

Verity Text Search DataBlade Module

User's Guide

Version 1.1
January 1999
Part No. 000-5030

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™; C-ISAM®; Cyber Planet™; Data Director™; DataBlade®; 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™; 4GL for ToolBus™; If you can imagine it, you can manage it™; Illustra®; INFORMIX®, Informix Data Warehouse Solutions... Turning Data Into Business Advantage™; INFORMIX®-Enterprise Gateway with DRDA®, INFORMIX®-4GL; InformixLink®, MetaCube®; NewEra™; ON-Bar™; OnLine Dynamic Server™; OnLine for NetWare®, OnLine/Secure Dynamic Server™; OpenCase®, Regency Support®, Solution Design Labs™; Solution Design Program™; SuperView®, Universal Web Connect™; ViewPoint®. The Informix logo is registered with the United States Patent and Trademark Office.

The following are worldwide trademarks of the indicated owners or their subsidiaries, registered in the United States of America as indicated by “®,” and in numerous other countries worldwide:

Verity, Inc: Verity®

All other marks or symbols are registered trademarks or trademarks of their respective owners.

Documentation Team: Rosanne Thornhill, Inge Halilovic, Oakland Editing and Production, Verity DataBlade module development team

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 of Defense use, with rights as restricted by vendor's standard license, unless superseded by a negotiated vendor license, as prescribed in DFARS 227.7202.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	4
Documentation Conventions	5
Typographical Conventions	5
Icon Conventions	6
Syntax Conventions	8
Software Dependencies	8
Additional Documentation	8
Printed Documentation	9
On-Line Documentation	11
Documentation from Verity	11
Related Reading	12
Informix Welcomes Your Comments	13

Chapter 1

DataBlade Module Overview

In This Chapter	1-3
Introduction to the DataBlade Module	1-3
Data Types Used for Text Searches	1-5
Auxiliary Results	1-5
Highlighting Search Terms	1-6
Expressive Searches	1-6
Filtering	1-6
Multilingual Support	1-7
Index Searches Versus Nonindex Searches	1-8
Index Customization	1-8
Verity Text Search DataBlade Module Components	1-8
The vts Access Method	1-9

Operator Classes	1-10
The vts_contains Operator	1-10
Contexts and Style Files	1-10

Chapter 2 **Using the DataBlade Module**

In This Chapter	2-3
Installing the DataBlade Module	2-4
Creating sbspaces	2-4
Creating the User-Defined Virtual Processor	2-5
Registering the DataBlade Module	2-6
Choosing Data Types	2-6
Data Type Guidelines	2-7
Creating Row and Distinct Data Types	2-7
Creating and Associating Contexts	2-9
Customizing Style Files	2-10
Adding Style Files to the Database.	2-10
Creating a Context	2-11
Associating a Context with a Row or Distinct Data Type	2-12
Creating a Table with Text Search Columns	2-12
Creating the vts Index	2-13
Creating an Index on a CLOB Data Type Column	2-13
Creating an Index on a Row Data Type Column	2-14
Creating a Fragmented Index	2-14
Multicolumn Indexing	2-15
Populating the Table with Text Data	2-15
Inserting a File into a CLOB Column	2-15
Performance Tips.	2-17
Performing Text Searches	2-17
Simple Queries	2-17
Searching in a Field of a Row Type Column	2-18
Ordering Results by Score.	2-19
Using the THESAURUS Operator	2-20
Using the PHRASE Operator.	2-20
Using the NEAR/ <i>n</i> Operator	2-21
Using the SOUNDEX Operator	2-21
Using the CASE Modifier	2-22
Using the TOPN and MANY Parameters	2-23
Clustering and Summarization	2-23
Creating and Populating the Poems Table	2-24
Simple Queries Using Verity Features.	2-27

Using a Wildcard	2-28
Retrieving Information in the Statement Local Variable	2-28
Highlighting the Results	2-32
Highlighting ASCII Documents	2-33
Highlighting HTML Documents	2-33
Working with Multilingual Databases	2-36

Chapter 3 DataBlade Data Types

In This Chapter	3-3
IfxDocDesc Data Type	3-4
LLD_Locator Data Type	3-7
IfxVtsReturnType Data Type	3-9
IfxVtsSummaryType Data Type	3-12
IfxVtsClusterType Data Type	3-14
Locale-Specific Data Types	3-16

Chapter 4 DataBlade Routines

In This Chapter	4-3
Vts_AlterContext() Procedure	4-4
Vts_AssocContext() Procedure	4-6
vts_contains Operator	4-9
Vts_CreateContext() Procedure	4-14
Vts_CreateLocale() Procedure	4-16
Vts_CreateType() Procedure	4-19
Vts_DisassoContext() Procedure	4-21
Vts_DropContext() Procedure	4-23
Vts_DropLocale() Procedure	4-25
Vts_DropStyle() Procedure	4-27
Vts_DropType() Procedure	4-29
Vts_GetHighlight() Function	4-31
Vts_QueryContext() Function	4-33
Vts_ReadStyle() Procedure	4-35
Vts_Release() Function	4-38
Vts_WriteStyle() Procedure	4-39

Chapter 5 SQL Syntax Usage

In This Chapter	5-3
ALTER INDEX Statement	5-4
CREATE INDEX Statement	5-5
CREATE TABLE Statement	5-10
DROP INDEX Statement	5-11
SELECT Statement	5-12

Chapter 6	Verity Style Files	
	In This Chapter	6-3
	Overview of Style Files	6-3
	Style Files That You Can Create	6-5
	Style Files You Can Modify	6-6
	Style Files You Cannot Modify	6-7
	style.go	6-8
	style.lex	6-9
	style.prm	6-17
	style.sfl	6-21
	style.stp	6-22
Appendix A	Verity Operators and Modifiers	
Appendix B	Regular Expressions	
Appendix C	DataBlade Tables	
Appendix D	Filtered Formats	
	Glossary	
	Index	

Introduction

In This Introduction	3
About This Manual.	3
Organization of This Manual	3
Types of Users	4
Documentation Conventions	5
Typographical Conventions	5
Icon Conventions	6
Comment Icons	7
Data Type Icons	7
Syntax Conventions	8
Software Dependencies	8
Additional Documentation	8
Printed Documentation	9
On-Line Documentation.	11
Documentation from Verity	11
Related Reading	12
Informix Welcomes Your Comments.	13

In This Introduction

This chapter introduces the *Verity Text Search DataBlade Module User's Guide*. Read this chapter for an overview of the information provided in this manual and for an understanding of the conventions used throughout.

About This Manual

This manual contains information to assist you in performing text search and retrieval operations using the Verity Text Search DataBlade module with the object-relational database management system. A DataBlade module adds custom data types and supporting routines to Informix Dynamic Server with Universal Data Option. The Verity Text Search DataBlade module also adds a new secondary access method and an operator that enables you to perform text searches using SQL.

This section discusses the organization and format of the manual and describes the intended audience.

Organization of This Manual

The *Verity Text Search DataBlade Module User's Guide* contains chapters on using the Verity Text Search DataBlade module and chapters with reference information on module-specific data types, routines, tables, and SQL syntax.

This manual consists of the following chapters:

- This introduction provides an overview of the manual, describes its documentation conventions and style, and lists background materials.

- [Chapter 1, “DataBlade Module Overview,”](#) describes basic concepts you need to understand to take full advantage of the Verity Text Search DataBlade module.
Concepts related to the search techniques supported by Verity products are discussed in the documentation provided by Verity, Inc.
- [Chapter 2, “Using the DataBlade Module,”](#) provides instructions for creating and populating tables, creating indexes, and performing searches using the Verity Text Search DataBlade module. This chapter includes a section on multilingual tables.
- [Chapter 3, “DataBlade Data Types,”](#) provides reference information about the data types specific to the module.
- [Chapter 4, “DataBlade Routines,”](#) provides reference information about the functions and procedures that come with the Verity Text Search DataBlade module.
- [Chapter 5, “SQL Syntax Usage,”](#) discusses module-specific uses of SQL syntax statements.
- [Chapter 6, “Verity Style Files,”](#) provides instructions for modifying the most commonly used style files.
- [Appendix A, “Verity Operators and Modifiers,”](#) provides reference information on Verity operators and modifiers that you can use in text searches.
- [Appendix B, “Regular Expressions,”](#) lists the expressions that are valid in Verity style files.
- [Appendix C, “DataBlade Tables,”](#) describes the **Vts_Contexts** and **Vts_Context_Type** tables, which record contexts and their associations with data types.
- [Appendix D, “Filtered Formats,”](#) lists the document formats supported by the Verity Text Search DataBlade module universal filter.

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

Types of Users

This manual is intended for programmers who want to build applications that use the Verity text search engine to perform text searches.

Database administrators and system administrators will find [Chapter 4](#), “[DataBlade Routines](#),” and the appendixes particularly useful.

Users who want to perform SQL searches on documents should read [Chapter 2](#), “[Using the DataBlade Module](#),” and [Chapter 5](#), “[SQL Syntax Usage](#).”

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:

- Typographical conventions
- Icon conventions
- Syntax conventions for routines

Typographical Conventions

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

Convention	Meaning
KEYWORD	All keywords appear in uppercase letters in a serif font.
<i>italics</i>	Within text, new terms and emphasized words appear in italics. Within syntax diagrams, values that you are to specify appear in italics.

(1 of 2)



Convention	Meaning
boldface	Identifiers (names of classes, objects, constants, events, functions, program variables, forms, labels, and reports), environment variables, database names, filenames, table names, column names, icons, menu items, command names, and other similar terms appear in boldface.
<code>monospace</code>	Information that you enter on the computer and information that the product displays on your screen appear in a monospace typeface. Examples are always shown in monospace font.
◆	This symbol indicates the end of data-type-specific information. In this manual, sections that describe the CLOB and IfxDocDesc data types are set apart by an icon in the margin and finish with this symbol.

(2 of 2)




***Tip:** The text and many of the examples in this manual show function and data type names in mixed case (uppercase and lowercase letters). Because the Informix Dynamic Server database management system is case insensitive, you do not need to enter function names exactly as shown: you can use uppercase letters, lowercase letters, or any combination of the two.*

Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.



Comment Icons

Comment icons identify warnings, important notes, or tips. This information is always displayed 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

Data Type Icons

Most examples in this book use the CLOB data type for columns. You can use the BLOB data type instead of the CLOB data type with minor changes to the examples. You can also use the IfxDocDesc data type, which involves significant changes to the examples. The following table tells you how this manual presents information about the BLOB and IfxDocDesc data types.

Icon	Description
	This icon marks sections that describe the changes you must make to examples to use the BLOB data type instead of the CLOB data type.
	This icon marks sections that describe the changes you must make to examples to use the IfxDocDesc data type instead of the CLOB data type.

A ♦ symbol indicates the end of a data-type-specific explanation.

Syntax Conventions

This guide uses the following conventions to specify DataBlade syntax for routines and SQL statements:

- Square brackets ([]) surround optional items.
- Curly brackets ({ }) surround items that can be repeated.
- A vertical line (|) separates alternatives.
- Variables (parameter names) are italicized; arguments that must be specified as shown are not italicized.

These syntax conventions are used in [Chapter 4, “DataBlade Routines,”](#) and [Chapter 5, “SQL Syntax Usage.”](#)

Software Dependencies

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

- Informix Dynamic Server with Universal Data Option
- Informix Large Object Locator DataBlade module (shipped with Informix Dynamic Server with Universal Data Option)
- Text Descriptor DataBlade module (shipped with the Verity Text Search DataBlade module)

See your release notes for the correct version numbers.

Additional Documentation

This documentation set includes printed manuals and PDF and HTML files included on the product media.

This section describes the following parts of the documentation set:

- Printed documentation provided by Informix
- On-line documentation

- Vendor-specific documentation provided by Verity
- Related reading

Printed Documentation

The *Verity Text Search DataBlade Module User's Guide* contains complete documentation for the Verity Text Search DataBlade module. This includes documentation for the Text Descriptor DataBlade module.

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

- [*DataBlade Module Installation and Registration Guide*](#) describes how to install DataBlade modules and register them in the database using the BladeManager application.
- [*Installation Guide for Informix Dynamic Server on UNIX*](#) and [*Installation Guide for Informix Dynamic Server on Windows NT*](#) provide instructions for installing Informix Dynamic Server.
- The [*Administrator's Guide for Informix Dynamic Server*](#) provides information about how to configure your server; it also explains how the server interacts with DataBlade modules. Before you can use the Verity Text Search DataBlade module, you must install and configure Informix Dynamic Server with Universal Data Option.
- [*Informix Large Object Locator DataBlade Module User's Guide*](#) explains how the Verity Text Search DataBlade module uses data types and functions defined by the LOB Locator DataBlade module. Refer to this manual for more information on these data types and functions.
- [*DataBlade Developers Kit User's Guide*](#) describes how to develop your own DataBlade module using the BladeSmith, BladePack, and BladeManager applications.
- [*Informix Guide to SQL: Tutorial*](#) provides a tutorial on SQL as it is implemented by Informix products. It also describes the fundamental ideas and terminology for planning and implementing a relational database.
- [*Informix Guide to SQL: Syntax*](#) is a reference manual for SQL statements used with the Verity Text Search DataBlade module. (Module-specific variations on this syntax are described in [Chapter 5, "SQL Syntax Usage,"](#) of this manual.)

- A companion volume to the *Tutorial* and *Syntax*, 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 column data types that Informix database servers support.
- [Getting Started with Informix Dynamic Server](#) describes the architecture and features of the database server and offers instructions for installing and configuring your database server.
- To learn more about creating your own data types, see the *Extending Informix Dynamic Server: Data Types*.
- The [DataBlade API Programmer's Manual](#) guide describes the application programming interface for the server. You can use this API to develop client and server applications to access data stored in a server database.
- *Informix Error Messages* is useful if you wish to use the **find** utility to look up your error messages on-line.

Consult your Informix representative to find out what recent documentation is available for Informix Dynamic Server with Universal Data Option or visit the Answers OnLine Web page, <http://www.informix.com/answers>.

On-Line Documentation

In addition to the Informix set of manuals, the following on-line files, located in the `$INFORMIXDIR/extend/VTs.rel_no` directory, supplement the information in this manual (*rel_no* represents the current version number of this module.)

On-Line File or Directory	Purpose
VTsREL.TXT	These release notes describe feature differences from earlier versions of Informix products and indicate how these differences might affect current products. This file also contains information about any known problems and their workarounds.
VTsDOC.TXT	These documentation notes describe features not covered in the manuals or modified since publication.
<code>\$INFORMIXDIR/extend/VTs.rel_no/vdkhome/doc</code>	The latest Verity documentation is available in this directory.
<code>\$INFORMIXDIR/extend/VTs.rel_no/docsamples/movies</code>	The data files used in the movie examples in this manual are in this directory.
<code>\$INFORMIXDIR/extend/VTs.rel_no/docsamples/poems</code>	The data files used in the poem examples in this manual are in this directory.

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

Documentation from Verity

Complete Verity documentation in HTML format is included on the Verity Text Search DataBlade module media.

You can obtain hard-copy Verity documentation by ordering the manuals from Verity, Inc. You can see essential documentation at Verity's Web site at <http://www.verity.com>. To find information, use Verity's Web search facility and type in the name of the specific query element you want to learn about. Search for operators or modifiers to get an overview.

The *SEARCH'97 Introduction to Topics Guide* provides an overview of the Verity search engine. This manual also includes a reference of all Verity operators and modifiers.

For more information about the Verity search engine, refer to the following manuals in the *Verity Developer's Kit* (Version 2.4):

- *SEARCH'97 Developer's Kit Getting Started Guide* introduces the features of the *Search'97 Developer's Kit*.
- *SEARCH'97 Developer's Kit API Reference Guide* introduces the Verity architecture and explains how to use the Verity Developer's Kit (VDK). The appendixes provide details about how VDK uses SQL, describe operators and modifiers, and explain how to use query expressions to retrieve data.
- *SEARCH'97 Developer's Kit Advanced Features Guide* explains the configuration options available with VDK. This guide contains detailed descriptions of style files.
- *Verity Introduction to Collections* describes collections used by VDK. It contains useful information about customizing indexes using style files.

Related Reading

For additional technical information on database management, see *An Introduction to Database Systems*, by C. J. Date (Addison-Wesley Publishing, 1995).

To learn more about the SQL language, read 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 use the Verity Text Search DataBlade module, you must 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.

DataBlade Module Overview

In This Chapter	1-3
Introduction to the DataBlade Module	1-3
Data Types Used for Text Searches	1-5
Auxiliary Results	1-5
Highlighting Search Terms	1-6
Expressive Searches	1-6
Filtering	1-6
Multilingual Support	1-7
Setting Up a Default Language	1-7
Using Multiple Languages in a Single Table	1-7
Index Searches Versus Nonindex Searches	1-8
Index Customization	1-8
Verity Text Search DataBlade Module Components.	1-8
The vts Access Method	1-9
Operator Classes	1-10
The vts_contains Operator	1-10
Contexts and Style Files	1-10
Custom Style Files	1-11
Contexts Associated with Row or Distinct Data Types	1-11

In This Chapter

This chapter introduces the Verity Text Search DataBlade module and discusses concepts you need to understand to use this product. It provides the following sections:

- [“Introduction to the DataBlade Module,”](#) next
This section describes what you can do with the Verity Text Search DataBlade module.
- [“Verity Text Search DataBlade Module Components”](#) on page 1-8
This section describes the objects that the Verity Text Search DataBlade module adds to Informix Dynamic Server with Universal Data Option, and how these objects work.

Introduction to the DataBlade Module

The Verity Text Search DataBlade module enables you to use the Verity search engine to index and search text data stored in a variety of languages and formats in non-ANSI databases.

Traditional text searches are often very slow, depending on the amount of text to be searched. The traditional B-tree access method does not effectively index text, resulting in full table scans when you search columns containing text.

The Verity Text Search DataBlade module solves this problem by dynamically linking in the Verity 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.

When you execute searches with the Verity Text Search DataBlade module, instead of using LIKE or MATCHES in the SELECT statement, you use an operator called **vts_contains**. The **vts_contains** operator 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 Verity Text Search DataBlade module you create a specialized secondary index called **vts** that indexes the text data, resulting in much faster searches.

The Verity Text Search DataBlade module provides you with the following feature choices:

- Store your documents as the data type that best fits your needs
For more information, see [“Data Types Used for Text Searches,”](#) next.
- Use auxiliary search results: ranking, summarization, and clustering
For more information, see [“Auxiliary Results”](#) on page 1-5.
- Highlight instances of your search string in the returned documents
For more information, see [“Highlighting Search Terms”](#) on page 1-6.
- Execute expressive searches using Verity operators, modifiers, and tuning parameters
For more information, see [“Expressive Searches”](#) on page 1-6.
- Perform filtering to convert documents in proprietary formats, such as Microsoft Word and PDF, to ASCII before they are indexed
For more information, see [“Filtering”](#) on page 1-6.
- Store and search documents in multiple languages
For more information, see [“Multilingual Support”](#) on page 1-7.
- Choose whether to perform an index or a nonindex search
For more information, see [“Index Searches Versus Nonindex Searches”](#) on page 1-7.
- Create custom index styles and use fragmented indexes
For more information, see [“Index Customization”](#) on page 1-8.

Data Types Used for Text Searches

You can create **vts** indexes on the following *indexable data types*:

- CHAR
- VARCHAR
- LVARCHAR
- BLOB
- CLOB
- IfxDocDesc
- Named row types with at least one field that has a data type from this list
- Distinct types built on one of the data types listed here

In general, use the BLOB data type for large binary documents, CLOB for large ASCII documents, and CHAR, VARCHAR, and LVARCHAR for shorter ASCII documents. Use IfxDocDesc for documents stored in external files. Use a row data type if you want to emulate multicolumn indexing (see [“Index Customization” on page 1-8](#)). Use a row data type or a distinct data type if you want to customize your search styles (see [“Contexts and Style Files” on page 1-10](#)).

For more information on choosing a data type, see [“Choosing Data Types” on page 2-6](#).

Auxiliary Results

The Verity Text Search DataBlade module returns auxiliary information about each document that matches the search criteria. You can choose to retrieve none, some, or all of this information.

You can request the following information in your query:

- **The *score*.** A value between 0 and 100 that indicates how closely the returned document matches the search criteria. A score of 100 indicates a perfect match; a score of 0 indicates a very weak match.
You can have results returned in order of score using the ORDER BY clause in the SELECT statement. To list the best matches first, specify the DESC keyword.

- **Summary information.** A set of keywords or sentences that summarize the selected document.
- **Cluster information.** A list of documents with content similar to that of the selected document.

You retrieve result information in the same SELECT statement that specifies the query. For more information, see [“Clustering and Summarization” on page 2-23.](#)

Highlighting Search Terms

After you search for a document, you can highlight the specified search string in the document. All occurrences of the specified string are distinguished by the specified highlight format when the document is displayed or printed.

You can use highlighting on documents stored in ASCII or HTML format.

For more information, see [“Highlighting the Results” on page 2-32.](#)

Expressive Searches

Any search option provided by the Verity query language in the Verity Developer's Kit is available to you with the Verity Text Search DataBlade module. With the Verity Text Search DataBlade module, you can search for words and phrases, use expressions as search criteria, do proximity matching, and search for synonyms. Descriptions of valid Verity operators and modifiers are in [Appendix A, “Verity Operators and Modifiers.”](#) For additional detail, see the Verity documentation.

Filtering

When you store your documents in a column of data type IfxDocDesc, CLOB, or BLOB, you usually do not need to convert them manually from their proprietary format into ASCII. The Verity Text Search DataBlade module uses a Verity universal filter that recognizes a number of document formats and converts them into ASCII format whenever it indexes the document. For a list of the supported formats, see [Appendix D, “Filtered Formats.”](#)

The filter first checks the header information in your document. If the filter does not recognize the format, it returns an error. If your document format is not supported, you must convert your document to ASCII before you can index it.

Multilingual Support

The Verity Text Search DataBlade module supports the ISO 8859-1 character set. ISO 8859-1 is a set of standardized eight-bit, single-byte, coded graphic characters. With this version of the Verity Text Search DataBlade module, you can insert, index, and search documents in the following languages:

- Dutch
- English
- French
- German
- Italian
- Spanish
- Swedish

Setting Up a Default Language

When you register the Verity Text Search DataBlade module for the first time, the registration script automatically sets a default locale for the DataBlade module, which is the same as the default locale for your database. You set the database locale using the environment variable **DB_LOCALE**. See the [Informix Guide to GLS Functionality](#) for more information about the **DB_LOCALE** environment variable.

Using Multiple Languages in a Single Table

You can use documents written in any supported language in a single table. Create a column for each language and insert your documents into the column with the language that corresponds to the documents' language. Instructions for creating multilingual tables are in [Chapter 2, "Using the DataBlade Module."](#)

Index Searches Versus Nonindex Searches

When you execute a query on an indexed column, the query optimizer determines whether to search the index or conduct a table scan. The optimizer acts on cost information provided by the DataBlade module at the time that the query is run. With the Verity Text Search DataBlade module, if an index exists, an index search is usually performed because it is generally faster.

You can create indexes only on columns that have indexable data types; see [“Data Types Used for Text Searches” on page 1-4](#).

When you create an index on a column that is a named row type, only those fields in the row type that have indexable data types are indexed.

Index Customization

You can create a fragmented index using the Verity Text Search DataBlade module. Fragmentation allows you to store the index in more than one subpage, based on an expression.

You can index multiple columns by including them in a row data type and then creating an index on the row data type. When you create an index on a row data type, all fields that have an indexable data type are indexed.

For more information, see [“Creating the vts Index” on page 2-13](#).

Verity Text Search DataBlade Module Components

The Verity Text Search DataBlade module adds the following components to the database server:

- The access method **vts**, to create an index that uses the Verity search engine
For more information, see [“The vts Access Method,” next](#).
- One operator class for each applicable data type, to specify valid syntax options
For more information, see [“Operator Classes” on page 1-9](#).

- The search operator **vts_contains**, to signal the database server to use the **vts** index and to specify query parameters
For more information, see [“The vts_contains Operator” on page 1-10](#).
- Contexts and style files to govern the behavior of indexing and search operations
For more information, see [“Contexts and Style Files” on page 1-10](#).
- Routines to manage contexts, highlight results, create locale-specific data types, and store documents as smart large objects
See [Chapter 4, “DataBlade Routines,”](#) for the syntax and descriptions of these routines.
- Data types to store documents, query results, and auxiliary information
For more information, see [Chapter 3, “DataBlade Data Types.”](#)
- DataBlade system catalog tables to store information about contexts and their associations with data types
For more information, see [Appendix C, “DataBlade Tables.”](#)

The vts Access Method

When you execute a query on an indexed column, the server determines how to access the index. The routines used to access the index are determined by the access method.

The server uses *secondary access methods* to manage indexes. Secondary access methods are sets of routines that execute in the server process. These routines create, drop, insert, delete, update, and scan indexes.

The Verity Text Search DataBlade module access method is **vts**. You must specify **vts** in the USING clause of the CREATE INDEX statement whenever you build an index on a column to be searched using Verity Text Search DataBlade module functions.

For more information on creating indexes, see [“CREATE INDEX Statement” on page 5-5](#).

Operator Classes

A secondary access method can have its own operators. These operators are grouped into *operator classes* and associated with specific data types. An operator class specifies the valid syntax options for the data types with which it is associated. When you create an index on a column, you specify the name of the column and the operator class associated with the data type of the column. For a list of **vts** operator classes, see [“Operator Classes for Supported Data Types” on page 5-7](#).

The vts_contains Operator

You use the **vts_contains** operator in SQL statements to perform searches. The **vts_contains** operator signals to the database server to use the **vts** access method to perform the search. The **vts_contains** operator allows you to specify the following information about your search:

- The column to search
- The search criteria, using Verity operators; for example, whether to search for a word or a phrase, the proximity of words in a phrase, wildcards, or synonyms
- Information describing the results: the score of the returned documents, cluster information about similar documents, and summary information about returned documents

For examples using **vts_contains**, see [Chapter 2, “Using the DataBlade Module.”](#) For the syntax of **vts_contains**, see [“vts_contains Operator” on page 4-9](#).

Contexts and Style Files

A *context* determines both the characteristics of a **vts** index at the time that it is created and the behavior of searches that use that index. The characteristics of a context are determined by a set of style files.

A context associates Verity style files with a data type. Verity *style files* provide information about an index and determine its search characteristics. For example, a style file might include a list of stopwords or a list of nonalphanumeric characters that are valid in searches.

The Verity Text Search DataBlade module provides a default context that uses default style files. You can create custom contexts with custom style files for row data types and distinct data types that you create. The following data types always use the default context: CHAR, VARCHAR, LVARCHAR, CLOB, BLOB, and IfxDocDesc.

See [Chapter 6, “Verity Style Files,”](#) for information about style files used by the Verity Text Search DataBlade module. For additional details about style files and their contents, see the *SEARCH'97 Developer's Kit Advanced Features Guide*, which is included on the CD-ROM with this product.

Custom Style Files

You can create custom style files for row data types and distinct data types. Then you can create a custom context based on the style file and subsequently associate that context with a row or distinct data type. The routines you use to accomplish this are documented in [Chapter 4, “DataBlade Routines.”](#)

Use the default style files to learn how the default context behaves and as a basis for custom contexts. [Chapter 2, “Using the DataBlade Module,”](#) describes the process. You can find basic information about the most commonly used style files in [Chapter 6, “Verity Style Files.”](#)

Contexts Associated with Row or Distinct Data Types

When you associate a context with a row or distinct data type, the context is associated with all columns of that data type. If you create an index on one of those columns, the index is associated with the same context as the column. A context can be associated with more than one data type; however, a data type can only be associated with one context.

