Full-Text Search Specialty Data Store User's Guide

Full-Text Search SDS Version 12.x
Document ID: 36521-01-1200-02
Last Revised:October 15, 2000

Principal authors: Lori Johnson and Jim Cluett

Contributing authors: Martin Ash, Vic Mesenzeff, Bill Seiger

Document ID: 36521-01-1200

This publication pertains to Full-Text Search SDS Version 12.x of the Sybase database management software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

Document Orders

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

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor.

Upgrades are provided only at regularly scheduled software release dates.

Sybase Trademarks

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, 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 Monitor, Adaptive Server IQ, 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, 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-

Restricted Rights

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

Table of Contents

4. Configuring Adaptive Server for Full-Text Searches

5. Setting Up Verity Functions

6. Writing Full-Text Search Queries

7. System Administration

8. Performance and Tuning

9. Verity Topics

A. System Procedures

List of Figures

List of Tables

About This Book

This book explains how to use the Full-Text Search Specialty Data Store product with Sybase® Adaptive Server™ Enterprise. Although this book refers to Adaptive Server throughout, the instructions for using it with OmniConnect™ are the same.

There are two versions of the Full-Text Search Specialty Data Store:

- The Standard version is included with your purchase of Adaptive Server Enterprise
- The Enhanced version is purchased separately and has additional capabilities

This book describes the features and functionality of both versions.

Audience

This book is for System Administrators who are configuring Adaptive Server for a Full-Text Search Specialty Data Store and for users who are performing full-text searches on Adaptive Server data.

How to Use This Book

This book includes the following chapters:

- Chapter 1, “Introduction,” provides an overview of Full-Text Search Specialty Data Store.
- Chapter 2, “Understanding the Full-Text Search Engine,” describes the components of the Full-Text Search Specialty Data Store and how it works.
- Chapter 3, “Installation,” provides step-by-step installation instructions.
- Chapter 4, “Configuring Adaptive Server for Full-Text Searches,” describes how to configure Adaptive Server so that Full-Text Search Specialty Data Store can perform full-text searches on the Adaptive Server databases.
- Chapter 5, “Setting Up Verity Functions,” describes the setup you need to do before issuing full-text search queries.
- Chapter 6, “Writing Full-Text Search Queries,” describes the components you use to write full-text search queries.

- Chapter 7, “System Administration,” provides information about system administration issues.
- Chapter 8, “Performance and Tuning,” provides information about performance and tuning issues.
- Chapter 9, “Verity Topics,” provides information about configuring the Verity engine.
- Appendix A, “System Procedures,” describes Full-Text Search Specialty Data Store system procedures.
- Appendix B, “Sample Files,” contains the text of the *textsvr.cfg* file, describes the sample files included with Full-Text Search Specialty Data Store, and discusses issues regarding the **sample_text_main.sql** script.
- Appendix C, “Unicode Support,” describes how to configure Full-Text Search Specialty Data Store to use Unicode.

Adaptive Server Enterprise Documents

The following documents comprise the Sybase Adaptive Server Enterprise documentation:

- The *Installation and Release Bulletin* for your platform – contains instructions on how to installthe Enhanced Full-Text Search Specialty Data Store, how to configure a Full-Text Search engine, and last-minute information that was too late to be included in the books.

  A more recent version of the *Installation and 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 Sybase Technical Library on the Web.
- The Adaptive Server installation documentation for your platform – describes installation and upgrade procedures for all Adaptive Server and related Sybase products.
- The Adaptive Server configuration documentation for your platform – describes configuring a server, creating network connections, configuring for optional functionality, such as auditing, installing most optional system databases, and performing operating system administration tasks.
- *Transact-SQL User’s Guide* – documents Transact-SQL®, Sybase’s enhanced version of the relational database language. This

manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the *pubs2* and *pubs3* sample databases.

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

Other Sources of Information

Use the Sybase Technical Library CD and the Technical Library Web site to learn more about your product:

- Technical Library CD contains product manuals and technical documents and is included with your software. The DynaText browser (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting Technical Library.

- Technical Library Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser.

  To use the Technical Library Web site, go to www.sybase.com and choose Documentation, choose Technical Library, then choose Product Manuals.

Conventions

Directory Paths

For readability, directory paths in this manual are in UNIX format. On Windows NT, substitute *$SYBASE* with *%SYBASE%* and replace slashes (/) with backslashes (\). For example, replace this user input:

```
$SYBASE/$SYBASE_FTS/scripts
```

with:

```
%SYBASE%\%SYBASE_FTS%\scripts
```

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.

Table 1: Syntax statement conventions (continued)

Key	Definition
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 Full-Text Search Specialty Data Store?

Full-Text Search Specialty Data Store (referred to in this book as the Full-Text Search engine) is an Open Server™ application built on Verity® technology available in the Verity Developer's Kit. Adaptive Server connects to the Full-Text Search engine through Component Integration Services (CIS), allowing queries written in the Verity query language to perform full-text searches on Adaptive Server data.

There are two versions of the Full-Text Search Specialty Data Store:

- The Standard version is included with your purchase of Adaptive Server Enterprise
- The Enhanced version is purchased separately and has additional capabilities

This book describes the features and functionality of both versions.

Capabilities of the Full-Text Search Engine

The Full-Text Search Specialty Data Store product performs powerful, full-text searches on Adaptive Server data. In Adaptive Server, without the Full-Text Search engine, you can 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.

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 “dogs,” and it returns documents that contain the words “dogs,” “canine,” “pooch,” “pup,” and so on.

Capabilities of the Enhanced Full-Text Search Engine

In addition to the Full-Text Search engine capabilities described above, the Enhanced Full-Text Search engine provides additional functionality that allows you to refine your search. Using Enhanced Full-Text Search engine, you can:

- Create your own custom thesaurus. For example, you can create a custom thesaurus that includes “working dogs,” “St. Bernard,” “large dogs,” and “European Breeds” as synonyms for “Saint Bernard.”
- Create topics that specify the search criteria for a query. For example, you can create a topic that returns documents that include the phrase “Saint Bernard” or “St. Bernard,” followed by documents that include the phrase “working dogs,” “large dogs,” or “European Breeds.”
- Return documents grouped in clusters to give you a sense of the major topics covered in the documents.
- Select a section of relevant text in a document and search for other, similar documents.
- Index many different document types, such as Microsoft Word, and FrameMaker. The Standard Full-Text Search engine allows you to index only SGML and HTML documents.
- Sort documents using up to 16 sort orders. The Standard Full-Text Search engine allows only a single sort order.

Enhanced Full-Text Search engine also provides additional system administration features such as:

- Integrated backup and restore capabilities
- Ability to change the value of a configuration parameter using a system procedure

- Ability to optimize indexes for text searches when your server is inactive, to enhance performance
- Additional system management reports for viewing setup information
- Ability to bring databases online automatically for text searches

2 Understanding the Full-Text Search Engine

This chapter describes how a Full-Text Search engine works. Topics include:

Components of the Full-Text Search Engine

The Full-Text Search engine uses the following components to provide full-text search capabilities:

- Source table
- Verity collections (text index)
- Filters for a variety of document types
- *text_db* database
- Index table
- *text_events* table

The Source Table

The **source table** is a user table maintained by Adaptive Server. It contains one or more columns using the *text*, *image*, *char*, *varchar*, *datetime*, or *small datetime* datatype, which holds the data to be searched. With the Enhanced Full-Text Search engine, the source table can also have *int*, *smallint*, and *tinyint* columns, which holds the data to be searched. The source table must have an IDENTITY column, which is used to join the source table with the *id* column of an index table during text searches.

The source table can be a local table, which holds the actual data, or it can be a proxy table that is mapped to remote data using CIS.

The Verity Collections

The Full-Text Search engine uses the Verity collections, which are located in *$SYBASE/$SYBASE_FTS/collections*. When you create the text indexes, as described in "Creating the Text Index and Index Table" on page 4-7, 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 Verity Web site:

http://www.verity.com

Filters

The text index uses a filter to strip out the tags in a document that is not ASCII text. The Standard Full-Text Search engine provides filtering for SGML and HTML documents. The Enhanced Full-Text Search engine provides additional filters for a variety of document types (Microsoft Word, FrameMaker, WordPerfect, SGML, and HTML).

The *text_db* Database

During the installation of the Full-Text Search engine, a database named *text_db* is added to Adaptive Server using the installation script **installtextserver**, as described in "Running the installtextserver Script" on page 4-2. The database does not contain any user data, but contains two support tables: *vesaux* and *vesauxcol*. These tables contain the metadata used by the Full-Text Search engine to maintain integrity between the Adaptive Server source tables and the Verity collections.

When updating the collections 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 are updated. The Full-Text Search engine also uses these tables when it is brought online, to make sure that all necessary collections exist.

The *vesaux* Table

The columns in the *vesaux* table are described in Table 2-1.

Table 2-1: Columns in the vesaux table

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

The *vesauxcol* Table

The columns in the *vesauxcol* table are described in Table 2-2.

Table 2-2: 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;* with the Enhanced Full-Text Search engine, also *int, smallint,* and *tinyint*)

The *Index* Table

The **index table** provides a means of locating and searching documents stored in the source table. The *index* table is maintained by the Full-Text Search engine and has an *id* 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 proxy

table to Adaptive Server through the Component Integration Services feature.

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 you use the *score* pseudo column in a query, to rank each document according to how well the document matches the 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 does not 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.

You use the *score* column and the other pseudo columns, *id*, *index_any*, *sort_by*, *summary*, and *max_docs*, to specify the parameters to include in your search. For a description of the pseudo columns, see "Pseudo Columns in the Index Table" on page 6-2.

The *text_events* Table

Each database containing tables for which there is a text index must contain an **events table**, which logs inserts, updates, and deletes to indexed columns. The name of this table is *text_events*. It is used to propagate updated data to the Verity collections.

The columns in the *text_events* table are described in Table 2-3.

Table 2-3: Columns in the *text_events* table

Column Name	Description
event_id	IDENTITY column.
id	ID of the row that was updated, inserted, or deleted.
tableid	Name of the table that contains the row that was updated, inserted, or deleted.

Table 2-3: Columns in the *text_events* table (continued)

Column Name	Description
columnid	Name of the column that the *text* index was created on.
event_date	Date and time of the **update**, **insert**, or **delete**.
event_type	Type of update (**update**, **insert**, or **delete**).
event_status	Indicates whether the **update**, **insert**, or **delete** has been propagated to the collections. Event Unread = 0. Event Read = 1. Event Succeeded = 2. Event Failed = 3.
srvid	Server ID of the Full-Text Search engine maintaining the collection.

Relationships Between the Components

The relationships between the Full-Text Search engine components are shown in Figure 2-1.

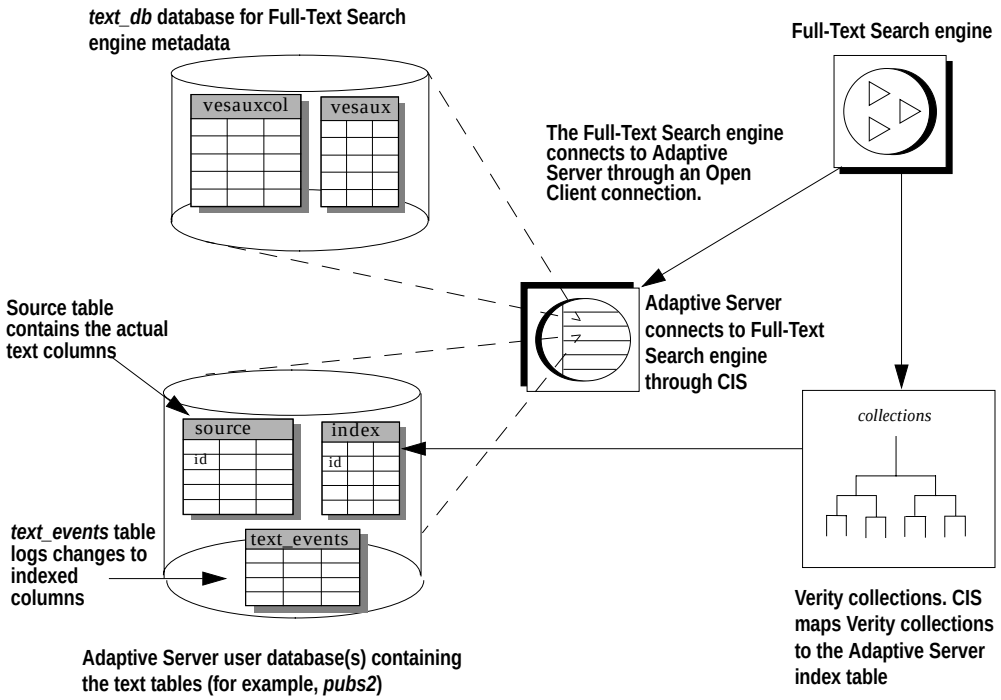


Figure 2-1: Components of the Full-Text Search engine

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 *id* 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" (the *i_blurbs* table is the index table):

```
select t1.score, t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id and t1.score > 20
and t1.max_docs = 10
and t1.index_any = "<near>(Greek, Gustibus)"
```

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

1. The Full-Text Search engine processes the query:

   ```
   select t1.score, t1.id
   from i_blurbs t1
   where t1.score > 20
   and t1.max_docs = 10
   and t1.index_any = "<near>(Greek, Gustibus)"
   ```

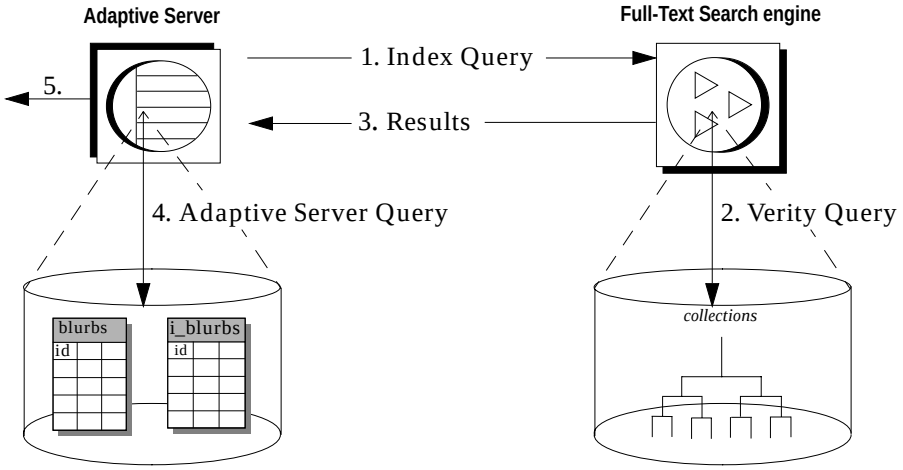
   The **select** statement includes the Verity operator **near** and the pseudo columns *score*, *max_docs*, and *index_any*. The operator and pseudo columns provide the parameters for the search on the Verity collections—they narrow the result set from the entire *copy* column to the 10 documents in which the words "Greek" and "Gustibus" appear closest to each other.

2. Adaptive Server processes the following **select** statement on the result set that is returned by the Full-Text Search engine in step 1:

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

   This joins the *blurbs* and *i_blurbs* tables (the source table and the index table, respectively) on the IDENTITY column of the *blurbs* table and the *id* column of the *i_blurbs* table.

Figure 2-2 describes how Adaptive Server and the Full-Text Search engine process the query.



1. Adaptive Server sends the index query to the Full-Text Search engine.
2. The Full-Text Search engine processes the Verity operators in the query and produces a result set from the collections.
3. The Full-Text Search engine returns the result set to Adaptive Server.
4. Adaptive Server processes the select statement on the local table.
5. Adaptive Server displays the results of the query.

Figure 2-2: Processing a full-text search query

3 Installation

This chapter explains how to install and configure Enhanced Full-Text Search.

There are three steps required to install the text server.

- Install the product using the Studio Installer
- Configure and start the text server
- Configure ASE for the text server (This is described in Chapter 4, “Configuring Adaptive Server for Full-Text Searches.”

Installing Enhanced Full-Text Search on Windows NT

Use the Studio Installer to install the Enhanced Full-Text Search engine.

The Studio Installer is the Sybase installation utility with a graphical user interface that creates the target directory (if necessary), sets Sybase environment variables, collects licensing certificate information, and performs the basic configuration.

The Enhanced full-Text Search should be installed into an existing Sybase directory structure that includes Adaptive Server Enterprise 12.0 or greater.

1. Log in to your Windows NT computer using an account with administrator privileges.
2. Insert the CD in the CD-ROM drive.
3. The Studio Installer should start automatically. If it does not:

   Select Start | Run, and type:

   `x:\setup.exe`

   where x: is your CD-ROM drive.

   Or, choose the Windows Explorer, select the CD-ROM drive, and double-click **setup.exe**.
4. Select the type of installation to be performed.
   - Standard Install - installs the default components a user needs.
   - Full Install - installs every component on the CD.

➤ ***Note***

In this release only the enhanced version will be installed regardless of the selection at this step.

5. The Studio Installer displays the default installation directory. Accept the default, or enter a different directory, then click Next.
6. A Summary screen displays a list of applications, and services that will be installed, the disk space required, and the disk space available. If the target directory does not have enough free space, the information appears in red.

   Click Next.

◆ ***WARNING!***

If you have insufficient disk space and click Next, an error occurs that stops the installation.

7. If the target installation directory does not already exist, the Studio Installer prompts: "OK to create directory?"

   Click Yes.

◆ ***WARNING!***

If you are prompted to overwrite any DLLs, select Yes only if the DLL version provided on the Sybase CD is more recent than the one currently installed on the system. The greater number is the most recent version.

8. A status bar shows the progress of the installation. This step may take quite a while.

➤ ***Note***

If you encounter problems during the installation, see the installation log file to see a record of the installation process.

%SYBASE%\Installer.log

`Where %SYBASE% is the SYBASE target installation directory.`

9. When prompted to reboot the computer, leave the CD in the CD-ROM drive and click Yes.

After rebooting, the computer accesses the CD for additional data. For this reason, the CD must remain in the CD-ROM drive during reboot, and the CD-ROM drive must be on the installation machine.

After installing Full-Text Search, you must configure the text server (Full-Text Search) and install it into ASE.

Configure Enhanced Full-Text Search for Windows NT

To configure the full-Text Search engine you must:

- Set the SYBASE environment variable
- Set the SYBASE_FTS environment variable
- Add the Full-Text Search engine to the PATH environment variable
- Add entries to the interfaces file (*sql.ini*)
- Start the Full-Text Search engine

Set the SYBASE Environment Variable

The SYBASE environment variable must point to the Sybase installation directory.

Set the SYBASE_FTS Environment Variable

The SYBASE_FTS must point to the directory where you installed the enhanced FTS.

Add the Full-Text Search engine to the PATH environment variable

Set PATH = %SYBASE%\%SYBASE_FTS%\dll;%PATH%

Add entries to the interfaces file (*sql.ini*)

Use the **dsedit** utility to add entries to the interfaces file (*sql.ini*). Adaptive Server uses the interfaces file to define connection information for servers and clients. Follow the steps below to use **dsedit**:

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

     ```
     dsedit
     ```

     dsedit displays the Select Directory Services dialog box.
2. Make sure that **dsedit** is pointing to the correct configuration file. The default is *%SYBASE% \ %SYBASE_OCS% \ ini \ libtcl.cfg*. If your configuration file is in another directory, change this setting.
3. In the DS Name list box, highlight the directory service you want to open, and choose OK.

   dsedit displays the Interfaces Driver dialog box. Table 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. From the Server Object drop-down list, choose Add. **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.

➤ ***Note***

The Full-Text Search engine uses the server name *textsvr* as the default name in installation scripts and sample files. If you use Sybase Central to start the Full-Text Search engine, Sybase recommends you add the extension _TS to your server name. For example, TEXTSVR_TS.

dsedit returns to the Interfaces Driver dialog box, which displays the values for Server Entry Version, Server Name, 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. In the Protocol entry box, enter the network protocol NLMSNMP (Named Pipes) or NLWNSCK (Winsock). These are the only valid values for the Full-Text Search engine.

   dsedit displays the Network Address dialog box.
9. Enter the network address information, and choose OK.

   dsedit returns to the Network Address Attribute dialog box and displays the new network connection information.
10. Choose OK to return to the Interfaces Driver dialog box.

Start the Full-Text Search Engine

You must start the Full-Text Search engine before you can continue the configuration. For instructions, see "Starting the Full-Text Search Engine on Windows NT" on page 7-3 of this book.

Configure ASE for the Text Server

To configure ASE for the text server proceed to Chapter 4, "Configuring Adaptive Server for Full-Text Searches."

Installing Enhanced Full-Text Search on UNIX

Use the Studio Installer to install the Enhanced Full-Text Search engine.

The Studio Installer is the Sybase installation utility with a graphical user interface that creates the target directory (if necessary), sets Sybase environment variables, collects licensing certificate information, and performs the basic configuration.

For configuration information see the appropriate chapters in this book.

Enhanced Full-Text Search should be installed into an existing Sybase directory structure that includes Adaptive Server Enterprise 12.0.

1. Insert the CD in the CD-ROM drive. If you get CD-reading errors, check your operating system kernel to make sure the ISO 9660 option is turned on.
2. Ensure that the CD-ROM drive is mounted.

➤ ***Note***

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

3. Verify that you are logged in as the user “sybase”.
4. At the UNIX prompt, start the Studio Installer:

   ```
   cd /<device_name>/cdrom
   ./install
   ```

5. Select the type of installation to be performed.
   - Standard Install - installs the default components a user needs.
   - Full Install - installs every component on the CD.
   - Customized Install - allows user to select which components to install.
6. The Studio Installer displays the default installation directory. Accept the default, or enter a different directory, then click Next.
7. A Summary screen displays a list of applications, and services that will be installed, the disk space required, and the disk space available. If the target directory does not have enough free space, the information appears in red.

   Click Next.

◆ ***WARNING!***

If you have insufficient disk space and click Next, an error occurs that stops the installation.

8. If the target installation directory does not already exist, the Studio Installer prompts: "OK to create directory?"

   Click Yes.

9. If a Full Text Search installation already exists, the Studio Installer will notify you and prompt: "Are you sure you want to continue?"
10. A status bar shows the progress of the installation. This step may take quite a while.
11. When prompted to configure the Full-Text Search click YES. The **srvbuild** utility will start automatically.

➤ ***Note***

If you encounter problems during the installation, see the installation log file to see a record of the installation process.

$SYBASE/Installer.log

`Where $SYBASE is the SYBASE target installation directory.`

The srvbuild Utility

srvbuild displays the Select Servers to Build menu.

1. Choose Full-Text Search SDS. **srvbuild** activates the Server Name box. The default server name that displays is the name of the machine on which you installed the Full-Text Search engine.
2. Accept the default name or enter a new name in the Server Name box.

➤ ***Note***

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. If you do not name the Full-Text Search engine "textsvr", **srvbuild** creates a configuration file named *server_name.cfg*. For example, if you enter KRAZYKAT as the name of your Full-Text Search engine, **srvbuild** creates a configuration file named *KRAZYKAT.cfg*.

3. 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 error log 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 *$SYBASE/$SYBASE_FTS/install/server_name.log*.
   - Collection directory – the location of the Verity collections. The default location of the collections is *$SYBASE/$SYBASE_FTS/collections*.
   - Default database – the name of the database that contains the metadata used by the Full-Text Search engine. 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 the database itself later, following the instructions in "Running the installtextserver Script" on page 4-2.
   - Language – the language used by the Full-Text Search engine. The default is us_english. Set the language parameter 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 "min_sessions and max_sessions" on page 8-4.
   - 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 "min_sessions and max_sessions" on page 8-4.

You can adjust any of these options for a configuration that best suits your site. **srvbuild** lists only the required configuration parameters. For a complete list of parameters and how to change them, see "Modifying the Configuration Parameters" on page 7-5.

4. Choose the Build Server! button 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 building the Full-Text Search engine, **srvbuild** creates the runserver and configuration files and also sets the SYBASE and LD_LIBRARY_PATH environment variables.
5. Exit **srvbuild**.

Starting the Full-Text Search Engine

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

Configure ASE for Full-Text Search

To configure ASE for the text server proceed to Chapter 4, "Configuring Adaptive Server for Full-Text Searches."

4 Configuring Adaptive Server for Full-Text Searches

This chapter describes how to configure Adaptive Server to perform full-text searches. Topics include:

Configuring Adaptive Server for a Full-Text Search Engine

The Full-Text Search engine is a remote server that Adaptive Server connects to through Component Integration Services (CIS). Before you can use the Full-Text Search engine, you must configure Adaptive Server for the Full-Text Search engine as follows:

- Enable the **enable cis** and **cis rpc handling** configuration parameters if you have not done so
- Run the **installtextserver** script to define one or more Full-Text Search engines
- Run the **installmessages** script to install messages for the Full-Text Search engine's system procedures
- Run the **installevent** script to create the *text_events* table in each user database which will contain text indexes.
- Name the local server and reboot.

