

Excalibur Text Search DataBlade Module

User's Guide

Version 1.2
March 1999
Part No. 000-5401

Published by INFORMIX® Press

Informix Corporation
4100 Bohannon Drive
Menlo Park, CA 94025-1032

© 1999 Informix Corporation. All rights reserved. The following are trademarks of Informix Corporation or its affiliates:

Answers OnLine™; CBT Store™; C-ISAM®; Client SDK™; ContentBase™; Cyber Planet™; DataBlade®; Data Director™; Decision Frontier™; Dynamic Scalable Architecture™; Dynamic Server™; Dynamic Server™, Developer Edition™; Dynamic Server™ with Advanced Decision Support Option™; Dynamic Server™ with Extended Parallel Option™; Dynamic Server™ with MetaCube® ROLAP Option; Dynamic Server™ with Universal Data Option™; Dynamic Server™ with Web Integration Option™; Dynamic Server™, Workgroup Edition™; FastStart™; 4GL for ToolBus™; If you can imagine it, you can manage itSM; Illustra®; INFORMIX®; Informix Data Warehouse Solutions... Turning Data Into Business Advantage™; INFORMIX®-Enterprise Gateway with DRDA®, Informix Enterprise Merchant™; INFORMIX®-4GL; Informix-JWorks™; InformixLink®, Informix Session Proxy™; InfoShelf™; Interforum™; I-SPY™; Mediazation™; MetaCube®, NewEra™; ON-Bar™; OnLine Dynamic Server™; OnLine for NetWare®, OnLine/Secure Dynamic Server™; OpenCase®, ORCA™; Regency Support®; Solution Design LabsSM; Solution Design ProgramSM; SuperView®; Universal Database Components™; Universal Web Connect™; ViewPoint®, Visionary™; Web Integration Suite™. The Informix logo is registered with the United States Patent and Trademark Office. The DataBlade logo is registered with the United States Patent and Trademark Office.

Documentation Team: Daniel Howard, Juliet Shackell, Oakland Editing and Production

Contributors: David Ashkenas, Charlie Bowen, Tony Corman, Tom Demott, Kevin Foster, Mike Frame, Frank Glandorf, Kumar Ramaiyer, Hari Rao, Jackie Ryan, Dave Segleau, Kanchan Sringhi, Joseph Wilson, Weiming Ye

GOVERNMENT LICENSE RIGHTS

Software and documentation acquired by or for the US Government are provided with rights as follows:

- (1) if for civilian agency use, with rights as restricted by vendor's standard license, as prescribed in FAR 12.212;
- (2) if for Dept. of Defense use, with rights as restricted by vendor's standard license, unless superseded by a negotiated vendor license, as prescribed in DFARS 227.7202. Any whole or partial reproduction of software or documentation marked with this legend must reproduce this legend.

Table of Contents

Introduction

In This Introduction	3
About This Manual	3
Organization of This Manual	3
Types of Users	5
Software Dependencies	5
Documentation Conventions	5
File-Naming Conventions	6
Typographical Conventions	6
Icon Conventions	7
Syntax Conventions	8
Additional Documentation	9
Printed Documentation	9
On-Line Documentation	11
Related Reading	11
Informix Welcomes Your Comments	12

Chapter 1

Overview

In This Chapter	1-3
Conceptual Overview	1-4
The Excalibur Text Search DataBlade Module	1-5
How the DataBlade Module Works	1-5
Components of the Excalibur Text Search DataBlade Module	1-7
The etx Access Method	1-7
The etx_contains() Operator	1-12
Routines Defined for the DataBlade Module.	1-16
Word Lists	1-16
Synonym Lists	1-16
Stopword Lists	1-18

	Character Sets	1-20
	Built-In Character Sets	1-21
	User-Defined Character Sets	1-21
	Highlighting	1-22
Chapter 2	Text Search Concepts	
	In This Chapter	2-3
	Excalibur Text Search Types	2-3
	Keyword Searches	2-3
	Boolean Searches	2-5
	Phrase Searches	2-7
	Proximity Searches	2-9
	Performing Fuzzy Searches	2-11
	Word Scoring for Fuzzy Searches	2-11
	Document Scoring for Fuzzy Searches	2-16
	Limiting the Number of Rows a Fuzzy Search Returns	2-19
Chapter 3	Tutorial	
	In This Chapter	3-3
	Creating a Table Containing a Text Search Column	3-4
	Creating Dbspaces and Sbspaces for Storage	3-4
	Creating the Table	3-5
	Populating the Table with Text Search Data	3-5
	Creating Word Lists and a User-Defined Character Set	3-6
	Creating a Stopword List	3-6
	Creating a Synonym List	3-7
	Creating a User-Defined Character Set	3-8
	Creating an etx Index	3-9
	Determining the etx Index Parameters	3-10
	Creating an etx Index	3-10
	Performing Text Search Queries	3-13
	Highlighting	3-15
Chapter 4	Data Types	
	In This Chapter	4-3
	etx_ReturnType	4-4
	etx_HiliteType.	4-6
	IfxDocDesc	4-8
	IfxMRData	4-13

Chapter 5

Routines

In This Chapter	5-3
etx_contains()	5-5
etx_CloseIndex()	5-14
etx_CreateCharSet()	5-16
etx_CreateStopWlst()	5-20
etx_CreateSynWlst()	5-22
etx_DropCharSet()	5-25
etx_DropStopWlst()	5-27
etx_DropSynWlst()	5-29
etx_Filter()	5-30
EtxFilterTraceFile()	5-32
etx_GetHilite()	5-33
etx_HiliteDoc()	5-36
etx_Release()	5-38
txt_Release()	5-39
etx_ViewHilite()	5-40
etx_WordFreq()	5-42

Chapter 6

etx Index Parameters

In This Chapter	6-3
Overview of the etx Index Parameters	6-3
CHAR_SET	6-4
FILTER	6-5
INCLUDE_STOPWORDS	6-7
PHRASE_SUPPORT	6-8
WORD_SUPPORT	6-9
STOPWORD_LIST	6-9

Chapter 7

DataBlade Module Administration

In This Chapter	7-3
Retrieving DataBlade Module Version Information	7-4
Configuring Your Database Server	7-4
Creating and Configuring Sbspaces	7-5
Setting ONCONFIG Tuning Parameters	7-8
Configuring Your Database Server for Filtering	7-9
Specifying the ETX VPCLASS	7-10
Logging Messages from the Filter Server	7-10
Enabling Tracing	7-11
Handling Filter Errors	7-12

Refining etx Indexes	7-14
Estimating the Size of an etx Index.	7-14
Fragmenting Indexes	7-16
Sharing etx Indexes	7-18
Reclaiming Unused Index Space	7-18
Choosing the Appropriate Storage Data Type	7-19
BLOB.	7-19
CLOB.	7-19
LVARCHAR	7-20
IfxDocDesc	7-20
IfxMRData	7-21
Loading Data	7-21
Nonindexed Tables	7-21
Preindexed Tables	7-21
Handling Deadlocks	7-22
Refining Queries	7-23
Queries That Contain Multiple etx_contains() Operators	7-23
Queries That Include the NOT Keyword.	7-25
Queries That Include Index Scans and Nonindex Scan Filters . .	7-25

Appendix A Character Sets

Appendix B Document Formats You Can Filter

Appendix C etx_lists Table

Glossary

Error Messages

Index

Introduction

In This Introduction	3
About This Manual.	3
Organization of This Manual	3
Types of Users	5
Software Dependencies	5
Documentation Conventions	5
File-Naming Conventions	6
Typographical Conventions	6
Icon Conventions	7
Syntax Conventions	8
Additional Documentation	9
Printed Documentation	9
On-Line Documentation.	11
Related Reading	11
Informix Welcomes Your Comments.	12

In This Introduction

This introduction provides an overview of the information in this manual and describes the conventions it uses.

About This Manual

This manual describes the Excalibur Text Search DataBlade module and how to access and use its components, which include the **etx** access method, the **etx_contains()** search operator, and data types and routines that enable sophisticated search operations, such as filtering, matching, and highlighting. The manual also includes a tutorial for setting up and performing example text searches.

This section discusses the organization of the manual, the intended audience, and the associated software products that you must have to develop and use the DataBlade module.

Organization of This Manual

This manual includes the following chapters:

- This introduction provides an overview of the manual, describes the documentation conventions used, and explains the generic style of this documentation.
- [Chapter 1, “Overview,”](#) provides an overview of the Excalibur Text Search DataBlade module and its principal components.
- [Chapter 2, “Text Search Concepts,”](#) discusses in detail the different types of searches enabled by the Excalibur Text Search DataBlade module and defines text search terminology.

- [Chapter 3, “Tutorial,”](#) provides step-by-step instructions that show you how to create tables with text search columns, insert data into the tables, create custom synonym and stopword lists and user-defined character sets, create text search indexes, perform text searches, and highlight results.
- [Chapter 4, “Data Types,”](#) is a reference chapter that describes the Informix constituent data types IfxMRData, IfxDocDesc, etx_ReturnType, etx_HiliteType, and LLD_Locator. It describes how to store text data in the IfxMRData and IfxDocDesc storage data types and how to use the etx_ReturnType and etx_HiliteType data types to access scoring and highlighting information.
- [Chapter 5, “Routines,”](#) is a reference chapter that describes the **etx_contains()** operator that you use to define, perform, and highlight text searches. It also describes the DataBlade-defined routines that enable you to create and maintain synonym and stopword lists and user-defined character sets.
- [Chapter 6, “etx Index Parameters,”](#) is a reference chapter that describes the parameters you can use to customize **etx** indexes.
- [Chapter 7, “DataBlade Module Administration,”](#) discusses DBMS administration topics that database administrators should read before they design or redesign a database that will use the Excalibur Text Search DataBlade module.
- [Appendix A, “Character Sets,”](#) describes the three built-in character sets that the Excalibur Text Search DataBlade module supports: ASCII, ISO, and OVERLAP_ISO. It also provides a generic map of the standard and extended characters to aid you in creating your own user-defined character sets.
- [Appendix B, “Document Formats You Can Filter,”](#) provides a list of document formats you can filter when you create indexes on columns of data that contains formatting information.
- [Appendix C, “etx_lists Table,”](#) describes the contents of a system table that provides information about stopword lists, synonym lists, and character set lists used in a database.

A description of error messages and a glossary of relevant terms follows the chapters, and an index directs you to areas of particular interest.

Types of Users

This manual is written for the following groups:

- Users who want to perform SQL searches of text data stored in proprietary format
- Programmers who want to build applications that call on the Excalibur text search engine to perform text searches
- Programmers who want to modify existing applications to take advantage of **etx** indexes

Software Dependencies

You must have the following Informix software to use the Excalibur Text Search DataBlade module:

- Informix Dynamic Server with Universal Data Option
- The Informix Large Object Locator DataBlade module
- The Informix Text Descriptor DataBlade module

You can use the following application development tools with the Excalibur Text Search DataBlade module:

- DB-Access
- INFORMIX-ESQL/C
- DataBlade API

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

The following conventions are discussed:

- File naming conventions
- Typographical conventions

- Icon conventions
- Syntax conventions

File-Naming Conventions

This manual uses the UNIX file separator when specifying a file or directory on the operating system: forward slash (/). If you are using the NT version of the Excalibur Text Search DataBlade module, substitute the UNIX file separator with a backslash (\). For example, the NT version of the UNIX file `/local/excal/dbms.txt` is `\local\excal\dbms.txt`.

This manual also refers to the UNIX environment variable `$INFORMIXDIR`. On NT, this environment variable is `%INFORMIXDR%`.

Typographical Conventions

This manual uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
<i>italics</i> <i>italics</i> <i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface <i>boldface</i>	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
monospace <i>monospace</i>	Information that the product displays and information that you enter appear in a monospace typeface.

(1 of 2)






Convention	Meaning
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
◆	This symbol indicates the end of product- or platform-specific information.
→	This symbol indicates a menu item. For example, “Choose Tools→Options ” means choose the Options item from the Tools menu.

(2 of 2)

***Tip:** When you are instructed to “enter” characters or to “execute” a command, immediately press RETURN after the entry. When you are instructed to “type” the text or to “press” other keys, no RETURN is required.*

Icon Conventions

Comment icons identify three types of information, as the following table describes. This information always appears in italics.

Icon	Label	Description
	<i>Warning:</i>	Identifies paragraphs that contain vital instructions, cautions, or critical information
	<i>Important:</i>	Identifies paragraphs that contain significant information about the feature or operation that is being described
	<i>Tip:</i>	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

Syntax Conventions

This section describes conventions for syntax diagrams. Each diagram displays the sequences of required and optional keywords, terms, and symbols that are valid in a given statement, command line, or other specification, as in the following diagram of the GOTO statement.



Keep in mind the following rules when you read syntax diagrams in this book:

- For ease of identification, all of the keywords (like GOTO in the preceding diagram) are shown in UPPERCASE characters, even though you can type them in either uppercase or lowercase characters.
- Terms for which you must supply specific values or names are in *italics*. In this example, you must replace *label* with an identifier. Below each diagram that contains an italicized term, a table identifies what you can substitute for the term.
- All of the punctuation and other nonalphabetic characters are literal symbols. In this example, the colon is a literal symbol.
- Each syntax diagram begins at the upper left corner and ends at the upper right corner with a vertical terminator. Between these points, any path that does not stop or reverse direction describes a possible form of the statement. (For a few diagrams, notes in the text identify path segments that are mutually exclusive.)

Syntax elements in a path represent terms, keywords, symbols, and segments that can appear in your statement. Except for separators in loops, which the path approaches counterclockwise from the right, the path always approaches elements from the left and continues to the right. Unless otherwise noted, at least one blank character separates syntax elements.

Additional Documentation

The Excalibur Text Search DataBlade module documentation set includes printed manuals, on-line manuals, and on-line notes.

This section describes the following parts of the documentation set:

- Printed documentation
- On-line documentation
- Related reading

Printed Documentation

The following related Informix documents complement the information in this manual set:

- Before you can use the Excalibur Text Search DataBlade module, you must install and configure Informix Dynamic Server with Universal Data Option. The [Administrator's Guide](#) for your database server provides information about how to configure Universal Data Option and also contains information about how it interacts with DataBlade modules.
- Installation instructions for the Excalibur Text Search DataBlade module are provided in the hard copy [Read Me First](#) sheet for DataBlade modules that is packaged with this product. Once you have installed the DataBlade module, you must use BladeManager to register it into the database where the DataBlade module will be used. See the [DataBlade Module Installation and Registration Guide](#) for details on registering DataBlade modules.
- The Excalibur Text Search DataBlade module uses data types and routines defined by the Informix Large Object Locator DataBlade module. Refer to the [Informix Large Object Locator DataBlade Module User's Guide](#) for more information on these data types and functions.
- The Excalibur Text Search DataBlade module uses data types defined by the Text Descriptor DataBlade module. Refer to the Text Descriptor DataBlade module release notes for more information on the Text Descriptor DataBlade module. [Chapter 4, "Data Types,"](#) of this guide also describes the data types in detail.

- If you plan to develop your own DataBlade modules using the Excalibur Text Search DataBlade module as a foundation, read the [DataBlade Developers Kit User's Guide](#). This manual describes how to develop DataBlade modules using BladeSmith and BladePack.
- If you have never used Structured Query Language (SQL), read the [Informix Guide to SQL: Tutorial](#). It provides a tutorial on SQL as it is implemented by Informix products. It also describes the fundamental ideas and terminology for planning and implementing an object-relational database.
- A companion volume to the tutorial, the [Informix Guide to SQL: Reference](#), includes details of the Informix system catalog tables, describes Informix and common environment variables that you should set, and describes the data types that Informix database servers support.
- The [Guide to GLS Functionality](#) describes how to use nondefault locales with Informix products. A locale contains language- and culture-specific information that Informix products use when they format and interpret data.
- The [DB-Access User Manual](#) describes how to invoke the DB-Access utility to access, modify, and retrieve information from Informix database servers.

On-Line Documentation

In addition to the Informix set of manuals, the following on-line files supplement the information in this manual. They are located in the directory **\$INFORMIXDIR/extend/ETX.version**, where **version** refers to the latest version of the DataBlade module installed on your computer.

On-Line File	Purpose
ETXREL.TXT	The release notes file describes known problems and their workarounds, new and changed features, and special actions required to configure and use the Excalibur Text Search DataBlade module on your computer.
ETXDOC.TXT	The documentation notes file describes features not covered in the manuals or modified since publication.
ETXMAC.TXT	The machine notes file describes platform-specific information regarding the release.

Please examine these files because they contain vital information about application and performance issues.

Related Reading

For additional technical information on database management, consult the following books. The first book is an introductory text for readers who are new to database management, while the second book is a more complex technical work for SQL programmers and database administrators. The third book discusses object-relational database management systems:

- *Database: A Primer* by C. J. Date (Addison-Wesley Publishing, 1983)
- *An Introduction to Database Systems* by C. J. Date (Addison-Wesley Publishing, 1994)
- *Object-Relational DBMSs: The Next Great Wave* by Michael Stonebraker and Dorothy Moore (Morgan Kaufman Publications, 1996)

To learn more about the SQL language, consider the following books:

- *A Guide to the SQL Standard* by C. J. Date with H. Darwen (Addison-Wesley Publishing, 1993)

- *Understanding the New SQL: A Complete Guide* by J. Melton and A. Simon (Morgan Kaufmann Publishers, 1993)
- *Using SQL* by J. Groff and P. Weinberg (Osborne McGraw-Hill, 1990)

To learn more about indexing and compressing text documents, consider the following book:

- *Managing Gigabytes* by Ian H. Witten, Alistair Moffat, and Timothy C. Bell (Van Nostrand Rheinhold, 1994)

To use the Excalibur Text Search DataBlade module, you should be familiar with your computer operating system. If you have limited UNIX system experience, consult your operating system manual or a good introductory text before you read this manual. The following texts provide a good introduction to UNIX systems:

- *Introducing the UNIX System* by H. McGilton and R. Morgan (McGraw-Hill Book Company, 1983)
- *Learning the UNIX Operating System*, by G. Todino, J. Strang, and J. Peek (O'Reilly & Associates, 1993)
- *A Practical Guide to the UNIX System*, by M. Sobell (Benjamin/Cummings Publishing, 1989)
- *UNIX for People* by P. Birns, P. Brown, and J. Muster (Prentice-Hall, 1985)
- *UNIX System V: A Practical Guide* by M. Sobell (Benjamin/Cummings Publishing, 1995)

Informix Welcomes Your Comments

Let us know what you like or dislike about our manuals. To help us with future versions of our manuals, we want to know about any corrections or clarifications that you would find useful. Include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Write to us at the following address:

Informix Software, Inc.
Technical Publications
300 Lakeside Dr., Suite 2700
Oakland, CA 94612

If you prefer to send electronic mail, our address is:

`doc@informix.com`

We appreciate your suggestions.

Overview

In This Chapter	1-3
Conceptual Overview	1-4
The Excalibur Text Search DataBlade Module	1-5
How the DataBlade Module Works	1-5
Components of the Excalibur Text Search DataBlade Module	1-7
The etx Access Method	1-7
Data Types	1-7
Operator Classes	1-8
Creating an etx Index	1-9
Specifying etx Index Parameters	1-10
Filtering	1-11
The etx_contains() Operator	1-12
Performing Simple Keyword Searches	1-12
Performing Boolean Searches	1-14
Routines Defined for the DataBlade Module.	1-16
Word Lists	1-16
Synonym Lists	1-16
Stopword Lists	1-19
Character Sets	1-20
Built-In Character Sets	1-21
User-Defined Character Sets	1-21
Highlighting	1-22

In This Chapter

This chapter provides an overview of the Excalibur Text Search DataBlade module. The Excalibur Text Search DataBlade module is a collection of data types and routines that extends Informix Dynamic Server with Universal Data Option to enable you to search your data in ways that are faster and more sophisticated than the keyword matching that SQL provides. Excalibur text search capabilities include phrase matching, exact and fuzzy searches, compensation for misspelling, and synonym matching.

This chapter includes the following topics:

- Text search concepts and terminology
- Overview of the **etx** access method
- Overview of creating an **etx** index
- Basics of performing keyword and Boolean searches
- Word list concepts and use
- User-defined character sets concepts and use
- Overview of highlighting the clue in the search text

Although this chapter does contain examples, it does not contain step-by-step instructions that explain how to use the Excalibur Text Search DataBlade module. See [Chapter 3, “Tutorial,”](#) for instructions on how to use the features described in this overview. Informix recommends, however, that you read the overview of the product provided in this chapter as well as the concepts described in [Chapter 2, “Text Search Concepts,”](#) before you begin the tutorial.

Conceptual Overview

Under traditional relational database systems, you are limited to using a **LIKE** or **MATCHES** condition when you search for words or phrases in a column that contains text data. For example, if you want to return all rows of the table **videos** in which the **VARCHAR description** column contains the phrase **multimedia text editor**, you are limited to executing a statement such as:

```
SELECT * FROM videos
WHERE description LIKE '%multimedia text editor%';
```

Although this **SELECT** statement returns a correct list of rows, it is probably very slow, depending on the size of the table and the amount of text data in the **description** column. Because a traditional secondary index on the column **description** is not useful in this type of search, the whole table has to be scanned.

In addition to being slow, the **SQL** statement only retrieves rows in which data has been entered exactly as it appears in the **LIKE** clause. Rows in the table that contain the word **multimedia** consistently misspelled as **multimedia**, for example, are not returned, although they are probably of interest to you.

If you want to also find documents that contain synonyms or alternate spellings of the words you are searching for, you must construct a complicated statement that contains many **ORs** in the **WHERE** clause, such as:

```
SELECT * FROM video
WHERE description LIKE '%multimedia text editor' OR
description LIKE '%multi-media text editor%' OR
description LIKE '%multimedia document editor%' OR
description LIKE '%multi-media document editor%';
```

This type of **SELECT** statement is often extremely slow. Traditional relational database systems cannot perform sophisticated and fast searches of this type.

The Excalibur Text Search DataBlade Module

The Excalibur Text Search DataBlade module allows you to perform more sophisticated searches by dynamically linking in the Excalibur class library, or text search engine, to perform the text search section of the SELECT statement, instead of having the database server perform a traditional search. The text search engine is specifically designed to perform sophisticated and fast text searches. It runs in one of the database server-controlled virtual processes.

When you execute searches with the Excalibur Text Search DataBlade module, instead of using LIKE or MATCHES in the SELECT statement, you use an operator called **etx_contains()** that is defined for the DataBlade module and that instructs the database server to call the text search library of functions to perform the text search. This operator takes a variety of parameters to make the search more detailed than one using LIKE.

With the Excalibur Text Search DataBlade module, you can enable faster searches by creating an **etx** index on the column that contains text data users might want to search. An **etx** index is a specialized type of secondary index you can create with the **etx** access method, which is described in the next section.

How the DataBlade Module Works

Standard SQL searches that include LIKE and MATCHES clauses use the Universal Data Option B-tree access method to scan data. When you perform searches with the **etx_contains()** operator, on the other hand, Universal Data Option relays that part of the query to the Excalibur text search engine instead. You can think of the Excalibur text search engine as a specialized part of Universal Data Option that is executed when you specify an **etx_contains()** operation in a SELECT statement.

The text search engine produces a group of rows that satisfy the conditions you specify in the **etx_contains()** operator and returns them to the database server. If the query contains criteria other than those specified by the **etx_contains()** operator, Universal Data Option further qualifies the set of rows. Finally, Universal Data Option returns a list of rows that satisfy all the conditions of the WHERE clause. [Figure 1-1](#) illustrates this process.

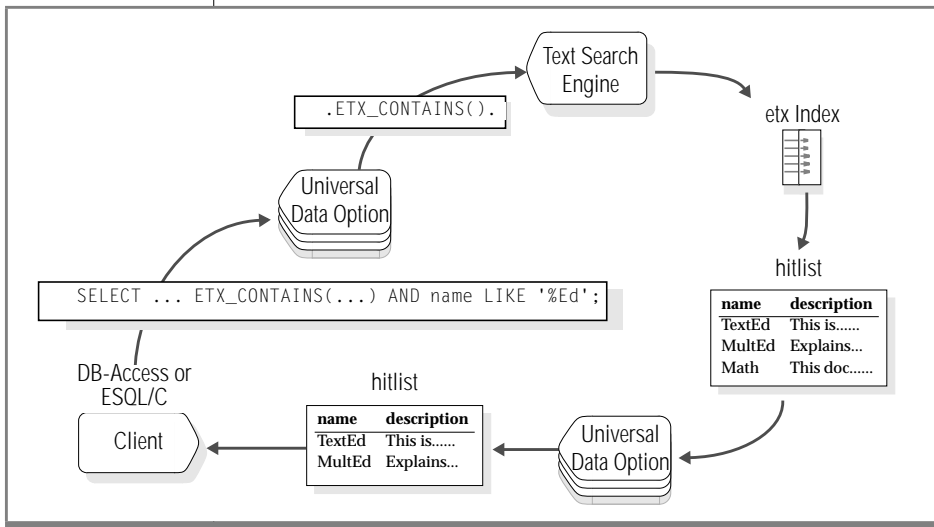


Figure 1-1
Text Search
Process Illustrated

Even though, for illustrative purposes, the preceding figure shows the Excalibur text search engine as a separate process, it is important to remember that it is actually dynamically linked to, and considered part of, Universal Data Option.

The following sections describe the components of the Excalibur Text Search DataBlade module and explain how it works in more detail.

Components of the Excalibur Text Search DataBlade Module

The Excalibur Text Search DataBlade module has three principal components: the **etx** access method, the **etx_contains()** operator, and the supporting routines defined for the DataBlade module. Each of the components is described in detail in the following sections.

The examples in this chapter are based on the **videos** table, a simple table with three columns, as defined in the following example:

```
CREATE TABLE videos
(
    id            INTEGER,
    name          VARCHAR(30),
    description    CLOB
);
```

The etx Access Method

The **etx** access method allows you to call on the Excalibur Text Retrieval Library to create indexes that support sophisticated searches on table columns that contain text. The indexes that you create with the **etx** access method are called **etx indexes**.

Data Types

To take advantage of the **etx** access method, you must store the text data you want to search in a column of type BLOB, CLOB, LVARCHAR, CHAR, VARCHAR, IfxDocDesc, or IfxMRData.

If most of your documents are large (over 2 KB), store them in columns of type BLOB or CLOB. If your documents contain binary data, such as proprietary formatting information from word processing programs such as Microsoft Word, store them in columns of type BLOB. If your documents contain only standard ASCII character data, such as HTML pages, store them in columns of type CLOB.

If all your documents are smaller than 2 KB and do not contain any binary data, consider storing the documents in columns of type LVARCHAR.

Columns of type CHAR or VARCHAR are limited in the amount of information they can hold and are not adequate for large documents. Columns of these two data types are useful for descriptive information, such as titles and names. You can also create **etx** indexes on tables that contain legacy data in CHAR or VARCHAR columns.

IfxDocDesc and IfxMRData types are designed specifically for use with text access methods. The IfxDocDesc data type enables you to store documents in either the database or in the operating system file system. IfxMRData is a multirepresentational data type, which means that the data type itself decides whether to store your document as an LVARCHAR or as a smart large object, depending on the initial size of the document.

See [“IfxDocDesc” on page 4-8](#) and [“IfxMRData” on page 4-13](#) for more detailed information on these data types. Refer to [“Choosing the Appropriate Storage Data Type” on page 7-19](#) for information about how these data types affect database performance.

Operator Classes

When you create an **etx** index, you must specify the *operator class* defined for the data type of the column being indexed. An operator class is a set of functions that Universal Data Option associates with the **etx** access method to optimize queries and build indexes. Each of the seven data types that supports an **etx** index has a corresponding operator class that must be specified when creating the index. The following table lists each data type and its corresponding operator class.

Data Type	Operator Class
BLOB	etx_blob_ops
CLOB	etx_clob_ops
LVARCHAR	etx_lvarc_ops
CHAR	etx_char_ops

(1 of 2)



Data Type	Operator Class
VARCHAR	etx_varc_ops
IfxDocDesc	etx_doc_ops
IfxMRData	etx_mrd_ops

(2 of 2)

Important: You must always specify an operator class when you create an **etx** index, even though each supported data type has only one operator class defined for it. Consult the preceding table to be sure you specify the correct operator class for the data type of the column you want to index.

The following section shows an example of specifying an operator class when creating an **etx** index.

Creating an etx Index

To create an **etx** index, you specify the **etx** access method in the USING clause of the CREATE INDEX statement. For example, suppose your search text is contained in the column **description**, of type CLOB, in the **videos** table. To create an **etx** index named **desc_idx** for this table that is stored in the sbspace **sbsp1**, use the following syntax:

```
CREATE INDEX desc_idx ON videos (description etx_clob_ops)
      USING etx in sbsp1;
```

The operator class **etx_clob_ops** is specified directly after **description**, the column to be indexed. Be sure to specify the correct operator class for the type of the indexed column. Refer to the table in the previous section for valid operator class names and their corresponding data types. Refer to [Informix Guide to SQL: Syntax](#) for detailed information on the CREATE INDEX statement.

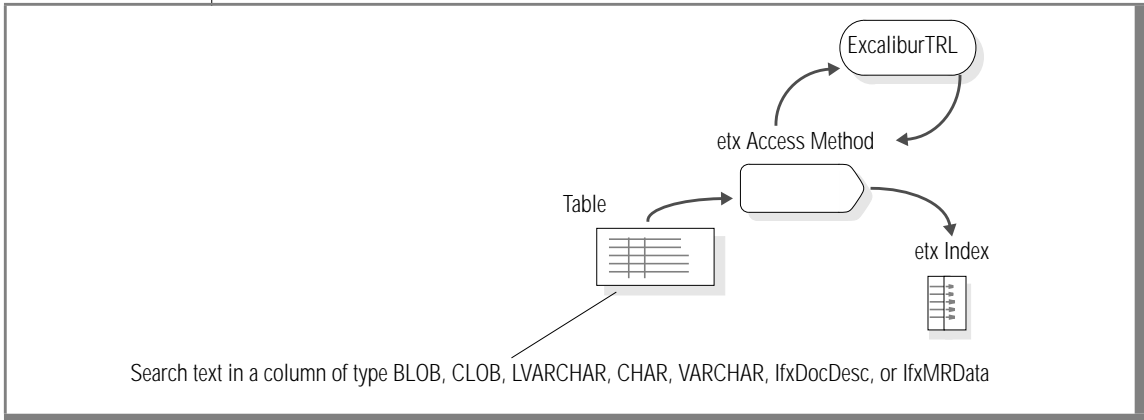
An *sbspace* is a logical storage area that contains one or more chunks that store BLOB and CLOB data types. Refer to the [Administrator's Guide](#) for your database server for detailed information on sbspaces and smart large objects.



Warning: The following options to the CREATE INDEX statement cannot be specified when you create **etx** indexes: CLUSTER, UNIQUE, DISTINCT, ASC, DESC, and FILLFACTOR.

Figure 1-2 shows how the DataBlade module creates an **etx** index.

Figure 1-2
etx Index Creation



The **etx** access method supports both table and index fragmentation. For more information on fragmentation, see [“Creating an etx Index” on page 3-9](#).

Specifying etx Index Parameters

An *index parameter* is a variable that you use to specify the characteristics of an index based on the searches you plan to perform. You set an index parameter in the USING clause of a CREATE INDEX statement when you create an **etx** index.

For example, suppose you want to create an index identical to the index created in the previous example, but with one difference: you want to index international characters as well as the standard ASCII characters. To do this, you must create the index specifying the ISO character set using the index parameter CHAR_SET.

The following example shows how to create this type of index:

```
CREATE INDEX desc_idx1 ON videos (description etx_clob_ops)
  USING etx (CHAR_SET = 'ISO') IN sbssl;
```

By creating indexes based on likely search characteristics, you can reduce the size of your indexes and improve the relevance of search results.

You cannot alter the characteristics of an **etx** index after you create it. Instead, you must drop the index and then create it again with the new or altered characteristic.

For more detailed information on index parameters, see [Chapter 6, “etx Index Parameters.”](#)

For more examples on how to create custom indexes, see [Chapter 3, “Tutorial.”](#)

Filtering

To avoid indexing binary data (which is not useful in **etx** searches), filter your documents before they are indexed. Filtering refers to the process of stripping away all the proprietary formatting information from a document so that only its text content remains in ASCII format.

For example, Microsoft Word documents usually contain formatting information that describes the fonts, paragraph styles, character styles, and layout of the text. Although this information can be indexed, it is not really useful for users who want to search the content of the document. Its inclusion in an **etx** index can significantly increase the size of the index and affect the performance of text searches. Filtering removes all this information and leaves just standard ASCII text.

To create an index on the filtered text of a column, specify the **FILTER** index parameter when you create your **etx** index. The following statement, for example, creates an **etx** index on the **abstract** column of the **my_table** table and specifies that the documents in the **abstract** column are to be filtered before they are added to the index:

```
CREATE INDEX abstract_index ON my_table (abstract etx_clob_ops)
  USING etx (FILTER = 'STOP_ON_ERROR');
```

For more detailed information about the **FILTER** index parameter and related requirements, refer to [“FILTER” on page 6-5](#).

For a full list of proprietary formats you can filter with the Excalibur Text Search DataBlade module, refer to [Appendix B, “Document Formats You Can Filter.”](#)

The `etx_contains()` Operator

You use the `etx_contains()` operator to perform searches of **etx** indexes. This section explains how to perform simple searches such as *keyword* and *Boolean searches* using the `etx_contains()` operator. These and other, more complex types of searches are described in more detail in [Chapter 2, “Text Search Concepts.”](#)

Performing Simple Keyword Searches

The `etx_contains()` operator is defined for the DataBlade module to allow you to define, qualify, and fine-tune searches of text. The `etx_contains()` operator takes three arguments: the first two required, the third optional. Use the first argument to specify the column that contains your search text. Use the second argument to specify what you are searching for, called a *clue* or a *search string*. The third optional argument is a *statement local variable* (SLV), used to return scoring information from a search.