Using the DataBlade Module

In This Chapter	2-3
Installing the DataBlade Module	2-4
Creating sbspaces	2-4
Creating the User-Defined Virtual Processor	2-5
Registering the DataBlade Module	2-6
Choosing Data Types	2-6
Data Type Guidelines	2-7
Creating Row and Distinct Data Types.	2-7
Creating and Associating Contexts	2-9
Customizing Style Files	2-10
Adding Style Files to the Database	2-10
Creating a Context.	2-11
Associating a Context with a Row or Distinct Data Type	2-12
Creating a Table with Text Search Columns	2-12
Creating the vts Index	2-13
Creating an Index on a CLOB Data Type Column	2-13
Creating an Index on a Row Data Type Column	2-14
Creating a Fragmented Index	2-14
Multicolumn Indexing	2-15
Populating the Table with Text Data	2-15
Inserting a File into a CLOB Column	2-15
Performance Tips	2-17

Performing Text Searches	2-17
Simple Queries	2-17
Searching in a Field of a Row Type Column	2-18
Ordering Results by Score	2-19
Using the THESAURUS Operator	2-20
Using the PHRASE Operator	2-20
Using the NEAR/ <i>n</i> Operator	2-21
Using the SOUNDEX Operator	2-21
Using the CASE Modifier	2-22
Using the TOPN and MANY Parameters	2-23
Clustering and Summarization	2-23
Creating and Populating the Poems Table.	2-24
Simple Queries Using Verity Features	2-27
Using a Wildcard	2-28
Retrieving Information in the Statement Local Variable	2-28
Requesting SLV Information	2-28
Requesting Summary Information	2-29
Requesting Cluster Information	2-30
Highlighting the Results	2-32
Highlighting ASCII Documents	2-33
Highlighting HTML Documents	2-33
Working with Multilingual Databases	2-36
Creating a VTS Locale	2-36
Creating a Locale-Specific Data Type	2-37
Creating a Table	2-38
Indexing Columns	2-39
Inserting Data into the Table	2-39
Querying the Table	2-41
Cleaning Up	2-41

In This Chapter

This chapter provides instructions for setting up and performing text searches using the Verity Text Search DataBlade module. Before you read this chapter, read [Chapter 1, “DataBlade Module Overview,”](#) to become familiar with the text searching concepts and the components of the DataBlade module.

Complete the steps described in the following table to set up your environment, store documents, and perform index searches on those documents.

Description	Optional or Mandatory?
Installing the DataBlade Module	Mandatory
Creating sbspaces	Mandatory
Creating the User-Defined Virtual Processor	Mandatory
Registering the DataBlade Module	Mandatory
Choosing Data Types	Mandatory
Creating and Associating Contexts	Optional
Creating a Table with Text Search Columns	Mandatory
Creating the vts Index	Mandatory
Populating the Table with Text Data	Mandatory
Performing Text Searches	Mandatory

(1 of 2)

Description	Optional or Mandatory?
Clustering and Summarization	Optional
Highlighting the Results	Optional
Working with Multilingual Databases	Optional

(2 of 2)

Installing the DataBlade Module

The instructions for installing the Verity Text Search DataBlade module are included on the *Read Me First* sheet provided with the product media. After you install the product, read the release notes and documentation notes. See [“On-Line Documentation” on page 11](#) for more information.

Creating sbspaces

This section shows you how to create sbspaces in which to store your **vts** indexes.

Every **vts** index is stored in a smart large object space (sbspace) that you specify when you create the index. You can use the default sbspace or create your own. The default sbspace is specified by the SBSPACENAME parameter in the ONCONFIG file.

Because all **vts** indexes are stored in sbspaces, they are always detached; the table that contains the indexed column is always stored in a dbspace.

If you are using a BLOB or CLOB data type for one of your text columns, you must store that column’s data in an sbspace.



Important: All sbspaces used to contain **vts** indexes must have logging turned on. By default, sbspace logging is turned off, so you must explicitly request logging when you create an sbspace. If your sbspace does not have logging, you must replace it with a new sbspace that does. See the [“Administrator’s Guide”](#) for information about sbspaces.

When you use the Verity Text Search DataBlade module, certain large objects are created in the default sbpace for internal use by the File System Emulation (FSE) subsystem. These large objects are always logged regardless of the default setting of the default sbpace. The default sbpace must already exist before you can register the Verity Text Search DataBlade module.

You use the **onspaces** utility to create sbspaces. For example, you can create two sbspaces, named **sbsp1** and **sbsp2**, that have initial offsets of 0 and a size of 100 megabytes.

The following example creates **sbsp1**:

```
onspaces -c -S sbsp1 -g 2 -p /dev/sbs/space1 -o 0 -s 100000
-Df "LOGGING=ON";
```

The following example creates **sbsp2**:

```
onspaces -c -S sbsp2 -g 2 -p /dev/sbs/space2 -o 0 -s 100000
-Df "LOGGING=ON";
```

To create the default sbpace, first set the SBSPACENAME parameter to the name of the sbpace in the ONCONFIG file, then create an sbpace with that name with the **onspaces** utility.

For a complete description of the **onspaces** utility, see your [Administrator's Guide](#).

Creating the User-Defined Virtual Processor

Verity Text Search DataBlade module routines run in a special user-defined virtual processor within the database server. You must create the user-defined virtual processor before you register and use the DataBlade module.

To create the user-defined virtual processor for the Verity Text Search DataBlade module, add the following line to the ONCONFIG file:

```
VPCLASS verity,num=1
```

Important: *There is no space between the comma and num.*

For more information on editing the ONCONFIG file and creating user-defined virtual processors, see your [Administrator's Guide](#).





Registering the DataBlade Module

Before you can use the DataBlade module, you must register it in each database in which you want to use it. Registration creates the DataBlade module objects and informs the database server of the location of the DataBlade module shared object file. You register DataBlade modules with the BladeManager application. See the [DataBlade Module Installation and Registration Guide](#) for instructions on registering.

Important: See the release notes for upgrading instructions and registration restrictions.

The Verity Text Search DataBlade module depends on the Informix Large Object Locator DataBlade module and the Text Descriptor DataBlade module. Therefore, when you register the Verity Text Search DataBlade module, BladeManager prompts you to first register the Informix Large Object Locator DataBlade module and the Text Descriptor DataBlade module, if they are not already registered in the database.

Choosing Data Types

This section explains how to choose data types to use for text columns in your database tables and how to create row and distinct data types if you choose to use them.

Data Type Guidelines

The Verity Text Search DataBlade module can perform index searches on columns whose data type is an *indexable data type*: one of the data types on which this DataBlade module can build indexes. Choose the appropriate data type for a column that will contain searchable text from the list in the following table.

Data Types	Description of Use
CHAR	For short ASCII documents stored in the database: for example, names, keywords, or brief comments
VARCHAR	For short ASCII documents stored in the database: for example, names, keywords, or brief comments
LVARCHAR	For short ASCII documents stored in the database that cannot be stored as VARCHAR
BLOB	For large binary formatted documents stored in the database
CLOB	For large ASCII documents stored in the database
IfxDocDesc	For files stored outside the database (<i>external files</i>)
Row data type	For multicolumn indexing and using custom contexts At least one field in the row data type must have a data type that is an indexable data type. Only indexable fields are indexed and used in index searches.
Distinct data type	For using custom contexts The source data type must be one of the indexable data types.

Creating Row and Distinct Data Types

If you want to customize searches on a particular column, define your own row or distinct data type and use that data type for the column.

A row data type must have at least one field that is an indexable data type. All fields that are indexable data types are indexed and used in index searches. In this way, you can emulate multicolumn searches. You can also use custom contexts with row data types.

A distinct data type must have an indexable data type as a source data type. Distinct data types are useful when you want to use custom contexts.

After defining your new data type, you can create a customized context for it. A context determines the characteristics of a **fts** index when it is created and the behavior of searches that use that index. For example, a context can specify a stopwords list or define locale-specific information (see “[Contexts and Style Files](#)” on page 1-10).

The following example creates a new distinct data type, `body_t`, that inherits all the properties of the CLOB data type:

```
CREATE DISTINCT TYPE body_t AS CLOB;
```

The next example creates a new row type, `movie_t`, that has four fields, three of which can support a **fts** index. Your new data type, `body_t`, is used to store document abstracts.

```
CREATE ROW TYPE movie_t
(
    director VARCHAR(30),
    title     VARCHAR(100),
    released  DATE,
    abstract  body_t
);
```

For more information on creating row and distinct data types, see the [Informix Guide to SQL: Syntax](#) manual.

Tip: If you want to create a language-specific data type, use the routines provided for that purpose instead of `CREATE DISTINCT TYPE` or `CREATE ROW TYPE`. See “[Working with Multilingual Databases](#)” on page 2-35.



Creating and Associating Contexts

This section explains how to create a new context for the Verity Text Search DataBlade module and how to associate it with a row or distinct data type.

A context guides the behavior of a **vts** index. It uses *style files* to customize searches by specifying index creation parameters such as indexing mode, case sensitivity, and stopwords.

When you associate a context with a row or distinct data type, the context is associated with all indexes built for columns of that data type.

All indexable data types except row and distinct data types use the default context provided with the Verity Text Search DataBlade module. If you create a new data type and do not associate it with a context, **vts** associates the default context with that data type. The default context is associated with the default set of style files. The default context is adequate for most searches.

To define and associate a context with a data type

1. Create a directory for your customized style files and copy the default Verity style files to that directory. The default style files are located in **\$INFORMIXDIR/extend/VTs.relno/vdkhome/common/style**, where *relno* represents the release number of the Verity Text Search DataBlade module.
2. Modify the newly copied style files. See [“Customizing Style Files” on page 2-9](#) for more information.
3. Read the modified style files into the database with the **Vts_ReadStyle()** routine. See [“Adding Style Files to the Database” on page 2-10](#) for more information.
4. Create a new context that uses your modified style files with the **Vts_CreateContext()** routine. See [“Creating a Context” on page 2-11](#) for more information.
5. Associate the custom context with the new data type using the **Vts_AssocContext()** routine. See [“Associating a Context with a Row or Distinct Data Type” on page 2-11](#) for more information.

Customizing Style Files

The default style files provided with this module are set to optimize the product's performance. Informix recommends that you use the default settings for most of these files. A few style files, such as **style.prm**, can be changed to accommodate features such as highlighting, summarization, and clustering. You can also add the files **style.go** (to index specialized vocabularies), **style.lex** (to specify nonalphanumeric characters to be indexed), and **style.stp** (to specify words to be omitted from indexes.)

Important: Do not change the default style files. Make any changes in your copies of style files. Do not customize the files **style.ddd**, **style.did**, **style.ngm**, **style.pdd**, **style.sid**, **style.vgw**, or **style.wld**.

Instructions for modifying and adding these style files are in [Chapter 6](#), “Verity Style Files.”

Important: Modification of style files after an index on them has been created has no effect on the index. The only way to change the characteristics of an index is to drop the index and re-create it with the modified style files and context.

Adding Style Files to the Database

When you finish customizing your style files, you must store them in an sbspace, using the **Vts_ReadStyle()** routine. The **Vts_ReadStyle()** routine defines a File System Emulation (FSE) directory. This is a construct that the Verity Text Search DataBlade module uses to associate a context with style file large objects. Specify a unique File System Emulation directory name that does not start with **/vts**. The FSE directory is not an operating system directory and is not available to you. It is used for internal purposes only.

In the following example, the **Vts_ReadStyle()** routine reads the style files found in the working directory **/mydir/mystyles/style1**, stores them as a smart large object in the sbspace named **sbspace**, and creates FSE entries in the FSE directory **/mydir/verity/fse_dir**:

```
BEGIN WORK;

EXECUTE PROCEDURE Vts_ReadStyle
(
  '$INFORMIXDIR/extend/VTS.1.10.UC1/vdkhome/common/style',
  'sbspace',
  '/mydir/fse_dir/styles1'
);

COMMIT WORK;
```

Use your own working directory instead of the one specified in the example. The FSE directory name is just an identifier; slashes are used only for convenience and are not required. The FSE directory has no relation to the file system on your computer.



Important: You must execute the **Vts_ReadStyle()** routine within a transaction.

For more information on the **Vts_ReadStyle()** routine, see [“Vts_ReadStyle\(\) Procedure” on page 4-35](#).

Creating a Context

Use the **Vts_CreateContext()** routine to create a context and make it known to your database, as shown in the following example:

```
EXECUTE PROCEDURE Vts_CreateContext
(
  'moviecntxt',
  '/mydir/fse_dir/styles1'
);
```

In this example, the name of the newly created context is **moviecntxt**. The routine looks for custom context information in the FSE directory **/mydir/fse_dir/styles1**. You must use the same FSE directory you specified as the third parameter to the **Vts_ReadStyle()** routine in the previous section.

If you execute the **Vts_CreateContext()** routine with a null style file directory, the context uses the default style files.

For more information on the **Vts_CreateContext()** routine, see [“Vts_CreateContext\(\) Procedure” on page 4-14](#).

Associating a Context with a Row or Distinct Data Type

Use the **Vts_AssocContext()** routine to associate a context with a row or distinct data type. A single data type can have only one context associated with it; however, a single context can be associated with many data types.

The following statements associate the context **moviecntxt** with the distinct data type **movie_t** and the row data type **body_t**:

```
EXECUTE PROCEDURE Vts_AssocContext (
    'moviecntxt', 'movie_t' );

EXECUTE PROCEDURE Vts_AssocContext (
    'moviecntxt', 'body_t' );
```

These statements ensure that when a **vts** index is built on a column of type **movie_t** or **body_t**, its behavior is governed by the characteristics of the **moviecntxt** context.



Important: The *CLOB*, *BLOB*, *VARCHAR*, *CHAR*, *LVARCHAR*, and *IfxDocDesc* data types are always associated with the default context, as are row and distinct data types not explicitly associated with a context.

Creating a Table with Text Search Columns

The following example creates a table with columns of the **movie_t** and **body_t** data types (created in [“Choosing Data Types” on page 2-6](#)):

```
CREATE TABLE movies (
    id          INTEGER PRIMARY KEY,
    item        movie_t,
    review      body_t
) PUT item IN (sbsp1), review IN (sbsp2);
```

In this example, the **item** column is stored in the sbspace **sbsp1**; the **review** column is stored in **sbsp2**. (These sbspaces were created in [“Creating sbspaces” on page 2-4](#)).

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

Creating the vts Index

You must create a **vts** index for every column on which you want to perform an index search. With the Verity Text Search DataBlade module, you can create both fragmented and unfragmented indexes.

When you create a **vts** index, you must specify the correct operator class for the indexable data type. See [“Operator Classes for Supported Data Types” on page 5-7](#) for a list of operator classes.



***Tip:** To speed index creation, turn logging off while you create a **vts** index. After you create an index, perform a level 0 archive. See the [“Archive and Backup Guide”](#) for archiving information.*

For more information on CREATE INDEX syntax, see [“CREATE INDEX Statement” on page 5-5](#).

Creating an Index on a CLOB Data Type Column

The following example creates an index, **movie_idx1**, of the **review** column of the **movies** table:

```
CREATE INDEX movie_idx1 ON movies (review Vts_clob_ops)
  USING vts (NOLOG='TRUE')
  IN sbsp2;
```

The operator class **Vts_clob_ops** is specified because the data type of the column **review** is **body_t**, which was created as a CLOB type.

This index is not fragmented. It is stored in the default sbspace, **sbsp2**.

Creating an Index on a Row Data Type Column

The following example creates a **vts** index on the **item** column of the **movies** table:

```
CREATE INDEX movie_idx2 ON movies (item Vts_row_ops)
  USING vts (NOLOG='TRUE')
  IN sbpace;
```

This time, the example specifies **Vts_row_ops** as the operator class because the data type of the indexed column is **movie_t**, a named row type. Only the **director**, **title**, and **abstract** fields in **movie_t** are indexed; they are of types **VARCHAR**, a built-in indexable data type, and **body_t**, a distinct type based on an indexable data type. The column **released** is not indexed because its data type is **DATE**, which is not indexable using the **vts** access method.

Creating a Fragmented Index

You can use the **FRAGMENT BY** clause of **CREATE INDEX** to create expression-based fragmentation.

The following example creates a fragmented index on the **review** column of the **movies** table:

```
CREATE INDEX movie_idx1 ON movies (review Vts_clob_ops)
  USING vts (NOLOG='TRUE')
  FRAGMENT BY EXPRESSION
  id < 1000 IN sbasp1,
  id >= 1000 IN sbasp2;
```

This example stores all documents whose **id** value is less than 1000 in the sbpace named **sbasp1** and all documents whose **id** value is greater than 1000 in **sbasp2**.

Multicolumn Indexing

The Verity Text Search DataBlade module enables indexing of row data types. However, it does not allow multicolumn indexing. If you want to index on a key that includes more than one column of a table, create a row data type that contains all the columns that you want to search. Then create a table with a column of this row data type, and move your data into this table. The examples in this section illustrate the use of a row data type, `movie_t`, for this purpose.

Indexing on a row data type achieves the same result as multicolumn indexing.

Populating the Table with Text Data

To load documents into a table that contains text columns, use an INSERT or UPDATE statement that references the documents.

Important: Row types must be explicitly cast to their data type. Null values inside row types must also be explicitly cast.

Inserting a File into a CLOB Column

In the following example, the abstract of the movie *Treasure of the Sierra Madre* is an ASCII document stored in the client operating system file `/mydir/movies/treasure.doc`. The review of the movie is an ASCII document stored in the file `/mydir/movies/treasure_rev.doc`.



The INSERT statement loads one row into the **movies** table. Both the **review** column and the **abstract** field of the **item** column have a CLOB data type, **body_t**. The INSERT statement uses the **FILETOCLOB()** function to convert the files containing the abstract and the review into character large objects for storage in the database table.

```
INSERT INTO movies values
(
    1,
    ROW('John Huston',
        'Treasure of the Sierra Madre',
        '5/7/48',
        FILETOCLOB (
            '/mydir/movies/treasure.doc',
            'client', 'movies', 'item'))::movie_t,
    FILETOCLOB (
        '/mydir/movies/treasure_rev.doc',
        'client', 'movies', 'review'))::body_t
);
```

The next example inserts similar files for the movie *Casablanca*:

```
INSERT INTO movies values
(
    2,
    row('Michael Curtiz',
        'Casablanca',
        '3/12/42',
        FILETOCLOB (
            '/mydir/movies/casablanca.doc',
            'client', 'movies', 'item'))::movie_t,
    FILETOCLOB (
        '/mydir/movies/casablanca_rev.txt',
        'client', 'movies', 'review'))::body_t
);
```

BLOB

To insert data into a BLOB column, use the **FILETOBLOB()** function instead of **FILETOCLOB()**. ♦

Refer to *[Informix Guide to SQL: Syntax](#)* for detailed information on the **FILETOBLOB()** and **FILETOCLOB()** functions.

IfxDocDesc

To see an insert statement that uses the **IfxDocDesc** data type, see *[“IfxDocDesc Data Type” on page 3-4](#)*. ♦

Performance Tips

You can significantly improve performance by using explicit transactions for inserting documents. In other words, if you have several inserts into a table with **vts** index, enclose them in a single transaction to minimize the overhead of repeatedly creating transactions with **vts** indexes. To do this, use the **BEGIN WORK** and **COMMIT WORK** statements to specify the beginning and end of each transaction. This strategy greatly reduces the transaction overhead associated with **vts** indexes. It also applies to update and delete statements.

You can find the syntax for these commands in the [Informix Guide to SQL: Syntax](#).

Performing Text Searches

To use the Verity Text Search DataBlade module to perform a text search, include the **vts_contains()** operator in the **WHERE** clause of a **SELECT** statement. The syntax of the **vts_contains()** operator is shown here and described in detail in “[vts_contains Operator](#)” on [page 4-9](#):

```
vts_contains (
    column_name,
    {query | ROW (query, tuning_parameters)}
    [, ret #IfxVtsReturnType]
)
```

The examples that follow illustrate several ways to search the indexed columns of the table **movies**, defined in the preceding sections. Some of the examples use Verity operators and modifiers that are described in [Appendix A, “Verity Operators and Modifiers.”](#)

Simple Queries

The following example searches for the titles of movies that had excellent reviews by looking for the word “excellent” in the **review** column:

```
SELECT item.title FROM movies
WHERE vts_contains (review, 'excellent' );
```

The next example shows searches for movies directed by John Huston that have the word “treasure” in one of the subfields of the **item** column:

```
SELECT item.title FROM movies
      WHERE vts_contains (item, 'treasure' ) AND
      item.director = 'John Huston';
```

An index scan is used in the previous searches because a **vts** index was built on both the **review** and the **item** columns. All the subfields of **item** are searched except **released**, whose data type is not indexable by the **vts** access method.

***Tip:** To be included in index searches, nonalphanumeric characters must be defined in a **style.lex** file.*



Searching in a Field of a Row Type Column

When you search one of the fields of an indexed column defined as a named row type (the **item** column in the **movies** table, for example), use one of the following notations:

- The Verity <IN> operator (for an index scan)
- Single-dot notation (for a nonindex scan)

The following examples show how to search for the word “treasure” in the field **abstract** of the column **item**:

```
SELECT item.title FROM movies
      WHERE vts_contains (item, '(treasure) <IN> abstract');

SELECT item.title FROM movies
      WHERE vts_contains (item.abstract, 'treasure');
```

In the first example, an index search is performed because the first parameter after **vts_contains** is a column name, **item**, and an index was created on this column.

In the second example, a nonindex scan is performed because the first parameter after **vts_contains** is a field name, **item.abstract**, and a **vts** index cannot be explicitly created on a field.

The following examples also search for the word “treasure” in the **abstract** field, but narrow the search by limiting results to movies directed by John Huston:

```
SELECT item.title FROM movies
WHERE vts_contains (item,
'(treasure) <IN> abstract AND (Huston) <IN> director');
```

In the first example, an index search is performed on only one of the **vts_contains** clauses; a nonindex scan is performed on the other. Only one **vts_contains** clause per **SELECT** statement can use an index search.

In the second example, an index search is done on both the word “treasure” and the name “Huston” because both parameters are specified in the same **vts_contains** clause. This example executes much faster than the first example.

Ordering Results by Score

Use the optional third parameter of the **vts_contains()** operator to order the rows returned in the previous example by their score. The statement local variable returns a value of type **IfxVtsReturnType**. One of the fields of this row type is **score**, which can be used with the **ORDER BY** clause, as shown in the next example:

```
SELECT item.title, ret.score FROM movies
WHERE vts_contains (review, 'excellent',
ret # IfxVtsReturnType )
ORDER BY 2 DESC;
```

The 2 in the **ORDER BY** clause refers to the number of the item in the **SELECT** list that is to be ordered. In this example, the **ret.score** is the second item in the list, so the number following **ORDER BY** is 2.

Summary and cluster information are not returned in the **SLV** because they are not requested.

Using the THESAURUS Operator

Use the Verity <THESAURUS> operator to search for documents that contain both the word you are looking for and its synonyms. For example, suppose you want to search for disaster movies, but are not sure whether the movies are described using the words “disaster,” “catastrophe,” or “cataclysm.” The following SELECT statement would return all movies whose **item** column contains synonyms of “disaster”:

```
SELECT item.title FROM movies
WHERE vts_contains (item, '<THESAURUS> disaster');
```

See [“THESAURUS” on page A-22](#) for more information on the THESARUS operator.

Using the PHRASE Operator

The Verity <PHRASE> operator treats a group of words like a single word instead of treating each word as a separate entity. For example, the following SELECT statement searches for the exact phrase “we’ll always have Paris” in the **item.abstract** field:

```
SELECT item.title FROM movies
WHERE vts_contains (item,
'<PHRASE>(We''ll always have Paris) <IN> abstract');
```

This query does not return documents that contain only the word “Paris.”

***Tip:** The single quote in the word “we’ll” is takes another single quote as an escape character to indicate that it is part of the search string.*

See [“PHRASE” on page A-18](#) for more information on the PHRASE operator.



Using the NEAR/n Operator

Use a proximity search if you want to specify how close to each other two or more words must be to satisfy the search criterion. For example, you might want to search for documents that contain the words “treasure” and “chest,” but only if they occur within three words of each other. Use the Verity <NEAR/n> operator to specify such a search, as shown in the following example:

```
SELECT item.title FROM movies
WHERE vts_contains (item,
  '(treasure <NEAR/3> chest) <IN> abstract');
```

This search returns movies that contain the strings “treasure chest” and “chest full of treasure” in the column **item.abstract**.

See [“NEAR/n” on page A-14](#) for more information on the NEAR/n operator.

Using the SOUNDEX Operator

The Verity SOUNDEX operator searches for both the specified word and words that sound like or have letter patterns similar to the search word. All retrieved words begin with the same letter as the search word.

SOUNDEX must be turned on in the **style.prm** file before creating the index or using the style files. The default style files turn SOUNDEX off. You can find syntax and other information about the **style.prm** file in [“style.prm” on page 6-17](#).

Once SOUNDEX is enabled, you can request it in a query by typing the string <SOUNDEX> in front of the search word. For example, the following query retrieves words such as “sale,” “sell,” “seal,” “shell,” “soul,” and “scale”:

```
SELECT item.title FROM movies
WHERE vts_contains (item,
  '<SOUNDEX>sale <IN> abstract');
```

See [“SOUNDEX” on page A-21](#) for more information on the SOUNDEX operator.



Using the CASE Modifier

The Verity CASE modifier allows you to exclude alternative lettercase variants from the search result.

Tip: By default, the Verity Text Search DataBlade module returns all case variants of search words entered in all uppercase or all lowercase letters. Search terms entered in mixed case return only exact matches and exclude other case combinations. To make all queries case-insensitive—even mixed case search terms—remove the “Casedex” option in your *style.prm* file. See [“style.prm” on page 6-17](#) for syntax and other information.

The <CASE> modifier excludes all case combinations except the one used in the search string.

For example, a normal search on the word “this” returns the following case variants:

```
this, THIS, This
```

Contrast these results with the results when using the <CASE> operator, next. The following example returns only movie reviews whose abstracts contain “this” in all lowercase letters:

```
SELECT title FROM poems
WHERE vts_contains (text,
'<CASE>tree <IN> abstract');
```

Two movie reviews contain the word “this.” The review of the movie *Casablanca* is: “this is a review of Casablanca. ‘Excellent’.” The review of the movie *Treasure of the Sierra Madre* is: “This is a review of ‘Treasure’.”

The example query returns only the review for *Casablanca*, because it has the word “this” in all lowercase. The review for *Treasure of the Sierra Madre* is not returned because “This” is capitalized.

Although stemming is generally turned on by default, when you use <CASE>, stemming is off. For example, if you search for the word “tree” using the <CASE> modifier, the word “trees” is not returned, as it would be without the <CASE> modifier. However, if you explicitly use the <STEM> modifier and the <CASE> modifier, then <CASE> is ignored.

Using the TOPN and MANY Parameters

Often, a search returns more documents than you need. You can use the **vts_contains** TOPN tuning parameter to specify the maximum number of rows returned from a search, based on their score. If the **vts** index is fragmented, TOPN represents the number of rows returned per fragment. For example, the following query returns only the top 10 rows that satisfy the search criterion:

```
SELECT item.title FROM movies
WHERE vts_contains (item,
ROW ('(<MANY> Sierra Madre) <IN> item',
'TOPN = 10'));
```

The Verity **<MANY>** operator ranks the rows returned from a search of the phrase “Sierra Madre” by their score. The TOPN tuning parameter indicates that you want only the top 10 rows in this ranking. The addition of a tuning parameter requires you to use the **ROW** constructor to separate the search string from the tuning parameter.

Tip: TOPN is ignored for nonindexed scans.

For more information on the TOPN tuning parameter, see [“List of Tuning Parameters” on page 4-10](#). For more information on the MANY operator, see [“MANY” on page A-28](#).



Clustering and Summarization

You can include summary and cluster information in the **summary** and **cluster** fields of the third parameter of the **vts_contains** operator. This parameter is a statement local variable, which has the data type `IfxVtsReturnType`.

A *summary* is a set of keywords or phrases that summarize a document and two sentences that represent the document. The maximum size of a summary is 512 bytes.

A *cluster* is a list of documents with similar content. Clustering automatically determines the subtopics in a set of documents and groups the documents by these subtopics. The typical size of clustering information is 1 to 2 kilobytes.

