f0 4   date
}
```

Similarly, to add a definition for the second column of a table with a precision of 10, and a datatype of *character*, enter:

```
data-table: fts
{
    fixwidth:   copy_f1 10  text
}
```

4. Re-create the index, using **sp_create_text_index**.

Enabling Summarization

The *summary* pseudo column allows you to specify that queries return only summaries of the documents that meet the search criteria, instead of entire documents. The *summary* pseudo column is not available by default; you must edit the *style.prm* file for the Full-Text Search engine to use this pseudo column. See “Configuring Summarization” on page 5-20 for information about enabling the *summary* pseudo column.

For example, the following query returns only summaries of documents that include the words “Iranian” and “book” (in this example, the *style.prm* file is configured to display 255 characters):

```
select t1.score, summary
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 70
and index_any like “<many>(Iranian and book)”
```

```
score summary
----- -----------------------------------------------------------
78    They asked me to write about myself and my book, so here
      goes: I started a restaurant called “de Gustibus” with two
      of my fri

(1 row affected)
```

The Full-Text Search engine supports summaries of up to 255 bytes in length.

Configuring Summarization

To enable summarization, edit the *style.prm* file. The *style.prm* file is located in the *$SYBASE/sds/text/collections/db.owner.index/style* directory, where *db.owner.index* is the database, the database owner, and the index created with **sp_create_text_index**. For example, if you created a text index called *p_blurbs* on the *pubs2* database, the full path to these files would be similar to:

$SYBASE/sds/text/collections/pubs2.dbo.p_blurbs/style

Perform the following steps to configure the Full-Text Search engine to use summarization:

1. Use **sp_drop_text_index** to drop the text index of the table that contains the documents you are summarizing.

   For example, if you are summarizing documents from the *blurbs* table of the *pubs2* database, and you have a text index named *p_blurbs*, enter:

   ```
   sp_drop_text_index “blurbs.p_blurbs”
   ```

2. Edit the *style.prm* file to configure Full-Text Search engine for summarization. The following lines in *style.prm* include information about summarization:

```
# The example below stores the best three sentences of
# the document, but not more than 255 bytes.
#$define DOC-SUMMARIES   "XS MaxSents 3 MaxBytes 255”

# The example below stores the first four sentences of
# the document, but not more than 255 bytes.
#$define DOC-SUMMARIES   "LS MaxSents 4 MaxBytes 255"

# The example below stores the first 150 bytes of
# the document, with whitespace compressed.
#$define DOC-SUMMARIES   "LB MaxBytes 150"
```

Un-comment one of the lines that starts with “#$define”. Each of these lines reflects a different level of summarization. You can

specify how many bytes of data you want the Full-Text Search engine to display, by altering the numbers at the ends of these lines. For example, if you want only the first 233 bytes of data summarized, edit the script to read:

```
# The example below stores the first four sentences of
# the document, but not more than 500 bytes.
$define DOC-SUMMARIES   "LS MaxSents 4 MaxBytes 233"
```

The maximum number of bytes displayed is 255. Any number greater than that is truncated to 255.

3. Re-create the text index you dropped in step 1. For example, to re-create the *p_blurbs* text index, enter something similar to the following:

```
sp_create_text_index "KRAZYKAT", "p_blurbs", "blurbs", "format
character", "copy"
```

Configuring Summarization for All Tables

You can configure the Full-Text Search engine so that all tables for which you create text indexes allow summarization. The *style.prm* file in the *$SYBASE/sds/text/verity/common/style* directory configures the Full-Text Search engine style parameters. Edit the *style.prm* file in that directory, following the steps in “Configuring Summarization” on page 5-20.

6 Full-Text Search Engine Commands

Full-Text Search Operators

This chapter describes using the Verity query language (which uses commands called **operators**) included with the Full-Text Search engine and provides examples for using these operators in queries.

For more information about Verity operators, see:

http://www.verity.com

Table 6-1 describes the operators Verity provides for performing full-text searches:

Table 6-1: Operators

Operator Name	Description
accrue	Selects documents that include at least one of the search elements you specify. The more search elements that are present, the higher the score will be.
and	Selects documents that contain all the search elements you specify.
complement	Returns the complement of the score value (the *score* value subtracted from 100).
in	Select documents that contain the search criteria in a specified column.
near	Selects documents containing specified search terms, where the closer the search terms are within a document, the higher the document's score.
near/n	Selects documents containing two or more search terms within *N* number of words of each other, where *N* is an integer up to 1000. The closer the search terms are within a document, the higher the document's score.
or	Selects documents that contain at least one of the search elements you specify.
paragraph	Selects documents that include all the search elements you specify within the same paragraph.
phrase	Selects documents that include a particular phrase. A phrase is a grouping of two or more words that occur in a specific order.
product	Multiplies the score values for each of the items of the search criteria.

Table 6-1: Operators

Operator Name	Description
sentence	Selects documents that include all the words you specify within the same sentence.
stem	Expands the search to include the specified word and its variations.
sum	Adds the score values for all items in the search criteria.
thesaurus	Expands the search to include the word specified and its synonyms.
wildcard	Matches wildcard characters included in search strings. Certain characters automatically indicate a wildcard specification.
word	Performs a basic word search, selecting documents that include one or more instances of the specified word.
yesno	Converts all nonzero score values to 100.

You **must** place the operators in angle brackets (<>) in the query. If they are not included in angle brackets, Adaptive Server issues error messages similar to the following:

```
Msg 20200, Level 15, State 0:
Server 'KRAZYKAT', Line 1:
Error   E1-0111 (Query Builder): Syntax error in query string near
character 5
Msg 20200, Level 15, State 0:
Server 'KRAZYKAT', Line 1:
Error   E1-0114 (Query Builder): Error parsing query: word(tasmanian)
Msg 20101, Level 15, State 0:
Server 'KRAZYKAT', Line 1:
VdkSearchNew failed with vdk error (-40).
Msg 20101, Level 15, State 0:
Server 'KRAZYKAT', Line 1:
VdkSearchGetInfo failed with vdk error (-11).
score copy
----- ------------------------------------------------------------
(0 rows affected) score
```

Relevance-Ranking Search Results

Relevance ranking is the ability of the Full-Text Search engine to assign the *score* parameter a value that indicates how often the item you are searching for appears in each document. The more often the item appears in the document, the higher the *score* value is for that document. If you set the *score* value in your query very high (such as

90), you limit the result set to documents that have a *score* value greater than that number.

➤ ***Note***

Note that Verity uses decimals for *score* values, while Sybase uses whole numbers. Therefore, if Verity reports a score value of .85, Sybase reports the same *score* value as 85.

For example, the following query searches for documents that contain the word "raconteur" and have a *score* of 90 or greater:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 90
and index_any = "<accrue>(raconteur)"

score copy
----- ------------------------------------------------------------

(0 rows affected)
```

The query does not find any documents that contain the word "raconteur" and have a score above 90. However, if the *score* value in the query is lowered to 39, you find that one document in the *blurbs* table mentions the word "raconteur":

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 39
and index_any = "<accrue>(raconteur)"

score copy
----- ------------------------------------------------------------
40    A chef's chef and a raconteur's raconteur, Reginald
      Blotchet-Halls calls London his second home. "Th' palace
      . . .

      hunger for delicious back-stairs gossip by serving up
      tidbits and entrees literally fit for a king!

(1 row affected)
```

Not all operators relevance-rank the result set. You must use the **many** operator to relevance-rank the result set for a number of the operators listed in Table 6-1.

Operator Descriptions and Examples

The following sections describe the operators included with the Full-Text Search engine. The result sets have been cut for formatting purposes.

accrue

The **accrue** operator selects documents that contain at least one of the search items specified in the query. Each result is relevance-ranked. For example, the following query searches for the word “business” in the *blurbs* column:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 35
and index_any = "<accrue>(business)"
```

```
score copy
----- ------------------------------------------------------------
41    They asked me to write about myself and my book, so here
      goes:  I started a restaurant called "de Gustibus" with two
. . .

      crowds for us every day.  They will work for you, too.
      Period!

39    Bennet was the classic too-busy executive.  After
      discovering computer databases he now has the time to run
      . . .

      or kill you.  If you get the right one, you can be like
      me.  If you get the wrong one, watch out.  Read my book!”
```

and, *or*

The **and** and **or** operators select documents that contain the specified search elements. The **and** operator selects documents that contain all the elements specified in the query. For example the following query selects documents that contain both “Iranian” and “business”:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 70
and index_any = "<many>(Iranian and business)"

score copy
----- ------------------------------------------------------------
78    They asked me to write about myself and my book, so here
      goes:  I started a restaurant called “de Gustibus” with two
      . . .