To perform a search, use the `etx_contains()` operator in the WHERE clause of a SELECT statement. For example, search the column **description** for the word `multimedia`:

```
SELECT id, description FROM videos
      WHERE etx_contains(description, 'multimedia');
```


This example illustrates a *keyword search*, the default search type, shown in [Figure 1-3](#). The top section of the figure shows a table and its contents, and the bottom section shows the results of the search.

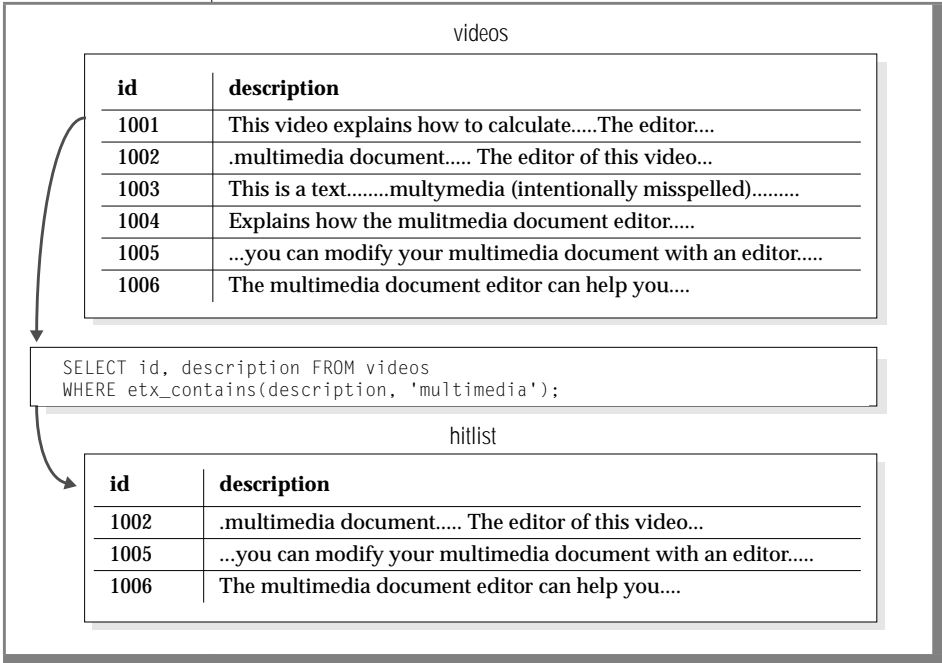


Figure 1-3
*Simple Keyword
Search Illustrated*

The example query can also be specified as:

```
SELECT id, description FROM videos
WHERE etx_contains(description, Row('multimedia'));
```

This example creates the clue `multimedia` by using the `Row()` constructor. If you do not specify any of the tuning parameters described in [“Tuning Parameters” on page 5-8](#), the `Row()` constructor in the `etx_contains()` operator is optional. For more information on the `Row()` constructor, see the [Informix Guide to SQL: Syntax](#).

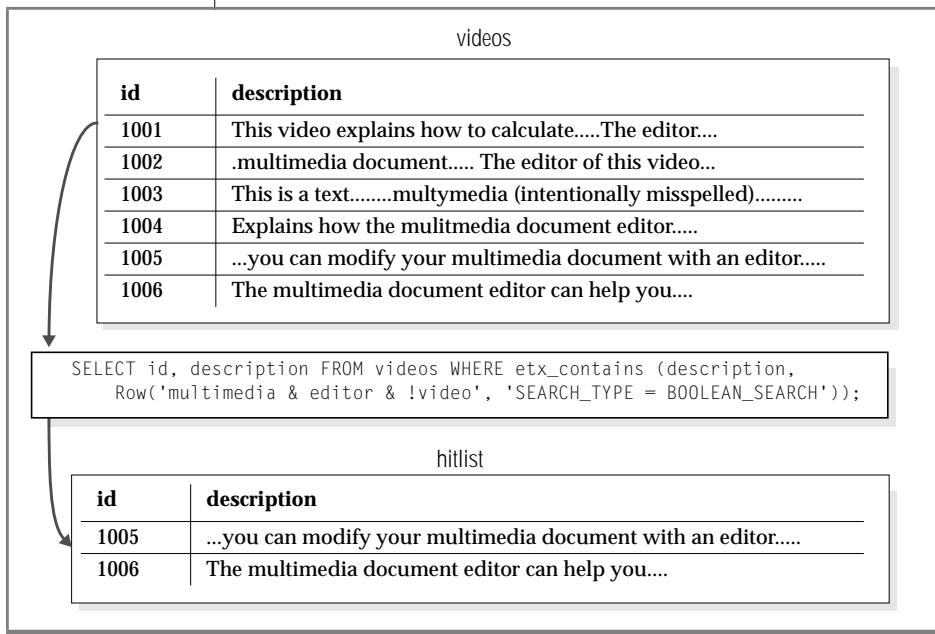
Performing Boolean Searches

You perform a Boolean search when you use any Boolean expression in a text search. For example, to search for text in the **description** column that contains the words **multimedia** and **editor** but not the word **video**, execute the following SQL statement:

```
SELECT id, description FROM videos
WHERE etx_contains (description,
Row('multimedia & editor & !video',
'SEARCH_TYPE = BOOLEAN_SEARCH'));
```

In this example, the **&** symbol represents a logical AND operator. You can also use the **|** symbol to represent a logical OR and the **!** or **^** symbols to represent the logical NOT. [Figure 1-4](#) displays the qualifying rows, also called a *hitlist*, when the query in the example is run on the **videos** table.

Figure 1-4
Sample Text Search
Query That Uses a
Boolean Expression



You can use the third, optional argument of the `etx_contains()` operator to pass an SLV to the search engine. The search engine uses this variable to record the document score it assigns to each row in the hitlist and to record highlighting information. The data type of the SLV is `etx_ReturnType`, an Informix-defined row type that consists of two fields that together contain the scoring and highlighting information.

For more information about using the scoring information of the SLV, see [“Document Scoring for Fuzzy Searches” on page 2-16](#). For more general information on SLVs, see the [Informix Guide to SQL: Syntax](#).

For more information on the `etx_ReturnType` data type, its two fields, and how to use the information contained in the fields, see [“etx_ReturnType” on page 4-4](#).

For an overview of the types of searches you can execute with the Excalibur Text Search DataBlade module, see [Chapter 2, “Text Search Concepts.”](#)

[Figure 1-5](#) provides a summary of the `etx_contains()` syntax used thus far.

Figure 1-5
Summary of `etx_contains` Syntax

```
CREATE INDEX desc_idx ON videos (description etx_clob_ops) USING etx();
```

Column of type BLOB, CLOB, LVARCHAR,
CHAR, VARCHAR, lfxDocDesc, or lfxMRData

```
SELECT * FROM videos WHERE etx_contains(description, Row('multimedia'));
```

Indexed column

Search string

Routines Defined for the DataBlade Module

In addition to the **etx_contains()** operator, the Excalibur Text Search DataBlade module includes several routines that you can use to perform tasks, such as creating and dropping synonym and stopword lists.

An example of one of these routines is **etx_CreateSynWlst()**. This routine allows you to create a list of words that you want the search engine to treat as synonyms. To use this procedure to create a synonym list, you use the EXECUTE PROCEDURE statement as shown in the following example:

```
EXECUTE PROCEDURE etx_CreateSynWlst  
('syn_list', '/local0/excal/synonyms','sbsp1');
```

This statement creates a synonym list named **syn_list** from an operating system file called **/local0/excal/synonyms** and stores it in the sbspace called **sbsp1**.

For a complete list of Excalibur Text Search DataBlade module routines and the functions they perform, see [Chapter 5, “Routines.”](#)

Word Lists

The Excalibur Text Search DataBlade module supports two types of word lists: synonym lists and stopword lists. Although you can perform successful searches without word lists, using them can be beneficial in ways that are explained in the following sections.

Synonym Lists

A *synonym list* consists of a *root word* and one or more words whose meaning is similar to the root word. Synonym lists are useful when you do not know the exact content of the text you are searching.

For example, suppose you want to search the **description** column of the **videos** table. You know that the **description** column contains references to videos that explain how to use multimedia document editors, but you are not sure if the editors are consistently described as multimedia *document* editors or multimedia *text* editors.

To perform this search, use a synonym list entry for the words *document* and *text*.

To instruct the search engine to use this word list when you perform a search, you use the `MATCH_SYNONYM` tuning parameter, as shown in the following example:

```
SELECT id, description from videos
WHERE etx_contains(description,
Row('document', 'MATCH_SYNONYM = syn_list'));
```

You can maintain multiple synonym lists within a database. You specify which synonym list you want the search engine to use by setting `MATCH_SYNONYM` to the name of the synonym list. For example, to make **syn_list2** the active synonym list while querying the **videos** table, execute the following statement:

```
SELECT id, description from videos
WHERE etx_contains(description,
Row('document', 'MATCH_SYNONYM = syn_list2'));
```

To create a synonym list, first create an operating system file that contains root words with one or more synonyms, all on one line and separated by blanks. The lines of text must be separated by one blank line. For example, the following is a possible excerpt from an operating system file that will be used to create a synonym list:

```
quick speedy fast

monitor terminal CRT screen
```

In this example, *quick* and *monitor* are the root words. A word must be present as a root word for the synonyms of the word to be used. This means that if you want to search for synonyms of *speedy*, it must itself be listed as a root word; it is not enough to simply exist as a synonym for *quick* in the synonym list.

After you have created a synonym list file, you can make it known to the Excalibur Text Search DataBlade module by executing the **etx_CreateSynWlst()** routine. See “[etx_CreateSynWlst\(\)](#)” on page 5-22 for details about this routine.

Important: You must put a blank line between the lines of text in the synonym file. If you omit the blank lines, the DataBlade module does not return an error, but it never finds any synonyms during a synonym matching search.



If you specify the `MATCH_SYNONYM` tuning parameter in the `etx_contains()` operator but do not set it equal to a value, `etx_contains()` refers to a default synonym list named `etx_thesaurus`. You can create your own `etx_thesaurus` synonym list from your own list of synonyms, or you can create one based on a list of standard English-language synonyms provided with your DataBlade module in the following location:

```
$INFORMIXDIR/extend/ETX.version/wordlist/etx_thesaurus.txt
```

version refers to the current version of the DataBlade module installed on your computer.

If a query refers to the `etx_thesaurus` synonym list, and the list does not exist, the Excalibur Text Search DataBlade module returns an error.

For more information on how to create and drop synonym lists, see [“etx_CreateSynWlst\(\)” on page 5-22](#) and [“etx_DropSynWlst\(\)” on page 5-29](#), respectively.



Important: *The Excalibur Text Search DataBlade module finds pattern matches of root words only, not of their synonyms.*

For example, assume you execute a text search and specify the `PATTERN_ALL` and `MATCH_SYNONYM` tuning parameters and that the specified synonym list contains the root word “abandon” with one synonym: “surrender.” The search returns documents that contain the word “abanden,” a pattern match of the root word “abandon,” but does not return documents that contain the word “surender,” a pattern match of the synonym “surrender.”

For more information about pattern searches, see [“Pattern Search” on page 2-12](#).

Stopword Lists

A *stopword* is a word that you want excluded from your index and, as a consequence, from your searches. A typical stopwords list includes words like *of*, *the*, and *by*. Stopword lists depend on the content and type of your data. Any frequently occurring word that you want excluded from your index is a candidate for inclusion in a stopwords list. Stopword lists can reduce the time it takes to perform a search, reduce index size, and help you avoid false hits.

To create and drop stopwords lists, you use procedures defined for the DataBlade module. For example, to create a stopwords list, first create an operating system file that contains the list of stopwords, one word per line. Then make the stopwords list known to the Excalibur Text Search DataBlade module by executing the procedure **etx_CreateStopWlst()**, as shown in the following example:

```
EXECUTE PROCEDURE etx_CreateStopWlst
('stopwlist', '/local0/excal/stopwlist');
```

This statement creates the stopwords list **stopwlist** from the operating system file **/local0/excal/stopwlist**. An optional third argument can be used to specify the sbpace where the list is to be stored. If you do not specify a specific sbpace to store the list, it is stored in the default sbpace. The default sbpace is specified by the SBSPACENAME parameter in the ONCONFIG file.

You can create your own stopwords list file, or you can create one based on a list of standard English-language stopwords provided with your DataBlade module in the following location:

```
$INFORMIXDIR/extend/ETX.version/wordlist/etx_stopwords.txt
```

version refers to the current version of the DataBlade module installed on your computer.

You can have at most one stopwords list associated with an **etx** index. The stopwords list is specified when the index is initially created using the index parameter STOPWORD_LIST. The stopwords list must exist when the **etx** index is created.

At times, you might want to include words in a search that currently exist in your stopwords list. For example, suppose that the following words exist in your stopwords list: *to*, *or*, and *be*. Suppose further that you want to search for the exact phrase “to be or not to be.” To occasionally search for stopwords using an **etx** index that has a stopwords list associated with it, specify the `INCLUDE_STOPWORDS` index parameter when you create the index. Then use the `CONSIDER_STOPWORDS` tuning parameter when you execute the search. The `CONSIDER_STOPWORDS` parameter forces the search engine to include words that you previously stipulated as stopwords. For example, you can search for the phrase `to be or not to be` as follows:

```
SELECT id, description FROM videos
WHERE etx_contains(description,
  Row('to be or not to be',
    'SEARCH_TYPE = PHRASE_EXACT & CONSIDER_STOPWORDS')));
```



Important: The `CONSIDER_STOPWORDS` tuning parameter of the `etx_contains()` operator works only if the `INCLUDE_STOPWORDS='TRUE'` index parameter is specified for the `CREATE INDEX` statement that creates the **etx** index.

For more information on how to create and drop stopwords lists, see [“etx_CreateStopWlst\(\)” on page 5-20](#) and [“etx_DropStopWlst\(\)” on page 5-27](#), respectively.

Character Sets

Character sets define the characters that are indexed when an **etx** index is built on a column that contains text documents. Any character in the text document not found in the specified character set is treated as white space in the index. The character is not changed in the text document itself.

Specify a character set by setting the `CHAR_SET` index parameter to the character set’s name when you create an **etx** index with the `CREATE INDEX` statement.

The Excalibur Text Search DataBlade module provides three built-in character sets: `ASCII`, `ISO`, and `OVERLAP_ISO`. You can also define your own character set if the ones provided are not adequate for your text documents. The following sections describe the built-in character sets and explain when and how to create your own.

Built-In Character Sets

ASCII is the default character set that is used if you do not specify the CHAR_SET index parameter in the CREATE INDEX statement. The ASCII character set includes the numbers 0 through 9, the uppercase letters A through Z, and the lowercase letters a through z. All lowercase letters are mapped internally to uppercase, which means that searches that use the ASCII character set are case-insensitive.

The ISO and OVERLAP_ISO character sets extend the ASCII character set by including many international characters such as Å and ñ. The OVERLAP_ISO character set maps similar international characters to a single ASCII character.

For more information about specifying character sets with the CHAR_SET index parameter, refer to [“CHAR_SET” on page 6-4](#). For specifications on the built-in character sets, refer to [Appendix A, “Character Sets.”](#)

User-Defined Character Sets

At times, the built-in character sets might be inadequate for your text documents or the types of searches you plan to perform. For example, you might want to index the hyphen character to be able to index and search for hyphenated words such as *English-language*. Since the three built-in character sets index only alphanumeric characters, you must create your own character set to index the hyphen character.

You must create a user-defined character set before you use it to create an **etx** index. You create and drop user-defined character sets with the routines **etx_CreateCharSet()** and **etx_DropCharSet()** provided by the DataBlade module. See [“etx_CreateCharSet\(\)” on page 5-16](#) and [“etx_DropCharSet\(\)” on page 5-25](#) for more information about these routines.

The **etx_CreateCharSet()** routine takes two parameters: the name of the new user-defined character set and the full pathname of an operating system file that contains a description of the character set. The operating system file contains a 16 x 16 matrix of hexadecimal numbers that represents which characters are indexed. Refer to [“ISO 8859-1” on page A-7](#) when you create your matrix.

The following example shows how to execute the **etx_CreateCharSet()** routine to create a new user-defined character set called **my_charset** from the description contained in the operating system file named **/local0/excal/my_char_set_file**:

```
EXECUTE PROCEDURE etx_CreateCharSet
    ('my_charset', '/local0/excal/my_char_set_file');
```

The following example shows how to create an **etx** index that uses the user-defined character set **my_charset** by specifying it as an option to the **CHAR_SET** index parameter:

```
CREATE INDEX desc_idx2 ON videos (description etx_clob_ops)
    USING etx (WORD_SUPPORT = 'PATTERN', CHAR_SET = 'my_charset')
    IN sbsp1;
```

Refer to **“CHAR_SET” on page 6-4** for details about the **CHAR_SET** index parameter.

Highlighting

The Excalibur Text Search DataBlade module enables you to highlight the occurrences of a clue in search results. *Highlighting* is the process of retrieving the location of every instance of a *clue* in the *search text*.

You can use the **etx_GetHilite()** routine in the **SELECT** list of a query to return highlighting information from a text search, as shown in the following example:

```
SELECT etx_GetHilite (description, rc) FROM videos
    WHERE etx_contains(description,
        'multimedia', rc # etx_ReturnType);
```

This query returns all documents that contain the keyword **multimedia** in the **description** column of the table **videos**. In addition, for each row returned, the query also returns **etx_HiliteType** values that contain information about the location of every instance of the word **multimedia** in the corresponding document.



Important: If you use the **etx_GetHilite()** function in a query that returns more than one row, the function executes once for each row. This means that each row has its own highlighting information, contained in the **etx_HiliteType** row data type returned by the **etx_GetHilite()** function. This highlighting information pertains only to the document contained in the row, not to any other document.

The **etx_HiliteType** row data type consists of two fields: **vec_offset** and **viewer_doc**. The **vec_offset** field contains pairs of integers that describe the location of every instance of the highlight string in the document. The **viewer_doc** field contains the text document itself.

You can highlight occurrences of a clue in an individual document you have retrieved with an **etx_contains** scan with the **etx_HiliteDoc()** routine.

You can view the highlighted text contained in the **etx_HiliteType** object by using **etx_ViewHilite()** routine.

For detailed information about the data type used for highlighting, see [“etx_HiliteType” on page 4-6](#).

For detailed information about highlighting routines, refer to [Chapter 5, “Routines.”](#)

You can also use standard Informix ESQL/C routines, such as **ifx_lo_open()** and **ifx_lo_read()**, or DataBlade API large object routines, such as **mi_lo_open()** and **mi_lo_read()**, to manipulate **etx_HiliteType** objects. Refer to the [INFORMIX-ESQL/C Programmer's Manual](#) and the [DataBlade API Programmer's Manual](#) for information about these routines.

Text Search Concepts

In This Chapter	2-3
Excalibur Text Search Types	2-3
Keyword Searches	2-3
Boolean Searches	2-5
Phrase Searches	2-7
Exact Phrase Searches	2-7
Approximate Phrase Searches	2-9
Proximity Searches	2-9
Performing Fuzzy Searches	2-11
Word Scoring for Fuzzy Searches	2-11
Substitution and Transposition	2-12
Pattern Search	2-12
Phrase Searching with Pattern Matching	2-14
Document Scoring for Fuzzy Searches	2-16
Limiting the Number of Rows a Fuzzy Search Returns	2-19
MAX_MATCHES	2-19
WORD_SCORE	2-20
The score Field of etx_ReturnType	2-20

In This Chapter

This chapter discusses the different types of searches you can perform with the Excalibur Text Search DataBlade module and characterizes how the text search engine assigns word scores and document scores.

Excalibur Text Search Types

The Excalibur Text Search DataBlade module supports a variety of searches, such as keyword, Boolean, proximity, pattern (or fuzzy), and phrase. Phrase searching can be classified into two types: exact and approximate.

Use the `SEARCH_TYPE` *tuning parameter* to indicate what kind of search you want to perform. A tuning parameter is a variable that you use to guide the way in which the text search engine performs a search. You pass this parameter to the search engine as the second input parameter of the `Row()` constructor of the `etx_contains()` operator.

The following sections provide examples of different types of searches. The examples use the same **videos** table described in Chapter 1. Refer to [“etx_contains\(\)” on page 5-5](#) for a full list of supported tuning parameters.

Keyword Searches

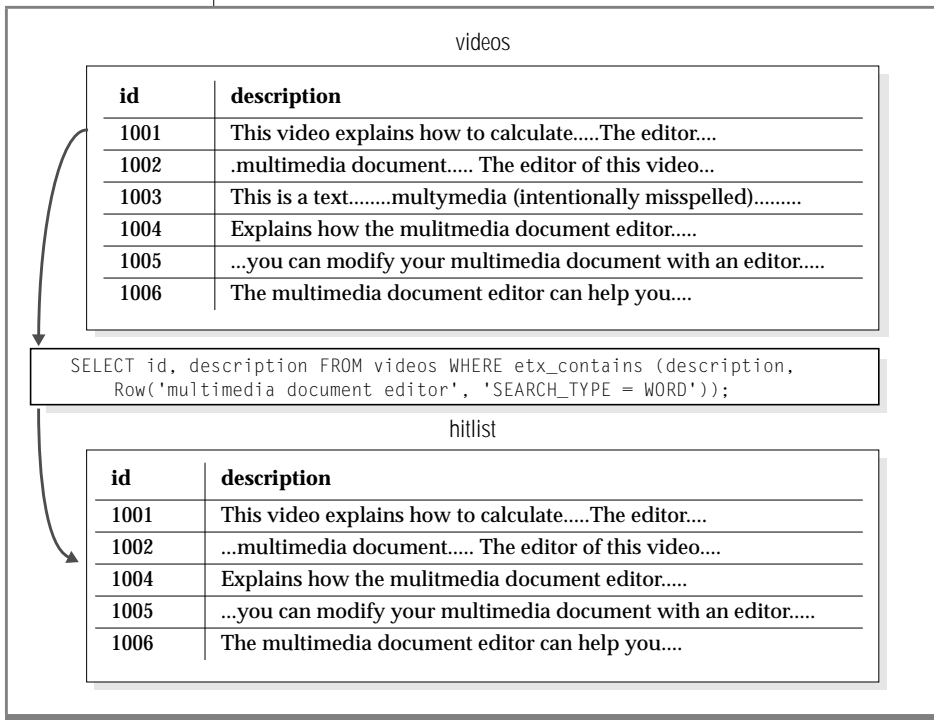
When the clue you specify contains more than one word, you can direct the search engine to treat each word as a separate entity. This type of search is called a *keyword search*. When the text engine performs a keyword search, it returns a row whenever it encounters one or more of the words in your clue.

You specify a keyword search by setting the `SEARCH_TYPE` tuning parameter to `WORD` and passing it to the search engine as the second parameter of the `Row()` constructor. For example, the following query instructs the search engine to return any row that contains one or more occurrences of the keywords `multimedia`, `document`, or `editor` in the `description` column:

```
SELECT id, description FROM videos
WHERE etx_contains (description,
Row('multimedia document editor', 'SEARCH_TYPE = WORD'));
```

Figure 2-1 illustrates this example.

Figure 2-1
Example of
Keyword Search



The search did not return the row with ID 1003 because the word `multimedia` is misspelled and the text does not contain the other two words in the clue. Even though the word `multimedia` is misspelled in the row with ID 1004, the row is still returned because it contains the other two words in the clue, `document` and `editor`.

The search engine assigns the rows returned by a keyword search a *document score*. The document score is based on the number of keywords found in a document. For example, a document that contains two of three keywords is scored twice as high as a document that contains only one of the keywords.

If you do not specify the `SEARCH_TYPE` tuning parameter in the **etx_contains()** operator, the text search engine defaults to keyword search. This means that the following two searches are equivalent and are therefore often used interchangeably in this manual:

```
SELECT id, description FROM videos
WHERE etx_contains (description,
Row('multimedia document editor'));

SELECT id, description FROM videos
WHERE etx_contains (description,
Row('multimedia document editor', 'SEARCH_TYPE = WORD'));
```

Boolean Searches

A Boolean search enables you to combine the keywords to create more complicated clues.

To specify documents that contain both the words `multimedia` and `document` but not the word `video`, use a Boolean search, as shown in the following example:

```
SELECT id, description FROM videos
WHERE etx_contains (description,
Row('multimedia & editor & !video',
'SEARCH_TYPE = BOOLEAN_SEARCH'));
```

The Boolean operators `&` and `!` are used to build the desired Boolean expression. You specify a Boolean search by setting the `SEARCH_TYPE` tuning parameter to `BOOLEAN_SEARCH`.

If you have created an **etx** index with **PHRASE_SUPPORT** enabled, you can specify Boolean phrases in your **etx_contains()** syntax, as shown in the following example:

```
SELECT id, description FROM videos
WHERE etx_contains (description,
Row ('vanilla wafers | chocolate chip cookies',
'SEARCH_TYPE = BOOLEAN_SEARCH'));
```

This query returns relevant records that contain either of the phrases vanilla wafers **or** chocolate chip cookies.



Warning: You encounter an error if you specify a Boolean phrase search on a column that has not been indexed with **PHRASE_SUPPORT**.

To perform a Boolean search for clues that themselves contain a character that represents a Boolean operator, escape the character with backslash, as shown in the following example:

```
SELECT id, description FROM videos
WHERE etx_contains (description,
Row ('Lord \& Taylor | Nordstrom',
'SEARCH_TYPE = BOOLEAN_SEARCH'));
```

This query returns relevant records that contain either of the names Lord & Taylor **or** Nordstrom. If the Boolean **&** operator were not escaped, the query would have returned documents that include the words Lord and Taylor or documents that include the word Nordstrom.

You can escape the Boolean operators **&**, **|**, **!**, and **^** by using the backslash.



Important: The search engine always applies a document score of 100 to all rows returned by a Boolean search, even when the Boolean search is combined with a pattern search. The search engine cannot assign a meaningful relative ranking to hits since a Boolean clue might be composed of many parts, and each hit might result from a different part of the clue. The search engine, therefore, scores Boolean searches arbitrarily—100—even in cases where it might seem to a user that a meaningful score could be applied.

Phrase Searches

A *phrase* is any clue that contains more than one word. In a phrase search, the text search engine treats the whole phrase as a single unit.

The Excalibur Text Search DataBlade module search supports two types of phrase searches: exact phrase search and approximate phrase search.



Important: *Exact and approximate phrase searches can be performed only on columns whose **etx** indexes were created with the **PHRASE_SUPPORT** index parameter. You encounter an error if you search for a phrase in a column whose index was not created with **PHRASE_SUPPORT**.*

Exact Phrase Searches

An exact phrase search returns only phrases that contain all the words in the clue in the exact order that you specify. To execute an exact phrase search, you set the **SEARCH_TYPE** tuning parameter to **PHRASE_EXACT**, as shown in the following example:

```
SELECT id, description FROM videos
WHERE etx_contains(description,
  Row ('multimedia document editor','SEARCH_TYPE = PHRASE_EXACT'));
```

Figure 2-2 displays the resulting hitlist when the query in the example is run on the **videos** table. Since an exact phrase search with no pattern matching is specified, the document that contains the phrase `multimedia document editor` (multimedia intentionally misspelled) is not returned.

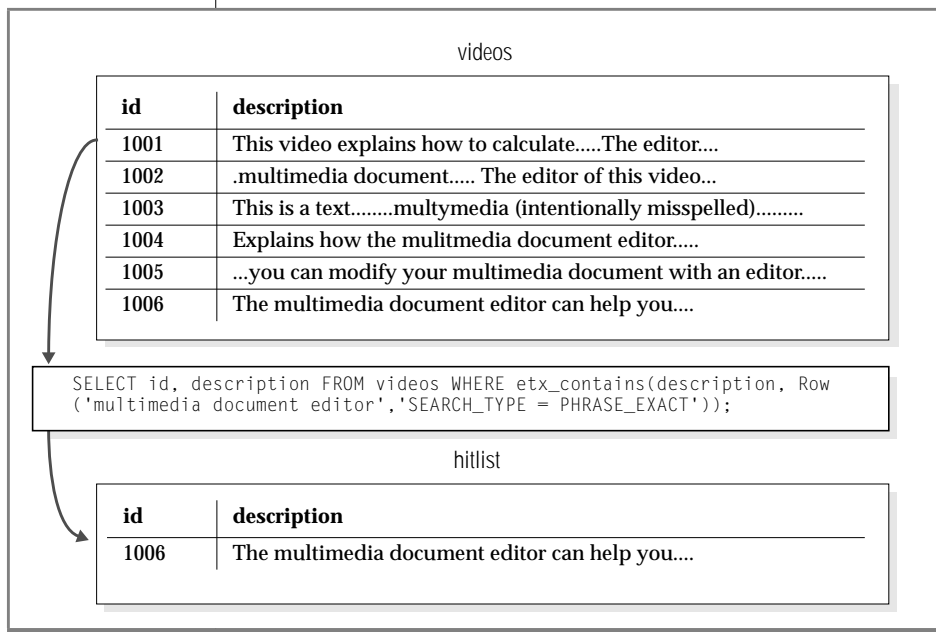


Figure 2-2
Example of Exact
Phrase Search

An exact phrase search for a clue that contains a stopword returns zero rows, even if the clue is contained in a document. This happens only if the **etx** index ignores stopwords, or in other words, if the index was created with the **STOPWORD_LIST** index parameter and the stopword list contains one or more words in the clue.

For example, assume the **etx** index was created with the index parameter **STOPWORD_LIST = 'my_list'**, and that the stopword list **my_list** includes the word *the*. In this case, an exact phrase search for the clue “walk the dog” returns zero rows, even if this exact phrase is contained in a document.

There are two ways to work around this behavior:

- Use an approximate phrase search instead of an exact phrase search. Approximate phrase searches, however, can be slower than exact phrase searches.

- Include stopwords in the **etx** index by specifying the `INCLUDE_STOPWORDS` index parameter when you create the index. Then specify that stopwords must be considered when executing an exact phrase search for the phrase “walk the dog” by specifying the `CONSIDER_STOPWORDS` and `SEARCH = PHRASE_EXACT` tuning parameters in the **etx_contains()** operator.

Approximate Phrase Searches

An approximate phrase search returns all phrases in the search text that contain words that are approximately the same as the words you specify in the clue. To execute an approximate phrase search, set the `SEARCH_TYPE` tuning parameter to `PHRASE_APPROX`, as shown in the following example:

```
SELECT * FROM videos
WHERE etx_contains(description,
  Row ('document editor', 'SEARCH_TYPE = PHRASE_APPROX'));
```

Use an approximate phrase search when you want to search for a phrase, but you do not know or remember all the words in a phrase. Use an exact phrase search with pattern matching if you know all of the words in a phrase, but are not sure how one or more of the words in the phrase is spelled.



Tip: Do not set `SEARCH_TYPE` to `PHRASE_APPROX` and pass the `PATTERN_BASIC` or `PATTERN_ALL` tuning parameters for the same search. The resulting query might perform poorly. The tuning parameters `PATTERN_BASIC` and `PATTERN_ALL` are discussed in [“Phrase Searching with Pattern Matching” on page 2-14](#).

Proximity Searches

When you perform a word search for a phrase that contains multiple words, the search engine returns any row that contains one or more of the words in the clue.

For example, suppose you want to search for the phrase `multimedia document editor`, as shown in the following example:

```
SELECT id, description FROM videos
WHERE etx_contains(description,
  Row('multimedia document editor',
    'SEARCH_TYPE = WORD'));
```

If you execute this search on the following row, the search engine reports a match, even though the sentence is probably not of interest to you (because it has nothing to do with a multimedia document editor):

```
The multimedia application is now in formal beta testing.
```

On the other hand, if you use an exact phrase search, the search engine fails to return rows similar to the following phrase:

```
The editor for multimedia documents .....
```

A proximity search resolves these problems by allowing you to specify the number of nonsearch words that can occur between two or more search words. For example, consider the following word search:

```
SELECT id, description FROM videos
WHERE etx_contains(description,
  Row('multimedia document editor',
    'SEARCH_TYPE = PROX_SEARCH(8)'));
```

The last line uses the `PROX_SEARCH` setting to specify that the search engine is to return a row only when each of the three search words occurs within eight words of the last word matched. Suppose you executed this query on the following sample text:

```
The multimedia application is now in formal beta testing. A
text editor.....
```

In this example, the search engine does not return this row because it has more than eight words between `multimedia` and `editor`. However, the search engine does return the following row because it has fewer than eight words between each of the search words:

```
The editor for multimedia documents .....
```

The following statement always fails to produce results because the value specified with `PROX_SEARCH` is fewer than the number of words to search for:

```
SELECT id, description FROM videos
WHERE etx_contains(description ,
  Row('multimedia document editor',
    'SEARCH_TYPE = PROX_SEARCH(2)'));
```



Performing Fuzzy Searches

A *fuzzy search* is a search for text that matches your clue closely instead of exactly. Fuzzy searches can also be referred to as *pattern searches*. Fuzzy searches help you return results of interest even when the search terms themselves are misspelled in the documents you are searching.

Important: *Fuzzy or pattern searches can be performed only on columns whose **etx** indexes were created with the `WORD_SUPPORT = PATTERN` index parameter.*

This section discusses the types of fuzzy searches you can perform with the Excalibur Text Search DataBlade module and describes, in general terms, how the search engine scores both word-level matches and document matches.