Enabling Configuration Parameters

To connect to the Full-Text Search engine, Adaptive Server must be running with the **enable cis** and **cis rpc handling** configuration parameters enabled. If those parameters are not enabled, log in to Adaptive Server using **isql** and use **sp_configure** to enable them. For example:

```
exec sp_configure "enable cis", 1
exec sp_configure "cis rpc handling", 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 parameters to take effect.

Running the *installtextserver* Script

The **installtextserver** script:

- Defines the Full-Text Search engine as a remote server of server class *sds* to Adaptive Server.
- Creates a database for storing text index metadata. For more information about this database, see "The text_db Database" on page 2-2.
- Installs the system procedures required by the Full-Text Search engine.

Run the **installtextserver** script only once (see "Running the installtextserver Script" on page 4-3). To add Full-Text Search engines at a later time, use **sp_addserver.** See "Configuring Multiple Full-Text Search Engines" on page 8-5 for more information about **sp_addserver.**

All Full-Text Search engines use the same database for storing text index metadata. This database is referred to in this book as the *text_db* database, the default name.

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

Editing the *installtextserver* Script

The **installtextserver** script is located in the *$SYBASE/$SYBASE_FTS/scripts* directory. Use a text editor (such as **vi** or **emacs**) to open the script, and make your edits. The edits you can make are as follows:

- Changing the name of the *text_db* database. If you use a different name, replace all occurrences of *text_db* with the appropriate name.

➤ ***Note***

If you change the name of the *text_db* database, you must change the name in the **defaultDb** configuration parameter (see "Modifying the Configuration Parameters" on page 7-5).

- Changing the name of the Full-Text Search engine. By default, the **installtextserver** script defines a Full-Text Search engine named "textsvr." If your Full-Text Search engine is named differently, edit this script so that it defines the correct server name.

- Adding multiple Full-Text Search engines (for information on how this can enhance performance, see "Configuring Multiple Full-Text Search Engines" on page 8-5). If you are initially defining more than one Full-Text Search engine, edit the **installtextserver** script so that it includes all the Full-Text Search engine definitions. **installtextserver** includes the following section for naming the Full-Text Search engine you are configuring ("textsvr" by default):

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

  Add an entry for each Full-Text Search engine you are configuring. For example, if you are configuring three Full-Text Search engines named KRAZYKAT, OFFICAPUP, and MOUSE, replace the default "textsvr" line with the following lines:

  ```
  exec sp_addserver KRAZYKAT, sds, KRAZYKAT
  exec sp_addserver OFFICAPUP, sds, OFFICAPUP
  exec sp_addserver MOUSE, sds, MOUSE
  go
  ```

- If you use OmniConnect to communicate with the Full-Text Search engine, change the server name specification in the **sp_addobjectdef** calls for the *vesaux* and *vesauxcol* tables to a valid remote server. For example, if your remote server is named REMOTE, change the lines:

```
exec sp_addobjectdef "vesaux","SYBASE.master.dbo.vesaux","table"
exec sp_addobjectdef "vesauxcol","SYBASE.master.dbo.vesauxcol", "table"
```

to:

```
exec sp_addobjectdef "vesaux","REMOTE.master.dbo.vesaux","table"
exec sp_addobjectdef "vesauxcol","REMOTE.master.dbo.vesauxcol", "table"
```

Running the *installtextserver* Script

Use **isql** to run the **installtextserver** script. For example, to run the **installtextserver** script in an Adaptive Server named MYSVR, enter:

```
isql -Usa -P -SMYSVR -i
$SYBASE/$SYBASE_FTS/scripts/installtextserver
```

Running the *installmessages* Script

The Full-Text Search engine has its own set of system procedure messages that you must install in Adaptive Server. Use the **installmessages** script to install the messages. You run the **installmessages** script only once, even if you have multiple Full-Text Search engines.

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

```
isql -Usa -P -SMYSVR -i
$SYBASE/$SYBASE_FTS/scripts/installmessages
```

Running the *installevent* Script

Each database containing tables referenced by a text index must contain a *text_events* table, which logs inserts, updates, and deletes to indexed columns. It is used to propagate updated data to the Verity collections.

Run the **installevent** script, as described below, to create the *text_events* table and associated system procedures in a database. Use the **installevent** script as follows:

- If all databases require text indexes, run the **installevent** script to create a *text_events* table in the *model* database. Each newly created database will then have a *text_events* table. To add a *text_events* table to existing databases, edit the script as described below to create the *text_events* table in the existing user database.
- If not all databases have text indexes, use the **installevent** script as a sample. For each existing database and each new database that includes tables that require text indexing, run the **installevent** script. You must edit the script as described below, to create the *text_events* table in the correct user database.

➤ ***Note***

If a *text_events* table does not exist in a database that includes source tables that require text indexing, changes to the source table will not be propagated to the Verity collections.

Editing the *installevent* Script

The **installevent** script is located in the $*SYBASE/$SYBASE_FTS/scripts* directory. Use a text editor (such as **vi** or **emacs**) to open the script, and make the edits. The edits you can make are:

- Changing the user database name. The **installevent** script creates an events table (named *text_events*) and associated system procedures in the *model* database. The *model* database is the default database. To install the *text_events* table in an existing user database, edit the script and replace all references to *model* with the user database name.
- Changing the *text_db* database name. If your database for storing text index metadata is named something other than *text_db*, replace all references to *text_db* with the appropriate name.

➤ ***Note***

The name of the *text_db* database must be the same as the name in the **defaultDb** configuration parameter (see "Modifying the Configuration Parameters" on page 7-5).

Running the *installevent* Script

Using **isql**, run the **installevent** script to install the *text_events* table and related system procedures in Adaptive Server. For example, to run the **installevent** script in a server named MYSVR, enter:

```
isql -Usa -P -SMYSVR -i $SYBASE/$SYBASE_FTS/scripts/installevent
```

➤ ***Note***

The *text_db* database must exist before you run the **installevent** script. If it does not exist, run the **installtextserver** script first.

Name the local server

When using the full-Text Search engine with ASE 12.0, you must name the local ASE server using the stored procedure, **sp_addserver <servername>, local.** After issuing **sp_addserver**, the local server must be rebooted. Do not install any system stored procedures in the *model* database. They should be installed in *sybsystemprocs*. If they are

installed in model, every new database that is created will inherit a copy.

Creating and Maintaining the Text Indexes

Before the Full-Text Search engine can process full-text searches, you must create text indexes for the source tables in the user database. After the text indexes are created, you must update them when the source data changes to keep the text indexes current. To create and maintain the text indexes:

- Set up the source table for indexing (see "Setting Up Source Tables for Indexing" on page 4-6).
- Create the text indexes and index tables (see "Creating the Text Index and Index Table" on page 4-7).
- Bring the databases online for full-text searches (see "Bringing the Database Online for Full-Text Searches" on page 4-9).
- Propagate changes in the user data to the text indexes (see "Propagating Changes to the Text Index" on page 4-10).
- If you are replicating text indexes, set up text indexing in the destination database (see "Replicating Text Indexes" on page 4-10).

For an example of setting up a text index, see the sample script **sample_text_main.sql** in the *$SYBASE/$SYBASE_FTS/sample/scripts* directory.

Setting Up Source Tables for Indexing

The source table contains the data on which you perform searches (for example, the *blurbs* table in the *pubs2* database). For more information on source tables, see "The Source Table" on page 2-1.

Before you can create text indexes on a source table, you must:

- Verify that the source table has an IDENTITY column
- Create a unique index on the IDENTITY column (optional)

Adding an IDENTITY Column to a Source Table

Every source table must contain an IDENTITY column to uniquely identify each row and provide 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 column value is stored in the text index and is mapped to the *id* column in the index table.

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

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

where m =< 38 and n always = 0

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

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

Adding a Unique Index to an IDENTITY Column

For optimum performance, Sybase recommends creating a unique index on the IDENTITY column. For example, to create a unique index named *comp_id* on the IDENTITY column created above, enter:

```
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 the **sp_create_text_index** system procedure to create the text indexes. **sp_create_text_index** does the following:

- Updates the *vesaux* and *vesauxcol* tables in the *text_db* database
- Creates the text index (Verity collections)
- Populates the Verity collections
- Creates the index table in the user database where the source table is located

The text index can contain up to 16 columns. Columns of the following datatypes can be indexed:

Standard Version Datatypes

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

Enhanced Version Datatypes

All Standard version datatypes, plus:

int, *smallint*, and *tinyint*

For example, to create a text index and an index table named *i_blurbs* for the *copy* column in the *blurbs* table in *pubs2* on KRAZYKAT, enter:

```
sp_create_text_index "KRAZYKAT", "i_blurbs", "blurbs", " ",
"copy"
```

where:

- KRAZYKAT is the name of the Full-Text Search engine
- *i_blurbs* is the name of the index table and text index you are creating
- *blurbs* is the source table on which you are creating the text indexes
- " " is a placeholder for text index creation options
- *copy* is the column in the blurbs table that you are indexing

See "sp_create_text_index" on page A-6 for more information.

➤ ***Note***

Make sure the *text_db* 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. Also, verify that the *text_events* table exists in the user database. If it does not exist, run the **installevent** script for that database (refer to "Running the installevent Script" on page 4-4).

Populating the Verity collections can take a few minutes or several hours, depending on the amount of data you are indexing. You may want to perform this step when the server is not being heavily used. Increasing the **batch_size** configuration parameter will also expedite the process. See "batch_size" on page 8-4 for more information.

➤ ***Note***

Do not rename an index because the Verity collection will not be renamed.

Specifying Multiple Columns When Creating a Text Index

When you create a text index on two or more columns, each column in the text index is placed into its own document zone. The name of the zone is the name of the column. For example, to create a text index and an index table named *i_blurbs* for both the *copy* column and the *au_id* column in the *blurbs* table in *pubs2* on KRAZYKAT, enter:

```
sp_create_text_index "KRAZYKAT", "i_blurbs", "blurbs", " ",
"copy", "au_id"
```

sp_create_text_index creates two zones in the text index named "copy" and "au_id." When you issue a query against the *i_blurbs* text index, the search includes the *copy* and *au_id* columns. However, you can limit your search to a particular column by using the **in** operator to specify a document zone (for more information, see "in" on page 6-12).

Bringing the Database Online for Full-Text Searches

With the Standard version of Full-Text Search engine, you must manually bring a database online before issuing full-text queries on a source table in the database. When you bring a database online, the Full-Text Search engine initializes the internal Verity structures and confirms that the Verity collections exist.

➤ ***Note***

With the Enhanced Full-Text Search engine, the database is automatically brought online when the **auto_online** configuration parameter is set to 1.

Use the **sp_text_online** system procedure to bring a database online for full-text searches if it is not automatically brought online. For example, to bring the *pubs2* database online before issuing full-text searches on the *blurbs* table in 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-33 for more information.

Propagating Changes to the Text Index

When you **insert**, **update**, or **delete** data in your source table, the text indexes are not updated automatically. After you update data, run the **sp_refresh_text_index** system procedure to log the changes to the *text_events* table. Then, run the **sp_text_notify** system procedure to notify the Full-Text Search engine that changes need to be processed. The Full-Text Search engine then connects to Adaptive Server, reads the entries in the *text_events* table, determines which indexes, tables, and rows are affected, and updates the appropriate collections.

See "sp_refresh_text_index" on page A-16 and "sp_text_notify" on page A-32 for more information on these system procedures.

To have **sp_refresh_text_index** run automatically after each **insert**, **update**, or **delete**, you can create triggers on your source tables, as follows:

- Create a trigger that runs **sp_refresh_text_index** after a **delete** operation.
- Create a trigger that runs **sp_refresh_text_index** after an **insert** operation.
- Create a trigger that runs **sp_refresh_text_index** after an **update** operation to an indexed column.

Triggers are not fired when you use **writetext** to update a *text* column. To have **sp_refresh_text_index** automatically run after a **writetext**:

- Set up a non-*text* column and **update** that column after each **writetext**.
- Create a trigger on the non-*text* column to run **sp_refresh_text_index**. Since the Full-Text Search engine reinserts the entire row when you issue **sp_text_notify**, the update to the *text* column gets propagated to the text index.

For examples of each of these triggers, see the sample script **sample_text_main.sql** in the *$SYBASE/$SYBASE_FTS/sample/scripts* directory.

Replicating Text Indexes

To replicate tables that have text indexes, follow these guidelines:

- Create the table definition in the destination database.
- Run the **installevent** script to create the *text_events* table in the destination database, if the *text_events* table does not already exist (see "Running the installevent Script" on page 4-4).

- Run **sp_create_text_index** to create the text index on the empty table in the destination database (see "Creating the Text Index and Index Table" on page 4-7).
- Create triggers for running **sp_refresh_text_index** to **insert** entries into the *text_events* table whenever you **insert**, **update**, or **delete** data into the table (see "Propagating Changes to the Text Index" on page 4-10).
- Create the replication definition in the Replication Server. This replicates all the data in the source table to the destination table. Refer to the "Replication Server Administration Guide" for more details.
- Run **sp_text_notify** to **update** the text index; run **sp_text_notify** periodically to process changes to the destination table (see "Propagating Changes to the Text Index" on page 4-10).

➤ ***Note***

You must issue an **update** against a non-*text* column whenever a **writetext** command is performed. This ensures that the trigger that inserts data into the *text_events* table is fired.

Example: Enabling a New Database for Text Searches

This example describes the steps for creating a *text* index on the *plot* column of the *reviews* table in the *movies* database. This process assumes that:

- You have created a *reviews* table in a new database named *movies* on the MYSVR server
- The *reviews* table has a column named *plot* that you are going to index
- Adaptive Server and the Full-Text Search engine named MYTXTSVR have been configured to connect to each other

Step 1. Verify that the *text_events* Table Exists

Each database containing tables referenced by a *text* index must contain a *text_events* table, which logs inserts, updates, and deletes to indexed columns.

If a *text_events* table is in your *model* database, it will be in all new databases. If a *text_events* table is not in your *model* database, run the

installevent script to install the *text_events* table in the new database. For example, to install the *text_events* table in the *movies* database*:*

- Save the **installevent** script as **installeventmovies.**
- Edit the script to replace all references to the word *model* with the word *movies*.
- Run the script as follows:

```
isql -Usa -P -SMYSVR -i
$SYBASE/$SYBASE_FTS/scripts/installeventmovies
```

See "Running the installevent Script" on page 4-4 for information on installing the *text_events* table.

Step 2. Check for 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.

For example, to add an IDENTITY column to the *reviews* table, enter:

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

See "Adding an IDENTITY Column to a Source Table" on page 4-6 for more information on creating an IDENTITY column.

Step 3. Create a Unique Index on the IDENTITY Column

This step is optional. To enhance performance, Sybase recommends creating a unique index that contains only the IDENTITY column. For example, to create a unique index named *reviews_id* on the IDENTITY column created in step 2, issue the command:

```
create unique index reviews_id on reviews(id)
```

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

Step 4. Create the Text Index and Index Table

The source tables in the user database need to be indexed so that you can perform full-text searches. For example, to create a text index and an index table named *reviews_idx* for the *plot* column in the *reviews* table, enter:

```
sp_create_text_index "MYTXTSVR", "reviews_idx", "reviews", " ",
"plot"
```

The *reviews* table is now available for running full-text searches.

See “sp_create_text_index” on page A-6 for more information.

Step 5. Bring the Database Online for a Full-Text Search

To bring the *movies* database online for the Full-Text Search engine named MYTXTSVR, enter:

```
sp_text_online MYTXTSVR, movies
```

➤ ***Note***

Omit this step if you have Enhanced Full-Text Search engine and your **auto_online** configuration parameter is set to “1”.

See “sp_text_online” on page A-33 for more information.

5 Setting Up Verity Functions

This chapter describes the setup required before you can write queries with certain Verity functionality. It includes:

Enabling Query-By-Example, Summarization, and Clustering

The *style.prm* file specifies additional data to include in the text indexes to support the following functionality:

- Query-by-example – Retrieves documents that are similar to a phrase (see "like" on page 6-13 for more information).

➤ ***Note***

The text indexes only need additional data to support phrases in the query-by-example specification of the **like** operator. If you use a document in the query-by-example specification, additional data is not required.

- Summarization – returns summaries of documents rather than entire documents (see "Using the summary Column to Summarize Documents" on page 6-6 for more information).
- Clustering – groups documents in result sets by subtopic (see "Using Pseudo Columns to Request Clustered Result Sets" on page 6-6 for more information). Clustering is available only with the Enhanced Full-Text Search engine.

You can enable these features for all text indexes by editing the master *style.prm* file, or you can enable them for an individual text index by editing its *style.prm* file. Both methods are describe below.

Query-By-Example and Clustering

To use phrases in a query-by-example specification and to use clustering, you must enable the generation of document feature

vectors at indexing time. To do this, uncomment the following line in the *style.prm* file:

```
$define DOC-FEATURES "TF"
```

Summarization

To configure the Full-Text Search engine for summarization, uncomment one of the following lines that starts with "#$define" in the *style.prm* file:

```
# 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"
```

Each of those 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:

```
$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.

Editing the Master *style.prm* File

The master *style.prm* file is located in:

$SYBASE/$SYBASE_FTS/verity/common/style

It contains the default Full-Text Search engine style parameters. Edit this file to configure the Full-Text Search engine so that all tables on which you create text indexes allow clustering and literal text in your query-by-example specifications, or summarization. Uncomment the applicable lines as described above.

➤ ***Note***

If you have existing text indexes, you must re-create the text index with these features enabled as described in "Editing Individual style.prm Files" below.

Editing Individual *style.prm* Files

Perform the following steps to configure the Full-Text Search engine so that the individual text index allows clustering and literal text in your query-by-example specifications, or summarization:

1. Create the text index using **sp_create_text_index**. Use the word "empty" in the *option_string* parameter so that the *style.prm* file is created for the text index, but the Verity collections are not populated with data. For example, if you are enabling clustering for the *copy* column of the *blurbs* table, use the following syntax:

   ```
   sp_create_text_index "KRAZYKAT", "i_blurbs",
   "blurbs", "empty", "copy"
   ```

➤ ***Note***

If the text index already exists, omit this step. You do not need to create the text index again.

2. Use **sp_drop_text_index** to drop the text index associated with the *style.prm* file you are editing.

   For example, to drop the text index created in step 1, enter:

   ```
   sp_drop_text_index "blurbs.i_blurbs"
   ```

3. Edit the *style.prm* file that exists for the text index. The *style.prm* file for an existing collection is located in:

   $SYBASE/$SYBASE_FTS/collections/db.owner.index/style

   where *db.owner.index* is the database, the database owner, and the index created with **sp_create_text_index**. For example, if you create a text index called *i_blurbs* on the *pubs2* database, the full path to these files is:

   $SYBASE/$SYBASE_FTS/collections/pubs2.dbo.i_blurbs/style

4. Uncomment the applicable lines as described above.

For example, to enable clustering, uncomment the following line:

```
$define DOC-FEATURES "TF"
```

5. Re-create the text index you dropped in step 2. For example, to re-create the *i_blurbs* text index, enter:

```
sp_create_text_index "KRAZYKAT", "i_blurbs", "blurbs", "", "copy"
```

Setting Up a Column to Use As a Sort Specification

Before you can sort by specific columns, you must modify the *style.vgw* and *style.ufl* files. (For information on including a column in a sort specification, see "Using the sort_by Column to Specify a Sort Order" on page 6-4.) Both files are in the directory:

$SYBASE/$SYBASE_FTS/collections/db.owner.index/style

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 *i_blurbs* on the *pubs2* database, the full path to those files would be similar to:

$SYBASE/$SYBASE_FTS/collections/pubs2.dbo.i_blurbs/style

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. (Dropping the text index does not drop the collection directory.)

   For example, to add definitions for the *copy* column in the *blurbs* table, use the following command to drop the text index:

   ```
   sp_drop_text_index i_blurbs
   ```

2. Edit the *style.vgw* file. Following this line:

   ```
   dda "SybaseTextServer"
   ```

   add an entry for the column you are defining. The syntax 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, by adding the column definition for a data table named *fts*. The syntax 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**.

Using Filters on Text That Contains Tags

To perform accurate searches on documents that contain tags (such as HTML or postscript), the text index must use a filter to strip out the tags. The Standard Full-Text Search engine provides filtering for SGML and HTML documents. The Enhanced Full-Text Search engine provides filters for a variety of document types (Microsoft Word, FrameMaker, WordPerfect, SGML, HTML, and others).

When you create the text index to use a filter, the data for each type of tag in the document is placed into its own document zone. For example, if you have a tag called "chapter," all chapter names are placed into one document zone. You can issue a query that searches the entire document, or that searches only for data in the "chapter" zone (for more information, see "in" on page 6-12).

To create a text index that uses a filter, modify the *style.dft* file for that text index. To edit the *style.dft* file, follow these steps:

1. Create the text index using **sp_create_text_index**. Use the word "empty" in the *option_string* parameter so that the *style.dft* file is created for the text index, but the Verity collections are not populated with data. For example, to create a text index for the *copy* column of the *blurbs* table, use the following syntax:

   ```
   sp_create_text_index "KRAZYKAT", "i_blurbs",
   "blurbs", "empty", "copy"
   ```

2. Drop the text index that you create in step 1. This drops the text index, but not the *style.dft* file. For example, use the following command to drop the *i_blurbs* text index:

   ```
   sp_drop_text_index i_blurbs
   ```

3. Edit the *style.dft* file. The *style.dft* file is in the directory:

   $SYBASE/$SYBASE_FTS/collections/db.owner.index/style

   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 *i_blurbs* on the *pubs2* database, the full path to the *style.dft* file would be similar to:

   $SYBASE/$SYBASE_FTS/collections/pubs2.dbo.i_blurbs/style

   Following this line:

   ```
   field:  f0
   ```

   add syntax to use a filter.

   With Standard Full-Text Search engine, use the following syntax:

- For SGML documents, use:

/filter="zone -nocharmap"

- For HTML documents, use:

/filter="zone -html -nocharmap"

With Enhanced Full-Text Search engine, use the following syntax for all document types:

/filter="universal"

For example, your *style.dft* file for an SGML document in the Standard version will look like this:

```
$control: 1
dft:
{
    field: f0
        /filter="zone -nocharmap"
    field: f1
    field: f2
    .
    .
    field: f15
{
```

Your *style.dft* file for an SGML document in the Enhanced version will look like this:

```
$control: 1
dft:
{
    field: f0
        /filter="universal"
    field: f1
    field: f2
    .
    .
    field: f15
{
```

➤ ***Note***

Use **getsend** to load the database with document data. **getsend** takes the following arguments: *database*, *table*, *column* and *row id*. Insert a null value for the *rowid* for each row of text you wish to **insert**. **getsend** must **insert** into an *image* column for filtering to work. For more information on **getsend**, refer to the *README.TXT* file and *getsend.c* file in *$SYBASE/$SYBASE_FTS/sample/source* directory.

4. Re-create the index, using **sp_create_text_index**. For example:

```
sp_create_text_index "KRAZYKAT", "i_blurbs", "blurbs",
"", "copy"
```

Creating a Custom Thesaurus (Enhanced Version Only)

The Verity **thesaurus** operator expands a search to include the specified word and its synonyms (for information on using the **thesaurus** operator, see “thesaurus” on page 6-16). In the Enhanced version of the Full-Text Search engine, you can create a custom thesaurus that contains application-specific synonyms to use in place of the default thesaurus.

For example, the default English language thesaurus contains these words as synonyms for “money”: “cash,” ”currency,” ”lucre,” ”wampum,” and ”greenbacks.” You can create a custom thesaurus that contains a different set of synonyms for “money”; for example, such as: ”bid,” ”tokens,” ”credit,” ”asset,” and ”verbal offer.”

To create a custom thesaurus, follow these steps:

1. Make a list of the synonyms that you will use with your application. It may help to examine the default thesaurus (see “Examining the Default Thesaurus (Optional)” on page 5-9).
2. Create a control file that contains the synonyms you are defining for your custom thesaurus (see “Creating the Control File” on page 5-9).
3. Create the custom thesaurus using the **mksyd** utility (see “Creating the Thesaurus” on page 5-10). This uses the control file as input.
4. Replace the default thesaurus with your custom thesaurus (see “Replacing the Default Thesaurus with the Custom Thesaurus” on page 5-11).

For more information on "Custom Thesaurus Support" and the **mksyd** utility, see the Verity Web site at:

http://www.verity.com

In the Enhanced version of Full-Text Search engine, two sample files illustrate how to set up and use a custom thesaurus:

- *sample_text_thesaurus.ctl* is a sample control file
- *sample_text_thesaurus.sql* issues queries against the custom thesaurus defined in the sample control file

These files are in the *$SYBASE/$SYBASE_FTS/sample/scripts* directory.

Examining the Default Thesaurus (Optional)

A control file contains all the synonym definitions for a thesaurus. To examine the default thesaurus, create its control file using the **mksyd** utility. Use the syntax:

```
mksyd -dump -syd
$SYBASE/$SYBASE_FTS/verity/common/vdkLanguage/vdk20.syd -f
work_location/control_file.ctl
```

where:

- *vdkLanguage* – is the value of the **vdkLanguage** configuration parameter (for example, "english")
- *work_location* – is the directory where you want to place the control file
- *control_file* – is the name of the control file you are creating from the default thesaurus

Examine the control file (*control_file.ctl*) that it creates to view the default synonym lists.

Creating the Control File

Create a control file that contains the new synonyms for your custom thesaurus. The control file is an ASCII text file in a structured format. Using a text editor (such as **vi** or **emacs**), either:

- Edit the control file from the default thesaurus and add new synonyms to the existing thesaurus (see "Examining the Default Thesaurus (Optional)" on page 5-9), or
- Create a new control file that includes only your synonyms

Control File Syntax

The control file contains synonym list definitions in a **synonyms:** statement. For example, the following is a control file named *colors.ctl*:

```
$control: 1
synonyms:
{
list: "red, ruby, scarlet, fuchsia,\
magenta"
list: "electric blue <or> azure"
/keys = "lapis"
}
$$
```

The **synonyms:** statement includes:

- **list:** keywords that specify the start of a synonym list. The synonyms in the list are either in query form or in a list of words or phrases separated by commas.
- Each **list:** can optionally have a **/keys** modifier that specifies one or more keys separated by commas. In the example above, no keys are specified in the first "list". This means the list is found when the thesaurus is queried for the word "red," "ruby," "scarlet," "fuchsia," or "magenta." The second "list" uses the **/keys** modifier to specify one key. This means the words or phrases in the list will satisfy a query only when you specify **<thesaurus>lapis**.

➤ ***Note***

If you use **emacs** to build a synonym list and any of your lists go beyond one line, turn off **auto-fill** mode. If you separate your list into multiple lines, include a backslash (\) at the end of each line so that the lines are treated as one list.

For more complex examples of control files, see the Verity Web site.

Creating the Thesaurus

The **mksyd** utility creates the custom thesaurus using a control file as input. It is located in:

$SYBASE/$SYBASE_FTS/verity/bin

Run, or define an alias to run, **mksyd** from this *bin* directory. Create your custom thesaurus in any work directory.

The **mksyd** syntax for creating a custom thesaurus is:

```
mksyd -f control_file.ctl -syd custom_thesaurus.syd
```

where:

- *control_file* – is the name of the control file you create in "Creating the Control File" above
- *custom_thesaurus* – is the name of the custom thesaurus you are creating

For example, to execute the **mksyd** utility reading the sample control file defined above, and directing output to a work directory, use the syntax:

```
mksyd -f /usr/u/sybase/dba/thesaurus/colors.ctl
-syd /usr/u/sybase/dba/thesaurus/custom.syd
```

Replacing the Default Thesaurus with the Custom Thesaurus

The default thesaurus named *vdk20.syd* is located in:

$SYBASE/$SYBASE_FTS/verity/common/vdkLanguage

where *vdkLanguage* is the value of the **vdkLanguage** configuration parameter (for example, the English directory is *$SYBASE/$SYBASE_FTS/verity/common/english*). Each application and user reading from this location at runtime uses this thesaurus. To replace it with your custom thesaurus, follow these steps:

1. Back up the default thesaurus before replacing it with the custom thesaurus. For example:

```
mv /$SYBASE/$SYBASE_FTS/verity/common/english/vdk20.syd
default.syd
```

2. Replace the *vdk20.syd* file with your custom thesaurus. For example:

```
cp custom.syd
/$SYBASE/$SYBASE_FTS/verity/common/english/vdk20.syd
```

3. Restart your Full-Text Search engine; no configuration file changes are required. The thesaurus is read from this location when the Full-Text Search engine is started, not when a query is executed.

Queries using the **thesaurus** operator will now use the custom thesaurus.

Creating Topics (Enhanced Version Only)

The section provides a condensed overview of Verity Topics. Topics are discussed in detail in Chapter 9, “Verity Topics.”

A TOPIC® is a grouping of information related to a concept or subject area. With topic definitions in place, a user can perform searches on the topic instead of having to write queries with complex syntax.

The user creates topics which can be combinations of words and phrases, Verity operators and modifiers, and weight values. Then, any user can query the topic.

Before you create topics, determine your application requirements, and establish standards for naming conventions and for the location of the following:

- Outline files – contains the topic definitions. Each topic has its own outline file.
- Topic set directories – contains the compiled topic. Each topic has its own topic set directory.
- Knowledge base map file – contains pointers to the topic set directories.

To implement topics, perform the following steps:

1. Create one or more outline input files to define your topics (see “Creating an Outline File” on page 5-13). Each outline file is used to populate one topic set.
2. Create and populate a topic set directory, using the **mktopics** utility (see “Creating a Topic Set Directory” on page 5-14). Each topic set directory is populated based on one topic outline input file.
3. Create a **knowledge base map**, specifying the locations of one or more topic set directories (see “Creating a Knowledge Base Map” on page 5-14)
4. Set the **knowledge_base** configuration parameter to point to the location of the knowledge base map (see “Defining the Location of the Knowledge Base Map” on page 5-15)
5. Execute queries against defined topics.

The following sample files illustrate the topics feature:

- *sample_text_topics.otl* is a sample outline file
- *sample_text_topics.kbm* is a sample knowledge base map

- *sample_text_topics.sql* issues queries using defined topics

These files are in the *$SYBASE/$SYBASE_FTS/sample/scripts* directory.

Creating an Outline File

A topic outline file specifies all the combinations of words and phrases, Verity operators and modifiers, and weight values that the search engine uses when you issue a query using the topic. The outline file is an ASCII text file in a structured format.

For example, the following outline file defines the topic "saint-bernard":

```
$control: 1
saint-bernard <accrue>
*0.80 "Saint Bernard"
*0.80 "St. Bernard"
* "working dogs"
* "large dogs"
* "European breeds"
$$
```

When you issue a query specifying the topic "saint-bernard", the Full-Text Search engine:

- Returns documents that contain one or more of the following phrases: "Saint Bernard," "St. Bernard," "working dogs," "large dogs," and "European breeds"
- Scores documents that contain the phrase "Saint Bernard" or "St. Bernard" higher than documents that contain the phrase "working dogs, "large dogs," or "European breeds"

This example is a very basic topic definition. An outline can introduce more complex relationships by using:

- Multiple levels of subtopics
- Combinations of Verity operators (this example uses **accrue**)
- Verity modifiers

➤ ***Note***

In Windows NT, you can use the graphical user interface of the Verity Intelligent Classifier product to create topic outlines. It is available from Verity. If you use Intelligent Classifier, it automatically creates a topic set directory, and you can go to "Creating a Knowledge Base Map" on page 5-14 to continue setting up your topics.

Creating a Topic Set Directory

Use the **mktopics** utility to create and populate a topic set directory. It is located in:

$SYBASE/$SYBASE_FTS/verity/bin

Run, or define an alias to run, **mktopics** from this *bin* directory. You can create a topic set directory or directories in any work directory.

The **mktopics** syntax is:

```
mktopics -outline outline_file.otl -topicset topic_set_directory
```

where:

- *outline_file* – is the name of the outline file you create in "Creating an Outline File" on page 5-13
- *topic_set_directory* – is the name of the topic set directory you are creating

For example, to execute the **mktopics** utility reading the *saint-bernard.otl* file defined above, and directing output to a work directory, use the syntax:

```
mktopics -outline /usr/u/sybase/topic_outlines/saint-bernard.otl
-topicset /usr/u/sybase/topic_sets/saint-bernard_topic
```

Creating a Knowledge Base Map

A **knowledge base map** specifies the locations of one or more topic set directories. Create an ASCII knowledge base map file that defines the fully-qualified directory paths to your topic sets.

For example, the following knowledge base map file illustrates how you can list multiple knowledge bases in the map. The first entry identifies the topic set directory created with **mktopics** above.

```
$control:1
kbases:
{
kb:
/kb-path = /usr/u/sybase/topic_sets/saint-bernard_topic
kb:
/kb-path = /usr/u/sybase/topic_sets/another_topic
}
```

Defining the Location of the Knowledge Base Map

Set the **knowledge_base** configuration parameter to point to the location of the knowledge base map. For example:

```
sp_text_configure KRAZYKAT, 'knowledge_base',
'/usr/u/sybase/topic_sets/sample_text_topics.kbm'
```

The **knowledge_base** configuration parameter is static, and you must restart the Full-Text Search engine for the definition to take effect.

Executing Queries Against Defined Topics

You can now execute queries using the defined topic instead of a complex query. For example, before you create the "saint-bernard" topic, you would have to use the following syntax:

```
...where i.index_any = "<accrue> ([80]Saint
Bernard, [80]St. Bernard, working dogs, large
dogs, European breeds)"
```

to find documents that:

- Contain one or more of the following phrases: "Saint Bernard," "St. Bernard," "working dogs," "large dogs," and "European breeds"
- Score documents containing the phrase "Saint Bernard" or "St. Bernard" higher than documents containing the phrase "working dogs," "large dogs," or "European breeds"

After you create the topic "saint-bernard", you can use this syntax:

```
...where i.index_any = "<topic>saint-bernard"
```

or:

```
...where i.index_any = "saint bernard"
```

➤ ***Note***

If you enter a word in a query expression, the Full-Text Search engine tries to match it with a topic name. If you enter a phrase in a query expression, the Full-Text Search engine replaces spaces with hyphens (-), and then tries to match it with a topic name. For example, the Full-Text Search engine matches “saint bernard” with the topic “saint-bernard”.

See the *sample_text_topics.sql* file for examples of using topics in queries.

Troubleshooting Topics

If the **knowledge_base** configuration parameter specifies a knowledge base map file that does not exist, the Full-Text Search engine will not be able to start a session with Verity, and the server will not start. If the map file exists but contains invalid entries, Verity issues warning messages at start-up time. You can correct errors by editing the <textserver>.cfg file in the $SYBASE directory. You can correct path information and change the line beginning: “knowledge_base=”.

6 Writing Full-Text Search Queries

This chapter describes the pseudo columns, search operators, and modifiers that you can include in a full-text search. Topics include:

Components of a Full-Text Search Query

To write a full-text search query, you enter the search parameters as part of an Adaptive Server **select** statement. Then the Full-Text Search engine processes the search. The **select** statement requires:

- A **where** clause that assigns a Verity language query to the *index_any* pseudo column
- Pseudo columns to further define the parameters of the search (optional)
- A join between the IDENTITY column from the source table and the *id* column from the index table

For example, to return the 10 documents from the *copy* column of the *blurbs* table that have the most occurrences of the word "software," enter:

```
select t1.score, t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id
and t1.index_any = "<many> <word> software"
and t1.max_docs = 10
```

Adaptive Server passes the Verity query to the Full-Text Search engine to process the search. For more information on how Adaptive Server processes the query, see "How a Full-Text Search Works" on page 2-5.

Default Behaviour

The default or simple syntax of a query to the full-Text Search engine is treated broadly:

1. Searches are case sensitive.
2. The STEM operator applies to search words.
3. The MANY modifier is applied.
4. The ACCRUE operator is activated at the parent level.

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. (For more information about index tables, see “The Index Table” on page 2-3.) These columns are valid only in the context of a 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 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 displays only the result set that falls within the parameters you define. For example, the following query displays only the results that have a *score* value greater than 90:

```
index_table_name.score > 90
```

Other pseudo columns are used to retrieve data generated by Verity for a particular document. Table 6-1 describes the pseudo columns that are maintained by the Full-Text Search engine.

Table 6-1: Full-Text Search engine pseudo columns

Pseudo Column Name	Description	Datatype	Length (in Bytes)
cluster_number	Enhanced Full-Text Search engine only. Contains the cluster that the row is part of. Clusters are numbered starting with 1. You can use the *cluster_number* column only in the **select** clause of a query.	*int*	4
cluster_keywords	Enhanced Full-Text Search engine only. Contains the keywords that Verity uses to build the cluster. You can use *cluster_keywords* only in the **select** clause of a query.	*varchar*	255
id	Uniquely identifies a document within a collection. Used to join with the IDENTITY column of the source table. You can use *id* in the **select** clause or **where** clause of a query.	*numeric*	6

Table 6-1: Full-Text Search engine pseudo columns (continued)

Pseudo Column Name	Description	Datatype	Length (in Bytes)
index_any	Provides a Verity language query to the Full-Text Search engine. You can use *index_any* only in a **where** clause.	*varchar*	255
max_docs	Limits results to the first *n* documents, based on the default sort order. In a clustered result set, limits results to the first *n* documents in each cluster. You can use *max_docs* only in a **where** clause.	*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. You can use *score* in a **select** clause or a **where** clause.	*int*	4
sort_by	Specifies the sort order in which to return the result set. • The Standard Full-Text Search engine allows a single sort specification in the *sort_by* column. • The Enhanced Full-Text Search engine allows up to 16 sort specifications in the *sort_by* column. You can use *sort_by* only in a **where** clause.	*varchar*	35
summary	Selects summarization data. You can use the *summary* column only in the **select** clause of a query.	*varchar*	255

The following sections describe the functionality of the pseudo columns.

Using the *score* Column to Relevance-Rank Search Results

Relevance ranking is the ability of the Full-Text Search engine to assign the *score* parameter a value that indicates how well a document satisfies the query. The *score* calculation depends on the search operator used in the query (for more information, see “Using the Verity Operators” on page 6-11). The closer the document satisfies the query, the higher the *score* value is for that document.

For example, if you search for documents that contain the word “rain,” a document with 12 occurrences of “rain” receives a higher *score* value than a document with 6 occurrences of “rain.”

If you set *score* to a high value (such as 90) in your query, you limit the result set to documents that have a *score* value greater than that number.

➤ ***Note***

Verity uses decimals for *score* values; Sybase uses whole numbers. For example, if Verity reports a score value of .85, Sybase reports the same value as 85.

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

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

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

(0 rows affected)
```

The query does not find any documents that contain the word "raconteur" or "Paris" and that have a score greater than 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" or "Paris":

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

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

Using the *sort_by* Column to Specify a Sort Order

The sort order specifies the collating sequence used to order the data in the result set. The default sort order is set by the **sort_order** configuration parameter (for more information, see "Setting the Default Sort Order" on page 7-11). Case insensitive sort order is not

supported in the Standard Full-Text Search. It is supported in the Enhanced version.

Use the *sort_by* pseudo column to return a result set with a sort order other than the default. With the Standard Full-Text Search engine, you can specify a single sort specification in the *sort_by* pseudo column. With the Enhanced Full-Text Search engine, you can specify up to 16 sort specifications in the *sort_by* pseudo column.

Table 6-2 lists the values for the *sort_by* pseudo column.

Table 6-2: Values for the sort_by pseudo column

Value	Description
fts_score desc	Returns a result set sorted by score in descending order.
fts_score asc	Returns a result set sorted by score 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.
***column_name* desc**	Returns a result set sorted according to the descending order of a column. *column_name* is the name of the source table's column.
***column_name* asc**	Returns a result set sorted according to the ascending order of a column. *column_name* is the name of the source table's column.
fts_cluster asc	Returns a clustered result set. Only available with the Enhanced Full-Text Search engine. (See "Using Pseudo Columns to Request Clustered Result Sets" on page 6-6 for more information.)

➤ ***Note***

Before you can sort by specific columns, you must modify the *style.vgw* and *style.ufl* files (see "Setting Up a Column to Use As a Sort Specification" on page 5-4).

For example, the following query sorts the documents by timestamp in ascending order:

```
select t1.score, t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id and t1.score > 90
and t1.index_any = "<accrue>(raconteur, Paris)"
and t1.sort_by = “fts_timestamp asc”
```

Using the *summary* Column to Summarize Documents

Use the *summary* pseudo column to have queries return only summaries of the documents that meet the search criteria, rather than returning entire documents. The *summary* column is not available by default; you must edit the *style.prm* file prior to creating the text index to enable summarization. See “Enabling Query-By-Example, Summarization, and Clustering” on page 5-1 for information about enabling the *summary* 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, t1.summary
from i_blurbs t1, blurbs t2
where t1.id=t2.id and t1.score > 70
and t1.index_any = "(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.

For additional examples of queries using summarization, see the sample script *sample_text_queries.sql* in the *$SYBASE/$SYBASE_FTS/sample/scripts* directory.

Using Pseudo Columns to Request Clustered Result Sets

The clustering function analyzes a result set and groups rows into clusters so that the rows in each cluster are semantically more similar to each other, on average, than they are to other rows in other clusters. Ordering rows by subtopics can help you get a sense of the major subject areas covered in the result set. Clustering is only available with the Enhanced Full-Text Search Specialty Data Store.

Returning a clustered result set can be significantly slower than returning a nonclustered result set. If the response time of a query is critical, use a nonclustered result set.

Preparing to Use Clustering

Before you request a clustered result set, you must build the text index with the clustering function enabled (see "Enabling Query-By-Example, Summarization, and Clustering" on page 5-1).

The Verity clustering algorithm attempts to group similar rows together, based on the values of the following configuration parameters:

- **cluster_style**
- **cluster_max**
- **cluster_effort**
- **cluster_order**

Use the **sp_text_cluster** system procedure to have a query use values that are different from the default values of these configuration parameters. (For values and how to set them for a query, see "sp_text_cluster" on page A-20.)

Writing Queries Requesting a Clustered Result Set

To obtain a clustered result set, specify "fts_cluster asc" as the sort specification in the *sort_by* pseudo column of the query. For example:

```
select t1.score, t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id
and t1.index_any = "<many> <word> software"
and t1.max_docs = 10
and t1.sort_by = "fts_cluster asc"
```

Include any of the following pseudo columns in your query to return additional clustering information:

- *cluster_number* – contains the number of the cluster the row belongs to. Clusters are numbered starting with 1.
- *cluster_keywords* – contains the most common words found in the cluster. The *cluster_keywords* column contains a null value for each row that does not fit into any cluster.

- *max_docs* – limits the number of rows returned for each cluster. (In a nonclustered query, the *max_docs* column limits the total number of rows that are returned in a result set.)
- *score* – contains a value of 0 to 10000. The higher the score, the closer the row is to the cluster center. A score of 0 indicates the row does not fit into any cluster. (In a nonclustered query, the *score* column can contain a value of 0 to 100.) The search engine does not return results with a score of 0. Logically a score of 0 represents "no match" but the user never sees a score of 0.

See the sample script named *sample_text_queries.sql* in the *$SYBASE/$SYBASE_FTS/sample/scripts* directory for examples of SQL statements using clustering.

Full-Text Search Operators

The special search operators that you use to perform full-text searches are part of the Verity search engine. Table 6-3 describes the Verity search operators provided by the Full-Text Search engine.

Table 6-3: Verity search operators

Operator Name	Description
accrue	Selects documents that contain at least one of the search elements specified in a query. The more search elements there are, the higher the score will be.
and	Selects documents that contain all the search elements specified in a query.
complement	Returns the complement of the *score* value (the *score* value subtracted from 100).
in	Selects documents that contain the search criteria in the document zone specified.
like	Selects documents that are similar to the sample documents or passages specified in a query.
near	Selects documents containing the specified search elements, where the closer the search terms are to each other in 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 to each other in a document, the higher the document's score.

Table 6-3: Verity search operators (continued)

Operator Name	Description
or	Selects documents that contain at least one of the search elements specified in a query.
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.
sentence	Selects documents that include all the specified words in 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 specified word and its synonyms.
topic	Specifies that the search term you enter is a topic.
wildcard	Matches wildcard characters included in search strings. Certain characters indicate a wildcard specification automatically.
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.

Considerations When Using Verity Operators

Consider the following when you write full-text search queries:

- You **must** enclose the operators in angle brackets (<>) in the query. If they are not enclosed in angle brackets, the Full-Text Search engine 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
```

- You must enclose the Verity language query in single quotes (') or double quotes ("). The Full-Text Search engine strips off the outermost quotes before it sends the query to Verity. For example, when you enter the query:

  ...where index_any = "'?own'"

  the Full-Text Search engine sends the following query to Verity:

  '?own'

- A query may be comprised of several "index_any" clauses **and**ed together in SQL. The right **and** value strings can be prefixed with "<snnn>". All such strings will be concatenated in Full-Text Search in the order determined by the "nnn" values. The "<snnn>" is removed. For instance:

  where index_any="<s001>hello"
  and index_any="<s002> world"

  is the same as:

  where index_any = "hello world"

  This is a handy work-around for search strings that are greater than 255 characters.

- Search terms entered in mixed case automatically become case sensitive. Search terms entered in all uppercase or all lowercase are not automatically case sensitive. For example, a query on "Server" finds only the string "Server". A query on "server" or "SERVER" finds the strings "Server", "server", and "SERVER".

- You can use alternative syntax for the query expressions shown in Table 6-4.

Table 6-4: Alternative Verity syntax

Standard Query Expression	Alternative Syntax
<MANY><WORD>*string*	"*string*"
<MANY><STEM>*string*	'*string*'

When using the alternative syntax, remember that the Full-Text Search engine strips off the outermost quotes before it sends the query to Verity. For example, when you enter the query:

```
...where index_any = "'play'"
```

the Full-Text Search engine sends the following query to Verity:

```
'play'
```

This is the same as:

```
<MANY><STEM>play
```

Using the Verity Operators

The following sections describe how to use the Verity operators shown in Table 6-3 on page 6-8.

accrue

The **accrue** operator selects documents that contain at least one of the search items specified in the query. There must be two or more search elements. Each result is relevance-ranked. For example, the following query searches for the word “restaurant” or “deli” or both in the *copy* column of the *blurbs* table:

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

and, *or*

The **and** and **or** operators select documents that contain the specified search elements. Each result is relevance-ranked. 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 t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id
and t1.index_any = "(Iranian <and> business)"
```

The **or** operator selects the documents that contain any of the search elements. For example, if the preceding query is rewritten to use the **or** operator, the query selects documents that contain the word “Iranian” or “business”:

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

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 selects documents that contain the specified search element in one or more document zones. Document zones are created for a text index in the following two scenarios:

- When you create an index on two or more columns using **sp_create_text_index**, a document zone is created for each column in the text index (for more information, refer to “Specifying Multiple Columns When Creating a Text Index” on page 4-9). A document zone is not created when you create a text index on a single column. For example, if you specify the *au_id* and *copy* columns of the *blurbs* table when you create the text index, you can issue the following query:

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

  This returns rows that contain the word “gorilla” in the *copy* column. However, if you specify only the *copy* column of the *blurbs* table when you create the text index, this query does not return any rows.

- When you create an index that uses a filter, a document zone is created for each tag in the document (for more information, see "Using Filters on Text That Contains Tags" on page 5-6). You can limit your search to a particular tag by specifying the tag name after the **in** operator. For example, to search for the word "automotive" in a "title" tag in an HTML document, specify:

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

Text indexes utilizing filters can contain only one column.

like

The **like** operator selects documents that are similar to the document(s) or passages you provide. The search engine analyzes the text to find the most important terms to use. If you specify multiple samples, the search engine selects important terms that are common across the samples. Each result is relevance-ranked.

The **like** operator accepts a single operand, called the query-by-example (QBE) specification. The QBE specification can be either literal text or document IDs. The document IDs are from the IDENTITY column in the source table. For example, to select documents that are similar to the document in the *copy* column in the row with an IDENTITY of "2", enter:

```
select t1.score, t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id and t1.score > 35
and t1.index_any = '<like> ( "{2}" )'
```

Before using literal text in the QBE specification, you must uncomment the following line in the *style.prm* file:

```
$define DOC-FEATURES "TF"
```

For more information, see "Enabling Query-By-Example, Summarization, and Clustering" on page 5-1.

See the sample script named *sample_text_queries.sql* in the *$SYBASE/$SYBASE_FTS/sample/scripts* directory for examples of SQL statements using QBE.

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:

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

or

See “and, or” on page 6-11.

phrase

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

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

paragraph

The **paragraph** operator selects documents in which the specified search elements appear in the same paragraph. The closer the words are to each other in a paragraph, the higher the score the document receives in relevance-ranking. The following example searches for documents in which the words “text” and “search” occur within the same paragraph:

```
select t1.score, t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id and t1.score > 50
and t1.index_any = "<many><paragraph>(text, search)"
```

product

The **product** operator multiplies the *score* value for the documents for each of the search elements. To arrive at a document's score, the Full-Text Search engine calculates a score for each search element and multiplies the *scores*. For example:

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

The *score* value for each search element is 78; however, because the *score* values for the items are multiplied, the document has a *score* value of 61 (.78 x.78 =.61(100) = 61).

sentence

The **sentence** operator selects documents in which the specified search elements appear in the same sentence. The closer the words are to each other in a sentence, the higher the score the document receives in relevance-ranking. The following example searches for documents in which the words "tax" and "service" occur within the same sentence:

```
select t1.score, t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id and t1.score > 50
and t1.index_any = "<many><sentence>(tax, service)"
```

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 documents that contain "cooked," "cooking," "cooks," and so on. To relevance-rank the result set, include the **many** modifier in the query (see "Operator Modifiers" on page 6-19).

The following query uses the **stem** operator to find documents that contain 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 in which "create" is not a perfect stem ("creative"), the document is still selected:

```
select t1.score, t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id and t1.score > 10
and t1.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
      . . .
78    If Chastity Locksley didn't exist, this troubled world
      would have created her!  Not only did she master the mystic
```

sum

The **sum** operator totals the *score* values for each search element, up 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 those scores.

thesaurus

The **thesaurus** operator searches for documents containing a synonym for a search element. 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). Each result is relevance-ranked.

The Full-Text Search engine supplies a default thesaurus. With the Enhanced Full-Text Search engine, you can create a custom thesaurus. For more information, see "Creating a Custom Thesaurus (Enhanced Version Only)" on page 5-8.