For more information on the `IfxVtsReturnType` data type, see [“IfxVtsReturnType Data Type” on page 3-9](#).

Creating and Populating the Poems Table

These examples create and use a table called **poems**. Except for the short poems, the poems are contained in files located in the directory **SINFOR-MIXDIR/extend/VTS.relno/docsamples/poems**. All poems are inserted into a CLOB column of the **poems** table. To use this data in your own examples, copy the poem files into your own directory and change the examples to point to their new location.

1. Create the **poems** table with the following statement:

```
CREATE TABLE poems
(
    author  VARCHAR(100),
    title   VARCHAR(100),
    text    CLOB
);
```

2. Create an index on the **text** column with the following statement:

```
CREATE INDEX poems_idx
ON poems (text Vts_clob_ops)
USING vts (NOLOG='TRUE')
IN sbspace;
```


3. Insert a few lines of a poem by Ogden Nash with the following statement:

```
EXECUTE PROCEDURE ifx_allow_newline('t');

INSERT INTO poems
  VALUES ( 'Ogden Nash', 'The Rabbits',
    FILETOCLOB(

'/informix/extend/VTS.1.10.UC1/docsamples/poems/nash.rabbit.
txt',
  'server', 'poems', 'text'));

EXECUTE PROCEDURE ifx_allow_newline('f');
```

Because the value in your INSERT statement span more than one line, you must execute the procedure **ifx_allow_newline('t')** before the INSERT statement and **ifx_allow_newline('f')** after the INSERT statement. Otherwise, the statement returns an error indicating that there are unmatched quotes.

4. Insert a poem by Carl Sandburg with the following statement:

```
INSERT INTO poems
  VALUES ('Carl Sandburg', 'Fog',
    FILETOCLOB(

'/informix/extend/VTS.1.10.UC1/docsamples/poems/sandburg.fog
.txt',
  'server', 'poems', 'text'));
```

5. Insert three Japanese poems with the following statements:

```
INSERT INTO poems
VALUES ('Issa', 'Autumn Wind (haiku)',
       FILETOCLOB(
'/informix/extend/VTS.1.10.UC1/docsamples/poems/haiku.autumn
.txt',
'server', 'poems', 'text'));

INSERT INTO poems
VALUES ('Basho', 'O Summer Snail (haiku)',
       FILETOCLOB(
'/informix/extend/VTS.1.10.UC1/docsamples/poems/haiku.summer
.txt',
'server', 'poems', 'text'));

INSERT INTO poems
VALUES ('Yamabe no Akahito', 'Cherry Blossoms',
       FILETOCLOB(
'/informix/extend/VTS.1.10.UC1/docsamples/poems/cherry.bloss
om.txt',
'server', 'poems', 'text'));
```

6. Insert a poem by Oscar Wilde with the following statement:

```
INSERT INTO poems
VALUES ('Oscar Wilde', 'Jardin des Tuileries',
       FILETOCLOB(
'/informix/extend/VTS.1.10.UC1/docsamples/poems/oscar.wilde.
txt',
'server', 'poems', 'text'));
```

7. Insert a poem by Dollie Radford with the following statement:

```
INSERT INTO poems
VALUES ('Dollie Radford', 'The Snow Queen',
       FILETOCLOB(
'/informix/extend/VTS.1.10.UC1/docsamples/poems/snow.queen.t
xt',
'server', 'poems', 'text'));
```

8. Insert a poem by Maya Angelou in HTML format with the following statement:

```
INSERT INTO poems
VALUES ('Maya Angelou', 'On the Pulse of Morning',
       FILETOCLOB(
         '/informix/extend/VTS.1.10.UC1/docsamples/poems/angelou.html',
         'server', 'poems', 'text'));
```

9. Insert one of Shakespeare's sonnets with the following statement:

```
INSERT INTO poems
VALUES ('William Shakespeare', 'Sonnet 18',
       FILETOCLOB(
         '/informix/extend/VTS.1.10.UC1/docsamples/poems/ws.sonnet.txt',
         'server', 'poems', 'text'));
```

Simple Queries Using Verity Features

Suppose you want to make a simple query on the **poems** table. This query requests all poems that have the word “haiku” in the title. Note that, because the **title** column has not been indexed, this statement performs a nonindex search:

```
SELECT author, title FROM poems
       WHERE vts_contains(title, 'haiku');
```

Two entries satisfy this criterion:

```
author  Issa
title   Autumn Wind (haiku)

author  Basho
title   O Summer Snail (haiku)
```

Using a Wildcard

To search for entries that begin with “silent,” use the * wildcard:

```
SELECT author, title FROM poems
WHERE vts_contains(text, 'silent*');
```

The results indicate that the expression “silent” is found in both “Fog” and “The Snow Queen”:

```
author    Carl Sandburg
title     Fog

author    Dollie Radford
title     The Snow Queen
```

Retrieving Information in the Statement Local Variable

The next few examples show how information about the results is returned.

Requesting SLV Information

The IfxVtsReturnType data type is a row type with three fields: **score**, **summary**, **cluster**.

The following query requests SLV information on each poem that contains the word “summer”:

```
SELECT title, ret FROM poems
WHERE vts_contains(text, 'summer',
ret #IfxVtsReturnType);
```

This statement produces these results:

```
title    Sonnet 18
ret      ROW(86.58000200000, NULL, NULL)

title    0 Summer Snail (haiku)
ret      ROW(79.66999800000, NULL, NULL)
```

The example returns the score because the default setting for the **score** field is ON. “Sonnet 18” has a score of about 86; “O Summer Snail” has a score of about 79. Because summarization and clustering are off by default, the results in the summary and cluster fields of `IfxVtsReturnType` are null.

Requesting Summary Information

Summary information is returned in a two-field row type: `IfxVtsSummaryType`. The fields are **keywords** (a list of important words in the text) and **summary** (selected sections from the text itself). For the complete syntax of the `IfxVtsSummaryType` data type, see “`IfxVtsSummaryType`” on page 3-9.

The following query requests SLV information, including summarization information. Note that the summary option is explicitly turned on in the **vts_contains** clause:

```
SELECT title, ret FROM poems
WHERE vts_contains(text,
    ROW('summer', 'summary=on'),
    ret #IfxVtsReturnType);
```

The first line of the results returns the titles of the matching documents, as requested. The second line returns the SLV, which is a ROW type consisting of three fields. The first field in the SLV contains the score. The second field contains summary information, which is returned as another ROW type consisting of two fields: keywords and a summary. The third field, **cluster**, returns `NULL` because clustering has not been turned on.

The results below are formatted so you can more easily understand them. The actual results are unformatted.

```
title Sonnet 18
ret ROW(78.00000000000,
      ROW('THOU, SUMMER, FAIR, SOMETIME, NOR, ETERNAL, THEE, ST, SHAKESPEARE,
          WILLIAM, SONNET, ROUGH, MAY, COMPLEXION, POSSESSION, UNTRIMMED,
          BREATHE, COMPARE, DECLINES, HEAVEN, LOVELY, TEMPERATE, CHANCE,
          CHANGING, COURSE',
          'Thou art more lovely and more temperate: Rough winds do shake the
          darling buds of May, And summer''s lease hath all too short a date:
          Sometime too hot the eye of heaven shines, And often is his gold
          complexion dimmed, And every fair from fair sometime declines,
          By chance, or nature''s changing course untrimmed: But thy eternal
          summer shall not fade, Nor lose possession of that fair thou ow''st,
          Nor shall death brag thou wand''rest in his shade, When in
          eternal lines to time thou grow''st, So long ...'),
      NULL)

title 0 Summer Snail (haiku)
ret ROW(78.00000000000,
      ROW('SLOWLY, BASHO, FUJI, SUMMER, CLIMB, SNAIL, TOP',
          'O summer snail you climb but slowly, slowly to the top of Fuji
          --Basho'),
      NULL)
```

Requesting Cluster Information

Cluster information is returned in a five-field row type. The fields are **id**, **type**, **score**, **doc_score**, and **keywords**. For the complete syntax of the `IfxVtsReturnType` data type, see [“IfxVtsReturnType Data Type” on page 3-9](#).

The following query requests score and cluster information:

```
SELECT title, ret FROM poems
WHERE vts_contains(text,
      ROW('summer', 'cluster=on'),
      ret #IfxVtsReturnType);
```

The query produces the following result:

```

title  Sonnet 18
ret    ROW(86.58000200000
        NULL,
        ROW(65534      ,1      ,0.00      ,0.00,
        'Outlier (Miscellaneous) Document'))
title  0 Summer Snail (haiku)
ret    ROW(79.66999800000
        NULL,
        ROW(65534      ,1      ,0.00      ,0.00,
        'Outlier (Miscellaneous) Document'))

```

Because summarization has not been turned on, a null value is returned in the **summary** field.

The term “Outlier” indicates that the document does not belong in the cluster.

You can produce the same result with the following query, in a slightly different format:

```

SELECT title, ret.score, ret.cluster_info FROM poems
WHERE vts_contains(text,
    ROW('summer', 'cluster=on'),
    ret #IfxVtsReturnType);

```

The next query requests all three types of SLV information on documents containing the word “blossom”: score, summary, and cluster, presented in order of score, best match first. Again, because the default behavior of clustering and summarization is turned off, the example explicitly turns them on.

```

SELECT title, ret.score, ret.summary_info, ret.cluster_info
FROM poems
WHERE vts_contains(text,
    ROW('silent*', 'summary=on, cluster=on'),
    ret #IfxVtsReturnType)
ORDER BY 2 DESC;

```

This query produces the following results (not formatted):

```
title      Cherry Blossoms
score      79.66999800000
summary    ROW('AKAHITO, YAMABE, BLOSSOMS, BRIGHT, CHERRY, DAYS, HILLS, PRIZE','F
           or these few days The hills are bright with cherry blossoms. Longer, a
           nd we should not prize them so. --Yamabe no Akahito ')
cluster    ROW(65534      ,1      ,0.00      ,0.00      ,'Outlier (Mi
           scellaneous) Document')
```



```
title      Jardin des Tuileries
score      77.41999800000
summary    ROW('SOMETIMES, WINTER, CLIMB, CHILDREN, MIMIC, BLUE, COLD, HAND, KEEN
           , TINY, TREE, TUILERIES, JARDIN, TRITON, OSCAR, HUGE, WILDE, DES, AH,
           GREENISH, LEAFLESS, BLOSSOMS, BOISTEROUS, BRIGANDS, SOLDIERS','This wi
           nter air is keen and cold, And keen and cold this winter sun, But roun
           d my chair the children run Like little things of dancing gold. Someti
           mes about the painted kiosk The mimic soldiers strut and stride, Somet
           imes the blue-eyed brigands hide In the bleak tangles of the bosk. And
           now in mimic flight they flee, And now they rush, a boisterous band-
           And, tiny hand on tiny hand, Climb up the black and leafless tree.')
cluster    ROW(65534      ,1      ,0.00      ,0.00      ,'Outlier (Mi
           scellaneous) Document')
```

Highlighting the Results

The Verity Text Search DataBlade module provides a routine for highlighting the search terms in results: **Vts_GetHighlight()**. The syntax for this routine is given in [“Vts_GetHighlight\(\) Function” on page 4-31](#).

When a document is highlighted, all occurrences of the specified string are distinguished by the specified highlight format when the document is displayed or printed.

To highlight the search terms in a document

1. Search for documents that contain the string to be highlighted.

Use the **SELECT** statement and the **vts_contains** operator to select the documents containing one or more strings that satisfy the search criterion. (See [“vts_contains Operator” on page 4-9](#).)

The result is a list of documents that contain the query string.

2. Provide highlighting instructions.

Execute the **Vts_GetHighlight()** routine for each document returned by the query to return highlighting instructions. For more information, see [“Vts_GetHighlight\(\) Function” on page 4-31](#).

3. Display the returned documents in a viewer. For example, to display HTML documents, use Netscape Navigator or Microsoft Internet Explorer.

Verity also provides a viewer that you can use to view many different formats. You can order this viewer directly from Verity, Inc. This viewer is sold and supported by Verity, Inc., not by Informix.

Highlighting ASCII Documents

Formatted ASCII documents contain embedded commands that provide formatting instructions. The text string to be formatted is preceded and followed by a pair of formatting instructions—the **prefix_string** and the **suffix_string**, respectively.

The **Vts_GetHighlight()** routine takes the specified **prefix_string** and **suffix_string** and combines them with the text that matches the **query_expression** to produce the desired result when the document is displayed.

Highlighting HTML Documents

This example shows how highlighting works for HTML documents.

One of the poems in the **poems** table, Maya Angelou’s “On the Pulse of Morning,” is in HTML format. The **text** column, which contains the text of the poem, is indexed.

To highlight text documents, you specify the prefix and suffix used by the document viewer to highlight sections of text. In HTML, you have a choice of highlighting methods.

The following table shows three different HTML prefix and suffix strings, one each for bold, italic, and underline formats. Using the word “mastodon,” it shows how these strings affect the appearance of a word or phrase.

prefix_string	suffix_string	query_expression	Result (in HTML document)	Result (as displayed)
		mastodon	mastodon	mastodon
<I>	</I>	mastodon	<I>mastodon</I>	<i>mastodon</i>
<U>	</U>	mastodon	<U>mastodon</U>	<u>mastodon</u>

The following SELECT statement underlines every instance of “mastodon” in the **text** column of the **poems** table:

```
SELECT Vts_GetHighlight (text, "mastodon", "<U>", "</U>")
FROM poems
WHERE vts_contains (text, "mastodon");
```

This query returns one poem—“On the Pulse of Morning”—as an ASCII file containing HTML tags, including the underline tags inserted by the statement. When displayed in an HTML browser, the highlighted version of the poem looks like [Figure 2-1](#).

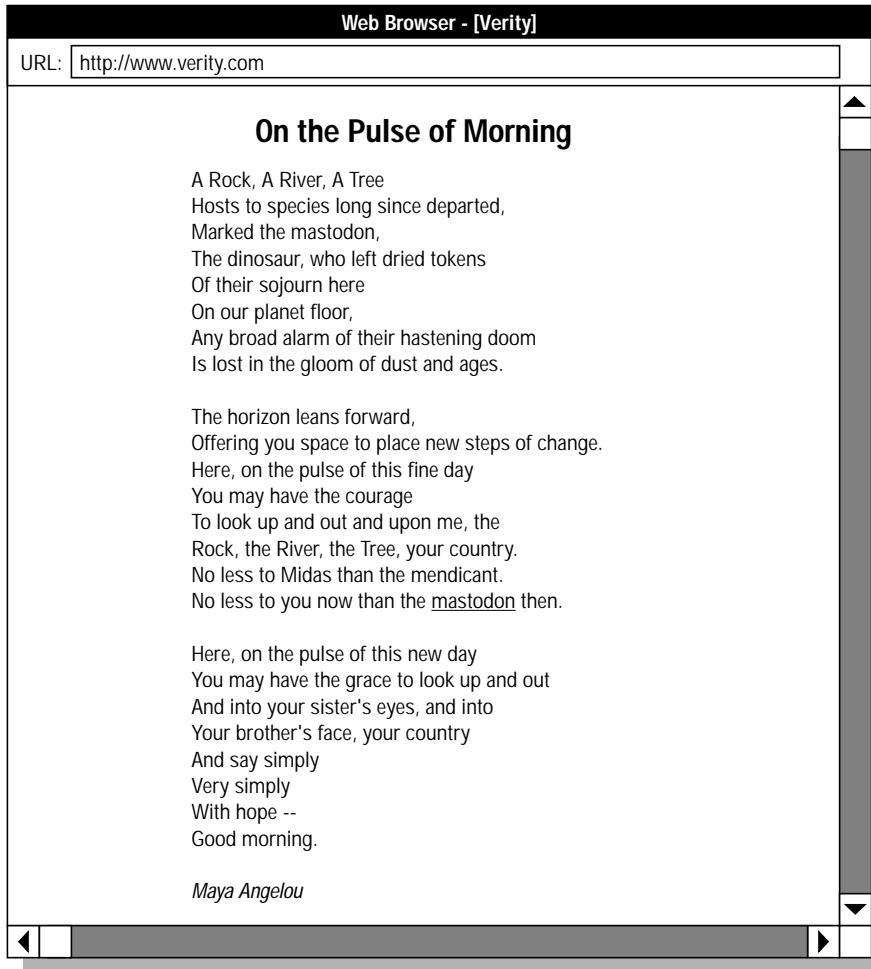


Figure 2-1
 Highlighted HTML
 Document

Working with Multilingual Databases

This section describes how to insert, index, and search documents in several languages in a single table. For a list of supported languages, see [“Multilingual Support” on page 1-7](#).

To create a multilingual database

1. Create a VTS locale.
2. Create a locale-specific data type.
3. Create a table.
4. Index columns in the table.
5. Insert data into the table.
6. Query the table.
7. Perform clean-up tasks.

Each of these tasks described in the following sections.

Creating a VTS Locale

To use documents written in a supported language, you must first define a locale for that language to your database. If you have customized style files for the language, you need to define them as well. Use the **vts_CreateLocale()** routine to make your locale and style files available to your database (see [“Vts_CreateLocale\(\) Procedure” on page 4-16](#)).

The following example creates a locale for the German language and specifies the style files to be used with German documents. German locale and style files are in `$INFOMIXDIR/extend/VTS.relno/vdkhome/common/deutsch`, where *relno* is the release number of the current version of the Verity Text Search DataBlade module.

```
BEGIN WORK;
EXECUTE PROCEDURE Vts_CreateLocale
(
    "deutsch",
    "$INFOMIXDIR/extend/VTS.1.10.UC1/vdkhome/common/",
    "$INFOMIXDIR/extend/VTS.1.10.UC1/vdkhome/common/style"
);
COMMIT WORK;
```

The following example creates a French locale from the files in `$INFOMIXDIR/extend/VTS.relno/vdkhome/common/francais`, using the style files in the **francais** directory:

```
BEGIN WORK;
EXECUTE PROCEDURE Vts_CreateLocale
(
    "francais",
    "$INFOMIXDIR/extend/VTS.1.10.UC1/vdkhome/common/",
    "$INFOMIXDIR/extend/VTS.1.10.UC1/vdkhome/common/style"
);
COMMIT WORK;
```

Creating a Locale-Specific Data Type

For each language whose documents you plan to store in your database, you must create at least one data type. Use this data type for columns in which you store documents in that language.

The following example creates a CLOB data type, `deutsch_clob`, to be used for columns in which German documents are stored. The `deutsch_clob` data type is associated with the **deutsch** locale:

```
EXECUTE PROCEDURE Vts_CreateType
(
    "clob",
    "deutsch_clob",
    "deutsch"
);
```

The following example creates a CLOB data type, `francais_clob`, to be used with French documents. The `francais_clob` data type is associated with the **francais** locale:

```
EXECUTE PROCEDURE Vts_CreateType
(
    "clob",
    "francais_clob",
    "francais"
);
```

Creating a Table

When you create a table to handle several languages, specify a separate column for each language whose documents will be stored in the table. In this example, documents in English, German, and French are stored in the table. Therefore, you create three CLOB columns, one for each language.

The following example uses a default locale of **en_us**:

```
CREATE TABLE intl_docs (
    docnum      int,
    English_doc  clob,
    German_doc   deutsch_clob,
    French_doc   francais_clob
);
```

Indexing Columns

The following statements create an index on each of the CLOB columns. Store the documents in the default sbspace, named **sbspace**:

```
CREATE INDEX Edoc_idx ON intl_docs (English_doc Vts_clob_ops)
    USING vts (NOLOG='TRUE') IN sbspace;

CREATE INDEX Gdoc_idx ON intl_docs (German_doc Vts_clob_ops)
    USING vts (NOLOG='TRUE') IN sbspace;

CREATE INDEX Fdoc_idx ON intl_docs (French_doc Vts_clob_ops)
    USING vts (NOLOG='TRUE') IN sbspace;
```

Inserting Data into the Table

The following statement inserts the poem “It is a Beauteous Evening, Calm and Free” by William Wordsworth, into the **intl_docs** table as document number 1. The German and French columns are left empty.

```
EXECUTE PROCEDURE ifx_allow_newline('t');

INSERT INTO intl_docs
VALUES
(
    1,
    FILETOCLOB(
        '/informix/extend/VTS.1.10.UC1/docsamples/poems/wordsworth.evening.txt',
        'server'),
    NULL::deutsch_clob,
    NULL::francais_clob
);

EXECUTE PROCEDURE ifx_allow_newline('f');
```

The following statement inserts some lines from “Der Fruhling,” by Friedrich Holderlin, into the **intl_docs** table as document number 2:

```
EXECUTE PROCEDURE ifx_allow_newline('t');

INSERT INTO intl_docs
VALUES
(
    2,
    NULL::clob,
    FILETOCLOB(

'/informix/extend/VTS.1.10.UC1/docsamples/poems/holderlin.fr
uhling.txt',
    'server')::deutsch_clob,
    NULL::francais_clob
);

EXECUTE PROCEDURE ifx_allow_newline('f');
```

The next statement inserts some lines from “Le Corbeau et le Renard,” by Jean de la Fontaine, into the **intl_docs** table as document number 3:

```
EXECUTE PROCEDURE ifx_allow_newline('t');

INSERT INTO intl_docs
VALUES
(
    3,
    NULL::clob,
    NULL::deutsch_clob,
    FILETOCLOB(

'/informix/extend/VTS.1.10.UC1/docsamples/poems/fontaine.cor
beau.txt',
    'server')::francais_clob
);

EXECUTE PROCEDURE ifx_allow_newline('f');
```


Querying the Table

The following statement searches for the word “summer” in the CLOB column and returns the document number of all documents that contain that phrase:

```
SELECT docnum FROM intl_docs
WHERE vts_contains (English_doc, 'summer');
```

This query returns the value 1.

The next statement searches for “renard” in the francais_clob column:

```
SELECT docnum FROM intl_docs
WHERE vts_contains (French_doc, 'renard');
```

This query returns the value 3.

Cleaning Up

When you end a multilingual session and want to drop a locale, drop your objects in the following order:

- Table
- Data type
- Locale

The following subsections show statements that drop the indexes, tables, data types, and locales used in the multilingual example.

Dropping a Table

When you drop a table, all indexes built on the table and all data in the table are also dropped. The following statement drops the table **intl_docs**:

```
DROP TABLE intl_docs;
```

Dropping a Data Type

The following statements disassociate the French and German CLOB data types from their contexts and drop the contexts and the data types:

```
EXECUTE PROCEDURE Vts_DropType("français_clob");  
  
EXECUTE PROCEDURE Vts_DropType("deutsch_clob");
```

Dropping a Locale

The following statements drop the French and German locales:

```
BEGIN WORK;  
  
EXECUTE PROCEDURE Vts_DropLocale( "français");  
  
EXECUTE PROCEDURE Vts_DropLocale( "deutsch");  
  
COMMIT WORK;
```

DataBlade Data Types

In This Chapter	3-3
IfxDocDesc Data Type	3-4
LLD_Locator Data Type	3-7
IfxVtsReturnType Data Type	3-9
IfxVtsSummaryType Data Type	3-12
IfxVtsClusterType Data Type	3-14
Locale-Specific Data Types	3-16

In This Chapter

This chapter describes the data types provided with the Verity Text Search DataBlade module. It also discusses the role of locale-specific data types used with languages other than English.

Two main data types are associated with the Verity Text Search DataBlade module: IfxDocDesc and IfxVtsReturnType. Both of these data types are named row types that have fields whose data types are also described in this chapter.

The following data types are described in this chapter.

Data Type	Description
IfxDocDesc	A named row type that allows you to index and search documents stored in external files. IfxDocDesc is a complex data type created by Informix and is used by text search DataBlade modules.
LLD_Locator	The data type of the location field of IfxDocDesc. LLD_Locator is a named row type used to locate documents stored as smart large objects or in external files.
IfxVtsReturnType	A named row type used to return information about results to the vts_contains operator. IfxVtsReturnType has fields for score, summary, and cluster information.
IfxVtsSummaryType	The data type of the summary field of IfxVtsReturnType. IfxVtsSummaryType is a named row data type.
IfxVtsClusterType	The data type of the cluster field of IfxVtsReturnType. IfxVtsClusterType is a named row data type.

IfxDocDesc Data Type

IfxDocDesc is one of the data types that can be indexed for a search by the Verity Text Search DataBlade module and is provided by the Verity Text Search DataBlade module.

IfxDocDesc is a named row data type composed of the following fields.

Field	Description	Data Type
format	<p>This field is for your own use, so you can establish your own guidelines for entering data here. The search module does not use this field.</p> <p>This field is typically used to specify the proprietary format in which the document is stored in the database: for example, MS Word or HTML.</p>	VARCHAR (18)
version	<p>This field is for your own use.</p> <p>This field is typically used to specify the version number of the document format: for example, for MS Word 6.0, this would be 6.0.</p>	VARCHAR (10)
location	This field contains information about the location of the text document. The document must be stored either as a smart large object in the database or in a file on either the client or the server system.	LLD_Locator See “LLD_Locator Data Type” on page 3-7.
params	This field is used to store document-specific information.	LVARCHAR

Only one IfxDocDesc field is currently used by the Verity Text Search DataBlade module: the **location** field. The **location** field uses a data type defined by the Informix Large Object Locator DataBlade module, which is included with Informix Dynamic Server with Universal Data Option.

The **location** field provides two pieces of information:

- The kind of large object in the document: a character large object, a binary large object, or an external file
- The location of the document

IfxDocDesc is part of the Informix Text Descriptor DataBlade module, which is provided on the media with the Verity Text Search DataBlade module.

Columns of data type IfxDocDesc are always associated with the default context. If you prefer, you can create your own row type similar to IfxDocDesc and associate it with a custom context. The Text Descriptor DataBlade module is also used by other Informix text search modules. If you are using more than one text search module, you may be able to retrieve data stored in IfxDocDesc format from the other modules. This is true of the BLOB and CLOB data types as well, because they are provided with the database server.

Usage

If the document is a binary or character large object, you can use the BLOB or CLOB data types. If it is an external file, you must use IfxDocDesc.

IfxDocDesc has a complex structure that consists of nested row data types. Writing INSERT statements that use the IfxDocDesc data type is difficult, error-prone, and time-consuming. For this reason, Informix recommends using BLOB and CLOB wherever possible, rather than IfxDocDesc.

IfxDocDesc is the only indexable data type that handles documents stored in external files on either the client or the server. If you plan to index and search documents stored in external files, use IfxDocDesc, or a distinct data type based on IfxDocDesc, as your data type. For smart large objects stored in the database, use a BLOB or a CLOB data type.

Example

The following statement inserts a row into the **movies** table, which was created in [Chapter 2, “Using the DataBlade Module.”](#) It inserts the same document inserted in [“Populating the Table with Text Data” on page 2-15.](#) In [Chapter 2](#), the document is inserted into a BLOB column. Here, the column’s data type is IfxDocDesc.

In this example, the abstract of the movie is a Microsoft Word 7.0 document stored in the operating system file **/tmp/verity/treasure.doc**. The **review** column is an ASCII document stored in the operating system file **/tmp/verity/treasure_rev.txt**.

```
INSERT INTO movies
VALUES
(
  1,
  ROW ('John Huston',
      'Treasure of the Sierra Madre',
      '5/7/48',
      FILETOCLOB (
        '/mydir/movies/treasure.doc',
        'client', 'movies', 'item'))::movie_t,
      FILETOCLOB (
        '/mydir/movies/treasure_rev.txt',
        'client', 'movies', 'review'))::body_t
);
```

You must use the **ROW()** constructor when specifying values for the named row types **movie_t**, **LLD_Lob**, **LLD_Locator**, and **body_t**. You must also explicitly cast all **NULLs** and all named row types to their data types.

LLD_Locator Data Type

LLD_Locator, the data type of the **location** field in the IfxDocDesc data type, is a named row data type used by the Verity Text Search DataBlade module and other DataBlade modules that use smart large objects.

LLD_Locator has the following fields.