Word Scoring for Fuzzy Searches

If you have created an index with the `WORD_SUPPORT = PATTERN` index parameter, the search engine considers words that match your clue approximately as well as words that match your clue exactly. The search engine uses fuzzy logic to determine whether or not a pattern match should be considered a hit. It assigns a *word score* to candidate matches based on its internal rules. By default, only words that match your search clue by a relative measure of 70 out of 100—that have a word score of 70 or better—are considered hits. You can change the default by specifying your own value for the `WORD_SCORE` tuning parameter.

For example, suppose you want the text search engine to count only as hits words that match your clues by a measure of 85 or better. You would specify this condition by setting `WORD_SCORE = 85` in the **etx_contains()** operator, as shown in the following example:

```
SELECT id, description FROM videos
WHERE etx_contains ( description,
  Row('multimedia document editor' ,
    'PATTERN_TRANS & PATTERN_SUBS' & WORD_SCORE = 85));
```

The following sections show how you perform the types of fuzzy searches enabled by the Excalibur Text Search DataBlade module.

Substitution and Transposition

This section discusses the simplest example of a fuzzy search: a search that takes into account misspellings in both the clue and the search text. Misspellings most frequently occur when letters in a word are mistakenly substituted or transposed. Misspelling *editor* as *editer* is an example of substitution; misspelling *multimedia* as *mulitmedia* is an example of transposition. Transposition means switching the order of two *adjacent* characters.

Pattern Search

A pattern search is a fuzzy search that takes into account multiple transpositions and substitutions and also returns any substrings or superstrings of the clue. To enable a pattern search, you pass the `PATTERN_ALL` tuning parameter to the search engine.

For example, to perform a pattern search for the word `retrieve`, execute the following SQL statement:

```
SELECT id, description FROM videos
WHERE etx_contains(description,
  Row ('retrieve','PATTERN_ALL'));
```


Given the search text shown in Figure 2-3, the search engine returns rows that contain retrieved, retrieval, or retrieve. Because of the high default word score threshold of 70, the word irretrievable is not found. However, if you use the WORD_SCORE tuning parameter to set the word score lower, the word irretrievable might be returned.

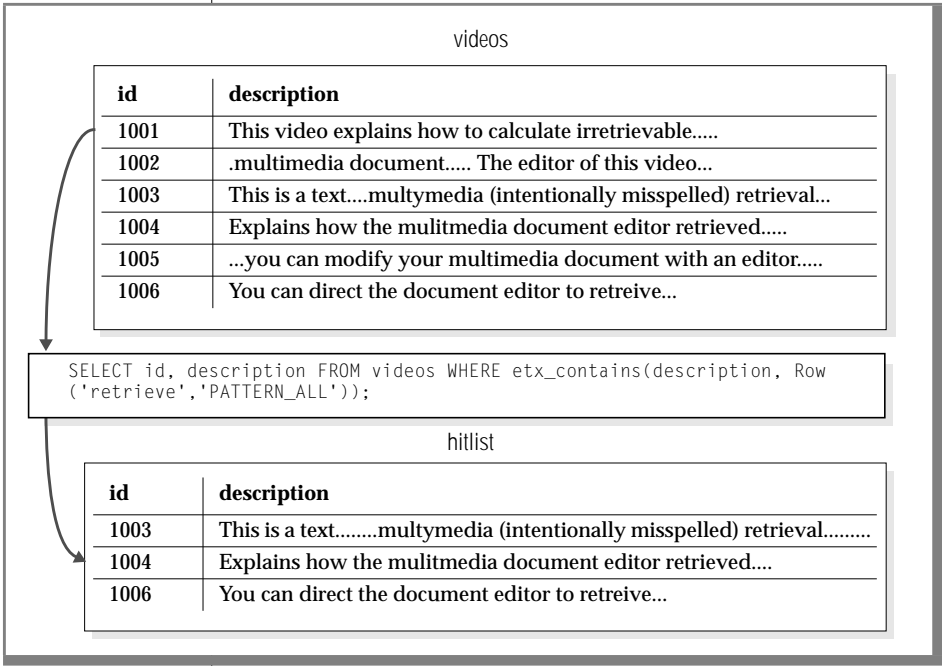


Figure 2-3
Example of Pattern Search

A pattern search differs from a substitution or a transposition search in that a pattern search allows multiple substitutions and transpositions. A substitution search returns words that differ from the clue by a *single* substitution. For example, a search for the word *travel* with PATTERNS_SUBS set might return *gravel*. A transposition search returns words that contain a single transposition. For example, a search for the word *travel* with PATTERN_TRANS set might return *travle*.

In contrast, a search for the same clue with PATTERN_ALL enabled returns words such as *traveled*, *travelled*, *unraveled*, and *travvel*, in addition to *gravel* and *travle*. The text search engine assigns a higher word score to words matched with PATTERNS_SUBS or PATTERN_TRANS enabled than it does to words that are matched with PATTERN_ALL enabled.

If you want to enable basic pattern matching, use the `PATTERN_BASIC` tuning parameter. The search might return transpositions, substitutions, and superstring and substring pattern matches, depending on the value of `WORD_SCORE`.

The `PATTERN_ALL` tuning parameter is equivalent to specifying the three parameters `PATTERN_BASIC`, `PATTERN_SUBS`, and `PATTERN_TRANS` at once.

Phrase Searching with Pattern Matching

When you perform a search with `SEARCH_TYPE = PHRASE_EXACT`, the search text must contain a phrase that is identical to the clue for a hit to occur. If you specify a pattern search in addition to an exact phrase search (by specifying `PATTERN_ALL`, for example), the individual words in the clue must pattern-match the corresponding words in the search text in the same order in which the words appear in the search text.

For example, in a pattern search combined with an exact phrase search, the text `jill john jones` matches the clue `jyll jonh gones` but does not match the clue `john jill jones`. That is, order always counts in an exact phrase search, regardless of whether you also specify pattern matching.

In an exact phrase search, all words in the clue (or pattern matches thereof) must be found in the search text. Partial matches, where one or more words are missing, do not count as hits.

When you perform a search with `SEARCH_TYPE = PHRASE_APPROX`, the search text must contain either a phrase identical to the clue, one or more words of the clue in the same order, or one or more words of the clue in a different order. The Excalibur Text Search DataBlade module uses the number of words and word order to produce a document score for all hits; the more words in closer order a search text has, the higher the score it is assigned.

For example, if the clue is “drop many balls,” the search engine produces the following ranking based on scores:

```
He can drop many balls(exact match)
He has many balls (2 words, in same order as clue)
He let five balls drop(2 words, different order from clue)
He has many children(1 word)
```

A keyword pattern search works like a search with `SEARCH_TYPE = PHRASE_APPROX`. However, a pattern search adds pattern matching on a word-by-word basis. For example, if the clue is “drop many balls,” then the search engine produces the following ranking:

```
He can dorp many valls(assuming dorp pattern matches drop, and so on)
He has mani balds
He let five galls frop
He has many children
```



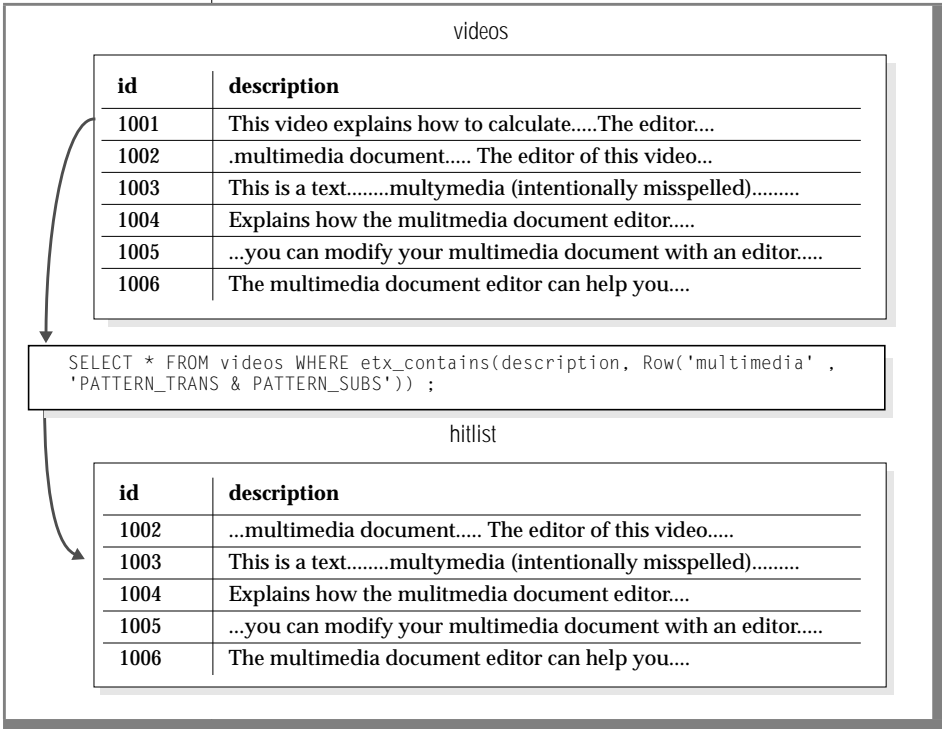
Tip: Informix recommends that you refrain from setting `SEARCH_TYPE` to `PHRASE_APPROX` and passing the `PATTERN_BASIC` or `PATTERN_ALL` tuning parameters for the same search. Doing so can result in poor performance for searches of this type.

The tuning parameters that instruct the text search engine that you want transpositions and substitutions taken into account are `PATTERN_TRANS` and `PATTERN_SUBS`. Unlike the `SEARCH_TYPE` tuning parameter, `PATTERN_TRANS` and `PATTERN_SUBS` do not have values that you set. Instead, you pass these tuning parameters directly to the search engine using the Boolean operator `&`, as shown in the following example:

```
SELECT * FROM videos
WHERE etx_contains(description,
  Row('multimedia' , 'PATTERN_TRANS & PATTERN_SUBS')) ;
```

This example initiates a fuzzy search for the `multimedia` clue. The search engine returns all words that have exactly one substitution or one transposition, as well as words that match the clue exactly. [Figure 2-4](#) displays the resulting hitlist when the query in the example is run on the `videos` table.

Figure 2-4
Sample Text Search
Query That Uses
Tuning Parameters



The tuning parameters `PATTERN_TRANS` and `PATTERN_SUBS` only take into account the transposition or substitution of a single letter per word.

Document Scoring for Fuzzy Searches

When you perform a fuzzy search, some of the rows that the text search engine returns might satisfy search criteria better than others. To determine the degree of similarity between your clue and each of the rows returned, you can instruct the search engine to assign a value, called a *document score*, to each of the rows.

The Excalibur Text Search DataBlade module uses the following two rules when it assigns document scores:

- It always scores exact matches slightly higher than pattern matches.
- It scores all pattern matches equally, even if some matches appear to approximate the clue better than others.

Document scores vary from 0 to 100, with 0 indicating no match and 100 indicating an exact match. Values between 0 and 100 indicate approximate matches; the higher the value, the closer the match. A null value indicates that the row was not ranked. This could happen if the row was returned because of a non-**etx_contains()** operator in a WHERE clause that contains an OR predicate.

To access document score information, use a statement local variable (SLV) as the optional third parameter to the **etx_contains()** operator. The data type of the SLV is **etx_ReturnType**, an Informix-defined row type that consists of two fields. The scoring information is contained in the **score** field.

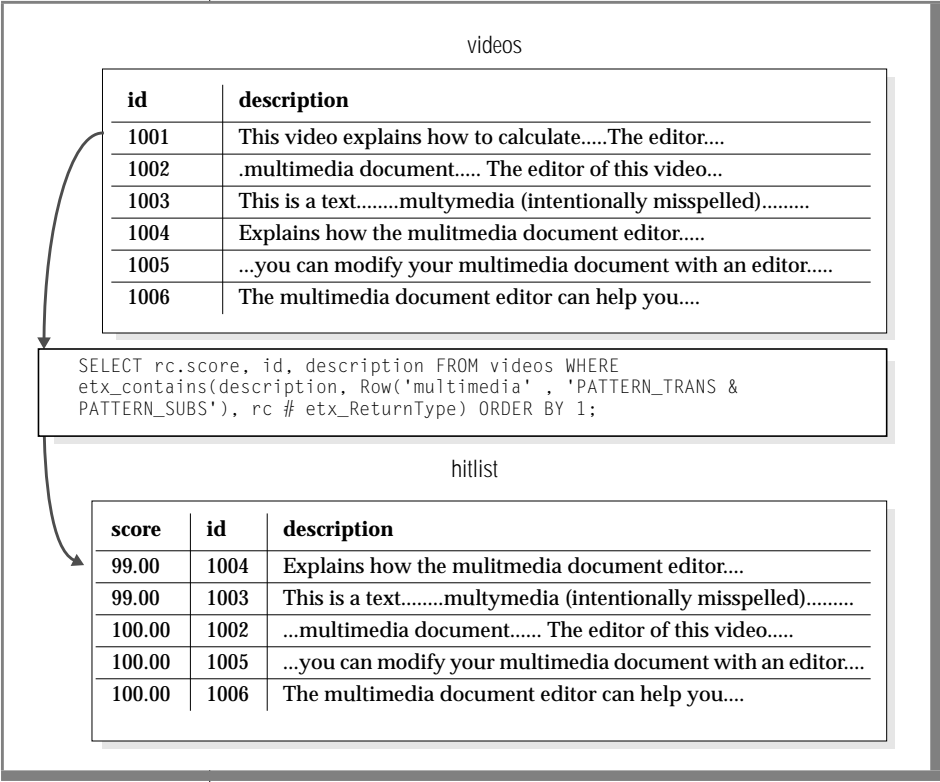
The **score** field contains a numeric value that indicates the relevance of a returned document to the search criteria, compared to that of other indexed records. The higher the document score value, the more closely the document matches the criteria.

You can use the **score** field to order the returned rows according to how closely documents match the search criteria. For example, if **rc** is an SLV, you can use it to obtain document score information, as shown in the following example:

```
SELECT rc.score, id, description FROM videos
WHERE etx_contains(description,
  Row('multimedia' ,
    'PATTERN_TRANS & PATTERN_SUBS'), rc # etx_ReturnType)
ORDER BY 1;
```

Figure 2-5 displays the resulting hitlist when the query in the example is run on the **videos** table. The column **score** in the hitlist contains scoring information.

Figure 2-5
Sample Text Search
Query That Uses
SLVs



The SLV has a scope that is limited to the statement in which you use it. It is a way for the Excalibur text search engine to send back information about the search it just performed to the **etx_contains()** operator that called it.

Although the data type of the SLV is always `etx_ReturnType`, you must still explicitly specify its type when you use it in the **etx_contains()** operator, as shown in the example. The example also shows how you can use the `ORDER BY` clause to instruct the database server to rank the rows it returns by the **score** field of the SLV.

For more information on the `etx_ReturnType` data type, its two fields, and how to use the information contained in the fields, see [“etx_ReturnType” on page 4-4](#). For more general information on SLVs, see the [Informix Guide to SQL: Syntax](#).

Limiting the Number of Rows a Fuzzy Search Returns

At times, you might find that the search engine returns more information than you require. The `MAX_MATCHES` and `WORD_SCORE` tuning parameters allow you to limit the number of rows that the search engine returns.

For example, suppose that, when you execute the following query, the text search engine takes a while to return one hundred rows, although you are interested in just the first three:

```
SELECT id, description FROM videos
WHERE etx_contains(description,
    Row('multimedia document editor' ,
        'PATTERN_TRANS & PATTERN_SUBS'));
```

MAX_MATCHES

You can specify the maximum number of rows the search engine returns by using the `MAX_MATCHES` tuning parameter, as shown in the following example:

```
SELECT id, description FROM videos
WHERE etx_contains(description,
    Row('multimedia document editor' ,
        'PATTERN_TRANS & PATTERN_SUBS & MAX_MATCHES=3'));
```

It is important to note that if you specify the `MAX_MATCHES` tuning parameter in a query, it is possible that not all rows that satisfy all the search criteria will be returned, since this is the nature of the tuning parameter. This can cause misleading results if a subsequent qualification, different from the `etx_contains()` operator, is applied in the query.

WORD_SCORE

You can also limit the number of pattern matches with the **WORD_SCORE** tuning parameter. Word score determines whether words are considered hits. Assigning a high value to **WORD_SCORE** results in fewer hits; assigning a low value to **WORD_SCORE** results in more, less exact matches. You can specify a high word score standard to limit the number of words that are considered matches, as shown in an earlier example and repeated here:

```
SELECT id, description FROM videos
WHERE etx_contains ( description,
Row('multimedia document editor' ,
'PATTERN_TRANS & PATTERN_SUBS & WORD_SCORE = 85'));
```

For a multiword clue, however, you have no way of knowing what the word score is for any one of the clues, so you might get unexpected results.

The score Field of etx_ReturnType

You can also limit the number of rows returned by setting a document score standard in the **score** field of the **etx_ReturnType**. The following query specifies that only those documents that have a document score of 85 or better should be returned:

```
SELECT rc.score, id, description FROM videos
WHERE etx_contains ( description,
Row('multimedia document editor' ,
'PATTERN_TRANS & PATTERN_SUBS'),
rc # etx_ReturnType) AND
rc.score > 85 ;
```

Refer to [“Document Scoring for Fuzzy Searches” on page 2-16](#) for an additional example of how to rank rows using the **rc.score** SLV and an **ORDER BY** clause.

Tutorial

In This Chapter	3-3
Creating a Table Containing a Text Search Column	3-4
Creating Dbspaces and Sbspaces for Storage.	3-4
Creating the Table	3-5
Populating the Table with Text Search Data	3-5
Creating Word Lists and a User-Defined Character Set	3-6
Creating a Stopword List	3-6
Creating a Synonym List	3-7
Creating a User-Defined Character Set.	3-8
Creating an etx Index	3-10
Determining the etx Index Parameters.	3-10
Creating an etx Index.	3-10
Creating a Nonfragmented Index	3-11
Creating a Fragmented Index	3-12
Creating an Index on a Filtered Column	3-13
Performing Text Search Queries	3-13
Highlighting	3-15

In This Chapter

This chapter provides a step-by-step procedure for setting up and performing text searches using the Excalibur Text Search DataBlade module. It covers the following topics:

- How to create and populate a text search table
- How to create and use word lists
- How to create and use a user-defined character set
- How to create an **etx** index
- How to perform text search queries
- How to highlight text

Read [Chapter 1, “Overview,”](#) and [Chapter 2, “Text Search Concepts,”](#) before you begin the tutorial.

The following table lists the steps you must take to set up and use the Excalibur Text Search DataBlade module.

Step	What You Must Do	Refer To
1	Create a table that contains a text search column	page 3-4
2	Populate the table with text search data	page 3-5
3	Create word lists and a user-defined character set	page 3-6
4	Create an etx index	page 3-9
5	Perform text search queries	page 3-13
6	Highlight text	page 3-15

Creating a Table Containing a Text Search Column

If the data you want to search is already contained in a BLOB, CLOB, LVARCHAR, CHAR, VARCHAR, IfxDocdesc, or IfxMRData column of a table, skip this step and proceed to [“Creating Word Lists and a User-Defined Character Set” on page 3-6](#).

Creating Dbspaces and Sbspaces for Storage

For optimal performance and to take advantage of the data management facilities of a DBMS, such as transaction rollback, store your text search table and associated indexes in dbspaces and sbspaces that you allocate specifically for this purpose.

If you are using a CLOB, BLOB, or IfxMRData data type, you must store your data in an sbspace. Synonym and stopword lists are also stored in sbspaces. And since **etx** indexes are also stored in sbspaces, they are always detached, because the table that contains the indexed column is stored in a dbspace.

The following example creates a dbspace named **dbsp1** with an initial offset of 0 and a size of 10 MB:

```
onspaces -c -d dbsp1 -p /Dbspace/dbsp1 -o 0 -s 10000
```

The following example creates two sbspaces named **sbsp1** and **sbsp2**, each with an initial offset of 0 and a size of 100 MB, and logging turned on:

```
onspaces -c -S sbsp1 -g 2 -p /SBspace/sbsp1 -o 0 -s 100000 -Df "LOGGING=ON"  
onspaces -c -S sbsp2 -g 2 -p /SBspace/sbsp2 -o 0 -s 100000 -Df "LOGGING=ON"
```

For more information on how to create dbspaces and sbspaces, and the complete syntax of the **onspaces** command, consult the [Administrator's Guide](#) for your database server.

Creating the Table

The following example shows how to create a table called **reports** using the CREATE TABLE statement:

```
CREATE TABLE reports
(
    doc_no INTEGER,
    author VARCHAR(60),
    title CHAR(255),
    abstract CLOB
);
```

The text data is stored in the CLOB column called **abstract**.

For the full syntax of the CREATE TABLE statement, see [Informix Guide to SQL: Syntax](#).

Populating the Table with Text Search Data

This example populates the CLOB column of the **reports** table with data from the operating system file **/local0/excal/dbms.txt** by using the **FileToCLOB()** smart large object function.

```
INSERT INTO reports (doc_no, author, title, abstract)
VALUES(
    1,
    'C.J. Date',
    'Introduction to Database Systems',
    FileToCLOB ('/local0/excal/dbms.txt', 'client')
);
```

You can also populate a CLOB column with data from another CLOB column in the database using the **LOCOPY()** smart large object function.

For more information on the **FileToCLOB()** and **LOCOPY()** functions, refer to [Informix Guide to SQL: Syntax](#).

Creating Word Lists and a User-Defined Character Set

To prevent stopwords from being indexed, always associate a stopwords list with an **etx** index. This section describes how to create a stopwords list based on the file of stopwords provided by the Excalibur Text Search DataBlade module.

The Excalibur Text Search DataBlade module provides three built-in character sets (ASCII, ISO, and OVERLAP_ISO) that are used to determine which characters are indexed. If the character sets provided do not index the desired characters, you can define your own character set. See [“Character Sets” on page 1-20](#) for an overview of how the Excalibur Text Search DataBlade module uses character sets.

This section describes how to create a user-defined character set that is similar to the ASCII character set but also indexes hyphens.

Creating a Stopword List

To create a stopwords list, use the **etx_CreateStopWlst()** procedure to indicate the name of the stopwords list, the location of the operating system file that currently contains the list of stopwords, and the name of the sbspace that will contain the list.

You can use your own operating system file of stopwords or copy the one provided by the Excalibur Text Search DataBlade module and edit it for your own use. The list of standard stopwords provided by the DataBlade module is called:

```
$INFORMIXDIR/extend/ETX.version/wordlist/etx_stopwords.txt
```

version refers to the current version of the DataBlade module installed on your computer.

For example, to create a stopwords list named **my_stopwordlist** from **/local0/excal/stp_word** and contained in the sbspace named **sbsp1**, execute the following statement:

```
EXECUTE PROCEDURE etx_CreateStopWlst  
('my_stopwordlist', '/local0/excal/stp_word', 'sbsp1');
```

The file must not contain only one stopword per line. The file must not contain any proprietary formatting information and must consist only of characters made available to the index by the CHAR_SET index parameter when the index was created.

To insert a new stopword into an existing stopword list, use the **etx_DropStopWlst()** procedure to drop the list, add the stopword to the operating system file, and then re-create the stopword list using the procedure **etx_CreateStopWlst()**. For more information about these routines, see [“etx_CreateStopWlst\(\)” on page 5-20](#) and [“etx_DropStopWlst\(\)” on page 5-27](#), respectively.

Creating a Synonym List

To create a custom synonym list, use the **etx_CreateSynWlst()** procedure to specify the name of the synonym list, the location of the file that contains the list of synonyms, and the name of the sbspace that will contain the list.

For example, to create a synonym list named **my_synonymlist** based on **/local0/excal/syn_file** and contained in **sbsp2**, execute the following statement:

```
EXECUTE PROCEDURE etx_CreateSynWlst
    ('my_synonymlist', '/local0/excal/syn_file', 'sbsp2');
```

The format of the file is one root word followed by one or more synonyms, separated by blank spaces. Each line of text must be followed by one blank line. The operating system file must not contain any proprietary formatting information and must consist only of characters made available to the index by the CHAR_SET index parameter when the index was created. The following example illustrates a synonym list's operating system file:

```
clay earth mud loam

clean pure spotless immaculate unspoiled
```

To use a particular synonym list for a search, set MATCH_SYNONYM to the name of the list.

If you specify the MATCH_SYNONYM tuning parameter in the **etx_contains()** operator but do not set it equal to a value, **etx_contains()** refers to a default synonym list named **etx_thesaurus**. See [“Synonym Lists” on page 1-16](#) for details about creating a default list.

To insert a new set of synonyms into an existing synonym list, you must use the **etx_DropSynWlst()** procedure to drop the list, add the set of synonyms to the operating system file, and re-create the synonym list using the procedure **etx_CreateSynWlst()**. For more information about these routines, see [“etx_CreateSynWlst\(\)” on page 5-22](#) and [“etx_DropSynWlst\(\)” on page 5-29](#), respectively.

Creating a User-Defined Character Set

To create a user-defined character set, use the **etx_CreateCharSet()** routine to specify the name of the new character set and the location of the operating system file that contains the definition of the character set.

For example, to create a user-defined character set named **my_new_charset** from the definition stored in **/local0/excal/my_new_char_set_file**, execute the following statement:

```
EXECUTE PROCEDURE etx_CreateCharSet  
('my_new_charset', '/local0/excal/my_new_char_set_file');
```

The operating system file must consist of 16 lines of 16 hexadecimal numbers. Each position corresponds to a specific ASCII character. If you want the character in the position to be indexed, enter the character that it should be indexed as. If you do not want the character to be indexed, enter 00.

The sample file `/local0/excal/my_new_char_set_file`, shown below, enables all alphanumeric characters to be indexed, as well as the hyphen (hexadecimal value `0x2D`). In addition, it maps lowercase letters to uppercase.

```
# Character set that indexes hyphens and
# alphanumeric characters. All lower case letters
# are mapped to upper case.
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 2D 00 00
30 31 32 33 34 35 36 37 38 39 00 00 00 00 00 00
00 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 00 00 00 00 00
00 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

To use this user-defined character set while creating an **etx** index, set the **CHAR_SET** index parameter to `my_new_charset`. If you do not specify the **CHAR_SET** index parameter in the **CREATE INDEX** statement, the ASCII character set is used by default.

For the complete syntax and additional examples illustrating the use of **etx_CreateCharSet()**, see [“etx_CreateCharSet\(\)” on page 5-16](#).

Creating an etx Index

To create an **etx** index, you specify the **etx** access method in the **USING** clause of the **CREATE INDEX** statement. You specify the operator class right after specifying the indexed column. Refer to [“Operator Classes” on page 1-8](#) for a list of the supported data types and their corresponding operator classes.

You must create an **etx** index for each text column you plan to search. You cannot alter the characteristics of an **etx** index once you create it. Instead, you must drop the index and re-create it with the desired characteristics.

Determining the etx Index Parameters

Index parameters are used to customize **etx** indexes based on the type of searches you plan to perform. Although they are not required, index parameters can increase performance in cases where you know the types of searches that will be executed, because the **etx** index can be built to suit a particular type of search.

For example, the `WORD_SUPPORT` index parameter specifies whether you plan on using exact or pattern-matching searches. The `PHRASE_SUPPORT` parameter indicates what level of phrase searching you want, from full to no support. Use the `CHAR_SET` parameter to specify what character set your documents use. The `STOPWORD_LIST` parameter lets you specify a custom stopword list, while the `INCLUDE_STOPWORDS` parameter indicates that you want the stopwords specified by the `STOPWORD_LIST` parameter to be included in the index.

Refer to [Chapter 6, “etx Index Parameters,”](#) for detailed information on these parameters.

Creating an etx Index

This section describes how to create fragmented and nonfragmented indexes. You should have already created an sbspace in which to store your index. If you have not, consult the [Administrator's Guide](#) for your database server for instruction on how to do this.

Creating a Nonfragmented Index

Suppose your search text is contained in a column of type CLOB, named **abstract**, and that the column is located in a table named **reports**. To create an **etx** index named **reports_idx1** for this table, use the following syntax:

```
CREATE INDEX reports_idx1 ON reports (abstract etx_clob_ops)
  USING etx
  IN sbsp1;
```

The preceding example creates an **etx** index that by default supports exact word searches but does not support phrase searches. The index is stored in the subspace **sbsp1** and indexes, by default, only ASCII characters. Since no stopword list is specified, all words in the document are indexed. The operator class **etx_clob_ops** is specified since the abstract column is of type CLOB.

The following example creates an **etx** index on the **title** column that is of type CHAR:

```
CREATE INDEX reports_idx2 ON reports (title etx_char_ops)
  USING etx (STOPWORD_LIST = 'my_stopwordlist',
  CHAR_SET = 'ISO') IN sbsp1;
```

In this case, the operator class is **etx_char_ops** instead of the previously used **etx_clob_ops** for columns of type CLOB. The index does not include the stopwords found in the list **my_stopwordlist**. By default, the index supports exact matches, but not phrase searches. The index uses the built-in ISO character set, specified by the CHAR_SET parameter. The index is stored in the subspace **sbsp1**.

The following statement creates an **etx** index that supports a medium level of exact and approximate phrase searches, supports exact word searches, and indexes ASCII characters:

```
CREATE INDEX reports_idx3 ON reports (abstract etx_clob_ops)
  USING etx (WORD_SUPPORT = 'EXACT',
  STOPWORD_LIST='my_stopwordlist', PHRASE_SUPPORT = 'MEDIUM')
  IN sbsp1;
```

The index does not include the stopwords found in the list **my_stopwordlist**. The index is stored in the subspace **sbsp1**.

The following statement creates an **etx** index that supports the most accurate level of exact and approximate phrase searching, supports pattern word searches, and indexes characters included in **my_new_charset**:

```
CREATE INDEX reports_idx4 ON reports (abstract etx_clob_ops)
  USING etx (WORD_SUPPORT = 'PATTERN',
  STOPWORD_LIST='my_stopwordlist', PHRASE_SUPPORT = 'MAXIMUM',
  CHAR_SET = 'my_new_charset' )
  IN sbsp1;
```

The index does not include the stopwords found in the list **my_stopwordlist**. The index is stored in the subspace **sbsp1**.

Creating a Fragmented Index

You can use the **FRAGMENT BY** clause of **CREATE INDEX** to create both round-robin and expression-based fragmentation. You cannot, however, use the **etx_contains()** operator as part of your fragmentation expression.

Suppose that you want to create a fragmented index on the CLOB column **abstract** of the table **reports** with the following requirements: documents with a **doc_no** value less than 1000 are stored in the sbspace **sbsp1**, and documents with a **doc_no** value greater than or equal to 1000 are stored in the sbspace **sbsp2**.

The following **CREATE INDEX** statement creates a fragmented **etx** index that meets the preceding requirements:

```
CREATE INDEX reports_idx5 ON reports (abstract etx_clob_ops)
  USING etx (WORD_SUPPORT = 'PATTERN',
    STOPWORD_LIST = 'my_stopwordlist', INCLUDE_STOPWORDS = 'TRUE',
    PHRASE_SUPPORT = 'MAXIMUM')
  FRAGMENT BY EXPRESSION
  doc_no < 1000 IN sbsp1,
  doc_no >= 1000 IN sbsp2 ;
```

This index supports pattern and word matching and supports maximum phrase support. Although the word list **my_stopwordlist** is specified, all of the stopwords are actually indexed, due to the **INCLUDE_STOPWORDS** index parameter. However, the stopwords are relevant in a search only if the **CONSIDER_STOPWORDS** tuning parameter is specified in the **etx_contains()** operator.

Creating an Index on a Filtered Column

When you create an index with the **CREATE INDEX** statement, you can set the **FILTER** index parameter to specify that you want to filter formatting information from the documents before they are added to the **etx** index.

The following statement creates an **etx** index on the **abstract** column of the **my_table** table and specifies that the documents in the **abstract** column should be filtered before they are added to the index:

```
CREATE INDEX abstract_index ON my_table (abstract etx_clob_ops)
  USING etx (FILTER = 'STOP_ON_ERROR');
```

You can enable filtering on columns of all seven data types supported by the Excalibur Text Search DataBlade module: CHAR, VARCHAR, BLOB, CLOB, LVARCHAR, IfxDocDesc, and IfxMRData.

The FILTER index parameter can be set to three values: NONE (the default value), STOP_ON_ERROR, or CONTINUE_ON_ERROR.

For more information about filtering and the FILTER index parameter, see [“FILTER” on page 6-5](#).

Performing Text Search Queries

To perform a text search, you use the **etx_contains()** operator in the WHERE clause of a SELECT statement. For example, suppose that you store your search text in an CLOB column named **abstracts**. To execute a pattern search on this column for the phrase `multimedia document editor`, execute the following statement:

```
SELECT title FROM reports
WHERE etx_contains(abstract,
  Row('multimedia document editor',
    'SEARCH_TYPE = PHRASE_EXACT & PATTERN_ALL'));
```

The search returns the **title** column of the documents that contain the phrases `multimedia document editor`, `multimedia document editor`, and even `multimillion documentary editorials`, although the last hit has a much lower score than the first two hits. Since the statement specified an exact phrase search, the search does not find documents that contain just the word `multimedia` or the phrase `editorial of a multimedia event` because of the differing order of the words `multimedia` and `editorial`.

Use a **keyword proximity search** to find documents that contain either of the phrases `multimedia editor` or `editor of a multimedia event`. The preceding example shows that phrase searching might not be your best choice, since the order of the words always counts in phrase searches. Order does not count in keyword searches because the words are treated as separate entities. Proximity searching ensures that the keywords are close to each other. The following is an example of a keyword proximity search:

```
SELECT title FROM reports
WHERE etx_contains(abstract,
  Row('multimedia editor',
    'SEARCH_TYPE = PROX_SEARCH(5) '));
```

The search returns the **title** column of documents that contain both the keywords `multimedia` and `editor` as long as they are no more than five words apart, inclusive. This means that the search does not return a document that contains the phrase `editor of a world class magazine` known for its cutting edge articles on multimedia because the keywords `multimedia` and `editor` are separated by more than five words.

Sometimes it is necessary to search for stopwords because they are relevant parts of the clue. For example, you may want to search for the exact phrase `plug and play` where the word `and` is a stopword. The text search engine by default does not search for stopwords, so the result of the search might not be exactly what you want. The following example shows how you can force the inclusion of stopwords in a search:

```
SELECT title FROM reports
WHERE etx_contains(abstract,
  Row('plug and play',
    'SEARCH_TYPE = PHRASE_EXACT & CONSIDER_STOPWORDS'));)
```



Important: You must specify the index parameter `INCLUDE_STOPWORDS` when you create the `etx` index if you want to use the tuning parameter `CONSIDER_STOPWORDS`. The index parameter `INCLUDE_STOPWORDS` forces the stopwords specified by the `STOPWORD_LIST` index parameter to be indexed; by default, the stopwords specified by this parameter are not indexed.

For the complete syntax of the `etx_contains()` operator and additional examples that use it, see [“etx_contains\(\)” on page 5-5](#).

Highlighting

To obtain highlighting information, specify the **etx_GetHilite()** function in the SELECT list of the query, as shown in the following example:

```
SELECT etx_GetHilite(abstract, rc) FROM reports
WHERE etx_contains(abstract,
  Row('multimedia editor', 'SEARCH_TYPE = PROX_SEARCH(5)'),
  rc # etx_ReturnType);
```

The example shows a proximity search, similar to the one in [“Performing Text Search Queries” on page 3-13](#). In addition to returning the document, the example also shows how to return highlighting information via the function **etx_GetHilite()**. The data type of the return value of the function is **etx_HiliteType**, an Informix-defined row data type that consists of two fields, **vec_offset** and **viewer_doc**, that contain highlighting information.

The **vec_offset** field contains offset information about every instance of the words **multimedia** and **editor** in the returned document, as long as the two words are within five words of each other. The **viewer_doc** field contains the text document itself.

You can use the **etx_ViewHilite()** routine together with the **etx_GetHilite()** routine to actually view the highlighted instances of **multimedia** and **editor**. The following complex query returns the text of relevant documents with every instance of the words **multimedia** and **editor** surrounded by the HTML tags **** and ****:

```
SELECT etx_ViewHilite (etx_GetHilite(abstract, rc), '<b>', '</b>')
FROM reports
WHERE etx_contains (abstract, 'multimedia editor', rc # etx_ReturnType);
```

If you were to view the results of this query in a browser, the clues **multimedia** and **editor** would appear in boldface type.

For more detailed information about these highlighting routines, refer to [Chapter 5, “Routines.”](#) This chapter also includes an example of using the **etx_ViewHilite()** routine with the **etx_HiliteDoc()** routine to highlight clues in an individual document.

You can also use standard Informix ESQL/C routines, such as **ifx_lo_open()** and **ifx_lo_read()**, or DataBlade API large object routines, such as **mi_lo_open()** and **mi_lo_read()**, to manipulate etx_HiliteType objects. Refer to the [INFORMIX-ESQL/C Programmer's Manual](#) and the [DataBlade API Programmer's Manual](#) for information about these routines.

Data Types

In This Chapter	4-3
etx_ReturnType.	4-4
etx_HiliteType	4-6
IfxDocDesc	4-9
IfxMRData	4-14

In This Chapter

This chapter describes the four new data types that come with the Excalibur Text Search DataBlade module.

Data Type	Description
etx_ReturnType	A named row type used to return side-effect data, such as document score and internal highlighting information, to the etx_contains() operator. etx_ReturnType is the data type of the statement local variable (SLV) of the etx_contains() operator.
etx_HiliteType	A named row type used to access highlighting information. etx_HiliteType is the data type of the information returned by the highlighting routine etx_GetHilite() .
IfxDocDesc	<p>A named row type that allows you to store your documents in the database or on the operating system file system.</p> <p>IfxDocDesc is a data type defined in the Text Descriptor DataBlade module, one of the DataBlade modules required by the Excalibur Text Search DataBlade module. Although it is used by all text search DataBlade modules, it is described in this guide.</p>
IfxMRData	<p>A multirepresentational opaque data type that dynamically decides, for the purposes of improving I/O performance, whether to store your text documents as LVARCHAR or CLOB values.</p> <p>IfxMRData is a data type defined in the Text Descriptor DataBlade module, one of the DataBlade modules required by the Excalibur Text Search DataBlade module. Although it is used by all text search DataBlade modules, it is described in this guide.</p>

etx_ReturnType

The data type `etx_ReturnType` is a named row type defined by the Excalibur Text Search DataBlade module as follows:

```
CREATE ROW TYPE etx_ReturnType
(
    score            REAL,
    hilite_info      etx_InternalHilite
);
```

`etx_ReturnType` is the data type of the optional statement local variable (SLV) of the **`etx_contains()`** operator. Use the SLV in a `SELECT` statement to obtain scoring information about the returned document or to pass internal highlighting information to the **`etx_GetHilite()`** routine. The following example shows a typical use of SLVs:

```
SELECT rc1.score, title FROM reports
WHERE etx_contains (abstract,
    Row('video'), rc1 # etx_ReturnType)
AND doc_no > 1005
ORDER BY 1;
```

In the example, **`rc1`** is the SLV, and its **`score`** field is used by the `SELECT` statement to order the returned rows by their document score.

The `etx_ReturnType` row type consists of two fields: **`score`** and **`hilite_info`**. The following sections describe the fields and how to use the information contained in them.

The score Field

The **`score`** field contains a numeric value that indicates the relevance of a returned document to the search criteria, compared to that of other indexed records. The higher the document score value, the more closely the document matches the criteria.

The document score is a value between 0 and 100:

- 0 indicates no match.
- 100 is always the score for exact search.
- 100 is the best possible score for a pattern search.

You can use **score** to order the returned rows according to how closely they match the search criteria. You can also use the **score** value in SELECT statements to limit a search by specifying the minimum document score of returned documents, as shown in the following example:

```
SELECT rc.score, id, description FROM videos
WHERE etx_contains ( description,
    Row('multimedia document editor' ,
        'PATTERN_TRANS & PATTERN_SUBS'),
    rc # etx_ReturnType) AND
    rc.score > 85 ;
```

The preceding query specifies that only those documents that have a document score of 85 or better should be returned.

The **hilite_info** Field

The **hilite_info** field is used internally by the Excalibur Text Search DataBlade module. The information contained in this field is only relevant to the **etx_GetHilite()** function. You should not attempt to access the information contained in this field.

See “[etx_GetHilite\(\)](#)” on page 5-34 for more information about this routine.

etx_HiliteType

The data type `etx_HiliteType` is a named row type defined by the Excalibur Text Search DataBlade module as follows:

```
CREATE ROW TYPE etx_HiliteType
(
    vec_offset CLOB,
    viewer_doc BLOB
);
```

`etx_HiliteType` is the data type of the information returned by the **`etx_GetHilite()`** and **`etx_HiliteDoc()`** routines. These routines are used to highlight the location of a clue in a document. The following example shows how to use the **`etx_GetHilite()`** function in a `SELECT` statement:

```
SELECT etx_GetHilite (description, rc) FROM videos
WHERE etx_contains(description,
    'multimedia', rc # etx_ReturnType);
```

In the example, `rc` is the statement local variable (SLV) passed to the function **`etx_GetHilite()`**.

Highlighting information is broken up into two parts: offset information about the location of the clue in a document and the document itself. The offset information is contained in the **`vec_offset`** field of the `etx_HiliteType` data type and the document is contained in the **`viewer_doc`** field.

The following sections describe in more detail the **`vec_offset`** and **`viewer_doc`** fields of `etx_HiliteType`, as well as how to use the information contained in them.

The `vec_offset` Field

The **`vec_offset`** field, of data type `CLOB`, contains a list of ordered pairs of integers that pinpoint the location of every instance of the clue in the search text.

Each integer is separated from other integers by a blank space. The first integer of the ordered pair is the beginning offset of the clue, and the second integer is the length of the highlighted string. The first character in a document has an offset of 0.

For example, the following is sample output from the **vec_offset** field if you search for the word *be* in the search text “to be or not to be”:

```
3 2 16 2
```

The first instance of the word *be* starts at offset 3, assuming the first character is at offset 0, and the length of the string to be highlighted is 2. The second instance of the word *be* starts at offset 16 and the length of the string to be highlighted is again 2.

The Excalibur text search engine sometimes expands clues to search for similar words. For example, if you search for the word *house* and request highlighting information by using the **etx_GetHilite()** or **etx_HiliteDoc()** routines, the word *housing* may also be highlighted. This explains why the length of the highlighted string specified via the **vec_offset** field may sometimes differ from the length of the original clue.

Refer to the following section for more information on the contents of the **viewer_doc** field.

For more information on the **etx_GetHilite()** and **etx_HiliteDoc()** routines, see [Chapter 5, “Routines.”](#)

You can also use standard Informix ESQL/C routines, such as **ifx_lo_open()** and **ifx_lo_read()**, or DataBlade API large object routines, such as **mi_lo_open()** and **mi_lo_read()**, to manipulate etx_HiliteType objects. Refer to the [INFORMIX-ESQL/C Programmer's Manual](#) and the [DataBlade API Programmer's Manual](#) for information about these routines.

The viewer_doc Field

The **viewer_doc** field contains the document that has just been searched. The offsets contained in the **vec_offset** field are relative to this document.

The **etx_ViewHilite()** routine manipulates the **vec_offset** and **viewer_doc** data returned by the **etx_GetHilite()** or **etx_HiliteDoc()** routines. Refer to “**etx_ViewHilite()**” on [page 5-41](#) for a description of usage and examples.

You can also use standard Informix ESQL/C routines, such as **ifx_lo_open()** and **ifx_lo_read()**, or DataBlade API large object routines, such as **mi_lo_open()** and **mi_lo_read()**, to manipulate etx_HiliteType objects. Refer to the [INFORMIX-ESQL/C Programmer's Manual](#) and the [DataBlade API Programmer's Manual](#) for information about these routines.

IfxDocDesc

IfxDocDesc is a named row type, defined by Informix in the Text Descriptor DataBlade module for use with the Excalibur Text Search DataBlade module. It is defined as follows:

```
CREATE ROW TYPE IfxDocDesc
(
    format VARCHAR(18),
    version VARCHAR(10),
    location LLD_Locator,
    params LVARCHAR
);
```

The Text Descriptor DataBlade module is one of the DataBlade modules required by the Excalibur Text Search DataBlade module.

The main advantage of the IfxDocDesc data type is that you can choose to store your text documents either on the operating system file system or in the database. The two types of storage are available for different text documents within the same column of the same table.

The following sections describe the IfxDocDesc data type in more detail and how to use it when inserting data into a column.

The Fields of the IfxDocDesc Row Data Type

The IfxDocDesc row data type has four fields:

- **format**
- **version**
- **location**
- **params**

This section describes the fields and recommends settings when appropriate.

The format Field

This field is used to store format information for the text document. The field is not used by the DataBlade module, so you can enter any text you want, including `NULL`. Informix recommends you use this field for your own record keeping.

An example of a value for this field is `MS Word` for a Microsoft Word document.

The version Field

This field is used to store version information for the text document, based on the information in its **format** field. The field is not used by the DataBlade module, so you can enter any text you want, including `NULL`. Informix recommends you use this field for your own record-keeping.

An example of a value for this field is `7.0`, which indicates the version of Microsoft Word used to create the document.

The location Field

`LLD_Locator`, the data type of the **location** field, is a named row data type, defined by Informix as follows:

```
CREATE ROW TYPE LLD_Locator
(
    lo_protocol      CHAR(18),
    lo_pointer       LLD_Lob,
    lo_location      LVARCHAR
);
```

`LLD_Locator` is defined in the Informix Large Object Locator DataBlade module, one of the DataBlade modules required by the Excalibur Text Search DataBlade module.

Use the three fields of `LLD_Locator` to specify the location of your text document, either in the database itself or a file on the operating system file system.

The following table summarizes the information about the fields of the LLD_Locator data type.

Field	Description
lo_protocol	Identifies the type of the large object.
lo_pointer	Points to a smart large object, or NULL if anything other than a smart large object.
lo_location	Points to the large object, if not a smart large object. Set to NULL if it is a smart large object.



Important: *LLD_Lob*, the data type of the **lo_pointer** field of the *LLD_Locator* data type, is an Informix-defined complex data type similar to the CLOB and BLOB types, but in addition to pointing to the location of a smart large object, it specifies whether the object contains binary or character data.

The value of the **lo_protocol** field determines the values of the other LLD_Locator fields. The following table lists the currently available protocols and summarizes the values for the other fields for each protocol.

lo_protocol	lo_pointer	lo_location	Description
IFX_BLOB	SMART LARGE OBJECT POINTER (LLD_LOB)	NULL	Smart large object that might contain binary data
IFX_CLOB	SMART LARGE OBJECT POINTER (LLD_LOB)	NULL	Smart large object that only contains character data
IFX_FILE	NULL	Full file pathname	Operating system file on the database server machine

For a complete description of the LLD_Locator and LLD_Lob data types, see the [Informix Large Object Locator DataBlade Module User's Guide](#).



The params Field

Reserved for use by the **etx** access method. When you load data into a table, specify **NULL**.

Important: *If you insert **NULL** into any of the fields of an **IfxDocDesc** column, you must explicitly cast it to its data type in the **INSERT** statement. The same is true for **UPDATE** statements that update the value to **NULL**. This is true for all fields of all row types in Informix Dynamic Server with Universal Data Option.*

Populating a Table That Contains an IfxDocDesc Data Type

The **IfxDocDesc** data type allows you to store your documents in a table as smart large objects or to reference the documents on the operating system by filename. Suppose that your search text is stored in the following collection of Microsoft Word files on a UNIX file system:

```
/local0/excal/desc1.doc
/local0/excal/desc2.doc
```

Further suppose that you want to load these files into a table called **videos1**, described by the following example:

```
CREATE TABLE videos1
(
    id            INTEGER,
    name          VARCHAR(30),
    description    IfxDocDesc
);
```

To load the first of these files, **/local0/excal/desc1.doc**, into a row of the **videos1** table, execute the following statement:

```
INSERT INTO videos1 (id, name, description)
VALUES ( 1010, 'The Unforgiven',
Row ('MS Word', '7.0',
    Row ('IFX_FILE', NULL::LLD_Lob,
        '/local0/excal/desc1.doc')::LLD_Locator,
    NULL::LVARCHAR)::IfxDocDesc
);
```

Because you specified **IFX_FILE** as the protocol, the **description** column does not actually contain the search text, but instead has a pointer (of data type **LLD_Locator**) to the operating system file specified by the **INSERT** statement.

If you want to store the second text file, **/local0/excal/desc2.doc**, in the database itself, use the **IFX_BLOB** or **IFX_CLOB** protocol, as shown in the following similar example:

```
INSERT INTO videos1 (id, name, description)
VALUES ( 1011, 'The Sting',
Row ('MS Word', '7.0',
    Row ('IFX_CLOB',
        FileToCLOB ('/local0/excal/desc2.doc', 'client'),
        NULL::LVARCHAR )::LLD_Locator,
    NULL::LVARCHAR )::IfxDocDesc
);
```

The **FileToCLOB()** routine reads the file from the operating system into the database.

For more information on the **FileToCLOB()** function and the **Row()** constructor, refer to the [Informix Guide to SQL: Syntax](#).

IfxMRData

IfxMRData is a multirepresentational opaque type, defined by Informix in the Text Descriptor DataBlade module for use with the Excalibur Text Search DataBlade module. It provides for fast and efficient storage of ASCII text data.

The main advantage of using the multirepresentational IfxMRData data type is that it dynamically determines whether to store your documents as LVARCHAR or CLOB data structures, depending on the size of the documents. Documents smaller than 2 KB are stored as LVARCHAR objects, and documents greater than 2 KB are stored as CLOB objects.



Important: *The precise cutoff point between storing documents as either LVARCHAR or CLOB data structures is 2040 bytes, and not 2048 bytes. This difference is due to the 8 bytes used to store the data structure itself.*

Inserting data into an LVARCHAR column is faster than inserting data into a CLOB column, so to speed up performance it is preferable to use LVARCHAR whenever possible. LVARCHAR columns, however, have a size limitation of 2 KB. Since it is often difficult to know ahead of time what the maximum size of a document is, it is risky to use a column type of LVARCHAR. Specifying a column type of IfxMRData solves this problem because the data type itself determines whether your document should be stored as an LVARCHAR or a CLOB data structure.

You can use the IfxMRData data type with ASCII text data only, because it uses CLOB, and not BLOB, as a possible storage type. If your documents contain binary data, you should not store them in columns of type IfxMRData, but in columns of type BLOB instead.

Updating a column can change where the data is stored. For example, suppose a document is initially stored as a CLOB type because its size is over 2 KB. If you update the value so that the size of the document is now smaller than 2 KB, the DataBlade module changes its storage to LVARCHAR. This change is transparent to users.

Populating a Table That Contains an IfxMRData Column

There are two ways to enter data into a column of type IfxMRData: either by specifying a string in the INSERT statement or by reading the data from an operating system file. This section describes both methods.

Suppose that some of your search text is stored in the following collection of files on the UNIX file system:

```
/local0/excal/desc1.txt  
/local0/excal/desc2.txt
```

Further suppose that you want to enter data into a table called **videos2**, described by the following example:

```
CREATE TABLE videos2  
(  
    id            INTEGER,  
    name          VARCHAR(30),  
    description   IfxMRData  
);
```

The next sections show how the IfxMRData behaves when you insert data into this example table.

Specifying a String

Some of the search text you want to insert into the table is not stored in operating system files since it is very small. In this case, you can enter it directly into the IfxMRData column, just as you would enter data into a standard SQL character column, as shown in the following example:

```
INSERT INTO videos2 (id, name, description)  
VALUES(  
    1010,  
    'The Unforgiven',  
    'Academy-award winning western directed by Clint Eastwood.' );
```

Since the data entered into the **description** column is smaller than 2 KB, the data is stored as an LVARCHAR object.

Using a Row Constructor

The second way to enter data into an IfxMRData column is to use an unnamed **Row()** constructor to reference the full pathname of an operating system file that contains the text data and to indicate whether the file is found on the client or the server machine:

```
INSERT INTO videos2 (id, name, description)
VALUES(
    1011,
    'The Sting',
    Row ('/local0/excal/desc1.txt', 'client')
);
```

The preceding example shows how to insert the contents of the operating system file **/local0/excal/desc1.txt** into an IfxMRData column. The Excalibur Text Search DataBlade module looks for the file on the client machine.

If the size of the file **/local0/excal/desc1.txt** exceeds 2 KB, the data will be stored as a CLOB object; otherwise, the data is stored as an LVARCHAR object.

Note that the contents of the operating system file are actually copied into either an LVARCHAR or a CLOB object; the source file **/local0/excal/desc1.txt** is not referenced again by the database server.

The following example is similar to the preceding one, but instead of looking for the operating system file **/local0/excal/desc2.txt** on the client machine, the Excalibur Text Search DataBlade module looks for the file on the server machine:

```
INSERT INTO videos2 (id, name, description)
VALUES(
    1012,
    'The Sting',
    Row ('/local0/excal/desc2.txt', 'server')
);
```

For more information on the **Row()** constructor, refer to the [Informix Guide to SQL: Syntax](#).

Routines

In This Chapter	5-3
etx_contains()	5-5
etx_CloseIndex()	5-14
etx_CreateCharSet()	5-16
etx_CreateStopWlst()	5-20
etx_CreateSynWlst()	5-22
etx_DropCharSet().	5-25
etx_DropStopWlst()	5-27
etx_DropSynWlst()	5-29
etx_Filter()	5-30
EtxFilterTraceFile()	5-32
etx_GetHilite()	5-33
etx_HiliteDoc()	5-36
etx_Release().	5-38
txt_Release().	5-39
etx_ViewHilite()	5-40
etx_WordFreq().	5-42

In This Chapter

This chapter provides you with reference information on the **etx_contains()** operator and other routines defined for the DataBlade module. You use the routines to do the following tasks:

- Execute a search
- Create and drop stopword lists
- Create and drop synonym lists
- Create and drop user-defined character sets
- Filter columns of data and enable filter tracing
- Highlight the location of a clue in a document and view the highlighted text
- Close an **etx** index

The following routines and operators are included in the Excalibur Text Search DataBlade module.

Routine or Operator	Description
etx_contains()	Executes a search
etx_CloseIndex()	Closes an etx index
etx_CreateCharSet()	Creates a user-defined character set
etx_CreateStopWlst()	Creates a list of words that will be eliminated from the search
etx_CreateSynWlst()	Creates a list of synonyms that can be used in place of a root word in a search
etx_DropCharSet()	Drops a user-defined character set

(1 of 2)

Routine or Operator	Description
etx_DropStopWlst()	Drops a stopword list
etx_DropSynWlst()	Drops a synonym list
etx_Filter()	Filters formatting information from data in a specified column so that only its contents will be returned
EtxFilterTraceFile()	Enables tracing for filter operations
etx_GetHilite()	Determines the location of a clue in a specified by an etx_contains operation
etx_HiliteDoc()	Returns the location of a clue in a specified document
etx_Release()	Returns version information for the Excalibur Text Search DataBlade module
etx_ViewHilite()	Highlights the data returned by the etx_GetHilite() or etx_HiliteDoc() routines
etx_WordFreq()	Counts the number of documents in a specified index in which a specified word occurs

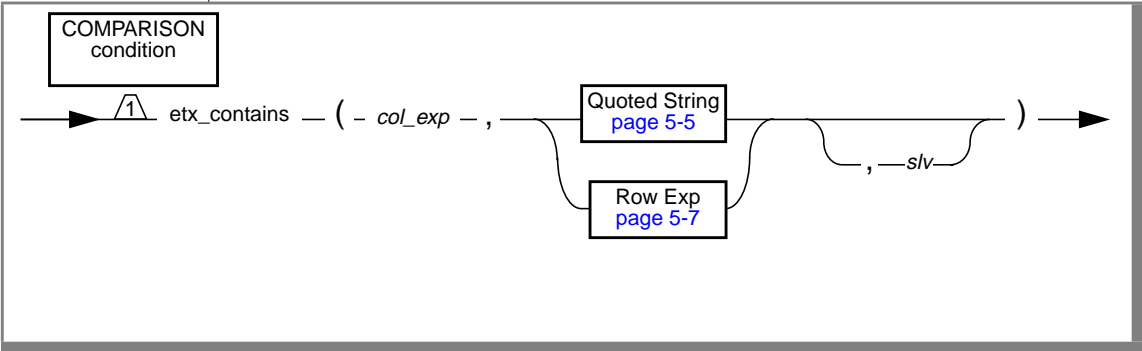
(2 of 2)

The function [txt_Release\(\)](#), included in the Text Descriptor DataBlade module, is also described in this chapter. The function returns version information for the Text Descriptor DataBlade module.

etx_contains()

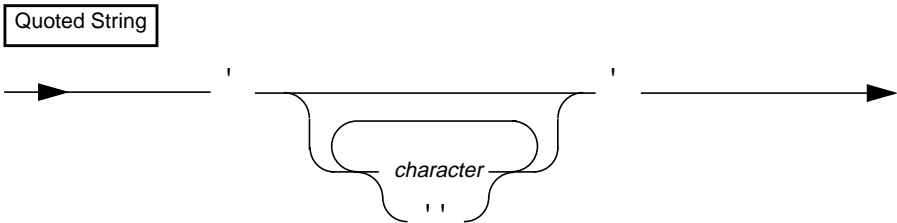
The **etx_contains()** operator executes a search that you define using a clue, tuning parameters, and an optional statement local variable (SLV).

Syntax



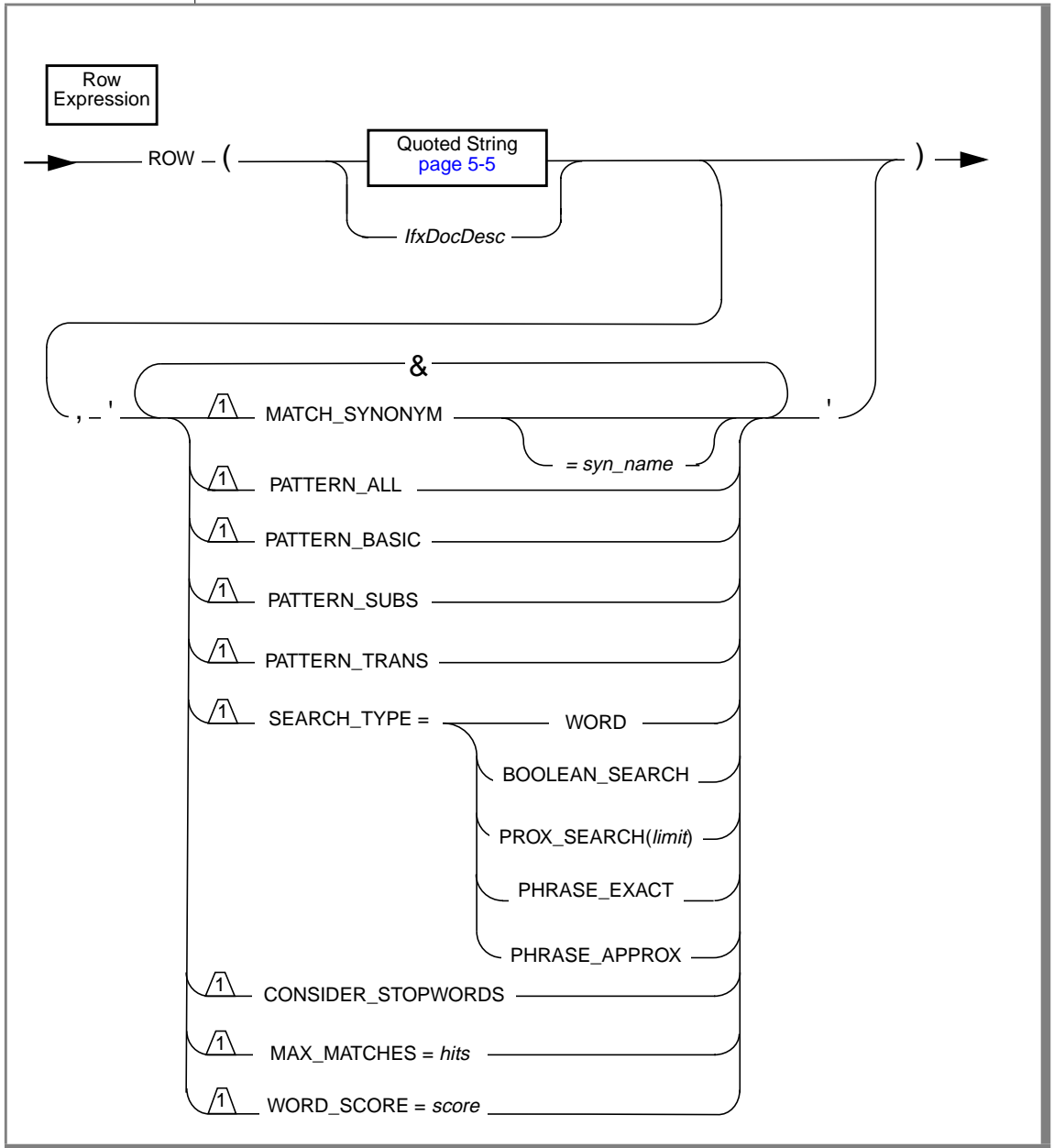
Element	Purpose	Syntax
col_exp	The name of the column you want to search.	Syntax must conform to the column expression syntax in Informix Guide to SQL: Syntax .
slv	A statement local variable (SLV) that the search engine uses to store the score and highlighting information of a particular row.	Syntax must conform to the identifier segment; see Informix Guide to SQL: Syntax .

Syntax for Quoted String



Element	Purpose	Restrictions	Syntax
<i>character</i>	A character that forms part of the quoted string	<p>When the quoted string is part of a Row() expression (see page 5-7), the following conventions apply:</p> <ul style="list-style-type: none">■ Spaces become delimiters for keywords when performing a keyword, proximity, or Boolean search.■ The characters & and correspond to the Boolean operators AND and OR, respectively, when performing a Boolean search. The two characters ! and ^ both correspond to the Boolean operator NOT. If you want to search for these characters themselves, you can escape their function as Boolean operators by preceding them with a backslash.	Characters are literal values that you enter from the keyboard.

Syntax Row Expression Usage



Element	Purpose	Restrictions	Syntax
<i>hits</i>	Maximum number of hits, per index fragment, returned by a text search.	The data type must be a positive integer.	Syntax must conform to the Literal Number syntax in Informix Guide to SQL: Syntax .
<i>limit</i>	Number of nonsearch words that can occur between two or more search words, inclusive.	The data type must be a nonzero integer.	Syntax must conform to the Literal Number syntax in Informix Guide to SQL: Syntax .
<i>score</i>	Minimum score for a word to be considered a pattern match.	The data type must be a real number.	Syntax must conform to the Literal Number syntax in Informix Guide to SQL: Syntax .
<i>syn_name</i>	Synonym list name.	The synonym list must exist.	Syntax must conform to the Literal Number syntax in Informix Guide to SQL: Syntax .
<i>IfxDocDesc</i>	The clue can be stored in an IfxDocDesc data type.	Only the data in the location field is consulted when a search is executed.	See “ <i>IfxDocDesc</i> ” on page 4-8 for more information on the IfxDocDesc row data type.

Tuning Parameters

The following table lists the tuning parameters that you can use to guide the search engine when it performs a search.

Tuning Parameter	Description	Restrictions	Default
CONSIDER_STOPWORDS	Indicates to the search engine that stopwords should be included in the search.	The etx index must have been created with the INCLUDE_STOPWORDS and STOPWORD_LIST index parameters.	Disabled

(1 of 4)

Tuning Parameter	Description	Restrictions	Default
MATCH_SYNONYM	<p>Enables synonym matching. If no value is specified, the default synonym list etx_thesaurus is consulted. If a value is specified, the specified synonym list is consulted instead of the default list.</p> <p>Follow the instructions given in “Synonym Lists” on page 1-16 to create a default or alternative list.</p>	<p>If a custom synonym list is specified, it must have already been created via the etx_CreateSynWlst() routine.</p> <p>When used together with pattern matching, such as PATTERN_ALL, only pattern matches of root words are found, not of synonyms.</p>	Disabled
MAX_MATCHES	<p>Allows you to specify the maximum number of hits <i>per index fragment</i> returned by the search engine. Informix recommends that if you want to use this tuning parameter to limit the number of rows returned from a search, you should set it to the larger of 1000 or 10% of the total number of rows in the table.</p> <p><i>Per index fragment</i> means that if, for example, the etx index is fragmented into two parts and this tuning parameter is set to 1000, a possible maximum of 2000 hits might be returned.</p>	None.	All hits.
PATTERN_ALL	<p>Enables all the pattern search options: PATTERN_BASIC, PATTERN_TRANS, and PATTERN_SUBS.</p>	The etx index must have been created with the WORD_SUPPORT = PATTERN index parameter.	Disabled

(2 of 4)

Tuning Parameter	Description	Restrictions	Default
PATTERN_BASIC	Enables the basic search option. The search returns the best pattern matches based on the value of WORD_SCORE. This may include words that are substring or superstring pattern matches of the words in the clue, as well as transpositions and substitutions, although it is not guaranteed.	The etx index must have been created with the WORD_SUPPORT = PATTERN index parameter.	Disabled
PATTERN_SUBS	Indicates to the search engine that you want words returned that match the clue except for one character.	The etx index must have been created with the WORD_SUPPORT = PATTERN index parameter.	Disabled
PATTERN_TRANS	Indicates to the search engine that you want words returned that match the clue except for a single transposition.	The etx index must have been created with the WORD_SUPPORT = PATTERN index parameter.	Disabled

(3 of 4)

Tuning Parameter	Description	Restrictions	Default
SEARCH_TYPE	<p>Allows you to specify the type of search you want to perform.</p> <p>To execute a keyword search, specify <code>WORD</code>.</p> <p>To execute a Boolean search, specify <code>BOOLEAN_SEARCH</code>.</p> <p>To execute a proximity search, specify <code>PROX_SEARCH</code>.</p> <p>To execute an exact phrase search, specify <code>PHRASE_EXACT</code>.</p> <p>To execute an approximate phrase search, specify <code>PHRASE_APPROX</code>.</p>	<p>One of the following values: <code>WORD</code>, <code>BOOLEAN_SEARCH</code>, <code>PROX_SEARCH(lim)</code>, <code>PHRASE_EXACT</code>, or <code>PHRASE_APPROX</code>.</p> <p>If set to <code>PROX_SEARCH</code>, <code>PHRASE_EXACT</code>, or <code>PHRASE_APPROX</code>, the etx index must have been created with the <code>PHRASE_SUPPORT</code> index parameter set to either <code>MEDIUM</code> or <code>MAXIMUM</code>.</p> <p>If set to <code>BOOLEAN_SEARCH</code> and used for Boolean phrase searches, the etx index must have been created with the <code>PHRASE_SUPPORT</code> index parameter set to either <code>MEDIUM</code> or <code>MAXIMUM</code>.</p>	WORD
WORD_SCORE	<p>Allows you to specify a minimum resemblance for pattern matches.</p> <p>The search engine counts as hits only words that meet the minimum standard set by <code>WORD_SCORE</code>.</p>	<p>Must be a value from 1 through 100, inclusive.</p> <p>Specifying 0 indicates that you want to set the value back to the default, 70.</p>	70

(4 of 4)

Return Type

The **etx_contains()** operator returns `BOOLEAN`.

Usage

Use **etx_contains()** to execute a search on a document stored in a column of a table. You can use the **etx_contains()** operator only in the WHERE clause of an SQL statement. For example:

```
SELECT title FROM reports
  WHERE etx_contains (abstract, Row('multimedia'))
  AND doc_no > 1005 ;
```

The **etx_contains()** operator has two required parameters: the name of the column containing text data that you want to search, and either a quoted clue or a **Row()** expression that contains the clue and optional tuning parameters. The clue can either be a quoted string or a document stored in an IfxDocDesc data type.



Warning: The column that you want to search must have an **etx** index defined on it if you want to use **etx_contains()** in the WHERE clause.

The optional third parameter of **etx_contains()** is an SLV that returns scoring and internal highlighting information. The contents of the SLV are valid only for the life of the query. The data type of the SLV is **etx_ReturnType**, an Informix-defined row data type. For more information on the **etx_ReturnType** data type, refer to [“etx_ReturnType” on page 4-4](#).

Although the **etx** access method supports fragmented indexes, you cannot use the **etx_contains()** operator to fragment an index by expression.

If you do not specify any tuning parameters, the **Row()** constructor in the **etx_contains()** operator is optional. This means that the preceding example can also be specified as:

```
SELECT title FROM reports
  WHERE etx_contains (abstract, 'multimedia')
  AND doc_no > 1005 ;
```

Typically, the clue is a quoted string of one or more words, such as the word **multimedia** in the preceding example. Sometimes, however, you might want to use an entire document as the clue. To do this, instead of specifying a quoted string as one of the parameters of the **etx_contains()** operator, specify an IfxDocDesc document.

Due to the flexibility of the LLD_Locator data type, the data type of the **location** field of the IfxDocDesc data type, you can specify as a clue either a document stored in the database or a document stored as a file on the operating system. Only the **location** field of the IfxDocDesc data type is consulted when a document is specified as a clue to the **etx_contains()** operator. The contents of the other fields, such as **format** and **version**, are ignored.

An example of a search that uses an IfxDocDesc document as a clue is shown in the next section.

For more information on the IfxDocDesc and LLD_Locator data types, see [Chapter 4, “Data Types.”](#)

Examples

The following statement searches for the specific word **multimedia** in the column **abstract** and includes other criteria in the WHERE clause:

```
SELECT title FROM reports
WHERE etx_contains(abstract, Row('multimedia'))
AND author = 'Joe Smith';
```

The following statement searches for either of the specific words **multimedia** or **video** in the column **abstract**, enables searching for letter transpositions and substitutions, and requests that synonyms from the list named **my_synonymlist** be included in the search:

```
SELECT title FROM reports
WHERE etx_contains(abstract,
Row('multimedia video',
'PATTERN_TRANS & PATTERN_SUBS & MATCH_SYNONYM = my_synonymlist'));
```

The following statement calls for the rank of a text search to be materialized as the statement local variable **rc1** and orders the returned rows by this rank:

```
SELECT rc1.score, title FROM reports
WHERE etx_contains (abstract,
Row('video'), rc1 # etx_ReturnType)
AND doc_no > 1005
ORDER BY 1;
```

The following statement executes a search on the **abstract** column, but instead of specifying a quoted string as the clue, it specifies an IfxDocDesc document stored as a file on the operating system as the clue:

```
SELECT title FROM reports
WHERE etx_contains (abstract,
  Row ( Row ('ASCII', '0',
    Row ('IFX_FILE', NULL::LLD_Lob,
      '/local0/excal/clue.txt')::LLD_Locator,
      NULL::LVARCHAR)::IfxDocDesc) );
```

The entire contents of the operating system file **/local0/excal/clue.txt** are automatically converted into the clue. Note that even though no tuning parameters are specified, the IfxDocDesc clue must still be encapsulated within a **Row()** constructor.

For additional examples of the **etx_contains()** operator, see [Chapter 2, “Text Search Concepts,”](#) and [Chapter 3, “Tutorial.”](#)

etx_CloseIndex()

The **etx_CloseIndex()** procedure closes an **etx** index.

Syntax

```
etx_CloseIndex( index_name )
```

Element	Purpose	Data Type
<i>index_name</i>	Name of the index to close.	LVARCHAR

Return Type

None.

Usage

The first time an **etx** index is used in a query (opened), resources such as shared memory are allocated in the database server, and they continue to be allocated even after the query has finished executing. Subsequent user sessions that use the **etx** index share these resources. Since new resources are not allocated for each session while the index is open, query performance is improved.

Once opened, an **etx** index does not automatically free the shared resources until the database server shuts down. To force an index to be closed, and thus free the resources, execute the procedure **etx_CloseIndex()**.

***Tip:** Informix recommends that **etx** indexes be left open, since the performance of queries that use the index is improved for subsequent user sessions. You need only use the **etx_CloseIndex()** procedure when database server resources become scarce and shared memory must be freed.*



etx_CloseIndex()

Example

The following example closes the **etx** index **reports_idx5**:

```
EXECUTE PROCEDURE etx_CloseIndex ('reports_idx5');
```


etx_CreateCharSet()

The **etx_CreateCharSet()** procedure creates a user-defined character set.

Syntax

```
etx_CreateCharSet (charset_name, file_name)
```

Element	Purpose	Data Type
<i>charset_name</i>	Name of your user-defined character set. If you enter a name longer than 18 characters, the name is silently truncated to 18 characters.	CHAR (18)
<i>file_name</i>	Absolute pathname of the operating system file from which the text search engine loads the character set. The file can be on either the server or the client machine. The client machine is searched first.	LVARCHAR

Return Type

None.

Usage

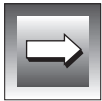
When you create an **etx** index on a column, you can specify the character set used to index the text data. The character set indicates which letters should be indexed; any characters in the text data that are not listed in the character set are converted to blanks. Use the **CHAR_SET** index parameter to specify the name of the character set. You must create a user-defined character set before you use it to create an **etx** index.

The Excalibur Text Search DataBlade module provides three built-in character sets: ASCII, ISO, and OVERLAP_ISO. Each of these built-in character sets includes only alphanumeric characters and maps lowercase letters to uppercase. This is sufficient for most text searches. For a complete description of the three built-in character sets, see [Appendix A, “Character Sets.”](#)

There are times, however, when you might want to index nonalphanumeric characters or distinguish between lowercase and uppercase letters. In these cases you must define your own character set.

To define your own character set, first create an operating system file that specifies the characters you want to index. The next section describes in detail the structure of this operating system file.

Then create the character set by executing the **etx_CreateCharSet()** routine. The routine takes two parameters: the name you give the user-defined character set and the full pathname of the operating system file that contains the characters to be indexed. The new user-defined character set is stored in the default sbspace.



Important: You cannot use the keywords *ASCII*, *ISO*, or *OVERLAP_ISO* (in any combination of uppercase and lowercase letters) as names for your user-defined character set, since these are reserved for the built-in character sets.

To use the user-defined character set, specify its name in the `CHAR_SET` index parameter of the `CREATE INDEX` statement.

Structure of the Operating System Character Set File

The operating system file consists of 16 lines of 16 hexadecimal numbers, plus optional lines that contain comments. Each position corresponds to an ASCII character. If you want the character in the position to be indexed, enter the character's hexadecimal value. If you do not want the character to be indexed, enter 00.

The ISO 8859-1 table in [Appendix A](#) lists the ISO 8859-1 character set that can be used as a reference when creating the operating system file.

Comments begin with a slash, a hyphen, or a pound sign, and they can appear anywhere in the file.

For example, if you want to create a user-defined character set that indexes hyphens (hexadecimal value 0x2D), underscores (hexadecimal value 0x5F), backslashes (hexadecimal value 0x5C), and forward slashes (hexadecimal value 0x2F), as well as the alphanumeric characters 0 through 9, a through z, and A through Z, and maps the lowercase letters a through z to uppercase, the operating system file would look like the following example:

```
# Character set that indexes hyphens and
/ alphanumeric characters. All lower case letters
\ are mapped to upper case.
- Note the different ways of specifying that a
# line is a comment.
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 2D 00 2F
30 31 32 33 34 35 36 37 38 39 00 00 00 00 00 00
00 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 00 5C 00 00 5F
00 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

This is very similar to the built-in ASCII character set, except that hyphens, underscores, forward slashes, and backslashes are also indexed instead of being converted to blanks. These four characters are indexed because the position in the matrix for the each character contains its hexadecimal representation: 0x2D, 0x5F, 0x5C, and 0x2F, respectively.

All lowercase letters are mapped to uppercase by specifying the uppercase hexadecimal value in the lowercase letter position.

For example, uppercase letter A has a hexadecimal value of 0x41. The position in the matrix of uppercase A contains the hexadecimal value 0x41, thus uppercase A is indexed as uppercase A.

However, the position in the matrix of lowercase *a* also contains the hexadecimal value 0x41 (which represents uppercase *A*) instead of the actual hexadecimal representation of lowercase *a*, 0x61. Thus, lowercase *a* is mapped to uppercase *A*, or in other words, lowercase *a* is indexed as if it were the same as uppercase *A*. The same is true for all the letters *a* through *z* and *A* through *Z*.

For more information on the ISO 8859-1 table, refer to [“ISO 8859-1” on page A-7](#).

Example

The following example creates a user-defined character set named **my_charset**:

```
EXECUTE PROCEDURE etx_CreateCharSet
    ('my_charset', '/local0/excal/my_char_set_file');
```

The search engine stores and loads the contents of **my_charset** from the file called **/local0/excal/my_char_set_file** on the operating system.

etx_CreateStopWlst()

The **etx_CreateStopWlst()** procedure creates a list of words that the text search engine ignores when it performs a search or builds an index.

Syntax

```
etx_CreateStopWlst (list_name, file_name, sbospace)
```

Element	Purpose	Data Type
<i>list_name</i>	Name of your stopword list. If you enter a name longer than 18 characters, the name is silently truncated to 18 characters.	CHAR (18)
<i>file_name</i>	Absolute pathname of the operating system file from which the text search engine loads the stopwords. The file can be on either the server or the client machine. The routine searches the client machine first.	LVARCHAR
<i>sbopace</i>	Optional parameter to specify the sbospace in which you want to store the stopword list. If you do not specify an sbospace, the database server stores the stopword list in the default sbospace.	CHAR (18)

Return Type

None.

Usage

Use the **etx_CreateStopWlst()** procedure to store a stopword list in an sbospace.

A stopword list is a list of words that you want eliminated from both an **etx** index and the text search clue. A typical word list might include the prepositions *of*, *by*, *with*, and so on. Eliminating stopwords from the search process can significantly improve the performance of your searches.

When you create a stopword list, you may use your own operating system file containing a list of stopwords or use the one provided by the DataBlade module:

```
$INFORMIXDIR/extend/ETX.version/wordlist/etx_stopwords.txt
```

version refers to the current version of the DataBlade module installed on your computer.

The format of the operating system file that contains the list of stopwords is one stopword per line. The operating system file must not contain any proprietary formatting information and must consist only of characters made available to the index by the CHAR_SET index parameter when the index was created.



Important: An *etx* index can be associated with at most one stopword list, which must specified by the *STOPWORD_LIST* index parameter when you create the index. If you want to change the stopword list associated with an *etx* index, you must first drop the index and then re-create it, specifying the name of the new stopword list.

Example

The following example creates a stopword list named **my_stopword**:

```
EXECUTE PROCEDURE etx_CreateStopWlst  
('my_stopword', '/local0/excal/stp_word', 'sbsp1');
```

The search engine stores and loads the contents of **my_stopword** from the operating system file **/local0/excal/stp_word**. The **etx_CreateStopWlst()** procedure stores the stopword list in an sbspace named **sbsp1**.

etx_CreateSynWlst()

The **etx_CreateSynWlst()** procedure creates a synonym list that the search engine uses to identify synonyms during a text search.

Syntax

```
etx_CreateSynWlst (list_name, file_name, sbospace)
```

Element	Purpose	Data Type
<i>list_name</i>	Name of your synonym list. If you enter a name longer than 18 characters, the name is silently truncated to 18 characters.	CHAR (18)
<i>file_name</i>	Absolute pathname of the operating system file from which the text search engine loads the synonym list. The file can be on either the server or the client machine. The routine searches the client machine first.	LVARCHAR
<i>sbopace</i>	Optional parameter to specify the sbospace in which you want to store the synonym list. If you do not specify an sbospace, the database server stores the synonym list in the default sbospace.	CHAR (18)

Return Type

None.

Usage

Use the **etx_CreateSynWlst()** procedure to store a synonym list in an sbospace.

A synonym list is a list of words that you want the search engine to treat as equal. For example, suppose you specify *java* as a synonym for the root word *coffee* in your synonym list. The search engine will record a hit when it encounters the word *java* even though you specified the word *coffee* in your clue.

All words in a clue that are candidates for replacement by their synonyms must be listed as root words. For example, *java* is a synonym for the root word *coffee*. A synonym-matching search for the word *coffee* records a hit if the text search engine finds the word *java*. The reverse situation, in which a search for *java* records a hit if the search engine finds *coffee*, is only true if *java* is also listed as a root word, with *coffee* as one of its synonyms.

The format of the operating system file that contains the synonyms is one root word per line, followed by one or more synonyms, separated by blanks. The root word and its synonyms must all be on one line. Each line that contains text should be followed by a blank line, as shown by the following example:

```
ABANDON RELINQUISH RESIGN QUIT SURRENDER

ABILITY APTITUDE SKILL CAPABILITY TALENT

SLANT SLOPE INCLINATION TILT LEANING
```

The operating system file must not contain any proprietary formatting information and must consist only of characters made available to the index by the CHAR_SET index parameter when the index was created.



Important: *Be sure to include an extra blank line between the lines of text in the operating system synonym file. If you omit the blank lines, the DataBlade module does not return an error, but it never finds any synonyms during a synonym-matching search. Also be sure that all synonyms are spelled correctly.*

The Default Synonym List

If you specify the MATCH_SYNONYM tuning parameter in the **etx_contains()** operator but do not set it equal to a value, **etx_contains()** refers to a default synonym list named **etx_thesaurus**. You can create your own **etx_thesaurus** synonym list from your own list of synonyms, or you can create one based on a list of standard English-language synonyms provided with your DataBlade module in the following location:

```
$INFORMIXDIR/extend/ETX.version/wordlist/etx_thesaurus.txt
```

version refers to the current version of the DataBlade module installed on your computer.

If a query refers to the **etx_thesaurus** synonym list, and the list has not yet been created, the Excalibur Text Search DataBlade module returns an error.

After you create your list, use the **etx_CreateSynWlst()** procedure to store the **etx_thesaurus** list in an sbpace.

Example

The following statement creates a synonym list named **my_synonym**:

```
EXECUTE PROCEDURE etx_CreateSynWlst  
  ( 'my_synonym', '/local0/excal/syn_file', 'sbsp2');
```

The search engine stores and loads the contents of **my_synonym** from the operating system file **/local0/excal/syn_file**. The **etx_CreateSynWlst()** procedure stores the synonym list in an sbpace named **sbsp2**.

etx_DropCharSet()

The **etx_DropCharSet()** procedure drops a user-defined character set.

Syntax

```
etx_DropCharSet (charset_name)
```

Element	Purpose	Data Type
charset_name	Name of the user-defined character set you want to drop	CHAR (18)

Return Type

None.

Usage

Use the **etx_DropCharSet()** procedure to drop an existing user-defined character set. The database server drops the user-defined character set as long as no index is currently using it.



***Important:** Use the routine **etx_DropCharSet()** to drop only user-defined character sets that are currently not being used by any **etx** indexes. If you want to change the character set being used by an index, you must first drop the index and then re-create it, specifying the name of the new character set via the **CHAR_SET** index parameter.*

To determine if a user-defined character set is currently being used by an **etx** index, query the system catalog tables, as shown by the following sample query:

```
SELECT idxname, amparam
FROM sysindices
WHERE amparam like '%charset_name%';
```

charset_name refers to the name of the user-defined character set you want to find information about. The search is case sensitive, so enter the name of the user-defined character set exactly as it was created.

If the query returns no rows, no index is currently using the specified user-defined character set. If the query returns one or more rows, the index or indexes returned in the **idxname** column are currently using the specified user-defined character set. The **amparam** column of the **sysindices** system table stores the index parameters used to create the **etx** index.

For example, the following query returns a row for each **etx** index that is currently using the user-defined character set **my_charset**:

```
SELECT idxname, amparam
FROM sysindices
WHERE amparam LIKE '%my_charset%';
```

Example

The following example drops the user-defined character set **my_charset**:

```
EXECUTE PROCEDURE etx_DropCharSet ('my_charset');
```

etx_DropStopWlst()

The **etx_DropStopWlst()** procedure drops a stopword list.

Syntax

```
etx_DropStopWlst (list_name)
```

Element	Purpose	Data Type
<i>list_name</i>	Name of the stopword list you want to drop	CHAR (18)

Return Type

None.

Usage

Use the **etx_DropStopWlst()** procedure to drop an existing stopword list. The database server drops the stopword list as long as no index is currently using it.



Important: Use the routine **etx_DropStopWlst()** to drop only stopword lists that are currently not being used by any **etx** indexes. If you want to change the stopword list being used by an index, you must first drop the index and then re-create it, specifying the name of the new stopword list.

To determine if a stopword list is currently being used by an **etx** index, query the system catalog tables, as shown by the following sample query:

```
SELECT idxname, amparam
FROM sysindices
WHERE amparam like '%stopwordlist_name%';
```

stopwordlist_name refers to the name of the stopword list you want to find information about. The search is case sensitive, so enter the name of the stopword list exactly as it was created.

If the query returns no rows, no index is currently using the specified stopword list. If the query returns one or more rows, the index or indexes returned in the **idxname** column are currently using the specified stopword list. The **amparam** column of the **sysindices** system table stores the index parameters used to create the **etx** index.

For example, the following query returns a row for each **etx** index that is currently using the stopword list **my_stopwordlist**:

```
SELECT idxname, amparam
FROM sysindices
WHERE amparam LIKE '%my_stopwordlist%';
```

Example

The following example drops the stopword list named **my_stopword**:

```
EXECUTE PROCEDURE etx_DropStopWlst ('my_stopword');
```

etx_DropSynWlst()

The **etx_DropSynWlst()** procedure drops a synonym list.

Syntax

```
etx_DropSynWlst (list_name)
```

Element	Purpose	Data Type
<i>list_name</i>	Name of the synonym list you want to drop	CHAR (18)

Return Type

None.

Usage

Use the **etx_DropSynWlst()** procedure to drop an existing synonym list. The database server drops the synonym list as long as no index is using it.

Example

The following example drops the synonym list named **my_synonym**:

```
EXECUTE PROCEDURE etx_DropSynWlst ('my_synonym');
```

etx_Filter()

The **etx_Filter()** routine filters proprietary formatting information from data in a specified column so that only its contents are returned.

Syntax

```
etx_Filter ( column )
```

Element	Purpose	Data Type
<i>column</i>	Name of the column you want to filter. <i>column</i> may be specified as a FileToBlob() operation with the following syntax: <code>FileToBlob('pathname', 'client')</code>	One of BLOB, CLOB, LVARCHAR, CHAR, VARCHAR, IfxMRData, or IfxDocDesc

Return Type

The **etx_Filter()** routine returns a CLOB value that contains the filtered text.

Usage

The Excalibur Text Search DataBlade module filters data in a column when you create an index on it and specify the FILTER index parameter. When you retrieve documents, however, you retrieve data that includes the formatting data. You can use the **etx_Filter()** routine to manually filter documents when you select them so that only text contents are returned.

If a filter error occurs while you are using the **etx_Filter()** routine, and you have enabled tracing, the Excalibur Text Search DataBlade module logs an error with row ID -1. This is because **etx_Filter()** was used by itself and is not associated with a row being indexed. See [“Enabling Tracing” on page 7-11](#) for information on tracing.

Example

The following example shows how to use the **etx_Filter()** routine in a SELECT statement:

```
SELECT etx_filter (abstract)
FROM my_table
WHERE id = 5;
```

The following example shows how to execute the **etx_Filter()** routine with a **FiletoBlob()** expression so that you can filter an operating system file before you add it to the database:

```
EXECUTE FUNCTION etx_Filter(FileToBLOB('/tmp/some.doc', 'client'));
```

EtxFilterTraceFile()

The **EtxFilterTraceFile()** routine enables tracing for filter operations.

Syntax

```
EtxFilterTraceFile ( filename )
```

Element	Purpose	Data Type
<i>filename</i>	The full pathname of the trace file, relative to the server computer. The informix user must be able to write to this file.	LVARCHAR

Return Type

None.

Usage

Use the **EtxFilterTraceFile()** procedure to set a trace log when you build an index on a column and specify `FILTER='CONTINUE_ON_ERROR'`.

After you have enabled tracing, you can disable tracing only by ending your user session.

See [“FILTER” on page 6-5](#) for more information about filtering and the FILTER index parameter.

Example

The following example sets up a trace file for filtering operations in a temporary directory:

```
EXECUTE PROCEDURE EtxFilterTraceFile ('/tmp/etx_errors');
```

etx_GetHilite()

The **etx_GetHilite()** function returns the location of a clue in a document.

Syntax

```
etx_GetHilite ( document, hilite_info)
```

Element	Purpose	Data Type
<i>document</i>	Name of the column in the table that contains the document for which you want highlighting information	One of BLOB, CLOB, LVARCHAR, CHAR, VARCHAR, IfxMRData, or IfxDocDesc
<i>hilite_info</i>	Name given to the statement local variable (SLV) of the etx_contains() operator that returns internal highlighting information	etx_ReturnType

Return Type

The **etx_GetHilite()** function returns **etx_HiliteType**, a named row data type defined by the Excalibur Text Search DataBlade module.

The **etx_GetHilite()** function returns a filtered document in the **viewer_doc** field of the returned **etx_HiliteType** value only if you have previously specified `FILTER="STOP_ON_ERROR"` or `FILTER="CONTINUE_ON_ERROR"` when you created the **etx** index.

If you previously created the **etx** index without filtering and you execute the **etx_GetHilite()** function on a document that contains proprietary formatting information (such as a Microsoft Word document), the **viewer_doc** field contains the unfiltered document and the **vec_offset** field contains offset information relative to the unfiltered document.

For more information on the **etx_HiliteType** row data type, refer to [“etx_HiliteType” on page 4-6](#).

Usage

The **etx_GetHilite()** function is used to return the location of a clue in a document or search text. This is referred to as *highlighting*.

You can only use the **etx_GetHilite()** function in the SELECT list of a query. It has two required parameters. The first is the name of the column that contains the document to be highlighted. The second is the name given to the SLV of the **etx_contains()** operator that executes the search.

The **etx_GetHilite()** function does not have as input the clue or search string to be highlighted. The function uses the clue specified as the second parameter to the **etx_contains()** operator as the highlight string.

The **etx_GetHilite()** function returns highlighting information via the row data type **etx_HiliteType**. This row data type consists of two fields that contain the two parts of the highlighting information: **vec_offset** and **viewer_doc**.

The **vec_offset** field contains offset information about the location of the clue in the document in the form of ordered pairs of integers. The integers in a pair are separated by a blank. The first integer in the pair describes the offset of an instance of the clue in the document, and the second integer describes the length of the highlight. The first character in a document is assumed to have an offset of 0. The **viewer_doc** field contains the text document itself.

To view highlighted text contained in the **etx_HiliteType** object, use the **etx_GetHilite()** function in conjunction with a function that can manipulate the **etx_HiliteType** data. See “[etx_ViewHilite\(\)](#)” on page 5-40 for an example.

You can also use standard Informix ESQL/C routines, such as **ifx_lo_open()** and **ifx_lo_read()**, or DataBlade API large object routines, such as **mi_lo_open()** and **mi_lo_read()**, to manipulate **etx_HiliteType** objects. Refer to the [INFORMIX-ESQL/C Programmer's Manual](#) and the [DataBlade API Programmer's Manual](#) for information about these routines.



Important: If you use the **etx_GetHilite()** function in a query that returns more than one row, the function executes once for each row. This means that each row has its own highlighting information, contained in the **etx_HiliteType** row data type returned by the **etx_GetHilite()** function. This highlighting information pertains only to the document contained in the row, not to any other document.



Important: If you use the **etx_GetHilite()** routine to highlight the result of a phrase search, all instances of each word in the clue are highlighted, not just instances of the entire phrase. You execute a phrase search by setting either of the tuning parameters `SEARCH_TYPE = PHRASE_EXACT` or `SEARCH_TYPE = PHRASE_APPROX`.

For example, if you execute a phrase search for the clue walk the dog and use the **etx_GetHilite()** routine to highlight the result, all instances of the individual words walk and dog (and the, if stopwords are significant) are highlighted in the resulting document, even if the instances are not close to each other.

Example

The following example shows a simple use of the **etx_GetHilite()** function:

```
SELECT etx_GetHilite (description, rc) FROM videos
WHERE etx_contains(description,
    'multimedia', rc # etx_ReturnType);
```

The function **etx_GetHilite()** is executed on the **description** column, the same column being searched with the **etx_contains()** operator. The SLV declared in the **etx_contains()** operator, **rc**, is passed as the second parameter to the **etx_GetHilite()** function. The string to be highlighted is the clue of the **etx_contains()** operator, `multimedia`.

etx_HiliteDoc()

The **etx_HiliteDoc()** function returns the location of a clue in a document.

Syntax

```
etx_HiliteDoc (document, index_name, clue)
```

Element	Purpose	Data Type
<i>document</i>	Specifies the name of the column in the table that contains the document for which you want highlighting information	One of BLOB, CLOB, LVARCHAR, VARCHAR, IfxMRData, or IfxDocDesc <i>You cannot specify columns of type CHAR.</i>
<i>index_name</i>	Specifies the name of the search index	LVARCHAR
<i>clue</i>	Specifies the search text	Either a quoted clue or a Row() expression that contains the clue and optional tuning parameters. The clue can either be a quoted string or a document stored in an IfxDocDesc data type.

Return Type

The **etx_HiliteDoc()** function returns **etx_HiliteType**, a named row data type defined by the Excalibur Text Search DataBlade module.

The **etx_HiliteDoc()** function returns a filtered document in the **viewer_doc** field of the returned **etx_HiliteType** value only if you have previously specified **FILTER="STOP_ON_ERROR"** or **FILTER="CONTINUE_ON_ERROR"** when you created the **etx** index.

If you previously created the **etx** index without filtering and you execute the **etx_HiliteDoc()** function on a document that contains proprietary formatting information (such as a Microsoft Word document), the **viewer_doc** field contains the unfiltered document and the **vec_offset** field contains offset information relative to the unfiltered document.

For more information on the **etx_HiliteType** row data type, refer to [“etx_HiliteType” on page 4-6](#).

Usage

You can use the **etx_HiliteDoc()** routine in a SELECT list to highlight occurrences of a clue in an individual document you have retrieved with an **etx_contains** scan.

You cannot use this routine by specifying EXECUTE FUNCTION.

Refer to [“etx_ViewHilite\(\)” on page 5-40](#) for information about a related routine that allows you to view the highlighted text.

Example

Suppose you have retrieved a few rows with the following query:

```
SELECT id FROM videos
      WHERE etx_contains(description,
      Row('multimedia','SEARCH_TYPE=PHRASE_APPROX'
      ));
```

You can use the **etx_HiliteDoc()** routine to retrieve highlighting information for one or more of the documents you retrieved with the previous query:

```
SELECT etx_HiliteDoc(description, 'desc_idx1',
      Row('multimedia', 'SEARCH_TYPE=PHRASE_APPROX'))
      FROM videos
      WHERE id = 1;
```

See [“etx_ViewHilite\(\)” on page 5-40](#) for an example of how you can view the highlighted text returned by this query.

etx_Release()

The **etx_Release()** function returns version information, such as the installed version and the release date, for the Excalibur Text Search DataBlade module.

Syntax

```
etx_Release()
```

The **etx_Release()** function does not take any parameters.

Return Type

The **etx_Release()** function returns an LVARCHAR value.

Usage

The information returned by the **etx_Release()** function includes:

- the version of the installed Excalibur Text Search DataBlade module.
- the date the installed Excalibur Text Search DataBlade module was compiled.
- the full version of the database server with which the Excalibur Text Search DataBlade module was compiled.
- the version of the GLS library with which the Excalibur Text Search DataBlade module was compiled.
- the version of the Excalibur Text Retrieval Library (TRL) with which the Excalibur Text Search DataBlade module was compiled.

Example

The following example shows how to execute the **etx_Release()** function:

```
EXECUTE FUNCTION etx_Release();
```

txt_Release()

The **txt_Release()** function returns version information, such as the installed version and the release date, for the Text Descriptor DataBlade module.

Syntax

```
txt_Release()
```

The **txt_Release()** function does not take any parameters.

Return Type

The **txt_Release()** function returns an LVARCHAR value.

Usage

The information returned by the **txt_Release()** function includes:

- the version of the installed Text Descriptor DataBlade module.
- the date the installed Text Descriptor DataBlade module was compiled.
- the full version of the database server with which the Text Descriptor DataBlade module was compiled.
- the version of the GLS library with which the Text Descriptor DataBlade module was compiled.

Example

The following example shows how to execute the **txt_Release()** function:

```
EXECUTE FUNCTION txt_Release();
```

etx_ViewHilite()

The **etx_ViewHilite()** routine highlights the data returned by the **etx_GetHilite()** or **etx_HiliteDoc()** routines.

Syntax

```
etx_ViewHilite (hilite_type, hilite_prefix, hilite_suffix)
```

Element	Purpose	Data Type
<i>hilite_type</i>	The etx_HiliteType data returned by the etx_GetHilite() or etx_HiliteDoc() routines	Etx_HiliteType
<i>hilite_prefix</i>	The text you want to appear before the highlight string	LVARCHAR
<i>hilite_suffix</i>	The text you want to appear after the highlight string	LVARCHAR

Returns

The **etx_ViewHilite()** routine returns an **LVARCHAR** value that contains the text of selected documents, with every instance of the search text surrounded by specified highlight tags.

Usage

The **etx_ViewHilite()** routine manipulates the **vec_offset** and **viewer_doc** data returned by the **etx_GetHilite()** or **etx_HiliteDoc()** routines.

Example

The following SQL statement uses the **etx_GetHilite()** and **etx_ViewHilite()** routines to retrieve the text of relevant documents with every instance of the words **multimedia** and **editor** surrounded by the HTML tags **** and ****:

```
SELECT etx_ViewHilite (etx_GetHilite(abstract, rc), '<b>', '</b>')
FROM reports
WHERE etx_contains (abstract, 'multimedia editor', rc # etx_ReturnType);
```

The following example uses **etx_HiliteDoc()** and **etx_ViewHilite()** routines to retrieve the text of the document with ID 1 with every instance of the word **multimedia** surrounded by the HTML tags **** and ****:

```
SELECT etx_ViewHilite(etx_HiliteDoc(description,
'desc_idx1',
                                Row('multimedia',
'SEARCH_TYPE=PHRASE_APPROX'))),
                                '<b>', '</b>')
FROM videos
WHERE id = 1;
```

etx_WordFreq()

The **etx_WordFreq()** routine counts the number of documents in a specified index in which a specified word occurs.

Syntax

```
etx_WordFreq ( index_name, clue )
```

Element	Purpose	Data Type
<i>index_name</i>	Specifies the name of the search index	LVARCHAR
<i>clue</i>	Specifies the search text (this must be a word)	LVARCHAR

Returns

The **etx_WordFreq()** utility function returns an INTEGER value that is the number of documents in the specified index in which the specified word appears.

Usage

You can use the **etx_WordFreq()** utility to get familiar with the data in your **etx** index. It might be helpful, for instance, to know that a generic word like *why* appears in every document. You can then add these sorts of words to your stopword list.

Example

The following example counts the number of documents in the **etx** index named **desc_idx1** in which the word `multimedia` appears:

```
EXECUTE FUNCTION etx_WordFreq('desc_idx1', 'multimedia');
```

etx Index Parameters

In This Chapter	6-3
Overview of the etx Index Parameters	6-3
CHAR_SET	6-4
FILTER	6-5
INCLUDE_STOPWORDS	6-7
PHRASE_SUPPORT	6-8
WORD_SUPPORT	6-9
STOPWORD_LIST	6-9

In This Chapter

This chapter describes the index parameters that can be used to customize an **etx** index. You specify index parameters in the USING clause of the CREATE INDEX statement. Use these parameters if you have an idea of the types of searches you plan to perform and want to customize your **etx** index to reflect these types of searches.

Overview of the etx Index Parameters

The following table summarizes the index parameters you can use to customize the **etx** index. Each parameter is discussed in greater detail later in the chapter.