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 t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id
and t1.index_any = "<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
      . . .
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
      . . .
```

topic (Enhanced Version Only)

The **topic** operator selects documents that meet the search criteria defined by the specified topic. For more information, see “Creating Topics (Enhanced Version Only)” on page 5-12. For example, use the following syntax to find documents that meet the criteria defined by the topic “engineering”:

```
select t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id
and t1.index_any = "<topic>(engineering)"
```

wildcard

The **wildcard** operator allows you to substitute wildcard characters for part of the item for which you are searching. Table 6-5 describes the wildcard characters and their attributes.

Table 6-5: Full-Text Search engine wildcard characters

Character	Function	Syntax	Locates
?	Specifies one alphanumeric character. 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 ({}).	**'?an'**	“ran,” “pan,” “can,” and “ban”

Table 6-5: Full-Text Search engine wildcard characters

Character	Function	Syntax	Locates
*	Specifies zero or more of any alphanumeric character. 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 ({}).	**'corp*'**	"corporate," "corporation," "corporal," and "corpulent"
[]	Specifies any single character in a set. If a word includes a set, you must enclose the word in backquotes (‘ ‘). Also, there can be no spaces in a set.	**<wildcard> ‘c[auo]t‘**	"cat," "cut," and "cot"
{}	Specifies one of each pattern separated by a comma. If a word includes a pattern, you must enclose the word in backquotes (‘ ‘). Also, there can be no spaces in a set.	**<wildcard> ‘bank{s,er,ing}‘**	"banks," "banker," and "banking"
^	Specifies one of any character not included in a set. The caret (^) must be the first character after the left bracket ([) that introduces a set.	**<wildcard> ‘st[^oa]ck‘**	Excludes "stock" and "stack," but locates "stick" and "stuck"
-	Specifies a range of characters in a set.	**<wildcard> ‘c[a-r]t‘**	Includes every three-letter word from "cat" to "crt"

To relevance-rank the result set, include the **many** modifier in the query (see "Operator Modifiers" on page 6-19).

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

```
select t2.copy
from i_blurbs t1, blurbs t2
where t1.id=t2.id
and t1.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,
     . . .
```

word

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

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

yesno

The **yesno** operator converts all nonzero *score* values to 100. For example, if the score values for five documents are 86, 45, 89, 89, and 100, each of those documents is returned with a *score* value 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 modifiers that you can use with the operators to refine a search. The modifiers are described in Table 6-6.

Table 6-6: Verity operator modifiers

Modifier Name	Description	Works with These Operators	Example
case	Performs case-sensitive searches. If you enter search terms in mixed case, the search is automatically case sensitive.	**wildcard** **word**	`<case><word>(Net)`
many	Counts the number of times that a word, stemmed word, or phrase occurs in a document. Relevance-ranks the document according to its density.	**paragraph** **phrase** **sentence** **stem** **word** **wildcard**	`<many><stem>(write)`

Table 6-6: Verity operator modifiers (continued)

Modifier Name	Description	Works with These Operators	Example
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 in which 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)`

7 System Administration

This chapter describes system administration issues for both the Standard and Enhanced versions of the Full-Text Search engine. Topics include:

Starting the Full-Text Search Engine on UNIX

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

```
startserver -f
$SYBASE/$SYBASE_FTS/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 of the configuration parameters.

Creating the Runserver File

The runserver file contains start-up commands for the Full-Text Search engine. The runserver file can include the flags shown in Table 7-1.

Table 7-1: 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.

Table 7-1: Definition of flags in the runserver file (continued)

Flag	Definition
-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 *$SYBASE/$SYBASE_FTS/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 correct path environment variable for your platform in the runserver file. Table 7-2 shows the path environment variable to use for each platform.

Table 7-2: Path environment variable for the runserver file

Platform	Environment Variable
RS/6000 AIX	LIBPATH
Sun Solaris	LD_LIBRARY_PATH
HP 9000(800)	SHLIB_PATH
Digital UNIX	LD_LIBRARY_PATH

For example, the runserver file on Sun Solaris for a Full-Text Search engine named KRAZYKAT would be *RUN_KRAZYKAT* and would be similar to:

```
#!/bin/sh
#

LD_LIBRARY_PATH="$SYBASE/$SYBASE_FTS/lib:$LD_LIBRAR
Y_PATH"
export LD_LIBRARY_PATH

$SYBASE/bin/txtsvr -SKRAZYKAT
```

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.

Starting the Full-Text Search Engine on Windows NT

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

- From Sybase Central – see your Sybase Central documentation for information about starting servers.
- As a service – see "Starting the Full-Text Search Engine As a Service" below.
- From the command line – use the following syntax:

```
%SYBASE%\%SYBASE_FTS%\bin\txtsvr.exe -Sserver_name
    [-t] [-i%SYBASE%path_to_sql.ini_file]
    [-l%SYBASE%path_to_errorlog]
```

where:

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

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

```
%SYBASE%\%SYBASE_FTS%\bin\txtsvr.exe -SKRAZYKAT -t
```

The Full-Text Search engine is up and running when you see the start-up complete message.

Starting the Full-Text Search Engine As a Service

Use the **instsvr** utility in Sybase Central to add the Full-Text Search engine to the list of items you can start and stop with the Services utility. **instsvr** is located in the *%SYBASE%\%SYBASE_FTS%\bin* directory.

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

```
instsvr.exe service_name %SYBASE%\%SYBASE_FTS%\bin\txtsvr.exe
"startup_parameters"
```

where:

- *service_name* is the name of the Full-Text Search engine you are adding as a service. With Sybase Central, Sybase recommends

you use a server name with the extension "_TS" (for example, KRAZYKAT_TS).

- *startup_parameters* are any parameters you want used at start-up.

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

```
instsvr.exe KRAZYKAT_TS %SYBASE%\sds\text\bin\txtsvr.exe
"-SKRAZYKAT_TS -t"
```

➤ ***Note***

If you need to include more than one parameter (for example, **-i**), you must include all the parameters in one set of double quotes.

To configure Sybase Central to start and stop your Full-Text Search engine, you must provide a service name that begins 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 in the interfaces file is KRAZYKAT_TS, run the following **instsvr** command to create a service that can be managed by Sybase Central:

```
instsvr SYBTXT_KRAZYKAT_TS %SYBASE%\%SYBASE_FTS%\bin\txtsvr.exe
"-SKRAZYKAT_TS -t"
```

Shutting Down the Full-Text Search Engine

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

***server_name*...sp_shutdown**

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

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

```
KRAZYKAT...sp_shutdown
```

Modifying the Configuration Parameters

Each Full-Text Search engine has configuration parameters with default values, as shown in Table 7-3.

Table 7-3: Configuration parameters

Parameter	Description	Default Value
batch_size	Determines the size of the batches sent to the Full-Text Search engine.	500
batch_blocksize	When enabled, the text server reads data in smaller chunks. This parameter instructs the text server to retrieve *n* number of rows at a time. Should be set to 0 (disabled) to 65535.	0
max_indexes	The maximum number of text indexes that will be created in the Full-Text Search engine.	126
max_stacksize	Size (in kilobytes) of the stack allocated for client threads.	34,816
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
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 engine.	850
vdkLanguage	Language used by Verity search engine.	english

Table 7-3: Configuration parameters (continued)

Parameter	Description	Default Value
vdkHome	Verity directory.	UNIX: *$SYBASE/$SYBASE_FTS/verity* Windows NT: *%SYBASE%\%SYBASE_FTS%\verity*
collDir	Storage location of the Full-Text Search engine's collection.	UNIX: *$SYBASE/$SYBASE_FTS/collections* Windows NT: *%SYBASE%\%SYBASE_FTS%\collections*
defaultDb	Name of the Full-Text Search engine database that stores text index metadata.	*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*
sort_order	Default sort order.	0
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
srv_traceflags	String containing numeric flag identifiers used to generate Open Server diagnostic information.	0

The Enhanced Full-Text Search engine has additional configuration parameters as shown in Table 7-4:

Table 7-4: Configuration parameters for Enhanced version only

Parameter	Description	Default Value
cluster_style	Clustering style to use.	Fixed
cluster_max	Maximum number of clusters to generate when **cluster_style** is set to **Fixed**.	0

Table 7-4: Configuration parameters for Enhanced version only (continued)

Parameter	Description	Default Value
cluster_effort	Amount of effort the Full-Text Search engine should expend on finding a good cluster.	Default
cluster_order	The order to return clusters and rows within a cluster.	0
auto_online	Specifies whether to bring indexes online automatically when the Full-Text Search engine is started. 0 indicates online is not automatic; 1 indicates automatic.	0
backDir	The default location for the placement of text index backup files.	UNIX: *$SYBASE/$SYBASE_FTS/backup* Windows NT: *%SYBASE%\%SYBASE_FTS%\backup*
knowledge_base	The location of a knowledge base map for implementing the Verity topics feature.	null
nocase	Sets the case-sensitivity of the Full-Text Search engine. If you are using a case-sensitive sort order in Adaptive Server, set to 0. If you are using a case-insensitive sort order, set to 1.	0

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

Modifying Values in the Standard Version

With Standard Full-Text Search Specialty Data Store, you use a configuration file to change the default values. The configuration file is named *server_name.cfg* and is in the *$SYBASE* directory. *server_name* is the name of the Full-Text Search engine.

- For UNIX, the **srvbuild** utility creates the configuration file when it builds the Full-Text Search engine.
- For Windows NT, you manually create the configuration file by copying a sample configuration file with default values.

To modify the default values, use a text editor to edit the configuration file. Uncomment the line that contains the configuration parameter you are modifying. You must restart the Full-Text Search engine for the new values to take effect.

Modifying Values in the Enhanced Version

With Enhanced Full-Text Search Specialty Data Store, you can use the **sp_text_configure** system procedure to change the value of a configuration parameter. The syntax is:

```
sp_text_configure server_name, config_name,
   config_value
```

where:

- *server_name* is the name of the Full-Text Search engine
- *config_name* is the name of the configuration parameter
- *config_value* is the value you assign to the configuration parameter

For more information, see “sp_text_configure” on page A-23.

➤ ***Note***

You can also modify the value of a configuration parameter by editing a configuration file as described in “Modifying Values in the Standard Version” on page 7-7.

Available Configuration Paramters

The following table provides a list of available configuration parameters with valid limits:

Table 7-5: Limits to Configuration parameters

Parameter	Values	Static/Dynamic
batch_size	0 - MAX_INT	Dynamic
batch_blocksize	0 - 65535	Dynamic
max_indexes	0 - MAX_INT	Static
max_stacksize	0 - MAX_INT	Static
max_threads	0 - MAX_INT	Static

Table 7-5: Limits to Configuration parameters (continued)

Parameter	Values	Static/Dynamic
max_packetsize	0 - MAX_INT	Static
max_sessions	0 - MAX_INT	Static
min_sessions	0 - max_sessions	Static
language	French, Spanish German, English	Static
charset	ascii_8, cp037, cp1047, cp437, cp500, cp850, deckanji, eucjis, iso_1, mac, roman8, sjis, utf8	Static
vdkCharset	850, 437, 1252, mac1 (Just the ones listed in the manual)	Static
vdkLanguage	French, Spanish German, English	Static
vdkHome	A string < 255 chars	Static
collDir	A string < 255 chars	Static
default_Db	A string < 32 chars	Static
interfaces	A string < 255 chars	Static
sort_order	0, 1, 2, 3	Dynamic
errorLog	A string < 255 chars	Static
traceflags	A string with comma delimited numbers ranging anywhere from 1 to 15.	Static
srv_traceflags	A string with comma delimited numbers ranging anywhere from 1 to 8	Static
cluster_style	Coarse, Medium, Fine, Fixed	Dynamic
cluster_max	0 - MAX_INT	Dynamic
cluster_effort	Low, Medium, High, Default	Dynamic
cluster_order	0 or 1	Dynamic
auto_online	0 or 1	Static
backCmd	A string < 255 chars	Dynamic
restoreCmd	A string < 255 chars	Dynamic
backDir	A string < 255 chars	Static
knowledge_base	A string < 255 chars	Static
nocase	0 or 1	Dynamic

Setting the Default Language

The default language for Verity is set with the **vdkLanguage** configuration parameter. By default, **vdkLanguage** is set to "english". You can configure Verity to use a different default language. Table 7-6 lists the locales supported by Sybase.

Table 7-6: vdkLanguage configuration parameters

Language	Default Locale Name
English	english
German	german
French	french

Additional language adapters are available in the *$SYBASE/$SYBASE_FTS/verity/common* directory; however, the Full-Text Search engine displays messages only in the languages shown in Table 7-6.

The **language** parameter is the language the Full-Text Search engine displays its error messages and Open Server and Open Client error messages. Set the **language** parameter to the Adaptive Server language.

For example, with the Standard Full-Text Search engine, to change the Verity language to Spanish in a server named KRAZYKAT, include the following line in the configuration file:

```
vdkLanguage = spanish
```

With the Enhanced Full-Text Search engine, run the following:

```
sp_text_configure KRAZYKAT, 'vdkLanguage', 'spanish'
```

For more information about the Verity languages, see the Verity Web site:

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. The files used for the Verity

character sets are in *$SYBASE/$SYBASE_FTS/verity/common*. Table 7-7 describes the character sets you can use with Verity.

Table 7-7: Verity character sets

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

The default character set for the Full-Text Search engine is set with the **charset** parameter. Set the **charset** parameter to the Adaptive Server character set.

For example, with the Standard Full-Text Search engine, to change the Verity character set to IBM PC in a server named KRAZYKAT, include the following line in the configuration file:

```
vdkCharset = 437
```

With the Enhanced Full-Text Search engine, run the following:

```
sp_text_configure KRAZYKAT, 'vdkCharset', '437'
```

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** configuration parameter to one of the values in Table 7-8.

Table 7-8: 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.

Table 7-8: Sort order values for the configuration file (continued)

Value	Description
3	Returns result sets sorted by a timestamp in ascending order.

For example, with the Standard Full-Text Search engine, to change the default sort order to sort by descending timestamp in a server named KRAZYKAT, include the following line in the configuration file:

```
sort_order = 2
```

With the Enhanced Full-Text Search engine, enter:

```
sp_text_configure KRAZYKAT, 'sort_order', '2'
```

When you sort a result set by descending timestamp (value 2 in Table 7-8), 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 7-8), 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, the latest document appears first in the result set.

Setting Trace Flags

The **traceflags** parameter enable the logging of certain events when they occur within the Full-Text Search engine. Each trace flag is

uniquely identified by a number. Trace flags are described in Table 7-9.

Table 7-9: Full-Text Search engine trace flags

Trace Flag	Description
1	Traces connects, disconnects, and attention events from Adaptive Server.
2	Traces language events. Traces the SQL statement that Adaptive Server sent to the Full-Text Search engine.
3	Traces RPC events.
4	Traces cursor events. Traces the SQL statement sent to the Full-Text Search engine by Adaptive Server.
5	Writes the errors that display to the log.
6	Traces information about text 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.
13	Traces backup operations. Available only with Enhanced Full-Text Search Specialty Data Store.
14	Logs Verity status and timing information.
15	Generates ngram index information for collections. ngrams increase the speed of wildcard searches. This trace flag is required for wildcard searches against data in unicode format.
30	This traceflag enables the Verity MaxClean feature that removes out of date collection files. It should only be used during maintenance since it could take extra time and interfere with normal usage. It is enabled in conjuntion with **sp_optimize_text_index**.

You can enable and disable trace flags interactively, using the remote procedure calls (RPCs) **sp_traceon** and **sp_traceoff** in the Full-Text Search engine.

To execute **sp_traceon**, use the following syntax:

```
textserver...sp_traceon 1,2,3,4
```

where *textserver* is the name of the Full-Text Search engine.

The traceflags will stay active until the session is terminated or until the **sp_traceoff** RPC is executed using the specific traceflag. To set a traceflag permanently, either set it in the config file or use the **sp_text_configure** command.

Setting Open Server Trace Flags

Use the **srv_traceflags** parameter to turn on trace flags to log Open Server diagnostic information. Open Server trace flags are described in Table 7-10.

Table 7-10: Open Server trace flags

Trace Flag	Description
1	Traces TDS headers.
2	Traces TDS data.
3	Traces attention events.
4	Traces message queues.
5	Traces TDS tokens.
6	Traces Open Server events.
7	Traces deferred event queues.
8	Traces network requests.

For example, with the Standard Full-Text Search engine, to trace attention events on the server named KRAZYKAT, include the following line in the configuration file:

```
srv_traceflags = 3
```

With the Enhanced Full-Text Search engine, run the following:

```
sp_text_configure KRAZYKAT, 'srv_traceflags', '3'
```

Setting Case Sensitivity

By default, the Full-Text Search engine is case sensitive. This means you must enter identifiers in the same case or they are not recognized. For example, if you have a table named *blurbs* (lowercase), you cannot issue an **sp_create_text_index** command that specifies the table name *BLURBS*. You must issue a command that uses the same case for the table name argument:

```
sp_create_text_index "KRAZYKAT", "i_blurbs", "blurbs", "", "copy"
```

With Enhanced Full-Text Search engine, use the **nocase** parameter to set the case sensitivity of the Full-Text Search engine. 0 indicates case sensitive; 1 indicates case insensitive. Set the **nocase** parameter to the sort order case sensitivity in Adaptive Server.

For example:

```
sp_text_configure KRAZYKAT, 'nocase', '1'
```

changes the KRAZYKAT server to case insensitive.

➤ ***Note***

The **nocase** parameter does not affect the case sensitivity of the Verity query. For information on Verity case sensitivity, see "Considerations When Using Verity Operators" on page 6-9.

Backup and Recovery for the Standard Full-Text Search Engine

The Adaptive Server user database and the Verity collections are physically separate. Backing up your user 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 user database and *text_db* database in Adaptive Server.

Make sure you follow the recommended schedule for backing up your databases that is described in Chapter 20, "Developing a Backup and Recovery Plan," of the *System Administration Guide*. Sybase recommends that when you back up a user database with text indexes, you also back up:

- The *text_db* database
- The text indexes

A regular backup schedule ensures the integrity of the text indexes, the Adaptive Server data, and the *text_events* table, all of which are integral to recovering your text indexes without having to drop and re-create them.

➤ ***Note***

You do not have to back up the user database and text indexes at the same time to recover the text indexes. However, you must restore the user database before you restore the text index. Doing so restores the *text_events* table, which the **sp_redo_text_events** system procedure uses to bring the text indexes in sync with the user database.

If you have Enhanced Full-Text Search Specialty Data Store, use the automated process described in "Backup and Recovery for the Enhanced Full-Text Search Engine" on page 7-18.

Backing Up Verity Collections

Follow these steps to back up your Verity collections:

1. Shut down the Full-Text Search engine:

   ***server_name*...sp_shutdown**

2. Back up the files. By default, the collections are located in:

   $SYBASE/$SYBASE_FTS/collections

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

   $SYBASE/$SYBASE_FTS/collections/pubs2.dbo.i_blurbs

   - In UNIX, back up the files by using the **tar** or **cpio** utility
   - In Windows NT, use a compression utility such as PKZIP to back up the files

3. For future reference, make a note of the time of the backup in a permanent location.
4. Back up the user database and the *text_db* database, using the **dump database** command. For more information on the **dump database** command, see the *Adaptive Server Reference Manual*.

5. Restart the Full-Text Search engine. For instructions, see "Starting the Full-Text Search Engine on UNIX" on page 7-1 or "Starting the Full-Text Search Engine on Windows NT" on page 7-3.

Restoring Verity Collections and Text Indexes from Backup

As Database Administrator, follow these steps to restore your Verity collections:

1. Restore the Adaptive Server user database and *text_db* database. This returns the source tables, metadata, and *text_events* table to a consistent and predictable state. See Chapter 21, "Backing Up and Restoring User Databases," in the *System Administration Guide* for more information.
2. Shut down the Full-Text Search engine:

   ***server_name*...sp_shutdown**
3. Restore your collections from the backup files created in step 2 in "Backing Up Verity Collections" on page 7-16.
4. Restart the Full-Text Search engine. For instructions, see "Starting the Full-Text Search Engine on UNIX" on page 7-1 or "Starting the Full-Text Search Engine on Windows NT" on page 7-3.
5. Log in to Adaptive Server, and run the **sp_redo_text_events** system procedure in the restored database. For example, if you are restoring the *pubs2* database, you have to be in that database to run the system procedure, **sp_redo_text_events**, as follows:

   sp_redo_text_events "*from_date*"

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

   For example:

   sp_redo_text_events "10/31/97:16:45"

   restores the collections up to October 31, 1997 at 4:45 PM. For more information, see "sp_help_text_index" on page A-11.
6. Run the **sp_text_notify** system procedure to notify the Full-Text Search engine that changes need to be propagated to the Verity collections. The Full-Text Search engine connects to Adaptive Server, reads all the unprocessed entries in the *text_events* table and applies them to the text index. For more information, see "sp_text_notify" on page A-32.

Your text indexes and collections are now fully restored.

Backup and Recovery for the Enhanced Full-Text Search Engine

Backup and recovery for the Enhanced Full-Text Search Specialty Data Store is automated with the **sp_text_dump_database** and **sp_text_load_index** system procedures. These system procedures provide a seamless interface for maintaining data and text index integrity.

The Adaptive Server user database and the Verity collections are physically separate. Backing up your user 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 user database and the *text_db* database in Adaptive Server.

Follow the recommended schedule for backing up your databases, as described in Chapter 20, "Developing a Backup and Recovery Plan," of the *System Administration Guide*. Sybase recommends that when you back up a user database with text indexes, you also back up:

- The *text_db* database
- The text indexes

➤ ***Note***

You do not have to back up the user database and text indexes at the same time to recover the text indexes. However, you must restore the user database before you restore the text index. This restores the *text_events* table, which the **sp_text_load_index** system procedure uses to bring the text indexes in sync with the user database.

A regular backup schedule ensures the integrity of the text indexes, the Adaptive Server data, and the *text_events* table, all of which are integral to recovering your text indexes without having to drop and re-create them.

If you have Standard Full-Text Search Specialty Data Store, use the process described in "Backup and Recovery for the Standard Full-Text Search Engine" on page 7-15.

Customizable Backup and Restore

backCmd and **restoreCmd** allow customizable backup and restore commands to be used instead of tar or zip commands when backing up collection files. If these two parameters are blank, the default commands are used, otherwise the specified command is executed. String substitution is performed before execution to allow specification of input and output directories and collection identification. The string substitution is defined as follows:

- ${backDir} is replaced by the *backup* directory specified as the "backDir" configuration parameter.
- ${collDir} is replaced by the full path name for the collection
- ${colID} is replaced by the collection ID which is the full name of the backup file.

Backing Up Verity Collections

The **sp_text_dump_database** system procedure backs up collections and (optionally) the user and *text_db* databases. **sp_text_dump_database** also maintains the *text_events* table by deleting entries that are no longer needed for recovery. It is available only with the Enhanced Full-Text Search engine.

During a backup, the Full-Text Search engine processes queries, but defers any update requests until the backup is complete. This eliminates the need to shut down and restart the Full-Text Search engine.

Run **sp_text_dump_database** from the database containing the text indexes you are backing up. Make sure all the required servers are running when issuing the **sp_text_dump_database** command. **sp_text_dump_database** unconditionally backs up all indexes of all enhanced text servers. The backup of the text indexes is placed in the directory specified in the **backDir** configuration parameter. The output of the **dump database** command is written to the Full-Text Search error log. Sybase recommends dumping the current database and the *text_db* database at the time the text indexes are backed up. However, this is optional.

For example, to back up the text indexes, the *sample_colors_db* database to the */work2/sybase/colorsbackup* directory, and the *text_db* database to the */work2/sybase/textdbbackup* directory, enter:

```
sp_text_dump_database @backupdbs =
INDEXES_AND_DATABASES, @current_to = "to
'/work2/sybase/colorsbackup'", @textdb_to="to
'/work2/sybase/textdbbackkup'"
```

➤ ***Note***

It is important to back up the *text_db* database whenever text indexes are backed up, since that database contains the metadata for all text indexes.

sp_text_dump_database may fail on Solaris if the required file size is greater than 2GB.

For more information, see "sp_text_dump_database" on page A-25.

Restoring Collections and Text Indexes from Backup

The **sp_text_load_index** system procedure restores text indexes that have been backed up with the **sp_text_dump_database** system procedure.

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

1. Restore your Adaptive Server user database and *text_db* database. This returns the source tables, metadata, and *text_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 user and *text_db* databases.
2. Run **sp_text_load_index** to restore the Verity collection from the most recent index dump. The procedure resets the status of all *text_events* table entries made since the last index dump to "unprocessed" and notifies the Full-Text Search engine to process those events.

Example:

To restore the *sample_colors_db* database and all of its text indexes:

1. Restore the *text_db* database:

```
1> use master
2> go
1> load database text_db from '/work2/sybase/textdbbackkup'
2> go
```

2. Restore the *sample_colors_db* database:

```
1> load database sample_colors_db from
'/work2/sybase/colorsbackup'
2> go
```

3. Bring the *text_db* and *sample_colors_db* databases online:

```
1> online database text_db
2> online database sample_colors_db
3> go
```

4. Restore the text index:

```
1> use sample_colors_db
2> go
```

```
1> sp_text_load_index
2> go
```

For more information, see “sp_text_load_index” on page A-30.

8 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. This chapter describes ways in which you can enhance performance. Topics include:

Updating Existing Indexes

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

- Enabling trace flag 11 disables Verity collection optimization. This means that 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 completion of **sp_text_notify**.

  With Enhanced Full-Text Search Specialty Data Store, you can use the **sp_optimize_text_index** system procedure to optimize a text index at a later time if trace flag 11 is enabled. (For more information, see “sp_optimize_text_index” on page A-12.)
- Enabling 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 the 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 most of the updates.

Trace flags 11 and 12 can be enabled and disabled interactively using the remote procedure calls **sp_traceon** and **sp_traceoff** in the Full-Text Search engine.

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* pseudo 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 source table and the index table.

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 when the text index is queried first during a join between the text index and one or more tables.

To 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** or enable trace flag 11205, and examine the join order. Trace flag 11205 dumps remote queries to the Adaptive Server error log file. 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*.

Reconfiguring Adaptive Server

You can improve the performance of the Full-Text Search engine by resetting the following Adaptive Server configuration parameters. (For information about setting configuration parameters with **sp_configure**, see Chapter 11,"Setting Configuration Parameters," in the *System Administration Guide*.)

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 will dynamically allocate to return the result set.

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 the 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 will allocate for that parameter.

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. If CIS is used to access other remote servers, the max network packet size on those servers must be increased as well.

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

Reconfiguring the Full-Text Search Engine

You can improve the performance of the Full-Text Search engine by reconfiguring the following Full-Text Search engine configuration parameters (see "Modifying the Configuration Parameters" on page 7-5):

batch_size

The **batch_size** configuration parameter determines the number of rows per batch the Full-Text Search engine indexes. **batch_size** 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 for 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 Full-Text Search engine will need to allocate almost 5MB of memory when creating the text index.

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

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 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.

Using *sp_text_notify*

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

Using the **sp_text_notify** system procedure 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 the 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 tables that are used frequently, you can improve performance by placing the text 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. Each Adaptive Server can connect to multiple Full-Text Search engines, but each Full-Text Search engine can connect to only one Adaptive Server.