Field	Description	Data Type
lo_protocol	The data type of the large object. This field can have one of the following values: IFX_BLOB Binary smart large object (BLOB) IFX_CLOB Character smart large object (CLOB) IFX_FILE An operating system file on the client or server machine	CHAR(18)
lo_pointer	If lo_protocol is IFX_BLOB or IFX_CLOB, this is a pointer to the smart large object. If lo_protocol is IFX_FILE, this value is NULL. You must explicitly cast a null value to LLD_Lob.	LLD_Lob See “ LLD_Lob Data Type ,” next
lo_location	If lo_protocol is IFX_FILE, this is the pathname of the large object. If lo_protocol is IFX_BLOB or IFX_CLOB, this value is NULL and should be explicitly cast.	LVARCHAR

LLD_Lob Data Type

The LLD_Lob data type is created using the following statement:

```
CREATE OPAQUE TYPE LLD_Lob
(
    internallength = 76,
    alignment = 8
);
```

See Also

[*Informix Large Object Locator DataBlade Module User's Guide*](#)

IfxVtsReturnType Data Type

IfxVtsReturnType is a named row type composed of the following fields.

Field	Description	Data Type
score	<p>A numeric value that indicates the degree to which the returned document matches the search criteria. The higher the score value, the more closely the document matches the criteria.</p> <p>score is a real value between 0 and 100. 0 indicates no match; 100 indicates the best possible match. score is a real number with 11 decimal places.</p>	REAL
summary_info	<p>Keywords, text phrases, and sentences that summarize the information contained in the returned document.</p> <p>The maximum length of a summary is 512 bytes.</p> <p>Summary information is returned on index queries only.</p>	IfxVtsSummaryType
cluste	<p>Information about additional documents that are similar to a returned document. Similarity is determined by frequency of keywords. Documents that are similar to one another, based on keywords, are grouped in clusters.</p> <p>Cluster information is returned on index queries only.</p>	IfxVtsClusterType

Usage

Use IfxVtsReturnType as the data type of the statement local variable (SLV) passed as the optional third parameter of the **vts_contains** operator. See [“vts_contains Operator” on page 4-9](#) for a description of the SLV.

You can return query results ordered by **score**, with the highest score first. Request the score and use the ORDER BY clause in the SELECT statement. To display the highest scoring documents first, specify DESC.

For example, the following statement returns the titles of all reviews that contain the word “excellent” in the **title** field of the **item** column of the **movies** table, listed in order of score, the highest score first.

Note the use of the ordinal number 2 to specify **ret.score**, which is the second item in the SELECT list.

```
SELECT item.title, ret.score FROM movies
WHERE vts_contains (review, 'excellent',
    ret # IfxVtsReturnType )
ORDER BY 2 DESC;
```

Results

With the Verity Text Search DataBlade module, you must specifically request results and specify which kinds of results you want.

If you want summary or cluster information, you must turn on the summary or cluster features in the **tuning_parameters** field of the **vts_contains** clause. (Score information is turned on by default when you use the SLV.)

The following query requests summary and cluster results, as well as score. Note that the ORDER BY clause uses a 2 to represent **ret.score**, which is the second item in the SELECT list.

```
SELECT title, ret.score, ret.summary_info, ret.cluster_info
FROM poems
WHERE vts_contains(text,
    ROW('silent*', 'summary=on, cluster=on'),
    ret #IfxVtsReturnType)
ORDER BY 2 DESC;
```

The previous query returns these results:

```

title          Fog
score          100.0000000000
summary_info   ROW('FOG, SANDBURG, CARL, HARBOR, HAUNCHES,
                  SILENT, LITTLE, CITY, COMES, FEET, LOOKING,
                  MOVES, SITS, CAT', 'The fog comes in on little cat
                  feet. It sits looking over harbor and city on
                  silent haunches and then moves on. Carl Sandburg')

cluster_info   ROW(65534      ,1      ,0.00      ,0.00
                  , 'Outlier (Miscellaneous) Document')

title          The Snow Queen
score          100.0000000000
summary_info   ROW('QUEEN, SNOW, FLOWERS, RADFORD, DOLLIE,
                  FELL, DAUNTLESS, SHOWERLIKE, DARKNESS,SILENTLY,
                  ENCHANTED, FLYING, CATCH, LIGHT, MAGIC, NIGHT,
                  STAND, BORNE, GOLD, HEAD, HOURS, MORNING, PASSED,
                  STAR,TREES', 'The Snow Queen The Snow Queen passed
                  our way last night, Between the darkness and the
                  light, And flowers from an enchanter

```

See Also

[“vts_contains Operator” on page 4-9](#)

IfxVtsSummaryType Data Type

IfxVtsSummaryType is a named row data type provided by the Verity Text Search DataBlade module and used to return summary information about documents that match a query's search criteria. IfxVtsSummaryType is the data type of the IfxVtsReturnType **summary_info** field.

IfxVtsSummaryType has the following fields.

Field	Description	Data Type
keywords	A list of key words from the document	LVARCHAR
summary	A summary of the document's contents, consisting of selected quotations from the document itself	LVARCHAR

You can retrieve summary information on indexed columns only.

Example

The following SELECT statement requests summary information as well as score. Note that the ORDER BY clause uses a 3 to represent **ret.score**, which is the third item in the select list.

```
SELECT title, author, ret.score, ret.summary_info.keywords
FROM poems
WHERE vts_contains(text, ROW('gold', 'summary=on'),
      ret #IfxVtsReturnType)
ORDER BY 3 DESC;
```

The query returns these results:

```

title      Sonnet 18
author     William Shakespeare
score      78.000000000000
keywords   THOU, SUMMER, FAIR, SOMETIME, NOR, ETERNAL, THEE,
           ST, SHAKESPEARE, WILLIAM, SONNET, ROUGH, MAY,
           COMPLEXION, POSSESSION, UNTRIMMED, BREATHE,
           COMPARE, DECLINES, HEAVEN, LOVELY, TEMPERATE,
           CHANCE, CHANGING, COURSE

title      The Snow Queen
author     Dollie Radford
score      78.000000000000
keywords   QUEEN, SNOW, FLOWERS, RADFORD, DOLLIE, FELL,
           DAUNTLESS, SHOWERLIKE, DARKNESS, SILENTLY,
           ENCHANTED, FLYING, CATCH, LIGHT, MAGIC, NIGHT,
           STAND, BORNE, GOLD, HEAD, HOURS, MORNING, PASSED,
           STAR, TREES

title      Jardin des Tuileries
author     Oscar Wilde
score      78.000000000000
keywords   SOMETIMES, WINTER, CLIMB, CHILDREN, MIMIC, BLUE,
           COLD, HAND, KEEN, TINY, TREE, TUILERIES, JARDIN,
           TRITON, OSCAR, HUGE, WILDE, DES, AH, GREENISH,
           LEAFLESS, BLOSSOMS, BOISTEROUS, BRIGANDS, SOLDIERS

```

The scores are identical, because each poem has exactly one instance of “gold.” Only keywords, not summaries, are shown because the **SELECT** statement requests only **ret.summary_info.keywords**, and not **ret.summary_info.summary**.

IfxVtsClusterType Data Type

IfxVtsClusterType is a named row data type built into the Verity Text Search DataBlade module and used to return cluster information on indexed fields. IfxVtsClusterType is the data type of IfxVtsReturnType’s **cluster_info** field.

IfxVtsClusterType has the following fields.

Field	Description	Data Type
id	The identifier of the cluster to which this document belongs. All documents in the same cluster have the same cluster ID.	INTEGER
type	The type of the cluster: 0 Normal 1 Outlier (outside the cluster) 2 Unclusterable	INTEGER
cluster_score	The overall score of the cluster. The higher the number, the better the fit.	REAL
doc_score	The degree to which the document fits into the cluster. The closer the number is to 10,000, the closer the document is related to the cluster concept.	REAL
keywords	The keywords used to match this document to the cluster. Clusters group documents by keyword. Documents that match the same keyword are in the same cluster and have the same cluster ID. If the document is outside the cluster, this field returns “Outlier (Miscellaneous) Document.”	LVARCHAR

You can retrieve cluster information on indexed columns only.

Example

The following example returns results for every field in the IfxVtsClusterType data type. Field names have been modified (using the AS keyword) for clarity. All returned documents are outside the cluster.

```
SELECT  title, author, ret.score,
        ret.cluster_info.id AS cluster_id,
        ret.cluster_info.type AS cluster_type,
        ret.cluster_info.cluster_score AS cluster_score,
        ret.cluster_info.doc_score AS cluster_docscore,
        ret.cluster_info.keywords AS cluster_keywords
FROM    poems
WHERE   vts_contains(text, ROW('gold', 'cluster=on'),
        ret #IfxVtsReturnType)
ORDER BY 3 DESC;
```

The query returns the following results:

title	Sonnet 18
author	William Shakespeare
score	78.000000000000
cluster_id	65534
cluster_type	1
cluster_score	0.00
cluster_docscore	0.00
cluster_keywords	Outlier (Miscellaneous) Document

title	The Snow Queen
author	Dollie Radford
score	78.000000000000
cluster_id	65534
cluster_type	1
cluster_score	0.00
cluster_docscore	0.00
cluster_keywords	Outlier (Miscellaneous) Document

title	Jardin des Tuileries
author	Oscar Wilde
score	78.000000000000
cluster_id	65534
cluster_type	1
cluster_score	0.00
cluster_docscore	0.00
cluster_keywords	Outlier (Miscellaneous) Document

Locale-Specific Data Types

You can use a single table to store documents in different languages. You do this by defining columns with locale-specific data types and inserting each document into the column whose data type corresponds to its locale.

Before you create a table and insert data, you must define to your database each locale whose documents will be stored in the table. Then, create at least one data type for each defined locale.

The Verity Text Search DataBlade module provides routines for creating locales and their data types. Use these routines, **Vts_CreateLocale()** and **Vts_CreateType()**, instead of the CREATE DISTINCT TYPE statement when you create a locale-specific data type. These routines are described in [Chapter 4, “DataBlade Routines.”](#)

See [“Working with Multilingual Databases” on page 2-36](#) for instructions on constructing, indexing, and searching a multilingual table.

DataBlade Routines

In This Chapter	4-3
Vts_AlterContext() Procedure	4-4
Vts_AssocContext() Procedure	4-6
vts_contains Operator	4-9
List of Tuning Parameters	4-10
Query Parser Options	4-12
Ordering Results	4-12
Vts_CreateContext() Procedure	4-14
Vts_CreateLocale() Procedure	4-16
Vts_CreateType() Procedure	4-19
Vts_DisassoContext() Procedure	4-21
Vts_DropContext() Procedure	4-23
Vts_DropLocale() Procedure	4-25
Vts_DropStyle() Procedure	4-27
Vts_DropType() Procedure	4-29
Vts_GetHighlight() Function	4-31
Vts_QueryContext() Function	4-33
Vts_ReadStyle() Procedure	4-35
Vts_Release() Function	4-38
Vts_WriteStyle() Procedure	4-39

In This Chapter

This chapter contains reference information about the routines provided by the Verity Text Search DataBlade module.

The routines are presented in this chapter in alphabetical order; however they fall into these categories.

Routine Category	Routine
Context Routines	Vts_Alter Context()
	Vts_AssocContext()
	Vts_CreateContext()
	Vts_DisassoContext()
	Vts_DropContext()
	Vts_QueryContext()
Multilingual Environment Routines	Vts_CreateLocale()
	Vts_CreateType()
	Vts_DropLocale()
	Vts_DropType()
Miscellaneous Routines	vts_contains()
	Vts_GetHighlight()
	Vts_Release()
Style File Management Routines	Vts_DropStyle()
	Vts_ReadStyle()
	Vts_WriteStyle()

Vts_AlterContext() Procedure

Vts_AlterContext() associates a custom context with a different set of style files.

Syntax

```
Vts_AlterContext (context_name, fse_style_dir)
```

Argument	Purpose	Restrictions	Data Type
<i>context_name</i>	Name of the context being modified.	Context must exist and must be a custom context. Wildcards are not permitted.	VARCHAR(64)
<i>fse_style_dir</i>	Pathname of the file system emulation (FSE) directory in which the new set of style files for the specified context is maintained. The FSE directory stores style file information as smart large objects. It is created by the routine Vts_ReadStyle() .	FSE directory must exist. Vts_ReadStyle() creates FSE directories for style files. If this field is NULL, the context uses the default style files. Wildcards are not permitted.	VARCHAR(255)

Usage

Use the **Vts_AlterContext()** procedure to modify a custom context by associating that context with set of style files different from those you have been using.

Style files contain index configuration options used by a context. They are stored in smart large objects and accessed via an FSE directory created by the **Vts_ReadStyle()** routine. This directory is not an operating system directory and cannot be queried or managed by operating system commands.

Important: Use this routine only with contexts created with **Vts_CreateContext()**. If your context was created with **Vts_CreateType()**, you cannot use **Vts_AlterContext()** to change its style file association.



To change the set of style files, change the value of *fse_style_dir* to the pathname of the FSE directory where the customized style files are stored.

You can modify a context only if the associated data type is not in use. A data type is in use if it is either the data type of an indexed column or the data type of a field in an indexed row type column.

To find out whether the context is in use, check the **vts_contexts** table. If the **refcount** for the context entry is zero, the context is not in use. If the **refcount** is not zero, you cannot use **Vts_AlterContext()** to modify this context.

For more information on this table, see [“Vts_Contexts Table” on page C-3](#).

Vts_AlterContext() is a procedure and can be executed using the EXECUTE PROCEDURE statement.

Example

The following example modifies the context named **report_cntxt** to use the style files in the FSE directory **/mydir/fse_dir/styles2**:

```
EXECUTE PROCEDURE Vts_AlterContext (
    'report_cntxt',
    '/mydir/fse_dir/styles2'
);
```

Related Routines

Vts_CreateContext() is used to create a context.

Vts_QueryContext() retrieves the name of the FSE directory.

Vts_ReadStyle() is used to create an FSE directory from style files.

See Also

[“Vts_Contexts Table” on page C-3](#)

[“Vts_Context_Type Table” on page C-2](#)

Vts_AssocContext() Procedure

Vts_AssocContext() associates a row or distinct data type with a custom context.

Syntax

`Vts_AssocContext (context_name, type_name)`

Argument	Purpose	Restrictions	Data Type
<i>context_name</i>	Name of the context being associated with a data type.	Context must already exist and must be a user-created context. Wildcards are not allowed.	VARCHAR(64)
<i>type_name</i>	Name of the data type with which the specified context is to be associated.	Data type must exist and must be a row or distinct data type. Data type cannot already have a context associated with it.	VARCHAR(64)

Usage

Use the **Vts_AssocContext()** procedure to associate a context with a row or distinct data type. The context is then associated with all columns that use that data type and with all indexes created on those columns.

A given data type can be associated with only one context. If you specify a data type that already has a context associated with it, **Vts_AssocContext()** returns an error. A context, however, can be associated with any number of data types.

To change the context for a data type that already has a context associated with it, you must first disassociate the current context from the data type using the **Vts_DisassoContext()** routine. Then, execute the **Vts_AssocContext()** procedure to associate a new context with the data type.

You can determine whether a data type has a context associated with it and, if so, which context it is associated with by querying the table **Vts_Context_Type**.

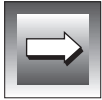
The following query returns all data type-context associations in your database:

```
SELECT * FROM Vts_Context_Type;
```

See [“Vts_Context_Type Table” on page C-2](#) for a description of this table.

To associate more than one context with a data type

1. Create one or more distinct data types built on the original data type.
2. Associate one context with each data type you created.
3. When you have finished, select a data type for a column based on the context associated with that data type.
4. Create a data type for each context.



Important: Only row or distinct data types can be associated with contexts using **Vts_AssocContext()**. The BLOB, CLOB, VARCHAR, LVARCHAR, CHAR, ROW, and IfxDocDesc data types are associated with default contexts when the product is installed. This association cannot be changed.

If you do not use **Vts_AssocContext()** to associate a context with a row or distinct data type, the data type is associated with the default context. The default context is defined by the Verity style files found in **\$INFORMIXDIR/extend/VTs.reln/vdkhome/common/style**.

Vts_AssocContext() adds a new row to the **Vts_Context_Type** table every time it associates a context with a data type. The **Vts_Context_Type** table is described in [Appendix C, “DataBlade Tables.”](#)

Vts_AssocContext() is a procedure and can be executed using the EXECUTE PROCEDURE statement.

Example

This example creates a data type, **report_t**, and associates it with the context **report_cntxt**, which was previously created using **Vts_CreateContext**. **report_t** is a distinct data type based on CLOB.

The example then creates a table with one column of type INTEGER, one column of type **report_t**, and one column of type VARCHAR. The example then creates an index on two of these columns.

The indexes built on the columns **body** and **summary** use different style files and therefore have different characteristics. The index **report_idx** uses the context **report_cntxt**, and the index **summary_idx** uses the default context.

```
CREATE DISTINCT TYPE movie_t2 AS CLOB;

EXECUTE PROCEDURE Vts_AssocContext (
    'moviecntxt',
    'movie_t2'
);

CREATE TABLE reports (
    id          INTEGER,
    body        movie_t2,
    summary     VARCHAR(128)
);

CREATE INDEX report_idx
    ON reports (body Vts_clob_ops)
USING vts (NOLOG='TRUE');

CREATE INDEX summary_idx
    ON reports(summary Vts_varchar_ops)
USING vts (NOLOG='TRUE');
```

Related Routines

Vts_AlterContext() associates a context with a different set of style files.

Vts_CreateContext() creates a new context.

Vts_DisassoContext() removes an association between a context and a data type.

See Also

[“Vts_Context_Type Table” on page C-2](#)

vts_contains Operator

The **vts_contains** operator is used in the WHERE clause of SQL statements to specify the column to be searched and the search criteria and to return information about the results.

Syntax

```
vts_contains(column_name, {query | ROW (query,
tuning_parameters)}) [SLV_name #IfxVtsReturnType])
```

Argument	Purpose	Rules	Data Type; Cross-References
column_name	Name of the column to be searched.	For index searches, the data type of the document must be an indexable data type. This argument is required.	Column expression, as described in the Informix Guide to SQL: Syntax .
query	Search criteria.	Search criteria can include regular expression matching, proximity matching, and all Verity operators. This argument is required.	LVARCHAR See Chapter 2, “Using the DataBlade Module,” for examples.
tuning_parameters	Parameters used to fine-tune the search.	One or more of the parameters in the “List of Tuning Parameters” on page 4-10 . This argument is optional.	LVARCHAR. See Chapter 2, “Using the DataBlade Module,” for examples.
SLV_name	Information about the returned documents. This argument is a statement local variable (SLV).	The return argument must declare its data type, IfxVtsReturnType, using the syntax #IfxVtsReturnType. Name the SLV in SLV_name. (Examples in his book use ret for this variable.) This argument is optional.	IfxVtsReturnType. See examples in “Performing Text Searches” on page 2-17 and in subsequent steps in Chapter 2 .

The **vts_contains** operator is used in SQL statements to specify the phrase to be searched, to customize the search parameters, and to return score, summary, and cluster information about returned documents.

The **vts_contains** operator performs text searches over a single column. The column being searched can be an indexed column or a nonindexed column. Searches on a nonindexed column are often slower. When an index is available, the optimizer determines whether or not to use it.

List of Tuning Parameters

The tuning parameters in the following table can be changed within a **vts_contains** call.

Tuning Parameter	Data Type	Purpose	Rules	Default Value
CLUSTER	Boolean	Turns clustering information on or off.	Takes one of the following values: <ul style="list-style-type: none">■ ON (turn on clustering)■ OFF (turn off clustering)	OFF
CUTOFFSCORE	INT	Specifies a score below which documents are not returned.	Must be an integer > 0 and <= 100	0
QP	CHAR	Specifies the kind of query parser to use to parse a Verity query string.	Takes one of the following values: <ul style="list-style-type: none">■ SIMPLE. Parser interprets simple syntax such as words and phrases separated by commas.■ EXPLICIT. Parser interprets syntax with explicit symbols, including Boolean queries.■ FREE_TEXT. Parser interprets natural language queries.	SIMPLE

(1 of 2)

Tuning Parameter	Data Type	Purpose	Rules	Default Value
SCORE	Boolean	Displays score information or not.	Takes one of the following values: <ul style="list-style-type: none">■ ON (display score information)■ OFF (do not display score information)	ON
SUMMARY	Boolean	Displays summary information or not.	Takes one of the following values: <ul style="list-style-type: none">■ ON (return summary information)■ OFF (do not return summary information)	OFF
TOPN	INT	The number of documents to return per index fragment, based on rank (used for index scans only).		All matched documents

(2 of 2)

To have a Verity text index built on it, a column’s data type must be one of the indexable data types listed in [“Operator Classes for Supported Data Types” on page 5-7](#).

You can use Verity operators in your search phrase. These operators are documented in [Appendix A, “Verity Operators and Modifiers.”](#) You can also find information about Verity operators and modifiers in the Verity manuals and on the Verity Web page.

Verity operators allow you to customize your search as follows:

- You can search for individual words or for phrases.
- You can specify that the search terms be near each other.
- You can use wildcards.
- You can request that synonyms also be returned from the Verity thesaurus.

See [Appendix A, “Verity Operators and Modifiers,”](#) for examples of these and other types of searches.

Query Parser Options

The following three queries each use a different query parser tuning parameter, but return the same result:

```
SELECT review FROM movies
  WHERE vts_contains(review, ROW('excellent', 'QP=SIMPLE'));

SELECT review FROM movies
  WHERE vts_contains(review, ROW('<PHRASE>("excellent")',
                                'QP=EXPLICIT'));

SELECT review FROM movies
  WHERE vts_contains(review, ROW('Show me the excellent
remarks',
                                'QP=FREE_TEXT'));
```

See [“Simple and Explicit Syntax” on page A-8](#) for more information.

Ordering Results

When you perform a pattern search, some of the returned rows may satisfy search criteria better than others. To determine the degree of similarity between your search string and a returned row, you can instruct the search engine to assign a value, called a *score*, to each returned row. Scores range from close to 0 (a very weak match) to 100 (a perfect match).

Score information is returned as the **score** field of the SLV.

Cluster and summary information are returned in the **cluster** and **summary** fields, respectively.

If an index search is used, the query returns all three kinds of information: score, cluster, and summary. In the following example, a null value is returned in the summary field, because summary has not been turned on:

```
SELECT title, ret FROM poems
  WHERE vts_contains(text,
                    ROW('summer', 'cluster=on'),
                    ret #IfxVtsReturnType);
```

See [Chapter 3, “DataBlade Data Types,”](#) for more information on the `IfxVtsReturnType` data type.

The **vts_contains** operator does not automatically return results in order. To order the results by score, use the ORDER BY keyword in the SELECT statement, specifying **ret.score** as the key on which to order results. To order results by **ret.score**, include it in the select list of the query and include its select number in the ORDER BY clause. Remember that if you want to order by score, you must include the return parameter in the **vts_contains** clause.

For more information on the ORDER BY clause of SELECT statements and SLVs, see [Informix Guide to SQL: Syntax](#). For examples that order results, see [Chapter 2, “Using the DataBlade Module.”](#)

Vts_CreateContext() Procedure

Vts_CreateContext() creates a new context.

Syntax

```
Vts_CreateContext (context_name, fse_style_dir)
```

Argument	Purpose	Restrictions	Data Type
<i>context_name</i>	Name of the context being created.	Context name must be unique within the database. Maximum name length: 64 bytes.	VARCHAR(64)
<i>fse_style_dir</i>	Pathname of the FSE directory where the set of style files to be used by this context is maintained.	Wildcards are not permitted. If this field is NULL, the context uses the default style files. The FSE directory must have already been created using the Vts_ReadStyle() routine.	LVARCHAR

Usage

Use **Vts_CreateContext()** to create a new context. Context names are unique within the database.

A context consists of a set of configuration options that define the characteristics of the index. Configuration options are specified in Verity style files. The default style files are provided with this product in the directory **\$INFORMIXDIR/extend/VTs.relno/vdkhome/common/style** (*relno* represents the release number of the Verity Text Search DataBlade module you are using).

Each context has its own style directory, where the specific set of style files used with that context is stored. The default context uses the default style files.

You can create customized style files for your context. Use the procedure **Vts_ReadStyle()** to create new style files and store them as smart large objects maintained in an FSE style directory. The association between a custom context and customized style files is established by this routine.

Once you have created a context, use **Vts_AssocContext()** to associate the context with a data type.

See [“Creating and Associating Contexts” on page 2-9](#) for instructions on creating contexts and associating them with data types.

Vts_CreateContext() adds a new row to the **Vts_Contexts** table every time it creates a context. The **Vts_Contexts** table is described in [Appendix C, “DataBlade Tables.”](#) **Vts_CreateContext()** is a procedure and can be executed using the EXECUTE PROCEDURE statement.

Example

The following example creates a context named **moviecntxt**, which uses the style files contained in the **/mydir/fse_dir/styles1** FSE directory:

```
EXECUTE PROCEDURE Vts_CreateContext
(
    'report_cntxt',
    '/mydir/fse_dir/styles1'
);
```

Related Routines

Vts_AlterContext() alters a context with a different set of style files.

Vts_AssocContext() associates a row or distinct data type with a custom context.

Vts_DisassoContext() removes the association between a context and a data type.

Vts_DropContext() drops an existing context.

Vts_QueryContext() retrieves the name of the FSE directory associated with the specified context.

Vts_CreateLocale() Procedure

Vts_CreateLocale() makes a nondefault locale available to your database.

Syntax

```
Vts_CreateLocale (locale_name, locale_path, os_style_dir)
```

Argument	Purpose	Restrictions	Data Type
<i>locale_name</i>	Name of the operating system directory where the locale files for the language are located.	<p>Name only the directory itself. Do not include the path.</p> <p>The name of this directory is typically the name of the language whose locale files are stored here.</p> <p>This value must be the same as the value specified in the <i>language</i> parameter of the Vts_CreateType() statement that creates the data type used for columns containing documents in this language.</p>	LVARCHAR
<i>locale_path</i>	<p>Path where <i>locale_name</i> is located.</p> <p>This is the directory from which the locale files are copied into an FSE directory.</p>	<p>This is an operating system directory.</p> <p>As delivered, locale directories are located in \$INFOR-MIXDIR/extend/VTS.reln/vdkhome/common/.</p> <p>Wildcards are not permitted.</p>	LVARCHAR
<i>os_style_dir</i>	Pathname of the operating system directory in which the style files used for this locale are stored.	<p>Use the style files that have been customized for this language. Do not use the directory where the default style files are located.</p> <p>Wildcards are not permitted.</p>	LVARCHAR

Usage

Use **Vts_CreateLocale()** to define a language locale to your database. Executing **Vts_CreateLocale()** enables the Verity Text Search DataBlade module to index and search documents written in a language other than the default language.

You must execute **Vts_CreateLocale()** once for each language whose files you plan to search. Then, create at least one data type for each locale you create, using **Vts_CreateType()**. The Verity Text Search DataBlade module uses this locale for all columns that use its data types.

The default locale is created when you register the Verity Text Search DataBlade module in your database.

Vts_CreateLocale() creates FSE directories for the style files and the locale. You do not need to run **Vts_ReadStyle()** for locale-specific style files.

After you execute this procedure, do not change the style files in the specified directory. Any changes made to these style files after **Vts_CreateLocale()** is executed have no effect on indexing or search operations on columns that use this language.

Example

The following statement makes the locale for French language documents, named **français**, available to your database. In the example, the files for the French language are in **\$INFORMIXDIR/extend/VTS.reln/vdkhome/common/français**, where *reln* is the release number. The customized style files to be used with this locale are in the same directory.

```
BEGIN WORK;
EXECUTE PROCEDURE Vts_CreateLocale
(
    "français",
    "$INFORMIXDIR/extend/VTS.1.10.UC1/vdkhome/common/",
    "$INFORMIXDIR/extend/VTS.1.10.UC1/vdkhome/common/style"
);
COMMIT WORK;
```

Whenever you index or search on a column that uses a French data type, the customized style files in **\$INFORMIXDIR/extend/VTs.reln/vdkhome/common/francais** guide the behavior of the index or the search.