Index Parameter	Brief Description	Valid Values	Default
CHAR_SET	Specifies the character set translation table that the index uses	ASCII, ISO, or OVERLAP_ISO, or the name of an existing user-defined character set	ASCII
FILTER	Specifies whether documents should be filtered before they are indexed and, if so, how the filter should handle errors	NONE, STOP_ON_ERROR, or CONTINUE_ON_ERROR	NONE
INCLUDE_STOPWORDS	Specifies that the stopwords in the list specified by the STOPWORD_LIST index parameter should be indexed	TRUE or FALSE Must be used together with the STOPWORD_LIST index parameter	FALSE

(1 of 2)

Index Parameter	Brief Description	Valid Values	Default
PHRASE_SUPPORT	Specifies the accuracy level of phrase searches	NONE, MEDIUM, or MAXIMUM	NONE
STOPWORD_LIST	Specifies the list of stopwords that will not be indexed	Value must be an existing stopwords list	No default
WORD_SUPPORT	Specifies the type of keyword searches the index will support	EXACT or PATTERN	EXACT

(2 of 2)

Index parameters are specified in the USING clause of the CREATE INDEX statement when you create an **etx** index. The parameters must always take the form *parameter*='value', such as CHAR_SET='ISO', as shown in the following example:

```
CREATE INDEX reports_idx2 ON reports (title etx_char_ops)
  USING etx (STOPWORD_LIST = 'my_stopwordlist',
  CHAR_SET = 'ISO') IN sbsp1;
```



Important: You cannot change the characteristics of an **etx** index once you have created it. The only way to change a customized **etx** index is to drop and re-create it.

For more examples of creating **etx** indexes with these index parameters, refer to [Chapter 3, “Tutorial.”](#)

CHAR_SET

The CHAR_SET parameter allows you to specify which characters in your text data are indexed. Characters that are not indexed are treated as white space.

You can set CHAR_SET to the value for one of the built-in character sets (ASCII, ISO, or OVERLAP_ISO) or to the name of an existing user-defined character set.

The ASCII setting means that only the alphanumeric characters 0 through 9, A through Z, and a through z are indexed. If the CHAR_SET index parameter is not specified when you create an **etx** index, the ASCII table is used by default.

When using the ASCII setting, words that contain international characters may actually be indexed as multiple words. For example, the word *cañon*, if indexed using the ASCII character set, is indexed as two words, *ca* and *on*, because the nonindexed character *ñ* is treated as white space.

The ISO setting means that both alphanumeric characters and ISO Latin-1 alphabetic characters are indexed. This 68-character set should be used to index data that might contain international characters from the ISO Latin-1 set.

The OVERLAP_ISO setting means that the same character set as the ISO setting is indexed, but similar-looking characters are grouped together so that a word matches as long as corresponding letters are from the same group.

Refer to [Appendix A, “Character Sets,”](#) for a full description of the characters that make up the ASCII, ISO, and OVERLAP_ISO character sets.

If the built-in character sets are inadequate for your text documents, you can define your own character set that defines the characters you want to index. You must create the user-defined character set prior to specifying it as an option to the CHAR_SET index parameter. Use the **etx_CreateCharSet()** routine to create your own character set.

Refer to [“etx_CreateCharSet\(\)” on page 5-16](#) for more information on using the **etx_CreateCharSet()** routine to create your own character set.

FILTER

The FILTER index parameter allows you to specify that data in a column should be filtered before it is indexed.

Important: *Filtering does not modify documents that are stored in the table column; documents remain stored with proprietary formatting text.*

Configure your database server for filtering, logging, and tracing before you create a filtered index. Read [“Configuring Your Database Server for Filtering” on page 7-9](#) for complete details.



After you have configured your database server, you can use the **FILTER** index parameter in a **CREATE INDEX** statement to filter proprietary information from documents before they are indexed.

The following statement creates an **etx** index on the **abstract** column of the **my_table** table and specifies that the documents in the **abstract** column should be filtered before they are added to the index:

```
CREATE INDEX abstract_index ON my_table (abstract etx_clob_ops)
  USING etx (FILTER = 'STOP_ON_ERROR');
```

You can enable filtering on columns of all seven data types supported by the Excalibur Text Search DataBlade module: **CHAR**, **VARCHAR**, **BLOB**, **CLOB**, **LVARCHAR**, **IfxDocDesc**, and **IfxMRData**.

You can set the **FILTER** index parameter to one of the three values described in the following table.

Value	Description
NONE	<p>Documents are not filtered before they are added to the etx index. This means that a Microsoft Word document, for example, is added to the index with all its formatting information.</p> <p>Setting the FILTER index parameter to NONE is the same as not specifying the FILTER index parameter at all.</p>
STOP_ON_ERROR	<p>If an error occurs during filtering, the client program that executed the statement gets a message identifying the row that caused the error, the same error is logged to the trace log (if tracing has been enabled), and the statement is aborted.</p> <p>See “Handling Filter Errors” on page 7-12 for information on deciding what to do if a filter error occurred.</p>
CONTINUE_ON_ERROR	<p>If an error occurs during filtering, a message identifying the row that caused the error is logged to the trace log, the unfiltered document with all its formatting information is inserted into the index, and the statement continues executing.</p> <p>See “Handling Filter Errors” on page 7-12 for information on deciding what to do if a filter error occurs.</p> <p>See “Enabling Tracing” on page 7-11 for information on how to set up tracing.</p>

Once you have created an **etx** index for a table that contains data, data that you add when you insert new rows or update rows in the table is automatically filtered.

You can also filter documents in a SELECT statement by using the **etx_Filter()** routine. Refer to [“etx_Filter\(\)” on page 5-30](#) for more information about this routine.

For a list of file formats you can filter with the Excalibur Text Search DataBlade module, see [Appendix B, “Document Formats You Can Filter.”](#)

INCLUDE_STOPWORDS

If you specify the index parameter `STOPWORD_LIST` when you create an **etx** index, the words in the list are excluded from the index and are ignored in the clue when you execute subsequent text searches. This behavior is desirable for most searches. Occasionally, however, you might want to execute searches in which stopwords are relevant. For example, you might want to search for the exact phrase “to be or not to be” and do not want the stopwords to be excluded.

To force the stopwords specified by the `STOPWORD_LIST` parameter to be indexed, specify `INCLUDE_STOPWORDS='TRUE'` when you create the **etx** index.

In this case, stopwords are indexed, similar to the default behavior, but they are considered part of the clue only if the tuning parameter `CONSIDER_STOPWORDS` is specified in the **etx_contains()** operator. If you do not specify `CONSIDER_STOPWORDS` during a search, the stopwords are ignored.

Important: The `CONSIDER_STOPWORDS` tuning parameter of the **etx_contains()** operator works only if you specified `INCLUDE_STOPWORDS` when you created the **etx** index.

The index parameter `INCLUDE_STOPWORDS` should always be used together with `STOPWORD_LIST`. The only way the DataBlade module can recognize stopwords is by consulting the stopword list associated with the index, specified via the `STOPWORD_LIST` index parameter. If you specify `INCLUDE_STOPWORDS` alone, the parameter is essentially ignored.



PHRASE_SUPPORT

Set this parameter only if you plan to perform phrase searches or proximity searches. If you are unsure what phrase and proximity searches are, refer to [Chapter 2, “Text Search Concepts.”](#)

The PHRASE_SUPPORT index parameter allows you to specify the type of phrase searches the index supports. You can set PHRASE_SUPPORT to one of the following settings:

- NONE
- MEDIUM
- MAXIMUM

The NONE setting means that the index does not support phrase or proximity searches.

Setting PHRASE_SUPPORT to lower levels of phrase and proximity support can cause false matches. However, lower levels of phrase support have the following two advantages:

- Improved performance for searches that use the index
- Reduced disk space consumption by the index

A setting of MEDIUM results in a low accuracy of matches, while MAXIMUM results in the highest accuracy. The default setting is NONE.

Given the nature of the Excalibur search engine, a false hit from a phrase search is more likely with the lower settings of PHRASE_SUPPORT and with smaller phrases. As the size of the phrase grows, the probability of a false hit decreases.

Consider using a MEDIUM setting if one or more of the following circumstances is true:

- Disk space is a factor.
- You can tolerate a small number of false hits.
- The search phrases you plan to use are lengthy.

Consider using a MAXIMUM setting if one or more of the following circumstances is true:

- Disk space is not a factor.
- Accuracy is important.
- The search phrases you plan to use are brief.

WORD_SUPPORT

The WORD_SUPPORT index parameter allows you to specify the type of word search the index supports. Specify `EXACT` if you plan to perform exact word searches only. Specify `PATTERN` if you want the index to support both exact and pattern searches. The default for this parameter is `EXACT`.

For an explanation of what exact and pattern searches are, see [Chapter 2, “Text Search Concepts.”](#)

STOPWORD_LIST

The STOPWORD_LIST parameter allows you to specify a list of words that you do not want indexed. These *stopwords* are words that are usually irrelevant in a text search, such as *and*, *by*, or *the*. All words on this list are automatically eliminated from text searches, significantly improving performance.

If no stopword list is specified when you create an **etx** index, all words in a document are indexed. In this case, words such as *the* and *an* become as relevant in a search as words such as *video* and *multimedia*, and searches return more rows, many of which might not be useful. The **etx** index needs to be significantly larger to include every word in a document, and thus searches are slower.

The stopword list must already exist before you specify it as an index parameter. To create a stopword list, use the routine **etx_CreateStopWlst()** and pass it the name of an operating system file that contains a list of stopwords.

When you create a stopword list, you can use your own list of stopwords, or you can use the one provided by the Excalibur Text Search DataBlade module:

```
$INFORMIXDIR/extend/ETX.version/wordlist/etx_stopwords.txt
```

version refers to the current version of the DataBlade module installed on your computer.

Refer to [“etx_CreateStopWlst\(\)” on page 5-20](#) for more detailed information about this routine.

DataBlade Module Administration

In This Chapter	7-3
Retrieving DataBlade Module Version Information.	7-4
Configuring Your Database Server	7-4
Creating and Configuring Sbspaces.	7-5
Creating a Default Sbspace	7-5
Configuring Sbspaces	7-6
Setting ONCONFIG Tuning Parameters	7-8
Configuring Your Database Server for Filtering	7-9
Specifying the ETX VPCLASS.	7-10
Logging Messages from the Filter Server	7-10
Enabling Tracing	7-11
Handling Filter Errors	7-12
Refining etx Indexes	7-14
Estimating the Size of an etx Index	7-14
Fragmenting Indexes.	7-16
Sharing etx Indexes	7-17
Reclaiming Unused Index Space	7-18
Choosing the Appropriate Storage Data Type.	7-18
BLOB	7-19
CLOB	7-19
LVARCHAR.	7-19
IfxDocDesc	7-20
IfxMRData	7-20
Loading Data.	7-21
Nonindexed Tables	7-21
Preindexed Tables	7-21

Handling Deadlocks	7-22
Refining Queries	7-23
Queries That Contain Multiple etx_contains() Operators	7-23
Queries That Include the NOT Keyword	7-25
Queries That Include Index Scans and Nonindex Scan Filters	7-25

In This Chapter

The purpose of this chapter is to make database administrators aware of configuration and performance issues related to the features of the Excalibur Text Search DataBlade module.

You should become familiar with the following topics before you design or redesign a database that uses **etx** indexes:

- [“Retrieving DataBlade Module Version Information” on page 7-3](#)
- [“Configuring Your Database Server” on page 7-4](#)
- [“Configuring Your Database Server for Filtering” on page 7-9](#)
- [“Refining etx Indexes” on page 7-14](#)
- [“Choosing the Appropriate Storage Data Type” on page 7-19](#)
- [“Loading Data” on page 7-21](#)
- [“Handling Deadlocks” on page 22](#)
- [“Refining Queries” on page 7-23](#)

See [“Software Dependencies” on page 5](#) for a list of required software, as well as application development tools that can be used with the Excalibur Text Search DataBlade module.

Check your release notes for the latest information about version compatibility, configuration requirements, and performance tunings.

Refer to the [Administrator’s Guide](#) and the [Performance Guide](#) for your database server for general suggestions for improving performance of your database.

Retrieving DataBlade Module Version Information

This section shows how to obtain precise version information that you might need to fully understand dependency and compatibility requirements or to identify problems to technical support staff.

Use the **etx_Release()** function to return version information for the Excalibur Text Search DataBlade module. The information returned by this function includes:

- the version of the installed DataBlade module.
- the date the installed DataBlade module was compiled.
- the full version of the database server with which the DataBlade module was compiled.
- the version of the GLS library with which the DataBlade module was compiled.
- the version of the Excalibur Text Retrieval Library (TRL) with which the DataBlade module was compiled.

Use the EXECUTE FUNCTION command to execute the **etx_Release()** function, as shown in the following example:

```
EXECUTE FUNCTION etx_Release();
```

The Text Descriptor DataBlade module has an equivalent function: **txt_Release()**. For more information on the **etx_Release()** and **txt_Release()** functions, refer to [Chapter 5, “Routines.”](#)

Configuring Your Database Server

This section describes the following database server administration topics:

- [“Creating and Configuring Sbspaces” on page 7-4](#)
- [“Setting ONCONFIG Tuning Parameters” on page 7-8](#)

Creating and Configuring Sbspaces

The Excalibur Text Search DataBlade module uses sbspaces in a variety of ways. This section covers the minimum you must know about sbspaces to correctly register and use the DataBlade module. For more detailed information about sbspaces, refer to the [Administrator's Guide](#) for your database server.

Creating a Default Sbspace

You must create a default sbspace before you register the DataBlade module into any database, or the registration fails. During registration, the Excalibur Text Search DataBlade module sets up internal directories in a default sbspace for synonym lists, stopword lists, and user-defined character sets. The Excalibur Text Search DataBlade module also stores **etx** indexes in the default sbspace unless you explicitly specify another sbspace for the index. Be sure the default sbspace is large enough to hold all of these objects.

To create the default sbspace

1. Set the **ONCONFIG** parameter **SBSPACENAME** to the name of your default sbspace.

For example, to name the default sbspace **sbsp1**:

```
SBSPACENAME      sbsp1 # Default sbspace name
```

You must update the **ONCONFIG** file before you start the database server.

2. Use the **onspaces** utility to create the sbspace.

The following example shows how to create an sbspace called **sbsp1** in the partition **/dev/sbspace**:

```
% onspaces -c -S sbsp1 -g 2 -p /dev/sbspace -o 0 -s 100000 -Df "LOGGING=ON"
```

The example sbspace has an initial offset of 0, a size of 100 MB, and logging is turned on.

You can use the **FileToBLOB()** function to test whether the default sbspace has been created and configured correctly, as shown in the following example executed in DB-Access:

```
EXECUTE FUNCTION FileToBLOB ('/tmp/some.txt', 'server');
```

The file **/tmp/some.txt** can contain any type of text, but it should be fully accessible to the user who started the database server.

If the function returns without an error, the default sbspace has been created and configured correctly.

For more information on the **FileToBLOB()** function, refer to the [Informix Guide to SQL: Syntax](#).

Configuring Sbspaces

You can use the **-Df** option of the **onspaces** utility to specify new default values for sbspace parameters. This section shows how to log sbspaces and how to customize size specifications for smart large objects stored in an sbspace.

Turning Logging On

You must use logged sbspaces when you create an **etx** index; otherwise, the **CREATE INDEX** statement fails.

The **onspaces** utility creates sbspaces with logging turned off by default. **onspaces** turns off logging by default to conserve resources. **etx** index users, however, benefit from logging. If there is a power outage while an **etx** index is being updated, for example, the index might become corrupt. If the sbspace that holds the index has logging turned off, the changes up to the point of corruption cannot be backed out. In this case, to ensure index integrity, the index must be dropped and re-created. If logging is turned on, the **etx** index can be recovered normally.

You turn logging on by specifying **-Df "LOGGING=ON"** when you create an sbspace with the **onspaces** utility:

```
% onspaces -c -S sbsp1 -g 2 -p /dev/sbospace -o 0 -s 100000 -Df "LOGGING=ON"
```

Leaving Buffering On

You must create all sbspaces that are to be used with the Excalibur Text Search DataBlade module with **BUFFERING** set to **ON**. This is the default value, so if you do not specify a value for **BUFFERING** in an **onspaces** command, your sbspace can be used with the Excalibur Text Search DataBlade module.

Any sbspace created with buffering off cannot be used by the Excalibur Text Search DataBlade module.

Customizing Size Specifications for Large Objects

Informix recommends that you customize the following two specifications for sbspaces that will contain **etx** indexes:

- AVG_LO_SIZE
- MIN_EXT_SIZE

An **etx** index fragment is composed of 6 to 13 smart large objects. The number of smart large objects depends on the combination of WORD_SUPPORT and PHRASE_SUPPORT index parameters used in the CREATE INDEX statement. The following table shows how many smart large objects make up an **etx** index fragment for each combination of index parameters.

Index Parameter Combination	Number of Smart Large Objects
WORD_SUPPORT=EXACT, PHRASE_SUPPORT=NONE	6
WORD_SUPPORT=EXACT, PHRASE_SUPPORT=MEDIUM	9
WORD_SUPPORT=EXACT, PHRASE_SUPPORT=MAXIMUM	10
WORD_SUPPORT=PATTERN, PHRASE_SUPPORT=NONE	9
WORD_SUPPORT=PATTERN, PHRASE_SUPPORT=MEDIUM	12
WORD_SUPPORT=PATTERN, PHRASE_SUPPORT=MAXIMUM	13



Important: The default value for the WORD_SUPPORT and PHRASE_SUPPORT index parameters are EXACT and NONE, respectively.

As an **etx** index fragment grows, some of the smart large objects that contain the index also grow. The smart large object extensions that are created as a result can be costly in both access time and consumption of metadata space. If you specify a relatively large MIN_EXT_SIZE value, you minimize the number of these extensions. The default values for MIN_EXT_SIZE and AVG_LO_SIZE are 4 and 64, respectively. Informix recommends that you increase these values substantially, to 1000 and 2000, for example.

The following example shows how to specify new values for AVG_LO_SIZE and MIN_EXT_SIZE with the **onspaces** utility:

```
% onspaces -c -S sbbsp1 -g 2 -p /dev/sbospace -o 0 \  
-s 1000000 -Df "AVG_LO_SIZE=1000,MIN_EXT_SIZE=2000"
```

The following output from an **oncheck -cS sbbsp1** command shows the effect of setting these options:

Large Objects									
ID	Ref		Size	Allocced		Creat		Last	
Sbs#	Chk#	Seq#	Cnt	(Bytes)	Pages	Extns	Flags	Modified	
4	4	2	1	25	1	2	L-N-H	Tue Mar 9 15:29:18 1999	
4	4	3	1	33280	25	4	L-N-H	Tue Mar 9 15:29:18 1999	
4	4	4	1	247564	164	4	L-N-H	Tue Mar 9 15:29:18 1999	
4	4	5	1	512	1	2	L-N-H	Tue Mar 9 15:29:18 1999	
4	4	6	1	52340	35	4	L-N-H	Tue Mar 9 15:29:18 1999	
4	4	7	1	131072	65	3	L-N-H	Tue Mar 9 15:29:18 1999	

The table whose **etx** index is stored in the sbospace **sbbsp1** contains 100,000 documents, averaging 2 KB in size. The number of extensions (Extns) is very low.

For more information on the **onspaces** and **oncheck** utilities and the ONCONFIG parameter SBSPACENAME, refer to the [Administrator's Guide](#) for your database server.

Setting ONCONFIG Tuning Parameters

You can optimize the performance of text searches that use **etx** indexes when you configure the following ONCONFIG tuning parameters for Informix Dynamic Server with Universal Data Option:

- **BUFFERS**

The BUFFERS parameter specifies the maximum number of shared memory buffers that Universal Data Option user threads have available for disk I/O on behalf of client applications. The default number is 200. Increasing the number of buffers has been found to improve the performance of text searches.

- **RA_PAGES**

The RA_PAGES parameter specifies the number of disk pages that the database server should attempt to read ahead during sequential scans of data or index records. Try setting this parameter to 64.

- **RA_THRESHOLD**

The **RA_THRESHOLD** parameter specifies the number of unprocessed pages in memory that signals the database server to perform the next read ahead. If **RA_PAGES** is set to 64, setting **RA_THRESHOLD** to 33 could improve the performance of your text searches.

- **RESIDENT**

The **RESIDENT** parameter specifies whether the resident portion of shared memory remains resident in operating system physical memory. If your operating system supports forced residency, you can improve the performance of searches by specifying that the resident portion of shared memory not be swapped to disk. To do this, set this parameter to 1 (on).

- **NOAGE**

The **NOAGE** parameter controls whether the UNIX operating system lowers the priority of database server processes as the processes run over a period of time. You can improve the performance of searches by setting this parameter to 1 (on, which disables priority aging).

This section provides only a minimum of information about the **ONCONFIG** tuning parameters. Before you make any changes to your **ONCONFIG** file, refer to the [Administrator's Guide](#) for your database server for more detailed information on each parameter.

Configuring Your Database Server for Filtering

This section describes steps you complete before you create an index with the **FILTER** parameter:

- [“Specifying the ETX VPCLASS” on page 7-10](#)
- [“Logging Messages from the Filter Server” on page 7-10](#) (if you want to log messages)
- [“Enabling Tracing” on page 7-11](#)

This section also includes a discussion of [“Handling Filter Errors” on page 7-12](#).

You do not need to know how to configure or use the filter server. It is described in this section for your information only.

Specifying the ETX VPCLASS

The filtering routines execute in their own virtual processor called ETX, not in the CPU virtual processor. To start an ETX virtual processor, add the following line to the **ONCONFIG** file and restart the Informix database server with the **oninit** utility:

```
VPCLASS ETX,noyield,num=XXX
```

XXX refers to the maximum number of ETX virtual processors you want to run. You might run more than one ETX virtual processor so that more than one user can use the filtering capability at a time.

Each ETX virtual processor starts up its own *filter server*. If you start up four ETX virtual processors, you might also have a maximum of four filter servers running. The filter servers run on the same computer as the database server.

For more information on virtual processors, how to start them, how many you should run, and the **oninit** utility, refer to the [Administrator's Guide](#) for your database server.

Logging Messages from the Filter Server

You can specify that the filter server log messages to a file by following these steps:

1. Log in as the user **informix**.
2. Set the environment variable **ETX_FILTER_FILE** to the full pathname of the filter log file. The **informix** user must be able to write to this file.

For example, the following UNIX C shell command sets the **ETX_FILTER_FILE** environment variable to the file **/tmp/filterserver.log**:

```
setenv ETX_FILTER_FILE "/tmp/filterserver.log"
```

3. Restart the Informix database server with the **oninit** utility.

The filter server writes to the filter log file when the database server instructs the filter server to start up, shut down, or report connection problems with the database server. If more than one filter server is running at one time, all filter servers write to the same filter log file. A filter writes a timestamp, the process ID, and a message to a log file, which appears similar to the following example:

```
Mon Mar 8 16:33:57 11613: _____
Mon Mar 8 16:33:57 11613: FilterServer process starting up.
Mon Mar 8 16:33:57 11613: Successfully connected to Informix server.
Mon Mar 8 16:33:57 11613: Connected to port 34459 Send/recv on socket 4
Mon Mar 8 17:10:55 24086: _____
Mon Mar 8 17:10:55 24086: FilterServer process starting up.
Mon Mar 8 17:10:55 24086: Successfully connected to Informix server.
Mon Mar 8 17:10:55 24086: Connected to port 35840 Send/recv on socket 4
Tue Mar 9 14:26:11 11613: GOODBYE. Lost connection to Informix server...
Tue Mar 9 14:26:11 24086: GOODBYE. Lost connection to Informix Server...
```

The final two messages occur when you shut down the database server with the **onmode -k** utility.

Enabling Tracing

This section describes how to set up the trace log.

If you specify `CONTINUE_ON_ERROR` for the `FILTER` index parameter, you must also set up a trace log to capture any errors encountered while filtering documents when you create an index, insert rows, or update rows in the table. A filtering error typically occurs when the format of a document that is about to be filtered is not in the list of supported formats. See [Appendix B, “Document Formats You Can Filter,”](#) for the full list of formats supported by the Excalibur Text Search DataBlade module.



Important: The trace log is different from the filter log file described in [“Logging Messages from the Filter Server” on page 7-10](#). The filter log file contains messages from the filter server; the trace log contains trace messages from the Excalibur Text Search DataBlade module.

To enable tracing, execute the **etx_FilterTraceFile()** procedure in DB-Access:

```
EXECUTE PROCEDURE EtxFilterTraceFile ('/tmp/etx_errors');
```

The trace messages are written to the file `/tmp/etx_errors`.

If the format of a document that is about to be filtered is not in the list of supported formats, you might get a message similar to the following message in the trace file:

```
10:29:17 Filter Message: Could not filter document
10:29:17 filter failed for row(row:257 fragid:1048870),ignoring error
```

In the example, the index parameter `FILTER='CONTINUE_ON_ERROR'` was specified when the `etx` index was created. If `STOP_ON_ERROR` had been specified, the last line would say `stopping` instead of `ignoring error`.

See [“EtxFilterTraceFile\(\)” on page 5-32](#) for detailed information about this procedure.



Tip: To enable more detailed tracing, use the `TxtSetTrace()` procedure:

```
EXECUTE PROCEDURE TxtSetTrace('EtXExcalIndex 1', '/tmp/etx_level1_msgs');
```

`EtXExcalIndex 1` specifies level-1, or filter, tracing. You can also specify `EtXExcalIndex 2` to enable level-2 tracing (messages for main indexing operations, such as inserts and scans) and `EtXExcalIndex 3` to enable level-3 tracing (messages regarding tuning parameters evaluated by the search engine).

Handling Filter Errors

This section describes how to handle errors you might encounter when you create a filtered index.

The Excalibur Text Search DataBlade module writes error messages to the trace file described in the previous section. The message includes the row ID of the row that caused the error. You can use the row ID to select the offending row from the table, fix the problem with the document, and update or re-insert the row in the table with the fixed document.



Important: You can use row IDs to find rows only in a nonfragmented table. The current implementation of row IDs in fragmented tables makes it impossible for you to map the row ID returned by the DataBlade module to an actual row in the table. Therefore, this discussion is relevant only to nonfragmented tables.

If you specify `STOP_ON_ERROR` for the `FILTER` index parameter, you cannot finish executing the statement until all filter problems have been resolved. This means that if you execute the `CREATE INDEX` statement to create an index on a table that currently contains rows, the index is created only when there are no filter problems with any existing documents. Similarly, if you execute the `INSERT` statement to insert new rows into a table that already has an **etx** index, the `INSERT` statement completes only when all the new documents have no filter problems.

If you specify `CONTINUE_ON_ERROR`, filtering errors do not stop the execution of the `CREATE INDEX`, `INSERT`, or `UPDATE` statement. If an error is encountered while filtering a document, the unfiltered document, along with any formatting information, is added to the index instead. When the statement is finished, a list of all the rows whose documents could not be filtered is written to the trace file. You can use the list of row IDs first to fix the problems with the documents and then to update the rows. As the rows in the table are updated, the corresponding entry in the index is also updated. If the original filtering problem was fixed, the new index entry contains the filtered document.

The following example shows one of many ways to correct a filtering problem after an index has been created.

Suppose you create an **etx** index on the **abstract** column of the table called **reports**. You also specify `FILTER=CONTINUE_ON_ERROR` so that the documents in the column are filtered. The **abstract** column is of data type `BLOB` and the table contains 100 rows.

Although the `CREATE INDEX` statement successfully finishes executing, the trace log indicates that row 65 had a filtering problem. This means that the document in the **abstract** column of row 65 was not filtered before it was added to the index; it was added with all its formatting information.

After you create the index, you decide to try to resolve the filtering problem in the document in row 65. You can select this row from the table, and simultaneously write the contents of the **abstract** column to a file on the operating system on the client computer, by executing the following `SELECT` statement:

```
SELECT LoToFile (abstract, '/tmp/outfile', 'client')
FROM my_table
WHERE rowid = 65;
```

You can then examine the contents of the file **/tmp/outfile** to find out what caused the filter error. Once you have fixed this error, you can update the table, as shown in the next example. For this example, the correctly formatted document is now contained in the file **/tmp/infile**:

```
UPDATE my_table  
SET abstract = FileToCLOB ('/tmp/infile' , 'client')  
WHERE rowid = 65;
```

When you update the **abstract** column in the table, you are also updating the **etx** index built on this column. The index on the **abstract** column for the preceding example reflects changes to the data in row 65.

For more information on the **LOToFile()** and **FileToBLOB()** routines, refer to the [Informix Guide to SQL: Syntax](#).

Refining etx Indexes

This section describes ways you can refine your **etx** index to improve the performance of searches. It includes the following topics:

- [“Estimating the Size of an etx Index,”](#) next
- [“Fragmenting Indexes”](#) on page 7-16
- [“Sharing etx Indexes”](#) on page 7-18
- [“Reclaiming Unused Index Space”](#) on page 7-18

Estimating the Size of an etx Index

This section discusses the factors that might make your **etx** index larger than it needs to be and consequently cause your queries to run slower than they need to run.

The size of an **etx** index can vary widely, depending on a number of factors. These factors include:

- The nature of the data to be indexed.
If your documents contain a lot of numerical data, such as the data found in financial reports, and you specify a character set that indexes numeric characters, each string of numbers is indexed as if it were a word. This can increase the number of unique words in the index.
- The index parameters that you specify.
If you specify `PHRASE_SUPPORT=MEDIUM` or `PHRASE_SUPPORT=MAXIMUM`, the index will be two to four times larger than if you specify `PHRASE_SUPPORT=NONE`.
- Stopword lists.
An **etx** index built with a stopwords list (specified by `STOPWORD_LIST='my_stopwordlist'`) is generally slightly smaller than an index that contains all the words in a document. However, if you also specify `INCLUDE_STOPWORDS='TRUE'`, the index is approximately 50% larger.

Predicting the size of an **etx** index is made more complicated by the fact that it is not directly proportional to the number of documents or words in a document. The following table shows **etx** index sizes for various combinations of index parameters.

Index Parameter				Total Index Size, in Disk Pages, for an etx Index Containing 100,000 Documents
Word Support	Phrase Support	Stopword List	Include Stopwords	
Exact	None	No	False	29 KB
Exact	None	Yes	False	28 KB
Exact	Medium	No	False	69 KB
Exact	Maximum	No	False	87 KB
Exact	Maximum	Yes	False	62 KB
Exact	Maximum	Yes	True	87 KB
Pattern	None	No	False	34 KB

(1 of 2)



Index Parameter				Total Index Size, in Disk Pages, for an etx Index Containing 100,000 Documents
Word Support	Phrase Support	Stopword List	Include Stopwords	
Pattern	None	Yes	True	34 KB
Pattern	Medium	No	False	73 KB
Pattern	Maximum	No	False	92 KB
Pattern	Maximum	Yes	False	67 KB
Pattern	Maximum	Yes	True	92 KB

(2 of 2)

The average size of the documents in this table is 2 KB.

Tip: The size of document you want to index is limited by the amount of virtual memory on your machine. For example, if you have 32 MB of virtual memory, you can only index documents that are smaller than 32 MB.

Fragmenting Indexes

Fragmenting indexes typically improves performance of text searches, particularly on multiprocessor computers. However, improperly fragmented indexes can degrade performance of certain queries. Carefully consider how your likely data and your likely queries might be affected by fragmentation.

As with all databases and indexing methods, spreading the text data and the **etx** index over multiple disk drives can improve performance. If enough disk drives are available, it is also beneficial to spread the table data into multiple dbspaces or sbspaces by using either expression-based or round-robin fragmentation.

Expression-based fragmentation is the best mechanism for achieving smaller index fragments because each index fragment functions as a completely isolated index and incurs open costs proportional to the size of just that fragment. Do not use round-robin fragmentation for the index, because it can result in worse performance than no fragmentation at all.

Text searches are performed via the **etx_contains()** operator, which is executed as an index scan. The key to lowering index scan costs is either to reduce the number of rows selected through the index scan by using very selective search criteria or to shrink the size of the index fragment being read.

As an example, consider a simple table with the following definition:

```
CREATE TABLE recipes
(
    id          INTEGER,
    recipeCLOB,
    meal_typeCHAR(1),
    ingredientsLVARCHAR
);
```

Suppose that most of the time the users are interested in particular recipes for meals at a certain time of the day: they do not want to eat pancakes for dinner or lasagna for breakfast. Query performance is improved if the index data is fragmented according to the **meal_type** column, as shown in the following example:

```
CREATE INDEX recipes_idx ON recipes (recipe etx_clob_ops)
    USING etx (PHRASE_SUPPORT = 'MAXIMUM')
    FRAGMENT BY EXPRESSION
    meal_type = 'B' in sbsp1,
    meal_type = 'L' in sbsp2,
    meal_type = 'D' in sbsp3;
```

To find dinner menus that use zucchini, the SQL statement would look like this:

```
SELECT id FROM recipes
    WHERE etx_contains ( recipe, 'zucchini')
    AND meal_type = 'D';
```

This immediately eliminates fragments from consideration and results in faster query execution.

Consider the impact of fragmentation on query performance and concurrency when users update data as well. You want to lock the index for as short a time as possible. Informix Dynamic Server with Universal Data Option exclusively write-locks a smart large object when it is being written to. Since text indexes are stored in smart large objects, each index fragment is locked exclusively while it is being updated.

Informix recommends that you fragment your index into no more than six fragments. You should experiment with tuning for your specific DBMS to determine the optimal number of index fragments for your data.

Excessive index fragmentation can degrade performance in certain situations. For example, if your index is fragmented into many fragments and a query retrieves information from every one, the overall performance of the query might be worse than if the index had not been fragmented at all.

Sharing etx Indexes

Informix recommends that **etx** indexes be left open to improve performance of subsequent queries that use the index.

The first time an **etx** index is used in a query (opened), resources such as shared memory are allocated in the database server, and these resources continue to be allocated after the query has finished executing. This causes the first query that opens the **etx** index to run slower than subsequent user sessions that use these already-opened resources.