      crowds for us every day.  They will work for you, too.
      Period!
```

The **or** operator selects the documents that contain any of the searched elements. For example, if the previous query is rewritten to use the **or** operator, the query returns two documents:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 70
and index_any = "<many>(Iranian or business)"

score copy
----- ------------------------------------------------------------
82    They asked me to write about myself and my book, so here
      goes:  I started a restaurant called "de Gustibus" with two
      . . .

      crowds for us every day.  They will work for you, too.
      Period!

78    Bennet was the classic too-busy executive.  After
      discovering computer databases he now has the time to run
      . . .

      or kill you.  If you get the right one, you can be like
      me.  If you get the wrong one, watch out.  Read my book!”
```

complement

The **complement** operator returns the complement of the *score* value for a document; that is, it subtracts the value of *score* from 100 and returns the result as the *score* value for the document.

in

The **in** operator limits your search to a particular column in your index. You can specify any column that you listed when you issued **sp_create_text_index**. For example, if you specified the *copy* column of

the *blurbs* table when you created the text index, you could issue the following:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 35
and index_any = "Iranian<in>copy"

score copy
----- ------------------------------------------------------------
78    They asked me to write about myself and my book, so here
      goes:  I started a restaurant called “de Gustibus” with two
      . . .
      crowds for us every day.  They will work for you, too.
      Period!
```

near, *near/n*

The **near** operator selects documents that contain the items specified in the query and are near each other (near being a relative term). The documents in which the search words appear closest to each other receive the highest relevance ranking. The **near/*n*** operator specifies how far apart the items can be (***n*** has a maximum value of 1000). The following example selects documents in which the words “raconteur” and “home” appear within 10 words of each other. (This example produces the same result whether you use the **near/*n*** operator or the **near** operator.)

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 50
and index_any = "<many><near/10>(raconteur, home)"

score copy
----- ------------------------------------------------------------
95    A chef’s chef and a raconteur’s raconteur, Reginald
      Blotchet-Halls calls London his second home. "Th’ palace
      . . .
      hunger for delicious back-stairs gossip by serving up
      tidbits and entrees literally fit for a king!
```

phrase

The **phrase** operator selects documents that contain a particular phrase (a group of two or more items that occur in a specific order). The following example selects the documents that contain the phrase “the gorilla’s head”:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 50
and index_any = "<many><phrase>(the gorilla's head)"
```

```
score copy
----- ------------------------------------------------------------
82    Albert Ringer was born in a trunk to circus parents, but
      another kind of circus trunk played a more important role
      . . .
      over the gorilla's head and would have landed head first on
      . . .
      remaining time to share what I learned out there.'  I owe
      it all to Nana!"
```

product

The **product** operator multiplies the *score* value for the documents for each of the items in the search criteria. To arrive at a document's *score*, the Full-Text Search engine calculates a *score* for each search element and multiplies the *scores* together. Note that the Verity operators use decimals for the *score* values; Adaptive Server converts these to whole numbers (.56 becomes 56 in the Full-Text Search engine). For example:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 50
and index_any = "<product>(cat, created)"
```

```
score copy
----- ------------------------------------------------------------
61    If Chastity Locksley didn't exist, this troubled world
      would have created her!  Not only did she master the mystic
      . . .
      pun, is the only civilized alternative to the gross
      etiquette often practiced on the public networks.
```

Individually, the *score* for each item is 78 ; however, because the *scores* for the items are multiplied, the document has a *score* of 61 (.78 x .78 = .61(100) = 61).

sentence, *paragraph*

The **sentence** and **paragraph** operators search for sequences of words within documents; the only difference between the two operators is the size of the area they search. The **sentence** operator requires that the search items appear in the same sentence; the **paragraph** operator requires that they appear in the same paragraph. The closer the

words are to each other in a sentence or paragraph, the higher the *score* the document receives in relevance ranking. The following example searches for documents in which the words "also" and "service" occur within the same sentence. For the *blurbs* table, the result set is the same if you substitute the **paragraph** operator for **sentence** in line four:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 50
and index_any = "<many><sentence>(also, service)"
score copy
----- ------------------------------------------------------------
82    Bennet was the classic too-busy executive.  After
      discovering computer databases he now has the time to run
      . . .
      Bennet also donates time to community service organizations.
      . . .
      or kill you.  If you get the right one, you can be like
      me.  If you get the wrong one, watch out.  Read my book!"
```

stem

The **stem** operator searches for documents containing the specified word and its variations. For example, if you specify the word "cook," the Full-Text Search engine produces any documents that contain "cooked," "cooking," "cooks," and so on. The following query uses the **stem** operator to find documents that contain any variations of the word "create", that is, words that contain the word "create" as a stem. Notice that even though the first document contains a word to which "create" is not a perfect stem ("creative"), it is still selected:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 10
and index_any = "<many><stem>create"
score copy
----- ------------------------------------------------------
78    Anne Ringer ran away from the circus as a child.  A
      university creative writing professor and her family
      . . .
      collaborative work, with Michel DeFrance, "The Gourmet
      Microwave."
```

```
78    If Chastity Locksley didn't exist, this troubled world
      would have created her!  Not only did she master the mystic
      . . .
      pun, is the only civilized alternative to the gross
      etiquette often practiced on the public networks.
```

sum

The **sum** operator totals the *score* values for each item in the search criteria, to a maximum of 100. To arrive at a document's *score*, the Full-Text Search engine calculates a *score* for each search element and totals these *scores* together.

