

# Oracle8 ConText<sup>®</sup> Cartridge

Administrator's Guide

Release 2.3

December 1997

Part No. A58165-01

---

Oracle8 ConText Cartridge Administrator's Guide

Part No. A58165-01

Release 2.3

Copyright © 1996, 1997, Oracle Corporation. All rights reserved.

Primary Author: D. Yitzik Brenman

Contributors: Dave Allewell, Paul Anderson, Peter Bell, Steve Buxton, Chandu Bhavsar, Jack Chen, Chung-Ho Chen, Roy Clarke, Franco Cravero, Paul Dixon, Mohammad Faisal, Elena Huang, Garrett Kaminaga, Hassan Karraby, Jacqueline Kud, Kavi Mahesh, Yasuhiro Matsuta, Colin McGregor, Rich Paulson, Gerda Shank, Steve Yang

**The programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be licensee's responsibility to take all appropriate fail-safe, back up, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle disclaims liability for any damages caused by such use of the Programs.**

This Program contains proprietary information of Oracle Corporation; it is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright patent and other intellectual property law. Reverse engineering of the software is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

If this Program is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

**Restricted Rights Legend** Programs delivered subject to the DOD FAR Supplement are 'commercial computer software' and use, duplication and disclosure of the Programs shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are 'restricted computer software' and use, duplication and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-14, Rights in Data -- General, including Alternate III (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

Oracle, SQL\*Net, SQL\*Plus, and ConText are registered trademarks of Oracle Corporation. Oracle8, Oracle Forms, Oracle Server, PL/SQL, and Gist are trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

---

---

# Contents

<b>Send Us Your Comments .....</b>	<b>xvii</b>
<b>Preface.....</b>	<b>xix</b>
<b>1 Introduction</b>	
<b>What is ConText?.....</b>	<b>1-2</b>
<b>ConText Features.....</b>	<b>1-3</b>
<b>ConText and the Oracle Server.....</b>	<b>1-4</b>
<b>Overview of Administration Tasks .....</b>	<b>1-6</b>
ConText Administration.....	1-7
Text Setup and Management .....	1-8
Linguistics Setup.....	1-9
<b>Administration Methods.....</b>	<b>1-10</b>
Command-line .....	1-10
Administration Tools .....	1-11
<b>Part I ConText Administration</b>	
<b>2 Administration Concepts</b>	
<b>Administrator Responsibilities.....</b>	<b>2-2</b>
System Administrator .....	2-2
Database Administrator (DBA) .....	2-2

<b>ConText Roles .....</b>	<b>2-3</b>
CTXADMIN Role.....	2-4
CTXAPP Role .....	2-4
CTXUSER Role .....	2-4
<b>Predefined ConText Users .....</b>	<b>2-5</b>
CTXSYS User .....	2-5
CTXDEMO User.....	2-5
<b>ConText Servers.....</b>	<b>2-7</b>
Text Operations.....	2-7
Text Request Queue.....	2-8
Server Log .....	2-8
Server Shutdown.....	2-8
<b>Personality Masks .....</b>	<b>2-9</b>
Loader (R) Personality .....	2-9
DDL (D) Personality .....	2-10
DML (M) Personality.....	2-10
Query (Q) Personality .....	2-10
Linguistic (L) Personality.....	2-11
DBA Personality.....	2-11
<b>Text Request Queue .....</b>	<b>2-13</b>
Administration Pipes .....	2-14
Query Pipe .....	2-14
DDL Pipe.....	2-14
<b>DML Queue.....</b>	<b>2-15</b>
Pending Table (DRQ_PENDING) .....	2-16
In-Progress Table (DRQ_INPROG).....	2-16
Waiting Table (DRQ_WAITING) .....	2-16
Batches Table (DRQ_BATCHES).....	2-16
Timestamps.....	2-17
Error Handling.....	2-17
Queue Management .....	2-17
<b>Services Queue .....</b>	<b>2-18</b>
Services Queue Table (CTX_SVCQ).....	2-18
Error Handling.....	2-19
Queue Management .....	2-19

### 3 Administering ConText

<b>Enabling One-step Queries</b> .....	3-2
Setting TEXT_ENABLE for All Users.....	3-2
Setting TEXT_ENABLE for the Session.....	3-2
<b>Managing Users</b> .....	3-3
Creating ConText Users.....	3-4
Granting ConText Roles to Users.....	3-4
Granting EXECUTE Privileges to Application Developers.....	3-5
<b>Managing ConText Servers</b> .....	3-6
Starting ConText Servers.....	3-7
Viewing the Status of ConText Servers.....	3-9
Changing the Personality Masks of ConText Servers.....	3-10
Shutting Down ConText Servers.....	3-10
<b>Managing ConText Queues</b> .....	3-11
Viewing the DML Queue .....	3-12
Viewing the Services Queue .....	3-12
Removing Requests from the Services Queue.....	3-13
Enabling and Disabling Queues.....	3-14

### 4 ConText Server Executables and Utilities

<b>ctxsrv Executable</b> .....	4-2
Syntax .....	4-2
Examples.....	4-3
<b>ctxctl Utility</b> .....	4-5
Syntax .....	4-5
Examples.....	4-6

### 5 PL/SQL Packages - Administration

<b>CTX_ADM: ConText Administration</b> .....	5-2
CHANGE_MASK .....	5-3
GET_QUEUE_STATUS.....	5-4
RECOVER.....	5-6
SET_QUERY_BUFFER_SIZE .....	5-7
SHUTDOWN.....	5-8

UPDATE_QUEUE_STATUS .....	5-9
<b>CTX_SVC: Services Queue Administration .....</b>	<b>5-11</b>
CANCEL .....	5-12
CANCEL_ALL .....	5-13
CANCEL_USER .....	5-14
CLEAR_ALL_ERRORS .....	5-15
CLEAR_ERROR .....	5-16
CLEAR_INDEX_ERRORS .....	5-17
CLEAR_LING_ERRORS .....	5-18
REQUEST_STATUS .....	5-19
<b>CTX_INFO: Product Information .....</b>	<b>5-21</b>
GET_INFO .....	5-22
GET_STATUS .....	5-23
GET_VERSION .....	5-24

## Part II Text Setup and Management

### 6 Text Concepts

<b>The ConText Data Dictionary .....</b>	<b>6-2</b>
<b>Text Operations.....</b>	<b>6-3</b>
Text Loading.....	6-3
DDL.....	6-3
DML.....	6-4
Text/Theme Queries .....	6-7
Linguistics Requests .....	6-9
<b>Text Columns .....</b>	<b>6-10</b>
Supported Datatypes.....	6-10
Textkeys.....	6-10
Composite Textkeys .....	6-11
<b>Text Loading.....</b>	<b>6-13</b>
Plain Text Loading.....	6-13
Batch Loading.....	6-13
Automated Batch Loading .....	6-14
Error Handling.....	6-15

<b>Text Storage</b> .....	6-16
Direct Storage .....	6-16
External Storage .....	6-17
Master-Detail Storage.....	6-18
<b>External Text</b> .....	6-21
Text Stored as File Names .....	6-21
Text Stored as URLs .....	6-22
Document Access Using HTTP or FTP.....	6-24
<b>Text Filtering</b> .....	6-27
Internal Filters .....	6-28
External Filters .....	6-30
Filters for Single-Format Columns.....	6-32
Filters for Mixed-Format Columns .....	6-32
Supported External Filter Formats for Mixed-Format Columns .....	6-35
Supplied External Filters .....	6-38
<b>ConText Indexes</b> .....	6-41
ConText Index Tables .....	6-42
Stages of ConText Indexing .....	6-42
Columns with Multiple Indexes.....	6-43
Index Fragmentation.....	6-44
Memory Allocation.....	6-44
Parallel Indexing.....	6-45
Index Updates .....	6-45
Index Optimization .....	6-46
Index Log.....	6-47
<b>Text Indexes</b> .....	6-49
Text Lexers.....	6-49
What's in a Text Index?.....	6-50
DDL and DML .....	6-52
<b>Theme Indexes</b> .....	6-53
Theme Lexer .....	6-53
What's in a Theme Index? .....	6-54
Linguistic Settings .....	6-55
Index Fragmentation.....	6-55
DDL and DML .....	6-56

<b>Base-letter Conversion .....</b>	<b>6-57</b>
Text Indexing.....	6-57
Text Queries.....	6-58
<b>Thesauri .....</b>	<b>6-59</b>
Thesaural Maintenance .....	6-59
Case-sensitivity .....	6-60
Query Expansion .....	6-60
<b>Types of Thesaural Relationships .....</b>	<b>6-62</b>
Synonyms.....	6-62
Hierarchical Relationships.....	6-64
Related Terms.....	6-67
Scope Notes .....	6-67
<b>Document Sections .....</b>	<b>6-68</b>
Sections.....	6-68
Section Groups .....	6-71
Startjoin and Endjoin Characters.....	6-72
Text Filtering .....	6-72
Document Section Setup.....	6-73
Predefined HTML Section Group and Sections .....	6-74
Section Searching .....	6-74

## 7 Understanding the ConText Data Dictionary: Indexing

<b>Policies .....</b>	<b>7-2</b>
What is a Policy? .....	7-3
Column Policies .....	7-4
Template Policies .....	7-4
Text Indexing Policies .....	7-4
Theme Indexing Policies.....	7-5
Policy Examples .....	7-6
Policy Attributes .....	7-8
Preferences in Policies.....	7-10
Predefined Template Policies.....	7-11



<b>Preferences for Indexing</b> .....	7-15
What is an Indexing Preference? .....	7-15
What is a Tile? .....	7-16
Data Store Predefined Preferences .....	7-17
Filter Predefined Preferences .....	7-19
Lexer Predefined Preferences .....	7-21
Engine Predefined Preferences .....	7-22
Wordlist Predefined Preferences .....	7-23
Stoplist Predefined Preferences .....	7-24
<b>Data Store Tiles</b> .....	7-26
List of Data Store Tiles and Attributes .....	7-26
DIRECT Tile .....	7-27
MASTER DETAIL Tile .....	7-27
MASTER DETAIL NEW Tile .....	7-28
OSFILE Tile .....	7-29
URL Tile .....	7-29
Data Store Example .....	7-31
<b>Filter Tiles</b> .....	7-32
List of Filter Tiles and Attributes .....	7-32
BLASTER FILTER Tile .....	7-33
FILTER NOP Tile .....	7-34
HTML FILTER Tile .....	7-34
USER FILTER Tile .....	7-35
Filter Examples .....	7-36
<b>Lexer Tiles</b> .....	7-37
Text Lexers .....	7-37
Theme Lexer .....	7-40
List of Lexer Tiles and Attributes .....	7-40
BASIC LEXER Tile .....	7-41
CHINESE V-GRAM LEXER Tile .....	7-46
JAPANESE V-GRAM LEXER Tile .....	7-46
KOREAN LEXER Tile .....	7-47
THEME LEXER Tile .....	7-47
Lexer Examples .....	7-48

<b>Engine Tiles</b> .....	7-49
List of Engine Tiles and Attributes.....	7-49
ENGINE NOP Tile.....	7-50
GENERIC ENGINE Tile.....	7-50
Engine Example .....	7-53
<b>Wordlist Tiles</b> .....	7-54
List of Wordlist Tiles and Attributes .....	7-56
GENERIC WORD LIST Tile .....	7-57
Wordlist Example .....	7-58
<b>Stoplist Tiles</b> .....	7-59
List of Stoplist Tiles and Attributes.....	7-59
GENERIC STOP LIST Tile .....	7-59
Stoplist Example .....	7-60

## 8 Understanding the ConText Data Dictionary: Text Loading

<b>Sources</b> .....	8-2
What is a Source? .....	8-3
Source Attributes .....	8-4
Preferences in Sources.....	8-4
<b>Preferences for Text Loading</b> .....	8-5
What is a Text Loading Preference? .....	8-5
Reader Predefined Preferences .....	8-6
Engine Predefined Preferences .....	8-6
Translator Predefined Preferences .....	8-6
<b>Reader Tiles</b> .....	8-7
DIRECTORY READER Tile.....	8-7
<b>Engine Tiles</b> .....	8-8
GENERIC LOADER Tile.....	8-8
<b>Translator Tiles</b> .....	8-9
NULL TRANSLATOR Tile.....	8-9
USER TRANSLATOR Tile.....	8-9

## 9 Setting Up and Managing Text

<b>Loading Text</b> .....	9-2
Using ctxload.....	9-2
Using ConText Servers for Automated Text Loading.....	9-4
Generating Document Textkeys.....	9-6
Updating/Exporting a Document .....	9-8
<b>Managing Preferences</b> .....	9-10
Creating a Preference .....	9-11
Creating an Engine Preference .....	9-12
Creating a Data Store Preference for a Master Table .....	9-13
Creating Filter Preferences .....	9-15
Creating a Theme Lexer Preference .....	9-18
Creating a Stoplist Preference.....	9-19
Deleting a Preference .....	9-20
<b>Managing Policies</b> .....	9-21
Creating a Column Policy .....	9-22
Creating a Theme Indexing Policy .....	9-23
Using Composite Textkeys in a Policy .....	9-24
Creating a Template Policy .....	9-25
Modifying a Policy .....	9-25
Deleting a Policy.....	9-26
<b>Managing Indexes</b> .....	9-27
Creating an Index .....	9-28
ConText Indexing in Parallel .....	9-29
Indexing Existing Columns (Hot Upgrade).....	9-31
Updating an Index.....	9-31
Dropping an Index .....	9-32
Optimizing an Index .....	9-32
Resuming Index Creation/Optimization.....	9-33
<b>Managing Thesauri</b> .....	9-34
Creating a Thesaurus .....	9-35
Creating a Case-sensitive Thesaurus .....	9-35
Creating/Updating a Thesaurus Entry .....	9-36
Deleting a Thesaurus .....	9-37
Creating a Thesaurus Output File.....	9-37

<b>Managing Document Sections .....</b>	<b>9-38</b>
Creating a Section Group.....	9-39
Creating a Section .....	9-39
Creating a Wordlist Preference with a Section Group .....	9-40
Creating a Policy for a Section Group.....	9-40
Viewing Sections and Section Groups .....	9-40
Removing a Section from a Section Group .....	9-40
Dropping a Section Group.....	9-41

## 10 Text Loading Utility

<b>Overview of ctxload.....</b>	<b>10-2</b>
Text Loading.....	10-2
Document Updating/Exporting.....	10-3
Thesaurus Importing and Exporting .....	10-3
<b>Command-line Syntax.....</b>	<b>10-4</b>
Mandatory Arguments .....	10-4
Optional Arguments.....	10-6
Usage Notes.....	10-8
<b>Command-line Examples .....</b>	<b>10-9</b>
Text Load Example .....	10-9
Document Update Example .....	10-9
Document Export Examples.....	10-10
Thesaurus Import Example .....	10-10
Thesaurus Export Example .....	10-10
<b>Structure of Text Load File .....</b>	<b>10-11</b>
Load File Structure .....	10-12
Load File Syntax.....	10-12
Example of Embedded Text in Load File .....	10-13
Example of File Name Pointers in Load File .....	10-13
<b>Structure of Thesaurus Import File .....</b>	<b>10-14</b>
Alternate Hierarchy Structure .....	10-16
Import File Structure for Terms.....	10-17
Import File Structure for Relationships.....	10-18
Examples of Import Files .....	10-19