Related Routines

Vts_CreateType() creates a new language-specific distinct data type.

Vts_DropLocale() makes a locale unavailable to your database.

See Also

[*Guide to GLS Functionality*](#)

Vts_CreateType() Procedure

Vts_CreateType() creates a new, language-specific, distinct data type.

Syntax

```
Vts_CreateType ( source_type, newtype_name, language )
```

Argument	Purpose	Restrictions	Data Type
<i>source_type</i>	Name of the existing data type on which the new data type is based.	Must be one of the indexable data types listed in “Indexable Data Types” on page 5-7 or a distinct data type built on an indexable data type. IfxDocDesc is not supported for locales other than en_us .	LVARCHAR
<i>newtype_name</i>	Name of the distinct data type being created.	It is a good idea to include both the language and the base type in this name. The name of the data type must be unique within the database.	LVARCHAR
<i>language</i>	Language to be used by the new data type.	This value must be the same as the value specified in the <i>locale_name</i> argument of the Vts_CreateLocale() routine that creates the locale for this language.	LVARCHAR

Usage

Use **Vts_CreateType()** to create a language-specific distinct data type based on an indexable data type. This data type is associated with a particular locale and set of style files, which govern the behavior of indexing and search operations performed on columns of this type.

You must create at least one language-specific data type for each language whose documents you will be searching.

Vts_CreateType() creates a context for the locale and associates it with the new data type. You do not need to execute **Vts_CreateContext()** or **Vts_AssocContext()** when you create a data type using **Vts_CreateType()**.

Each time **Vts_CreateType()** executes successfully, it adds a row to the **Vts_Contexts** table, the **Vts_Context_Type** table, and to the **sysxdtypes** system table.

To determine the name of the context associated with the new data type, query the **Vts_Context_Type** table.

Example

The following statement creates a distinct data type for the French language, based on the CLOB data type. The new data type is named **francais_clob**.

```
EXECUTE PROCEDURE Vts_CreateType
(
    "clob",
    "francais_clob",
    "francais"
);
```

The locale for the French language uses the style files in **\$INFORMIXDIR/extend/VTs.reln/vdkhome/common/francais**. Therefore, whenever you index or search on a column that uses the **francais_clob** data type, the style files in **\$INFORMIXDIR/extend/VTs.reln/vdkhome/common/francais** guide the behavior of the index or the search.

Related Routines

Vts_CreateLocale() makes a nondefault locale available to your database. **Vts_DropType()** makes a row or distinct data type unavailable to the DataBlade module by removing its entry from the **sysxdtypes** system table.

See Also

CREATE DISTINCT TYPE in the [Informix Guide to SQL: Syntax](#)
“**Vts_Contexts** Table” on page C-3
“**Vts_Context_Type** Table” on page C-2

Vts_DisassoContext() Procedure

Vts_DisassoContext() removes the association between a context and a data type.

Syntax

```
Vts_DisassoContext ( context_name, type_name )
```

Argument	Purpose	Restrictions	Data Type
<i>context_name</i>	Name of the context being disassociated from a data type.	Context must exist and must be associated with the data type specified in <i>type_name</i> .	VARCHAR(64)
<i>type_name</i>	Name of the data type associated with the specified context.	Data type must exist and must be associated with the context specified in <i>context_name</i> . Reference count of indexes that use this column must be zero.	VARCHAR(64)

Usage

Use **Vts_DisassoContext()** to remove the association between a context and a data type. When you disassociate a context from a data type, the data type uses the default context from then on.

You can disassociate a context from a data type only if the data type is not in use—that is, the reference count of the indexes that use that context is zero. The reference count is maintained in the **vts_contexts** table, which is described in [“Vts_Contexts Table” on page C-3](#).

If the data type and context specified in this statement are not associated, an error is returned.

Important: Use this routine only with contexts that were created with the **Vts_CreateContext()** routine. If your context and data type were created with the **Vts_CreateType()** routine, you cannot use **Vts_DisassoContext()** to remove the association.



Vts_DisassoContext() deletes a row from the **Vts_Context_Type** table every time it dissolves the association between a context and a data type.

Vts_DisassoContext() is a procedure and can be executed using the EXECUTE PROCEDURE statement.

Example

The following example removes the association between the distinct data type **report_t** and the context **report_cntxt**:

```
EXECUTE PROCEDURE Vts_DisassoContext (  
    'report_cntxt',  
    'report_t'  
);
```

An error is returned if **report_f** has not previously been associated with **report_cntxt**.

Related Routines

Vts_AssocContext() associates a row or distinct data type with a custom context.

Vts_CreateContext() creates a new context.

See Also

[“Vts_Contexts Table” on page C-3](#)

Vts_DropContext() Procedure

Vts_DropContext() drops an existing context.

Syntax

```
Vts_DropContext (context_name)
```

Argument	Purpose	Restrictions	Data Type
context_name	Name of the context being dropped.	The named context must exist and must not be in use.	VARCHAR(64)

Usage

Use **Vts_DropContext()** to drop a previously created context by name.

A context can be dropped only if it is not in use. In other words, a context can only be dropped if the reference count of the indexes that use the context is zero. The reference count is maintained in the **Vts_Contexts** table, which is described in “[Vts_Contexts Table](#)” on page C-3.

Vts_DropContext() deletes a row from the **Vts_Contexts** table every time it creates a context.



***Important:** Use this routine only with contexts that were created with the **Vts_CreateContext()** routine. If your context was created with the **Vts_CreateType()** routine, do not use **Vts_DropContext()** to drop it: use **Vts_DropType()** instead.*

Vts_DropContext() is a procedure and can be executed using the EXECUTE PROCEDURE statement.

Example

The following example drops the context named **report_cntxt**:

```
EXECUTE PROCEDURE Vts_DropContext (  
    'report_cntxt'  
);
```

Related Routines

Vts_CreateContext() creates a new context.

See Also

[“Vts_Contexts Table” on page C-3](#)

Vts_DropLocale() Procedure

Vts_DropLocale() makes a locale unavailable to your database.

Syntax

```
Vts_DropLocale ( locale_name )
```

Argument	Purpose	Restrictions	Data Type
locale_name	Name of the directory where the locale files for the language are stored.	Name only the directory itself. Do not include the path. The locale must not be in use. If it is, this routine returns an error. Wildcards are not permitted.	LVARCHAR

Usage

Use **Vts_DropLocale()** to remove a locale from your database.

You must execute **Vts_DropLocale()** once for each locale that you want to remove.

You must delete the data types for a language before you drop the locale.

Example

The following statements drop the French and German locales:

```
BEGIN WORK;  
  
EXECUTE PROCEDURE Vts_DropLocale( "français");  
  
EXECUTE PROCEDURE Vts_DropLocale( "deutsch");  
  
COMMIT WORK;
```

Related Routines

Vts_CreateLocale() makes a nondefault locale available to your database.

Vts_CreateType() creates a new, language-specific, distinct data type.

Vts_DropType() makes a row or distinct data type unavailable to the Verity Text Search DataBlade module by removing its entry from the **sysxdtypes** system table.

Vts_DropStyle() Procedure

Vts_DropStyle() deletes the specified FSE directory and the smart large objects it contains.

Syntax

```
Vts_DropStyle ( fse_style_dir )
```

Argument	Purpose	Restrictions	Data Type
fse_style_dir	Pathname of the FSE directory where style file smart large objects are stored.	Directory must exist and must be an FSE directory created using the routine Vts_ReadStyle() .	VARCHAR(128)

Usage

Use **Vts_DropStyle()** to delete the specified FSE directory and its contents. The style is dropped if it is not associated with any context.

Vts_DropStyle() is a procedure and can be executed using the EXECUTE PROCEDURE statement.

Important: You must execute this routine within a transaction.

Example

This example drops all style files in the FSE directory **/mydir/fse_dir/styles**:

```
BEGIN WORK;  
  
EXECUTE PROCEDURE Vts_DropStyle (  
    '/mydir/fse_dir/styles2'  
);  
  
COMMIT WORK;
```



Related Routines

Vts_CreateContext() creates a new context.

Vts_AlterContext() associates a context with a different set of style files.

Vts_ReadStyle() creates an FSE directory from style files that currently reside on the operating system.

Vts_WriteStyle() writes all smart large objects in a specified FSE directory to a specified directory on the operating system.

See Also

SEARCH'97 Developer's Kit Advanced Features Guide

Vts_DropType() Procedure

Vts_DropType() makes a distinct data type unavailable to the Verity Text Search DataBlade module by removing its entry from the **sysxdtypes** system table.

Syntax

```
Vts_DropType ( type_name )
```

Argument	Purpose	Restrictions	Data Type
<i>type_name</i>	Name of the data type being deleted.	This data type must not be in use.	LVARCHAR

Usage

Use **Vts_DropType()** to remove a language-specific data type from the database. This routine automatically disassociates the data type from its context and removes the context from the database.

As with DROP TYPE, the operation fails if the data type to be dropped is in use. If any indexes are built on columns of this data type or if you have created any functions or casts that use this data type, the drop operation fails. To find out whether the context is in use, check the **Vts_Context_Type** table. If the reference count for the data type's entry is zero, the data type is not in use. If the reference count is not zero, **Vts_DropType()** returns an error.

Each time **Vts_DropType()** executes successfully, it deletes a row from the **Vts_Contexts** table, the **Vts_Context_Type** table, and the **sysxdtypes** system table.

You must execute **Vts_DropType()** once for each data type to be removed.

Example

The following statements drop the French and German data types:.

```
EXECUTE PROCEDURE Vts_DropType("français_clob");  
  
EXECUTE PROCEDURE Vts_DropType("deutsch_clob");
```

Related Routines

Vts_CreateType() creates a new, language-specific, distinct data type.

Vts_DropLocale() makes a locale unavailable to your database.

See Also

DROP TYPE in the [Informix Guide to SQL: Syntax](#)

Vts_GetHighlight() Function

Vts_GetHighlight() highlights strings that match the search criteria in a selected document. Use this function for documents in ASCII format.

Syntax

```
Vts_GetHighlight (column_name, query_expression,  
                prefix_string, suffix_string)
```

Argument	Purpose	Restrictions	Data Type
<i>column_name</i>	Name of the column containing the document to be highlighted. If the column has a ROW data type, you can specify a field within the row, using single-dot notation.	This document must be in ASCII format.	One of the following: BLOB CLOB LVARCHAR VARCHAR(255) IfxDocDesc LLD_Locator
<i>query_expression</i>	Expression representing the string to be highlighted. This is the <i>search criterion</i> .	This is the same expression used in the vts_contains clause that returned the document to be highlighted.	LVARCHAR See “ vts_contains Operator ” on page 4-9 . Also, see the Informix Guide to SQL: Syntax .
<i>prefix_string</i>	Instruction to the formatter to turn on the desired highlighting mechanism.	Restrictions vary according to the formatter used to display the results.	VARCHAR(255)
<i>suffix_string</i>	Instruction to the formatter to turn off the desired highlighting mechanism.	Restrictions vary according to the formatter used to display the results.	VARCHAR(255)



Returns

Vts_GetHighlight() returns a CLOB data type containing an ASCII version of the document, with the specified terms highlighted by the highlighting prefix and suffix strings.

Important: The returned CLOB data type is not persistent. You must store returned CLOB data types in a table, using the standard SQL INSERT statement, to make them available to future calls.

Usage

Use **Vts_GetHighlight()** to highlight text strings in a document that was originally in or has been converted to ASCII. For example, HTML or PDF files, or MIF files for FrameMaker documents. (See [“Vts_QueryContext\(\) Function”](#) to learn how to highlight non-ASCII strings.)

To highlight a word or phrase, you specify a prefix and suffix, representing on and off commands, that change the display characteristics of that string, distinguishing it from the surrounding text. When you display the document, **Vts_GetHighlight()** uses these commands to highlight strings that match the query string.

Vts_GetHighlight() can be used in SQL statements.

Vts_GetHighlight() adds highlights to one document at a time. If the column's data type is a row data type, you must write a separate SELECT statement for each field, because only one field is highlighted at a time.

Example

The following example highlights an HTML file. See [“Highlighting the Results” on page 2-32](#) for the results.

```
SELECT Vts_GetHighlight (text, "mastodon", "<U>", "</U>")
FROM poems
WHERE vts_contains (text, "mastodon");
```

Vts_QueryContext() Function

Vts_QueryContext() retrieves the name of the FSE directory associated with the specified context.

Syntax

Vts_QueryContext (*context_name*)

Argument	Purpose	Restrictions	Data Type
<i>context_name</i>	Name of the context being queried.	Context must exist. Wildcards are not permitted.	VARCHAR(64)

Usage

Use **Vts_QueryContext()** to find out the name of the FSE directory associated with the specified context.

Use the routine **Vts_WriteStyle()** to write the style files from sbspaces in the database to operating system files. Then you can use a text editor to read the operating system files to find out what stopword list, case sensitivity, and other options are supported by the context.

Returns

Vts_QueryContext() returns a VARCHAR(255) string containing the name of the FSE directory associated with the specified context.

Example

The following example returns the name of the FSE directory associated with the context **report_cntxt**:

```
EXECUTE FUNCTION Vts_QueryContext (
    'report_cntxt'
);
```

Your application must store this information in a variable.

Related Routines

Vts_CreateContext() creates a new context.

Vts_AlterContext() associates a context with another set of style files.

Vts_ReadStyle() Procedure

Use **Vts_ReadStyle()** to create an FSE directory from style files that currently reside on the operating system.

Syntax

```
Vts_ReadStyle (os_style_dir, space_name, fse_style_dir)
```

Argument	Purpose	Restrictions	Syntax
<i>os_style_dir</i>	Source directory. This is the pathname of an operating system directory in which you have stored modified Verity style files or the name of an individual style file in this directory.	Specified directory must exist on the operating system and should be a working directory. Wildcards are not permitted.	VARCHAR(128)
<i>space_name</i>	Name of the sbpace to which the style files are written.	The sbpace must exist. If you do not specify a space name here, the default sbpace is used.	VARCHAR(18)
<i>fse_style_dir</i>	Destination directory. Pathname of the FSE directory in which the FSE entries for the style files are created.	This name must be unique within the database.	VARCHAR(128)

Usage

Use **Vts_ReadStyle()** to store Verity style files as smart large objects and make them accessible to the Verity Text Search DataBlade module. **Vts_ReadStyle()** reads all the files in the specified working directory and writes them to smart large objects. Then it creates an FSE directory for these smart large objects that can be used to define contexts.

Important: *You must execute this routine within a transaction.*



If you are using the default style files and the default context, you do not need to use this routine.

Before you create smart large objects for your style files, you can create your own customized set of style files. Copy the default style files to a new directory and modify them with your editor to meet your requirements. Then, use **Vts_ReadStyle()** to read those style files that you want to use into a location where the Verity Text Search DataBlade module can use them in a search.

Always specify a new and unique FSE directory name with **Vts_ReadStyle()**. If an FSE directory with this name already exists and you want to replace it with a newer one, use **Vts_DropStyle()** and **Vts_DropContext()** to remove the old one first. Be sure that the context is not in use before you try to drop it.

Once you have set up your FSE directory, use **Vts_CreateContext()** to define a context using this directory or **Vts_AlterContext()** to modify a context to use this directory.

Once style files have been read into smart large objects in the database, you can use the routine **Vts_WriteStyle()** to write the files back to the operating system to the specified directory. The files can then be edited and later read back into the database using **Vts_ReadStyle()**.

The Verity Text Search DataBlade module uses several Verity style files when defining the default context. These style files are located in **\$INFOR-MIXDIR/extend/VTs.reln/vdkhome/common/style** (*reln* represents the release number of the Verity Text Search DataBlade module you are using). You can use these style files as a basis to create your own custom context.



Important: Do not modify the style files in the default directory. Copy the style files to a working directory before editing them.

For a list of style files installed with this DataBlade module and used by the default context, see [“Style Files You Can Modify” on page 6-6](#) and [“Style Files You Cannot Modify” on page 6-7](#).

You can create your own style files for words to be indexed, lexical preferences, and stopwords. See [“Style Files You Can Modify” on page 6-6](#) for more information.

Vts_ReadStyle() is a procedure and can be executed using the EXECUTE PROCEDURE statement.

Example

This example reads the Verity style files located in **\$INFORMIXDIR/extend/VTS.relno/vdkhome/common/style** and creates an FSE directory called **/mydir/fse_dir/styles1**. The style files are stored in the sbspace named **sbspace**.

```
BEGIN WORK;

EXECUTE PROCEDURE Vts_ReadStyle
(
  '$INFORMIXDIR/extend/VTS.1.10.UC1/vdkhome/common/style',
  'sbspace',
  '/mydir/fse_dir/styles1'
);

COMMIT WORK;
```

Related Routines

Vts_CreateContext() creates a new context.

Vts_AlterContext() associates a custom context with a different set of style files.

Vts_WriteStyle() writes all smart large objects in a specified FSE directory on the operating system.

Vts_DropStyle() deletes all smart large objects in a specified FSE directory.

See Also

[Chapter 6, “Verity Style Files”](#)

SEARCH'97 Developer's Kit Advanced Features Guide

Vts_Release() Function

Vts_Release() returns version information about the current DataBlade module.

Syntax

```
Vts_Release ( )
```

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

Usage

Use **Vts_Release()** to learn:

- 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.

Returns

An LVARCHAR data type containing the DataBlade module version information.

Example

The following example returns version information about the current Verity Text Search DataBlade module:

```
EXECUTE FUNCTION Vts_Release();
```


Vts_WriteStyle() Procedure

Vts_WriteStyle() writes all smart large objects in a specified FSE directory to a specified directory on the operating system.

Syntax

```
Vts_WriteStyle (os_style_directory, space_name, fse_style_dir)
```

Argument	Purpose	Restrictions	Data Type
os_style_dir	Destination directory. The pathname of the operating system directory to which the style files contained in the smart large objects are written.	Specified directory must exist. Wildcards are not permitted.	VARCHAR(128)
space_name	Name of the sbpace in which style files are stored.	The sbpace must exist. If you do not specify a space name here, the default sbpace is used.	VARCHAR(18)
fse_style_dir	Source directory. The pathname of the FSE directory in which the FSE entries for the style smart large objects are maintained.	FSE directory must exist and must contain at least one style file. Wildcards are not permitted.	VARCHAR(128)

Usage

Use **Vts_WriteStyle()** to write an individual smart large object or an entire FSE directory to a set of operating system files in a specified directory. You can then use a text editor to display and customize the style files.

Vts_WriteStyle() is a procedure and can be executed using the EXECUTE PROCEDURE statement.

Important: You must execute this routine within a transaction.



Example

This example writes the style files stored as smart large objects in the FSE directory **/mydir/fse_dir/styles1** to **\$INFORMIXDIR/extend/VTs.reln/vdkhome/common/style**:

```
BEGIN WORK;  
  
EXECUTE PROCEDURE Vts_WriteStyle (  
    '/mydir/stylewrite',  
    'sbspace',  
    '/mydir/fse_dir/styles1'  
);  
  
COMMIT WORK;
```

Related Routines

Vts_CreateContext() creates a new context.

Vts_AlterContext() associates a custom context with a different set of style files.

Vts_ReadStyle() creates an FSE directory from style files that currently reside on the operating system.

Vts_DropStyle() deletes all smart large objects in a specified FSE directory.

See Also

[Chapter 6, “Verity Style Files”](#)

SEARCH'97 Developer's Kit Advanced Features Guide

SQL Syntax Usage

In This Chapter	5-3
ALTER INDEX Statement	5-4
CREATE INDEX Statement.	5-5
FRAGMENT BY EXPRESSION	5-5
Index Key Specification	5-5
USING Clause.	5-6
Turning Logging Off	5-7
Operator Classes for Supported Data Types	5-7
Indexable Data Types	5-7
Creating an Index on an Indexable Data Type	5-8
Creating an Index on a Distinct Data Type	5-8
CREATE TABLE Statement.	5-10
DROP INDEX Statement	5-11
SELECT Statement	5-12
WHERE Clause	5-12
Select List	5-12

In This Chapter

This chapter discusses the SQL syntax specific to the Verity Text Search DataBlade module. It describes additions to or special uses of the following existing SQL statements:

- ALTER INDEX
- CREATE INDEX
- CREATE TABLE
- DROP INDEX
- SELECT

For complete syntax of these statements, see the [Informix Guide to SQL: Syntax](#).

ALTER INDEX Statement

Use the ALTER INDEX statement to modify a **vts** index.

The TO CLUSTER clause of the ALTER INDEX statement is not available for the **vts** access method. See the [Informix Guide to SQL: Syntax](#) manual for more information about ALTER INDEX.

CREATE INDEX Statement

Use the CREATE INDEX statement to create a **vts** index.

DataBlade-Specific Syntax

The following clauses of the CREATE INDEX statement are not available with the **vts** access method:

- CLUSTER
- DISTINCT
- FILLFACTOR
- UNIQUE

FRAGMENT BY EXPRESSION

You can use the FRAGMENT BY EXPRESSION clause for both tables and **vts** indexes. If you do not specify a fragmentation strategy during index creation, the index uses the same fragmentation strategy as the table whose column it indexes. If you do specify a fragmentation strategy during index creation, the index uses that strategy.

You cannot use the **vts_contains** operator within the FRAGMENT BY EXPRESSION clause.

Within the FRAGMENT BY EXPRESSION clause, you can specify the name of an sbpace in which to store the index. If you omit the sbpace name, the default sbpace is used.

Important: *The sbpace must have logging turned on. You cannot create an index in an unlogged sbpace.*

For more information on fragmentation strategies, see the [Administrator's Guide](#) for your database server.

Index Key Specification

For the Verity Text Search DataBlade module, the operator class part of the Index Key Specification clause is required.



USING Clause

Always specify **vts** as the secondary access method in the USING clause.

The Verity Text Search DataBlade module supports one index parameter: **NOLOG**. Set **NOLOG=TRUE** when you create an index.

Do not specify any other index parameters. This module uses contexts to customize indexes. You cannot customize indexes when you create them; you must customize the style files that define a context instead.

Usage

In the Verity Text Search DataBlade module, the CREATE INDEX statement creates an index on a single column. You can create indexes on any column whose data type is an indexable data type. (See the [“Operator Classes for Supported Data Types”](#) for a list of indexable data types.)

When the column being indexed is defined as a row data type, only those fields whose data types are indexable are indexed.

In this release of the Verity Text Search DataBlade module, you cannot index individual fields of row data types; you must index the entire row type. When you create an index on a row type column, all fields whose data types are indexable are automatically indexed.

When you create an index, the reference count of the context associated with the data type of the indexed column is increased by 1. In other words, 1 is added to the **refcount** column in the **Vts_Contexts** table whenever an index is successfully created. See [“Vts_Contexts Table”](#) on page C-3 for more information.

You cannot customize an index when you create it. Custom behavior for indexes is provided by Verity style files through contexts. This DataBlade module includes routines that assign style files to contexts. Contexts are associated with data types. Table columns and their indexes use the context associated with the data type of the column. See [Chapter 4, “DataBlade Routines,”](#) to learn about the routines that create and manage contexts.

Under certain conditions, two or more indexes can have the same context:

- The indexes are built on columns that have the same data type.

- The indexes are built on columns whose data types share the same context.

Turning Logging Off

Set logging off when you create indexes.

Creating indexes with logging on generates a large number of log records and can take a long time. Setting NOLOG to TRUE creates an index without logging the smart large object activity in the transaction. Logging is restored for subsequent activities.

After you create an index, perform a level 0 archive. See the [Archive and Backup Guide](#) for more information.

Operator Classes for Supported Data Types

The Verity Text Search DataBlade module uses the **vts** access method and its associated operator classes, which are supplied with the product.

Indexable Data Types

Each data type on which the Verity Text Search DataBlade module can do an index search has an operator class defined for it. The indexable data types and the related operator classes are listed in the following table.

Indexable Data Type	Operator Class
BLOB	Vts_blob_ops
CLOB	Vts_clob_ops
CHAR	Vts_char_ops
IfxDocDesc	Vts>IfxDocDesc_ops
LVARCHAR	Vts_lvvarchar_ops

(1 of 2)

Indexable Data Type	Operator Class
VARCHAR	Vts_varchar_ops
Any named row type	Vts_row_ops
A distinct type based on another indexable type	The operator class of its source type

(2 of 2)

Creating an Index on an Indexable Data Type

When you create a **vts** index on an indexable column of a table, you must specify the operator class that corresponds to the data type of the column being indexed. You do this in the Index Key Specification clause of the CREATE INDEX statement. Use the preceding table to determine the correct name of the operator class.

The following example uses a table named **movies** that has a column named **review**, whose data type is BLOB. The statement creates an index named **movie_idx** on the **review** column of the **movies** table:

```
CREATE INDEX movie_idx ON movies (review Vts_blob_ops)
  USING vts (NOLOG='TRUE')
  IN sbpace;
```

Creating an Index on a Distinct Data Type

When you create an index on a distinct data type, use the operator class of the data type on which it is based in its CREATE DISTINCT TYPE statement.

For example, if you have a column named **cast** whose data type is a distinct type named **actor** that is based on VARCHAR, your CREATE INDEX statement looks like the following example:

```
CREATE INDEX movie_idx ON movies (item Vts_row_ops)
  USING vts (NOLOG='TRUE')
  IN sbpace;
```

For more examples, see [“Creating Row and Distinct Data Types” on page 2-7](#).

See [Informix Guide to SQL: Syntax](#) for more general information about access methods, operator classes, and the CREATE INDEX statement.

CREATE TABLE Statement

Use the CREATE TABLE statement to create a table that contains a column whose data type is of one of the supported indexable data types.

When you create a table, you must plan your columns with index searches in mind. Any columns that will have index searches must be of an indexable data type. Indexable data types are listed in [“Operator Classes for Supported Data Types” on page 5-7](#). If a column has a row data type, any fields in that column that you want to search should also have an indexable data type.

Only fields or columns of indexable data types can be indexed for searching with the Verity Text Search DataBlade module.

In this module, a SELECT statement can search on only one indexed column in a table. To get around this limitation, use row data types. All indexable fields of a row type column are searched.

DROP INDEX Statement

Use the DROP INDEX statement to drop an existing **vts** index.

In addition to the effects documented in the [Informix Guide to SQL: Syntax](#), DROP INDEX has the following effect: the reference count of the context associated with the data type of the indexed column is decreased by 1. In other words, 1 is subtracted from the **refcount** column in the **Vts_Contexts** table whenever an index is successfully dropped. See “[Vts_Contexts Table](#)” on page C-3 for more information.

SELECT Statement

Use the SELECT statement to perform searches.

DataBlade-Specific Syntax

You can use DataBlade-specific syntax in the SELECT statement in the WHERE clause and in the select list.

WHERE Clause

Use the **vts_contains** operator one or more times in the WHERE clause to specify the column to be searched, the search criteria, and, optionally, the score and other information returned from the search. See [“vts_contains Operator” on page 4-9](#) for more information.

You can use the **vts_contains** operator more than once in a single WHERE clause if your table has different indexes on more than one column. However, the first instance of **vts_contains** uses a sequential scan instead of an index scan, thus slowing the query. If you want to index on a key that includes more than one column of a table, create a row data type that contains all the columns that you want to search. Then create a table with a column of this row data type and move your data into this table.

Select List

You can specify a DataBlade-specific function in the select list, such as **Vts_GetHighlight()**. See [“Vts_GetHighlight\(\) Function” on page 4-31](#) for more information.

Usage

The Verity Text Search DataBlade module does not allow “dirty reads.” A dirty read occurs when a table row has been modified or deleted, but the transaction has not been committed.

If your isolation level is dirty read, you can change it by issuing a SET ISOLATION statement before your SELECT statement:

```
SET ISOLATION TO COMMITTED READ;  
SELECT item.title FROM movies  
      WHERE vts_contains (review, 'excellent');
```

See the [Informix Guide to SQL: Syntax](#) manual to learn more about isolation levels.