Once opened, an **etx** index does not automatically free the shared resources until the database server shuts down. An **etx** index does free shared resources after an update, insertion, or deletion to the table, or during a rollback of a transaction that uses the index. If you must free resources, you can force an **etx** index to close by executing the **etx_CloseIndex()** procedure. See [“etx_CloseIndex\(\)” on page 5-14](#). Be aware, however, that the next user who reopens the **etx** index again incurs the initial cost of storing index information in memory.

Reclaiming Unused Index Space

You can improve performance by reclaiming unused index space.

etx indexes do not shrink. Although data that has been deleted is marked as such in the index structure, it is never removed. If you insert and then delete many rows from an indexed table, the index remains at its maximum insertion size even after you delete the rows. To reclaim unused space, you need to drop and re-create the index.

Choosing the Appropriate Storage Data Type

You can improve performance by choosing the appropriate storage data type for your data. This section describes the advantages and disadvantages of storing text documents in the following supported data types: BLOB, CLOB, LVARCHAR, IfxDocDesc, and IfxMRData.

Although you can also build an **etx** index on columns of type CHAR and VARCHAR, their small storage capacities make them unlikely choices for storing text documents and they are not discussed in this section.

BLOB

BLOB columns work well for large documents that contain binary data, such as Microsoft Word or Acrobat PDF. A column of type BLOB can contain documents of essentially unlimited size, since the data is stored in sbspaces.

There is, however, a significant amount of resource overhead when you store documents in sbspaces. If all your documents are smaller than 2 KB, you might consider storing them in a column of type LVARCHAR.

CLOB

The CLOB data type should be used for large documents that contain only standard ASCII text, such as HTML and SGML documents. A column of type CLOB can contain documents of essentially unlimited size, since the data is stored in sbspaces.

There is, however, a significant amount of resource overhead when you store documents in sbspaces. If your documents are smaller than 2 KB, you might consider storing them in a column of type LVARCHAR.

Tip: If your documents contain only standard ASCII text, but are a mixture of small (under 2 KB) and large (over 2 KB) sizes, consider using the IfxMRData data type, described later in this section and in [“IfxMRData” on page 4-13](#).



LVARCHAR

Text documents inserted into columns of type LVARCHAR are stored in dbspaces, just like data inserted into the standard SQL character data types such as CHAR and VARCHAR. This incurs less resource overhead than storing documents in sbspaces, which is where data in BLOB and CLOB columns are stored.

The disadvantage of storing documents in columns of type LVARCHAR is that the size of the documents is limited to 2 KB.

IfxDocDesc

The main advantage of using a column of type IfxDocDesc is that you can choose to store your text documents either on the operating system file system or in the database itself. A single column of type IfxDocDesc can have rows of each storage type. You specify how you want to store the documents via the **lo_protocol** field of the **location** field of IfxDocDesc by entering one of IFX_FILE, IFX_CLOB, or IFX_BLOB.

One of the disadvantages of using the IfxDocDesc data type, however, is its complexity. IfxDocDesc is a row type whose **location** field is also a row type, LLD_Locator. Inserting into a column of type IfxDocDesc requires two uses of the **Row()** constructor, which although not impossible, is more complex than inserting into the other supported data types.

A disadvantage to storing your document in a file on the operating system file system is that it is up to you to maintain the document and be sure that it is not moved or renamed. Only a pointer to the file is stored in the database, not the data itself. If you edit the operating system file outside the context of the database server, the **etx** index is not automatically updated. In this case, you must manually update the index by either issuing an UPDATE statement to update the row that contains the pointer to the file or dropping and re-creating the **etx** index.

For more detailed information on the IfxDocDesc data type, see [“IfxDocDesc” on page 4-8](#).

IfxMRData

The main advantage of using the multirepresentational data type IfxMRData is that it dynamically determines whether to store your document in an LVARCHAR data type or a CLOB data type, depending on the size of the document.

This feature is useful if the documents you want to store in a single column of a table vary greatly in size, where some fit in an LVARCHAR data type and some do not. It is preferable to store documents in LVARCHARs columns if at all possible, since it is faster to insert documents into and retrieve documents from this data type than into smart large object types such as CLOB.

IfxMRData lets the data type itself decide where to store the data so that you do not have to make this decision at the time you create the table.

IfxMRData is also designed to improve I/O performance of documents that contain only standard ASCII text. Therefore, if your documents do not contain binary data, storing them in a column of this data type might improve performance.

For more detailed information on the IfxMRData data type, see [“IfxMRData” on page 4-13](#).

Loading Data

You can expedite indexing by the way you load your data into the table.

Nonindexed Tables

Indexing is fastest when you load the data into a nonindexed table and then create the **etx** index.

Preindexed Tables

It is not always possible to insert all data into a table before you create the index. If you must insert large numbers of rows into a preindexed table, Informix suggests that you collect multiple rows in a nonindexed staging or temporary table and then insert them into the main table as a transaction.

For example, suppose the preindexed table is called **maintab**. Create a table called **stagingtab** that looks exactly like the **maintab** table, but with no **etx** index built on it. Insert a batch of documents into the **stagingtab** table; then insert these documents into the **maintab** table using an INSERT statement similar the following example:

```
BEGIN WORK;  
INSERT INTO maintab ... SELECT * FROM stagingtab;  
COMMIT WORK;
```

You can also improve performance by encapsulating the INSERT statement in a transaction via the BEGIN WORK and COMMIT WORK statements; doing this allows a table lock to be taken immediately on the **maintab** table, so you avoid the cost of lock escalation.

If you are inserting data into a fragmented **etx** index, you can further optimize the insertion throughput if you can segment your data to match the index fragmentation scheme, if you have multiple CPUs available, and if you can run concurrent data loading.

Informix recommends that you create a temporary table and data loading process for each index fragment. Each data loading process would first insert the data into its own copy of a staging table and then into the target table. Doing this affects only one index fragment and thereby locks only that fragment. Additional loaders can load data into separate staging tables and index fragments and take advantage of the parallelism of the multiple CPU system.

You must balance the number of rows you stage against concurrency requirements, because the affected index fragment or fragments are locked against all other users until the insertion is completed and the transaction is committed.

Handling Deadlocks

Informix recommends that you program your application or instruct your users to retry data loading statements that encounter a deadlock error.

A number of factors can cause a deadlock error. With concurrent updates and inserts, the database server can encounter a deadlock detected error if the updates are using table scans to evaluate the WHERE clause.

Concurrent inserts and selects can result in a deadlock if the table uses page-level locking.

Concurrent searches with the Excalibur Text Search DataBlade module might also raise deadlock errors.

When the **etx_contains()** operator is executed, it immediately generates a hitlist that contains the IDs of all the rows that match the query.

The application then begins fetching the rows specified by the hitlist. After the hitlist is generated, but before a row is fetched, a second user session might try to update or delete a row on the hitlist. This places an exclusive lock on that row. The update or insert process waits for a lock on the index that is being held by the original application. When the original application then tries to access that row, the database server issues a deadlock detected or “Could Not Position Within a Table” error.

Refining Queries

You might have to refine some of the SQL syntax you are used to using to take advantage of the features of the Excalibur Text Search DataBlade module. Keep in mind the following rules:

- You can specify only one **etx_contains** operation in the WHERE clause of a SELECT statement.
- You cannot specify the NOT keyword in the WHERE clause of a SELECT statement
- The database server completes **etx** index scans before it applies any additional filters specified in the WHERE clause.

The following sections provide examples of how these rules affect your query planning.

Queries That Contain Multiple **etx_contains()** Operators

If you specify the **etx_contains()** operator in a query, the operator must be able to scan the index. If you execute a query and no **etx** index is available, the Excalibur Text Search DataBlade module returns an error.

This affects how complex queries must be written. For example, suppose you are interested in finding recipes that mention “lemon zest” or ingredients that include “orange rind.” Two separate **`etx_contains()`** operators must be used in the query. Unfortunately, the following type of query returns an error because the database server cannot use a single index scan for the two **`etx_contains()`** operators:

```
SELECT id FROM recipes
  WHERE etx_contains(recipe, Row ('lemon zest', 'SEARCH_TYPE = PHRASE_EXACT'))
 OR
      etx_contains(ingredients, Row ('orange rind', 'SEARCH_TYPE =
PHRASE_EXACT'));
```

The query must be rewritten into two separate queries combined with the **UNION** operator, as shown in the following example:

```
SELECT id FROM recipes
  WHERE etx_contains(recipe, Row ('lemon zest', 'SEARCH_TYPE = PHRASE_EXACT'))
UNION
SELECT id FROM recipes
  WHERE etx_contains(ingredients, Row ('orange rind', 'SEARCH_TYPE =
PHRASE_EXACT'));
```

Similarly, an **AND** clause involving two **`etx_contains()`** operators returns an error. The alternative is to rewrite the query as shown by the following example:

```
SELECT id FROM recipes
  WHERE etx_contains(recipe, Row ('lemon zest', 'SEARCH_TYPE = PHRASE_EXACT'))
AND id IN
  (SELECT id FROM recipes
    WHERE etx_contains(ingredients, Row ('orange rind', 'SEARCH_TYPE =
PHRASE_EXACT')));
```

The query plan for this type of query shows a semi-join of the result of the two index scans. Another alternative is to use two table aliases for the same table and to join the table to itself via a unique column.

It is possible to execute multiple **`etx_contains()`** operators against more than one column in the same or multiple tables as long as you do not specify them in the same **WHERE** clause. Examine the query plan to make sure that the database server has selected index scans for the **`etx_contains()`** operator.

Queries That Include the NOT Keyword

You cannot use the NOT keyword before the **etx_contains()** operator in the WHERE clause of a query. You must rewrite the query so that it does not use the NOT keyword.

For example, the following query returns an error:

```
SELECT id FROM recipes
WHERE NOT etx_contains (ingredients, 'eggs');
```

Consider splitting the query into the following two queries:

```
SELECT id FROM recipes
WHERE etx_contains(ingredients, 'eggs') INTO TEMP t1;
SELECT id FROM recipes
WHERE id NOT IN (SELECT id FROM t1);
```

Queries That Include Index Scans and Nonindex Scan Filters

You might experience slowness in queries that combine index scans and nonindex scan filters. For example, you might expect the following query to run fast because the WHERE clause filters any row that does not have ID 18:

```
SELECT COUNT(*) FROM recipes
WHERE etx_contains ( ingredients, 'eggs') and id = 18;
```

The preceding query, however, runs about as fast as the following query:

```
SELECT COUNT(*) FROM recipes
WHERE etx_contains ( ingredients, 'eggs');
```

If a query includes an **etx_contains** scan, it completes the scan before it applies any additional filters. When you execute a query that contains an **etx_contains** scan, remember that the time needed to perform the query is first affected by the number of rows found in the index and then by the number of rows found by the SQL statement.

In each of the preceding cases, the time it takes to execute the query is proportional to the number of recipes that contain the word *eggs*—even if recipe 18 does not use any eggs.

Character Sets

When you create an **etx** index, you specify the character set the search engine uses by setting the `CHAR_SET` index parameter. The value you specify determines which characters in your text data are indexed and which are treated as white space.

Informix provides three built-in character sets: ASCII, ISO, and `OVERLAP_ISO`. You can also define your own character sets if these are inadequate for your particular text.

You cannot change the setting of the `CHAR_SET` parameter once you create the index unless you first drop and then re-create the index. If you do not specify a setting for `CHAR_SET`, the text engine uses the ASCII character set by default.

This appendix contains a complete description of the following three built-in character sets:

- ASCII
- ISO
- `OVERLAP_ISO`

The last section of this appendix, “[ISO 8859-1](#),” contains a 16 x 16 mapping of all the characters in the ISO 8859-1 character set. You might want to use this mapping as a reference when you define your own character set.

ASCII

The text search engine uses this character set by default. The text engine uses this table to translate just the alphanumeric characters in the ASCII set (0 through 9, A through Z, and a through z). The lowercase characters are translated to uppercase, so this table results in 36 possible unique character values.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3	0	1	2	3	4	5	6	7	8	9						
4		A a	B b	C c	D d	E e	F f	G g	H h	I i	J j	K k	L l	M m	N n	O o
5	P p	Q q	R r	S s	T t	U u	V v	W w	X x	Y y	Z z					
6																
7																
8																
9																
A																
B																
C																
D																
E																
F																

ISO

The ISO character set is a superset of the ASCII character set. In addition to the standard ASCII characters, this character set contains the ISO Latin-1 characters. Each ISO character has its own value, except that lowercase characters are translated to uppercase. Two characters that have no uppercase equivalent are:

- German small sharp s (ß - 0xDF)
- Lowercase y diaeresis/umlaut (ÿ - 0xFF)

This table results in 68 possible unique characters.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3	0	1	2	3	4	5	6	7	8	9						
4		A a	B b	C c	D d	E e	F f	G g	H h	I i	J j	K k	L l	M m	N n	O o
5	P p	Q q	R r	S s	T t	U u	V v	W w	X x	Y y	Z z					
6																
7																
8																
9																
A																
B																
C	À à	Á á	Â â	Ã ã	Ä ä	Å å	Æ æ	Ç ç	È è	É é	Ê ê	Ë ë	Ì ì	Í í	Î î	Ï ï
D		Ñ ñ	Ò ò	Ó ó	Ô ô	Õ õ	Ö ö		Ø ø	Ù ù	Ú ú	Û û	Ü ü			ß
E																
F																ÿ

OVERLAP_ISO

The ISO Latin-1 character set is a superset of the ASCII character set. In addition to the standard ASCII characters, this character set contains the ISO Latin-1 alphabetic characters, arranging the ISO characters into groups of similar-looking characters so that they will share the same numeric values. Uppercase and lowercase characters are grouped together.

Four exceptions that defy convenient categorization are assigned their own unique values and are listed in the following table.

Character Name	Lowercase Symbol	Lowercase Value	Uppercase Symbol	Uppercase Value
AE diphthong	æ	0xE6	Æ	0xC6
Icelandic Eth		0xF0		0xD0
Icelandic Thorn		0xFE		0xDE
German small sharp s	ß	0xDF		

The OVERLAP_ISO character set is shown on the following page.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2																
3	0	1	2	3	4	5	6	7	8	9						
4		Aa Ăă Ää Ȧȧ ǼǼ	Bb Çç	Cc Çç	Dd d	Ee Ěě Êê Èè Éé	Ff f	Gg g	Hh h	Ii İi Íí Îî Ïï	Jj j	Kk k	Ll l	Mm m	Nn Ññ	Oo Ōō Ȫȫ Ȭȭ Ȯȯ Ȱȱ Ȳȳ
5	Pp	Qq	Rr	Ss	Tt	Uu Ūū Ǫǫ Ǭǭ	Vv v	Ww w	Xx x	Yy ÿ	Zz z					
6																
7																
8																
9																
A																
B																
C							Ææ									
D																b
E																
F																

ISO 8859-1

The following table shows the ISO 8859-1 character set. You can reference this table when you create the file that contains the 16 x 16 matrix of characters you want indexed when you create your own character set. Use the routine **etx_CreateCharSet()** to make the user-defined character set known to the Excalibur Text Search DataBlade module.

See [“etx_CreateCharSet\(\)” on page 5-16](#) for more information about this routine.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	NL	VT	NP	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[/]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL
8																
9																
A	NBS	ı	ç	£	¤	¥		§	¨	©	ª	«	¬	-	®	¯
B	°				´		¶	·	¸		º	»				¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D		Ñ	Ò	Ó	Ô	Õ	Ö		Ø	Ù	Ú	Û	Ü			ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F		ñ	ò	ó	ô	õ	ö		ø	ù	ú	û	ü			ÿ

Document Formats You Can Filter

This appendix lists the document formats and versions you can filter with the Excalibur Text Search DataBlade module.

Generic Word Processing Formats

Format	Version
ASCII Text (7 and 8 bit)	All
ANSI Text (7 and 8 bit)	All
Unicode Text	All
HTML	Through 3.0
IBM Revisable Form Text	All
IBM FFT	All
Microsoft Rich Text Format	Through 2.0

DOS Word Processing Formats

Format	Version
DEC WPS-PLUS	Through 4.1
DEC WPS-PLUS (DX)	Through 4.1
DisplayWrite 2 & 3 (TXT)	All
DisplayWrite 4 & 5	Through Release 2.0
Enable	3.0, 4.0, and 4.5
First Choice	Through 3.0
Framework	3.0
IBM Writing Assistant	1.01
Manuscript	Through 2.0
MASS11	Through 8.0
Microsoft Word	Through 6.0
Microsoft Works	Through 2.0
MultiMate	Through 4.0
Navy DIF	All
Nota Bene	3.0
Office Writer	4.0 through 6.0
PC-File Letter	Through 5.0
PC-File+ Letter	Through 3.0
PFS:Write	A, B, and C
Professional Write	Through 2.1
Q&A	2.0
Samna Word	Through IV+

(1 of 2)

Format	Version
SmartWare II	1.02
Sprint	Through 1.0
Total Word	1.2
Volkswriter 3 & 4	Through 1.0
Wang PC (IWP)	Through 2.6
WordMARC	Through Composer Plus
WordPerfect	Through 7.0
WordStar	Through 7.0
WordStar 2000	Through 3.0
XyWrite	Through III Plus

(2 of 2)

Windows Word Processing Formats

Format	Version
AMI/AMI Professional	Through 3.1
JustWrite	Through 3.0
Microsoft Windows Works	Through 4.0
Microsoft Windows Write	Through 3.0
Microsoft Word 97	
Microsoft Word for Windows	Through 7.0
Microsoft WordPad	All
Novell Perfect Works	2.0
WordPerfect for Windows	Through 7.0
Professional Write Plus	1.0
Legacy	Through 1.1
Q&A Write for Windows	3.0
WordStar for Windows	1.0

Macintosh Word Processing Formats

Format	Version
Microsoft Word	4.0 through 6.0
WordPerfect	1.02 through 3.0
Microsoft Works (Mac)	Through 2.0
MacWrite II	1.1

Spreadsheet Formats

Format	Version
Enable	3.0, 4.0, and 4.5
First Choice	Through 3.0
Framework	3.0
Lotus 1-2-3 (DOS & Windows)	Through 6.x
Lotus 1-2-3 Charts (DOS & Windows)	Through 5.0
Lotus 1-2-3 (OS/2)	Through 2.0
Lotus 1-2-3 Charts (OS/2)	Through 2.0
Lotus Symphony	1.0, 1.1, and 2.0
Microsoft Excel 97	
Microsoft Excel Windows	2.2 through 7.0
Microsoft Excel Macintosh	3.0 through 4.0
Microsoft Excel Charts	2.x through 7.0
Microsoft Multiplan	4.0
Microsoft Windows Works	Through 4.0
Microsoft Works (DOS)	Through 2.0
Microsoft Works (Mac)	Through 2.0
Mosaic Twin	2.5
Novell Perfect Works	2.0
QuattroPro for DOS	Through 5.0
QuattroPro for Windows	Through 7.0
PFS:Professional Plan	1.0

(1 of 2)

Format	Version
SuperCalc 5	4.0
SmartWare II	1.02
VP Planner 3D	1.0

(2 of 2)

Database Formats

Format	Version
Access	Through 2.0
dBASE	Through 5.0
DataEase	4.x
dBXL	1.3
Enable	3.0, 4.0, and 4.5
First Choice	Through 3.0
FoxBase	2.1
Framework	3.0
Microsoft Windows Works	Through 4.0
Microsoft Works (DOS)	Through 2.0
Microsoft Works (Mac)	Through 2.0
Paradox (DOS)	Through 4.0
Paradox (Windows)	Through 1.0
Personal R:BASE	1.0
R:BASE	Through 3.1
R:BASE System V	1.0
Reflex	2.0
Q & A	Through 2.0
SmartWare II	1.02

Standard Graphic Formats

Format	Version
Binary Group 3 Fax	All
BMP (including RLE, ICO, CUR & OS/2 DIB)	Windows
CDR—Corel Draw (if TIFF image is embedded)	2.0 through 5.0
CGM—Computer Graphics Metafile	
ANSI, CALS, NIST,	3.0
CMXCorel Clip Art Format	
DCX (multipage PCX) Microsoft Fax	
DRW—Micrografx Designer	3.1
DXF (binary and ASCII) AutoCAD	
Drawing Interchange Format	Through 13
EPS Encapsulated PostScript (if TIFF image is embedded)	
FMV FrameMaker	Vector and raster format
GDF—IBM Graphics Data Format	
GEM—Graphics Environment Manager Metafile	Bitmap and vector
GIF—Graphics Interchange Format	CompuServe
GP4—Group 4 CALS Format	
HPGL—Hewlett-Packard Graphics Language	2.0
IMG—GEM Paint	
JPEG	All
MAC	MacPaint
MET—OS/2 PM Metafile	3.0

(1 of 2)

Format	Version
PCD—Kodak Photo CD	
PCX	PC Paintbrush
Perfect Works (Draw)	Novell Version 2.0
PIC	Lotus
PICT1 & PICT2 (Raster)	Macintosh Standard
PIF—IBM Picture Exchange Format	
PNG—Portable Network Graphics Internet Format	Non-LZW
PNTG	MacPaint
RND—AutoShade Rendering File Format	
SDW	Ami Draw
Snapshot (Lotus)	All
SRS—Sun Raster File Format	
TGA (TARGA)	Truevision
TIFF	Through 6
TIFF CCITT Group 3 & 4	Fax Systems
WMF	Windows Metafile
WordPerfect Graphics (WPG and WPG2)	Through 2.0
XBMX-Windows Bitmap	
XPMX-Windows Pixmap	
XWDX-Windows Dump	

(2 of 2)

High-End Graphic Formats

Format	Version
AI—Adobe Illustrator	Through 6.0
CDR—Corel Draw	Through 7.0
DSF—Micrografx Designer	Windows 95, Version 6.0
DWG—AutoCAD Native Drawing Format	12 and 13
IGES—Initial Graphics Exchange Specification	5.1
PDF—Portable Document Format	Acrobat Version 2.1 (LZW)
PS—PostScript	Level 2 (LZW)

Presentation Formats

Format	Version
Corel Presentations	7.0
Harvard Graphics for DOS	2.x and 3.x
Freelance 96 for Windows 95	
Freelance for Windows	1.0 and 2.0
Freelance for OS/2	Through 2.0
Microsoft PowerPoint for Windows	Through 7.0
Microsoft PowerPoint for Macintosh	4.0

Compressed and Encoded Formats

Format	Version
LZH Compress	
LZA Self Extracting Compress	
Microsoft Binder	7.0
MIME (text mail)	
UUE	
UNIX Compress	
UNIX TAR	
ZIP	PKWARE versions through 2.04g

Other Formats

You can also filter the following types of formats:

- Executable (EXE, DLL)
- Executable for Windows NT

etx_lists Table

This appendix describes the **etx_lists** table, a system table that contains information about stopword lists, synonym lists, and user-defined character sets. Each time you create or drop one of these objects, an entry for the object is inserted or deleted, respectively, in the table.

The **etx_lists** table has the following columns.

Column Name	Type	Explanation
list_name	VARCHAR(255)	Internal name for word lists and user-defined character sets that exist in a database.
slob_handle	BLOB	Points to the smart large object that contains the DataBlade object.



Important: The **etx_lists** table is described for your information only; you should never update this table manually. You can, however, query the table to find out what objects (such as synonym and stopword lists) currently exist in your database.



Glossary

access method	A set of server routines that the database server uses to access and manipulate an index or a table. etx is an example of a secondary access method.
approximate phrase search	A search in which the search text must contain a phrase identical to the clue, or one or more words of the clue. The order of the words in the clue is not important. For example, if the clue is <code>buy three dolls</code> , the search engine returns documents that contain the exact phrase as well as those that contain the phrases <code>three dolls</code> , <code>buy dolls</code> , or <code>dolls buy</code> .
BLOB	<p>A smart large object data type that stores any kind of binary data, including images. The database server performs no interpretation on the contents of a BLOB column.</p> <p>See also <i>smart large object</i>.</p>
Boolean search	A search that uses the Boolean expressions (logical operators) <code>&</code> (AND), <code> </code> (OR), <code>!</code> and <code>^</code> (NOT). Use the <code>&</code> Boolean operator when you want to search for documents that contain all the words in a keyword list; use <code> </code> when you want to search for documents that contain at least one word in the list; and use <code>!</code> OR <code>^</code> when you want to search for documents that do not contain a specified word. The Boolean operators can be combined to make more complicated expressions. This type of search is activated by setting the <code>SEARCH_TYPE</code> tuning parameter to <code>BOOLEAN_SEARCH</code> .
CLOB	<p>A smart large object data type that stores blocks of text items, such as ASCII or PostScript files.</p> <p>See also <i>smart large object</i>.</p>

clue	The data that you are searching for, specified as the second argument to the etx_contains() operator.
document score	A value that the text search engine assigns to each of the returned rows of a fuzzy search that specifies the degree of similarity between your clue and each of the returned rows. Scores vary between 0 and 100, with 0 indicating no match and 100 indicating a perfect match. You access scoring information via the third parameter of the etx_contains() operator, a statement local variable (SLV). The data type of the SLV is etx_ReturnType , an Informix-defined row type that consists of three fields. The scoring information is contained in the score field.
filtering	A component of the Excalibur Text Search DataBlade module that automatically filters out all proprietary formatting information from a formatted document and converts it into ASCII form.
exact phrase search	A search for text that matches your clue exactly. An exact phrase search is successful when the text search engine finds a phrase that contains all the words in the clue in the exact order that you specify.
fuzzy search	A search for text that matches your clue approximately instead of exactly. A fuzzy search takes into account substitutions, transpositions, and basic pattern matching. A search that returns a document that contains the word <code>editor</code> when searching for <code>editor</code> is an example of a fuzzy search.
hit	The result (a row) of a text search.
hitlist	A list of hits (rows).
highlighting	The process of retrieving the location of every instance of a clue in the search text. The Excalibur Text Search DataBlade module returns highlighting information via pairs of beginning and ending offsets relative to filtered documents.
index parameter	A variable that you use to specify the characteristics of an etx index to accommodate the searches you plan to perform. An example of an index parameter is <code>WORD_SUPPORT='EXACT'</code> .
keyword	Any contiguous group of characters found in the search text or clue, delimited by nonindexable characters such as spaces or tabs.

keyword search	A search in which the words in the clue are treated as separate entities (keywords) instead of a single unit (phrase). When the text search engine performs a keyword search, it returns a row whenever it encounters one or more of the words in your clue.
operator class	The set of operators that the database server associates with a secondary access method. When an index is created, it is associated with a particular operator class.
pattern search	See <i>fuzzy search</i> .
phrase search	A search in which the words in the clue are treated as a single unit (phrase) instead of separate entities (keywords). The two types of phrase searches are exact and approximate.
proximity search	A search in which you specify the number of nonsearch words that can occur between two or more of the search words. You use a proximity search if, for example, you are searching for a phrase that contains the words <code>editor</code> and <code>multimedia</code> but do not want the two keywords separated by more than four nonsearch words. This type of search is activated by setting the tuning parameter <code>SEARCH_TYPE</code> equal to <code>PROX_SEARCH</code> .
rank	The order given to a hitlist based on the score of each of the returned rows.
root word	The word in a synonym list that has one or more synonyms defined for it. It is the leftmost word of a single row of the synonym operating system file. When synonym matching is activated, the keyword being searched for must be a root word for its synonym to be returned instead.
row data type	<p>A complex data type consisting of a group of ordered data elements (fields) of the same or different data types. The fields of a row type can be of any supported built-in or extended data type, including complex data types, except <code>SERIAL</code> and <code>SERIAL8</code> and, in certain situations, <code>TEXT</code> and <code>BYTE</code>.</p> <p>There are two kinds of row data types:</p> <ul style="list-style-type: none"> ■ Named row types, created using the <code>CREATE ROW TYPE</code> statement ■ Unnamed row types, created using the <code>ROW</code> constructor
sbspace	A logical storage area that contains one or more chunks that store only smart large object data.
score	See <i>document score</i> , <i>word score</i> .

search string	See <i>clue</i> .
search text	The data that is to be searched, stored in a column of a table. The column can be of type IfxDocDesc, BLOB, CLOB, CHAR, VARCHAR, or LVARCHAR.
SLV	Abbreviation for <i>statement local variable</i> .
smart large object	<p>A large object that:</p> <ul style="list-style-type: none"> ■ is stored in an sbspace, a logical storage area that contains one or more chunks. ■ has read, write, and seek properties similar to a UNIX file. ■ is recoverable. ■ obeys transaction isolation modes. ■ can be retrieved in segments by an application. <p>Smart large objects include CLOB and BLOB data types.</p>
statement local variable (SLV)	A variable for storing a value that a function returns indirectly, through a pointer, in addition to the value that the function returns directly. An SLV's scope is limited to the statement in which it is used. The third optional parameter of the etx_contains() operator is an SLV that holds scoring and highlighting information. The data type of the SLV is etx_ReturnType.
stopword	A keyword that you want excluded from your index or your search. Stop-words are typically common words such as <i>and</i> , <i>or</i> , <i>the</i> , and <i>to</i> , or any word that appears frequently in your document that you want to exclude.
substitution	A misspelling of a word, in which one letter has been substituted by another, incorrect one. Misspelling <i>searck</i> for <i>search</i> is an example of a substitution.
synonym	One of two or more words or expressions that have the same or nearly the same meaning in some or all senses. The word <i>java</i> is a synonym of the word <i>coffee</i> .
text search engine	The component of the Excalibur Text Search DataBlade module that calls the Text Retrieval Library (TRL) of Excalibur Technologies to perform a search. The TRL is a library of C-language object modules designed to perform fast retrieval and automatic indexing of text data. The text search engine is dynamically linked into Informix Dynamic Server with Universal Data Option whenever a text search is performed or text data is indexed.

transposition	A misspelling of a word in which two adjacent letters switch positions. Misspelling <i>saerch</i> for <i>search</i> is an example of a transposition.
tuning parameter	A variable used to guide the way the text search engine conducts a search. Tuning parameters are passed to the text search engine via the second parameter of the Row() constructor of the etx_contains() operator. An example of a tuning parameter is <code>SEARCH_TYPE = WORD</code> .
word score	The search engine uses fuzzy logic to determine whether or not a pattern match should be considered a hit. It assigns a word score to candidate matches based on its internal rules. By default, only words that match your search clue by a relative measure of 70 out of 100—that have a word score of 70 or better—are considered hits.

Error Messages

This section lists all the Excalibur Text Search DataBlade module errors, the error text, and a brief description of the error, as well as corrective action to take if you encounter the error.

EUET02 Internal Error: Control block not found in (%UDRNAME%).

This is an internal error. If it occurs again, take note of all the circumstances that caused the error and contact Informix technical support.

EUET03 Memory allocation failed in (%UDRNAME%).

The Excalibur Text Search DataBlade module needed to allocate data-space memory to process the query, but none was available.

This error might reflect a hardware limit, an operating system configuration limit, or a temporary shortage of disk space. Retry the query after a short delay. If the query continues to fail with the same error, consult your system administrator. If you think the error is not due to a shortage of memory, take note of all the circumstances that caused the error and contact Informix technical support.

EUET04 SAPI error (Sql %SQL%, Isam %ISAM%) in (%UDRNAME%).

The Excalibur Text Search DataBlade module received an error from the database server.

If the SQL or ISAM error codes are initialized—if they have a value other than -999—use the **finderr** utility to get more information on the failure.

You can use the **finderr** utility to find out what error -12036 means in the following example:

```
SAPI error (Sql -9810, Isam -12036 ) in etx_FseMkDir.
```

finderr points to the real problem: the specified sbospace does not exist.

There are some errors in which the SQL or ISAM error codes might not be initialized. The most likely cause of this error is a problem with accessing the sbospace. Check to be sure the sbospace exists and that there is sufficient space available. If this does not solve the problem, take note of all the circumstances that caused the error and contact Informix technical support.

EUET05 Etx Index does not exist on column or optimizer chose Non-index scan.

The **etx_contains()** operator is being used on a column that does not have an **etx** index. Be sure you have created an **etx** index on the column.

This error might also occur in cases where the **etx** index exists, but the query optimizer does not choose to use the index when it evaluates the **etx_contains()** operator. The SQL command SET EXPLAIN ON can provide more information on the strategy used by the query optimizer to execute the query. Informix suggests that you try to rewrite the query to force the optimizer to use the index.

For example, assume that the following query returns the EUET05 error, and an **etx** index exists on the column specified in the **etx_contains()** operator:

```
SELECT id FROM recipes
WHERE id = 1 OR etx_contains (ingredients, 'eggs');
```

One way to force the optimizer to use the **etx** index is to execute the following two queries instead of the original query:

```
SELECT id FROM recipes
WHERE etx_contains (ingredients, 'eggs') INTO TEMP t1;
SELECT id FROM recipes
WHERE id = 1 OR id IN (SELECT id FROM t1);
```

See [“Refining Queries” on page 7-23](#) for a related discussion.

EUET20 Smart Blob space does not exist or not specified.

Verify that the sbospace exists or see if you have specified a dbspace instead of an sbospace.

If you want to use the default sbspace, verify that you specify the SBSPACENAME parameter correctly in your ONCONFIG file. You might need to explicitly specify an sbspace if you have previously created a fragmented table in a dbspace other than the **rootdbs** default dbspace.

EUET22 Word list (%NAME%) already exists.

You tried to create a synonym or stopword list that already exists. Pick a new name for the list or drop the original list.

EUET23 Word list (%NAME%) does not exist.

You tried to access a synonym or stopword list that does not exist. Either create the list or verify that the list you specified is spelled correctly.