thesaurus

The **thesaurus** operator searches for documents containing a synonym of a search item. For example you might perform a search using the word "dog" looking for documents that use any of its synonyms (canine, pooch, pup, watchdog, and so on). The following example uses the **thesaurus** operator to find a result set that contains synonyms for the word "crave." The first document is selected because it contains the word "want"; the second, because it contains the word "hunger":

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 70
and index_any = "<many><thesaurus>(crave)"
score copy
----- ------------------------------------------------------------
78    They asked me to write about myself and my book, so here
      goes:  I started a restaurant called "de Gustibus" with two
      . . .
      of restaurant over another, when what they really want is a
      . . .
      crowds for us every day.  They will work for you, too.
      Period!
78    A chef's chef and a raconteur's raconteur, Reginald
      Blotchet-Halls calls London his second home. "Th' palace
      . . .
      his equal skill in satisfying our perpetual hunger for
      . . .
      hunger for delicious back-stairs gossip by serving up
      tidbits and entrees literally fit for a king!
```

wildcard

The **wildcard** operator allows you to substitute wildcard characters for part of the item for which you are searching. Table 6-2 describes the different wildcard characters and their attributes:

Table 6-2: Full-Text Search engine wildcard characters

Character	Function
?	Specifies one alphanumeric character, as in '?an', which locates "ran," "pan," "can," and "ban." You do not need to include the **wildcard** operator when you include the question mark in your query. The question mark is ignored in a set ([ ]) or in an alternative pattern ({ }).
*	Specifies zero or more of any alphanumeric character, as in 'corp*', which locates "corporate," "corporation," "corporal," and "corpulent." You do not need to include the **wildcard** operator when you include the asterisk in your query; you should not use the asterisk to specify the first character of a wildcard character string. The asterisk is ignored in a set ([ ]) or in an alternative pattern ({ }).
[ ]	Specifies any single character in a set, as in <wildcard> 'c[auo]t', which locates "cat," "cut," and "cot." You must enclose the word that includes a set in backquotes (` `), and there can be no spaces in a set.
{ }	Specifies one of each pattern separated by a comma, as in <wildcard> 'bank{s,er,ing}', which locates "banks," "banker," and "banking." You must enclose the word that includes a pattern in backquotes (` `), and there can be no spaces in a set.
^	Specifies one of any character not in the set, as in <wildcard> 'st[^oa]ck', which excludes "stock" and "stack" but locates "stick" and "stuck." The caret (^) must be the first character after the left bracket ([) that introduces a set.
-	Specifies a range of characters in a set, as in <wildcard> 'c[a-r]t', which locates every three-letter word from "cat" to "crt."

For example, the following query searches for documents that include variations of the word "slingshot":

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score >50
and index_any = "`slingshot*`"
```

```
score copy
----- ------------------------------------------------------------
100   Albert Ringer was born in a trunk to circus parents, but
      another kind of circus trunk played a more important role
      . . .
      gorilla.  “Slingshotting” himself from the ring ropes,
      . . .
      remaining time to share what I learned out there.’  I owe
      it all to Nana!"
```

word

The **word** operator searches for documents containing a particular word. To relevance-rank the result set, include the **many** operator in the query. The following example searches the *blurbs* column for documents containing the word “palates”:

```
select t1.score, t2.copy
from p_blurbs t1, blurbs t2
where t1.id=t2.id and score > 50
and index_any = "<word>(palates)"
```

```
score copy
----- ------------------------------------------------------------
100   A chef’s chef and a raconteur’s raconteur, Reginald
      Blotchet-Halls calls London his second home. "Th’ palace
      . . .
      hunger for delicious back-stairs gossip by serving up
      tidbits and entrees literally fit for a king!
```

yesno

The **yesno** operator converts all nonzero score values to 100. For example, if the actual score values for five documents are 86, 45, 89, 89, and 100, all of those documents are returned with a *score* of 100. *score* values of 0 are not changed. The **yesno** operator is helpful for ensuring that all documents containing the search criteria are returned in the result set, regardless of the sort order.