Creating Multiple Full-Text Search Engines at Start-Up

If you are initially creating multiple Full-Text Search engines, you can edit the **installtextserver** script so that it includes all of those Full-Text Search engines. For more information, see “Editing the installtextserver Script” on page 4-2.

Adding Full-Text Search Engines

You can add 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 server_name [, server_class [, physical_name]]
```

where:

- *server_name* 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".
- *physical_name* is the name in the interfaces file used by the server *server_name*.

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
```

After you configure and start the Full-Text Search engine, you can use the following syntax to see if you can connect to the Full-Text Search engine from the Adaptive Server:

```
server_name...sp_show_text_online
```

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

```
BLUE...sp_show_text_online
```

Configuring Additional Full-Text Search Engines

Follow the steps described in "Configuring the Full-Text Search Engine" in the *Installation and Release Bulletin* for your platform, to configure additional Full-Text Search engines. Each Full-Text Search engine requires its own:

- Interfaces file entry
- Configuration file

All Full-Text Search engines use the same database (named *text_db* by default) for storing text index metadata and the same *vesaux* and *vesauxcol* tables.

Multiple Users

The following tips will help avoid deadlocks with multiple users:

1. Make sure the ASE is using the same number of connections as the Full-Text Search. 100 is the default.

   ```
   sp_configure "user connections", 100
   ```

2. Make sure the *vesaux*, *vesauxcol* and *text_events* tables (in the model, or in each of your new databases) are using row level locking.

   For existing tables: **alter table** *table_name* **lock datarows**

   For new tables: **create table ... lock datarows**

3. For large batches of commands, try to break them into smaller transactions.

4. If deadlocks still occur, increase the number of locks available to the ASE, and tweak the row lock promotion settings. See the ASE System Administration Guide to assist with setting locks.

9 Verity Topics

This chapter is reproduced with permission from Verity. It is a section of Verity documentation provided to give Full-Text Search users insight into the complex issue of Verity Topics.

What are Topics?

A topic is a grouping of information related to a concept, or a subject area. Topics provide a convenient means by which you can encapsulate knowledge, and make it available to end users as a shared resource. By adding topics to your Verity application, users can more easily perform searches over the subject matter which the topics represent.

Topics are combined to form knowledge bases that represent a catalogue of knowledge that users can tap into when performing searches. Knowledge bases offer users the ability to find the information they want without having to compose sophisticated queries using complex syntax.

Topic Organization

Topics organize groups of related search criteria in a format similar to that of an outline. Operators and modifiers act as the glue that joins related groups of search criteria. You can create topics as independent units, or as units with relationships to other topics in a hierarchical structure.

Weight Assignments

You can even give some groups of search criteria more weight than other groups of search criteria in a topic's structure. Assigning weight to search criteria affects the importance of documents selected in a search; the closer a document is to the top of the results list, the more important, or relevant, the document is to the search criteria. A search criteria weight is a number between 0.01 and 1.00. The position of a selected document in the results list can help you determine at a glance how relevant the document is compared to the search criteria.

Using a Topic Outline File

You can compose topics by creating a topic outline file.

A topic outline file is an ASCII text file in a structured format that contains topic definitions. A topic outline file might appear as follows:

```
$Control:1
art <Accrue>
*performing-arts <Accrue>
**0.80 "ballet"
**0.50 "drama"
**0.50 'dance'
**0.80 "opera"
**0.80 "symphony"
**0.90 "chamber music"
**"Isaac Stern"
*film <Accrue>
**directors <Filter>
/definition="title CONTAINS Truffaut"
*visual-arts <Accrue>
literature <Accrue>
philosophy <Accrue>
language <Accrue>
history <Accrue>
$$
```

You can create a topic outline file with any text editor.

Making Topics Available

The topics you make available to users must exist within a topic set that is generated using the **mktopics** utility. Verity topic sets generated by **mktopics** can be used by any Verity application. A single topic set supports a maximum of 20,000 topic definitions, and the exact number of topics allowed for one topic set depends on the Verity query language used to define them.

Setup Process

Making topics available to users is a three-step process, as outlined below.

1. Create topic definitions using a topic outline file.
2. Generate a topic set. You can create a topic set using the **mktopics** utility. The **mktopics** utility creates the topic set and can also index the topics over a specific collection.
3. Import the topic set to the Full-Text Search engine.

Knowledge Bases of Topics

This section discusses the principle features of knowledge bases, and the organization format used to define topics for them.

The following aspects of topic knowledge bases are covered:

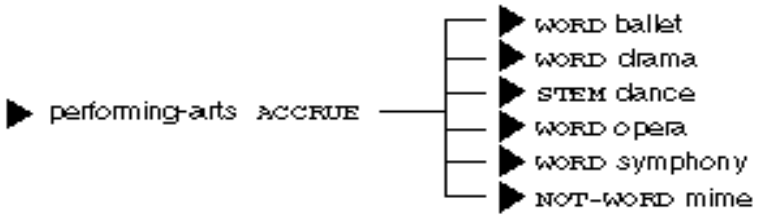
- Combining topics into a knowledge base
- The structure of topics
- The relationship between topics and subtopics
- Topic types
- Naming topics

Combining Topics into a Knowledge Base

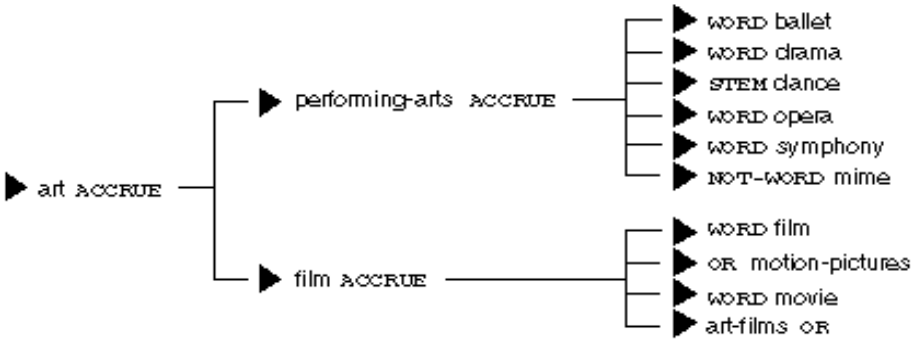
A topic is simply a grouping of information related to a concept, or a subject area. A knowledge base is a grouping of these concepts called topics. Combining topics into a knowledge base provides users with the ability to look up concepts saved as topics in a convenient fashion.

The subject area of a topic is typically identified by the topic's name. In the example below, the subject of the topic is **performing-arts**. This topic is composed of two structural elements, its *name*, **performing-arts**, and its *evidence* topics, *ballet*, *musical*, *dance*, *opera*, *symphony*, and *drama*.

Operators and *modifiers* act as the glue that joins related evidence topics. Operators represent logic to be applied to evidence topics. This logic defines the qualifications of the kinds of documents you want to find. Modifiers apply further logic to evidence topics. For example, a modifier can specify that documents containing an evidence topic not be included in the list of results.



A topic's structure becomes more sophisticated as topics are added to it. In the next example, the topic film has been added to the structure to form what is now the top-level topic, art. In this structure, performing-arts and film are subtopics of the topic art.



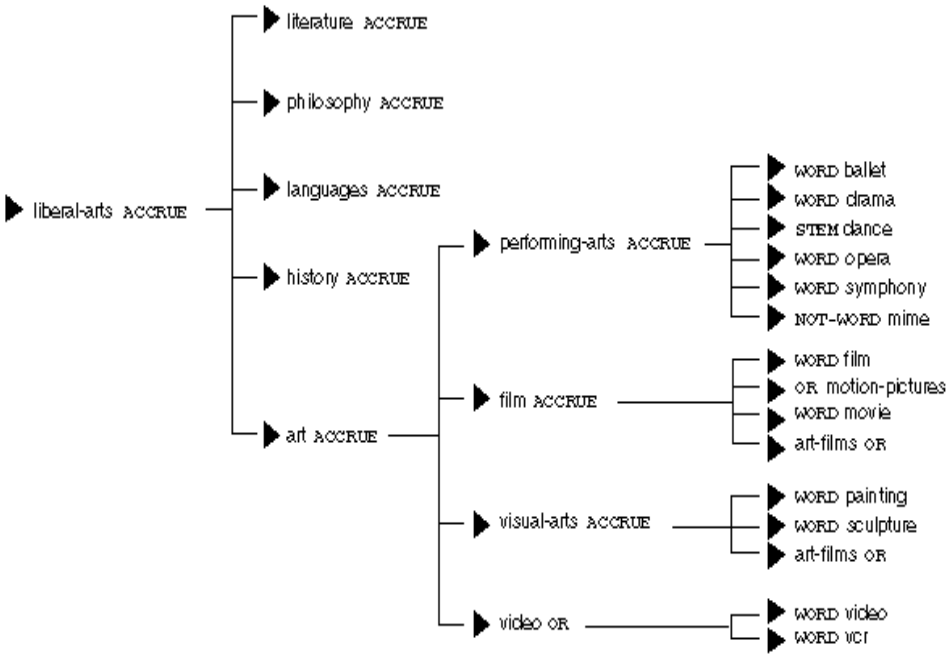
Sophisticated topics are composed of top-level topics, subtopics, and evidence topics. These elements determine the related subject areas of a topic. Typically, a knowledge base consists of several top-level topics. Note that subtopics and evidence topics can be used by multiple top-level topics.

Structure of Topics

The structure of topics affects how the topic is interpreted during search processing. Designing topics so that they accurately express a concept involves defining a topic structure with the components described below.

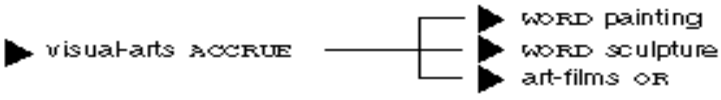
Top-Level Topics

Top-level topics are the highest topics defined in a topic structure. Top-level topics represent the subject areas you want a Verity search agent to find. In the example below you could think of, literature, philosophy, languages, history, and art as top-level subtopics that comprise the top-level topic, liberal-arts.



Subtopics

Subtopics form the levels between top-level topics and evidence topics. The name of a subtopic should identify the subject area that its subtopics or evidence topics combine to describe. For example, the subtopic visual-arts includes several related words, or evidence topics, as shown below:

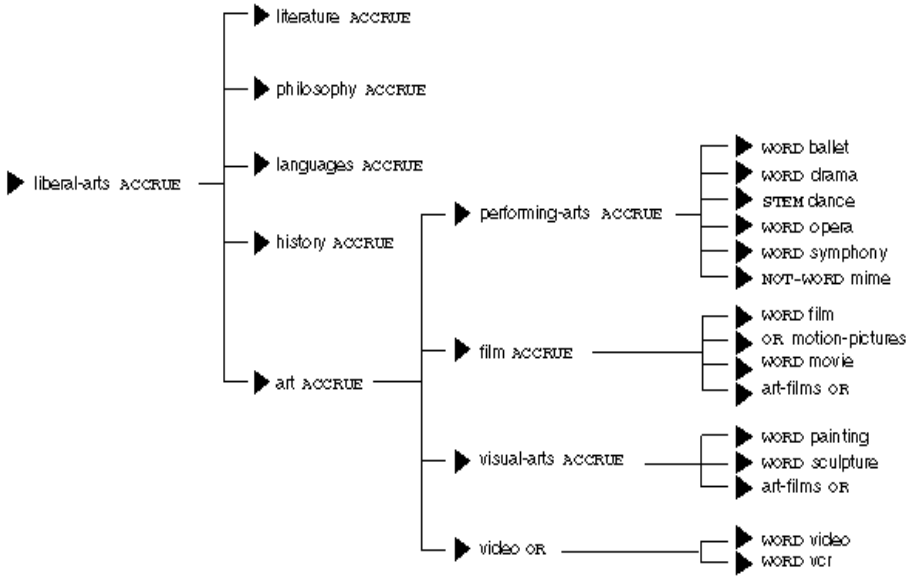


Evidence Topics

Evidence topics are the lowest units of a topic structure. Evidence topics are strings, made up of combinations of alphanumeric characters. An evidence topic can contain up to 128 alphanumeric characters.

Topic and Subtopic Relationships

Each topic and its associated subtopics form a hierarchical parent and child relationship. In the example below, the subtopics performing-arts, film, visual-arts, and video are children of the art topic. The art topic itself is a child of the liberal-arts topic. The liberal-arts topic could in turn be a child of successively higher parent topics within the structure.



When you use a topic to perform a search, the subject area defined by the topic includes its subtopics, their subtopics, and so on, down to the evidence topics of the structure. Topics that are not direct descendants of the topic you use are not included in the search.

In the example above, for instance, a search using the film topic would cause the Verity search engine to find documents containing information on film, motion pictures, movies, and art films. In this

example, the search would not find documents related to the performing-arts, visual-arts, or video topics since these topics are not children, of the film topic. However, if the art topic was used, the search would find documents related to all the art topic's children, which includes performing-arts, film, visual-arts, and video.

Maximum Number of Topics

A single topic set representing a knowledge base can consist of as many as 20,000 topics. This includes top-level topics, subtopics, and evidence topics. Topics containing as many as 1,000 subtopics may exceed memory limitations when used in a search.

Topic Naming Issues

Note the following issues surrounding the naming of topics.

Topic Name Length

A topic name can contain up to 128 alphanumeric characters, including hyphens and underscores.

Case Sensitivity

Topic names and evidence topics are normally case-insensitive. You can name a evidence topic using all caps, as in APPLE, initial caps, as in Apple, or all lower-case, as in apple. Case is not considered when a search is performed. Thus, if your evidence topic is entered as APPLE, the Verity search engine will select documents containing "APPLE", "Apple", or "apple".

You can, however, use the CASE modifier to specify that case match the entry of a evidence topic.

Verity Query Language

This section describes the Verity Query Language, consisting of operators and modifiers that you can use to create topics. Operators represent logic to be applied to search elements which can be combined to create a topic. This logic defines the qualifications of the kinds of documents you want to find. Modifiers apply further logic

to search elements. For example, a modifier can specify that a search element be case-sensitive.

The information in this section includes the following:

- Query Language Summary
- Operator Precedence Rules
- Sample Topic Outlines
- Operator Reference
- Modifier Reference

Query Language Summary

The Verity Query Language consists of operators and modifiers. Both operators and modifiers represent logic to be applied to a search element. This logic defines the qualifications a document must meet to be retrieved. Operators are classified by their type, as follows:

- Evidence operators
- Proximity operators
- Relational operators
- Concept operators
- Boolean operators

Modifiers extend the logic applied by operators and are used in combination with operators.

Evidence Operators

Evidence operators expand a search word into a list of related words which are then searched for as well. When you perform a search using an evidence operator, documents containing one or more occurrences of the words in the expanded word list are documents containing the word specified, as well as its synonyms. Documents retrieved using evidence operators are not relevance-ranked unless you use the MANY modifier. See "MANY Modifier" in this section

for information. The following table describes each evidence operator.

Table 9-1: Evidence Operators

Operator Name	Description
WORD	Selects documents that include one or more instances of a word you specify.
STEM	Selects documents that include one or more variations of the search word you specify.
THESAURUS	Selects documents that contain one or more synonyms of the word you specify.
WILDCARD	Selects documents that contain matches to a character string containing variables.
SOUNDEX	Selects documents that include one or more words that "sound like," or whose letter pattern is similar to, the word specified.
NEAR/N	Expands the search to include the word you enter plus words that are similar to the query term. This operator performs "approximate pattern matching" to identify similar words.

Proximity Operators

Proximity operators specify the relative location of specific words in the document; that is, specified words must be in the same phrase, paragraph, or sentence for a document to be retrieved. In the case of the NEAR and NEAR/N operators, retrieved documents are relevance-ranked based on the proximity of the specified words. When proximity operators are nested, the ones with the broadest scope should be used first; that is, phrases or individual words can appear within SENTENCE or PARAGRAPH operators, and SENTENCE operators can appear within PARAGRAPH operators. The following table describes each proximity operator.

Table 9-2: Proximity Operators

Operator Name	Description
IN	Selects documents that contain specified values in one or more document zones. A document zone represents a region of a document, such as the document's summary, date, or body text.
PHRASE	Selects documents that include a phrase you specify. A phrase is a grouping of two or more words that occur in a specific order.

Table 9-2: Proximity Operators

Operator Name	Description
SENTENCE	Selects documents that include all of the words you specify within a sentence.
PARAGRAPH	Selects documents that include all of the search elements you specify within a paragraph.
NEAR	Selects documents containing specified search terms within close proximity to each other.
NEAR/N	Selects documents containing two or more words within N number of words of each other, where N is an integer.

Relational Operators

Relational operators search document fields (such as AUTHOR) that have been defined in the collection. These operators perform a filtering function by selecting documents that contain specified field values. The fields that are used with relational operators can contain alphanumeric characters. Documents retrieved using relational operators are not relevance-ranked, and you cannot use the MANY modifier with relational operators.

When creating topics, relational operators are always used in conjunction with the special FILTER operator. See the example under the topic "visual-arts" in "Sample Topic Outlines" later in this section for the proper syntax.

A number of relational operators are available for numeric and date comparisons, including the following: = (equals), > (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to).

A number of relational operators are available for text comparisons, including the following.

Table 9-3: Relational Operators

Operator Name	Description
CONTAINS	Selects documents by matching the word or phrase you specify with values stored in a specific document field.
MATCHES	Selects documents by matching the character string you specify with values stored in a specific document field.
STARTS	Selects documents by matching the character string you specify with the starting characters of the values stored in a specific document field.

Table 9-3: Relational Operators

Operator Name	Description
ENDS	Selects documents by matching the character string you specify with the ending characters of the values stored in a specific document field.
SUBSTRING	Selects documents by matching the character string you specify with a portion of the strings of the values stored in a specific document field.

Concept Operators

Concept operators combine the meaning of search elements to identify a concept in a document. Documents retrieved using concept operators are relevance-ranked. The following table describes each concept operator.

Table 9-4: Concept Operators

Operator Name	Description
AND	Selects documents that contain all of the search elements you specify.
OR	Selects documents that show evidence of at least one of your search elements.
ACCRUE	Selects documents that include at least one of the search elements you specify.

Boolean Operators

Boolean operators can be assigned to topics to retrieve documents containing any or all of the children of that topic. Unlike topics created using the concept operators, Boolean operators do not accept weights. The following table describes each Boolean operator..

Table 9-5: Boolean Operators

Operator Name	Description
ALL	Selects documents that contain all children of a topic.
ANY	Selects documents that contain at least one of the children of a topic.

Modifiers

Modifiers affect the behavior of operators. The following table describes each modifier..

Table 9-6: Modifiers

Operator Name	Description
CASE	Performs a case-sensitive search.
MANY	Counts the density of words or phrases in a document and produces a relevance-ranked score for the retrieved documents.
NOT	Excludes documents that show evidence of the specified word or phrase.
ORDER	Specifies the order in which search elements must occur.

Operator Precedence Rules

The Verity search engine uses precedence rules to determine how operators can be assigned. These rules state that some operators rank higher than others when assigned to topics, and affect how document selections are performed.

The following table describes how precedence rules apply to operators.

Table 9-7: Precedence rules

Operator	Precedence	How Precedence is Determined
AND OR ACCRUE	Highest precedence	The concept operators take the highest precedence over the other operators. Thus, subtopics of topics using these operators can be assigned any of the operators listed below under "incremental precedence" or "lowest precedence."
ALL PARAGRAPH SENTENCE NEAR NEAR/N PHRASE ANY	Incremental precedence (in descending order)	The proximity operators refer to incremental ranges which exist within a document. Subtopics of topics using these operators can be assigned their next lowest operator in the precedence order. Thus, a phrase takes precedence over a word; a sentence takes precedence over a phrase or a word; and a paragraph takes precedence over a sentence, a phrase, or a word.

Table 9-7: Precedence rules

Operator	Precedence	How Precedence is Determined
WORD STEM SOUNDEX WILDCARD THESAURUS	Lowest precedence	The evidence operators reside at the lowest level in a topic structure. Because evidence operators are used with words contained in documents, these operators all have the same precedence.

To avoid a precedence violation, do not use ANY or ALL in a parent topic whose child topic includes a concept operator (AND, OR, ACCRUE). Topics that use ANY or ALL cannot have variable weights assigned to them, so you cannot use these operators in a parent topic with any child topic that allows variable weights (such as AND, OR, ACCRUE). Topics using ANY and ALL limit evaluation to present or not present (a score of 0.00 or 1.00). If the criteria are met, the children of these topics get an automatic score of 1.00; if the criteria are not met, the children of these topics get an automatic score of 0.00; so it is not meaningful to assign these children variable weights such as 0.80.

Sample Topic Outlines

The following are the same topics as you would create them in a topic outline file:

```
$Control:1
art <Accrue>
*performing-arts <Or>
**0.80 "drama"
**0.50 "theater"
**0.80 'dance'
*film <And>
**0.90 "cinema"
**0.90 "documentary"
**newsreel <Filter>
/definition="DATE  >= 05/01/96"
*film-makers <Accrue>
**"Woody Allen"
*film-making <Paragraph>
**"direct"
**"produce"
*visual-arts <Accrue>
**sculpture <In>
```

```
/zonespec="title"
**painters <Filter>
/definition="Title MATCHES Famous Painters"
**<Thesaurus>
/wordtext="paint"
literature <Accrue>
*novels <Near>
**0.80 "Proust"
**0.80 "Remembrance" <Case>
*american-novel <Sentence>
**"American"
**"novel"
history <Accrue>
*<Wildcard>
/wordtext="histor*"
music <Accrue>
*jazz
**"bebop"
**<Not> "fusion"
*classical
**"Italian opera"
$$
```

Operator Reference

Each operator is listed below alphabetically. Examples for many of these operators can be found in the topic outline in the previous section.

ACCRUE Operator

Selects documents that include at least one of the search elements you specify. Valid search elements are two or more words or phrases. Selected documents are relevance-ranked.

The ACCRUE operator scores selected documents according to the presence of each search element in the document using a "the more, the better" approach: the more search elements found in the document, the better the document's score. Several examples of the ACCRUE operator appear in the sample outline file in the previous section, "Sample Topic Outlines."

ALL Operator

Selects documents that include all of the search elements you specify. Unlike the ACCRUE operator, you cannot assign weights when you use the ALL operator.

AND Operator

Selects documents that contain all of the search elements you specify. Documents selected using the AND operator are relevance-ranked. The example in "Sample Topic Outlines" shows how the AND operator might be used with the topic "film." In the example, only those documents that contain both search words and a date greater than or equal to 05/01/96 are selected and ranked according to their score.

ANY Operator

Selects documents include at least one of the search elements you specify. Unlike the ACCRUE operator, you cannot assign weights when you use the ANY operator.

CONTAINS Operator

Selects documents by matching the word or phrase you specify with values stored in a specific document field. When you use the CONTAINS operator, you specify the field name to search, and the word or phrase to search for.

With the CONTAINS operator, the words stored in a document field are interpreted as individual, sequential units. You may specify one or more of these units as search criteria. To specify multiple words, each word must be sequential and contiguous, and must be separated by a blank space. Use CONTAINS with the FILTER operator.

The syntax for CONTAINS is the same as that for MATCHES. See the example for MATCHES under the topic "visual arts" in "Sample Topic Outlines." The example assumes that the field TITLE has been created for the collection.

The CONTAINS operator does not recognize non-alphanumeric characters. The CONTAINS operator interprets non-alphanumeric characters as a space and treats the separated values as individual

units. For example, if you have defined a slash (/) as a valid character, and you enter search criteria that include this character, as in OS/2, "OS" and "2" are treated as individual units.

Note that the CONTAINS operator does not refer to the *style.lex* file for the definition of which characters are included in a word.

ENDS Operator

Selects documents by matching the character string you specify. Use ENDS with the FILTER operator. The syntax for ENDS is the same as that for MATCHES. See the example for MATCHES under the topic "visual arts" in "Sample Topic Outlines." The example assumes that the field TITLE has been created for the collection.

= (EQUALS) Operator

Selects documents whose document field values are exactly the same as the search string you specify. Use EQUALS with the FILTER operator. The syntax for EQUALS is the same as that for GREATER THAN OR EQUAL TO. See the example for GREATER THAN OR EQUAL TO under the topic "film" in "Sample Topic Outlines." The example assumes that the field DATE has been created for the collection.

FILTER Operator

The special FILTER operator is used in conjunction with the relational operators to do field searches. See the example under the topic "visual-arts" in "Sample Topic Outlines" for the proper syntax.

> (GREATER THAN) Operator

Selects documents whose document field values are greater than the search string you specify. Use GREATER THAN with the FILTER operator. The syntax for GREATER THAN is the same as that for GREATER THAN OR EQUAL TO. See the example for GREATER THAN OR EQUAL TO under the topic "film" in "Sample Topic Outlines." The example assumes that the field DATE has been created for the collection.

>= (GREATER THAN OR EQUAL TO) Operator

Selects documents whose document field values are greater than or equal to the search string you specify. Use GREATER THAN OR EQUAL TO with the FILTER operator. See the example under the topic "film" in "Sample Topic Outlines." The example assumes that the field DATE has been created for the collection.

< (LESS THAN) Operator

Selects documents whose document field values are less than the search string you specify. Use LESS THAN with the FILTER operator. The syntax for LESS THAN is the same as that for GREATER THAN OR EQUAL TO. See the example for GREATER THAN OR EQUAL TO under the topic "film" on "Sample Topic Outlines." The example assumes that the field DATE has been created for the collection.

<= (LESS THAN OR EQUAL TO) Operator

Selects documents whose document field values are less than or equal to the search string you specify. Use LESS THAN OR EQUAL TO with the FILTER operator. The syntax for LESS THAN OR EQUAL TO is the same as that for GREATER THAN OR EQUAL TO. See the example for GREATER THAN OR EQUAL TO under the topic "film" on "Sample Topic Outlines." The example assumes that the field DATE has been created for the collection.