## 11 PL/SQL Packages - Text Management

<b>CTX_DDL: Text Setup and Management</b> .....	11-2
ADD_SECTION .....	11-4
CLEAR_ATTRIBUTES .....	11-8
CREATE_INDEX .....	11-9
CREATE_POLICY .....	11-12
CREATE_PREFERENCE .....	11-15
CREATE_SECTION_GROUP .....	11-16
CREATE_SOURCE.....	11-17
CREATE_TEMPLATE_POLICY.....	11-19
DROP_INDEX.....	11-21
DROP_INTRIG .....	11-22
DROP_POLICY .....	11-23
DROP_PREFERENCE.....	11-24
DROP_SECTION_GROUP.....	11-25
DROP_SOURCE .....	11-26
OPTIMIZE_INDEX.....	11-27
REMOVE_SECTION .....	11-30
RESUME_FAILED_INDEX .....	11-31
SET_ATTRIBUTE.....	11-33
UPGRADE_INDEX .....	11-36
UPDATE_POLICY.....	11-37
UPDATE_SOURCE .....	11-39
<b>CTX_DML: ConText Index Update</b> .....	11-41
REINDEX.....	11-42
SYNC .....	11-44
SYNC_QUERY .....	11-46
<b>CTX_THES: Thesaurus Management</b> .....	11-47
CREATE_PHRASE .....	11-48
CREATE_THESAURUS.....	11-50
DROP_THESAURUS .....	11-51

## Part III Appendices

### A Supplied Stoplists

English Stoplist .....	A-2
French Stoplist.....	A-3
German Stoplist .....	A-4
Italian Stoplist .....	A-5
Spanish Stoplist .....	A-6

### B ConText Views

ConText Server Views .....	B-2
CTX_ALL_SERVERS.....	B-2
CTX_SERVERS.....	B-3
ConText Queue Views.....	B-4
CTX_ALL_DML_QUEUE.....	B-4
CTX_ALL_DML_SUM.....	B-4
CTX_ALL_QUEUE.....	B-5
CTX_INDEX_ERRORS.....	B-6
CTX_INDEX_STATUS .....	B-7
CTX_LING_ERRORS .....	B-7
CTX_USER_DML_QUEUE .....	B-8
CTX_USER_DML_SUM.....	B-8
CTX_USER_QUEUE.....	B-9
CTX_USER_SVCQ .....	B-10
ConText Data Dictionary Views.....	B-11
CTX_ALL_PREFERENCES .....	B-11
CTX_ALL_SECTIONS.....	B-12
CTX_ALL_SECTION_GROUPS .....	B-12
CTX_ALL_THESAURI.....	B-13
CTX_CLASS.....	B-13
CTX_COLUMN_POLICIES.....	B-14
CTX_INDEX_LOG.....	B-15
CTX_OBJECTS.....	B-16
CTX_OBJECT_ATTRIBUTES .....	B-17

CTX_OBJECT_ATTRIBUTES_LOV .....	B-18
CTX_POLICIES .....	B-19
CTX_PREFERENCES .....	B-20
CTX_PREFERENCE_ATTRIBUTES.....	B-20
CTX_PREFERENCE_USAGE .....	B-21
CTX_SOURCE.....	B-22
CTX_SQES .....	B-23
CTX_SYSTEM_PREFERENCES.....	B-23
CTX_SYSTEM_PREFERENCE_USAGE.....	B-24
CTX_SYSTEM_TEMPLATE_POLICIES .....	B-24
CTX_TEMPLATE_POLICIES.....	B-24
CTX_USER_COLUMN_POLICIES.....	B-25
CTX_USER_INDEX_LOG .....	B-26
CTX_USER_POLICIES.....	B-27
CTX_USER_PREFERENCES.....	B-27
CTX_USER_PREFERENCE_ATTRIBUTES .....	B-28
CTX_USER_PREFERENCE_USAGE .....	B-28
CTX_USER_SECTIONS.....	B-29
CTX_USER_SECTION_GROUPS.....	B-29
CTX_USER_SOURCES .....	B-30
CTX_USER_SQES.....	B-31
CTX_USER_TEMPLATE_POLICIES .....	B-31
CTX_USER_THESAURI .....	B-31

## C ConText Index Tables and Indexes

<b>ConText Index Tables</b> .....	C-2
DR_nnnnn_I1Tn .....	C-2
DR_nnnnn_KTB .....	C-3
DR_nnnnn_LST .....	C-3
DR_nnnnn_NLT .....	C-3
DR_nnnnn_I1W.....	C-4
<b>Oracle Indexes for ConText Index Tables</b> .....	C-5

<b>SQR Table</b> .....	C-6
DR_#####SQR .....	C-6
Oracle Index for DR_#####SQR.....	C-6

**Index**



---

---

# Send Us Your Comments

**Oracle8 ConText Cartridge Administrator's Guide, Release 2.3**

**Part No. A58165-01**

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to us in the following ways:

- FAX - (650) 506-7200. Attn: ConText Documentation Manager
- postal service:  
Oracle Corporation  
Attn: ConText Documentation Manager  
500 Oracle Parkway  
Redwood Shores, CA 94065  
USA

If you would like a reply, please give your name, address, and telephone number below.

---

---

---



---

# Preface

This manual explains how to administer Oracle8 ConText Cartridge and perform the necessary setup and maintenance to allow application developers to develop text-enabled applications.

## Audience

This manual is intended for the DBA or system administrator responsible for maintaining the ConText system.

It is also intended for the user who is responsible for setting up and maintaining the text stored in the database. This user could be a DBA or system administrator. It could also be an application designer or developer.

## Prerequisites

This document assumes that you have experience with the Oracle relational database management system, SQL, SQL\*Plus, and PL/SQL. See the documentation provided with your hardware and software for additional information.

If you are unfamiliar with the Oracle RDBMS and related tools, read Chapter 1, “An Introduction to the Oracle Server”, in *Oracle8 Server Concepts*. The chapter is a comprehensive introduction to the concepts and terminology used throughout Oracle documentation.

## Related Publications

For more information about ConText, see:

- *Oracle8 ConText Cartridge QuickStart*
- *Oracle8 ConText Cartridge Application Developer's Guide*
- *Oracle8 ConText Cartridge Workbench User's Guide*
- *Oracle8 Error Messages*

For more information about the Oracle8 server, see:

- *Oracle8 Server Concepts*
- *Oracle8 Server Administrator's Guide*
- *Oracle8 Server Utilities*
- *Oracle8 Server Tuning*
- *Oracle8 Server SQL Reference*
- *Oracle8 Server Reference Manual*
- *Oracle8 Server Application Developer's Guide*

For more information about PL/SQL, see:

- *PL/SQL User's Guide and Reference*

## How This Manual Is Organized

The manual is divided into an introduction and three parts:

### Chapter 1: Introduction

This chapter introduces ConText and provides an overview of the features and process model for the product.

### PART I: CONTEXT ADMINISTRATION

This section provides a discussion of ConText concepts and provides instructions for managing ConText users, servers, and queues. It also provides reference information for the executables, utilities, and PL/SQL packages provided by ConText for administering ConText. It contains the following chapters:

- Chapter 2: Administration Concepts
- Chapter 3: Administering ConText
- Chapter 4: ConText Server Executables and Utilities
- Chapter 5: PL/SQL Packages - Administration

### PART II: TEXT SETUP AND MANAGEMENT

This section provides a discussion of text concepts, introduces the ConText data dictionary, and provides instructions for setting up and managing text so that users (application developers) can perform text queries. It also provides reference information for the utilities and PL/SQL packages provided by ConText for managing text. It contains the following chapters:

- Chapter 6: Text Concepts
- Chapter 7: Understanding the ConText Data Dictionary: Indexing
- Chapter 8: Understanding the ConText Data Dictionary: Text Loading
- Chapter 9: Setting Up and Managing Text
- Chapter 10: Text Loading Utility
- Chapter 11: PL/SQL Packages - Text Management

## PART III: APPENDICES

This section provides a list of the stoplists provided by ConText, descriptions of the ConText views, and descriptions of the tables that comprise a ConText index. It contains the following appendices:

- Appendix A: Supplied Stoplists
- Appendix B: ConText Views
- Appendix C: ConText Index Tables and Indexes

## Type Conventions

This manual adheres to the following type conventions:

Type	Meaning
UPPERCASE	Uppercase letters indicate Oracle commands, standard database objects and constants, and standard Oracle PL/SQL procedures.
<i>lowercase italics</i>	Italics indicate variable names, names of user objects (tables, views, preferences, policies, etc.), PL/SQL parameter/argument names, and names of example PL/SQL procedures.  Italics also indicate emphasis.
<code>monospace</code>	Monospace type indicate example SQL*Plus commands and example PL/SQL code. Type in the command or code exactly as it appears.

## Customer Support

You can reach Oracle Worldwide Customer Support 24 hours a day.

In the USA: **1.415.506.1500**

In Europe: + **44.344.860.160**

Please be prepared to supply the following information:

- your CSI number (This helps Oracle Corporation track problems for each customer)
- the release numbers of the Oracle Server and associated products
- the operating system name and version number
- details of error numbers and descriptions (Write down the exact errors you encounter)
- a description of the problem
- a description of the changes made to the system

## Your Comments Are Welcome

Please use the Reader's Comment Form at the back of this document to convey your comments to us. You can also contact us at:

Documentation Manager  
ConText Group  
Server Technologies  
Oracle Corporation  
500 Oracle Parkway  
Redwood Shores, California 94065  
Phone: 1.415.506.7000 FAX: 1.415.506.7360





---

# Introduction

This chapter introduces Oracle8 ConText Cartridge and discusses the various administration tasks that you may need to perform for the system.

The topics covered in this chapter are:

- What is ConText?
- ConText Features
- ConText and the Oracle Server
- Overview of Administration Tasks
- Administration Methods

## What is ConText?

ConText is an Oracle server option which enables text queries to be performed through SQL and PL/SQL from most Oracle interfaces.

By installing ConText with an Oracle server, client tools such as SQL\*Plus, Oracle Forms, and Pro\*C are able to retrieve and manipulate text in an Oracle database. Most tools which can call an Oracle stored procedure can perform text queries and other text operations.

ConText manages textual data in concert with traditional datatypes in an Oracle database. When text is inserted, updated, or deleted, ConText automatically manages the change.

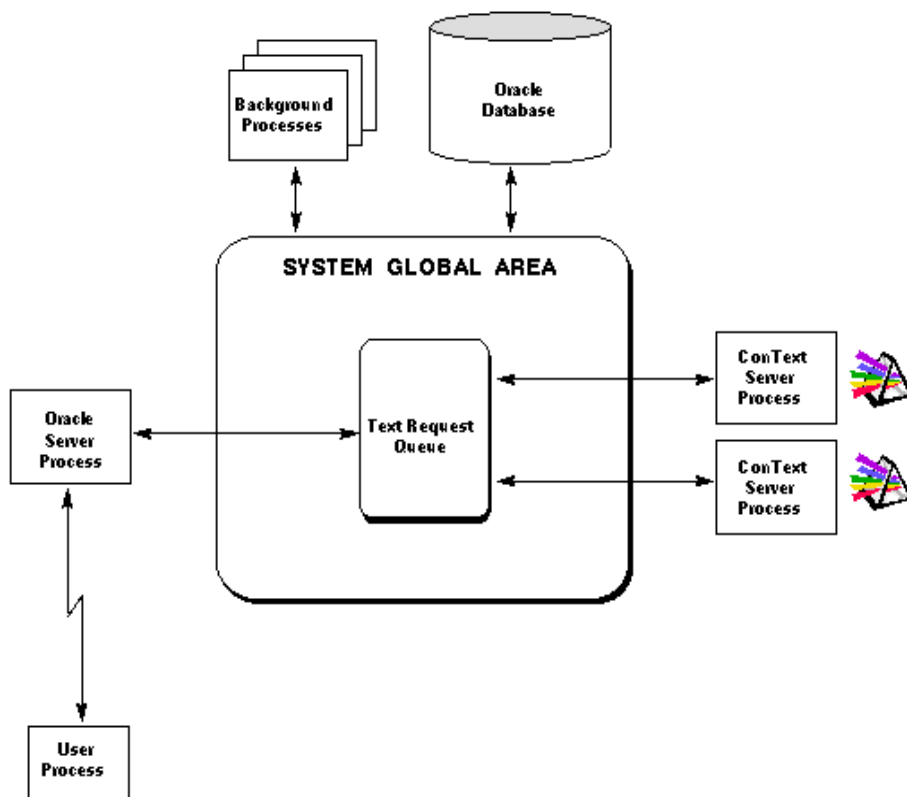
## ConText Features

ConText provides advanced indexing, analysis, retrieval, and viewing functionality that can be integrated into any text applications that use the Oracle8 server. The list of features include:

- full integration of structured data and text
- external and internal storage of text in the database
- support for a variety of document formats, including plain text, plain text with HTML tags, and many of the popular word processing formats
- text retrieval using Boolean, statistical, thesaural, and linguistic methods
- indexing and retrieval for all NLS-supported languages, including the following multi-byte languages: Japanese, Chinese, and Korean
- advanced text retrieval (stemming, fuzzy matching, etc.) for English, French, German, Italian, Spanish, and Dutch text
- advanced linguistic analysis, including theme-based queries, for English text
- text highlighting and viewing
- application development using any tools that support SQL or PL/SQL, including:
  - SQL\*Plus
  - Designer 2000
  - Power Objects
  - Oracle WebServer
  - Visual Basic and Oracle Objects for OLE
  - OCI and the PRO languages

**See Also:** For more information about text retrieval, linguistic analysis, highlighting, and document viewing, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## ConText and the Oracle Server



The ConText process model uses the Oracle server model with the addition of one or more ConText server processes and a queue for handling text operations.

ConText can be used in both dedicated server (one server process for each user process) and multi-threaded server environments (dispatcher, shared server, and background processes).

This diagram shows ConText servers and the Text Request Queue working in concert with an Oracle server process in a dedicated server environment.

In the standard Oracle server model, when a user process connects to the database, it spawns a dedicated server process which handles all incoming requests from the user process.

With ConText, if a request comes in that includes a text operation, the request is picked up by the server process for the user process. The server process sends the request to the Text Request Queue for processing by the next available ConText server processes.

For example, a text query is submitted by a user. The query is picked up by the Oracle server process and sent to the Text Request Queue.

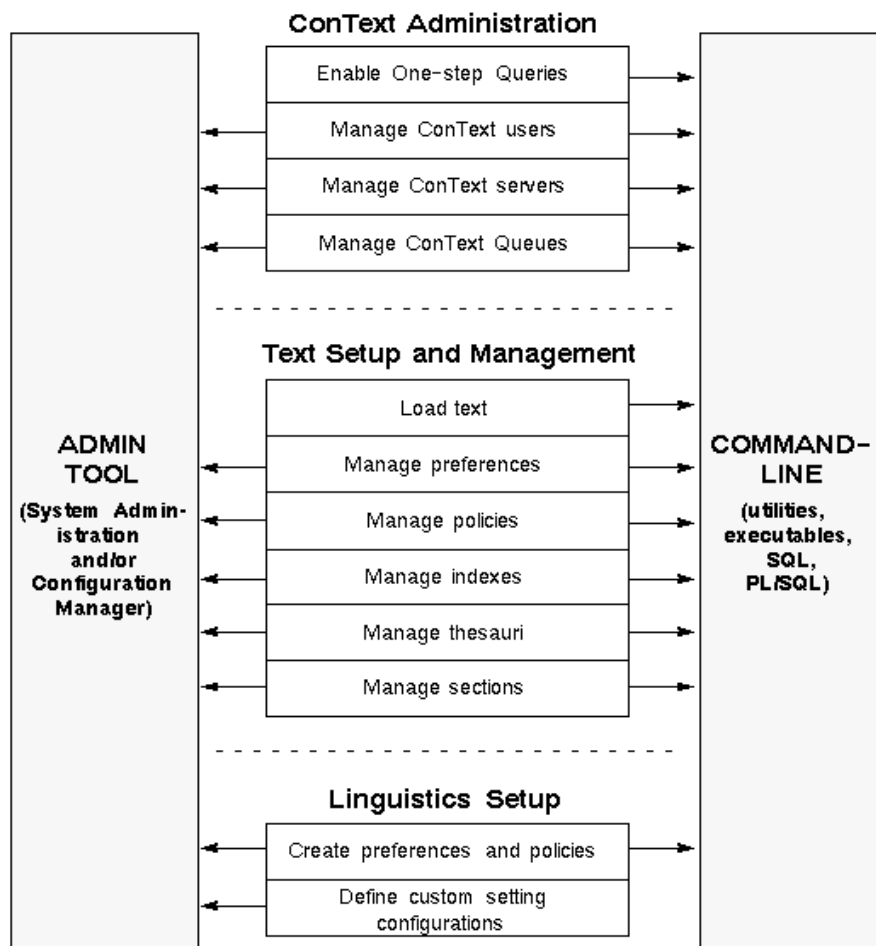
The first available ConText server picks up the text query from the Text Request Queue and processes it. The results from the text query are then returned to the user process.