Verity Style Files

In This Chapter	6-3
Overview of Style Files	6-3
Style Files That You Can Create	6-5
Style Files You Can Modify.	6-6
Style Files You Cannot Modify	6-7
style.go	6-8
style.lex	6-9
style.prm	6-17
style.sfl	6-21
style.stp	6-22



In This Chapter

The Verity Text Search DataBlade module uses style files to specify the characteristics of indexes and searches and to provide locale-specific information. This chapter provides instructions for creating and modifying the most commonly used style files. To learn how to use style files to create new contexts, refer to [“Vts_ReadStyle\(\) Procedure” on page 4-35](#).

The material in this chapter is adapted from Verity’s on-line support FAQ and documentation for the Verity search engine, especially the *SEARCH’97 Developer’s Kit Advanced Features Guide* and the *SEARCH’97 Collection Building Guide*.

Important: *Never modify the style files in the default directory. Copy the style files to a working directory before editing them.*

For more details about the style files discussed in this chapter, and for a detailed description of additional style files, refer to the Verity manuals provided with this product in `$INFORMIXDIR/extend/VTS.relno/vdkhome/doc/html`. You can also find instructions on the Verity Web site.

Overview of Style Files

The Verity Text Search DataBlade module uses style files to specify the parameters used by a context. You can define a new context that uses customized style files to change the configuration options for your indexes and searches.

Default style files are provided with this module. These style files are located in `$INFORMIXDIR/extend/VTS.relno/vdkhome/common/style` directory (where *relno* represents the release number of the Verity Text Search DataBlade module you are using).

The default style files provided with this module are set to optimize the product's performance. Informix recommends that you use the default settings for most of these files. A few style files, such as **style.prm**, can be changed to accommodate features such as highlighting, summarization, and clustering. You can add the files **style.go** (to index specialized vocabularies), **style.lex** (to specify nonalphanumeric characters to be indexed), and **style.stp** (to specify words to be omitted from indexes).

To learn about contexts, see [“Contexts and Style Files” on page 1-10](#).

There are three categories of style files:

- **Style files you can create.** These files are not provided with the product but are useful in customizing indexes. These files are documented in this chapter.
- **Style files you cannot modify.** These files are not documented in this chapter.
- **Style files you can modify.** While you are permitted to modify all these files, Informix recommends against modifying some of them. Files that Informix recommends you do not change are not documented here. Files that are useful in configuring indexes are documented.

Style Files That You Can Create

You can create new style files for words to be indexed, lexical preferences, and stopwords. None of these style files are included in the default context. These files are described in the following table.

Filename	Description	Required?
style.go	Contains the list of the words to be included in index: for example, words specific to a particular industry or science discipline. If a style.go file is included in a style directory, only those words specified in this file are indexed.	Optional
style.lex	Specifies nonalphanumeric characters that are to be interpreted as valid characters in words stored in a index. Also defines locale-specific preferences. If a character is not specified in a style.lex file, or if there is no style.lex file, nonalphanumeric characters are not recognized as part of words.	Optional
style.stp	Specifies words to be excluded from indexes.	Optional

Style Files You Can Modify

The following table describes each configurable style file installed with this DataBlade module and used by the default context. While you are allowed to change these files, Informix recommends against modifying any files in this list except for **style.prm**. You can comment and uncomment lines in **style.sfl** and **style.uni**. The default settings for the rest of the files are specifically set for this DataBlade module.

Filename	Description	Required?
style.dft	Instructs the Verity Text Search DataBlade module to use the universal filter. Informix recommends against modifying this file. The universal filter includes filters for all supported formats: ASCII, HTML, PDF, and the Microsoft Office file formats.	Required
style.plc	Specifies the indexing mode to be used. You can optimize the way documents are indexed by changing the settings in this file.	Optional
style.prm	Enables and disables index schema features. Determines the format and contents for a full-word index. This file is included in all other style files.	Optional
style.sfl	Includes field definitions for the Verity standard fields. By default, these fields are populated by the universal filter.	Required
style.ufl	Defines the fields to be included in the internal documents table.	Optional
style.uni	Tells the universal filter which helper filters to load in what order for every possible document type.	Required
style.vgw	Describes how external data is stored and how to access it. Used with data stored outside the database.	Optional

Style Files You Cannot Modify

The style files listed in the following table are provided with the Verity Text Search DataBlade module default context and must not be changed.

File Name	Description	Required?
style.ddd	Contains the default schema definition for a collection's documents table. Defines supported field types; contains syntax for all statements, keywords, and modifiers that can appear in a file; and defines which collection fields are to be indexed.	Required
style.did	Specifies additional data to be stored in the index. Includes information about the format and contexts for a full-word index.	Optional
style.ngm	Defines N-gram index. This enables fast wildcard searches and fuzzy searches.	Optional
style.pdd	Defines the format of the partition management structure.	Optional
style.sid	Describes how topic sets are indexed, enabling fast topic searches.	Optional
style.wld	Determines which word-assist indexes are built.	Optional



style.go

The **style.go** file contains words you want to include in your word searches and therefore in your **vts** indexes.

***Important:** If you include a **style.go** file in a style directory, only those words specified in this file are included in the **vts** indexes. No other words are indexed.*

A **style.go** file is optional and is used only in specialized situations. For example, if you want to search press releases for particular terms or phrases, list the words you want indexed in a **style.go** file. All other words are ignored.

No **style.go** file is shipped with the Verity Text Search DataBlade module. If you need an included word list, you must create a new **style.go** file.

Syntax

A **style.go** file is a flat ASCII file made up solely of words to be indexed. You can use regular expressions to represent words or word groups. Regular expressions are summarized in [Appendix B, “Regular Expressions.”](#)

style.lex

In the Verity Text Search DataBlade module, nonalphanumeric characters are not normally recognized as parts of words. Only alphanumeric characters are indexed, so only alphanumeric characters are returned by queries.

If you want to use nonalphanumeric characters as search criteria, you must specify each of these characters in a **style.lex** file. Typical nonalphanumeric characters in a **style.lex** file include punctuation, white spaces, line break characters, and symbols used in names, such as “&” and “/.”

If a **style.lex** file is present in the style directory, **vts** indexes use only characters specified in the **style.lex** file. Therefore, the **style.lex** file must include a statement specifying the set of alphanumeric characters as well as the special characters you want indexed.

The **style.lex** file is only for 8-bit locales. This file overrides any other specifications, including the locale and the default lexer.

Syntax

A **style.lex** file uses regular expressions to describe the additional characters to be indexed. See [Appendix B, “Regular Expressions,”](#) to learn how to use regular expressions.

To create a style.lex file

1. Create the first noncomment lines exactly as shown here:

```
$control:1  
lex:
```

2. Follow the **lex:** statement with either of the following kinds of statements:

define	Specifies macros used in token statements.
token	Specifies the words, symbols, spaces, and so forth to be included in vts indexes.

Symbols Used in style.lex

You can use the following symbols to create the macro and token definitions in the **style.lex** file.

Symbol	Symbol Name	Description
' '	Single and double quotation marks (quotes)	Specifies the elements that make up the define statement macro or token statement definition.
[]	Brackets	Defines a character class.
{ }	Braces	Specifies a macro that has been specified in a define statement.
+	Plus	Specifies one or more occurrences of a combination of characters or numbers.
*	Asterisk	Specifies zero or more occurrences of a combination of characters or numbers.
\	Single backslash	Indicates that the alternative meaning of the succeeding character is to be used. For example, the character <code>␣</code> is usually interpreted as the letter "t", but when it follows a backslash, it is interpreted as a tab.
\\	Double backslash	Represents the literal backslash character. Because a single backslash is interpreted as an instruction, you must tell the search engine when a backslash is to be read as a backslash character and not as an instruction. You must use double backslashes to represent the backslashes in PC pathnames. For example, to specify the directory C:\temp\mail , type: <code>C:\\temp\\mail</code>
#	Pound sign	Specifies that the characters that follow a comment.

Define Statements

A **define** statement in the **style.lex** file specifies a macro to be used in the **token** statements that follow it. You can give these macros any name you choose.

When you use **define** statement macros in **token** statements, you must enclose the macro in braces.

Use of **define** statements is optional.

Using Define Statements to Define Characters

The following statement defines ALPHANUM to represent all alphanumeric characters. This is the default setting:

```
DEFINE: ALPHANUM "[A-Za-z0-9]"
```

To add characters to the ALPHANUM character set, type them inside the brackets. For example, the following statement adds the ampersand (&), the forward slash (/), and the apostrophe (') to the ALPHANUM character set:

```
DEFINE: ALPHANUM "[A-Za-z0-9&' /]"
```

With this ALPHANUM definition, the following terms are recognized:

- women's
- OS/2
- S&P500



Tip: The **define** statement shown here also returns words that have an ampersand at the beginning or the end: for example, ATT&.

Using Define Statements to Define Spaces

Another typical use for a **define** statement is to define spaces. The following statement instructs the search engine to interpret tabs, form feeds, carriage returns, and vertical tabs as space

```
define: WHITESPC "[ \t\f\r\v]"
```



Token Statements

Each **token** statement contains a flag identifying tokens such as end-of-sentence, end-of-paragraph, tab, and space.

Token statements can include regular expressions and macros defined in the **define** statements in the same **style.lex** file.

You can use a token in more than one statement. For performance reasons, however, it is a good idea to limit the number of WORD tokens in your **style.lex** file.

Tip: ***Token** statements are interpreted in the order in which they appear in **style.lex**. If a query string satisfies more than one token, it is treated as a match for the first token whose condition it satisfies. For this reason, it is a good idea to make the PUNCT token the last statement in the file.*

Tokens

Tokens are listed in the following table. [“A Sample style.lex File” on page 6-15](#) shows how these tokens look in a **style.lex** file.

Token	Description
EOS	<p>An end-of-sentence character. Typically, the following characters are defined:</p> <ul style="list-style-type: none"> ■ Period (.) ■ Question mark (?) ■ Exclamation point (!)
NEWLINE	<p>A line break. Typically, this is represented by the newline character: <code>\n</code></p> <p>It can also be represented by the following string, which uses the WHITESPC macro: <code>{WHITESPC}*\n</code></p>
PARA	<p>The end of a paragraph. This is typically represented by two or more newline characters: <code>[\n\n]+</code></p> <p>It can also be represented by the following string, which uses the WHITESPC macro: <code>{WHITESPC}*\n({WHITESPC}*\n)+</code></p>
PUNCT	<p>Characters that are not indexed.</p> <p>To indicate that any nonalphanumeric character except characters defined in the style.lex file should be considered punctuation, use a period: <code>[.]</code></p> <p>When you use a period to represent punctuation, it is a good idea to put the PUNCT statement at the end of your style.lex file.</p> <p>Another way to specify these terms is to use the NOT operator (<code>[^]</code>) and list all the characters that you want indexed after the <code>^</code> character. See the sample style.lex file for an example.</p> <p>Using the NOT token is more expensive at indexing time.</p> <p>Important: Be sure to enclose the <code>^</code> symbol inside straight brackets. Otherwise, it may be interpreted as a beginning-of-line character.</p>

(1 of 2)



Token	Description
TAB	A tab. Typically, this is represented by the tab character: <code>\t</code>
WHITE	A blank space, represented by one or more spaces. You can list the spaces or define them in a define statement and refer to that statement as follows: <code>{WHITESPC}+</code>
WORD	A word or a regular expression that represents one of the following types: <ul style="list-style-type: none"> ■ A word, represented as any string composed of alphanumeric characters (including uppercase and lowercase letters) ■ A floating-point decimal <p>The following expression uses the ALPHANUM macro:</p> <code>{ALPHANUM}+(\.\{ALPHANUM}\.)*</code> <p>The next expression interprets a word as a floating-point decimal:</p> <code>[0-9]+\.\.[0-9]+</code> <p>Minimize your use of WORD statements. A large number of WORD statements slows down the indexing process.</p>

(2 of 2)

Statement Interpretation

If the **style.lex** file contains two or more statements for the same kind of token, the search engine uses the first token that matches the search pattern and ignores the others.

For example, if your **style.lex** statement contains both of the following **token** statements, the search engine interprets a word as being one of the following values:

- A string, composed of alphanumeric characters (both uppercase and lowercase) and numeric characters:
token: WORD "[A-Za-z0-9]+"
- A numeric value with a floating-point decimal:
token: WORD "[0-9]+\.\.[0-9]+"

A Sample style.lex File

A sample default **style.lex** file is shown here:

```
$control: 1
lex:
{
    define: WHITESPC "[ \t]"
    define: NEWLINE "{WHT}*\\n"

    token:  WORD      "[A-Za-z0-9]+"      #word
    token:  WORD      "[0-9]+\\.[0-9]+"    #word
    token:  EOS        "[.?!]"            #end of sentence
    token:  NEWLINE    "{NL}"              #single end-of-line
    token:  PARA       "{NL}{NL}"         #end of paragraph
    token:  WHITE      "{WHT}"             #whitespace
    token:  PUNCT      "."                 #all other text
}
$$
```

To include the hyphen (-), forward slash (/), and apostrophe, change the first WORD token to read "[A-Za-z0-9- ' /]+"

Once you are familiar with the use of **style.lex**, you can use it to hone your indexes more precisely. For example, to index ampersands (&) that are in the middle of a word but not ampersands that appear at the beginning or the end of a word, follow these steps:

1. Define three macros to represent the beginning, middle, and end of words:

```
define: BEGINCHARS "[A-Za-z0-9']"
define: MIDCHARS "[A-Za-z0-9&]"
define: ENDCHARS "[A-Za-z0-9']"
```

2. Define the WORD token using these macros:

```
token: WORD "{BEGINCHARS}+{MIDCHARS}+{ENDCHARS}+"
```

If you use these statements in your **style.lex** file, the Verity Text Search DataBlade module indexes “AT&T” but not “&ATT” or “ATT&”.

Character Mapping

The default **style.lex** file does not index 8-bit characters, even though they are valid in English-language documents. In addition, the character set for the default **style.lex** file is the internal character set even if you set everything else to a different code page. To specify a different character set, add the **Scharmap** option to the **style.lex** file, as shown:

```
$control: 1
$charmap: 8859
lex:
{
  [...]
}
$$
```


style.prm

Use the **style.prm** file to include optional data in **vts** indexes. Use this file if you want to use specialized features such as clustering and summarization, SOUNDEX data, and highlight data.

The default **style.prm** file contains descriptions of the default settings and alternatives. It also contains examples of different settings and how they affect the results. Syntax information is provided here.

Syntax

To modify the **style.prm** file, change the **define** statements, using the syntax described in this section. The **define** statements used in the **style.prm** file determine what results are returned and how results are presented.



***Tip:** Some of the **define** statements are commented out. Be sure to uncomment these statements (delete the **#** character at the beginning of the line) if you want a **define** statement to affect the output.*

Define Statements

The **define** statements always begin with a dollar sign (\$) followed by the word **define**, followed by keywords and parameters:

```
$define xxx [xxx] [xxx]
```

Optional parameters are shown in square brackets.

The **define** statements in the following table can be used in the **style.prm** file.

Define Statement	Parameters and Possible Values	Description
IDX-CONFIG (Defines the storage format used to encode a term's position in an index.)	WCT [MANY]	Stores word count. Records the ordinal counting position of the word from the beginning of the document. WCT is the default setting. Specifying MANY increases accuracy but uses more disk space and can affect performance.
	PSW [MANY]	Stores location of paragraphs, sentences, and words. Records semantically accurate paragraph and sentence boundaries and counts the number of paragraphs and sentences that match the search string, as well as the number of times the string occurs. The PSW option uses 15% to 20% more disk space and can affect performance. Specifying MANY increases accuracy but uses more disk space and can affect performance.

(1 of 3)

Define Statement	Parameters and Possible Values	Description
WORD-IDXOPTS	[Stemdex]	<p>Enables stemming. Stem variants of words are included in the index. For example, if you specify the word “hope,” the words “hoped,” “hopes,” “hoping,” “hopeful,” “hopefulness,” and other variants are also indexed.</p> <p>If you specify the word “nation,” a stemmed index also contains “national,” “nationality,” “nationalism,” and “nationhood,” among others.</p> <p>Stemdex is the default setting.</p>
	[Casedex]	<p>Enables case sensitivity. Allows case-sensitive searching in the following cases:</p> <ul style="list-style-type: none"> ■ The search term is in mixed lettercase (for example, Spring). ■ The search term is preceded by the <CASE> modifier. <p>Queries where the search terms are in all uppercase or all lowercase returns all case variants of the search term. For example, if the search term is spring, the variants spring, SPRING, and Spring are returned.</p> <p>Casedex enabled is the default setting.</p> <p>When Casedex is omitted, all searches are case-insensitive and the <CASE> modifier has no effect.</p>
	[Soundex]	<p>Enables phonetic representations. That is, words that sound like or have letter patterns similar to the specified word are included in the index. Only words beginning with the same letter as the specified word are indexed.</p> <p>For example, for the surname Wilde, the alternatives Wild, Wilder, Wildey, and Wolde would be indexed, among others.</p> <p>For example, if the word “face” is specified, the words “fast,” “faith,” “facial,” and “fist” are indexed, but “phase” is not.</p>
ZONE-IDXOPTS	See WORD-IDXOPTS	
ATTR-IDXOPTS	See WORD-IDXOPTS	
DOC-FEATURES	TF	<p>Generates and stores keyword lists at indexing time.</p> <p>This statement is required for clustering.</p>

Define Statement	Parameters and Possible Values	Description
DOC-SUMMARIES	<i>summary_type</i>	<p><i>summary_type</i> can be any one of the following values:</p> <ul style="list-style-type: none">XS Extracts the best sentences from the document and stores them in the index. Takes the optional <i>maxparm</i> parameter.LS Extracts the first few sentences from the document and stores them in the index. Takes the optional <i>maxparm</i> parameter.LB Extracts the first <i>num</i> bytes from a document and stores them in the index, compressing white space. Takes the optional <i>maxparm</i> parameter for MaxBytes only. <p>This statement is required for summarization.</p> <p>For example, the following statement stores the best three sentences of the searched document, up to a length of 500 bytes:</p> <pre>\$define DOC-SUMMARIES "XS MaxSents 3 MaxBytes 500"</pre>
	[MaxBytes <i>num</i>]	<p>Maximum size of a summary, in bytes.</p> <p>The default is 400.</p> <p>This parameter is optional and follows <i>summary_type</i>.</p>
	[MaxSents <i>num</i>]	<p>Maximum number of sentences in a summary.</p> <p>The default is 2.</p> <p>This parameter is optional and follows <i>summary_type</i>.</p>
	[TruncSent <i>num</i>]	<p>Maximum length of any sentence in the summary, in bytes.</p> <p>The default is 400.</p> <p>This parameter is optional and follows <i>summary_type</i>.</p>

style.sfl

The **style.sfl** file defines the standard fields in a document file, such as title, author, character set, and date. It also specifies which filters can be used to fill each of these standard fields.

All filters shipped with the Verity Text Search DataBlade module are included in **style.sfl**. You do not need to add more filters.

The size of the index, and therefore performance of your queries, is affected by the number of filters specified in this file. For that reason, it is a good idea to comment out any filters that you do not use. Refer to the comments in the **style.sfl** file to determine whether or not to comment out a line.

For example, the PageMap line is required to perform highlighting in PDF documents. If you do not use documents in PDF format, comment out the following line and all other lines that use only the flt_pdf filter:

```
varwidth: PageMap      _sv
```

The following fields are commented out by default:

```
#varwidth: Ext          _sv
#varwidth: URL          _sv
#varwidth: _Created     _sv
#varwidth: _Modified    _sv
#fixwidth: Created      4 date
#fixwidth: Modified     4 date
#fixwidth: Size         4 unsigned-integer
```

If you want to be able to fill in fields with URL information, uncomment these lines.



style.stp

The **style.stp** file contains *stopwords*—those terms that you want to exclude from a word search, and therefore from your **vt**s indexes.

A **style.stp** file is optional. It is rarely used. Its main purpose is for excluding rare constructs that look like words in documents (such as the 70-character "words" starting with M found in encoded files and hexadecimal strings.)

***Important:** In general, excluding common words, such as “the,” “and,” and “of” is not recommended because any queries that contain these stopwords, alone or in phrases, does not work. For example, if the word “of” is listed in **style.stp**, a search for “Joan of Arc” fails. If you implement such a change to the stop file, you should warn your users.*

No **style.stp** file is shipped with the Verity Text Search DataBlade module. If you want to list stopwords, you must create a new **style.stp** file.

***Important:** This style file has no effect when indexing HTML files or when a table scan is performed.*

Syntax

A **style.stp** file is a flat ASCII file made up entirely of a list of excluded words. You can use regular expressions in your **style.stp** file. Regular expressions are summarized in [Appendix B, “Regular Expressions.”](#)

Follow these rules when creating or modifying **style.stp**:

- A separate word must appear on each line.
- The words in the list can appear in any order.
- The word list must be left justified.

Case Sensitivity

The Verity Text Search DataBlade module indexes all case variants of a search term if the term is entered in all uppercase or all lowercase letters. For example, specifying either `spring` or `SPRING` as your search term returns the following variants: `spring`, `SPRING`, `Spring`, and any other combination of uppercase and lowercase letters found in the document.

If the search term is in mixed lettercasing, only the specified variant is retrieved. For example, the search string `Spring` returns words that have only initial capital letters: `Spring`. The variants `spring` and `SPRING` are not returned.

Because the Verity Text Search DataBlade module sometimes returns all case combinations, you must specify every case variant for words you want to exclude in your **style.stp** file. For example, if you want to stop both `html` and `HTML`, you must include both entries in your **style.stp** file.

Examples Using Regular Expressions

The following examples use regular expressions to exclude classes of words from the **vt**s index:

<code>[a-zA-Z]</code>	Excludes every one-letter word appearing in your documents from appearing in your vt s indexes.
-----------------------	--

<code>[0-9.][0-9.][0-9.][0-9.][0-9.]+</code>	Excludes all numbers longer than 4 digits.
--	--

<code>.....+</code>	Excludes all words greater than 30 characters.
---------------------	--

<code>^M.....+</code>	Excludes lines encoded with uuencode.
-----------------------	---------------------------------------

A Sample style.stp File

The following file excludes uuencoded lines, all one-letter words, and all versions of “California”:

```
[0-9a-zA-Z]
^M.....+
California
CALIFORNIA
california
CA
Cal.
CAL.
Calif
CALIF
```


Verity Operators and Modifiers

This appendix describes the Verity operators and modifiers you can use to customize your queries using the Verity Text Search DataBlade module.

This appendix contains information originally published in the Verity manual *Introduction to Topics*. This information is also on the Verity Web site at <http://search97.verity.com/>. At the Web site, enter the terms `operators` and `modifiers` in the search box. The Verity search engine returns a list of Web pages that describe Verity operators and modifiers and their use.

This appendix contains the following sections:

- “[Verity Query Language Summary](#),” next
- “[Elements of Query Expressions](#)” on page A-7
- “[Operator Reference](#)” on page A-11
- “[Modifier Reference](#)” on page A-27
- “[Natural Language Operators](#)” on page A-30
- “[Score Operators](#)” on page A-32

Verity Query Language Summary

The Verity Query Language consists of operators and modifiers that specify logic to be applied to a search element. This logic defines the qualifications that a document must meet to be retrieved. Modifiers extend the logic applied by operators.

Operators are classified by their type. This section contains the following subsections to describe each type of operator:

- [“Evidence Operators,”](#) next
- [“Proximity Operators”](#) on page A-4
- [“Concept Operators”](#) on page A-5
- [“Score Operators”](#) on page A-6
- [“Natural Language Operators”](#) on page A-6
- [“Modifiers”](#) on page A-7

In a query, use an operator to the left of the term to which it applies: for example: <SOUNDEX> Smith. See [“Performing Text Searches”](#) on page 2-17 for examples.

Evidence Operators

Evidence operators are used to specify either a *basic word search* or an *intelligent word search*. A basic word search finds documents that contain only the word or words specified in the query. An intelligent word search expands the query terms to create an expanded word list so that the search returns documents that contain variations of the query terms. For example, the THESAURUS operator in an intelligent word search selects documents containing the word specified plus its synonyms.

Documents retrieved using evidence operators are not relevance-ranked unless you use the MANY modifier. See [“MANY” on page A-28](#) for information. The following table describes each evidence operator.

Operator Name	Description
SOUNDEX	<p>Expands the search to include words that sound like or have a letter pattern similar to the specified word.</p> <p>The Soundex parameter of the WORD-IDXOPTS statement must be specified in your style.prm file.</p>
STEM (' ')	<p>Expands the search to include variations of the specified word, as well as the word itself. Single quotation marks achieve the same result.</p> <p>Stemming is the default for simple searches.</p> <p>The Stemdex parameter of the WORD-IDXOPTS statement must be specified in your style.prm file.</p>
THESAURUS	Expands the search to include synonyms of the specified search term.
TYPO/ <i>n</i>	Expands the search to include words similar in spelling to the query term (like “typos,” meaning typing mistakes.)
WILDCARD (*, ?)	<p>Selects documents that contain matches to a character string containing one of the following wildcards:</p> <ul style="list-style-type: none"> ■ An asterisk (*), representing any number of characters ■ A question mark (?), representing a single character
WORD (“ ”)	Selects documents that contain one or more instances of the specified term, spelled exactly as entered. Double quotation marks achieve the same result.

Proximity Operators

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

Operator Name	Description
PHRASE	Selects documents that include a phrase you specify. A phrase is a grouping of two or more words that occur in a specific order.
SENTENCE	Selects documents that include all of the words you specify within a sentence.
PARAGRAPH	Selects documents that include all of the search elements you specify within a paragraph.
NEAR	Selects documents containing specified search terms within close proximity to each other. The closer the search terms are within a document, the higher the document's score.
NEAR/ <i>n</i>	Selects documents containing two or more words within <i>n</i> number of words of each other, where <i>n</i> is an integer up to 1000. The closer the search terms are within a document, the higher the document's score.

Concept Operators

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

Operator Name	Description
AND	Selects documents that contain all of the search elements.
OR	Selects documents that contain at least one of the search elements.
ACCRUE	Selects documents that include at least one of the search elements. The more unique matches found, the higher the score.

To use the concept operators with two terms, enter the first term, the operator, and then another term, as follows:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'computers  
<ACCRUE> laptops');
```

To use concept operators with more than two terms, enter the operator, and then list the terms in parentheses, separated by commas, as follows:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<ACCRUE>  
(computers, laptops, smartcards)');
```

Score Operators

You can retrieve a score that indicates how relevant a document is to your query. Score operators determine how the score of a document is calculated. Some score operators apply to the score of individual search terms, while others determine how the scores of each search term are combined to produce a document score. The following table describes the score operators.

Operator Name	Description
COMPLEMENT	Calculates document scores by taking the complement (subtracting from 1) of the combined scores for individual search terms.
PRODUCT	Calculates document scores by multiplying the scores for individual search terms.
SUM	Calculates document scores by adding together, to a maximum of 1, the scores for the individual search terms.
YESNO	Forces a Boolean result for each search term to which it is applied. Search terms preceded by the YESNO operator score 1 if the term is found in the document, and 0 if it is not.

Natural Language Operators

Natural language operators enable you to specify search criteria using natural language syntax. The search engine uses natural language analysis to translate the query text into Verity query language expressions for evaluating and scoring documents. The FREETEXT and LIKE natural language operators are intended mainly for use by application developers.

Operator Name	Description
FREETEXT	Interprets text using the free text query parser and scores documents using the resulting query expression.
LIKE	Searches for other documents similar to the sample documents or text passages you provide.

Modifiers

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

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

Elements of Query Expressions

A *query expression* is a statement you enter as criteria for performing a search. The words and operators you use in a query expression are its *elements*. The following illustration shows the parts of a query expression.