IN Operator

Selects documents that contain specified values in one or more document zones. A document zone represents a region of a document, such as the document's summary, date, or body text. The IN operator only works if document zones have been defined in your collections. If you use the IN operator to search collections for which zones are not defined, no documents will be selected. In addition, the zone name you specify must match the zone names defined in your collections. Consult your collection administrator to determine which zones have been defined for specific collections. The example in "Sample Topic Outlines" shows how IN might be used with the word "sculpture" and the TITLE zone.

MATCHES Operator

Selects documents by matching the character string you specify with values stored in a specific document field. When you use the MATCHES operator, you specify the field name to search, and the word, phrase, or number to search for.

Unlike the CONTAINS operator, the search criteria you specify with a MATCHES operator must match the field value exactly for a document to be selected. With the MATCHES operator, any occurrence of a search string that appears as a portion of a value is not selected; only values matching the entire search string are selected.

You can use question marks (?) to represent individual variable characters within a string, and asterisks (*) to match multiple variable characters within a string.

Use MATCHES with the FILTER operator. The example in "Sample Topic Outlines" shows how MATCHES might be used with the phrase "famous painters" and the TITLE field. The example assumes that the field TITLE has been created for the collection.

NEAR Operator

Selects documents containing specified search terms within close proximity to each other. Document scores are calculated based on the relative number of words between search terms. For example, if the search expression includes two words, and those words occur next to each other in a document (so that the region size is two words long), then the score assigned to that document is 1.00. Thus, the document with the smallest region containing all search terms always receives the highest score. Documents whose search terms are not within 1000 words of each other are not selected, since the search terms are probably too far apart to be meaningful within the context of the document.

The NEAR operator is similar to the other proximity operators in the sense that the search words you enter must be found within close proximity of one another. However, unlike other proximity operators, the NEAR operator calculates relative proximity and assigns scores based on its calculations.

The example in "Sample Topic Outlines" shows how NEAR might be used with the topic "novels."

NEAR/N Operator

Selects documents containing two or more words within N number of words of each other, where N is an integer. Document scores are calculated based on the relative distance of the specified words when they are separated by N words or less. Documents containing the specified words separated by more than N words are not selected. For example, if the search expression NEAR/5 is used to find two words within five words of each other, a document that has the specified words within three words of each other is scored higher than a document that has the specified words within five words of each other.

The N variable can be an integer between 1 and 1,024, where NEAR/1 searches for two words that are next to each other. Note that if N is 1,000 or above, you must specify its value without commas, as in NEAR/1000.

The NEAR/N operator is similar to the other proximity operators in the sense that the search words you enter must be found within a close proximity of one another. However, unlike other proximity operators, the NEAR/N operator assigns scores based on relative proximity.

OR Operator

Selects documents that show evidence of at least one of your search elements. Documents selected using the OR operator are relevance-ranked. The example in "Sample Topic Outlines" shows how you might use OR with the topic "performing-arts."

PARAGRAPH Operator

Selects documents that include all of the search elements you specify within a paragraph. Valid search elements are two or more words or phrases. You can specify search elements in a sequential or a random order. Documents are retrieved as long as search elements appear in the same paragraph. The example in "Sample Topic Outlines" shows you how you might use PARAGRAPH with the topic "film-making."

PHRASE Operator

Selects documents that include a phrase you specify. A phrase is a grouping of two or more words that occur in a specific order. You

must use the PHRASE operator when you enter more than one word in the evidence field. Words with the PHRASE operator are displayed in double quotes. The example in "Sample Topic Outlines" shows "Woody Allen" and "Italian opera" as uses of the PHRASE operator.

SENTENCE Operator

Selects documents that include all of the words you specify within a sentence. You can specify search elements in a sequential or a random order. Documents are retrieved as long as search elements appear in the same sentence. The example in "Sample Topic Outlines" shows how you how you might use SENTENCE with the topic "american-novel."

SOUNDEX Operator

Selects documents that include one or more words that "sound like," or whose letter pattern is similar to, the word specified. Words have to start with the same letter as the word you specify to be selected. For example, when you use SOUNDEX with "sale," the documents selected will include words such as "sale," "sell," "seal," "shell," "soul," and "scale." Documents are not relevance-ranked unless the MANY modifier is used.

STARTS Operator

Selects documents by matching the character string you specify with the starting characters of the values stored in a specific document field. Use STARTS with the FILTER operator. The syntax for STARTS is the same as that for MATCHES. See the example for MATCHES under the topic "visual arts" in "Sample Topic Outlines." The example assumes that the field TITLE has been created for the collection.

STEM Operator

Selects documents that include one or more variations of the search word you specify. Words with the STEM operator are displayed in single quotes. In the example in "Sample Topic Outlines," the word "dance" is used with the STEM operator. Documents selected will therefore include words such as "dances," "danced," "and "dancing," as well as "dance."

SUBSTRING Operator

Selects documents by matching the character string you specify with a portion of the strings of the values stored in a specific document field. The characters that comprise the string can occur at the beginning of a field value, within a field value, or at the end of a field value. The syntax for SUBSTRING is the same as that for MATCHES. See the example for MATCHES under the topic "visual arts" in "Sample Topic Outlines." The example assumes that the field TITLE has been created for the collection.

THESAURUS Operator

Selects documents that contain one or more synonyms of the word you specify. For example, when you use the word "altitude" with the THESARUS operator, the documents selected will include words such as "height" and "elevation." Documents are not relevance-ranked unless the MANY modifier is used.

WILDCARD Operator

Selects documents that contain matches to a character string containing variables. The WILDCARD operator lets you define a search string with variables, which can be used to locate related word matches in documents. The example in "Sample Topic Outlines" shows how you might use the string "histor*" to search for words such as "history," "historical," and "historian." Documents are not relevance-ranked unless the MANY modifier is used.

Using Wildcard Special Characters

You can use the following wildcard characters to represent variable portions of search strings with the WILDCARD operator.

Table 9-8: Wildcard Special Characters

Character	Function
?	Specifies one of any alphanumeric character, as in ?an, which locates "ran," "pan," "can," and "ban." Note that it is not necessary to specify the WILDCARD operator when you use the question mark. The question mark is ignored in a set ([ ]) or in an alternative pattern ({ }).

Table 9-8: Wildcard Special Characters

Character	Function
*	Specifies zero or more of any alphanumeric character, as in corp*, which locates "corporate," "corporation," "corporal," and "corpulent." Note that it is not necessary to specify the WILDCARD operator when you use the asterisk, and you should not use the asterisk to specify the first character of a wildcard string. The asterisk is ignored in a set ([ ]) or in an alternative pattern ({ }).
[ ]	Specifies one of any character in a set, as in <WILDCARD> `c[auo]t`, which locates "cat," "cut," and "cot." Note that you must enclose the word which 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." Note that you must enclose the word which 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." Note that 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."

Searching for Non-alphanumeric Characters

Remember that you can only search for non-alphanumeric characters if the style.lex file used to create the collections you are searching is set up to recognize the characters you want to search for. Consult your collection administrator for information.

Searching for Wildcard Characters as Literals

The wildcard characters listed above are interpreted as wildcard characters, not literal characters, unless they are delimited by a backslash (\). If you want a wildcard character to be interpreted as a literal in a wildcard string, you must precede the character with a backslash. For example, to match the literal asterisk (*) in a wildcard string, you delimit the character as follows:

```
<WILDCARD> a\*
```

Searching for Special Characters as Literals

The following non-alphanumeric characters perform special, internal functions, and by default are not treated as literals in a wildcard string:

- comma ,
- left and right parentheses ( )
- double quotation mark "
- backslash \
- at sign @
- left curly brace {
- left bracket [
- less than sign <
- backquote `

To interpret special characters as literals, you must surround the whole wildcard string in backquotes (`). For example, to search for the wildcard string "a{b", you surround the string with backquotes, as follows:

```
<WILDCARD> `a{b`
```

To search for a wildcard string that includes the literal backquote character (`), you must use two backquotes together and surround the whole wildcard string in backquotes (`), as follows:

```
<WILDCARD> `*n``t`
```

Note that you can only search on backquotes if the style.lex file used to create the collections you are searching is set up to recognize the backquote character. Consult your collection administrator for information.

WORD Operator

Selects documents that include one or more instances of a word you specify. Words with the WORD operator are displayed in double quotes. The example in "Sample Topic Outlines" displays many instances of the WORD operator.

Modifier Reference

Modifiers further specify the behavior of operators. For example, you can use the CASE modifier with an operator to specify that the case of the search word you enter be considered a search element as well. Modifiers include CASE, MANY, NOT, and ORDER, which are described below.

CASE Modifier

Use the CASE modifier with the WORD or WILDCARD operator to perform a case-sensitive search, based on the case of the word or phrase specified.

By default, documents containing any occurrences of a search word or phrase are retrieved regardless of case. To use the CASE modifier, you simply enter the search word or phrase as you wish it to appear in retrieved documents - in all uppercase letters, in mixed uppercase and lowercase letters, or in all lowercase letters. The example in "Sample Topic Outlines" shows how you might use the word "Remembrance" with the CASE modifier in order to refer to the first word of Proust's novel, Remembrance of Things Past.

MANY Modifier

Counts the density of words, stemmed variations, or phrases in a document, and produces a relevance-ranked score for retrieved documents. The more occurrences of a word, stem, or phrase proportional to the amount of document text, the higher the score of that document when retrieved. Because the MANY modifier considers density in proportion to document text, a longer document that contains more occurrences of a word may score lower than a shorter document that contains fewer occurrences.

The MANY modifier can be used with the following operators: WORD, WILDCARD, STEM, SOUNDEX, PHRASE, SENTENCE, PARAGRAPH and THESAURUS.

Note that the MANY modifier cannot be used with AND, OR, ACCRUE, or relational operators.

NOT Modifier

Use the NOT modifier with a word or phrase to exclude documents that show evidence of that word or phrase. The example in "Sample Topic Outlines" shows how you might use the NOT modifier to retrieve documents that mention "bebop" but not "fusion."

ORDER Modifier

Use the ORDER modifier to express the order in which search elements must occur. If search values do not occur in the specified order in a document, the document is not selected. Always place the ORDER modifier just before the operator.

Note that you can only use the ORDER modifier with the operators ALL, PARAGRAPH, SENTENCE, and NEAR/N.

Weights and Document Importance

This section describes assigning weights to search criteria in topics, and the affect of weights on selected documents. The specific information covered includes the following:

- Which operators accept weights
- How weights affect importance
- Assigning weights
- Topic scoring and document importance

Topic Weights

When processing a search agent, the Verity search engine calculates a score for each selected document behind the scenes. A document score can be in the range from 1.0 to 0.01. The higher a document's score, the more relevant it is. Using the score assignments for documents selected by a search agent, Verity applications can present relevance-ranked results in descending order to application users.

The ranking of documents is determined by the elements which comprise your search criteria. Document ranking can be affected depending on whether the search criteria includes topics, and whether topics include weights.

When creating topics, you can assign weights to the topic structure to indicate the relative importance of specific aspects of the topic definition. For example, you may be interested in two related subjects, but one subject is more important than another. Note that you do not have to assign weights when you compose topics because default weights are assigned as appropriate when a topic set is indexed. However, by assigning weights you can fine-tune the importance of things you are looking for.

Which Operators Accept Weights

Weights are used in conjunction with operators to compute scores for parent and child topics during a search. The weight you assign to a topic child can be a number between 0.01 and 1.00. A child's weight indicates its importance relative to the other children that have been defined for its parent. The higher a child's weight, the more important that child is considered to be with respect to its siblings.

Weights can only be assigned to the children of topics which use the concept operators, as follows:

- AND
- OR
- ACCRUE

Topics which use the proximity operators SENTENCE and PARAGRAPH, cannot be assigned a weight. These operators assume a simple "yes" or "no" presence for their children.

Note that if a topic assigned a proximity operator is, in turn, the child of a topic which has been assigned a concept operator, such as the AND operator, that child can be assigned a weight.

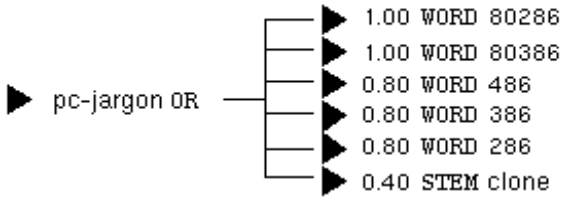
It is not mandatory that you assign weights to the children of a topic just because the operator can accept weighted children. When weights are not assigned, the child has an automatic weight assignment based on its operator. Children of topics using AND and OR operators assume a weight of 1.00, and children of topics using the ACCRUE operator assume a weight of 0.50. If these operators are changed-for example, if an OR operator is changed to an ACCRUE operator-the weights of children which have not been specifically assigned a weight change accordingly. Thus, if an unweighted child of an AND topic has an assumed weight of 1.00, this assumed weight changes to 0.50 if the operator is changed to ACCRUE.

If you assign a variable weight to a topic child, then change the operator used with the parent to one which does not accept weighted children, such as the SENTENCE operator. The Verity search engine will automatically assume a weight of 1.00 while this operator is in effect. If the operator is subsequently changed to one which accepts variable-weighted children, the previously-assigned variable weights will become effective once again.

How Weights Affect Importance

When you assign a weight to the child of a topic which uses a concept operator, you specify the relative contribution of that child to the overall score produced by a topic. The higher the weight you assign to the child, the higher selected documents which contain that child will appear in the list of results. Thus, weights directly affect the importance, or ranking, of selected documents.

For example, assume you have the following topic:

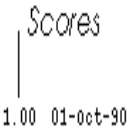


The evidence topics 80286 and 80386 (which describe the microprocessors used in PC products) have an automatic weight assignment of 1.00. The evidence topics 486, 386, and 286 have a relatively high probability of referring to their parent topic, so these evidence topics are assigned weights of 0.80. The evidence topic clone may or may not refer to PC clones at all; therefore, this evidence topic is assigned a weight of 0.40.

A search agent using this topic and its assigned weights might produce the following scores for the matched documents:


```
1.00 01-Oct-90 New Toshiba Portable Desktop Computers Offer a Serious Alternative for Desktop Users
1.00 14-Feb-90 Technology: 'Chip Set' Unveiled for Use in Making Faster Computers
0.80 13-Feb-91 CMS Enhancements Inc. Unveils New Products
0.80 01-Oct-90 Top Selling Microsoft Windows Applications Now Available in One Convenient Package
0.80 01-Oct-90 KeyCorp Successful Bidder to Acquire New York State Divisions of Empire of America Federal
0.80 15-Feb-90 Health: U.S. Birth Control Lags
```

If you change the weights of each evidence topic, the importance of your selection results are affected, as well. In this example, if you change the weights of the evidence topic 486 to 0.60, the evidence topic 386 to 0.45, the evidence topic 286 to 0.35, and the evidence topic clone to 0.20, your selected document scores will change as follows:



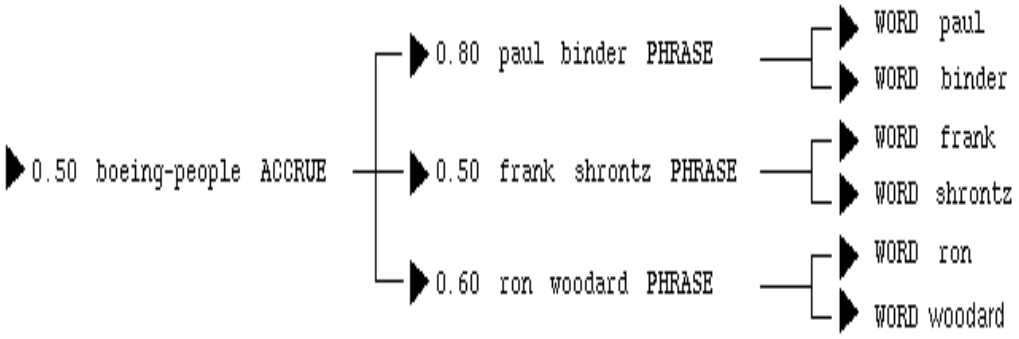

```
1.00 01-Oct-90 New Toshiba Portable Desktop Computers Offer a Serious Alternative for Desktop Users
1.00 14-Feb-90 Technology: 'Chip Set' Unveiled for Use in Making Faster Computers
0.60 13-Feb-91 CMS Enhancements Inc. Unveils New Products
0.60 01-Oct-90 KeyCorp Successful Bidder to Acquire New York State Divisions of Empire of America Federal
0.60 15-Feb-90 Health: U.S. Birth Control Lags
0.45 01-Oct-90 Top Selling Microsoft Windows Applications Now Available in One Convenient Package
```

Assigning Weights

When you assign a weight to a child, keep in mind that the weight you use reflects the importance of a child to its parent topic. The matched documents will be ranked by importance to the search; thus, your selection results are directly affected by the weights you assign. If you change a weight, your selection results will be changed, as well.

Example:

The topic boeing-people includes three weighted children, binder, shrontz, and woodard, as shown below.



These subtopics are assigned various weights, as follows: the child binder is assigned a weight of 0.80, since this child is considered to be the most important of the three. The subtopic hitsman is assigned a "median" weight of 0.50, since this child is reasonably important with respect to the other two children. The subtopic johnson is assigned a low weight of 0.30, since this child is considered to be the least important with respect to the other children.

When the topic boeing-people is used for a search, the Verity search engine assumes that if the phrase "Paul Binder" is located within a document, there is a high probability that the document will be relevant to a search which uses the topic boeing-people. Documents which contain the phrase "Frank Shrontz" will be reasonably relevant to this search; documents which contain the phrase "Ron Woodard" will be the least relevant.

Because the topic boeing-people has been assigned the ACCRUE operator, the documents displayed at the top of the results list will be those which contain the greatest number of children; therefore all documents with references to all three people will be given the most importance. Documents which contain just one name will be selected in an order that reflects the weights of each child. Thus, because the binder child has the highest weight, documents which include only one individual will be ranked by those which refer to Paul Binder first, followed by Frank Shrontz, and finally Ron Woodard.

Automatic Weight Assignments

When you create a child, the Verity search engine automatically assigns a default weight of 0.50 for children of topics which use the ACCRUE operator. A weight of 1.00 is assigned automatically to children of topics which use the AND or OR operators. These default weights can be manually adjusted up or down, as described in "Changing Weights" in this section. When you create a evidence topic off of a topic which uses a proximity operator, default weight of 1.00 is assigned, and it cannot be changed.

Tips for Assigning Weights

When initially assigning weights, start with a weight of 0.50 for children of ACCRUE topics, and 1.00 for children of all other topics.

When assigning weights to children of topics which use the ACCRUE operator, you may select more relevant results if the children do not have overly high weights. For example, assigning all of the children of an ACCRUE topic weights of 1.00 will cause all documents to have equal importance, regardless of how many of the children are present within the documents. The Verity search engine will assign equal importance to all documents containing only one child as well as for documents which contain all children, so you will not be able to distinguish between these documents when you view the selection results.

Assign weights between 0.80 and 0.20 for the best selection results.

Changing Weights

Once you have assigned weights to children, you can test these weights by running a search using the parent topics to see if the documents you want are selected. If you find that you need to change the weights, you can edit the existing weight assigned to that subtopic or evidence topic. Note that when you edit topic definitions in the topic outline file, you must rebuild the topic set using mktopics. For complete information about using mktopics, refer to your Verity application's administration guide.

Topic Scoring and Document Importance

When you use a topic to perform a search, the search agent starts its analysis by considering the evidence topics for that topic. If the evidence topic is present, it is given 1.00 score and is considered relevant to the search. If the evidence topic is absent, it is given a 0.00 score and is considered irrelevant to the search. If the evidence topics are weighted, the scores of the evidence topics are multiplied by the weights, then combines the resulting products in a manner specified by the operator of the parent topic. If this parent topic is, in turn, the child of another topic which is being searched, its score is multiplied by its assigned weight, and the resulting product is combined with the products of its siblings in a manner specified by the operator assigned to the parent topic. This process continues until the parent topic is reached.

The operators you use determine how parent and child scores contribute to the importance of a selected document. As each child in the topic is given an importance score, the following calculations are performed:

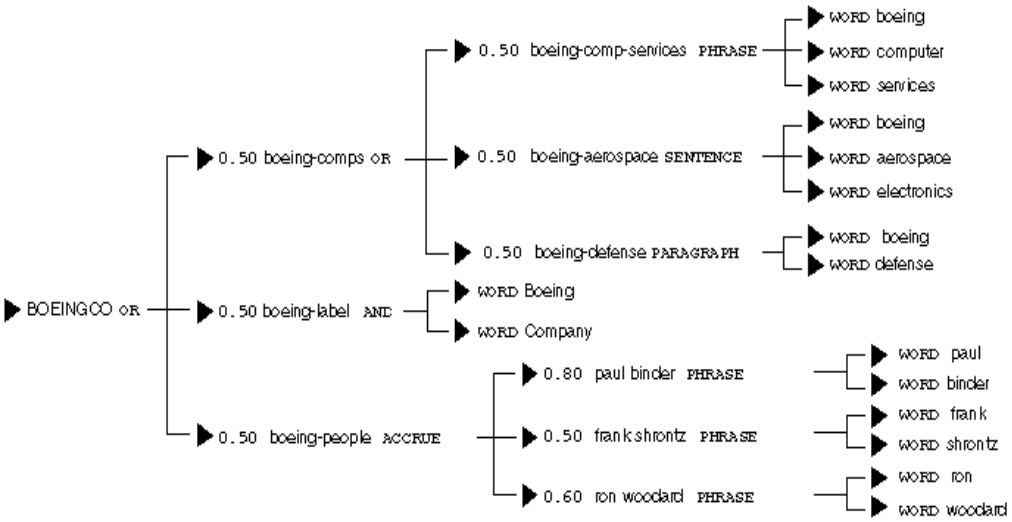
- If a topic uses an ACCRUE operator, the highest ranking result is taken from the products of each child's weight and score, then adds a little to the score for each child which is present in the document.
- If a topic uses an AND operator, the products of each child's own weight and score are compared, and the lowest product (the minimum) is taken as the score.
- If a child uses an OR operator, the products of each child's weight and score are compared, and highest product (the maximum) is taken as the score.
- If a child uses a proximity operator (PHRASE, SENTENCE, or PARAGRAPH), or a relational operator, the child receives a score of 1.00 if the topic is present, and a score of 0.00 if the topic is not present.
- An evidence topic receives a score of 1.00 if it is present, and no score of 0.00 if it is not present.

Once the final calculations for the parent topic have been performed, a matched document becomes available to the Verity application so that users can view it with its highlights.

Example:

The following example provides a breakdown of how evidence topics and subtopics are calculated to illustrate the process by which importance is assigned to selected documents.

In the following illustration, the parent topic BOEINGCO is being used in a search.



The evidence topics of each subtopic are first checked against the documents to determine if they are present. Evidence topics that are present are assigned scores of 1.00; evidence topics that are absent are assigned a score of 0.00.

The operators at the next level of a topic structure are used to combine the scores of the evidence topics. Because the operators

at this level are all proximity operators (thus, no weights assigned), they all produce scores that are either 0.00 or 1.00.

For example, assume that the following evidence topics appear within a given document:

- The evidence topic "Boeing Computer Services" appears within a phrase
- The evidence topic "Boeing Defense" appears within a paragraph The evidence topic "Boeing Company" appears within the document
- The evidence topic "Ron Woodard" appears within a phrase

The other evidence topics are only partially present, or are absent. The following table shows how the presence or absence of these evidence topics affect topic scores. Note that the score for each topic reflects the presence of all related evidence topics, based on the operators which have been assigned to the parent topics.

Table 9-9: Evidence Topics and Scores

Topic	Evidence topic	Evidence topic Present	Evidence topic Absent	Topic Score
boeing-comp-services	boeing computer services	1 1 1		1
boeing-aerospace	boeing aerospace electronics	1	 1 1	0
boeing-defense	boeing defense	1 1		1
boeing-label	boeing company	1 1		1
paul-binder	paul binder		1 1	0
frank-shrontz	frank shrontz		1 1	0
ron-woodard	ron woodard	1 1		1

Given the above topic scores, the operators at the next level of topics in the structure are calculated as follows:

- The subtopic boeing-comps, which uses the AND operator, has a score of 0.50.
- The subtopic boeing-people, which uses the ACCRUE operator, has a score of 0.50.

Finally, the topic BOEINGCO, which uses the OR operator, compares the products of each child's weight and score, and takes the highest

product (the maximum) as its score. The selected document is thus scored as 0.50.

This process is repeated for each document. The documents are sorted by the scores of the BOEINGCO topic, and displayed in ranked order.

Designing Topics

This section discusses methodologies you can use to design effective topics. You can apply the methodologies and strategies described here whether you plan to compose topics using a topic outline file or one of the Verity clients. The information in this section includes the following:

- Preparing your topic design
- Topic design strategies
- Designing the initial topic

Preparing Your Topic Design

As you prepare your topic design, consider the naming conventions you will use. Your topic names should help identify the subject matter of the kinds documents you want to find.

To ensure the best search performance, use alphanumeric characters (A through Z, and 0 through 9) for topic names. You can also use foreign characters whose ASCII value is greater than or equal to 128, as well as these symbols: $ (dollar sign), % (percentage sign), ^ (circumflex), + (plus sign), - (dash), and _ (underscore). Using other non-alphanumeric characters, could cause misinterpretation of the topic name and affect results.

Understanding Your Information Needs

You should have an understanding of the subject areas to be addressed by your topic design and be familiar with the search requirements of users at your site. The next step is to understand your informational needs, as well as the document types to be searched.