**See Also:** For more information about ConText servers, see “ConText Servers” in Chapter 2, “Administration Concepts”.

For more information about text operations, see “Text Operations” in Chapter 6, “Text Concepts”.

For more information about the Oracle server model, see *Oracle8 Server Concepts*.

## Overview of Administration Tasks



Administration is divided into three areas:

- ConText Administration
- Text Setup and Management
- Linguistics Setup

This diagram illustrates the different administration tasks for ConText and identifies whether the task can be performed through the GUI administration tool or the command-line.

The tasks in the three areas may be performed by a single user or may be divided between different users/responsibilities.

## ConText Administration

ConText administration includes the following tasks:

- enabling one-step queries
- managing ConText users
- monitoring and managing ConText servers
- monitoring and managing the ConText queues

**See Also:** For examples of the ConText administration tasks, see Chapter 3, “Administering ConText”.

For more information about ConText users, servers, and queues, see Chapter 2, “Administration Concepts”.

### Who Performs ConText Administration?

ConText administration tasks are always performed by the ConText administrator, who may be the system and/or database administrator or a separate user.

## Text Setup and Management

Text setup and management includes the following tasks:

- loading, updating, and deleting text
- defining text storage and indexing options through preferences
- identifying text columns through policies
- creating ConText text and theme indexes
- creating and managing thesauri for use in queries
- creating and managing document sections to enable section searching in queries

**See Also:** For examples of the text setup and management tasks, see Chapter 9, “Setting Up and Managing Text”.

For more information about loading text, text storage, ConText indexes, and thesauri, see Chapter 6, “Text Concepts”.

For more information about preferences, policies, and other objects in the ConText data dictionary, see Chapter 7, “Understanding the ConText Data Dictionary: Indexing” or Chapter 8, “Understanding the ConText Data Dictionary: Text Loading”.

### Who Performs Text Setup and Management?

Text setup and management may be performed by the ConText administrator because they require access to system and database resources.

For example, loading text into the database requires access to the appropriate tables.

In addition, creating indexes requires ConText servers to be running with the appropriate designation and only ConText administrators can manage ConText servers.

However, some tasks, such as defining policies and preferences, may be performed by application developers because the options used to create an index have an effect on how an application retrieves text.



## Linguistics Setup

The ConText Linguistics setup includes the following tasks:

- creating preferences and policies
- defining custom setting configurations

---

---

**Note:** The preference and policy setup tasks for the Linguistics are identical to the setup tasks for text and theme indexing.

Custom setting configurations can only be defined in the administration tools provided by the ConText Workbench and, as such, are *not* discussed in this manual.

---

---

**See Also:** For examples of creating preferences and policies, see Chapter 9, “Setting Up and Managing Text”.

For more information about creating custom setting configurations, see the online help system provided with the administration tools.

For more information about ConText Linguistic, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Who Performs Linguistics Setup

Linguistics setup may be performed by the ConText administrator or by the application developer.

For example, the ConText administrator may be responsible for setting up all of the preferences and policies in the system, including policies that are used for theme indexing or requesting the Linguistics for documents.

On the other hand, application developers may be responsible for creating custom setting configurations as part of the process of creating linguistic output for use in applications.

## Administration Methods

ConText provides two different methods of administration:

- Command-line
- Administration Tools

Most of the administration tasks can be accomplished using either method; however, some tasks can only be accomplished using one or the other method.

For example, setting configurations for the ConText Linguistics can only be enabled for a database session through the command-line, while custom setting configurations can only be created/modified in the administration tool. As a result, if you want to use custom settings for the Linguistics, you must use *both* the administration tool and the command-line to administer ConText.

---

---

**Note:** Much of the conceptual and reference information presented in this manual are relevant to *both* the command-line and the administration tools.

However, the instructions presented in this manual are for performing administration tasks using the command-line.

For more information about performing administration tasks using the administration tools, see the help systems provided with the specific tool.

---

---

## Command-line

Command-line administration includes:

- machine command-line for running ConText executables and utilities
- SQL\*PLus and PL/SQL

For example, the command-line for the server machine on which ConText is installed must be used to start ConText servers and access the administration utilities provided with ConText.

All remaining ConText administration tasks, such as shutting down ConText servers, managing queues, and creating indexes, can be performed using SQL or PL/SQL, either on the server machine or on any other machine that has SQL\*Plus and PL/SQL installed and is connected to the server machine through SQL\*Net.

## Administration Tools

ConText provides two tools for administering ConText:

- System Administration tool
- Configuration Manager

Both administration tools are distributed with the ConText Workbench, which can be installed on any PC running Microsoft Windows NT or Windows 95.

**See Also:** For more information about the administration tools and the other components that are included in the ConText Workbench, see *Oracle8 ConText Cartridge Workbench User's Guide*.

### System Administration Tool

The System Administration tool is a client-based application that provides a 32-bit Windows, graphical user interface (GUI) for administering ConText servers, text, and the Linguistics from a 32-bit Windows platform, such as Windows NT or Windows 95.

**See Also:** For more information about using the System Administration tool, see the online help provided with the tool.

### Configuration Manager

The Configuration Manager is a Web-based application that allows a ConText administrator to manage various administration tasks quickly and easily. It also incorporates a simple ad-hoc query tool.

In contrast to the System Administration tool, which is for Windows NT and Windows 95 only and is installed separately on each client machine, the Configuration Manager is platform-independent and is installed only once per database. Each Configuration Manager installation runs under the CTXSYS user.

**See Also:** For more information about using the Configuration Manager, see the online help provided with the tool.



# Part I

---

## ConText Administration



---

# Administration Concepts

This chapter describes the concepts necessary for understanding system administration for ConText.

The following concepts are discussed in this chapter:

- Administrator Responsibilities
- ConText Roles
- Predefined ConText Users
- ConText Servers
- Personality Masks
- Text Request Queue
- DML Queue
- Services Queue

## Administrator Responsibilities

Administrative responsibilities for ConText can be divided into two functional areas:

- System Administrator
- Database Administrator (DBA)

### System Administrator

The system administrator's main responsibility is managing ConText servers. This includes:

- starting and stopping ConText servers to balance the processing loads of the machines on which the servers are running
- changing the personality masks of ConText servers as needed

### Database Administrator (DBA)

The DBA's main responsibilities include:

- creating and maintaining Oracle users for ConText
- monitoring and administering the ConText queues
- monitoring the ConText data dictionary



## ConText Roles

Three database roles provided with ConText:

- CTXADMIN Role
- CTXAPP Role
- CTXUSER Role

Each ConText user should be assigned one of the ConText roles. The role assigned to a ConText users depends on the tasks performed by the user.

---

**Note:** It is not necessary to assign more than one ConText role to a user because the roles are defined hierarchically in the following descending order: CTXADMIN, CTXAPP, CTXUSER.

Each higher-level role provides additional privileges in addition to the privileges of the lower-level roles.

---

**See Also:** For an example of assigning ConText roles to users, see “Managing Users” in Chapter 3, “Administering ConText”.

For more information about database roles, see *Oracle8 Server SQL Reference*.

The ConText tasks are grouped into the following areas:

- managing ConText servers, including:
  - shut down
  - monitoring ConText server status
  - changing the personality masks for ConText servers
- administering ConText queues, including:
  - removing entries from the queues
  - changing the status of the queues
  - setting the buffer size of the Query pipes
- managing the ConText data dictionary, including:
  - creating/deleting preferences
  - creating/deleting/modifying policies

- creating/deleting ConText indexes
- creating/deleting/updating thesauri
- managing linguistics, including:
  - creating custom setting configurations for linguistics
  - enabling setting configurations for linguistics
  - generating linguistic output
- performing queries

## **CTXADMIN Role**

The CTXADMIN role provides users with the ability to perform all of the ConText tasks.

CTXADMIN should be assigned to all users who perform system and database administration for ConText.

## **CTXAPP Role**

The CTXAPP role users with the ability to perform the following tasks/actions:

- managing the ConText data dictionary
- managing linguistics
- performing queries

CTXAPP should be assigned to all users who develop text applications.

## **CTXUSER Role**

The CTXUSER role provides users with the ability to perform ConText queries (text and theme). It should be assigned to all users of a text application.

## Predefined ConText Users

ConText provides two Oracle users which are created during installation:

- CTXSYS User
- CTXDEMO User

### CTXSYS User

CTXSYS is used primarily to perform administration tasks, such as starting ConText servers and administering the ConText queues.

CTXSYS has the following functions and privileges:

- Oracle DBA with all privileges
- owner of the ConText data dictionary
- CTXADMIN role

---

---

**Note:** Only the CTXSYS user can start ConText servers.

---

---

### CTXDEMO User

CTXDEMO is used primarily to run the sample applications provided with ConText.

CTXDEMO has the following functions and privileges:

- owner of the ConText database schema (tables, views, etc.) and data dictionary objects (policies, preferences) used in the sample applications
- CTXAPP role

The sample applications, provided to help application developers develop applications using ConText, include:

- EMP and DEPT tables with text indexes for performing ad-hoc text queries
- SQL scripts for setting up text in a ConText system for performing text queries
- SQL scripts for performing different types of queries
- SQL script for generating linguistic output necessary for advanced document viewing
- SQL scripts for viewing documents by themes or Gists/theme summaries

- Oracle Forms application, distributed with the ConText Workbench, for performing text queries

**See Also:** For more information about setting up and using the SQL scripts, see *Oracle8 ConText Cartridge Application Developer's Guide*.

For more information about setting up and using the Oracle Forms application, see *Oracle8 ConText Cartridge Workbench User's Guide*.

## ConText Servers

ConText servers are shared server processes that handle text operations in SQL requests. ConText server processes mirror their shared Oracle server counterparts, but process only text-related operations. ConText servers can be started using the `ctxsrv` executable or the `ctxctl` utility.

ConText servers can *only* be started from the command-line of the server machine where ConText is installed. In addition, *only* the CTXSYS user can start ConText servers.

If the server machine can support multiple ConText servers, multiple ConText servers can run at the same time to help balance the processing load. In addition, ConText servers can work with databases installed on either the server machine or on remote machines.

**See Also:** For more information about `ctxsrv` and `ctxctl`, see Chapter 4, “ConText Server Executables and Utilities”.

## Text Operations

ConText servers can process five types of text operations:

- text loading
- DDL operations (creating, dropping, and updating text indexes)
- DML operations (updating the text index for a text column when inserts, updates, and deletes are performed on the table)
- Linguistics requests (generating linguistic output for documents)
- text/theme queries

The type of text operations processed by each ConText server is determined by the personality mask assigned to the server.

**See Also:** For more information about each of the text operations that can be processed by ConText servers, see “Text Operations” in Chapter 6, “Text Concepts”.

For more information about personality masks, see “Personality Masks” in this chapter.

## Text Request Queue

Requests for text operations are stored in the Text Request Queue. Available ConText servers monitor the queue for text requests. As text operations are submitted, available ConText servers with the appropriate personalities pick up the operations and process them.

**See Also:** For more information about the database tables and pipes that comprise the Text Request Queue, see “Text Request Queue” in this chapter.

## Server Log

ConText provides a timestamped report for each action performed by a ConText server. These reports are written to the standard output on which the server was started; however, the reports can be directed to a log file if one is specified when the ConText server is started.

**See Also:** For examples of starting ConText servers, see “Starting ConText Servers” in Chapter 3, “Administering ConText”.

## Server Shutdown

A ConText server performs the following tasks before shutting down:

- releases all currently held resources
- cleans up and closes the server’s mailboxes
- updates the server process table by removing the corresponding entry for the server from the table

## Personality Masks

The personality mask for a ConText server indicates the text operations that the server can process. When a ConText server is started, it is assigned a personality mask. A personality mask consists of one or more of the following five personalities (corresponding to the five text operations supported by ConText):

- Loader (R) Personality
- DDL (D) Personality
- DML (M) Personality
- Query (Q) Personality
- Linguistic (L) Personality

---

---

**Note:** For ConText to process text and/or theme queries, at *least* one ConText server with the Query personality must be running.

---

---

In addition to the user-specified personalities, all ConText servers automatically take on a DBA Personality.

The DBA personality does not appear as part of the personality mask because it is automatically assigned to all ConText servers.

**See Also:** For more information about the text operations supported by ConText, see “Text Operations” in Chapter 6, “Text Concepts”.

### Loader (R) Personality

The Loader personality enables a ConText server to scan directories at regular intervals for files to be loaded into columns in the database. When the ConText server detects a new file in a specified directory, it uses the ctxload utility to load the contents of the file into a specified column.

The directories scanned by ConText servers running with the Loader personality, as well as the scanning intervals and the columns to which the text is loaded, are specified by a ConText object, called a source, which can be attached to a column.

**See Also:** For more information about automated text loading, see “Text Loading” in Chapter 6, “Text Concepts”.

For an example of setting up ConText servers to load text, see “Using ConText Servers for Automated Text Loading” in Chapter 9, “Setting Up and Managing Text”.

## DDL (D) Personality

The DDL personality enables a ConText server to process requests for creating, optimizing, and dropping text indexes. The text index on a column is what allows users to query the text stored in the column.

The DDL personality also enables a ConText server to process DML requests when DML operations are processed in batch mode.

**See Also:** For more information about batch DML processing, see “DML” in Chapter 6, “Text Concepts”.

## DML (M) Personality

The DML personality enables a ConText server to automatically update the text index for a table when changes to the table are made that require the text index to be update. Such changes include inserting new documents, updating existing documents, and deleting existing documents.

---

---

**Suggestion:** When systems have a high volume of text inserts, updates, and deletes, assign the DML personality to multiple servers to better distribute the system load.

---

---

## Query (Q) Personality

The Query personality enables a ConText server to process queries for text stored either internally or externally in a text column in a database table. If no running ConText servers have the Query personality, queries submitted to ConText will fail.

---

---

**Note:** The Query personality is *not* required to use the output generated by the Linguistics. Linguistic output is stored as structured data and, as such, no ConText servers are not required to process queries for this type of information.

---

---



## Linguistic (L) Personality

The Linguistic personality allows a ConText server to process requests for the Linguistics and generate linguistic output. Linguistic output includes:

- document themes
- document Gists and/or document theme summaries

---

---

**Note:** The Linguistic personality is *only* required to process requests for Linguistics. Once the requests have been processed, Linguistic servers can be shut down or the Linguistic personality can be removed from the personality mask of a running ConText server.

For more information about the ConText Linguistics, see *Oracle8 ConText Cartridge Application Developer's Guide*.