EUET25 Only simple qualifications are supported. Please rewrite your query.

The Excalibur Text Search DataBlade module does not support the use of more than one **etx_contains()** operator on the same column in a single SELECT statement. You must rewrite the query so that only one **etx_contains()** operator is used.

For example, the following query is not supported:

```
SELECT id FROM recipes
WHERE etx_contains (ingredients, 'lemon zest')
AND etx_contains (ingredients, 'orange rind');
```

You can rewrite this query as a Boolean phrase search. See [“Boolean Searches” on page 2-5](#) for details. See [“Refining Queries” on page 7-23](#) for additional discussion about queries that contain multiple **etx_contains()** scans in a WHERE clause.

EUET26 Qualification predicate does not match strategy function in opclass.

This is an internal error. If it occurs again, take note of all the circumstances that caused the error and contact Informix technical support.

EUET27 Improper type for search clue specified

This is an internal error. If it occurs again, take note of all the circumstances that caused the error and contact Informix technical support.

EUET28 Negated qualifications not supported.

You cannot use the NOT keyword before the **etx_contains()** operator in the WHERE clause of a query. You must rewrite the query so that it does not use the NOT keyword.

For example, the following query is not supported:

```
SELECT id FROM recipes
WHERE NOT etx_contains (ingredients, 'eggs');
```

A suggested workaround is to split the query into the following two queries:

```
SELECT id FROM recipes
WHERE etx_contains(ingredients, 'eggs') INTO TEMP t1;
SELECT id FROM recipes
WHERE id NOT IN (SELECT id FROM t1);
```

EUET29 Invalid argument in (%UDRNAME%).

You passed an invalid argument to the specified Excalibur Text Search DataBlade module routine.

If, however, you did not execute the routine directly, and the error occurs again, contact Informix technical support.

EUET30 Index fragment locked (Sql %SQL%, Isam %ISAM%).

The Excalibur Text Search DataBlade module is unable to get a lock on the index fragment.

To work around locking problems, use the SET LOCK MODE TO WAIT or SET LOCK MODE TO WAIT *n* SQL statements. If you already set the lock mode to wait and the error occurs again, or the error occurs when there is not more than one query running concurrently, use the **finderr** utility to check the SQL and ISAM error codes for more information.

You can use the **finderr** utility to find out what the error -143 means in the following example:

```
Index Fragment locked (Sql -9810, Isam -143)
```

In this case, the database server has encountered a deadlock. Another possibility is that the sbspace has run out of space.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET31 Internal Error: (%ERRNO%) returned from TRL to %UDRNAME%.

This is an internal error. If it occurs again, take note of all the circumstances that caused the error and contact Informix technical support.

EUET33 Access Method does not support clustered index.

You cannot create a clustered **etx** index or alter an **etx** index to be clustered. Trying to alter an **etx** index to be clustered can return a misleading error message.

EUET34 Incorrect creation parameters specified. (%UDRNAME%)

Verify that the value assigned to the WORD_SUPPORT, PHRASE_SUPPORT, or FILTER index parameter in the CREATE INDEX statement is valid.

This error will also occur if the INCLUDE_STOPWORDS index parameter is set to TRUE but the stopwords list specified in the STOPWORD_LIST index parameter does not exist.

EUET35 Index (%NAME%) not created with WORD_SUPPORT=PATTERN.

This error indicates you tried to perform a fuzzy or pattern search on an **etx** index that was not created with the WORD_SUPPORT=PATTERN. If this is the case, drop the index and re-create it with the specified index parameter.

For more information on what index parameters must have been specified for you to be able to use the various tuning parameters, refer to the [Chapter 6, “etx Index Parameters.”](#)

EUET36 Index (%NAME%) not created with PHRASE_SUPPORT enabled.

This error indicates you tried to perform a phrase or proximity search on an **etx** index that was not created with the PHRASE_SUPPORT=MEDIUM or PHRASE_SUPPORT=MAXIMUM index parameter. If this is the case, drop the index and re-create it with the specified index parameter.

For more information on what index parameters must have been specified for you to be able to use the various tuning parameters, refer to the [Chapter 6, “etx Index Parameters.”](#)

EUET37 Index (%NAME%) not created with INCLUDE_STOPWORDS=TRUE.

This error indicates that you tried to use the CONSIDER_STOPWORDS tuning parameter in a search that uses an **etx** index that was not created with the INCLUDE_STOPWORDS=TRUE index parameter. If this is the case, drop the index and re-create it with the specified index parameter.

For more information on what index parameters must have been specified for you to be able to use the various tuning parameters, refer to the [Chapter 6, “etx Index Parameters.”](#)

EUET38 Cannot parse line %LINENO% of charset definition file.

EUET39 Too few or too many lines in charset definition file.

The operating system file that contains the definition of a user-defined character set should consist of 16 lines of 16 hexadecimal numbers, with optional comment lines.

For more information on how to create user-defined character sets, refer to the [“Creating a User-Defined Character Set” on page 3-8.](#)

EUET3A Charset (%NAME%) does not exist.

The Excalibur Text Search DataBlade module attempted to access the code table for a character set that does not exist.

Be sure you spelled the character set correctly, or create it if it does not exist.

EUET3B Cannot use ASCII, ISO, OVERLAP_ISO as the name of user-defined charset.

The words ASCII, ISO, and OVERLAP_ISO are reserved words and cannot be used as names for your user-defined character sets. You must pick a different name.

EUET3C Charset (%NAME%) already exists.

The Excalibur Text Search DataBlade module attempted to create a character set that already exists. Be sure you spelled the character set correctly or choose a different name.

EUET3D Error accessing requested synonym list.

The Excalibur Text Search DataBlade module was unable to access the specified synonym list.

Be sure you spelled the synonym list correctly and that it exists. If the list does exist and the error occurs again, contact Informix technical support.

EUET42 Syntax error in tuning parameters.

EUET60 Error (Sql %SQL%, Isam %ISAM%) in opening (%NAME%).

The Excalibur Text Search DataBlade module was unable to open a file or directory.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

The following problems could have caused this error:

- A synonym or stopword list used in a query has not been created yet.
- External files specified by the IFX_FILE protocol of the IfxDocDesc data type might not exist.
- The specified sbspace does not have available space.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET61 Error (Sql %SQL%, Isam %ISAM%) in closing a file in (%UDRNAME%).

The Excalibur Text Search DataBlade module was unable to close an internal file.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET63 Error (Sql %SQL%, Isam %ISAM%) in removing dir (%NAME%).

The Excalibur Text Search DataBlade module was unable to remove an internal directory.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET64 (%NAME%) already exists.

The Excalibur Text Search DataBlade module was unable to create a file or a directory.

If the directory or file that could not be created is internal and the problem occurs again, note all the circumstances that caused the error and contact Informix technical support.

EUET65 Error (Sql %SQL%, Isam %ISAM%) in creating (%NAME%).

The Excalibur Text Search DataBlade module was unable to create an internal file or directory.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET67 Error (Sql %SQL%, Isam %ISAM%) in rewinding a dir in (%UDRNAME%).

The Excalibur Text Search DataBlade module was unable to rewind an internal directory.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET68 Error (Sql %SQL%, Isam %ISAM%) in removing a file in (%UDRNAME%).

The Excalibur Text Search DataBlade module was unable to remove an internal file.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET69 Error (Sql %SQL%, Isam %ISAM%) in writing to a file in (%UDRNAME%).

The Excalibur Text Search DataBlade module was unable to write to an internal file.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET6A Error (Sql %SQL%, Isam %ISAM%) in reading (%NAME%).

The Excalibur Text Search DataBlade module was unable to read a file.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET6B Error (Sql %SQL%, Isam %ISAM%). Seek failed in (%UDRNAME%).

The Excalibur Text Search DataBlade module was unable to seek within an internal file.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET6C Error (Sql %SQL%, Isam %ISAM%). Cannot obtain stat on file (%NAME%).

The Excalibur Text Search DataBlade module was unable to obtain a stat on a file.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET6D Error (Sql %SQL%, Isam %ISAM%). Cannot truncate file (%UDRNAME%).

The Excalibur Text Search DataBlade module was unable to truncate an internal file.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

EUET81 Cannot copy Wordlist/Character Set file. (Sql %SQL%, Isam %ISAM%)

The Excalibur Text Search DataBlade module was unable to copy the stopword list, synonym word list, or character set file.

The following problems could have caused this error:

- The pathname to the specified file is incorrect.
- The permissions for the file are incorrect.
- The specified sbspace does not exist or does not have available space.

If the SQL or ISAM error codes are initialized, or in other words, if they are set to a value other than -999, use the **finderr** utility to get more information on the failure.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET83 String length exceeds %LIMIT%.

If the SQL and ISAM error codes are not initialized, which means they have a value of -999, and locking errors occur despite setting lock mode to wait, note all the circumstances that caused the error and contact Informix technical support.

EUET94 Illegal combination of tuning parameters specified.

See [Chapter 6, “etx Index Parameters,”](#) for information about tuning parameters.

EUET95 Syntax error in boolean search clue.

Check for errors in your search clue. This error is also returned if you include a phrase in the clue for a Boolean search on an index that was not created with PHRASE_SUPPORT enabled.

EUET96 Argument to (%UDRNAME%) is too long.

If you executed the Excalibur Text Search DataBlade module routine directly, check that the length of the passed argument does not exceed the maximum length of the argument as documented in the [Chapter 5, “Routines.”](#)

If the error occurs as a result of an internal call of the Excalibur Text Search DataBlade module, contact Informix technical support.

EUET97 File (%NAME%) does not exist.

Be sure you spelled the synonym list file, stopword list file, or character set file correctly and that it exists.

If the error occurs for an internal file and the problem occurs again, contact Informix technical support.

- EUETF1** Filter Error: Could not bind socket. Rowid %ROWNUM%, Frigid %FRAGID%.
- This indicates an operating system problem. You might need to restart **oninit** or reboot the machine to clear this error. Check for other conditions on the machine that might have an adverse effect on interprocess communication.
- EUETF2** Filter Error: Create socket failed. Rowid %ROWNUM%, Frigid %FRAGID%.
- This indicates an operating system problem. You might need to restart **oninit** or reboot the machine to clear this error. Check for other conditions on the machine that might have an adverse effect on interprocess communication.
- EUETF3** Filter Error: Set Socket option failed. Rowid %ROWNUM%, Frigid %FRAGID%.
- This indicates an operating system problem. You might need to restart **oninit** or reboot the machine to clear this error. Check for other conditions on the machine that might have an adverse effect on interprocess communication.
- EUETF4** Filter Error: FilterServer not found. Rowid %ROWNUM%, Frigid %FRAGID%.
- The filtering subsystem is unable to successfully start up and establish the required communication with the filter server. Be sure you have the correct version of the filter server. Also check that no old filter server process is still running. If it is, the DBA must stop it manually. Confirm that you have the correct VPCLASS entry for the ETX VP in your ONCONFIG file. See [“Specifying the ETX VPCLASS” on page 7-10](#) for guidelines.
- EUETF5** Filter Error: Init lo_spec failed. Rowid %ROWNUM%, Frigid %FRAGID%.
- Within the ETX_FILTER subsystem, the DataBlade API was unable to successfully initialize a large object (LO) for writing out the filtered data. Filtering was not performed, and a NULL LO handle was returned to the DataBlade module.
- EUETF6** Filter Error: Socket information unavailable. Rowid %ROWNUM%, Frigid %FRAGID%.
- This indicates an operating system problem. You might need to restart **oninit** or reboot the machine to clear this error. Check for other conditions on the machine that might have an adverse effect on interprocess communication.

- EUETF7** Filter Error: Socket listen failed. Rowid %ROWNUM%, Fragid %FRAGID%.
- This indicates an operating system problem. You might need to restart **oninit** or reboot the machine to clear this error. Check for other conditions on the machine that might have an adverse effect on interprocess communication.
- EUETF8** Filter error: Seek failed. Rowid %ROWNUM%, Fragid %FRAGID%.
- This is an internal error. The filtering subsystem is unable to seek to the beginning of the unfiltered input data for some reason.
- EUETF9** Filter error: Read failed. Rowid %ROWNUM%, Fragid %FRAGID%.
- This is an internal error. The filtering subsystem cannot read the unfiltered input data.
- EUETFa** Filter Error: FilterServer timed out. Rowid %ROWNUM%, Fragid %FRAGID%.
- This most likely indicates an operating system problem or possibly a problem with the filter server process. You might need to restart **oninit**, or possibly reboot the machine to clear this error. Check for other conditions on the machine that might have an adverse effect on interprocess communication. If you stop **oninit**, check to see if the filter server process remains up. If so, stop it manually.
- If you had the **ETX_FILTER_FILE** environment variable set when you started **oninit**, you should look at your log file. See [“Logging Messages from the Filter Server” on page 7-10](#) for instructions.
- EUETFB** Filter Error: FilterServer died/hung. Rowid %ROWNUM%, Fragid %FRAGID%.
- Check to be sure the filter server process is still running. There are several possible causes for this error. The filter server might have been killed, or it might have crashed. If you had the **ETX_FILTER_FILE** environment variable set when you started **oninit**, you should look at your log file. See [“Logging Messages from the Filter Server” on page 7-10](#) for instructions.
- The filter server might be filtering a document, such as a large PDF file, that takes a long time to filter. You might try setting the environment variable **FILTER_RETRIES** to some value greater than 50 (the default) and restarting **oninit**. Be sure to stop the old filter server process if it remains running after you stop **oninit**.

If this condition repeats each time you try to filter a particular document, remove that document temporarily from your application and call Informix Technical Support.

EUETFC	Filter Error: Socket accept failed. Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates an operating system problem. You might need to restart oninit or reboot the machine to clear this error. Check for other conditions on the machine that might have an adverse effect on interprocess communication.
EUETFD	Filter Error: Invalid File descriptor. Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates operating system error 9 (EBADF) returned to the DataBlade module while performing some socket operation.
EUETFE	Filter Error: Interrupt received. Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates operating system error 4 (EINTR) returned to the DataBlade module while performing some socket operation.
EUETFF	Filter Error: I/O Error. Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates operating system error 5 (EIO) returned to the DataBlade module while performing some socket operation.
EUETFG	Filter Error: Out of shared memory. Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates operating system error 12 (ENOMEM) returned to the DataBlade module while performing some socket operation.
EUETFH	Filter Error: Insufficient STREAMS resources. Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates operating system error 63 (ENOSR) returned to the DataBlade module while performing some socket operation.
EUETFI	Filter Error: Not a socket. Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates operating system error 95 (ENOTSOCK) returned to the DataBlade module while performing some socket operation.
EUETFJ	Filter Error: Stale NFS handle. Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates operating system error 151 (ESTALE) returned to the DataBlade module while performing some socket operation.

EUETFK	Filter Error: Operation would block. Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates operating system error 11 (EWOULDBLOCK) returned to the DataBlade module while performing some socket operation.
EUETFL	Filter Error: Rowid %ROWNUM%, Fragid %FRAGID%.
	This indicates an unexpected operating system error returned to the DataBlade module while performing some socket operation.
EUETFM	Filter Error %ERROR% from FilterServer. Rowid %ROWNUM%, Fragid %FRAGID%.
	The error message from the filter server is displayed in %ERROR%.
EUETFP	Invalid fd for output LO. Cannot filter document.
	Within the ETX_FILTER subsystem, the DataBlade API was unable to successfully initialize a large object (LO) for writing out the filtered data. Filtering was not performed, and a NULL LO handle was returned to the DataBlade module.
EUETFQ	Get named memory failed. Cannot filter document.
	This error reflects an internal problem with the DataBlade API within the Informix server. Try restarting oninit .
EUETFR	Filter input file: error on file length.
	This is an internal error. The filtering subsystem cannot get the length of the unfiltered input data.
EUETFS	Failed to write filtered data to output LO.
	The filtering subsystem failed on an mi_lo_write call while trying to write out the filtered data. One possible cause is that the default smart large object space is full or corrupted in some way. Check what other activity on the system might be using the default sbpace.
EUETFT	mi_open failed. Cannot filter document.
	This is an internal error. The filtering subsystem is unable to open a connection with the database server. Try restarting oninit .

Index

A

Access method, etx 1-7, 1-9, 2-3, 3-9
 Approximate phrase searches 2-9, 5-10
 ASC option to CREATE INDEX 1-9
 ASCII value for CHAR_SET index parameter 1-21, A-2

B

BLOB data type 1-7, 3-4, 7-19
 Boldface type Intro-6
 Boolean operators, escaping 2-6
 Boolean searches 1-14, 2-5, 5-10
 BOOLEAN_SEARCH value for SEARCH_TYPE tuning parameter 2-5, 5-10
 BUFFERS, ONCONFIG parameter 7-8
 Built-in character sets 1-21

C

Character sets 1-20, 3-10, 5-16, 5-25, 6-3, 6-4, A-1
 CHAR_SET index parameter 1-10, 1-20, 3-10, 5-16, 5-25, 6-3, 6-4, A-1
 CLOB data type 1-7, 3-4, 3-5, 4-13, 7-19
 Closing an index 5-14, 7-18
 Clue, definition of 1-12
 CLUSTER option to CREATE INDEX 1-9
 Comment icons Intro-7
 Concurrency 7-22

Configuring sbspaces 7-5, 7-6
 Configuring your database server for etx indexing 7-4
 Configuring your database server for filtering 7-9
 CONSIDER_STOPWORDS tuning parameter 1-20, 3-14, 5-8, 6-7
 Contacting Informix Intro-13
 Creating etx indexes
 filtered 3-12
 fragmented 3-12
 nonfragmented 3-10
 overview of 1-9
 tutorial for 3-10
 Creating sbspaces 7-5
 Creating stopword lists 1-19, 3-6, 5-20, 5-21, 6-9
 Creating synonym lists 1-16, 3-7, 5-22, 5-24
 Creating user-defined character sets 1-20, 3-8, 5-16, A-7

D

Data
 loading 7-21
 storing 3-4, 7-19
 Data types
 BLOB 1-7, 3-4, 7-19
 CLOB 1-7, 3-4, 3-5, 4-13
 etx_HiliteType 3-15, 4-6 to ??, 5-33, 5-36
 etx_ReturnType 1-15, 2-17, 4-4 to 4-7, 5-11
 IfxDocDesc 1-8, 4-8 to 4-12, 5-12, 7-20

IfxMRData 1-8, 4-13, 7-21
 list of supported 1-7
 LLD_Locator 4-9
 LVARCHAR 4-13, 7-20
 dbspaces, usage with DataBlade
 module 3-4, 7-16
 Deadlocks 7-22
 Default sbpace 7-5
 Default synonym list 5-9, 5-23
 Dependencies, software Intro-5
 DESC option to CREATE
 INDEX 1-9
 Determining character set
 usage 5-25
 Determining stopword list
 usage 5-27
 DISTINCT option to CREATE
 INDEX 1-9
 Document scoring 2-16, 5-13
 Documentation
 conventions Intro-5
 list of Intro-9
 Dropping stopword lists 3-7, 5-27
 Dropping synonym lists 3-8, 5-29
 Dropping user-defined character
 sets 1-21, 5-25

E

Environment variables Intro-6
 etx access method 1-7, 1-9, 2-3, 3-9
 etx index parameters
 CHAR_SET 1-10, 1-20, 3-10, 5-16,
 5-25, 6-3, 6-4, A-1
 INCLUDE_STOPWORDS 1-20,
 3-10, 5-8, 6-3, 6-7, 7-15
 PHRASE_SUPPORT 3-10, 3-11,
 5-10, 6-4, 6-9, 7-15
 specifying 1-10, 3-10, 6-3
 STOPWORD_LIST 1-19, 3-10,
 3-11, 5-8, 5-21, 6-4, 6-9, 7-15
 WORD_SUPPORT 3-10, 3-11, 5-9,
 6-4, 6-9
 etx indexes
 closing 5-14, 7-18
 creating 1-9, 3-10 to 3-12
 creating fragmented 3-12
 estimating size of 7-14

 sharing 5-14, 7-18
 tutorial for creating 3-9
 ETX VPCLASS 7-10
 EtxFilterTraceFile() routine 5-32
 etx_blob_ops operator class 1-8
 etx_char_ops operator class 1-8
 etx_clob_ops operator class 1-8
 etx_CloseIndex() routine 5-14, 7-18
 etx_contains() operator
 description of 1-5, 1-6, 3-13, 5-5
 reference 5-5
 return type of 5-11
 syntax for 5-5
 tuning parameters for. *See* Tuning
 parameters.
 usage for 5-7, 5-11
 used in a Boolean search 1-14
 used in a keyword search 2-4
 used in a pattern search 2-12,
 2-15, 5-13
 used in a phrase search with
 pattern matching 3-13
 used in a proximity search 2-10,
 3-14
 used in an approximate phrase
 search 2-9
 used in an exact phrase search 2-7
 used with a statement local
 variable (SLV) 2-17, 3-15, 5-13
 used with stopwords,
 ignored 1-20, 3-14
 used with synonym
 matching 1-17, 5-13
 etx_CreateCharSet() routine 1-21,
 3-8, 5-16, A-7
 etx_CreateStopWlst() routine 1-19,
 3-6, 5-20, 5-21, 6-9
 etx_CreateSynWlst() routine 1-16,
 3-7, 5-22, 5-24
 etx_doc_ops operator class 1-9
 etx_DropCharSet() routine 1-21,
 5-25
 etx_DropStopWlst() routine 3-7,
 5-27
 etx_DropSynWlst() routine 3-8,
 5-29
 etx_Filter() routine 5-30
 ETX_FILTER_FILE environment
 variable 7-10

etx_GetHilite() routine 1-22, 3-15,
 5-33
 etx_HiliteDoc() routine 1-23, 5-36
 etx_HiliteType data type
 description of 4-6 to ??
 reference for 4-6
 usage of 3-15, 5-33, 5-36
 etx_lvarc_ops operator class 1-8
 etx_mrd_ops operator class 1-9
 etx_Release() routine 5-38, 7-4
 etx_ReturnType data type
 reference for 4-4
 usage of 1-15, 2-17, 5-11
 etx_stopwords.txt, provided
 stopword list 1-19, 3-6, 5-21, 6-9
 etx_thesaurus default synonym
 list 5-9, 5-23
 etx_varc_ops operator class 1-9
 etx_ViewHilite() routine 1-23, 3-15,
 5-40
 etx_WordFreq() routine 5-42
 Exact phrase searches 2-7, 3-13,
 3-14, 5-10
 EXACT, value for
 WORD_SUPPORT index
 parameter 6-9
 Example
 closing an etx index 5-15
 complex queries 7-23, 7-25
 creating a fragmented etx
 index 3-12
 creating a table 1-7
 creating a user-defined character
 set 1-22, 3-8, 5-19
 creating an etx index 1-9, 1-10,
 3-10 to 3-12
 creating dbspaces 3-4
 creating sbspaces 3-4, 7-8
 creating stopword lists 1-19, 3-6,
 5-21
 creating synonym lists 1-16, 3-7,
 5-24
 determining character set
 usage 5-25
 determining stopword list
 usage 5-27
 dropping a user-defined character
 set 5-26
 highlighting 3-15, 5-35

ignoring stopwords 1-20, 3-14
 inserting into a table 4-11
 inserting into an IfxDocdesc
 column 4-11
 inserting into an IfxMRData
 column 4-14
 limiting a search 2-11, 2-20
 matching synonyms 1-17
 performing a Boolean search 1-14
 performing a keyword
 search 2-4, 2-5
 performing a pattern search 2-12,
 2-15, 5-13
 performing a proximity
 search 2-9, 2-10, 3-14
 performing a synonym
 search 1-17
 performing an approximate
 phrase search 2-9
 performing an exact phrase
 search 2-7, 3-13
 ranking the results of a
 search 2-17, 2-20, 4-5
 returning version information for
 the Excalibur Text Search
 DataBlade module 5-38
 returning version information for
 the Text Descriptor DataBlade
 module 5-39
 using statement local variables
 (SLVs) 2-17, 2-20, 4-4, 4-5
 viewing highlights 5-40
 Expression-based
 fragmentation 7-16

F

File-naming conventions Intro-6
 FileToBLOB() routine 7-5
 FileToCLOB() routine 3-5, 4-12
 FILLFACTOR option to CREATE
 INDEX 1-9
 FILTER index parameter 3-12, 6-5
 Filter server 7-10
 Filtering
 configuring database server
 for 7-9
 during index creation 6-5

handling errors 7-12
 logging messages 7-10
 overview of 1-9, 1-11
 specifying the ETX VPCLASS
 for 7-10
 tutorial for creating an index on a
 filtered column 3-12
 format, field of IfxDocDesc data
 type 4-9
 Fragmenting indexes 3-12, 7-16
 Fuzzy searches 2-11, 2-12, 2-14,
 3-13, 5-13

G

Generic mapping for user-defined
 character sets A-7

H

Highlighting
 description of 1-20, 1-22, 4-6, 5-33
 example of 1-20, 3-15
 tutorial 3-15
 viewing 5-40
 hilite_info field of
 etx_ReturnType 4-5
 hitlist, definition of 1-14

I

Icons
 Important Intro-7
 Tip Intro-7
 Warning Intro-7
 IfxDocDesc data type
 fields of 4-8
 overview of 1-8, 4-8
 recommendation 7-20
 reference for 4-8
 usage of 4-11, 7-20
 used as a clue 5-12
 IfxMRData data type
 definition of 1-8
 recommendation 7-21
 reference for 4-13
 usage of 4-13, 7-21

Important paragraphs, icon
 for Intro-7
 INCLUDE_STOPWORDS index
 parameter 1-20, 3-10, 5-8, 6-3,
 6-7, 7-15
 Index parameters. *See* etx index
 parameter.
 Indexes. *See* etx indexes.
 ISO value for CHAR_SET index
 parameter 1-21, A-3

K

Keyword searches 1-12, 1-13, 2-3,
 5-10

L

Large objects, customizing size
 for 7-7
 Limiting the number of rows a
 fuzzy search returns 2-19
 LLD_Locator data type
 4-9
 Loading data 7-21
 location field of IfxDocDesc 4-9
 LOCOPY() routine 3-5
 Logging messages from the filter
 server 7-10
 LVARCHAR data type
 recommendation 7-20
 reference for 4-13

M

MATCH_SYNONYM tuning
 parameter 1-17, 5-9
 MAXIMUM value for
 PHRASE_SUPPORT index
 parameter 5-10, 6-8
 MEDIUM value for
 PHRASE_SUPPORT index
 parameter 5-10, 6-8
 Multirepresentational data type. *See*
 IfxMRData data type.

N

NOAGE ONCONFIG
parameter 7-9
NONE value for
PHRASE_SUPPORT index
parameter 6-8
Nonfragmented indexes 3-10

O

oncheck utility 7-8
ONCONFIG parameter 7-5, 7-8, 7-9
oninit utility 7-10
On-line documentation Intro-11
onspaces utility 3-4, 7-5
Operator classes 1-8, 1-9
OVERLAP_ISO value for
CHAR_SET index
parameter 1-21, A-5

P

params field of IfxDocDesc 4-11
Pattern searches 2-11, 2-12, 3-13,
5-13
Pattern searches with phrase
matching 2-14
PATTERN value for
WORD_SUPPORT index
parameter 2-11, 5-9, 6-9
PATTERN_ALL tuning
parameter 1-18, 2-12, 2-13, 2-14,
3-13, 5-9
PATTERN_BASIC tuning
parameter 2-14, 5-10
PATTERN_SUBS tuning
parameter 2-13, 2-15, 5-10
PATTERN_TRANS tuning
parameter 2-13, 2-15, 5-10
Performance issues 7-8, 7-21
Phrase searches
approximate 2-9, 5-10
Boolean 2-6
exact 2-7, 3-13, 3-14, 5-10
general 2-7
pattern matching 2-14

PHRASE_APPROX value for
SEARCH_TYPE tuning
parameter 2-9, 2-14, 5-10
PHRASE_EXACT value for
SEARCH_TYPE tuning
parameter 2-7, 2-14, 5-10
PHRASE_SUPPORT index
parameter 3-10, 3-11, 5-10, 6-4,
6-8, 6-9, 7-15
Printed documentation, list
of Intro-9
Product overview 1-4
Proximity searches 2-9, 3-14, 5-10
PROX_SEARCH value for
SEARCH_TYPE tuning
parameter 2-10, 5-10

Q

Queries, refining 7-23

R

Ranking the results of a search 2-5,
2-14, 2-16, 5-13
RA_PAGES ONCONFIG
parameter 7-8
RA_THRESHOLD ONCONFIG
parameter 7-9
Reclaiming unused index
space 7-18
Refining etx indexes 7-14
Refining queries 7-23, 7-25
Related reading Intro-11
RESIDENT ONCONFIG
parameter 7-9
Retrieving DataBlade module
version information 7-4
Root word 1-16, 1-17, 5-23
Round-robin fragmentation 7-16
Routines
EtxFilterTraceFile() 5-32
etxViewHilite() 3-15
etx_CloseIndex() 5-14, 7-18
etx_contains(). See etx_contains()
operator.
etx_CreateCharSet() 1-21, 3-8,
5-16, A-7

etx_CreateStopWlst() 1-19, 3-6,
5-20, 5-21, 6-9
etx_CreateSynWlst() 1-16, 3-7,
5-22, 5-24
etx_DropCharSet() 1-21, 5-25
etx_DropStopWlst() 3-7, 5-27
etx_DropSynWlst() 3-8, 5-29
etx_Filter() 5-30
etx_GetHilite() 3-15, 5-33
etx_getHilite() 1-22
etx_HiliteDoc() 1-23, 5-36
etx_Release() 5-38, 7-4
etx_ViewHilite() 1-23, 5-40
etx_WordFreq() 5-42
FileToCLOB() 3-5
LOCopy() 3-5
txt_Release() 5-39
Row() constructor 1-13, 4-15

S

SBSPACENAME ONCONFIG
parameter 7-5
sbspaces
buffering 7-6
configuring 7-5, 7-6
creating 3-4, 7-5
default 7-5
logging 7-6
usage with DataBlade
module 1-9, 1-16, 1-19, 3-4,
5-20, 5-22, 7-5
score field of etx_ReturnType 2-17,
2-20, 4-4
Scoring
document 2-5, 2-16, 5-13
documents 2-16
word 2-11
Searches
approximate phrase 2-9, 5-10
Boolean 1-14, 2-5, 5-10
exact phrase 2-7, 3-13, 3-14, 5-10
keyword 1-13, 2-3, 5-10
pattern 2-11, 2-12, 3-13, 5-13
phrase matching with
pattern 2-14
proximity 2-9, 3-14, 5-10
simple 1-12, 5-12

SEARCH_TYPE tuning
 parameter 2-4, 2-7, 2-9, 2-14,
 3-14, 5-10
 Setting ONCONFIG tuning
 parameters 7-8
 Sharing etx indexes 7-18
 Software dependencies Intro-5
 Statement local variable (SLV) 1-15,
 2-17, 4-4, 5-13
 Stopword lists
 creating 3-6, 5-20
 discussion of 1-18
 dropping 3-7, 5-27
 STOPWORD_LIST index
 parameter 1-19, 3-10, 3-11, 5-8,
 5-21, 6-4, 6-9, 7-15
 Storing data 3-4, 7-19
 Substitution 2-11, 2-12, 2-13, 5-10
 Synonym lists
 creating 3-7, 5-22
 default 5-9, 5-23
 discussion of 1-16, 5-22
 dropping 3-8, 5-29
 Syntax conventions Intro-8

T

Text Descriptor DataBlade
 module 4-3, 4-8, 4-13, 5-39, 7-4
 Text searches
 illustration of 1-6
 tutorial 3-13
 types of 2-3
 Tip icons Intro-7
 Tracing 7-11
 Transposition 2-11, 2-12, 2-13, 5-10
 Tuning parameters
 CONSIDER_STOPWORDS 1-20,
 3-14, 5-8, 6-7
 description of 2-3
 MATCH_SYNONYM 1-17, 5-9
 PATTERN_ALL 1-18, 2-12, 2-13,
 2-14, 3-13, 5-9
 PATTERN_BASIC 2-14, 5-10
 PATTERN_SUBS 2-13, 2-15, 5-10
 PATTERN_TRANS 2-13, 2-15,
 5-10

SEARCH_TYPE 2-4, 2-7, 2-9,
 2-14, 3-14, 5-10
 TxtSetTrace() procedure 7-12
 txt_Release() routine 5-39
 Types of text searches 2-3
 Typographical conventions Intro-6

U

UNIQUE option to CREATE
 INDEX 1-9
 User-defined character sets
 creating 1-20, 3-8, 5-16, A-7
 creating operating system
 file 5-17
 dropping 5-25
 generic mapping for A-7
 overview of 1-21
 tutorial 3-8
 using 1-20, 6-5
 Utility
 oncheck 7-8
 onspaces 3-4, 7-5

V

vec_offset field of
 etx_HiliteType 3-15, 4-6
 version field of IfxDocDesc 4-9
 Version, of the Excalibur Text
 Search DataBlade module 5-38,
 7-4
 Version, of the Text Descriptor
 DataBlade module 5-39
 viewer_doc field of
 etx_HiliteType 3-15, 4-7
 Viewing highlights 5-40

W

Warning icons Intro-7
 Word lists
 stopword 1-18, 3-6, 5-20, 5-27
 synonym 1-16, 3-7, 5-22, 5-29
 tutorial for creating 3-6
 Word scoring 2-11

WORD value for SEARCH_TYPE
 tuning parameter 2-4, 5-10
 WORD_SUPPORT index
 parameter 3-10, 3-11, 5-9, 6-4,
 6-9