Operator Modifiers

The Verity query language includes the modifiers shown in Table 6-3:

Table 6-3: Operator modifiers for Verity query language

Modifier Name	Description	Works with These Modifiers	Example
case	Performs case sensitive searches	**word** **wildcard**	`<case><word>(Net)`
many	Counts the density of words, stemmed words, or phrases in a document. Relevance-ranks the document according its density.	**word** **wildcard** **stem** **soundex** **phrase** **sentence** **paragraph**	`<many><stem>(write)`
not	Excludes documents that contain the items for which the query is searching.	**and** **or**	`cat<and><not>elephant`

order	Specifies that the items in the documents occur in the same order they appear in the query.
	Always place the **order** modifier just before the operator
		near/n
		paragraph
		sentence
			Simple syntax:
			`tidbits<order><paragraph>king`
			Explicit syntax:
			`<order><paragraph>(tidbits,king)`

A System Procedures

This chapter describes the Sybase-supplied system procedures used for updating and getting reports from system tables. Table A-1 lists the system procedures discussed in this appendix.

Table A-1: System procedures

Procedure	Description
sp_clean_text_events	Removes entries from the *event* table.
sp_clean_text_indexes	Cleans up stray indexes.
sp_create_text_index	Creates an external text index.
sp_drop_text_index	Drops text indexes.
sp_redo_text_events	Changes the status of entries in the *text_events* table to force the re-indexing of the modified table.
sp_refresh_text_index	Notifies the external text server of a modified text column.
sp_show_text_online	Displays information about databases or indexes that are currently online.
sp_text_notify	Notifies the Full-Text Search engine that the *events* table has been modified.
sp_text_online	Makes a database available to Adaptive Server.

sp_clean_text_events

Function

Removes entries from the *event* table.

Syntax

```
sp_clean_text_events [up_to_date]
```

Parameters

up_to_date – the date up to which all entries will be deleted.

Examples

1. `sp_clean_text_events "09/15/95"`

   Removes data entered on or before 09/15/95.

Comments

- This procedure removes all rows from the *event* table whose *status* column is set.
- If the *up_to_date* parameter is specified, all entries with a set *status* column having a date less than or equal to *up_to_date* will be deleted.
- If *up_to_date* is omitted, all entries with a set *status* column will be removed.
- Entries should be removed from the *event* table only after you have backed up the collection associated with the text index.

Messages

None

Permissions

Any user can execute **sp_clean_text_events**.

sp_clean_text_indexes

Function

Removes indexes from the *vesaux* table that are not associated with a table.

Syntax

```
sp_clean_text_indexes
```

Parameters

None.

Examples

```
1. sp_clean_text_indexes
```

Comments

- This procedure reads entries from the *vesaux* tables, verifying that both the source table and corresponding index table exist. If either is missing, the index is dropped.

Messages

- `Fetch resulted in an error`
- `Unable to drop objectdef for` *`index_name`*`!`

Permissions

Any user can execute **sp_clean_text_indexes.**

sp_create_text_index

Function

Creates an external text index.

Syntax

```
sp_create_text_index server_name, index_table_name,
   table_name, option_string, column_name
   [, column_name ... ]
```

Parameters

server_name – is the name of the Full-Text Search engine.

index_table_name – is the name of the index table and will be the name of the table mapped to the text index.

table_name – is the name of the table containing the text being indexed.

option_string – is a placeholder for future character – format definitions.

column_name – is the name of the column indexed by the text index.

Examples

1.
```
sp_create_text_index "blue", "p_blurbs", "blurbs",
" ", "copy"
```

   Creates an index table on the *blurbs* table of the *pubs2* database.

Comments

- Up to 16 columns can be indexed in a single text index.
- Columns of the following datatypes can be indexed:

  char, *varchar*, *nchar*, *nvarchar*, *text*, *image*, *datetime*, *smalldatetime*
- The content of *option_string* is not case sensitive.
- *option_string* uses a null string (“ ”) to specify “No Options”.
- **sp_create_text_index** writes entries to the *vesaux* table and tells the Full-Text Search engine to create the text index.
- Execution of **sp_create_text_index** is synchronous. The Adaptive Server process executing this system procedure remains blocked until the index is created. The time required to index large

amounts of data may be very great, taking as long as several hours to finish.

Messages

- Can't run sp_create_text_index from within a transaction
- '*column_name*' cannot be NULL.
- Column '*column_name*' does not exist in table '*table_name*'
- Index table mapping failed - Text Index creation aborted
- Invalid text index name - '*index_name*' already exists
- '*parameter*' is not in the current database
- Server name '*server_name*' does not exist in sysservers.
- '*table_name*' does not exist
- '*table_name*' is not a valid object name
- Table '*table_name*' does not have an identity column - text index creation aborted
- Text index creation failed
- User '*user_name*' is not a valid user in the database

Permissions

Any user can execute **sp_create_text_index.**

sp_drop_text_index

Function

Drops the text indexes.