---

---

## DBA Personality

The DBA personality allows a ConText server to detect and correct client/server failures and perform system cleanup (recovery).

The DBA personality is automatically assigned to each ConText server during start up, which prevents a single point of failure in a multi-server configuration.

### ConText Server Monitoring

When a working ConText server detects a failed ConText server, it performs the following DBA actions:

- the working server cleans up the failed server's mailbox and logs the event
- if the failed server was processing a Query request, the working server releases any allocated resources
- if the failed server was processing a DDL request, the working server drops any tables that are not needed for recovery of the requested action
- if the failed server was processing a DML request, the working server places any uncompleted DML requests back in the queue
- if the failed server was processing a Query or DDL request, the working server notifies the waiting client

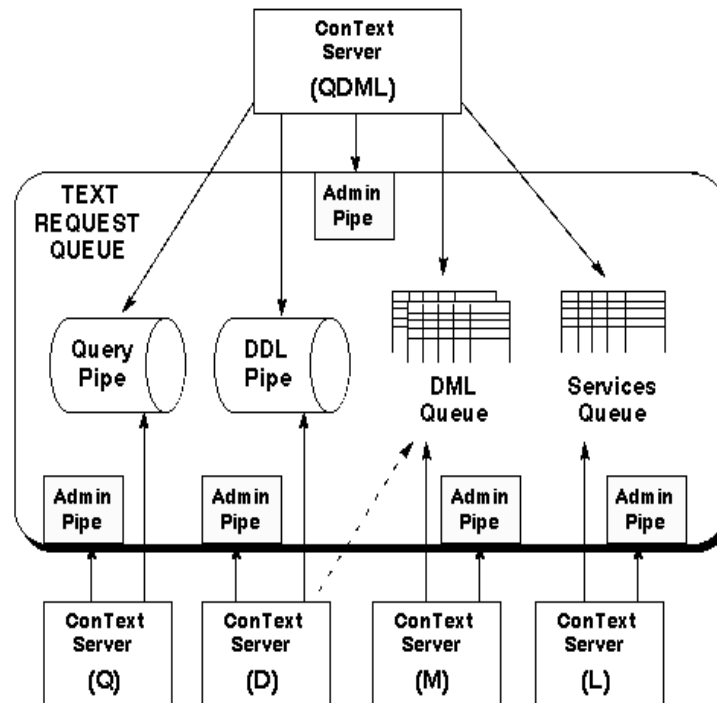
### **System Recovery**

When a table has a text column with an existing ConText index and the table is dropped without first dropping the index and policy for the column, the index tables and policy for the column remain in the system until recovery is performed.

System recover is performed automatically by ConText servers approximately every fifteen minutes.

System cleanup can also be performed manually using the RECOVER stored procedure in the CTX\_ADM PL/SQL package.

## Text Request Queue



The Text Request Queue is the logical mechanism ConText servers use to process all text operations, *except* for text loading.

When a client submits a request for a text operation, such as a query, the request is stored in the Text Request Queue. All available ConText servers scan the Text Request Queue regularly, retrieve pending requests if they have the appropriate personality mask, and perform the requested operations. This diagram illustrates how ConText servers with different personality masks access the Text Request Queue.

ConText servers process text requests in the Text Request Queue in the following order:

Administration --> Query --> DDL --> DML --> Services

When a ConText server finishes processing a request of any type, it always checks the pipes and queues for the next pending request according to the precedence order.

The Text Request Queue is made up of the following database pipes and tables:

- pipes assigned to each ConText server for performing administrative (DBA) tasks
- pipes for storing Query and DDL requests
- tables for storing DML and Linguistics requests

**See Also:** For more information about the tables used to process DML, see “DML Queue” in this chapter.

For more information about the tables used to process Linguistic requests, see “Services Queue” in this chapter.

## Administration Pipes

Each ConText server has a dedicated administration pipe for administrative tasks that control its operation (e.g. server shutdown).

## Query Pipe

When a SQL statement or PL/SQL block includes a text query, the system dispatches the query to the query pipe.

The ConText servers notify the calling client when the operation is finished.

## DDL Pipe

ConText dispatches all requests for DDL operations (e.g. CREATE\_INDEX, DROP\_INDEX, and OPTIMIZE\_INDEX) to the DDL pipe.

The ConText servers notify the calling client when the operation is finished.

If a ConText server encounters a problem with a request in the DDL pipe, the error does not affect the pipe or the server processing the pipe. The errored request is recorded as a row in the Services Queue and the server continues processing the remaining requests in the pipe.

The CTX\_INDEX\_ERRORS view can be used to display errored DDL requests.

## DML Queue

The ConText DML Queue stores requests for index update when changes are made to a table containing a text column with an existing index.

When a DML operation is performed (i.e. data in a text table is modified, deleted, or added), a request for an index update is automatically recorded in the DML Queue. Requests are placed in the DML Queue by internal database triggers that are created automatically the first time a ConText index is generated by a ConText server for the text column.

If DML processing is running in immediate mode, the next available ConText server with the DML (M) personality picks up the requests and updates the index as soon as resources and system load allow.

If DML processing is running in batch mode, the requests are stored in the queue until a user explicitly requests DML processing. Then, available ConText servers with the DDL (D) personality pick up all the pending requests and process the requests as one or more batches.

The DML Queue consists of the following internal tables:

- Pending Table (DRQ\_PENDING)
- In-Progress Table (DRQ\_INPROG)
- Waiting Table (DRQ\_WAITING)
- Batches Table (DRQ\_BATCHES)

---

**Note:** The DML tables are internal tables and should *not* be accessed directly. To view the queue, use the queue views or the GUI administration tools provided with the ConText Workbench. To administer the queue, use the CTX\_DML package.

For more information about DML operations, see “DML” in Chapter 6, “Text Concepts”.

For more information about the DML queue views, see “ConText Queue Views” in Appendix B, “ConText Views”.

For more information about the CTX\_DML package, see “CTX\_DML: ConText Index Update” in Chapter 11, “PL/SQL Packages - Text Management”.

---

## Pending Table (DRQ\_PENDING)

This table contains one row for each DML operation (request for index update) that has not yet been picked up by a ConText server. The row indicates the specific cell that has changed in the text table.

---

**Note:** Requests are visible in the pending table only after the insert, update, or delete has been committed.

---

When a request has been picked up by a ConText server with the DML or DDL personality, the row is deleted from the pending table and the server begins the index update. In addition, a row is written to the in-progress table to indicate that the request is being processed.

## In-Progress Table (DRQ\_INPROG)

This table contains one row for each DML request being processed by a ConText server.

Once the ConText server finishes processing the request, the row is deleted from the in-progress table, indicating that the appropriate index has been updated to reflect the document modification that generated the request.

## Waiting Table (DRQ\_WAITING)

This table contains a row for each DML request for which a duplicate request exists in the pending table. This condition occurs when a DML request for a row has not been picked up by a ConText server and additional requests for the row are issued. This table ensures that DML requests for a row in a text table are processed in order.

## Batches Table (DRQ\_BATCHES)

This table contains one row for each batch of DML requests. A batch consists of up to 10,000 DML requests for an indexed column. If the queue contains more than 10,000 DML requests for a column or requests for different columns, ConText uses multiple batches to process the requests.

For each batch, the table stores information such as the number of rows in the batch, the batch ID, and the ID of the server processing the batch.

If immediate DML is enabled, batches are created automatically as ConText servers with the DML personality pick up requests from the queue. If batch DML is enabled, batches are created by users calling CTX\_DML.SYNC.

**See Also:** For more information about immediate and batch DML, see “DML” in Chapter 6, “Text Concepts”

## Timestamps

Requests in the DML Queue are processed in the order they are received, based on a time-stamp. Rows are inserted into the pending table without a timestamp. At a specified time interval, all unmarked rows within the pending table are marked with a time-stamp. The timestamp is based on the time each change was committed.

## Error Handling

If a ConText server encounters a problem with a request in the DML Queue, the error does not affect the queue or the server processing the queue. The errored request is recorded as a row in the Services Queue and the server continues processing the remaining requests in the queue.

The `CTX_INDEX_ERRORS` view of the Services Queue can be used to display errored DML requests.

## Queue Management

The DML Queue can be enabled/disabled manually to control processing of DML requests. When the DML Queue is disabled, the queue continues to accept requests; however, new requests and any pending requests in the disabled DML Queue are not picked up by ConText servers until the queue is enabled manually.

## Services Queue

The Services Queue is used for processing requests for all ConText services. The Services Queue is designed to be extensible. As additional services are provided by ConText, the Services Queue is the mechanism by which the services will be managed. Currently, the Services Queue supports the following services:

- requests for the ConText Linguistics (theme, Gist, and theme summary generation)
- error handling for index creation and optimization (DDL) and index updating (DML)

When a request is submitted for the Linguistics, the request is stored in the Services Queue. A request is picked up by the first available ConText server with the Linguistic personality and the server generates linguistic output for the specified request.

ConText servers with the Linguistic personality pick requests out of the queue based on the request priority and creation time-stamp. Clients may queue a request and continue to work while the request is being processed.

The Services Queue is asynchronous. Clients that place a request in the queue do not automatically block their subsequent requests while waiting for a reply. However, clients can choose to block their subsequent requests for a specified time when they submit requests.

### Services Queue Table (CTX\_SVCQ)

The Services Queue consists of the CTX\_SVCQ internal table. This table stores a row for each request for the ConText Linguistics, as well as the request status.

---

**Note:** CTX\_SVCQ is an internal table and should *not* be accessed directly. To view the queue, use the queue views or the GUI administration tools provided with the ConText Workbench.

To administer the Services Queue (e.g. cleaning up entries), use the CTX\_SVC package or the GUI administration tools.

For more information about the CTX\_SVC package, see “CTX\_DDL: Text Setup and Management” in Chapter 11, “PL/SQL Packages - Text Management”.

---



## Error Handling

If a ConText server encounters a problem with a request in the Services Queue, the error does not affect the queue or the server processing the queue. The errored request is recorded as a row in the Services Queue and the server continues processing the remaining requests in the queue.

The CTX\_LING\_ERRORS view of the Services Queue can be used to display errored requests for Linguistics.

## Queue Management

The Services Queue can be enabled/disabled manually to control processing of Linguistics requests. When the Services Queue is disabled, requests are still accepted into the queue; however, new requests and any pending requests in the disabled Services Queue are not picked up by ConText servers until the queue is enabled manually.



---

# Administering ConText

This chapter provides details on how to administer ConText users, servers, and queues from the command-line.

The process of administering ConText can be divided into three main tasks:

- Enabling One-step Queries
- Managing Users
- Managing ConText Servers
- Managing ConText Queues

Management of ConText users can be performed by any Oracle DBA user or the CTXSYS user. Management of ConText servers and queues is performed by the CTXSYS user.

---

**Note:** Many of the ConText server and queue administration tasks can be performed from the GUI administration tools (System Administration tool or Configuration Manager). These tasks are diagramed in each section of the chapter.

---

## Enabling One-step Queries

If you want to use one-step queries in ConText, you must set the ConText initialization parameter `TEXT_ENABLE` for each database instance to `TRUE`. `TEXT_ENABLE` enables Oracle8 to recognize the `CONTAINS` SQL function utilized in one-step queries.

You can set `TEXT_ENABLE` for all users and sessions in the `initSID.ora` file. You can also set `TEXT_ENABLE` for the current session using the SQL command, `ALTER SESSION`.

---

---

**Note:** `TEXT_ENABLE` only needs to be set once for each database instance. Also, once you have set `TEXT_ENABLE`, start one or more ConText servers with the Query (Q) personality to ensure one-step queries are processed.

---

---

**See Also:** For the location of the `initSID.ora` file, see the Oracle8 installation documentation specific to your operating system.

For more information about setting initialization parameters, see *Oracle8 Server Administrator's Guide*.

### Setting `TEXT_ENABLE` for All Users

To set `TEXT_ENABLE` for all users, perform the following steps:

1. Shut down the database.
2. Add the following line to the `initSID.ora` file:

```
text_enable=true
```

3. Restart the database

---

---

**Note:** Once you set `TEXT_ENABLE` in the `initSID.ora` file, one-step queries are enabled each time you start up a database instance.

---

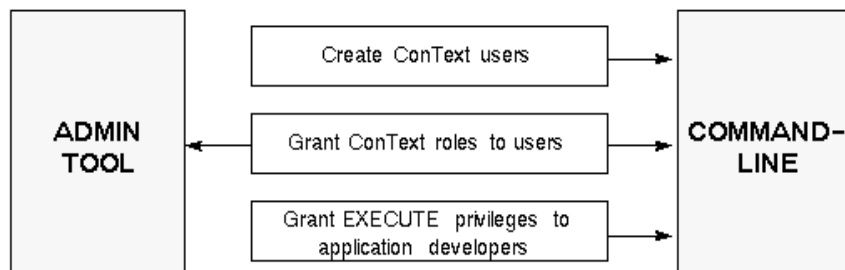
---

### Setting `TEXT_ENABLE` for the Session

To set `TEXT_ENABLE` for the current session only, issue the following SQL command:

```
alter session set text_enable = true
```

## Managing Users



This section provides details for performing the following user administration tasks from the command-line:

- Creating ConText Users
- Granting ConText Roles to Users
- Granting EXECUTE Privileges to Application Developers

---

**Note:** In addition to these tasks, a ConText administrator may be responsible for importing/exporting the database schema objects for ConText users.

ConText does *not* currently support the importing/exporting of schema objects (tables and Oracle indexes) for which ConText indexes have been created.

To replicate a ConText user's database schema in another database, a full export/import of the user's schema, as well as the CTXSYS user's schema would have to be performed, then the ConText indexes would have to be recreated in the new schema.

---

## Creating ConText Users

ConText provides a predefined Oracle user called CTXSYS for performing system and database administration.

To create additional Oracle users for ConText, log in to SQL\*Plus as an Oracle DBA and use the SQL command CREATE USER.

For example:

```
create user app_dev  
identified by 123abc  
default tablespace app_tables;
```

---

---

**Note:** Do *not* use PL/SQL and SQL reserved words in usernames. In addition, certain words, such as *ascii*, *html*, *blaster*, and *filter*, are used internally by ConText and should *not* be used by themselves as usernames.

---

---

**See Also:** For more information about creating Oracle users, see *Oracle8 Server SQL Reference*.

## Granting ConText Roles to Users

To assign a ConText role to a user, log in to SQL\*Plus as an Oracle DBA and use the SQL command GRANT. For example:

```
grant ctxapp to ctxdev;
```

**See Also:** For more information about the ConText roles, see “ConText Roles” in Chapter 2, “Administration Concepts”.

For more information about granting roles to users, see *Oracle8 Server SQL Reference*.

## Granting EXECUTE Privileges to Application Developers

To enable users (i.e. application developers) to call procedures from within their own stored procedures and triggers, you must explicitly grant to each user EXECUTE privileges for the ConText PL/SQL packages that contain the procedures.

To grant EXECUTE privileges to users, log in to SQL\*Plus as CTXSYS and use the GRANT command. For example:

```
grant execute on ctx_query to ctxdev;
```

In this example, CTXSYS grants EXECUTE privileges to the user CTXDEV for all the stored procedures in the CTX\_QUERY package.