In planning your initial topic design, keep in mind that you are developing a strategy, and the topics you define are the tactics you will use to implement that strategy.

As you develop your strategy, try to answer the following questions:

- What do you wish to gain by using a Verity search agents?
- What issues are to be solved by Verity search agents?
- Who will use search agents?
- What kind(s) of source material will be used?
- What kinds of searches will be performed?
- How are searches currently being performed?

Consider the topics you define as questions to be asked. Just as you might ask a reference librarian at your local library for information relating to a subject area, the topics you create should pose questions when creating Verity search agents.

When considering your strategy, and how Verity search applications will be implemented to provide a solution, keep in mind that a topic you design performs several roles, as follows:

- A librarian
- A research assistant
- An information repository
- A knowledge base

Understanding Your Documents

To build effective topics, you must have a good understanding of the types of documents being used as information sources. For example, your documents may consist of one or more of the following types of information:

- Letters
- Memos
- Reports
- Articles

Collect representative samples of the types of documents to be searched. Note common characteristics you will need to apply to the topics you design. For example, if your documents contain important terms, acronyms, or jargon, highlight them so you can create topics that search for this text.

As you collect your document samples, identify their sources-whether they are internal sources, such as internal auditing reports;

or external sources, such as electronic mail messages from outside organizations. This information will enable you to define the subtopics for top-level topics.

Using Scanned Data

If your documents are scanned into electronic files using an OCR facility, determine whether the document files will be reviewed for accuracy prior to indexing. If scanned files are reviewed, consult with reviewers to ensure that standards are applied to terms, acronyms, and jargon. If scanned files are not reviewed, note possible variations that may occur. You can develop a topic that uses an OR operator to include variations.

Categorizing Document Samples

Once you have collected your representative document samples and have performed an initial analysis of their contents, you may want to categorize them further. The categorization process can help you to define the top-level topics and children contained in your topic design, and help determine the operators and weights to assign.

Following are categorization examples:

- Geographic location
- Sit
- Project
- Subject area
- Date

The categorization process can help you understand the common, meaningful elements which exist in your information sources. For example, if you categorize your information by date (such as a month), it makes sense to create topics that use relational operators, such as EQUALS.

Topic Design Strategies

Once you have an understanding of your documents, you are ready to choose a topic design strategy. There are two topic design strategies

- The "top-down" strategy considers the major subject classifications first, followed by classifications of increasing detail.
- The "bottom-up" strategy considers the detailed areas first, followed by classifications which group each detailed area by a more generalized subject.

Top-Down Design

A top-down strategy assumes you are designing a topic from the top-level topics down through the individual evidence topics of each subtopic. To design from the top down, you must adopt a taxonomy, or scientific classification approach, to creating a topic, as follows:

- Top-level topics: use general headings to identify the subject area
- Subtopics: use more specific headings to identify the primary groupings within the subject area, as well as topics which are increasingly more specific.
- Evidence topics: use important terms, acronyms, or jargon, to define the subject.

A top-down design works best when you have clearly-defined requirements. This approach is also ideal if your set of searchable documents is constantly growing or changing. With this strategy, for example, you are likely to define subjects which may not yet be evident in your information sources. Keep in mind that you can always add new topics, if you find that a number of new documents contain information which are not identified in your topic design.

If your information sources (that is your set of indexed documents) changes constantly, specific subjects within documents may be missed, especially at the lowest levels. So, you should periodically analyze the information being selected by your topics to ensure that topics critical to your application are current, and the appropriate information is being found.

Bottom-Up Design

A bottom-up strategy assumes you are designing a topic from the individual evidence topics up through the top-level topics which will be defined. With this strategy, your topic design objective is to select documents containing information similar to your lower-level topics.

When you use a bottom-up design, you can start with a document which contains a good representative sample of the words or phrases you want to search for. Then you can group these words by successively higher classifications.

A bottom-up design works best when you have documents which are representative of many other documents that contain similar information. This approach is also useful when your information sources are not subject to many changes or additions.

Keep in mind that topic designs based on the contents of specific documents may miss related subject areas in other documents. For example, if a name is used in the sample document and that name changes in other documents, the new name may be missed in searches.

In addition, the bottom-up strategy implies that your topic design is tuned to the specific document set being used to develop

your topics. These documents may not be representative of all documents contained in your information sources. So, you should

periodically review the effectiveness of your searches.

Designing the Initial Topic

When you have decided whether to use the top-down approach or the bottom-up approach for your initial topic design, it can be helpful to create a topic outline to identify the topic levels to be defined.

Outlining a Topic

Making a topic outline can help you determine how information will be categorized at the various levels within a topic. You can use a topic outline with the top-down or the bottom-up design approach, but it is particularly useful for the top-down approach. We recommend that every topic you build be developed as an outline first, so that you can understand the relationships between topics and subtopics, and organize them to be the most useful.

A topic outline helps you understand how information might be searched for by the people who use Verity search agents at your site. You can use a topic outline to fine-tune the information specified by topics and subtopics to pinpoint document selection. Try to do the following as you develop a topic outline:

- Identify the specific areas of information people will use when performing searches.
- Identify any related subtopics which may be grouped as children under a parent topic .
- Consider the initial level of detail to be covered by your topic design.

Keep the scope of your topic outline relatively small to begin with. A smaller, simpler topic outline is easier to define, and you can always add additional information later. As you develop your topic outline, determine how many levels your topic design will include.

Top-Down Topic Outline Example

Developing a top-down topic outline involves three steps.

- Establishing an information hierarchy
- Establishing individual search categories
- Establishing the topics to be built

As you work through these steps, you should meet with the people who use Verity search agents at your site to develop a topic outline that best meets their search needs, as described below.

Step One: Establishing an Information Hierarchy

Talk to the people at your site to learn what types of documents contain the information they need.

For example, assume you are developing a topic design for people in the medical industry to find information relating to current drug testing. Based on discussions with the people who will use Verity search agents at your site, you learn that the following types of documents are prime sources of current drug testing information:

- Research reports
- Product literature

These documents form the information sources to be searched by Verity search agents.

Step Two: Establishing Individual Search Categories

Review the documents that will form the information sources at your site. Look for ways to categorize documents.

In our example, a review of the medical research reports and product literature shows information contained in these documents is divided into several categories. You determine that the following categories will be used to define the top-level topics in your topic design:

- Lab reports
- Clinical trials, data, or research
- Product literature

Step Three: Establishing the Topics to be Built

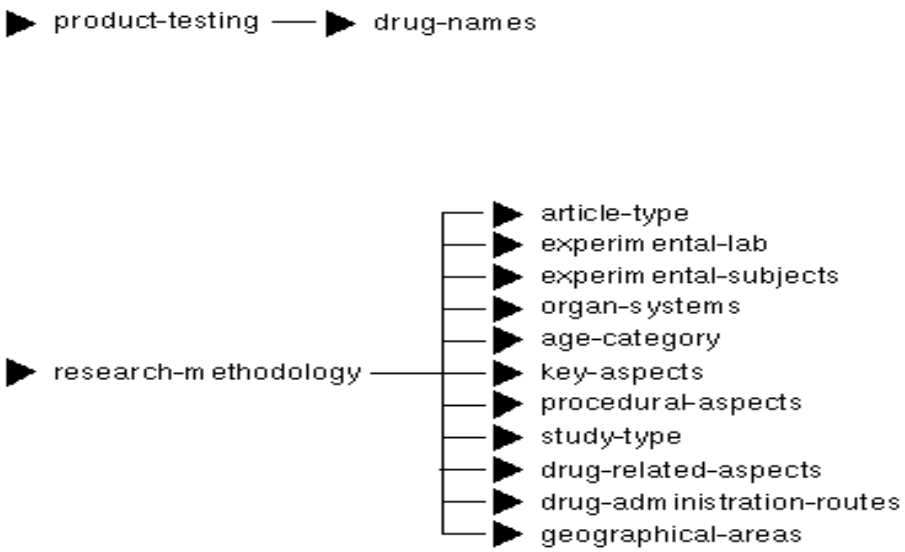
Discuss categories you define with the people who create Verity search agents at your site to determine the most important concepts that selected documents should contain, and to determine the top-level topics you need to develop for each category.

For example, you determine that the category "clinical trials" includes the following top-level topics:

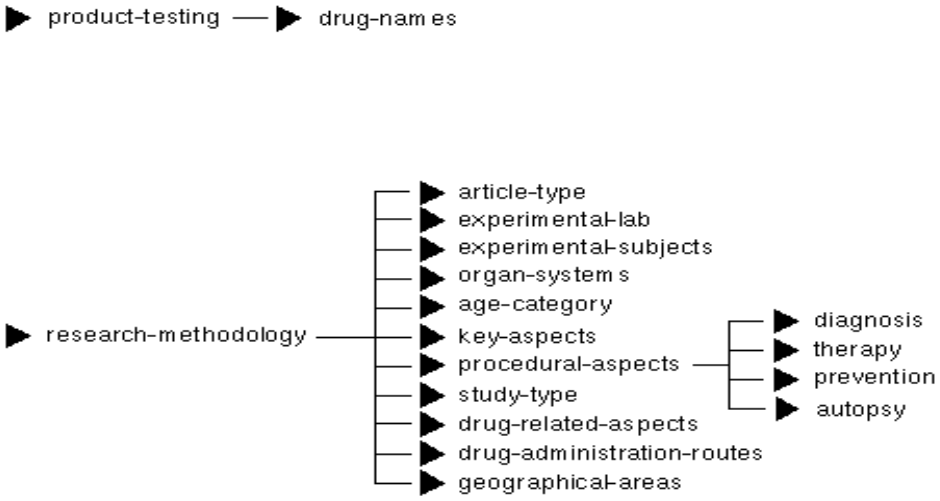
product-testing

research-methodology

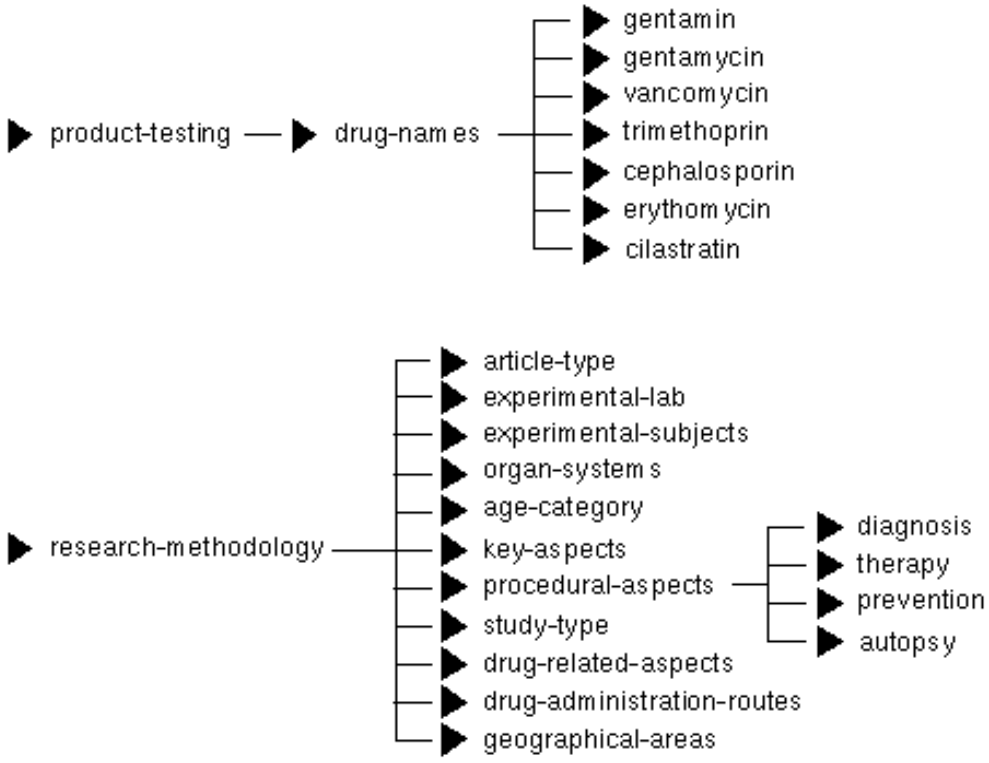
Within these top-level topics, for example, the following subtopics are identified by subject-area experts:



Once these topics are classified, you consult the people who use Verity search agents at your site to determine subtopics. Following is an example of subtopics classified as children for the topic procedural-aspects:



As the topic outline is defined, you consult the people who use Verity search agents at your site to ensure the topics select meaningful documents. In the next example, a topic called drug-names enables the users at your site to search clinical trials data for drugs, based on their names.



Bottom-Up Topic Outline Example

Developing a bottom-up topic outline involves three steps.

- Identifying the subtopics which will form the lowest levels of the topic design
- Categorizing related subtopics into higher-level topics
- Establishing the top-level topic classifications

As you work through these steps, meet with the people who create Verity search agents at your site to develop a topic outline that best meets your search needs, as described below.

Step One: Identifying Low-level Topics

Find a document you can use as a model whose information is representative of other documents you want to find.

For example, assume you are developing a topic design to find information on the computer industry. As a start, you build a topic that searches for documents related to Apple Computer and related products.

You use the following sample as a model document whose information is representative of other documents you want to find:

A system developed specifically for networked Apple Computer, Inc. Macintosh computers has been announced by Human Designs, Inc.

Dubbed Chorus, the floor-standing unit reportedly can contain up to 16 floating-point processors and connects to networked Macintoshes to create a multiuser desktop environment.

The product offers performance of eight million to 32 million floating-point operations per second and was designed to accommodate software development, according to the vendor. Options include an Ethernet I/O upgrade and a software simulator.

A Chorus 1 single floating-point processor entry-level system costs $9,700. A Chorus 4 configuration with four floating point processors is available at $25,000, which includes a dedicated I/O processor with an Apple Appletalk port and system software. Both systems are upgradable.

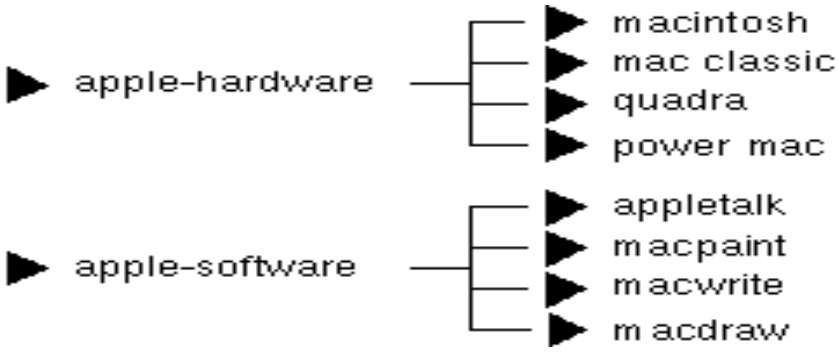
Human Designs, 322 W. 71st St., New York, N.Y. 10023. 212-580-0257.

This document makes you decide you want to locate other documents which refer to "Appletalk" and "Macintosh," so you define two parent topic names, apple-software and apple-hardware.

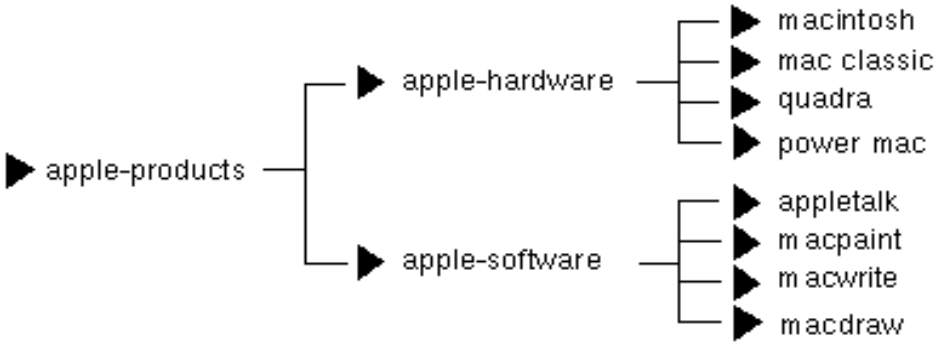
You decide you want to add additional evidence topics to select documents containing related information, such as "Macintosh,"

"Mac Classic," "Quadra," and "Power Mac." In addition, you decide you want to include the evidence topics "AppleTalk" "MacPaint," "MacWrite," and "MacDraw," as related software products. You

assign these evidence topics to your apple-hardware and apple-software topics, as follows:



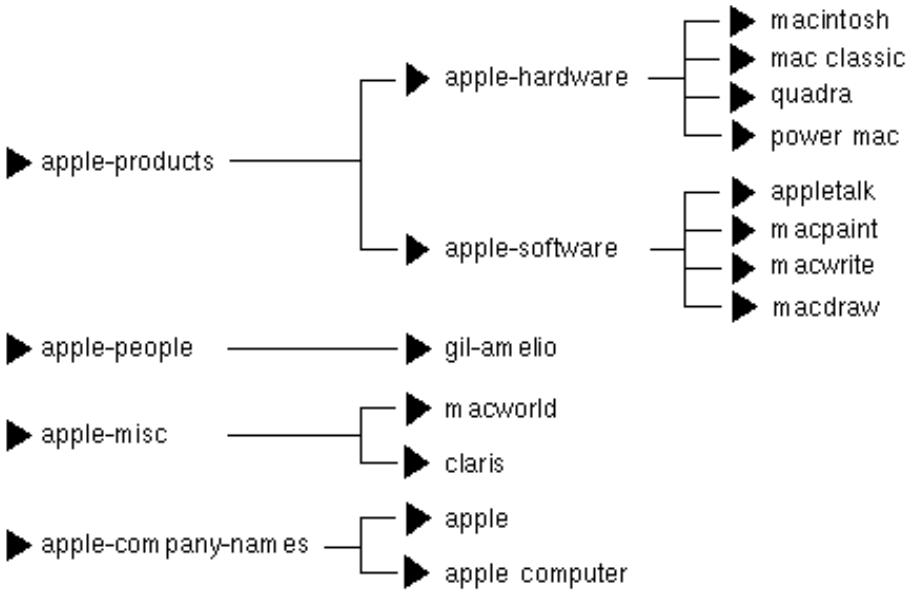
Finally, you want to combine these topics into the topic apple-products, as follows:



Step Two: Categorizing Related Subtopics

Discuss subtopics with the people who use Verity search agents at your site to determine if other subtopics exist that can be logically grouped in a category.

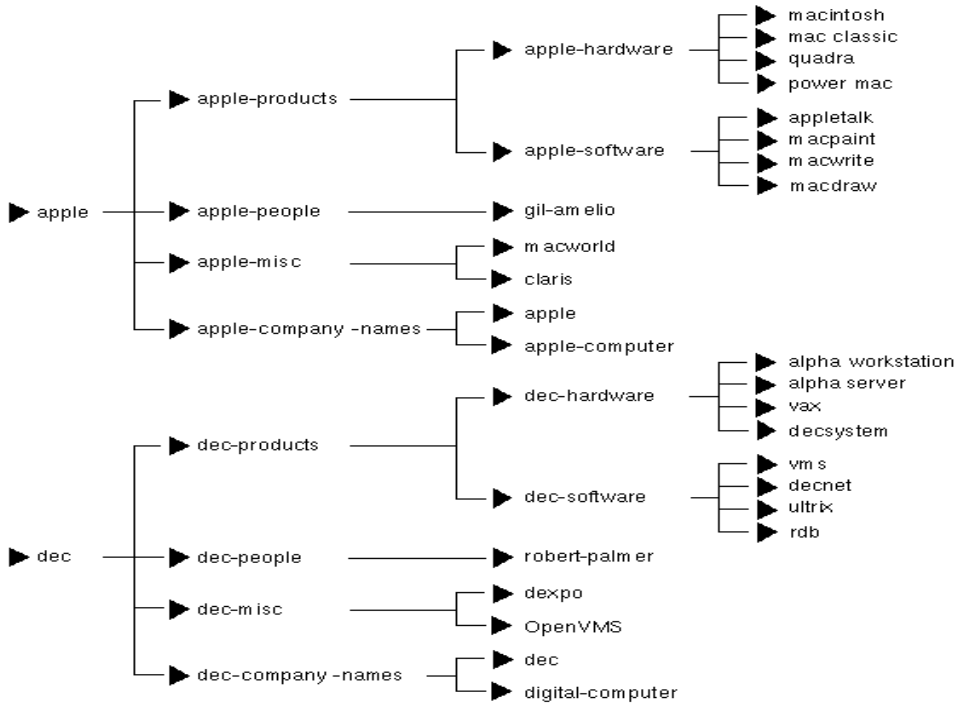
In our example, some of the people who use Verity search agents are interested in finding information on personnel at Apple Computer, and others are interested in finding any documents which refer to Apple Computer. In the example below, a logical group of topics addresses several aspects of Apple Computer:



Step Three: Establishing Top-Level Topics

Determine whether other top-level topics are necessary to find related information.

In the following example, a new topic, dec, is developed for another computer company, Digital Equipment Corporation. This topic was assigned a top-level topic and contains subtopics similar to those defined for the apple topic, as shown below.



A System Procedures

This appendix describes the Sybase-supplied system procedures used for updating and getting reports from system tables. Table A-1 lists the system procedures included with the Full-Text Search engine.

Table A-1: System procedures

Procedure	Description
sp_check_text_index	Reports or fixes consistency problems in FTS index and source tables.
sp_clean_text_events	Removes processed entries from the *text_events* table.
sp_clean_text_indexes	Removes text indexes which are note associated with a table.
sp_create_text_index	Creates an external text index.
sp_drop_text_index	Drops text indexes.
sp_help_text_index	Enhanced version only. Displays text indexes.
sp_optimize_text_index	Enhanced version only. Runs the Verity optimization routines.
sp_redo_text_events	Changes the status of entries in the *text_events* table and forces re-indexing of the modified table.
sp_refresh_text_index	Adds an entry to the *text_events* table reflecting updates to the corresponding source table.
sp_show_text_online	Displays information about databases or indexes that are currently online.
sp_text_cluster	Enhanced version only. Displays or modifies clustering options.
sp_text_configure	Enhanced version only. Displays or modifies Full-Text Search engine configuration parameters.
sp_text_dump_database	Enhanced version only. Makes a backup copy of the text indexes in a database and optionally dumps the *text_db* and current databases.
sp_text_kill	Enhanced version only. Terminates all connections to a specific text index.
sp_text_load_index	Enhanced version only. Restores text indexes from a backup.
sp_text_notify	Notifies the Full-Text Search engine that the *text_events* table has been modified.
sp_text_online	Makes a database available to Adaptive Server.

sp_check_text_index

Function

Reports or fixes consistency problems in the FTS index and source tables.

Syntax

```
sp_clean_text_events server, "index_name",
   "id_column", "fixit"
```

Parameters

server – the name of the text server.

index_name – the name of the text server.

id_column – the source identity column name.

fixit – if FALSE, just reports problems. If TRUE, doesn't report but repairs problems.

Examples

1. `sp_check_text_index "textsvr", "text.i_text", "id", "false"`

   Lists problems on the server named *textsvr* with the column name *text.i_text*.

Comments

- Before using **sp_check_text_index** you must issue **sp_dboption "select into", true**
- This procedure addresses three problems:

  It generates an **sp_refresh_text_index insert for entries in the source table that do not have a matching entry in the index.**

  It generates an **sp_refresh_text_index delete for entries in the index table that have no source table entry.**

  It generates an **sp_refresh_text_index delete** for each extra entry where duplicate index entries exist.
- In order to determine the index duplicates, it is necessary to select all of the ID values from the index table into a temporary table. If the collection has more than 64K ID values, it will be necessary to change the "batch_blocksize configuration parameter from its

default of 0 to 65536 to enable blocked reading of the returned Verity information. If this is not done, FTS will attempt to real all ID values in one read and fail with a Verity error of "-27."

Messages

None

Permissions

Any user can execute **sp_check_text_index.**

sp_clean_text_events

Function

Removes processed entries from the *text_events* table.

Syntax

sp_clean_text_events [*up_to_date*]

Parameters

up_to_date – the date and time through which all processed entries will be deleted.

Examples

1. sp_clean_text_events "01/15/98:17:00"

Removes data entered on or before January 15, 1998 at 5:00 p.m.

Comments

- If the *up_to_date* parameter is specified, all entries having a date less than or equal to *up_to_date* and whose status is set to processed is deleted.
- If *up_to_date* is omitted, all entries whose status is set to processed is deleted.
- Remove entries from the *text_events* table only after you have backed up the collection associated with the text index.
- With the Enhanced Full-Text Search engine, the **sp_text_dump_database** system procedure automatically runs this.

Messages

None

Permissions

Any user can execute **sp_clean_text_events**.

See Also

sp_text_dump_database

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* and *vesauxcol* tables, verifying that both the source table and the corresponding index table exist. If either is missing, the index is dropped.

Messages

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

Permissions

Any user can execute **sp_clean_text_indexes.**

sp_create_text_index

Function

Creates a text index.