Syntax

```
sp_drop_text_index  "table_name.index_name"
   [,"table_name.index_name"...]
```

Parameters

table_name.index_name – is the name of the text indexes to be dropped.

Examples

1. `sp_drop_text_index "blurb.id"`

   Drops the text index from the *blurbs* table.

Comments

- The **sp_drop_text_index** system procedure first issues an RPC to the Full-Text Search engine to delete the Verity collection. Then, it removes the associated entries from the *vesaux* and *vesauxcol* tables, drops the index table, and removes the index table object definition.
- Up to 255 indexes can be specified in a single **sp_drop_text_index** request.
- If *database* and *owner* are not specified, the current owner and database are used.

Messages

- `Can't run sp_drop_text_index from within a transaction.`
- `Index '`*`index_name`*`' is not a Text Index`
- `'`*`parameter_name`*`' is not a valid name`
- `Server name '`*`server_name`*`' does not exist in sysservers`
- `Unable to drop index table '`*`table_name`*`'. This table must be dropped manually`

- `User '`*`user_name`*`' is not a valid user in the 'database_name' database`
- `vs_drop_index failed with code '`*`code_name`*`'.`

Permissions

Any user can execute **sp_drop_text_index.**

sp_redo_text_events

Function

Changes the status of entries in the *event* table by forcing the re-indexing of the modified columns.

Syntax

```
sp_redo_text_events [from_date [,to_date]]
```

Parameters

from_date – is the starting date in a date range of entries to be modified.

to_date – is the ending date in the specified date range of the entries to be modified.

Examples

```
1. sp_redo_text_events "01/05/94", "04/12/97"
```

Re-indexes columns that were modified between 01/05/94 and 04/12/97.

Comments

- Resets the *event_status* column in the *events* table for all entries that have the *status* bit set. The Full-Text Search engine is notified that a re-index operation is required.
- Useful for synchronizing a text index after a recovery of the Verity collection from a backup.
- If *to_date* is omitted, the *status* column for all entries between *from_date* and the current date with a set *status* column will be reset.
- If both *from_date* and *to_date* are omitted, the *status* column for all entries in the event table with a set *status* column will be reset.

Messages

- *to_date* cannot be specified without *from_date*
- You have not specified the full range.

Permissions

Any user can execute **sp_redo_text_events.**

sp_refresh_text_index

Function

Stores information about modifications to an indexed column.

Syntax

sp_refresh_text_index *table_name*, *column_name*, *rowid*, *mod_type*

Parameters

table_name – is the name of the table being updated, in the format *owner.table_name*. Defaults to *dbo* if the owner is not specified.

column_name – is the name of the column being updated.

rowid – is the IDENTITY column value of the changed row.

mod_type – specifies the type of the change. Must be **insert**, **update**, or **delete**.

Examples

1. **sp_refresh_text_index "blurbs", "copy", 2.000000, "update", true**

   Sends notification to the Full-Text Search engine that you have updated the row in the *blurbs* table with an *id* of 2.000000.

Comments

- The user must maintain the consistency of the text index. You can create triggers that issue **sp_refresh_text_index** for non-*text* and non-*image* columns. However, because triggers are not fired for changes to *text* and *image* columns, all **writetext** operations on indexed columns should be followed by a **sp_refresh_text_index**.

Messages

- `Column 'column_name' does not exist in table 'table_name'`
- `Invalid mod_type specified (modification_type). Correct values: INSERT, UPDATE, DELETE`
- `Owner 'owner_name' does not exist`
- `Table 'table_name' does not exist`

- '*table_name*' is not a valid name.
- Text event table not found

Permissions

Any user can execute **sp_refresh_text_index**.

sp_show_text_online

Function

Displays information about databases or indexes that are currently online.

Syntax

```
sp_show_text_online server_name [,{INDEXES |
   DATABASES} ]
```

Parameters

server_name – is the name of theFull-Text Search engine to which the request is sent

INDEXES | DATABASES – is used to specify whether the request should contain data about online indexes or online databases. The default is INDEXES

Examples

```
1. exec sp_show_text_online textsvr
```

Displays all indexes that are currently online in the specified Full-Text Search engine

```
2. exec sp_show_text_online textsvr, DATABASES
```

Displays all databases that are currently online in the Full-Text Search engine.