**See Also:** For more information about granting privileges to users, see *Oracle8 Server SQL Reference*.

### Application Development Packages

The ConText packages for which users may need EXECUTE privileges are:

- CTX\_LING: Linguistics
- CTX\_QUERY: Query and Highlighting

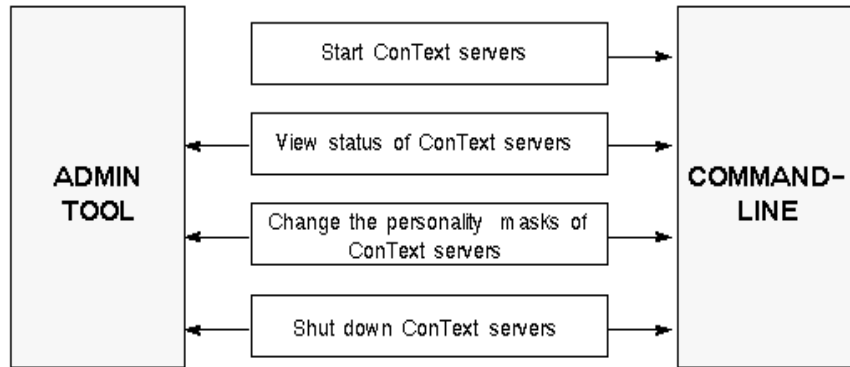
**See Also:** For more information about the CTX\_QUERY and CTX\_LING packages, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Administration Packages

If the application developer is building administration functionality into an application, they may need EXECUTE privileges on the remaining ConText packages

- CTX\_ADM: ConText Administration
- CTX\_DDL: Text Setup and Management
- CTX\_DML: ConText Index Update
- CTX\_INFO: Product Information
- CTX\_SVC: Services Queue Administration
- CTX\_THES: Thesaurus Management

## Managing ConText Servers



This section provides details for performing the following ConText server administration tasks from the command-line:

- Starting ConText Servers
- Viewing the Status of ConText Servers
- Shutting Down ConText Servers
- Changing the Personality Masks of ConText Servers



## Starting ConText Servers

You can start ConText servers using a number of methods, all from the command-line:

- Using `ctxsrv`
- Using `ctxsrv` (Masking the Password for CTXSYS)
- Using `ctxctl`

---

**Suggestion:** If your machine supports running multiple ConText servers, to prevent contention between text operations, you should start multiple servers, each with a different personality mask.

For example, if your machine supports running four servers, you could designate one server for processing queries, one server for processing DML and DDL, one server for linguistics, and one for text loading.

In addition, the Linguistic personality is only required for generating linguistic output. Once the output has been generated, the separate Linguistic server could be shut down or reassigned to other text operations.

---

### Using `ctxsrv`

To start a ConText server, run the `ctxsrv` executable from the command-line of the server machine.

For example, in a UNIX-based environment, to start a ConText server in the background with a personality mask of D (DDL) and L (Linguistics), type the following command on the command-line of the server host machine:

```
ctxsrv -user ctxsys/password -personality DL &
```

**See Also:** For more information about `ctxsrv`, see “`ctxsrv` Executable” in Chapter 4, “ConText Server Executables and Utilities”.

For more information about the text operations that ConText servers can process, see “Text Operations” in Chapter 6, “Text Concepts”.

### Using ctxsrv (Masking the Password for CTXSYS)

When `-user` is specified in the command-line for `ctxsrv`, the password for CTXSYS is visible to users of the machine on which the ConText server process is running.

If you wish to mask the password from users, you can run `ctxsrv` without specifying `-user`. The required user information can be supplied in two ways:

- system-provided prompt
- text file (containing the user information)

**system-provided prompt** If you do not specify the `-user` argument, ConText prompts you to enter user information. The information must be entered in the format 'CTXSYS/*password*' where *password* is the password for CTXSYS.

The disadvantage of using this method is ConText servers cannot be run as background processes in environments that support background processes.

**text file** If your environment supports it, you can pass the required information for `-user` to `ctxsrv` through a plain text file.

The file must contain a single line of text in the format 'CTXSYS/*password*' where *password* is the password for CTXSYS.

For example, in a UNIX-based environment, the following command starts a ConText Server with the Loader (R) personality. The user information is passed to `ctxsrv` through a file named *pword.txt*.

```
ctxsrv -personality R < pword.txt
```

The advantage of using this method is ConText servers can be run as background processes in environments that support background processes. In addition, this method provides the highest level of security for masking the password.

### Using ctxctl

The `ctxctl` utility provides a command-line interface for starting, shutting down, and viewing the status of your ConText servers.

To start the `ctxctl` utility, type the following command on the command-line of the server host machine:

```
ctxctl
```

Then, use the *start* command at the `ctxctl` command prompt to start ConText servers. For example, to start a single ConText server with the Loader personality, type:

```
command> start 1 load
```

**See Also:** For more information about ctxctl, see “ctxctl Utility” in Chapter 4, “ConText Server Executables and Utilities”.

## Viewing the Status of ConText Servers

You can view the status of currently running ConText servers using ctxctl. You can also use the CTX\_SERVERS or CTX\_ALL\_SERVERS views to monitor the status of ConText servers.

### Using ctxctl

To view the status of ConText servers, start the ctxctl utility by typing the following command on the command-line of the server host machine:

```
ctxctl
```

Then, at the ctxctl command prompt, use the *status* command:

```
command> status
```

The *status* command display results similar to the following:

PID	LING.	QUERY	DDL	DML	LOAD
23266	X	X			
23285		X	X	X	
Total	1	2	1	1	0

### Using the Server Views

To view all the currently running ConText servers using CTX\_ALL\_SERVERS, issue the following SQL statement:

```
column ser_name format a30
select ser_name, ser_status, ser_started_at
from ctx_all_servers;
```

If a ConText server is running, the query will display results similar to the following output:

SER_NAME	SER_STAT	SER_START
-----	-----	-----
DRSRV_1120	IDLE	18-MAR-97

## Changing the Personality Masks of ConText Servers

To change the personality mask for a ConText server, use the PL/SQL procedure `CTX_ADM.CHANGE_MASK`.

For example:

```
execute ctx_adm.change_mask('DRSRV_1120','QD')
```

This example illustrates a personality mask consisting of the Query (Q) and DDL (D) personalities replacing the existing personality mask for the ConText server.

Also, in this example, *drsrv\_1120* is the name (identifier) for the ConText server. A server identifier is generated automatically when you start up a ConText server. You can use the `ctxctl` utility or the `CTX_ALL_SERVERS` view to obtain the identifier for a ConText server.

## Shutting Down ConText Servers

To shut down a ConText server, use `CTX_ADM.SHUTDOWN`.

For example:

```
execute ctx_adm.shutdown ('DRSRV_1120',1)
```

In this example, *drsrv\_1120* is the name (identifier) of the ConText server and the shutdown method is 1 (immediate). A server identifier is generated automatically when you start up a ConText server. You can use the `ctxctl` utility or the `CTX_ALL_SERVERS` view to obtain the identifier for a ConText server.

---

---

**Note:** You do not need to specify a server identifier when calling `CTX_ADM.SHUTDOWN`. If you do not specify an identifier, `SHUTDOWN` shuts down all currently running ConText servers. For example:

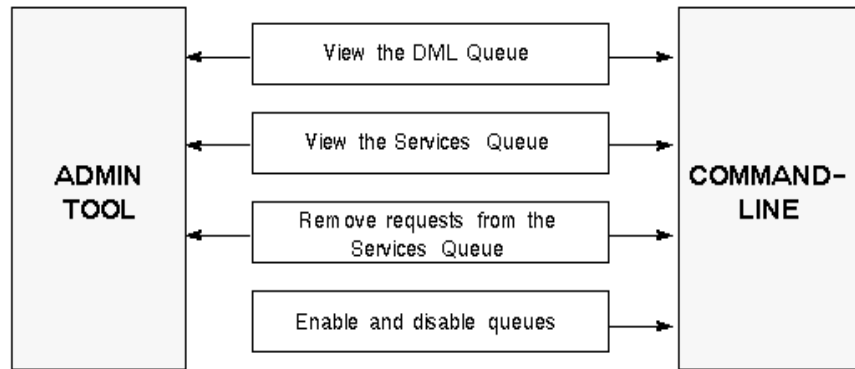
```
execute ctx_adm.shutdown
```

---

---

You can also use the `ctxctl` utility to shutdown ConText servers.

## Managing ConText Queues



This section provides details for performing the following ConText server administration tasks from the command-line:

- Viewing the DML Queue
- Viewing the Services Queue
- Removing Requests from the Services Queue
- Enabling and Disabling Queues

## Viewing the DML Queue

You can view the status of requests in the DML Queue, using the following views provided by ConText:

- CTX\_ALL\_DML\_QUEUE
- CTX\_ALL\_DML\_SUM
- CTX\_ALL\_QUEUE
- CTX\_USER\_DML\_QUEUE
- CTX\_USER\_DML\_SUM
- CTX\_USER\_QUEUE

## Viewing the Services Queue

To view the status of requests in the Services Queue, you can use the CTX\_SVC.REQUEST\_STATUS function. For example:

```
declare status varchar2(10);
declare timestamp date;
declare errors varchar2(60);
begin
    status := ctx_svc.request_status(3341,timestamp,errors);
    dbms_output.put_line(status,timestamp,substr(errors,1,20);
end;
```

In this example, variables are defined for *status*, *timestamp*, and *errors*. Then, REQUEST\_STATUS is called to return the status of the request with *handle* 3341 and the DBMS\_OUTPUT package is used to display the results of the output.

A handle is generated automatically when a request is submitted to the Services Queue using CTX\_LING.SUBMIT.

**See Also:** For more information about submitting requests to the Services Queue and using the CTX\_LING package, see *Oracle8 ConText Cartridge Application Developer's Guide*.

For more information about the DBMS\_OUTPUT package, see *Oracle8 Server Application Developer's Guide*.

## Removing Requests from the Services Queue

You can remove pending and errored requests from the Services Queue using procedures in the CTX\_SVC package.

### Pending Requests

To remove a request with a status of PENDING, use the CTX\_SVC.CANCEL procedure. For example:

```
execute ctx_svc.cancel(3341)
```

In this example, the request with *handle* 3341 is removed from the Services Queue.

In addition, you can use the CTX\_SVC.CANCEL\_ALL procedure to remove all requests with a status of PENDING from the Services Queue. For example:

```
execute ctx_svc.cancel_all
```

### Errored Requests

To remove a request with a status of ERROR, use the CTX\_SVC.CLEAR\_ERROR procedure. For example:

```
execute ctx_svc.clear_error(3341)
```

In addition, you can use the CTX\_SVC.CLEAR\_ALL\_ERRORS procedure to remove all requests with a status of ERROR from the Services Queue. For example:

```
execute ctx_svc.clear_all_errors
```

You can also use the CLEAR\_INDEX\_ERRORS or CLEAR\_LING\_ERRORS to remove all errored indexing/Linguistics requests from the Services Queue.

## Enabling and Disabling Queues

You can enable or disable the following queues in the Text Request Queue:

- Text Queue (DDL and Query pipes)
- DML Queue
- Services Queue

If a queue is disabled, pending requests in the queue are not processed by ConText servers.

---

---

**Note:** A disabled queue continues to accept requests. The queues should be monitored regularly to prevent an excessive backlog of pending requests.

To obtain the status of a queue, use the CTX\_ADM.GET\_QUEUE\_STATUS function.

---

---

**See Also:** For more information about the Text Request Queue, see “Text Request Queue” in Chapter 2, “Administration Concepts”.

### Enabling Queues

To enable a queue, use the CTX\_ADM.UPDATE\_QUEUE\_STATUS procedure. For example:

```
execute ctx_adm.update_queue_status(ctx_adm.DML_QUEUE, ctx_adm.ENABLE_QUEUE)
```

In this example, the DML Queue is enabled, which allows entries in the queue to be processed by ConText servers.

### Disabling Queues

To disable a queue, use the CTX\_ADM.UPDATE\_QUEUE\_STATUS procedure. For example:

```
execute ctx_adm.update_queue_status(ctx_adm.TEXT_QUEUE, ctx_adm.DISABLE_QUEUE)
```

In this example, the Text Queue (DDL and Query pipes) is disabled, which prevents all text/theme queries and DDL requests from being processed by ConText servers.



---

## ConText Server Executables and Utilities

This chapter provides reference information for using the ConText server executables and utilities provided with ConText.

The executables and utilities discussed in this chapter are:

- ctxsrv Executable
- ctxctl Utility

## ctxsrv Executable

The ctxsrv executable starts ConText servers. You execute ctxsrv for each ConText server that you want to start.

---

**Note:** ctxsrv can *only* be executed by the Oracle user, CTXSYS.

---

You can also use the ctxctl utility to start and shut down ConText servers.

**See Also:** For more information about the CTXSYS user, see “CTXSYS User” in Chapter 2, “Administration Concepts”.

For more information about ctxctl, see “ctxctl Utility” in this chapter.

## Syntax

The syntax for ctxsrv:

```
ctxsrv -user ctxsys/passwd[@sqlnet_address]
      [-personality RQDML]
      [-logfile log_name]
      [-sqltrace]
```

where:

**-user**

specifies the username and password for the Oracle user CTXSYS.

The username and password may be immediately followed by *@sqlnet\_address* to permit logon to remote databases. The value for *sqlnet\_address* is a database connect string. If the TWO\_TASK environment variable is set to a remote database, you do not have to specify a value for *sqlnet\_address* to connect to the database.

---

**Note:** If you do not specify *user* in the ctxsrv command-line, you are prompted by ConText to enter the required information in the format: 'CTXSYS/*password*' where *password* is the password for CTXSYS.

---

This is useful if you wish to mask the CTXSYS password from other users of the machine on which the ConText server is running.

---

**-personality**

specifies the personality mask for the ConText server started by ctxsrv. The possible values can be any combination of:

- R (Loader personality)
- Q (Query personality)
- D (DDL personality)
- M (DML personality)
- L (Linguistic personality)

The default is QDM.

---

---

**Note:** Oracle does not recommend assigning all the personalities to a single ConText server. This will result in the server bearing the majority of the processing load.

---

---

**-logfile**

specifies the name of a log file to which the ConText server writes all session information and errors.

**-sqltrace**

enables the ConText server to write to a trace file in the directory specified by the USER\_DUMP\_DEST initialization parameter.

Before you specify -sqltrace for ctxsrv, you should specify a value for USER\_DUMP\_DEST in your *initsid.ora* file.

**See Also:** For more information about SQL trace and the USER\_DUMP\_DEST initialization parameter, see *Oracle8 Server Administrator's Guide*.

## Examples

The following example starts a ConText server with a Query and DDL personality mask and writes all server messages to a file named *ctx.log*:

```
ctxsrv -user ctxsys/ctxsys -personality QD -log ctx.log &
```

---

**Note:** In this example, the server is run as a background process in a UNIX-based environment. This is useful if you need to use the window/screen from which you started the server for other tasks.

---

The following example starts a linguistically-enabled ConText server with a Linguistic personality and writes all server messages to a file named *ctx.log*. Because *-user* is not specified, ConText prompts you to enter a user:

```
ctxsrv -personality L -log ctx.log

...
ConText: Release 2.0.6.0.0 - Production on Sat Jun  7 14:06:26 1997
...
Copyright (c) Oracle Corporation 1979, 1996. All rights reserved.
...
Enter user:
```

At the prompt, enter 'CTXSYS/*password*', where *password* is the password assigned to the CTXSYS user.

---

**Note:** In this example, the process is *not* run in the background.

In environments where you can run processes in the background, if you do not specify *-user* in the ctxsrv command-line, you must run the server process in the foreground or pass a value for *-user* to ctxsrv from an operating system file.

For example:

```
ctxsrv -personality L -log ctx.log < pword.txt
```

The file must contain a single line consisting of the following text:  
'CTXSYS/*password*'

If you pass a value to ctxsrv from a file, ConText does not prompt you to enter a user.

---

## ctxctl Utility

The ctxctl utility is a shell script that can be used to start up and shut down ConText servers on the system from which you run ctxctl. It can also be used to check the status of all the ConText servers currently running on the system.