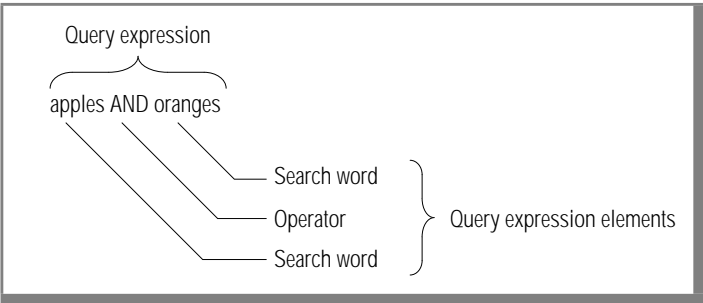


Figure A-1
Query Expression Elements

Simple and Explicit Syntax

You can state a query expression using *simple* or *explicit* syntax. The syntax you use determines whether the search words you enter are stemmed and whether the words that are found contribute to relevance-ranked scoring.

Simple Syntax

When you use simple syntax, the search engine interprets single words you enter as if they were preceded by the MANY modifier and the STEM operator. By implicitly applying the MANY modifier, the search engine calculates each document's score based on the *word density* it finds; the denser the occurrence of a word in a document, the higher the document's score.

As a result, the search engine relevance-ranks documents according to word density as it searches for the word you specify (including words that have the same stem. For example, "films," "filmed," and "filming" are stemmed variations of the word "film.") To search for documents containing the word "film" and its stem words, enter the word `film` without quotation marks; this syntax is simple syntax:

```
film
```

When documents are relevance-ranked, they are listed in an order based on their relevance to your search criteria. Relevance-ranked results are returned with the most relevant documents at the top of the list.

Explicit Syntax

When you enclose individual words in double quotation marks, the Verity search engine interprets those words literally. For example, if you specify the word "film" in double-quotation marks, the words "films," "filmed," and "filming" are not considered in the search. To select documents containing the word "film" without searching for its stemmed words, enter the word `film` using explicit syntax, as follows:

```
SELECT title FROM vtstest WHERE Vts_contains(text, "film");
```


The following example retrieves documents that contain both the literal phrase “pharmaceutical companies” and the literal word “stock.” The AND operator does not require angle brackets, because it is automatically interpreted as an operator.

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'AND  
("pharmaceutical companies", "stock"))');
```

The following example retrieves documents containing the phrase “black and white.” The PHRASE operator does require angle brackets, and the word “and” is enclosed in double quotation marks because it is to be interpreted as a literal word, not as an operator.

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<PHRASE>  
(black "and" white)');
```

For example, the following query uses simple syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<PHRASE>  
(format a date)');
```

The following query uses explicit syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<PHRASE>  
("format", "a", "date")');
```

Using English Operators in Other Locales

If you are using a locale other than English, and your application has defined locale-specific names for the query language in the message database, users can use English query language to compose their queries. To specify English query language, you need to enclose the English operator or modifier name between angle brackets (< and >).

For example, to search for the words “parle” and “vous,” you can use either of the following queries:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'parle et
vous');

SELECT title FROM vtstest WHERE Vts_contains(text, 'parle
<AND> vous');
```

In the first example, the **et** operator is the French translation of the AND operator.

Syntax Options

The Verity query language provides a few alternatives you can use to specify evidence operators. In the following table, *word* represents the word to be located.

Standard Query Expression	Equivalent Format
<MANY><WORD> <i>word</i>	"word"
<MANY><STEM> <i>word</i>	'word'

Operator Precedence Rules

The Verity search engine uses rules of precedence to determine how operators are assigned. These rules affect how documents are selected.

The following table describes how precedence rules apply to operators.

Operator	Precedence	How Precedence Is Determined
AND OR ACCRUE	Highest precedence	The concept operators take precedence over the other operators.
ALL PARAGRAPH SENTENCE NEAR NEAR/ <i>n</i> PHRASE ANY	Incremental precedence (in descending order)	The proximity operators refer to incremental ranges that exist within a document. A phrase takes precedence over a word; a sentence takes precedence over a phrase or a word; and a paragraph takes precedence over a sentence, a phrase, or a word.
WORD STEM SOUNDEX WILDCARD THESAURUS	Lowest precedence	The evidence operators have the lowest precedence. These operators all have the same precedence.

Operator Reference

This section describes each Verity operator in detail. Where appropriate, each description includes an example of simple syntax and explicit syntax. Operators are listed alphabetically.

ACCRUE

The ACCRUE operator selects documents that include at least one of the search elements you specify. Valid search elements are two or more words or phrases.

Documents retrieved using the ACCRUE operator are relevance-ranked.

ACCRUE can be combined with other operators. The rules of precedence for combining operators are explained in [“Operator Precedence Rules” on page A-11](#).

Simple Syntax

You can combine the search elements and the ACCRUE operator in any of the orders shown next. These examples all select documents containing stemmed variations of the words “computers” and “laptops.”

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'computers
<ACCRUE> laptops');
```

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'computers, laptops');
```

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<ACCRUE>
(computers, laptops)');
```

Explicit Syntax

The following example selects documents containing any of the literal words “IBM,” “Apple,” and “Sun”:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<ACCRUE>
("IBM","Apple","Sun")');
```

Because the words “IBM,” “Apple,” and “Sun” are enclosed in double-quotation marks, no stemming is done, and documents containing these words are not relevance-ranked.

AND

The AND operator selects documents that contain *all* of the words or phrases you specify. Documents retrieved using the AND operator are relevance-ranked. AND can be combined with other operators, following the rules of precedence explained in [“Operator Precedence Rules” on page A-11](#).

Simple Syntax

To select documents that contain stemmed variations of the phrase “pharmaceutical companies” and the word “stock,” use the following search query:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'pharmaceutical companies AND stock');
```

Only documents that contain both search elements are retrieved. They are relevance-ranked.

Explicit Syntax

To select documents that contain both the literal phrase “pharmaceutical companies” and the literal word “stock,” use the following query:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'AND
("pharmaceutical companies", "stock")');
```

Because the phrase “pharmaceutical companies” and the word “stock” are enclosed in double quotation marks, no stemming is done, and documents containing these words are not relevance-ranked.

NEAR

The NEAR operator selects documents containing specified search terms in proximity to each other. Document scores are calculated based on the relative number of words between search terms.

For example, if the search expression includes two words, and those words occur next to each other in a document (so that the region size is two words long), the score assigned to that document is 1.0. The document with the smallest possible region containing all search terms always receives the highest score. Documents that contain search terms that are not within 1000 words of each other are not selected, because the search terms are probably too far apart to be meaningful.

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

Simple Syntax

Use simple syntax to retrieve relevance-ranked documents based on word stem variations and relative proximity. For example, to retrieve relevance-ranked documents that contain stemmed variations of the words “war” and “peace” in close proximity to each other, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'war
<NEAR> peace');
```

Explicit Syntax

Use explicit syntax to retrieve relevance-ranked documents based on relative proximity only. To retrieve relevance-ranked documents that contain the literal words “war,” “and,” and “peace” in close proximity to each other, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<NEAR>("war", "and", "peace")');
```

NEAR/*n*

The NEAR/*n* operator selects documents containing two or more words within a specified number of words of each other. Document scores are calculated based on the relative distance of the specified words when they are separated by *n* words or fewer.

For example, if the search expression NEAR/5 is used to find two words within five words of each other, a document that has the specified words within three words of each other is scored higher than a document that has the specified words within five words of each other.

The *n* variable must be an integer between 1 and 1024, where NEAR/1 searches for two words that are next to each other. If *n* is 1000 or above, specify its value without commas, as in NEAR/1000. You can specify multiple search terms using multiple instances of NEAR/*n*, as long as the value of *n* is the same.

For example, to retrieve relevance-ranked documents that contain stemmed variations of the words “commute,” “bicycle,” “train,” and “bus” within *n* words of each other, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'commute
<NEAR/10> bicycle <NEAR/10> train <NEAR/10> bus');
```

You can use the NEAR/*n* operator with the ORDER modifier to perform ordered proximity searches. For more information about the ORDER modifier, see [“ORDER” on page A-29](#).

Simple Syntax

Use simple syntax to retrieve relevance-ranked documents based on word stem variations and relative proximity. For example, to retrieve relevance-ranked documents that contain stemmed variations of the words “commute,” “bicycle,” “train,” and “bus” within ten words of each other, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'commute
<NEAR/10> bicycle <NEAR/10> train <NEAR/10> bus');
```

The value of *n* must be the same.

Explicit Syntax

Use explicit syntax to retrieve relevance-ranked documents based on relative proximity only. To retrieve relevance-ranked documents that contain the literal words “commute,” “bicycle,” “train,” and “bus” within ten words of each other, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<NEAR/10>
("commute", "bicycle", "train", "bus")');
```

Using the ORDER Modifier

You can use the NEAR/*n* operator with the ORDER modifier to perform ordered proximity searches. For more information about the ORDER modifier, see [“ORDER” on page A-29](#).

The following syntax examples search for documents containing the words “diver,” “kills,” “shark” in that order within 20 words of each other.

The following example uses simple syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'diver
<ORDER><NEAR/20> kills <ORDER><NEAR/20> shark');
```

The following example uses explicit syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<ORDER><NEAR/20> ("diver", "kills", "shark")');
```

You can also use the NEAR/*n* operator with the ORDER modifier to duplicate the behavior of the PHRASE operator. For example, to search for documents containing the phrase “world wide web,” you can use the syntax in the following examples.

The following example uses simple syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'world
<ORDER><NEAR/1> wide <ORDER><NEAR/1> web');
```


The following example uses explicit syntax:

```
SELECT title FROM vtstest WHERE
Vts_contains(text, '<ORDER><NEAR/1> ("world", "wide",
"web")');
```

OR

Selects documents that contain one or more of your search elements. Documents selected using the OR operator are relevance-ranked. Following are examples of search syntax.

Simple Syntax

To select documents that contain stemmed variations of the word “election” or the phrases “national elections” or “senatorial race,” enter the following:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'election
OR national elections OR senatorial race');
```

Only those documents that contain at least one of the search elements (or a stemmed variation of at least one of them) are retrieved.

Explicit Syntax

To retrieve documents that contain either the literal word “computer” or the literal word “security,” enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'OR
("computer", "security")');
```

PARAGRAPH

Selects documents that include all of the words or phrases that you specify within a paragraph. You can specify search elements in any order. Documents are retrieved as long as search elements appear in the same paragraph. Following are examples of search syntax.

Simple Syntax

To retrieve relevance-ranked documents that contain stemmed variations of the word “drug” and the phrase “cancer treating” in the same paragraph, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'drug  
<PARAGRAPH> cancer treating');
```

To search for three or more words or phrases, you must use the PARAGRAPH operator between each word or phrase.

Explicit Syntax

To retrieve documents that contain the literal words “activity” and “management” in the same paragraph, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text,  
'<PARAGRAPH> ("activity", "management")');
```

Documents are not relevance-ranked with explicit syntax unless you use the MANY modifier, as follows:

```
SELECT title FROM vtstest WHERE Vts_contains(text,  
'<MANY><PARAGRAPH> ("activity", "management")');
```

PHRASE

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

Simple Syntax

By default, two or more words separated by a space are considered to be a phrase in simple syntax. Two or more words enclosed in double quotes are always considered a phrase. To retrieve relevance-ranked documents that contain the phrase “mission oak,” you can use either of the following queries:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'mission oak');
```

```
SELECT title FROM vtstest WHERE Vts_contains(text, ' "mission, oak" ');
```

Explicit Syntax

To retrieve documents containing the phrase “black and white,” enter the following:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<PHRASE> (black "and" white)');
```

In this example, the word “and” is in double quotes to ensure that it is considered a search term and not an operator.

Documents are not relevance-ranked with explicit syntax unless you use the **MANY** modifier, as in:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<MANY><PHRASE> (black "and" white)');
```

The following example retrieves documents containing the phrase “black and white”:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<PHRASE> (black "and" white)');
```

The PHRASE operator does require angle brackets, and the “and” is enclosed in double quotation marks because it is to be interpreted as a literal word, not as an operator.

SENTENCE

The following example uses simple syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'format a
date');
```

The following example uses explicit syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<PHRASE>("format", "a", "date")' );
```

SENTENCE

Selects documents that include all of the words you specify within a sentence. You can specify search elements in a sequential or a random order. Documents are retrieved as long as search elements appear in the same sentence. Following are examples of search syntax.

Simple Syntax

To retrieve relevance-ranked documents that contain stemmed variations of the words “American” and “innovation” within the same sentence, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'american
<SENTENCE> innovation ' );
```

Explicit Syntax

To retrieve documents containing the literal words “merge,” “annual,” and “purchases” in the same sentence, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<SENTENCE> ("merge", "annual", "purchases")' );
```

Documents are not relevance-ranked with explicit syntax unless you use the **MANY** modifier, as follows:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<MANY><SENTENCE> ("merge", "annual", "purchases")' );
```

SOUNDEX

Selects documents that contain one or more words that sound like or have a letter pattern that is similar to the specified search word. Selected words all begin with the same letter as the specified word.

For example, to retrieve documents containing a word that is close in structure to the word “sale,” use the following query string:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<SOUNDEX>
sale' );
```

The documents retrieved include words such as “sale,” “sell,” “seal,” “shell,” “soul,” and “scale.”

Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<MANY><SOUNDEX> sale' );
```

The SOUNDEX parameter of the WORD-IDXOPTS statement must be specified in your **style.prm** file.

STEM

Expands the search to include documents that contain one or more variations of the search term. For simple searches, the STEM and the MANY operators are the default, so you do not need to enter the STEM operator to return stemmed variations of a word in a simple search. Whenever you enclose the term in single quotation marks, stemming occurs.

For example, both of the following search strings retrieves documents that contain the word “film,” “films,” “filmed,” and “filming.”

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<STEM>
film' );
SELECT title FROM vtstest WHERE Vts_contains(text, 'film' );
```

When you use the STEM operator, documents are not relevance-ranked unless the MANY modifier is used.

```
SELECT title FROM vtstest WHERE  
Vts_contains(text,'<MANY><STEM> film');
```

However, when you use single quotes, MANY is assumed. Therefore, the following two search strings are equivalent:

```
SELECT title FROM vtstest WHERE Vts_contains(text,'film');  
  
SELECT title FROM vtstest WHERE  
Vts_contains(text,'<MANY><STEM>film');
```



Important: For stemming to occur, the Stemdex parameter of the WORD-IDXOPTS statement must be specified in your *style.prm* file.

THESAURUS

The THESAURUS operator expands the search to include documents that contain one or more synonyms of the specified term.

For example, to retrieve documents containing synonyms of the word “altitude,” use the following query:

```
SELECT title FROM vtstest WHERE  
Vts_contains(text,'<THESAURUS> altitude');
```

The retrieved documents include synonyms for “altitude,” such as “height” or “elevation.”

Documents are not relevance-ranked unless the MANY modifier is used, as in:

```
SELECT title FROM vtstest WHERE  
Vts_contains(text,'<MANY><THESAURUS> altitude');
```

TYPO/n

Expands the search to include documents that contain words that are similar to the specified word.

The `TYPO/n` operator performs approximate pattern matching to identify similar words. This operator is useful for searching documents that have been scanned using an optical character reader (OCR).

The optional *n* variable in the operator name expresses the maximum number of spelling changes between the query term and a matched term, a value called the *error distance*. The default error distance is 2.

The following example searches for words within three transformations of “Colombia”:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<TYPO/3>
Colombia)');
```

The error distance between two words is based on the calculation of errors, where an error is defined as a character insertion, deletion, or transposition. For example, for these sets of words, the second word matches the first within an error distance of 1.

Specified Word	Similar Word	Error Distance	Explanation
mouse	house	1	“m” changes to “h”.
agreed	greed	1	“a” is deleted.
cat	coat	1	“o” is inserted.
sweeping	swimming	3	The first “e” changes to “i”. The second “e” changes to “m”. The “p” changes to “m”.
swept	kept	1	“s” is deleted. “w” changes to “k”.

WILDCARD



The `TYPO/n` operator scans the collection's word list to find candidate matching words. This makes it impractical for use in large collections (greater than 100,000 documents unless a current spanning word list is available) or in performance-sensitive environments. Performance can be improved by generating a spanning word list for the collections to be used.

Important: A query term specified with `TYPO/n` can have a maximum length of 32 characters. `TYPO/n` is not supported for multibyte character sets.

WILDCARD

Selects documents that contain matches to a character string containing wildcard characters.

Use wildcards in place of regular characters in search words. You can use the default wildcard characters—the asterisk (*) and the question mark (?)—in any query string without specifying `WILDCARD`. To use any other wildcards, you must explicitly specify `WILDCARD`.

If you want to search for a wildcard character, you must precede that character with backslashes. Instructions follow in the section [“WORD” on page A-27](#).

Supported wildcards are described in the following table. Examples follow the table.

Character	Function
?	<p>Represents a single alphanumeric character. Any word that has a single letter in this position is a match.</p> <p>It is not necessary to use the WILDCARD operator with this wildcard. The question mark is ignored in a set ([]) or in an alternative pattern ({ }).</p>
*	<p>Represents zero or more alphanumeric characters. Any word that has zero or more characters in this position is a match.</p> <p>It is not necessary to use the WILDCARD operator with this wildcard. The asterisk is ignored in a set ([]) or in an alternative pattern ({ }).</p> <p>Do not use an asterisk to specify the first character of a wildcard string.</p>
[]	<p>Encloses a set of acceptable characters within a word. Any word that has one of these characters in the position is a match.</p> <p>You must enclose the word that includes a set in single quotes ('), and there can be no spaces in a set.</p>
-	<p>Specifies a range of characters in a set, as in <WILDCARD> `c[a-r]t`, that locates every three-letter word from "cat" to "crt."</p>
{ }	<p>Lists alternative patterns, separated by a comma. You must enclose the word that includes a pattern in single quotes ('), and there can be no spaces in a set.</p>
^	<p>Used with a set, the caret (^) precedes a list of characters to be excluded from the results.</p> <p>The caret must be the first character after the left bracket ([) that introduces a set.</p>

Examples

You can perform the following types of wildcard searches:

- Question mark (?)

The following query retrieves documents that contain “fan,” “ran,” “can,” “pan,” “ban,” or any other three-letter word that ends with “an”:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<WILDCARD> (?an)');
```

- Asterisk (*)

The following query retrieves documents that contain words such as “pharmaceutical,” “pharmacology,” and “pharmacodynamics”:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<WILDCARD> (pharmac*)');
```

- Excluded character list

The following query string specifies characters to be excluded from the search. The search does not search for words that have the letters “o” or “a” between “st” and “ck.” Therefore, the words “stock” and “stack” are excluded. The result set includes “stick” and “stuck.”

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<WILDCARD> ''st[^oa]ck''');
```

Documents returned from wildcard searches are not relevance-ranked unless the MANY modifier is used, as in:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<WILDCARD> <MANY> pharmac*');
```

Searching for Nonalphanumeric Characters

By default, wildcard searches search for alphanumeric characters. If you want to include nonalphanumeric characters in your search, set up your **style.lex** file to recognize the characters you want to search for. See [“style.lex” on page 6-9](#) for instructions.

WORD

Selects documents that include one or more instances of a specified word, spelled exactly as entered. You can use double quotation marks instead of the WORD operator to achieve the same result.

For example, the following search strings retrieve documents that contain the word “spring” but not those that contain “springer,” “springtime,” or “springboard”:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<WORD>
spring');
```

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'spring');
```

When you use the WORD operator, documents are not relevance-ranked unless the MANY modifier is used.

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<MANY><WORD> spring');
```

However, when you use double quotes, MANY is assumed. Therefore, the following two examples are equivalent:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'spring');
```

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<MANY><WORD> spring');
```

Modifier Reference

Modifiers change the behavior of operators in some way. For example, you can use the CASE modifier with an operator to specify that the case of the search word you enter be considered a search element as well. The Verity Text Search DataBlade module supports the modifiers CASE, MANY, NOT, and ORDER, each of which is described in this section.

CASE

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

By default, documents containing any occurrences of a search word or phrase are retrieved regardless of case. To use the CASE modifier, enter the search word or phrase as it appears in the documents: uppercase, mixed, or lowercase.

For example, to retrieve documents that contain the word “Apple” in mixed case letters, enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<CASE>  
<WORD> Apple');
```

Only those documents that contain the word “Apple” are selected. Occurrences of “apple,” “apples,” or “APPLE” are not returned.

MANY

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

For example, to select documents based on the density of stemmed variations of the word “apple,” enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<MANY>  
<STEM> apple');
```

To select documents based on the density of the phrase “mission oak,” enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<MANY>
mission oak');
```

The MANY modifier cannot be used with the AND, OR, or ACCRUE operators.

NOT

Use the NOT modifier with a word or phrase to exclude documents that show evidence of that word or phrase. The NOT modifier can be used only with the operators AND and OR. For example, to select documents that contain the words “cat” and “mouse” but not the word “dog,” enter the following syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'cat <AND>
mouse <AND> <NOT> dog');
```

ORDER

Use the ORDER modifier to indicate that search elements must occur in the same order that they were specified in the query. If search elements do not occur in the specified order in a document, the document is not selected. Always place the ORDER modifier immediately before the operator. You can use the ORDER modifier only with the operators PARAGRAPH, SENTENCE, and NEAR/*n*.

The following syntax examples show how you can use either simple syntax or explicit syntax to retrieve documents containing the word “president” followed by the word “washington” in the same paragraph.

The following example uses simple syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text, 'president
<ORDER><PARAGRAPH> washington');
```

The following example uses explicit syntax:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<ORDER><PARAGRAPH> ("president", "washington")');
```

Natural Language Operators

Natural language operators enable you to specify search criteria using natural language syntax. The search engine uses natural language analysis to translate the query text into Verity query language expressions for evaluating and scoring documents.

FREETEXT

The FREETEXT operator interprets text using the Verity free text query parser and scores documents using the resulting query expression. All retrieved documents are relevance-ranked. For information about the free text query parser, refer to the *Verity Developer's Kit API Reference Guide*.

This operator allows you to combine free text queries with other search criteria using the full Verity query language. Free text is a phrase or sentence written as it is written in text, as follows:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<FREETEXT> ("peace negotiations in the Middle East")');
```

The quotation marks are required. If you want to include embedded quotes, they must be preceded with backslashes. For example, to enclose the movie title "Independence Day" in quotes, enter it as follows:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<FREETEXT> ("\Independence Day\")');
```

The FREETEXT operator can be combined with other operators in the same way as the ACCRUE operator.



Tip: In the case where a query or document contains only words defined as stop words in the collection **style.stp** files, the free text query parser uses the stopwords for the query, ignoring the stopword list.

LIKE

The LIKE operator searches for other documents that resemble the sample documents or text passages you provide. The search engine analyzes the provided text to find the most important terms to use for the search. Retrieved documents are relevance-ranked.

Syntax

Use the following syntax to specify documents for the LIKE operator (where value is a text literal specifying the content you are seeking):

Text: *value*

The LIKE operator can be combined with other operators using the same rules as for the ACCRUE operator.

Quotation marks and backslashes embedded in LIKE expressions must be preceded by backslashes. The backslash indicates to the engine that the character that follows it is supposed to be treated as a literal character.

Examples

The following sample search strings use the LIKE operator with literal text and external keys. Because the *name* argument is omitted, all are treated as positive examples.

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<LIKE> (
  "{text:"sample text"}" )' );
```

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<LIKE> (
  "{text:"sample text"}" )');
```

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<LIKE> (
  "{text:"sample quote"}" )');
```

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<LIKE> (
  "{text:"sample quote"}" )');
```

Score Operators

Score operators specify how the search engine calculates scores for retrieved documents. When a score operator is used, the search engine first calculates a separate score for each search element found in a document. The search engine then performs a mathematical operation on the individual element scores to arrive at the final score for each document.

The YESNO operator has wide application, whereas the PRODUCT, SUM, and COMPLEMENT operators are intended for use mainly by application developers who want to generate queries programmatically.

COMPLEMENT

The COMPLEMENT operator calculates scores for documents matching a query by taking the complement (subtracting from 1) of the scores for the query's search elements. To arrive at a document's score, the Verity search engine calculates a score for each search element and takes the complement of these scores.

The COMPLEMENT operator is a unary operator. It combines search elements using the ACCRUE operator to generate a single score. This score is then converted to its complement.

This example illustrates use of the COMPLEMENT operator.

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<COMPLEMENT> ("computers","laptops")');
```

This query is evaluated as the word “computers” accrued using the ACCRUE operator with the word “laptops.” The result is subtracted from 1 to get the score.

PRODUCT

The PRODUCT operator calculates the score of a document by multiplying the scores for the individual search terms together. To arrive at a document's score, the Verity search engine calculates a score for each search element and multiplies these scores together.

This example illustrates the use of the PRODUCT operator:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<PRODUCT>
("computers","laptops")');
```

If a search on “computers” generates a score of .5 and a search on “laptops” generates a score of .75, the product score is .375 ($.5 * .75 = .375$).

SUM

The SUM operator calculates scores for documents matching a query by adding together, to a maximum of 1, the scores for the query's search elements. To arrive at a document's score, the Verity search engine calculates a score for each search element and adds these scores together.

This example illustrates the use of the SUM operator:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '<SUM>
("computers","laptops")');
```

If a search on “computers” generates a score of .5, and a search on “laptops” generates a score of .2, the total score for the query is .7 ($.2 + .5 = .7$).

If a search on “computers” generates a score of .5, and a search on “laptops” generates a score of .75, the total score for the query is 1.00. Although .5 and .75 adds up to a score greater than 1, the maximum score for any document is 1.00.

YESNO

The YESNO operator performs a Boolean search. It asks whether or not a document contains the affected search term, but does not rank documents according to how many times the search term occurs. YESNO assigns a score of 1 to all documents that contain one or more instances of the search term, and a 0 to all documents that do not contain the search term.

For example, if a regular search for the term “March” in a document produces a score of .75, a search using the YESNO operator scores the document as 1. Documents that do not contain the word “March” are scored as 0.

When a query has more than one search term, all search terms that include the YESNO operator score as 1, but terms not including that operator score normally. The document score is the average score of all search terms.

In the following query, if a document scores .75 for “March” and .5 for “wind,” the document score is .5, because the score of AND is normally the minimum score of all its search elements:

```
SELECT title FROM vtstest WHERE Vts_contains(text, '"March"
<AND> "wind"');
```

In the following query, the same document scores 1 for “March” and .5 for “wind,” so the document score is the average of the two scores, or .75:

```
SELECT title FROM vtstest WHERE Vts_contains(text,
'<YesNo>"March" <AND> "wind"');
```

Regular Expressions

The following table lists the regular expressions used in Verity style files and describes the values that they represent. Italicized values are variables that can stand for any alphanumeric character.

Operator	Matched Pattern
<i>x</i>	The specified character
<i>\x</i>	<p>The specified character, even if it is also an operator (except for the <i>\b</i>, <i>\f</i>, <i>\n</i>, <i>\r</i>, <i>\t</i>, and <i>\v</i> operators listed in this table). The backslash indicates that it is treated as a character and not an operator.</p> <p>For example, to search for the dollar sign (\$) character, which can also be used as an operator, use <i>\\$</i>.</p>
<i>\b</i>	A backspace
<i>\f</i>	A form-feed character
<i>\n</i>	A newline character
<i>\r</i>	A carriage return
<i>\t</i>	A tab character
<i>\v</i>	A vertical tab character

Operator	Matched Pattern
[xy]	Any of the characters enclosed in square brackets. For example, [Amy] represents any one of the three characters: <ul style="list-style-type: none"> ■ uppercase A ■ lowercase m ■ lowercase y
[x-z]	Any character in the specified range. For example, [a-e] represents any of the following letters: a, b, c, d, e; [5-7] represents the numbers 5, 6, or 7.
.	Any character but newline. This symbol can be used to indicate a word of a specified length. For example, represents any word that has five characters.
[^x]	Any character except the specified character.
^x	The specified character when it appears at the beginning of a line.
x\$	The specified character when it appears at the end of a line.
x?	0 or 1 occurrence of the specified character.
x*	0 or more occurrences of the specified character.
x+	1 or more occurrences of the specified character. See the entry for “.” for an example.
x y	Either of the specified characters.
(x y)z	Either of two combinations, consisting of either of the characters in parenthesis and the character represented by z, in the specified order. For example, m(e y) represents these two character combinations: me and my.
{symbol}	The translation of a symbol defined earlier in the file.