Comments

sp_show_text_indexes issues a Remote Procedure Call (RPC) to the Full-Text Search engine to retrieve information about the indexes or database currently online

If a database is not listed in the results of this procedure, use **sp_text_online** to bring the desired database online.

Messages

- **sp_show** failed for server %1!.
- The parameter value '*value*' is invalid
- The RPC sent to the server returned a failure return code

- The second parameter was neither INDEXES or DATABASES

Permissions

Any user can execute **sp_show_text_indexes**

See Also

sp_text_online

sp_text_notify

Function

Notifies the Full-Text Search engine that the *events* table has been modified.

Syntax

```
sp_text_notify {true | false},
   {Full_Text_Search_Engine_name}
```

Parameters

true – causes the procedure to run synchronously.

false – causes the procedure to run asynchronously.

Full_Text_Search_Engine_name – name of the Full-Text Search engine you are notifying.

Examples

```
1. sp_text_notify true
```

Comments

- This system procedure must be run after you issue **sp_refresh_text_index** to inform the Full-Text Search engine that you have modified the text tables.
- If you do not specify any parameters, **sp_text_notify** runs synchronously.

Messages

- `Can't run sp_text_notify from within a transaction`
- `Notification failed, server = 'server_name'`
- `Server name 'server_name' does not exist in sysservers`
- `The parameter value 'value' is invalid`

Permissions

Any user can execute **sp_text_notify.**

sp_text_online

function

Makes a database available to Adaptive Server.

Syntax

sp_text_online {*server_name*}, {*database_name*}

Parameters

server_name – name of the Full-Text Search engine.

database_name – name of the database that you are bringing online.

Examples

1. sp_text_online pubs2

Makes the *pubs2* database available for full-text searches using the Full-Text Search engine.

Comments

- If a database is not specified, all databases are brought online.
- If a server name is not specified, all the Full-Text Search engines listed in *vesaux* are notified.

Messages

- `All Databases using text indexes are now online`
- `Databases containing text indexes on server 'database_names' are now online`
- `Database 'database_name' is now online"`
- `Server name 'server_name' does not exist in sysservers.`
- `The parameter value 'value' is invalid`
- `The specified database does not exist`
- `vs_online failed for server 'server_name'`

Permissions

Any user can execute **sp_text_online**.

B Sample Files

This appendix contains the text of the default configuration file (*textsvr.cfg*), the *style.vgw* file. and the *style.ufl* file. It also discusses the **text_sample** script.

Default *textsvr.cfg* Configuration File

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  @(#)  File: textsvr.cfg  1.10 11/21/97
;
;  Standard Full Text Search Specialty Data Store -- 11.5
;             Sample Configuration File
;
;  The installation procedure places this file in the
;  “SYBASE” directory.
;
;  Lines with a semi-colon in column 1 are comment lines.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;             copyright (c) Sybase, Inc.
;             Emeryville, CA 1997.
;             All rights reserved.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;                                DIRECTIONS
;
;  Modifying the textsvr.cfg file:
;  ------------------------------
;  An installation can run the Text Search SDS product
;  as supplied, with no modifications to configuration
;  parameters. Default values from the executable program
;  are in effect.
;
;  The "textsvr.cfg"" file is supplied with all configuration
;  parameters commented out.
;
;  The hierarchy for setting configuration values is:
;
;   default value internal to the executable program (lowest)
;   configuration file value  (overrides default value)
;   command line argument     (overrides default value and *.cfgfile)
;
;  Command line arguments are available to override
;  settings for these options:
;
```

```
;     -i<file specification for interfaces file>
;     -l<file specification for log file>
;     -t (no arg) directs text server to write start-up
; information to stderr (default is DO NOT write start-up information)
;
;  To set configuration file parameters, follow these steps:
;
;  (1) If changing the server name to other than "textsvr":
;      (1A)  Copy “textsvr.cfg” to "your_server_name.cfg"
;            Example: server11.cfg
;      (1B)  Modify the [textsvr] line to [your_server_name]
;            Example: [server11]
;      The maximum length of "your_server_name" is 30 characters.
;
;  (2) Set any configuration values in the CONFIG VALUES SECTION below.
;      Remove the semi-colon from column 1.
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;               DEFINITIONS OF TRACE FLAG AND SORT ORDER VALUES
;
; "traceflags" parameter, for text server
;  Available "traceflags" values: 1,2,3,4,5,6,7,8,9,10,11,12
;
;  1  trace connect/disconnect/attention events
;  2  trace language events
;  3  trace rpc events
;  4  trace cursor events
;  5  log error messages returned to the client
;  6  trace information about indexes
;  7  trace senddone packets
;  8  write text server/Verity api interface records to the log
;  9  trace sql parser
; 10  trace Verity processing
; 11  disable Verity collection optimization
; 12  disable returning of sp_statistics information
;
; "srv_traceflags" parameter, for Open Server component of text server
; Available "srv_traceflags" values: 1,2,3,4,5,6,7,8
;  1  trace TDS headers
;  2  trace TDS data
;  3  trace attention events
;  4  trace message queues
;  5  trace TDS tokens
;  6  trace open server events
;  7  trace deferred event queue
;  8  trace network requests
;
; "sort_order" parameter
; Available "sort_order" values: 0,1,2,3
;  0  order by score, descending (default)
;  1  order by score, ascending
```