### Syntax

To start ctxctl, at the operating system prompt, type:

```
ctxctl
```

Once ctxctl is running, you can issue the following commands from the ctxctl command prompt:

#### **help [command]**

Provides online help for the specified command. If called without a command, it provides a list of all the commands you can use in ctxctl.

#### **status**

Provides a list of all the ConText servers and their personality masks currently running on the server host.

---

---

**Note:** The ConText servers listed in the status output may be connected to different database instances.

---

---

#### **start *n* [load query ddl dml ling]**

Starts *n* number of servers, each with the personalities specified. The personalities can be typed in any order, but must be typed in lowercase and exactly as they are named (e.g. *load*, *query*, *ddl*, *dml*, *ling*).

If you do not specify a personality, ctxctl starts the specified number of servers, each with the *query*, *ddl*, and *dml* personalities.

The first time you type the *start* command for a ctxctl session, ConText prompts you to enter the password for the ConText administrator (CTXSYS). After you enter the password, ConText starts the specified number of servers.

---

---

**Note:** The ConText server(s) are started on the host machine from which the start command is issued.

---

---

**stop *pid* | all**

Shuts down the ConText server identified by *pid* or shuts down all ConText servers (*all*).

The *status* command can be used to obtain the *pid* for all currently running ConText servers.

---

---

**Note:** ctxctl does not use CTX\_ADM.SHUTDOWN to shut down the ConText server. Instead, it aborts the server process running on the host machine.

---

---

**quit | exit**

Terminates ctxctl and returns you to the command-line of the host machine.

## Examples

The following example starts two ConText servers, each with a DML, DDL, and Query personality mask:

```
command> start 2 query dml ddl
```

The following example shuts down a ConText server with a *pid* of 230454:

```
command> stop 23054
```

---

## PL/SQL Packages - Administration

This chapter provides reference information for using the PL/SQL packages provided with ConText to administer ConText servers and queues, as well as to obtain product information about ConText.

The topics covered in this chapter are:

- CTX\_ADM: ConText Administration
- CTX\_SVC: Services Queue Administration
- CTX\_INFO: Product Information

## CTX\_ADM: ConText Administration

The CTX\_ADM PL/SQL package is used to manage ConText servers and queues.

CTX\_ADM contains the following stored procedures and functions:

Name	Description
CHANGE_MASK	Modifies the personality mask for a ConText server
GET_QUEUE_STATUS	Returns the status of the specified queue
RECOVER	Cleans up database objects for deleted text tables
SET_QUERY_BUFFER_SIZE	Increases the size of the pipe used for queries
SHUTDOWN	Shuts down a single ConText server or all currently running ConText servers
UPDATE_QUEUE_STATUS	Updates the status of the specified queue



## CHANGE\_MASK

The CHANGE\_MASK procedure changes the personality mask of the specified ConText server.

### Syntax

```
CTX_ADM.CHANGE_MASK(name          IN VARCHAR2  
                    personality_mask IN VARCHAR2 DEFAULT 'QDM');
```

#### **name**

Specify the name (internal identifier) of the server for which you are changing the personality mask.

#### **personality\_mask**

Specify the new personality mask that you want to assign to the server. Can be any combination of:

- R (Loader)
- Q (Query)
- D (DDL)
- M (DML)
- L (Linguistic)

Default is QDM.

### Examples

```
execute ctx_adm.change_mask('DRSRV_8025', 'D')
```

### Notes

The names of all currently running ConText servers can be obtained from the CTX\_SERVERS or CTX\_ALL\_SERVERS views.

---

## GET\_QUEUE\_STATUS

The GET\_QUEUE\_STATUS function returns the status of the specified ConText queue.

### Syntax

```
CTX_ADM.GET_QUEUE_STATUS(qname IN VARCHAR2)
RETURN VARCHAR2;
```

#### **qname**

Specify the queue/pipe for which you want to return the status:

- TEXT\_QUEUE (DDL and Query pipes)
- DML\_QUEUE
- SERVICES\_QUEUE

### Returns

Status of the queue, which is one of the following:

#### **ENABLED**

The specified queue is enabled.

#### **DISABLED**

The specified queue is disabled.

### Examples

```
declare status varchar2(8);
begin
    status := ctx_adm.get_queue_status('DML_QUEUE');
end;
```

## Notes

A status of **DISABLED** indicates the queue or pipe is inactive and requests in the queue will not be processed by any of the available ConText servers.

When a queue or pipe has a status of **DISABLED**, the queue continues to accept requests. The ConText administrator should regularly monitor the status of the queues and pipes to prevent accumulation of requests in disabled queues.

To enable a disabled queue, you must call `CTX_ADM.UPDATE_QUEUE_STATUS`.

---

## RECOVER

The RECOVER procedure deletes all database objects for text tables that have been deleted without first dropping the index or policies for the tables.

### Syntax

```
CTX_ADM.RECOVER;
```

### Examples

```
execute ctx_adm.recover
```

### Notes

ConText Servers automatically perform recovery approximately every fifteen minutes. CTX\_ADM.RECOVER provides a method for users to manually perform recovery on command.

---

## SET\_QUERY\_BUFFER\_SIZE

The SET\_QUERY\_BUFFER\_SIZE procedure sets the size of the database pipe used for queries.

### Syntax

```
CTX_ADM.SET_QUERY_BUFFER_SIZE(buffer_size IN NUMBER);
```

#### **buffer\_size**

Specify the size, in bytes, of the query buffer.

### Examples

```
execute ctx_adm.set_query_buffer_size(100000);
```

### Notes

The default size of the buffer is 8192 bytes.

CTX\_ADM.SET\_QUERY\_BUFFER\_SIZE can only be used to *increase* the size of the buffer from the default size.

---

## SHUTDOWN

The SHUTDOWN procedure shuts down the specified ConText server.

### Syntax

```
CTX_ADM.SHUTDOWN(name    IN VARCHAR2 DEFAULT 'ALL',  
                  sdmode IN NUMBER   DEFAULT NULL);
```

#### **name**

Specify the name (internal identifier) of the ConText server to shutdown.

Default is ALL.

#### **sdmode**

Specify the shutdown mode for the server:

- 0 or NULL (Normal)
- 1 (Immediate)
- 2 (Abort)

Default is NULL.

### Examples

```
execute ctx_adm.shutdown('DRSRV_3321', 1)
```

### Notes

If you do not specify a ConText server to shut down, CTX\_ADM.SHUTDOWN shuts down all currently running ConText servers.

The names of all currently running ConText servers can be obtained from the CTX\_SERVERS view.

## UPDATE\_QUEUE\_STATUS

The UPDATE\_QUEUE\_STATUS procedure is used to change the status of the specified ConText queue (Text, DML, or Services).

For example, the GET\_QUEUE\_STATUS returns a status of DISABLED for one of the queues. Once the error that caused the queue to become disabled is cleared, UPDATE\_QUEUE\_STATUS can be called with an action of ENABLE\_QUEUE to reactivate the queue.

UPDATE\_QUEUE\_STATUS can also be used to control request processing in the system. When you disable a queue, you prevent any currently running ConText servers from picking up queued requests until you enable the queue.

### Syntax

```
CTX_ADM.UPDATE_QUEUE_STATUS(qname IN VARCHAR2,  
                             qstatus IN VARCHAR2 DEFAULT ENABLE_QUEUE);
```

#### **qname**

Specify the queue or pipe for which you want to return the status:

- TEXT\_QUEUE (DDL and Query pipes)
- DML\_QUEUE
- SERVICES\_QUEUE

#### **qstatus**

Specify the action to perform on the queue:

- DISABLE\_QUEUE
- ENABLE\_QUEUE

Default is ENABLE\_QUEUE.

### Examples

```
execute ctx_adm.update_queue_status(ctx_adm.dml_queue,ctx_adm.enable_queue)
```

## Notes

A queue with a status of DISABLED will remain inactive until it is enabled using UPDATE\_QUEUE\_STATUS; however, the queue will continue to accept requests. The ConText administrator should regularly monitor the status of the queues and pipes to prevent accumulation of requests in disabled queues.

Both *qname* and *qstatus* must be fully qualified with the PL/SQL package name (CTX\_ADM) as shown in the examples.



## CTX\_SVC: Services Queue Administration

The CTX\_SVC PL/SQL package is used to query requests in the Services Queue and to manage the queue.

CTX\_SVC contains the following stored procedures and functions:

Name	Description
CANCEL	Removes a pending request from the Services Queue
CANCEL_ALL	Removes all pending requests from the Services Queue
CANCEL_USER	Removes a pending request from the Services Queue for the current user
CLEAR_ALL_ERRORS	Removes all requests with an error status from the Services Queue
CLEAR_ERROR	Removes a request with an error status from the Services Queue
CLEAR_INDEX_ERRORS	Removes errored indexing requests from the Services Queue
CLEAR_LING_ERRORS	Removes errored Linguistics requests from the Services Queue
REQUEST_STATUS	Returns the status of a request in the Services Queue

---

## CANCEL

The CANCEL procedure removes a request with a status of PENDING from the Services Queue.

### Syntax

```
CTX_SVC.CANCEL(request_handle IN NUMBER);
```

#### **request\_handle**

Specify the handle, returned by CTX\_LING.SUBMIT, of the service request to remove.

### Examples

```
execute ctx_svc.cancel(3321)
```

### Notes

Requests with a status other than pending in the Services Queue cannot be removed using CTX\_SVC.CANCEL. To cancel requests that ConText has not yet entered into the Services Queue, use CTX\_LING.CANCEL.

**See Also:** For more information about the CTX\_LING PL/SQL package, see *Oracle8 ConText Cartridge Application Developer's Guide*.

---

## CANCEL\_ALL

The CANCEL\_ALL procedure removes all requests with a status of PENDING from the Services Queue.

### Syntax

```
CTX_SVC.CANCEL_ALL;
```

### Examples

```
execute ctx_svc.cancel_all
```

---

## CANCEL\_USER

The CANCEL\_USER procedure removes all requests with a status of PENDING for the current user from the Services Queue.

### Syntax

```
CTX_SVC.CANCEL_USER;
```

### Examples

```
execute cancel
```

---

## CLEAR\_ALL\_ERRORS

The `CLEAR_ALL_ERRORS` procedure removes all requests (text indexing, theme indexing, and linguistics) that have a status of `ERROR` in the Services Queue.

### Syntax

```
CTX_SVC.CLEAR_ALL_ERROR;
```

### Examples

```
execute ctx_svc.clear_all_errors
```

---

## CLEAR\_ERROR

The `CLEAR_ERROR` procedure can be used to remove a request with a status of `ERROR` from the Services Queue.

### Syntax

```
CTX_SVC.CLEAR_ERROR(request_handle IN NUMBER);
```

#### **request\_handle**

Specify the handle, returned by `CTX_LING.SUBMIT`, of the errored service request to remove.

**See Also:** For more information about `SUBMIT` and the `CTX_LING` PL/SQL package, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Examples

```
execute clear_error(214)
```

### Notes

If you call `CLEAR_ERROR` with a 0 (zero) value for *request\_handle*, all requests with a status of `ERROR` in the Services Queue are removed.

Use the `CTX_SVC.REQUEST_STATUS` function to return the status of a request in the Services Queue.

---

## CLEAR\_INDEX\_ERRORS

The `CLEAR_INDEX_ERRORS` procedure removes all indexing requests that have a status of `ERROR` in the Services Queue.

### Syntax

```
CTX_SVC.CLEAR_INDEX_ERROR;
```

### Examples

```
execute ctx_svc.clear_index_errors
```

---

## CLEAR\_LING\_ERRORS

The `CLEAR_LING_ERRORS` procedure removes all Linguistics requests that have a status of `ERROR` in the Services Queue.

### Syntax

```
CTX_SVC.CLEAR_LING_ERROR;
```

### Examples

```
execute ctx_svc.clear_ling_errors
```



## REQUEST\_STATUS

The REQUEST\_STATUS function returns the status of a request in the Services Queue.

### Syntax

```
CTX_SVC.REQUEST_STATUS(request_handle IN NUMBER,  
                        timestamp      OUT DATE,  
                        errors         OUT VARCHAR2)  
RETURN VARCHAR2;
```

#### **request\_handle**

Specify the handle of the service request, as returned by CTX\_LING.SUBMIT.

#### **timestamp**

Returns the time at which request was submitted.

#### **errors**

Returns the error message stack for the request. The message stack is returned only if the status of the request is ERROR.

**See Also:** For more information about SUBMIT and the CTX\_LING PL/SQL package, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Returns

Status of the request, which is one of the following:

#### **PENDING**

The request has not yet been picked up by a ConText server.

#### **RUNNING**

The request is being processed by a ConText server.

#### **ERROR**

The request encountered an error (see *errors* argument).

#### **SUCCESS**

The request completed successfully.

## Examples

```
declare status varchar2(10);
declare timestamp date;
declare errors varchar2(60);
begin
    status := ctx_svc.request_status(3461,timestamp,errors);
    dbms_output.put_line(status,timestamp,substr(errors,1,20);
end;
```

## Notes

Specifying an invalid value for *request\_handle* causes CTX\_SVC.REQUEST\_STATUS to return a status of SUCCESS.

## CTX\_INFO: Product Information

The CTX\_INFO PL/SQL package is used to obtain information about the installed version of ConText.

CTX\_INFO contains the following stored procedures and functions:

Name	Description
GET_INFO	Returns the status and version number for the installed ConText
GET_STATUS	Returns the status of ConText
GET_VERSION	Returns the version number for the installed ConText

---

## GET\_INFO

The GET\_INFO procedure calls the GET\_VERSION and GET\_STATUS functions in CTX\_INFO to return version and status information for ConText.

### Syntax

```
CTX_INFO.GET_INFO(product IN VARCHAR2,  
                  version OUT VARCHAR2,  
                  status OUT VARCHAR2);
```

#### **product**

Specify the product code for which information is returned. Currently, the only valid value for *product* is OCO.

#### **version**

Specify the version of the product.

#### **status**

Specify the status of the product.

### Examples

```
declare  
version varchar2(20);  
status varchar2(20);  
begin  
    ctx_info.get_info('CTX_INFO.OCO', :version, :status);  
    dbms_output.put_line ('OCO version is '||version||');  
    dbms_output.put_line ('OCO status is '||status||');  
end;
```

### Notes

*product* must be fully qualified with the PL/SQL package name (CTX\_INFO) as shown in the examples.

## GET\_STATUS

---

The GET\_STATUS function returns the product status for ConText.

### Syntax

```
CTX_INFO.GET_STATUS(product IN VARCHAR2)  
RETURN VARCHAR2;
```

#### **product**

Specify the product for which a status returned. Currently, the only valid value for *product* is OCO.

### Returns

The product status for ConText.

### Examples

```
declare  
status varchar2(60);  
begin  
    status := ctx_info.get_status('CTX_INFO.OCO');  
    dbms_output.put_line ('OCO status is '||status||');  
end;
```

### Notes

*product* must be fully qualified with the PL/SQL package name (CTX\_INFO) as shown in the example.

---

## GET\_VERSION

The GET\_VERSION function returns the version number for the version of ConText.

### Syntax

```
CTX_INFO.GET_VERSION(product IN VARCHAR2)  
RETURN NUMBER;
```

#### **product**

Specify the product for which a version number is returned. Currently, the only valid value for *product* is OCO.

### Returns

The version number for ConText.

### Examples

```
declare  
version number;  
begin  
    version := ctx_info.get_version('CTX_INFO.OCO');  
    dbms_output.put_line ('OCO version is '||version||');  
end;
```

### Notes

*product* must be fully qualified with the PL/SQL package name (CTX\_INFO) as shown in the example.