(2 of 2)

The following example illustrates the use of regular expressions in combination.

<code>[0-9.][0-9.][0-9.][0-9.][0-9.]+</code>	All numbers with five or more digits
<code>.....+</code>	All words longer than 30 characters
<code>[0-9a-zA-Z]</code>	Every one-letter word
<code>^M.....+</code>	Lines encoded with uuencode

DataBlade Tables

This appendix describes internal tables created and used by the Verity Text Search DataBlade module. Use these tables as you would use system tables.

The DataBlade-specific tables discussed in this chapter are:

Vts_Context_Type Lists the associations between data types and contexts

Vts_Contexts Lists and describes all custom contexts and indicates whether or not they are currently used by indexes

Vts_Context_Type Table

The **Vts_Context_Type** table tracks associations between contexts and data types. Each time the **Vts_AssocContext()** routine associates a context with a row or distinct data type, a row is added to this table. When the **Vts_DisassoContext()** routine removes that association, the row is deleted. An entry is also generated when a language-specific data type is created using the **Vts_CreateType()** routine; the entry is removed when the data type is dropped by the **Vts_DropType()** routine.

Use this table to find out whether a row or distinct data type is associated with a custom context or to find out which data types and contexts are associated with each other.

The following table lists the columns of the **Vts_Context_Type** table.

Column Name	Type	Description
type_name	VARCHAR (64)	The name of a data type associated with the context specified in context_name . This column is the primary key.
context_name	VARCHAR (64)	The name of the context associated with the data type specified in type_name .
refcount	INT	The reference count for the data type. This number indicates how many indexes are built on this data type.

Vts_Contexts Table

The **Vts_Contexts** table serves as a catalog of information for the contexts that have been created in a database.

Each time the **Vts_CreateContext()** routine creates a context, a row is added to this table. When the **Vts_DropContext()** routine removes that context, the row is deleted.

This table is also used for contexts associated with language-specific data types, such as french_clob. An entry is made in this table when the context is created by the **Vts_CreateLocale()** routine. When the context is created by the **Vts_CreateContext()** routine, no entry is made in the **language** column.

The following table lists the columns in the **Vts_Contexts** table.

Column Name	Type	Description
context_name	VARCHAR (64)	The name of the context. This column is the primary key.
creation_params	CHAR (1024)	The FSE directory containing the style files for the context.
refcount	INT	The reference count for the context. This number indicates how many indexes currently use this context.
language	VARCHAR (64)	Name of the locale associated with this context, if it is a language-specific context.

When an index is created on a column whose data type is associated with a context, the reference count (in the **refcount** column) for that context entry in **Vts_Contexts** is increased by 1. Whenever an index is dropped, using the DROP INDEX statement, the refcount for that context is reduced by 1.

A refcount greater than 0 indicates that the context is in use. Some routines, such as **Vts_AlterContext()**, require a refcount of 0.

Filtered Formats

The Verity Text Search DataBlade module includes a universal filter that automatically converts a variety of documents into ASCII format, with all proprietary formatting information stripped away. This appendix lists the formats filtered by the universal filter.



Tip: You cannot insert more than one filtered format into a row.

KeyView Filters

The Verity universal filter invokes the KeyView Filter Kit to index documents in the following formats:

- Microsoft Office formats, including
 - Office `95, `97
 - Word for Windows 2.0, 6.0, `95, `97
 - Word for DOS 4.x, 5.x, 6.x
 - Word for Mac 4.0, 5.0, 6.0
 - WordPad all versions
 - Write for Windows, all versions
 - Rich Text Format (RTF) 1.x, 2.0
 - Excel Windows 3.0, 4.0, 5.0, `95, `97
 - Excel Macintosh 3.0, 4.0
 - PowerPoint (Windows) `95, `97

- Lotus Suite
 - AMI Pro 2.x, 3.0.3.1
 - 1-2-3 (DOS/Win) 2.0, 3.0, 4.0, 5.0
 - 1-2-3 (OS/2) release 2
- Corel
 - WordPerfect 5.0, 5.1, 6.0x, 6.1, 7.0
 - WordPerfect for Windows 5.1, 5.2, 6.0x, 6.1, 7.0
 - WordPerfect for Macintosh 2.0, 2.1, 3.0-3.5

Check the release notes to see whether any more filters are supported with your release.

Zone Filters

The Verity universal filter uses zone filters to index markup language documents and documents in internet message formats. Files in this group are ASCII files that use tags for formatting.

The following zone formats are supported:

- Plain ASCII text
- ANSI text
- HTML and SGML
- Internet email (SMTP)
- Usenet news (NNTP)

Glossary

access method	<p>A set of server routines that the database server uses to access and manipulate an index or a table. B-tree is the default secondary access method used by DataBlade modules. The Verity Text Search DataBlade module uses its own secondary access method, the vts access method, which is defined by the module.</p> <p>See also <i>secondary access method</i>.</p>
BLOB	<p>Binary large object. A smart large object data type that stores any kind of binary data, including images.</p> <p>See also <i>smart large object</i>.</p>
built-in data type	<p>A fundamental data type defined by the database server: for example, INTEGER, CHAR, or SERIAL8.</p>
built-in function	<p>A predefined, SQL-invoked function that provides some basic arithmetic and other operations, such as cos, log, or today.</p>
cast	<p>A mechanism that the database server uses to convert data from one data type to another. The server provides built-in casts that it performs automatically. Users can create both implicit and explicit casts.</p>
CLOB	<p>Character large object. 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>
clustering	<p>A method of grouping similar documents. Verity performs document clustering by identifying the subtopics in a set of documents and grouping the documents by those subtopics.</p>

connection	An association between an application and a database environment, created by a CONNECT or DATABASE statement. Database servers can also have connections to one another.
constructed data type	A complex data type created with a type constructor, for example, a collection data type or an unnamed row data type.
constructor	See <i>type constructor</i> .
context	<p>A Verity Text Search DataBlade module construct that determines the characteristics of a vts index at the time that it is created and the behavior of searches that use that index.</p> <p>A context associates Verity style files with a data type. The Verity Text Search DataBlade module provides a routine for creating contexts and associating them with data types and indexes created on columns of that data type.</p>
DataBlade module	A collection of database objects and supporting code that extends an object-relational database to manage new kinds of data or add new features. A DataBlade module can include new data types, routines, casts, access methods, SQL code, client code, and installation programs.
detached index	An index created with a fragmentation strategy or stored in a separate dbspace or sbspace from the associated table. With Universal Data Option, all indexes are detached.
distinct data type	A data type that is created with the CREATE DISTINCT TYPE statement. A distinct data type is based on an existing opaque, built-in, distinct, or named row data type, known as its source type. The distinct data type has the same internal storage representation as its source type, but it has a different name. To compare a distinct data type with its source type requires an explicit cast. A distinct data type inherits all routines that are defined on its source type.
external file	An operating system file that is accessible from a database table. You can specify that a document in a Verity Text Search DataBlade module table is stored in an external file rather than in an sbspace by using IfxDocDesc as your data type and specifying IFX_FILE in the location field.
field	A component of a named row data type. A field has a name and a data type and is accessed in an SQL statement by using dot notation in the form: <i>row_type_name.field_name</i> .
FSE directory	File system emulation directory. For the Verity Text Search DataBlade module, this is an area in an sbspace where style files and locales are stored.

function	<p>A routine that can accept arguments and returns one or more values.</p> <p>See also <i>built-in function</i>, <i>routine</i>, <i>user-defined function</i>.</p>
Global Language Support (GLS)	<p>An application environment that allows Informix application-programming interfaces (APIs) and database servers to handle different languages, cultural conventions, and code sets. Developers use the GLS libraries to manage all string, currency, date, and time data types in their code. Using GLS, you can add support for a new language, character set, and encoding by editing resource files, without access to the original source code, and without rebuilding the DataBlade module or client software.</p>
index	<p>(noun) A structure of pointers to rows of data in a table. An index optimizes the performance of database queries by ordering rows to make access faster.</p> <p>(verb) To prepare an index.</p>
index fragment	<p>Zero or more index items grouped together, which can be stored in the same dbspace as the associated table fragment or in a separate dbspace or sbspace.</p>
INFORMIXDIR	<p>The UNIX or Windows NT environment variable that specifies the directory in which the database server is installed.</p>
large object	<p>A data object that exceeds 255 bytes in length. A large object is logically stored in a table column but physically stored independently of the column, because of its size. Large objects can contain non-ASCII data. Informix Dynamic Server with Universal Data Option recognizes two kinds of large objects: simple large objects (TEXT, BYTE) and smart large objects (CLOB, BLOB).</p> <p>See also <i>simple large object</i>, <i>smart large object</i>.</p>
locale	<p>A set of files that define the native-language behavior of the program at runtime. The rules are usually based on the linguistic customs of the region or the territory. The locale can be set through an environment variable that dictates output formats for numbers, currency symbols, dates, and time as well as collation order for character strings and regular expressions.</p> <p>See also <i>Global Language Support (GLS)</i>.</p>
LVARCHAR	<p>A built-in data type that stores varying-length character data greater than 256 bytes. It is used for input and output casts for opaque data types. LVARCHAR supports code-set order for comparisons of character data.</p>

named row data type	<p>A row data type that is created with the CREATE ROW TYPE statement and has a name. A named row data type can be used to construct a typed table and can be part of a type or table hierarchy.</p> <p>See also <i>row data type</i>, <i>unnamed row data type</i>.</p>
opaque data type	<p>A fundamental data type of a predefined fixed or variable length whose internal structure is hidden. Opaque data types are created with the SQL statement CREATE OPAQUE TYPE. Support functions must always be defined for opaque types.</p>
opclass	<p>See <i>operator class</i>.</p>
operator	<p>A symbol, such as =, >, +, -, that invokes an operator function.</p>
operator class	<p>The set of operators that the database server associates with a secondary access method or query optimization and index creation. When an index is created, it is associated with a particular operator class. Users with Resource privileges can create new operator classes by using the CREATE OPCLASS statement.</p>
procedure	<p>A routine that can accept arguments but does not return a value.</p>
query optimizer	<p>A server facility that estimates the most efficient plan for executing a query in the DBMS. The optimizer considers the CPU cost and the I/O cost of executing a plan.</p>
registration	<p>The process of executing SQL statements to create DataBlade module objects or individual user-defined routines in a database and giving the database server the location of the associated shared object file. Registration makes a DataBlade module available for use by client applications that open that database.</p>
routine	<p>A named collection of program statements that perform a particular task and can accept arguments. Routines include functions, which return one or more values, and procedures, which do not return values.</p> <p>See also <i>function</i>, <i>procedure</i>, <i>user-defined routine</i>.</p>
ROW constructor	<p>A type constructor used to construct unnamed row data types.</p>
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 SERIAL and SERIAL8 and, in certain situations, TEXT and BYTE.</p>

There are two kinds of row data types:

- Named row types, created using the CREATE ROW TYPE statement
- Unnamed row types, created using the ROW constructor

See also *named row data type*, *unnamed row data type*.

sbspace	Smart large object space. An sbspace is a logical storage area that contains one or more chunks that only store BLOB or CLOB type data. Sbspaces are created by the onspaces utility.
score	A numeric representation of how closely the results of a search match the search criteria. The score is a value > 0 and ≤ 100 , with 0 representing no match and 100 representing the best possible match.
secondary access method	<p>A set of server functions that build, access, and manipulate an index structure: for example, B-tree, R-tree, or vts. When an index is built on a secondary access method, the database server uses the access method's functions rather than built-in functions to perform index-related tasks. Typically, a secondary access method speeds up the retrieval of data.</p> <p>A secondary access method is sometimes referred to as an <i>index access method</i>.</p> <p>See also <i>operator class</i>.</p>
simple large object	A large object that is stored in a blobspace, is not recoverable, and does not obey transaction isolation modes. Simple large objects include TEXT and BYTE data types.
SLV	Abbreviation for <i>statement local variable</i> .
smartblob	See <i>smart large object</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.
style file	A Verity style file provides information about an index and determines its search characteristics. Style files can also specify index creation characteristics, such as indexing modes.
system catalog	A group of database tables that contain information about the database itself, such as the names of tables or columns in the database, the number of rows in a table, the information about indexes and database privileges, and so on.
table	A rectangular array of data in which each row describes a single entity and each column contains the values for each category of description. A table is sometimes referred to as a <i>base table</i> to distinguish it from the views, indexes, and other objects defined on the underlying table or associated with it.
type constructor	An SQL keyword that indicates to the database server the type of complex data to create.
unnamed row data type	A row type created with the ROW constructor that has no defined name and no inheritance properties. Two unnamed row types are equivalent if they have the same number of fields and if corresponding fields have the same data type, even if the fields have different names.
user-defined function	A user-defined routine that returns a value.
user-defined routine	A routine, written in one of the languages that Informix Dynamic Server with Universal Data Option supports, that provides added functionality for data types or encapsulates application logic.
user-defined procedure	A user-defined routine that does not return a value.

Index

A

Access method 1-9
 ALTER INDEX statement 5-4
 Archive, of vts index 2-13
 Auxiliary results from queries 1-5

B

BladeManager 2-6
 BLOB data types
 documents used for 2-7
 locating with LLD_Locator 3-7
 operator class for 5-7
 sbspace, creating for 2-4

C

CASE modifier 2-22
 Case-sensitivity, enabling 6-19
 CHAR data types
 documents used for 2-7
 operator class for 5-7
 Choosing data types 2-7
 CLOB data types
 documents used for 2-7
 index, creating on 2-13
 locating with LLD_Locator 3-7
 operator class for 2-13, 5-7
 sbspace, creating for 2-4
 Cluster information
 definition of 1-6
 how returned 4-12
 returned to IfxVtsReturnType 3-9
 returning 2-30

 returning in
 IfxVtsClusterType 3-14
 turning on in style.prm 6-17
 CLUSTER parameter 4-10
 Contexts
 creating 2-9, 4-14
 custom 1-11
 data types, associating with 2-12,
 4-6
 default 2-9
 defined 1-10
 description of 2-9
 disassociating from data
 types 4-21
 dropping 2-42, 4-23
 modifying 4-4
 name of FSE directory for 4-33
 CREATE INDEX statement
 DataBlade-specific syntax for 5-5
 NOLOG parameter in 5-7
 operator class, specifying in 5-5
 USING clause in 5-6
 CREATE TABLE statement 5-10
 Creating
 contexts 2-9, 4-14
 distinct data types 2-7
 fragmented indexes 2-14
 language-specific data types 2-37
 locales 2-36, 4-16
 row data types 2-7
 tables 2-12
 vts indexes 2-13, 5-5 to 5-9
 CUTOFFSCORE parameter 4-10

D

Data types

- BLOB 2-7
- CHAR 2-7
- CLOB 2-7
- creating row and distinct 2-7
- distinct 2-7
- dropping 2-42
- guidelines for choosing 2-7
- IfxDesc 2-7, 3-4
- IfxVtsClusterType 3-14
- IfxVtsReturnType 3-9
- IfxVtsSummaryType 3-12
- indexable 1-5, 5-7
- language-specific, creating 2-37, 4-19
- language-specific, dropping 4-29
- language-specific, using 3-16
- LLD_Locator 3-7
- LVARCHAR 2-7
- operator classes for 5-7
- row 2-7
- VARCHAR 2-7

Data, inserting into a table 2-15

define statements 6-11, 6-17

Distinct data types

- contexts, associating with 2-12, 4-6
- contexts, disassociating from 4-21
- creating 2-7
- creating an index on 5-8
- creating language specific 4-19
- dropping language-specific 4-29
- operator class for 5-8
- when to use 2-7

Documentation

- conventions Intro-5 to Intro-8
- from Verity Intro-11
- organization Intro-3
- related Intro-8, Intro-12
- this manual Intro-3 to Intro-5

Documents

- data types for 2-7
- filtering of 1-6
- formats supported D-1
- highlighting ASCII 2-33
- highlighting HTML 2-33
- indexable data types for 1-5

location of, specifying 3-4

multilingual 1-7

DROP INDEX statement 5-11

Dropping

- contexts 2-42, 4-23
- data types 2-42
- data types, language-specific 4-29
- FSE directory 4-27
- indexes 5-11
- locales 2-42, 4-25
- style files 4-27
- tables 2-41

F

File System Emulation subsystem.

See FSE directory.

Files

- inserting into CLOB
- columns 2-15
- locating with LLD_Locator 3-7

Filtering of documents

- described 1-6
- formats supported D-1

Formats

- list of supported 6-1

Formats supported 6-3, D-1

FRAGMENT BY EXPRESSION

- clause 5-5

Fragmented indexes, creating 2-14, 5-5

FSE directory

- dropping 4-27
- name of, returning 4-33
- sbspaces, creating for 2-5
- specifying 2-10
- style files, storing in 4-35

H

Highlighting

- performing 4-31
- search strings 1-6
- search terms 2-32
- terms in ASCII documents 2-33
- terms in HTML documents 2-33
- turning on in style.prm 6-17

I

IfxDesc data types

- described 3-4
- documents used for 2-7
- operator class for 5-7

IfxVtsClusterType data type 3-14

IfxVtsReturnType data type 3-9

IfxVtsSummaryType data type 3-12

IN operator 2-18

Indexable data types 1-5, 2-7, 5-7

Inserting

- data into a table 2-15
- documents with transactions 2-17
- files into CLOB columns 2-15

Installing the DataBlade

module 2-4

Isolation levels 5-13

K

KeyView Filter Kit D-1

Keywords 2-29

L

Languages supported 1-7

LLD_Locator data type 3-7

Locales

- creating 2-36, 4-16
- data types for 3-16
- data types, creating for 2-37
- dropping 2-42, 4-25
- setting default 1-7

Location of documents,

specifying 3-4

Logging of indexes 2-13, 5-6

LVARCHAR data types

- documents used for 2-7
- operator class for 5-7

M

Manual

- conventions Intro-5 to Intro-8
- description Intro-3 to Intro-5

organization Intro-3
 MANY operator 2-23
 Modifier, CASE 2-22
 Multicolumn indexing 2-15
 Multilingual documents
 languages for 1-7
 locale, setting for 1-7
 multiple languages, using in 1-7
 using 2-36 to 2-42

N

NEAR/n operator 2-21
 NOLOG index parameter 5-7
 Nonalphanumeric characters,
 indexing on 6-9
 NULL values
 casting required for 2-15

O

ONCONFIG file
 SBSPACENAME parameter
 in 2-5
 sbspaces, creating in 2-5
 user-defined virtual processor,
 creating in 2-5
 onspaces utility, creating sbspaces
 with 2-5
 Operator classes
 defined 1-10
 list of 5-7
 specifying 2-13
 specifying during index
 creation 5-8
 Vts_blob_ops 5-7
 Vts_char_ops 5-7
 Vts_clob_ops 5-7
 Vts_ifxDesc_ops 5-7
 Vts_lvarchar_ops 5-7
 Vts_row_ops 5-8
 Vts_varchar_ops 5-8
 Operators
 IN 2-18
 MANY 2-23
 NEAR/n 2-21
 PHRASE 2-20
 SOUNDEX 2-21

THESAURUS 2-20
 Ordering results 4-12
 Ordering results by score 2-19

P

Parameters, tuning 4-9
 Phonetic representations,
 enabling 6-19
 PHRASE operator 2-20
 Proximity search 2-21

Q

QP parameter 4-10
 Queries
 auxiliary results of 1-5
 CASE modifier, using in 2-22
 cluster information, returning
 for 2-30
 highlighting results of 1-6, 2-32
 MANY operator, using in 2-23
 NEAR/n operator, using in 2-21
 on multilingual tables 2-41
 ordering by score 2-19, 4-12
 performing 2-17 to 2-23
 PHRASE operator, using in 2-20
 row type field in 2-18
 SOUNDEX operator, using
 in 2-21
 summary information, returning
 for 2-29
 THESAURUS operator, using
 in 2-20
 TOPN parameter, using in 2-23
 wildcards, using in 2-28
 Query parser options 4-12

R

Registering the DataBlade
 module 2-6
 Regular expressions B-1
 Results
 cluster information 1-6, 4-12
 ordering by score 4-12
 score information 1-5, 4-12

summary information 1-6, 4-12
 Returning
 auxiliary results from queries 1-5
 DataBlade module version
 information 4-38
 name of FSE directory 4-33
 Routines
 SELECT statement, using in 5-12
 Vts_AlterContext() 4-4
 Vts_AssocContext() 4-6
 vts_contains 4-9
 Vts_CreateContext() 4-14
 Vts_CreateLocale() 4-16
 Vts_CreateType() 4-19
 Vts_DisassoContext() 4-21
 Vts_DropContext() 4-23
 Vts_DropLocale() 4-25
 Vts_DropStyle() 4-27
 Vts_DropType() 4-29
 Vts_GetHighlight() 4-31
 Vts_QueryContext() 4-33
 Vts_ReadStyle() 4-35
 Vts_Release() 4-38
 Vts_WriteStyle() 4-39
 Row data types
 casting required for 2-15
 contexts, associating with 2-12,
 4-6
 contexts, disassociating from 4-21
 creating 2-7
 index, creating on 2-14, 5-6
 multicolumn indexing with 2-15
 operator class for 2-14, 5-8
 searching in a field of 2-18
 when to use 2-7

S

SBSPACENAME parameter 2-5
 Sbspaces
 columns, storing in 2-13
 creating 2-4
 indexes, storing in 2-13
 style files, storing in 2-10
 Score
 description of 1-5
 how returned 4-12
 ordering results with 2-19

- returned to IfxVtsReturnType 3-9
- SCORE parameter 4-11
- Searches
 - excluding stemmed variations A-8
 - field in a row type, of 2-18
 - indexed 1-8
 - nonindexed 1-8
- Secondary access method 1-9
- SELECT statement
 - DataBlade routines, using in 5-12
 - DataBlade-specific syntax for 5-12
 - select list in 5-12
 - WHERE clause in 5-12
- SET ISOLATION statement 5-13
- Smart large objects
 - dropping 4-27
 - locating with LLD_Locator 3-7
 - style files 4-39
- SOUNDEX operator 2-21
- SQL clauses
 - FRAGMENT BY EXPRESSION 5-5
 - USING 5-6
 - WHERE 5-12
- SQL statements
 - ALTER INDEX 5-4
 - CREATE INDEX 5-5
 - CREATE TABLE 5-10
 - DROP INDEX 5-11
 - SELECT 5-12
 - SET ISOLATION 5-13
- Statement local variable
 - auxiliary results, returning in 2-28
 - cluster information, returning in 2-30
 - summary information, returning in 2-29
- Stemmed variations A-8
- Stopwords, in style.stp style file 6-22
- Style files
 - alterable 6-6
 - custom 1-11
 - customizing 2-10
 - defined 1-10
 - dropping 4-27

- editing 4-39
- list of 4-36
- location of 6-3
- making available 4-35
- new 6-5
- reading 2-10
- regular expressions in B-1
- storing in FSE directory 4-35
- style.ddd 6-7
- style.dft 6-6
- style.did 6-7
- style.go 6-5, 6-8
- style.lex 6-5, 6-9
- style.ngm 6-7
- style.pdd 6-7
- style.plc 6-6
- style.prm 6-6, 6-17
- style.sfl 6-6, 6-21
- style.sid 6-7
- style.stp 6-5, 6-22
- style.ufl 6-6
- style.uni 6-6
- style.vgw 6-6
- style.wld 6-7
- unalterable 6-7
- style.ddd style file 6-7
- style.dft style file 6-6
- style.did style file 6-7
- style.go style file 6-5, 6-8
- style.lex style file 6-5
 - character mapping in 6-16
 - creating 6-9
 - define statements in 6-11
 - defining characters in 6-11
 - defining spaces in 6-11
 - described 6-9
 - interpreting statements in 6-14
 - sample of 6-15
 - symbols in 6-10
 - token statements in 6-12
- style.ngm style file 6-7
- style.pdd style file 6-7
- style.plc style file 6-6
- style.prm style file 6-6
 - define statements in 6-17
 - described 6-17
 - syntax for 6-17
- style.sfl style file 6-6, 6-21
- style.sid style file 6-7

- style.stp style file 6-5
 - case sensitivity with 6-22
 - description of 6-22
 - regular expressions in 6-23
 - sample of 6-23
 - syntax for 6-22
- style.ufl style file 6-6
- style.uni style file 6-6
- style.vgw style file 6-6
- style.wld style file 6-7
- Summary information
 - description of 1-6
 - how returned 4-12
 - returning 2-29
 - returning in
 - IfxVtsReturnType 3-9
 - returning in
 - IfxVtsSummaryType 3-12
 - style.prm, turning on in 6-17
- SUMMARY parameter 4-11
- Symbols 6-10

T

- Tables
 - creating 2-12, 5-10
 - dropping 2-41
 - populating 2-15
 - Vts_Contexts C-3
 - Vts_Context_Type C-2
- THESAURUS operator 2-20
- token statement 6-12
- TOPN parameter 2-23, 4-11
- Transactions for inserting documents 2-17
- Tuning parameters
 - CLUSTER 4-10
 - CUTOFFSCORE 4-10
 - QP 4-10
 - SCORE 4-11
 - SUMMARY 4-11
 - TOPN 2-23, 4-11

U

- User-defined virtual processor, creating 2-5
- User's Guide Intro-3 to Intro-5

USING clause 5-6

V

VARCHAR data types
 documents used for 2-7
 operator class for 5-8
 Verity Query Language
 quotation marks A-30
 Verity Text Search DataBlade
 module
 components 1-8 to 1-11
 document formats supported for D-1
 features 1-3 to 1-8
 FSE directory for 2-5
 indexable data types for 2-7
 installing 2-4
 queries, performing with 2-17 to 2-23
 registering 2-6
 sbspaces, creating for 2-4
 user-defined virtual processor, creating for 2-5
 version information, returning 4-38
 Version information, returning 4-38
 Virtual processor 2-5
 vts access method 1-9
 See also vts indexes.
 vts indexes
 archiving 2-13
 CLOB column, creating on 2-13
 contexts for 1-10
 creating 2-13, 5-5 to 5-9
 customizing 1-8
 dropping 5-11
 fragmented, creating 2-14, 5-5
 indexable data types for 1-5, 2-7
 logging during creation of 5-6
 logging of, turning off 2-13
 modifying 5-4
 nonalphanumeric characters in 6-9
 nonindex searches, compared to 1-8

 operator class, specifying for 2-13, 5-7
 row data type, creating on 2-14
 sbspaces, creating for 2-4
 sbspaces, stored in 2-13
 stopwords, using with 6-22
 Vts_AlterContext() procedure 4-4
 Vts_AssocContext() procedure 4-6
 Vts_blob_ops operator class 5-7
 Vts_char_ops operator class 5-7
 Vts_clob_ops operator class 5-7
 vts_contains operator
 described 1-10
 ordering results with 4-12
 query parser options for 4-12
 syntax for 4-9
 tuning parameters for 4-10
 WHERE clause, using in 5-12
 Vts_Contexts table C-3
 Vts_Context_Type table C-2
 Vts_CreateContext()
 procedure 4-14
 Vts_CreateLocale() procedure 4-16
 Vts_CreateType() procedure 4-19
 Vts_DisassoContext()
 procedure 4-21
 Vts_DropContext() procedure 4-23
 Vts_DropLocale() procedure 4-25
 Vts_DropStyle() procedure 4-27
 Vts_DropType() procedure 4-29
 Vts_GetHighlight() function 4-31
 Vts_IfxDocDesc_ops operator class 5-7
 Vts_lvarchar_ops operator class 5-7
 Vts_QueryContext() function 4-33
 Vts_ReadStyle() procedure 4-35
 Vts_Release() function 4-38
 Vts_row_ops operator class 5-8
 Vts_varchar_ops operator class 5-8
 Vts_WriteStyle() procedure 4-39

W

WHERE clause 5-12
 Wildcards, using in queries 2-28

Z

Zone filters D-2