Syntax

```
sp_create_text_index server_name, index_table_name,
   table_name, "batch", 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. *index_table_name* has the form [*dbname.*[*owner.*]]*table*, where:

- *dbname* is the name of the database containing the index table.
- *owner* is the name of the owner of the index table.
- *table* is the name of the index table.

table_name – is the name of the source table containing the text being indexed. *table_name* has the form [*dbname.*[*owner.*]]*table.*

batch – The "batch" operator (must be in quotes) tells the Full-Text Search to reallocate every session after each batch sent to the VDK.

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

Examples

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

   Creates a text index and an index table named *i_blurbs* on the *copy* column of the *blurbs* table.

Comments

- Up to 16 columns can be indexed in a single text index.
- Columns of the following datatypes can be indexed:
  - Standard version: *char, varchar, nchar, nvarchar, text, image, datetime,* and *smalldatetime*

  - Enhanced version: all datatypes in the Standard version, plus *int*, *smallint*, and *tinyint*

- The content of *option_string* is not case sensitive.
- *option_string* uses a null string (" ") to specify "No Options".
- Assign the value "empty" t*o option_string* to create a text index that you will immediately drop. This creates the Verity collection directory and the style files, but does not populate the collections. For example, when you configure an individual table for clustering, you create the text index and immediately drop it. After you edit the *style.prm* file, you re-create the text index. For more information, see "Editing Individual style.prm Files" on page 5-3.
- **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 take as long as several hours to complete.
- When you create a text index on two or more columns, each column in the text index is placed into its own document zone. The name of the zone is the name of the column. The zones can be used to limit your search to a particular column. For more information, see "in" on page 6-12.
- **Do not rename an index after creating.**

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 index table and text indexes.

Syntax

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

Parameters

table_name – is the name of the table associated with the text indexes you are dropping. *table_name* has the form [*dbname.*[*owner.*]]*table*, where:

- *dbname* is the name of the database containing the table.
- *owner* is the name of the owner of the table.
- *table* is the name of the table.

index_table_name – is the name of the index table and text index you are dropping. *index_table_name* has the form [*dbname.*[*owner.*]]*index*.

Examples

1. `sp_drop_text_index "blurbs.i_blurbs"`

   Drops the index table and text index associated with the *blurbs* table.

Comments

- First, the **sp_drop_text_index** system procedure issues a remote procedure call (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_help_text_index

(Enhanced version only)

Function

Displays a list of text indexes for the current database.

Syntax

sp_help_text_index [*index_table_name*]

Parameters

index_table_name – is the name of the text index you want to display.

Examples

1. **sp_help_text_index**

   Displays all indexes.

2. **sp_help_text_index "i_blurbs"**

   Displays information about the text index *i_blurbs*.

Comments

- **sp_help_text_index** is available only with Enhanced Full-Text Search Specialty Data Store.
- If the *index_table_name* parameter is specified, information about that text index is displayed. This information includes the name of the text index, the name of the Verity collection for the index, the name of the source table, the name of the IDENTITY column, and the name of the Full-Text Search engine that created the index.
- If *index_table_name* is omitted, a list of all text indexes in the current database is displayed

Messages

- `No text indexes found in database 'database_name'`
- `Text index 'index_name' does not exist in database 'database_name'`
- `Object must be in the current database`

Permissions

Any user can execute **sp_help_text_index**.

sp_optimize_text_index

(Enhanced version only)

Function

Performs optimization on a text index.

Syntax

sp_optimize_text_index *index_table_name*

Parameters

index_table_name – is the name of the text index you want to optimize. *index_table_name* has the form [*dbname.*[*owner.*]]*table*, where:

- *dbname* is the name of the database containing the index table. If present, the *owner* or a placeholder is required.
- *owner* is the name of the owner of the index table.
- *table* is the name of the index table.

Examples

1. sp_optimize_text_index "i_blurbs"

Optimizes the text index *i_blurbs* to improve query performance.

Comments

- **sp_optimize_text_index** is available only with Enhanced Full-Text Search Specialty Data Store.
- This system procedure causes the Full-Text Search engine to run the specified text index through the Verity optimization routines.
- **sp_optimize_text_index** is useful for optimizing a text index that has been updated with Verity optimization disabled (trace flag 11 turned on).
- To enable MaxClean optimization turn on traceflag 30. This traceflag should only be used during maintenance since it could take extra time and interfere with normal usage. MaxClean is a Verity optimization feature that removes out-of-date collection files.

Messages

- '*index_table_name*' is not in the current database

- '*index_table_name*' does not exist
- Index '*index_table_name*' is not a Text Index
- This procedure is not supported against remote server 'server_name'

Permissions

Any user can execute **sp_optimize_text_index.**

See Also

"Updating Existing Indexes" on page 8-1

sp_redo_text_events

Function

Changes the status of entries in the *text_events* table and forces the re-indexing of the modified columns.

Syntax

sp_redo_text_events [*from_date* [,*to_date*]]

Parameters

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

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

Examples

1. sp_redo_text_events "01/05/98:17:00", "02/12/98:08:30"

Re-indexes columns that were modified between January 5, 1998 at 5:00 p.m. and February 12, 1998 at 8:30 a.m.

Comments

- Resets the status to "unprocessed" for all entries in the *text_events* table that currently have a status of "processed." 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. When you use the Enhanced Full-Text Search engine, this procedure is run automatically during **sp_text_load_index**.
- If *to_date* is omitted, all entries between *from_date* and the current date with a status of "processed" are reset to "unprocessed."
- If both *from_date* and *to_date* are omitted, all entries in the *text_events* table with a status of "processed" are reset to "unprocessed."

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

Records modifications in the *text_events* table when you change the text index's source table data.

Syntax

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

Parameters

table_name – is the name of the source table being updated. *table_name* has the form [*dbname.*[*owner.*]]*table*, where:

- *dbname* is the name of the database containing the table.
- *owner* is the name of the owner of the table.
- *table* is the name of the table.

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"**

   Records in the *text_events* table that you have updated the *copy* column of the *blurbs* table. The row you have updated has an *id* of 2.000000.

Comments

- The user maintains the consistency of the text index. You must run **sp_refresh_text_index** anytime you update source data that has been indexed so that the *text_events* table reflects the change. This keeps the collections in sync with your source data. The collections are not updated until you run **sp_text_notify.**
- You can create triggers that issue **sp_refresh_text_index** for non-*text* and non-*image* columns. For more information on creating

triggers, see “Propagating Changes to the Text Index” on page 4-10.

Messages

- `Column 'column_name' does not exist in table 'table_name'`
- `Invalid mod_type specified ('mod_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**.

See Also

sp_text_notify

sp_show_text_online

Function

Displays information about databases or text indexes that are currently online.

Syntax

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

Parameters

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

INDEXES | DATABASES – specifies whether the request should contain data about online indexes or online databases. The default is **INDEXES**.

Examples

1. **exec sp_show_text_online KRAZYKAT**

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

2. **exec sp_show_text_online KRAZYKAT, DATABASES**

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

Comments

- **sp_show_text_online** issues a remote procedure call (RPC) to the Full-Text Search engine to retrieve information about the indexes or the databases that are currently online.
- If the results of this procedure do not list a database, use **sp_text_online** to bring the desired database online.

Messages

- `sp_show_text_online failed for server` *`server_name`*.
- `The parameter value '`*`value`*`' is invalid`
- `The RPC sent to the server returned a failure return code`
- `The second parameter must be INDEXES or DATABASES`

Permissions

Any user can execute **sp_show_text_indexes.**

See Also

sp_text_online

sp_text_cluster

(Enhanced version only)

Function

Displays or changes clustering parameters for the active thread.

Syntax

```
sp_text_cluster server_name, cluster_parameter [,
   cluster_value]
```

Parameters

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

cluster_parameter – is the name of the clustering parameter. Values are shown in Table A-2.

cluster_value – is the value you assign to the clustering parameter for the active thread. Values are shown in Table A-2.

Table A-2: Clustering configuration parameters

Values for *cluster_parameter*	Values for *cluster_value*
cluster_style	Specifies the type of clustering to use. Valid values are: • **fixed** – generates a fixed number of clusters. The number is set by the **cluster_max** parameter. • **coarse** – automatically determines the number of clusters to generate, based on fewer, coarse grained clusters. • **medium** – automatically determines the number of clusters to generate, based on medium sized clusters. • **fine** – automatically determines the number of clusters to generate, based on smaller, finer grained clusters.
cluster_max	Specifies the maximum number of clusters to generate when **cluster_style** is set to **fixed**. A value of **0** means that the search engine determines the number of clusters to generate.

Table A-2: Clustering configuration parameters (continued)

Values for *cluster_parameter*	Values for *cluster_value*
cluster_effort	Specifies the amount of effort (time) that the search engine should expend on finding a good clustering. Valid values are: • **effort_default** – the search engine spends the default amount of time. You can also use the Verity term "default" if you enclose it in double quotes ("""). • **high** – the search engine spends the longest time. • **medium** – the search engine spends less time. • **low** – the search engine spends the least amount of time.
cluster_order	Specifies the order in which to return the rows within the clusters. Valid values are: • **"0"** – indicates rows are returned in order of similarity to the cluster center. This means the first row returned for a cluster is the one that is most prototypical of the rows in the cluster. • **"1"** – indicates that rows are returned in the same relative order in which they were submitted for clustering. For example, if cluster 1 contains the first, third and seventh rows found for the query, they will be returned in that relative order within the cluster.

Examples

1. `sp_text_cluster KRAZYKAT, cluster_order, "1"`

   Changes the *cluster_order* parameter to 1 for the active thread.

2. `sp_text_cluster KRAZYKAT, cluster_style`

   Displays the current value of the *cluster_style* parameter.

Comments

- The Verity clustering algorithm attempts to group similar rows together, based on the values of the clustering parameters.
- If the *cluster_parameter* parameter is specified, but the *cluster_value* parameter is omitted, **sp_text_cluster** displays the value of the clustering parameter that is specified.
- **sp_text_cluster** does not modify the value of the clustering configuration parameter. The *cluster_value* is valid only for the thread that is currently executing. To modify the default values, use the **sp_text_configure** system procedure.

- For information on how to request a clustered result set, see “Using Pseudo Columns to Request Clustered Result Sets” on page 6-6.

Messages

- `This procedure is not supported against remote server '`*`server_name`*`'`
- `The parameter value ‘`*`value`*`’ is invalid`
- `sp_text_cluster failed (status = `*`status`*`)`

Permissions

Any user can execute **sp_text_cluster.**

See Also

sp_text_configure

sp_text_configure

(Enhanced version only)

Function

Displays or changes Full-Text Search engine configuration parameters.

Syntax

sp_text_configure *server_name* [, *config_name* [, *config_value*]]

Parameters

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

config_name – is the name of the configuration parameter to be displayed or modified.

config_value – is the value you assign to the configuration parameter.

Examples

1. **sp_text_configure KRAZYCAT, backdir, "/data/backup"**

   Changes the backup destination directory to */data/backup*.

2. **sp_text_configure KRAZYCAT, backdir**

   Displays the backup destination directory.

Comments

- When you execute **sp_text_configure** to modify a dynamic parameter:
  - The configuration and run values are updated
  - The configuration file is updated
  - The change takes effect immediately
- When you execute **sp_text_configure** to modify a static parameter:
  - The configuration value is updated
  - The configuration file is updated
  - The change takes effect only when you restart the Full-Text Search engine

- When issued with no parameters, **sp_text_configure** displays a report of all Full-Text Search engine configuration parameters and their current values.
- If the *config_name* parameter is specified, but the *config_value* parameter is omitted, **sp_text_configure** displays the report for the configuration parameter specified.
- For information on the individual configuration parameters, see "Modifying the Configuration Parameters" on page 7-5.

Messages

- `Configuration value cannot be specified without a configuration option`
- `This procedure is not supported against remote server 'server_name'`
- `sp_text_configure failed - possible invalid configuration option ('config_name')`

Permissions

Any user can execute **sp_text_configure**.

sp_text_dump_database

(Enhanced version only)

Function

Makes a backup copy of a text index.

Syntax

```
sp_text_dump_database backupdbs [, current_to] [,
   current_with] [, current_stripe01 [, ... [,
   current_stripe31]]] [, textdb_to] [, textdb_with]
   [, textdb_stripe01 [, ... [, textdb_stripe31]]]
```

Parameters

backupdbs – specifies whether the current database and the *text_db* database are backed up before the text index is backed up. Valid values are shown in Table A-3.

Table A-3: Values for backupdbs

Value	Description
CURRENT_DB_AND_INDEXES	Indicates that the current database is backed up before the text indexes are backed up.
CURRENT_DB_AND_CURRENT_INDEXES	Indicates that the current database is backed up before the text indexes are backed up, and only the indexes associated with the current database are dumped.
TEXT_DB_AND_INDEXES	Indicates that the *text_db* database is backed up before the text indexes are backed up.
INDEXES_AND_DATABASES	Indicates that the current and *text_db* databases are backed up before the text indexes are backed up.
ONLY_INDEXES	Indicates that only the text indexes are backed up.

current_to – is the **to** clause of the **dump database** command for dumping the current database. Use this only if you specify **CURRENT_DB_AND_INDEXES** or **INDEXES_AND_DATABASES** for the *backupdbs* parameter.

current_with – is the **with** clause of the **dump database** command for dumping the current database. Use this only if you specify **CURRENT_DB_AND_INDEXES** or **INDEXES_AND_DATABASES** for the *backupdbs* parameter.

current_stripe – is the **stripe** clause of the **dump database** command for dumping the current database. Use this only if you specify **CURRENT_DB_AND_INDEXES** or **INDEXES_AND_DATABASES** for the *backupdbs* parameter.

textdb_to – is the **to** clause of the **dump database** command for dumping the *text_db* database. Use this only if you specify **INDEXES_AND_DATABASES** for the *backupdbs* parameter. Use this only if you specify **TEXT_DB_AND_INDEXES** or **INDEXES_AND_DATABASES** for the *backupdbs* parameter.

textdb_with – is the **with** clause of the **dump database** command for dumping the *text_db* database. Use this only if you specify **TEXT_DB_AND_INDEXES** or **INDEXES_AND_DATABASES** for the *backupdbs* parameter.

textdb_stripe – is the **stripe** clause of the **dump database** command for dumping the *text_db* database. Use this only if you specify **TEXT_DB_AND_INDEXES** or **INDEXES_AND_DATABASES** for the *backupdbs* parameter.

Examples

1. **`sp_text_dump_database ONLY_INDEXES`**

   Only text indexes are backed up.

2. **`sp_text_dump_database CURRENT_DB_AND_INDEXES, "to '/data/db1backup'"`**

   The current database is dumped to */data/db1backup* before the text indexes are backed up.

3. **`sp_text_dump_database @backkupdbs = "TEXT_DB_AND_INDEXES", @textdb_to = "to '/data/textdbbackup'"`**

   The *text_db* database is dumped to */data/textdbbackup* before the text indexes are backed up.

4. **`sp_text_dump_database @backupdbs = "INDEXES_AND_DATABASES", @current_to = "to '/data/db1backup'", @textdb_to = "to '/data/textdbbackup'"`**

   The current database is dumped to */data/db1backup* and the *text_db* database is dumped to */data/textdbbackup* before the text indexes are backed up.

Comments

- The Full-Text Search engine concatenates the values of *current_to*, *current_with*, and *current_stripe01* to *current_stripe31* to **dump database *currentdbname*** and then executes the **dump database** command. The output from the execution of the **dump database** command is sent to the Full-Text Search error log.
- The Full-Text Search engine concatenates the values of *textdb_to*, *textdb_with*, and *textdb_stripe01* to *textdb_stripe31* to the string "**dump database *currentdbname***" and then executes the **dump database** command. The output from the execution of the **dump database** command is sent to the Full-Text Search error log.
- All entries in the *text_events* table that have a "processed" status in the current database are deleted when all indexes have been backed up.
- The backup files for the Verity collections are stored in the directory specified in the **backDir** configuration parameter.
- See references to the configuration paramter **backCmd** for customizing backups.

Messages

- `The parameter value ‘value’ is invalid`
- `Server name ‘server’ does not exist in sysservers`
- `Attempt to dump database ‘database_name’ failed - use the 'dump database' command`
- `Attempt to backup text indexes on server 'server_name' failed`
- `Attempt to clean text_events in database ‘database_name’ failed (date = 'date')`
- `Parameter 'parameter_name' is required when dumping database ‘database_name’`
- `Dumping database 'database_name' - check Full Text Search SDS error log for status`

Permissions

Any user can execute **sp_text_dump_database.**

See Also

dump_database in the *Adaptive Server Reference Manual*

sp_text_kill

(Enhanced version only)

Function

Terminates all connections to a specific text index.

Syntax

sp_text_kill *index_table_name*

Parameters

index_table_name – is the name of the text index from which all connections will be terminated. *index_table_name* has the form [*dbname.*[*owner.*]]*table*, where:

- *dbname* is the name of the database containing the index table. If present, the *owner* or a placeholder is required.
- *owner* is the name of the owner of the index table.
- *table* is the name of the index table.

Examples

1. sp_text_kill "i_blurbs"

Terminates all existing connections to the text index *i_blurbs*.

Comments

- **sp_text_kill** is available only with Enhanced Full-Text Search Specialty Data Store.
- This system procedure causes the Full-Text Search engine to terminate all connections to the specified index, except for the connection that initiated the request.
- Attempts to drop a text index that is currently in use will fail. **sp_text_kill** can be used to terminate all existing connections so that the index can be successfully dropped.

Messages

- `Index 'index_table_name' is not a text index`
- `This procedure is not supported against remote server 'server_name'`
- `'index_table_name' does not exist`

- Only the System Administrator (SA) may execute this procedure

Permissions

Only user "sa" can execute **sp_text_kill.**

See Also

sp_drop_text_index

sp_text_load_index

(Enhanced version only)

Function

Restores a text index backup.

Syntax

sp_text_load_index

Parameters

None.

Examples

1. sp_text_load_index

Restores all text indexes in the current database.

Comments

- Run **sp_text_load_index** after the *text_db* database and the current database have been fully recovered.
- **sp_text_load_index** restores the Verity collections from the most recent backup. The Full-Text Search engine then runs **sp_redo_text_events** and **sp_text_notify** to reapply all entries in the *text_events* table since the date and time the index was backed up.
- See references to the configuration paramter **restoreCmd** for customizing backups.

Messages

- `Server name ‘server_name’ does not exist in sysservers`
- `Unable to restore text indexes for server ‘server_name’`
- `This procedure is not supported against remote server 'server_name'`
- `Update to text_events table in database database_name failed for server ‘server_name’ - text_events not rolled forward`

Permissions

Any user can execute **sp_text_load_index.**

See Also

sp_redo_text_events; **sp_text_notify**

sp_text_notify

Function

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

Syntax

sp_text_notify [{true | false}] [, *server_name*]

Parameters

true – causes the procedure to run synchronously.

false – causes the procedure to run asynchronously.

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

Examples

```
1. sp_text_notify true
```

Comments

- You must run **sp_text_notify** after you issue **sp_refresh_text_index** to inform the Full-Text Search engine that the source tables have been modified.
- If you do not specify **true** or **false**, **sp_text_notify** runs synchronously.
- If no server name is specified, all Full-Text Search engines are notified.

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.**

See Also

sp_refresh_text_index

sp_text_online

Function

Makes a database available for full-text searches to Adaptive Server.

Syntax

sp_text_online [*server_name*], [*database_name*]

Parameters

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

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

Examples

1. **sp_text_online @database_name = 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 for full-text searches.
- If a server name is not specified, all Full-Text Search engines listed in the *vesaux* table are notified.
- With the Enhanced Full-Text Search engine, databases are brought online automatically if the **auto_online** configuration parameter is set to **1**.

Messages

- `All Databases using text indexes are now online`
- `Databases containing text indexes on server 'database_names' are now online`
- `Server name 'server_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 following:

- The text of the default configuration file (*textsvr.cfg*)
- An overview of the **sample_text_main.sql** sample script
- A list of all the sample files provided by the Full-Text Search engine
- An overview of the **getsend** program

Default *textsvr.cfg* Configuration File

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  @(#)  File: textsvr.cfg  1.17  07/26/99
;
;  Full Text Search Specialty Data Store
;             Sample Configuration File
;
;  The installation procedure places this file in the
;  “SYBASE” directory.
;
;  Lines with a semi-colon in column 1 are comment lines.
;
;  Modification History:
;  ---------------------
;  11-21-97     Create file for Standard Full Text Search SDS
;  03-02-98     Add trace flags and config values for
;               Enhanced Full Text Search SDS
;  05-26-99     remove references to sds/text
;  07-09-99     added batch block size
;  08-24-99remove version string and correct copyright
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;             copyright (c) 1997, 1999
;             Sybase, Inc. Emeryville, CA
;             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 *.cfg file)
;
;  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: text_server.cfg
;      (1B)  Modify the [textsvr] line to [your_server_name]
;            Example: [text_server]
;      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: 1,2,3,4,5,6,7,8,9,10,11,12,13
;
;  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
; 13  trace backup operations (Enhanced Full Text Search only)
;
; “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 = english
;charset = iso_1
;vdkLanguage =
;vdkCharset = 850
;traceflags = 0
;srv_traceflags = 0
;max_indexes = 126
;max_packetsize = 2048
;max_stacksize = 34816
;max_threads = 50
;collDir = <txtsvr directory tree location on UNIX>/collections
;collDir = <txtsvr directory tree location on Win-NT>\collections
;vdkHome = <txtsvr directory tree location on UNIX>/verity
;vdkHome = <txtsvr location on Win-NT>\verity
;interfaces = <$SYBASE location on UNIX>/interfaces
```

```
;interfaces = <%SYBASE% location on Win-NT>\ini\sql.ini
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; The parameters in this section apply only to the Enhanced Full Text
Search SDS.
; If defined to a Standard Full Text Search engine they will be ignored.
;
;auto_online = 0
;backDir = <txtsvr directory tree location on UNIX>/backup
;backDir = <txtsvr directory tree location on Win-NT>\backup
;backCmd =
;restoreCmd =
;knowledge_base =
;nocase = 0
;cluster_max = 0
;cluster_order = 0
;cluster_style = Fixed
;cluster_effort = Default
;batch_blocksize = 0
```

The *sample_text_main.sql* Script

The installation of the Full-Text Search engine copies the **sample_text_main.sql** script to the *$SYBASE/$SYBASE_FTS/sample/scripts* directory. This script illustrates the following operations:

- Setting up a text index.
- Modifying data and propagating changes to the collections. This includes inserts, updates, and deletes.
- Dropping a text index.

Execution of this script is not required for installation or configuration; Sybase supplies the script as a sample.

Before you run the **sample_text_main.sql** script:

- Your Adaptive Server and Full-Text Search engine must be configured and running.
- Use a text editor to edit the **sample_text_main.sql** script. Change "YOUR_TEXT_SERVER" to the name of your Full-Text Search engine in Step 4 in the **sample_text_main.sql** script.
- Verify that your *model* database contains a *text_events* table. If your *model* database is **not** configured this way, you need to:
  - Modify the **sample_text_main.sql** script to exit after creating the database

- Apply the **installevent** script to the new database (see "Running the installevent Script" on page 4-4)
- Execute the remainder of the sample script

Direct the script as input to your Adaptive Server. For example, to run the **sample_text_main.sql** script on an Adaptive Server named MYSVR:

```
isql -Ulogin -Ppassword -SMYSVR
-i $SYBASE/$SYBASE_FTS/sample/scripts/sample_text_main.sql -omain.out
```

When you finish with this sample environment, log in to your Adaptive Server and drop the sample database. For example:

```
1> use master
2> go
1> drop database sample_colors_db
2> go
```

The **sample_text_main.sql** script can be rerun.

Sample Files Illustrating Full-Text Search Engine Features

The Full-Text Search engine supplies a set of sample files for illustrating text server operations. The files are located in the *$SYBASE/$SYBASE_FTS/sample/scripts* directory. Execution of the sample files is not required for installation, configuration, or operation of a Full-Text Search engine.

Custom Thesaurus

The following files illustrate how to set up and use a custom thesaurus:

- *sample_text_thesaurus.ctl* – is a sample control file.
- *sample_text_thesaurus.sql* – provides sample queries using the custom thesaurus created by the sample control file.

You can create a custom thesaurus only with the Enhanced Full-Text Search engine. The scripts can be rerun.

Topics

The following files illustrate how to set up and use topics:

- *sample_text_topics.otl* – is a sample outline file.

- *sample_text_topics.kbm* – is a sample knowledge base map.
- *sample_text_topics.sql* – provides sample queries using the defined topics.

Topics is available only with the Enhanced Full-Text Search engine. The scripts can be rerun.

Clustering, Summarization, and Query-by-Example

The following files illustrate how to set up and use clustering, summarization and query-by example:

- *sample_text_setup.sql* – creates a sample environment.
- *sample_text_queries.sql* – issues queries against the environment and drops the environment.

You can use these scripts only with the Enhanced Full-Text Search engine. These scripts can be rerun as a pair.

getsend Sample Program

The Enhanced Full-Text Search engine supplies a program named **getsend** to load *text* or *image* data from a file into a column defined in Adaptive Server.

The required source and header files, a makefile, and directions for building and running the program are included in the directory:

$SYBASE/$SYBASE_FTS/sample/source

Refer to the *README.TXT* file and *getsend.c* file for information on how to use the program.

C Unicode Support

The Unicode standard, a subset of the International Standards Organization's ISO 10646 standard, is an international character set. Unicode is identical to the Basic Multilingual Plane (BMP) of ISO 10646, which supports all the major scripts and languages in the world. Therefore, it is a superset of all existing character sets.

The major advantages of Unicode are:

- Provides single-source development. This means you develop an application once and it can then be localized for multiple locales and in multiple languages. By using a single unified character set, you do not have to modify your applications to take into account differences between character sets, thus reducing development, testing, and support costs.
- Allows you to mix different languages in the same database. An all-Unicode system does not require that you design your database to keep track of the character set of your data.

The Enhanced Full-Text Search engine supports Unicode. To use this feature, you need to obtain and install the Unicode Developer's Kit (also known as UDK). This contains everything you need to set up a Unicode-enabled client / server database system.

To configure the Full-Text Search engine to store data in Unicode format, set the **charset** configuration value to **utf8** (see "Modifying the Configuration Parameters" on page 7-5).

➤ ***Note***

If you issue wildcard searches against data in Unicode format, turn on trace flag 15. For more information, refer to "Setting Trace Flags" on page 7-12,

Index

D

E

F

G

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Z