# Part II

---

## Text Setup and Management





---

## Text Concepts

This chapter introduces the concepts necessary for understanding how text is setup and managed by ConText.

The following topics are discussed in this chapter:

- The ConText Data Dictionary
- Text Operations
- Text Columns
- Text Loading
- Text Storage
- External Text
- Text Filtering
- ConText Indexes
- Text Indexes
- Theme Indexes
- Base-letter Conversion
- Thesauri
- Types of Thesaural Relationships
- Document Sections

## The ConText Data Dictionary

ConText utilizes a data dictionary, separate from the Oracle data dictionary, for storing indexing options. The following objects are registered in the ConText data dictionary:

- policies (template and user-created) and the table and column to which each policy is assigned
- indexing preferences (default and user-created)
- sources for automated batch-loading of text into database columns
- text loading preferences
- Tiles

The ConText data dictionary also stores resource limits and the status of all ConText servers that are currently running.

The ConText data dictionary is owned by the Oracle user CTXSYS. CTXSYS and the data dictionary tables and views are created during installation of ConText.

## Text Operations

ConText supports five types of operations that are processed by ConText servers:

- Text Loading
- DDL
- DML
- Text/Theme Queries
- Linguistics Requests

---

---

**Note:** The personality mask for a ConText server determines which operations the server can process.

For more information about personality masks, see “Personality Masks” in Chapter 2, “Administration Concepts”.

---

---

### Text Loading

Text loading is an ongoing operation performed by ConText servers running with the Loader personality. It differs from the other text operations in that a request is not made to the Text Request Queue for handling by the appropriate ConText server.

Instead, ConText servers with the Loader personality regularly scan a document repository (i.e. operating system directory) for documents to be loaded into text columns for indexing.

If a file is found in the directory, the contents of the file are automatically loaded by the ConText server into the appropriate table and column.

**See Also:** For more information about text loading using ConText servers, see “Automated Batch Loading” in this chapter.

### DDL

A ConText DDL operation is a request for the creation, deletion, or optimization of a text/theme index on a text column. DDL requests are sent to the DDL pipe in the Text Request Queue, where available ConText servers with the DDL personality pick up the requests and perform the operation.

DDL operations are requested through the GUI administration tools (System Administration or Configuration Manager) or the CTX\_DDL package.

**See Also:** For more information about the CTX\_DDL package, see “CTX\_DDL: Text Setup and Management” in Chapter 11, “PL/SQL Packages - Text Management”.

## DML

A text DML operation is a request for the ConText index (text or theme) of a column to be updated. An index update is necessary for a column when the following modifications have been made to the table:

- insertion of a new row
- deletion of an existing row
- update of the primary key or text column(s) for an existing row

Requests for index updates are stored in the DML Queue where they are picked up and processed by available ConText servers. The requests can be placed on the queue automatically by ConText or they can be placed on the queue manually.

In addition, the system can be configured so DML requests in the queue are processed immediately or in batch mode.

### Automatic DML Queue Notification

DML requests are automatically placed in the queue via an internal trigger that is created on a table the first time a ConText index is created for a text column in the table.

ConText supports disabling automatic DML at index creation time through a parameter, *create\_trig*, for CTX\_DDL.CREATE\_INDEX. The *create\_trig* parameter specifies whether the DML trigger is created/updated during indexing of the text column in the column policy.

In addition, a DML trigger can be removed at any time from a table using CTX\_DDL.DROP\_INTTRIG.

---

**Note:** DROP\_INTTRIG deletes the trigger for the table. If the table contains more than one text column with existing ConText indexes, automatic DML is disabled for *all* the text columns.

DROP\_INTTRIG is provided mainly for maintaining backward compatibility with previous releases of ConText and should be used *only* when it is absolutely necessary to disable automatic DML for *all* the text columns in a table.

---

If the DML trigger is not created during indexing or is dropped, the ConText index is not automatically updated when subsequent DML occurs for the table. Manual DML can always be performed, but automatic DML can only be reenabled by first dropping, then recreating the ConText index or creating your own trigger to handle updates.

### **Manual DML Queue Notification**

DML operations may be requested manually at any time using the CTX\_DML.REINDEX procedure, which places a request in the DML Queue for a specified document.

### **Immediate DML Processing**

In immediate mode, one or more ConText servers are running with the DML personality. The ConText servers regularly poll the DML Queue for requests, pick up any pending requests (up to 10,000 at a time) for an indexed column and update the index in real-time.

In this mode, an index is only briefly out of synchronization with the last insert, delete, or update that was performed on the table; however, immediate DML processing can use considerable system resources and create index fragmentation.

### **Batch DML Processing**

If a text table has frequent updates, you may want to process DML requests in batch mode. In batch mode, *no* ConText servers are running with the DML personality. The queue continues to accept requests, but the requests are not processed because no DML servers are available.

To start DML processing, the CTX\_DML.SYNC procedure is called. This procedure batches all the pending requests for an indexed column in the queue and sends them to the next available ConText server with a DDL personality. Any DML requests that are placed in the queue after SYNC is called are not included in the batch. They are included in the batch that is created the next time SYNC is called.

SYNC can be called with a level of parallelism. The level of parallelism determines the number of batches into which the pending requests are grouped. For example, if SYNC is called with a parallelism level of two, the pending requests are grouped into two batches and the next two available DDL ConText servers process the batches.

Calling SYNC in parallel speeds up the updating of the indexes, but may increase the degree of index fragmentation.

### **Concurrent Index Creation**

A text column within a table can be updated while a ConText server is creating an index on the same text column. Any changes to the table being indexed by a ConText server are stored as entries in the DML Queue, pending the completion of the index creation.

After index creation completes, the entries are picked up by the next available DML ConText server and the index is updated to reflect the changes. This avoids a race condition in which the DML Queue request might be processed, but then overwritten by index creation, even though the index creation was processing an older version of the document.

## Text/Theme Queries

A text query is any query that selects rows from a table based on the contents of the text stored in the text column(s) of the table.

A theme query is any query that selects rows from a table based on the themes generated for the text stored in the text column(s) of the table.

---

---

**Note:** Theme queries are only supported for English-language text.

---

---

ConText supports three query methods for text/theme queries:

- Two-step Queries
- One-step Queries
- In-memory Queries

In addition, ConText supports Stored Query Expressions (SQEs).

Before a user can perform a query using any of the methods, the column to be queried must be defined as a text column in the ConText data dictionary and a text and/or theme index must be generated for the column.

**See Also:** For more information about text columns, see “Text Columns” in this chapter.

For more information about text/theme queries and creating/using SQEs, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Two-step Queries

In a two-step query, the user performs two distinct operations. First, the ConText PL/SQL procedure, CONTAINS, is called for a column. The CONTAINS procedure performs a query of the text stored in a text column and generates a list of the text-keys that match the query expression and a relevance score for each document. The results are stored in a user-defined table.

Then, a SQL statement is executed on the result table to return the list of documents (hitlist) or some subset of the documents.

### **One-step Queries**

In a one-step query, the ConText SQL function, CONTAINS, is called directly in the WHERE clause of a SQL statement. The CONTAINS function accepts a column name and query expression as arguments and generates a list of the textkeys that match the query expression and a relevance score for each document.

The results generated by CONTAINS are returned through the SELECT clause of the SQL statement.

### **In-memory Queries**

In an in-memory query, PL/SQL stored procedures and functions are used to query a text column and store the results in a query buffer, rather than in the result tables used in two-step queries.

The user opens a CONTAINS cursor to the query buffer in memory, executes a text query, then fetches the hits from the buffer, one at a time.

### **Stored Query Expressions**

In a stored query expression (SQE), the results of a query expression for a text column, as well as the definition of the SQE, are stored in database tables. The results of a SQE can be accessed within a query (one-step, two-step, or in-memory) for performing iterative queries and improving query response.

The results of an SQE are stored in an internal table in the index (text or theme) for the text column. The SQE definition is stored in a system-wide, internal table owned by CTXSYS. The SQE definitions can be accessed through the views, CTX\_SQES and CTX\_USER\_SQES.

**See Also:** For more information about the SQE result table, see “SQR Table” in Appendix C, “ConText Index Tables and Indexes”.



## Linguistics Requests

The ConText Linguistics are used to analyze the content of English-language documents. The application developer uses the Linguistics output to create different views of the contents of documents.

The Linguistics currently provide two types of output, on a per document basis, for English-language documents stored in an Oracle database:

- list of themes
- document Gist and/or theme summaries

**See Also:** For more information about themes, Gists, and theme summaries, as well as using the Linguistics in applications, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## Text Columns

A text column is any column used to store either text or text references (pointers) in a database table or view. ConText recognizes a column as a text column if one or more policies are defined for the column.

### Supported Datatypes

Text columns can be any of the supported Oracle datatypes; however, text columns are usually one of the following datatypes:

- CHAR
- VARCHAR2
- LONG
- LONG RAW
- BLOB
- CLOB
- BFILE

A table can contain more than one text column; however, each text column requires a separate policy.

**See Also:** For more information about policies and text columns, see “Policies” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

For more information about Oracle datatypes, see *Oracle8 Server Concepts*.

For more information about managing LOBs (BLOB, CLOB, and BFILE), see *Oracle8 Application Developer's Guide* and *PL/SQL User's Guide and Reference*.

### Textkeys

ConText uses textkeys to uniquely identify a document in a text column. The textkey for a text column usually corresponds to the primary key for the table or view in which the column is located; however, the textkey for a column can also reference unique keys (columns) that have been defined for the table.

When a policy is defined for a column, the textkey for the column is specified.

## Composite Textkeys

A textkey for a text column can consist of up to sixteen primary or unique key columns.

During policy definition, the primary/unique key columns are specified, using a comma to separate each column name.

In two-step queries, the columns in a composite textkey are returned in the order in which the columns were specified in the policy.

In an in-memory queries, the columns in a composite textkey are returned in encoded form (e.g. 'p1,p2,p3'). This encoded textkey must be decoded to access the individual columns in the textkey.

---

---

**Note:** There are some limits to composite textkeys that must be considered when setting up your tables and columns, and when creating policies for the columns.

---

---

**See Also:** For more information about encoding and decoding composite textkeys, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Column Name Limitations

There is a 256 character limit, including the comma separators, on the string of column names that can be specified for a composite textkey.

Because the comma separators are included in this limit, the actual limit is 256 minus (no. of columns minus 1), with a maximum of 241 characters (256 - 15), for the combined length of all the column names in the textkey.

This limit is enforced during policy creation.

### Column Length Limitations

There is a 256 character limit on the combined lengths of the columns in a composite textkey. This is due to the way the textkey values for composite textkeys are stored in the index.

For a given row, ConText concatenates all of the values from the columns that constitute the composite textkey into a single value, using commas to separate the values from each column.

As such, the actual limit for the lengths of the textkey columns is 256 minus (no. of columns minus 1), with a maximum of 241 characters (256 - 15), for the combined length of all the columns.

---

---

**Note:** If you allow values that contain commas (e.g. numbers, dates) in your textkey columns, the commas are escaped automatically by ConText during indexing. The escape character is the backslash character.

In addition, if you allow values that contain backslashes (e.g. dates or directory structures in Windows) in your textkey columns, ConText uses the backslash character to escape the backslashes.

As a result, when calculating the limit for the length of columns in a composite textkey, the overall limit of 256 (241) characters must include the backslash characters used to escape commas and backslashes contained in the data.

---

---

## Text Loading

The loading of text into database tables is required for using ConText to perform queries and generate linguistic output. This task can be performed within an application; however, if you have a large document set, you may want to perform loading as a batch process.

**See Also:** For more information about building text loading capabilities into your applications, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Plain Text Loading

For loading strings of plain (ASCII) text into individual rows (documents), you can use the INSERT command in SQL.

**See Also:** For more information about the INSERT command, see *Oracle8 Server SQL Reference*.

### Batch Loading

Either SQL\*Loader or ctxload can be used to perform batch loading of text into a database column.

#### SQL\*Loader

To perform batch loading of plain (ASCII) text into a table, you can use SQL\*Loader, a data loading utility provided by Oracle.

**See Also:** For more information about SQL\*Loader, see *Oracle8 Server Utilities*.

#### ctxload Utility

For batch loading plain or formatted text, you can use the ctxload command-line utility provided by ConText.

The ctxload utility loads text from a load file into a specified database table. The load file can contain multiple documents, but must use a defined structure and syntax. In addition, the load file can contain plain (ASCII) text or it can contain pointers to separate files containing either plain or formatted text.

---

---

**Note:** ctxload is best suited for loading text into columns that use direct data store. If you use external data store to store file pointers in the database, it is possible to use ctxload; however, you should use another loading method, such as SQL\*Loader.

---

---

**See Also:** For an example of loading text using ctxload, see “Using ctxload” in Chapter 9, “Setting Up and Managing Text”.

## Automated Batch Loading

If you set up sources for your columns, you can use ConText servers running with the Loader personality to automate batch loading of text from load files.

If a ConText server is running with the Loader personality, it regularly checks all the sources that have been defined for columns in the database, then scans specified directories for new files. When a new file appears, it calls ctxload to load the contents of the file into the appropriate column.

When loading of the file contents is successful, the server deletes the file to prevent the contents from being loaded again.

**See Also:** For an example of automated text loading, see “Using ConText Servers for Automated Text Loading” in Chapter 9, “Setting Up and Managing Text”.

## User-Defined Translators

If the contents of the file to be loaded are not in the load file format required by ctxload, the file needs to be formatted before loading.

To ensure that the files are in the correct format, a user-defined translator can be specified as one of the preferences in the source for the column.

A user-defined translator is any program that accepts a plain text file as input and generates a load file formatted for ctxload as its output. The user-defined translator could also be used to perform pre-loading cleanup and spell-checking of your text.

After the contents of the load file have been successfully loaded into the column, the load file generated by the translator is deleted along with the original input file to prevent the contents from being loaded again.

**See Also:** For more information about translators for text loading, see “Translator Tiles” in Chapter 8, “Understanding the Con-Text Data Dictionary: Text Loading”.

## Error Handling

If an error occurs while loading, the error is written to the error log, which can be viewed using `CTX_INDEX_ERRORS`. In addition, the original file is not deleted.

# Text Storage

ConText supports three methods of storing text in a column:

- Direct Storage
- External Storage
- Master-Detail Storage

---

**Note:** The tables illustrated in the following sections are *examples* only. The column names and definitions for actual tables used to store text will vary depending on the needs of your application.

---

## Direct Storage

With direct storage, text for documents is stored directly in a database column. The following table description illustrates a table in which text is stored directly in a column:

Table Name	Column Name	Datatype	Description
DIR_TEXT	TEXTKEY	NUMBER	Primary or unique key for table
	TEXTDATE	DATE	Document publication date
	AUTHOR	VARCHAR2(50)	Document author
	NOTES	VARCHAR2(2000)	Text column with direct storage
	TEXT	LONG	Text column with direct storage

The requirements for storing text directly in a column are relatively straightforward. The text is physically stored in a text column and the policy for the text column contains a Data Store preference that utilizes the DIRECT Tile.



## External Storage

With external storage, the text column does not contain the actual text of the document, but rather stores a pointer to the file that contains the text of the document.

---

**Suggestion:** If text is stored as external text in a column, the column should be either a CHAR or VARCHAR2 column. LONG and LONG RAW columns are best suited for documents stored internally in the database.