```
;  2  order by timestamp, descending
;  3  order by timestamp, ascending
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;                       CONFIG VALUES SECTION
;
; The "textsvr.cfg" file is supplied with the values commented out.
; To override value(s) in the executable program:
;    - Set required value(s) below
;    - Remove the semicolon from column 1
;
[textsvr]
;min_sessions = 10
;max_sessions = 100
;batch_size = 500
;sort_order = 0
;defaultDb = text_db
;errorLog = textsvr.log
;language = us_english
;charset = iso_1
;vdkLanguage = english0
;vdkCharset = 850
;traceflags = 0
;srv_traceflags = 0
;max_indexes = 126
;max_packetsize = 2048
;max_stacksize = 34816
;max_threads = 50
;collDir = <$SYBASE location on UNIX>/sds/text/collections
;collDir = <%SYBASE% location on Win-NT>\sds\text\collections
;vdkHome = <$SYBASE location on UNIX>/sds/text/verity
;vdkHome = <%SYBASE% location on Win-NT>\sds\text\verity
;interfaces = <$SYBASE location on UNIX>/interfaces
;interfaces = <%SYBASE% location on Win-NT>\ini\sql.ini
```

The *style.vgw* File

```
#
#       Sybase Text Server Gateway
#

$control: 1
gateway:
{
dda:    "SybaseTextServer"

#
# Uncomment the following lines if you want to sort by an index column.
# Change the copy statement to correspond to the index column that you
```

```
# want to sort by. Index columns are named f0 - ff, and must correspond
# to copy_f0 - copy_ff.
#

#    table: DOCUMENTS
#    {
#       copy: f0 copy_f0
#    }

}
```

The *style.ufl* File

```
# $Id: style.ufl,v 1.7 1997/04/01 01:32:47 edwin Exp $
# Copyright (C) 1987-1996 Verity, Inc.
#
# style.ufl - Application-specific User Fields
#
# These fields are included in the internal documents table. For
# more information about adding fields to the internal documents
# table, see the "Defining Custom Fields" chapter in the
# Collection Building Guide.
#
# Example:
#
# data-table:   ddf
# {
#   varwidth: MyTitle         dxa
# }

# ------------------------------------------------------------------
# Specify additional application-specific fields here in their own
# data-table[s].

#
# If a copy statement was added to the style.vgw file,
# uncomment the sts data table definition and change the
# line describing copy_f0 to describe that copy statement
# that was added to the style.vgw file.
# The following example describes a datetime column.
#

# data-table:   sts
# {
#   fixwidth: copy_f0 4 date
# }
```

The *text_sample* Script

The installation of the Full-Text Search engine copies the **text_sample** script to the *$SYBASE/sds/text/scripts* directory. This script provides examples of how to build tables, insert data, build a text index, and perform queries on this data using the Full-Text Search engine. After it is finished, the **text_sample** script deletes the data and removes all the tables it creates. You **do not** need to run this script. Sybase supplies it to provide examples of how full-text searches are run.

Before you run the **text_sample** script, you must first edit it for your environment. By default, the **text_sample** script creates a table named *demo_table* in a database called *demo_db*. If you do not have a database named *demo_db*, change the name of the database in this script to match the name of a database on which you have created a text index. Note that the database used by this script must have the *text_events* table installed. Do not use *text_db* (created by the **installtextserver** script) as the database for the **text_sample** script; that database should be reserved for the information required of your permanent text indexes.

The database in which you are running the **text_sample** script (named *demo_db* in the script) must have **select into/bulkcopy/pllsort** turned on in the database. See the *Adaptive Server Reference Manual* for more information about using the **sp_dboption** command to turn on **select into/bulkcopy/pllsort**.

Use **isql** to run the **text_sample** script. For example, to run the **text_sample** script on an Adaptive Server named IGNATZ:

```
isql -Ulogin -Ppassword -SIGNATZ -i $SYBASE/sds/text/scripts/text_sample
```

Index

E

F

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y