---

The pointer can be either:

- a file name for accessing text stored in local operating system files (OSFILE Tile)
- a Uniform Resource Locator (URL) for accessing text stored in Web files on the World Wide Web or locally (URL Tile)

The following table description illustrates a table that uses external data storage:

Table Name	Column Name	Datatype	Description
EXT_TEXT	TEXTKEY	NUMBER	Primary or unique key for the table
	TEXTDATE	DATE	Document publication date
	AUTHOR	VARCHAR2(50)	Document author
	NOTES	VARCHAR2(2000)	Text column with direct text storage
	TEXT	VARCHAR2(100)	Text column with names of operating system files that contain the document text

In this example, the only difference between a table used to store text internally and externally is the datatype of the text column. In an external table, the text column would typically be assigned a datatype of VARCHAR2, rather than LONG, because the column contains a pointer to a file rather than the contents of the file (which requires more space to store).

However, there are additional requirements for storing text externally due to the different methods (file names and URLs) of accessing text stored in flat files.

**See Also:** For more information about the requirements for storing text externally, see “External Text” in this chapter.

## Master-Detail Storage

Master-detail storage is for documents stored directly in a text column, similar to direct storage; however, each document consists of one or more rows which are indexed as a single row.

In a master-detail relationship, the master table contains the textkey column and the detail table contains the text column, the line number column, and a foreign key to a primary or unique key column in the master table.

The foreign key and the line number columns comprise the primary key for the detail table, which is used to store the text.

The following table description illustrates two tables with a master-detail relationship:

Table Name	Column Name	Datatype	Description
MASTER	PK	NUMBER	Primary key for table
	AUTHOR	VARCHAR2	Document author
	TITLE	VARCHAR2	Document title
DETAIL	FK	NUMBER	Foreign key to <i>master.pk</i>
	LINENO	NUMBER	Detail information for document
	TEXT	VARCHAR2	Text column

The following query illustrates the relationship between the two tables:

```
select DETAIL.TEXT
from DETAIL
where DETAIL.FK = MASTER.PK
order by DETAIL.LINENO
```

ConText supports two methods of creating policies for text columns in master-detail tables:

- Policies on Columns in Master Table
- Policies on Columns in Detail Table

### Policies on Columns in Master Table

With this method, the MASTER DETAIL NEW Tile is used to create Data Store preferences, which are used in the policy assigned to one of the columns in the master table. The column to which the policy is assigned (i.e. the text column) can be any column in the master table, *except* the column that serves as the textkey column for the policy.

---

---

**Note:** The contents of the text column are not actually indexed. The text column only serves as a place-holder for the policy.

---

---

The detail table name and attributes, including the name of the column that contains the text to be indexed, are specified in the Data Store preference.

Using the tables described above, the textkey for the policy would be *pk* in *master*. The text column for the policy could be either *author* or *title*.

The Data Store preference for the policy would identify *detail* as the detail table, *lineno* as the line number column, and *text* as the column containing the text to be indexed.

**See Also:** For an example of creating a policy on a master table column, see “Creating a Data Store Preference for a Master Table” in Chapter 9, “Setting Up and Managing Text”

**Advantages** This method has the following advantages:

1. DML is handled with one insert to the DML Queue, resulting in a smaller queue and quicker processing
2. Structured data queries in text/theme queries can be applied to the master table

For example:

```
exec ctx_query.contains('MY_POL','Oracle','ctx_temp', struct_query=>'author=' 'SMITH'');
```

**Limitations** This method has the following limitations:

1. The column storing text in the detail table is limited to CHAR, VARCHAR2, and LONG datatypes.
2. Updates to individual rows in the detail table are no longer automatically detected, since the DML trigger is on the master table. Updates to the text in the detail table must be manually reindexed using CTX\_DML.REINDEX or by creating a trigger on the detail table that calls CTX\_DML.REINDEX.

### Policies on Columns in Detail Table

With this method, the policy is created on the detail table, rather than on the master table, and the MASTER DETAIL Tile is used instead of the MASTER DETAIL Tile, to create Data Store preferences.

The textkey column and text column for the detail table, along with the line number column, are specified in the policy. The textkey column and the line number column together uniquely identify rows in the detail table.

Using the tables described above, the textkey for the policy would be *fk* in *detail*. The text column for the policy would be *text*.

**Disadvantages** This method has the following disadvantages:

1. Structured data queries in text/theme queries may be slow. The relevant relational criteria is often stored in a different table, resulting in sub-selects to return structured data.
2. DML may be slow, because the DML trigger is created on the detail table. When a new row is created in the master table and its corresponding rows are created in the detail table, one request is sent to the DML queue for each new detail row, thereby slowing down the queue.
3. The syntax for one-step queries is non-intuitive. Since the policy is created on the detail table, the one-step query is on the detail table, which may result in multiple rows per document returned by a query.

---

---

**Note:** This method is provided primarily to maintain backward compatibility with previous versions of ConText.

If you want to index text stored in master-detail tables, Oracle Corporation suggests that you create policies on the master table.

---

---

## External Text

The requirements for storing text externally are more complicated than storing text directly in a column due to the different methods of accessing text stored in external files. This section provides detailed information about the two different external file methods supported by ConText:

- Text Stored as File Names
- Text Stored as URLs

### Text Stored as File Names

For text stored as file names pointing to external files (OSFILE Tile), the name and location of the file must be stored.

#### File Names

The names of the external text files are stored in the text column.

#### Directory Path Names

The directory path(s) where the external text files are located can be stored in the text column as part of the file name or in the Data Store preference that you create for the OSFILE Tile.

---

---

**Note:** If the preference does not contain the directory path for the files, ConText requires the directory path to be included as part of the file name stored in the text column.

---

---

#### File Access

All the external files referenced in the text column must be accessible from the server machine on which the ConText server is running. This can be accomplished by storing the files locally in the file system for the server machine or by mounting the remote file system to the server machine.

#### File Permissions

File permissions for external files in which text is stored must be set accordingly to allow ConText to access the files. If the file permissions are not set properly for a file and ConText cannot access the file, the file cannot be indexed or retrieved by ConText.

## Text Stored as URLs

For text stored in external Web files, the complete address for each file must be stored as a URL in the text column and the URL Tile utilized in the policy for the column.

---

---

**Note:** Text that contains HTML tags and is stored directly in a text column is considered internal, rather than external, text. As such, the Data Store preference for the text column policy would use the Data Store Tiles which support direct text storage.

In addition, Web files can be any format supported by the World Wide Web, including HTML files, plain (ASCII) text files, and proprietary formats, such as PDF and Word. The filter for the column must be able to recognize and process any of the possible documents formats that may be encountered on the Web.

---

---

A URL consists of the access scheme for the Web file and the address of the file, in the following format:

*access\_scheme://file\_address*

The ConText URL Tile supports three access scheme protocols in URLs:

- Hypertext Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)
- File Protocol

### Hypertext Transfer Protocol (HTTP)

If a URL uses HTTP, the file address contains the host name of the Web server where the file is located and, optionally, the URL path for the file on the Web server.

For example:

`http://my_server.com/welcome.html`

`http://www.oracle.com`

---

---

**Note:** The file address may also (optionally) contain the port on which the Web server is listening.

---

---

In this context, a Web server is any host machine that is running an HTTP daemon, which accepts requests for files and transfers the files to the requestor.

### File Transfer Protocol (FTP)

If a URL uses FTP, the file address contains the host name of the Web server where the file is located and, optionally, the directory path for the file on the Web server.

For example:

```
ftp://my_server.com/code/samples/sample1.tar.Z
```

---

---

**Note:** The file address may also (optionally) contain a username/password for accessing the host machine.

---

---

In this context, a Web server is any host machine that is running an FTP daemon, which accepts requests for files and transfers the files to the requestor.

### File Protocol

If a URL uses the file protocol, the address for the file contains the absolute directory path for the location of the file on the local file system.

For example:

```
file://private/docs/html/intro.html
```

The file referenced by a URL using the file protocol must reside locally on a file system that is accessible to the machine running ConText.

Because the file is accessed through the operating system, the machine on which the file is located does *not* need to be configured as a Web server. However, the same requirements that apply to text stored as file names apply to text stored as URLs which use the file protocol.

If the requirements are not met, ConText returns one or more error messages.

**See Also:** For more information, see “Text Stored as File Names” in this chapter.

For the error messages returned by the URL data store, see *Oracle8 Error Messages*.

### **Intranet Support**

Through HTTP and FTP, the URL Tile can be used to index files in an intranet, as well as files on any publicly-accessible Web servers on the World Wide Web.

Intranets are private networks that use the Internet to link machines in the network, but are protected from public access on the Internet via a gateway proxy server which acts as a firewall.

Outside a firewall, a URL request for a Web file is processed directly by the host machine identified in the URL. Within a firewall, requests are processed by the proxy server, which passes the request to the appropriate host machine and transfers the response back to the requestor.

For security reasons, access to an intranet is generally restricted to machines within the firewall; however, machines in an intranet can access the World Wide Web through the gateway proxy server if they have the appropriate permission and security clearance.

## **Document Access Using HTTP or FTP**

When HTTP or FTP is used in a URL stored in the database, ConText acts as a client, submitting a request to a Web server for the file (document) referenced by the URL. If the request is successful, the Web server returns the file to ConText where it can be indexed for querying or highlighted for viewing.

### **Proxy Servers**

If the document to be accessed is located on the World Wide Web outside a firewall and the machine on which ConText is installed is inside the firewall, a host machine that serves as the proxy (gateway) for the firewall must be specified as an attribute for the URL Tile.

A single machine can be specified as the proxy for handling HTTP and FTP requests or two separate machines can be specified, one for each protocol. If network traffic is expected to be heavy or a large number of FTP requests are expected, separate proxies should be specified for HTTP and FTP, since FTP is generally used for accessing large, binary files which may affect performance on the proxy server.

In addition to specifying proxy servers, a sub-string of host or domain names, which identify all or most of the machines internal to the firewall, should be specified. Access to these machines does not require going through the proxy server, which helps reduce the request load that your proxy server(s) have to process.



**Multi-threading**

In a single-threaded environment, a request for a URL blocks all other requests until a response to the request is returned. Because a response may not be returned for a long time, a single-threaded environment in any text system using HTTP or FTP to access files could create a bottleneck.

To prevent this type of bottleneck, the URL Tile supports multi-threading. With multi-threading, while one thread is blocked, waiting to communicate with a Web server, another thread can retrieve a document from another Web server.

**Redirection**

The response to a request to retrieve a URL may be a new (redirected) document to retrieve. The URL Tile supports this type of redirection by automatically processing the redirection to retrieve the new document. However, to avoid infinite loops, the URL Tile limits the number of redirections that it attempts to process to three (3).

**Timeouts**

The time necessary to retrieve a URL using HTTP may vary widely, depending on where the Web server is geographically located. The Web server may even be temporarily unreachable.

To allow control over the length of time an application waits for a response to an HTTP request for a URL, the URL data store supports specifying a maximum timeout.

### Exception Handling

When using URLs as your data store, a number of exceptions can occur when a file is accessed. These exceptions are written as errors to the CTX\_INDEX\_ERRORS view.

The URL data store returns error messages for the following exceptions:

- the document referenced in the URL has been permanently moved or cannot be found
- access to the document referenced in the URL requires authentication which the user does not have or requires payment which the user must provide
- access to the document referenced in the URL is denied by the Web server
- the Web server referenced in the URL does not comply with HTTP standards
- the specified URL is incorrectly formatted
- connection to the Web server is denied (this may occur when the incorrect port is referenced in the URL or the Web server is outside the firewall of an intranet)
- the wait for a response to a request to retrieve a URL from a Web server exceeds the maximum timeout specified for the URL preference in the text column policy
- the maximum number of supported redirections were encountered in attempting to retrieve the document referenced in the URL
- the length of the URL exceeds the maximum specified for the URL preference in the text column policy
- the size of the document referenced in the URL exceeds the maximum specified for the URL preference in the text column policy

**See Also:** For the error messages returned by the URL data store, see *Oracle8 Error Messages*.

## Text Filtering

ConText supports both plain text and formatted text (i.e. Microsoft Word, WordPerfect). In addition, ConText supports text that contains hypertext markup language (HTML) tags.

Regardless of the format, ConText requires text to be filtered for the purposes of indexing the text or processing the text through the Linguistics, as well as highlighting the text for viewing.

This section discusses the following topics relevant to text filtering:

- Internal Filters
- External Filters
- Filters for Single-Format Columns
- Filters for Single-Format Columns
- Filters for Mixed-Format Columns
- Supported External Filter Formats for Mixed-Format Columns
- Supplied External Filters

**See Also:** For more information about indexing, see “ConText Indexes” in this chapter.

For more information about the Linguistics and text highlighting, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## Internal Filters

ConText provides internal filters for:

- Plain Text Filtering
- HTML Filtering (plaint text containing HTML tags)
- Formatted Text Filtering

### Plain Text Filtering

Plain text requires little or no filtering because the text is already in the format that ConText requires for identifying tokens.

### HTML Filtering

ConText provides an internal filter that supports English and Japanese text with HTML tags for versions 1, 2, and 3.

---

---

**Note:** For non-English and non-Japanese documents that contain HTML tags, an external filter must be used.

---

---

The HTML filter processes all text that is delimited by the standard HTML tag characters (angle brackets).

All HTML tags are either ignored or converted to their representative characters in the ASCII character set. This ensures that only the text of the document is processed during indexing or by the Linguistics.

### Formatted Text Filtering

ConText provides internal filters for filtering English and Western European text in a number of proprietary word processing formats.

---

---

**Note:** For Japanese, Korean, and Chinese formatted text, external filters must be used.

---

---

The filters extract plain, ASCII text from a document, then pass the text to ConText, where the text is indexed or processed through the Linguistics. The following document formats are supported by the internal filters:

Format	Version
AmiPro for Windows	1, 2, 3
Lotus 123 for DOS	4, 5
Lotus 123 for Windows	2, 3, 4, 5
Microsoft Word for DOS	5.0, 5.5
Microsoft Word for Macintosh	3, 4, 5.x
Microsoft Word for Windows	2, 6.x, 7.0
WordPerfect for DOS	5.0, 5.1, 6.0
WordPerfect for Windows	5.x, 6.x
Xerox XIF for UNIX	5, 6

---

**Note:** Only the following formats support WYSIWYG viewing in the ConText viewer:

- Microsoft Word for Windows 2 and 6.x
- Word Perfect for DOS 5.0, 5.1, 6.0
- Word Perfect for Windows 5.x, 6.x

For more information about the ConText viewer, see *Oracle8 Con-Text Cartridge Workbench User's Guide*.

---

For those formats not supported by the internal filters, user can define/create their own external filters.

## External Filters

ConText provides a framework for users to plug-in third-party filters to extract pertinent text information from documents. These external filters can be used for a number of purposes, including:

- indexing text stored in a format, such as PDF, for which an internal filter does not exist
- removing unnecessary text or markup in a document prior to indexing or processing through the ConText Linguistics

For example, the Linguistics rely on text that is grouped into logical paragraphs. If the text stored in the database does not contain clearly-identified paragraphs, the quality of the output generated by the Linguistics may be poor.

An external filter that outlines the paragraph boundaries according to ConText standards could be created to ensure that the Linguistics are provided with an ordered, logical text feed.

---

---

**Note:** External filters do not support WYSIWYG viewing in the ConText Workbench Viewer Control (Windows 32-bit).

For more information about the Viewer Control, see *Oracle8 ConText Cartridge Workbench User's Guide*.

---

---

### External Filter Requirements

An external filter can be any executable (e.g. shell script, C program, perl script) that processes an input file and produces a plain text output file. The text in the output file then can be indexed or processed through the Linguistics.

If the document is in a proprietary format, the executable must recognize the format tags for the document and be able to convert the formatted text into plain (ASCII) text.

In addition, the executable must be able to run from the operating system command-line and accept two arguments:

- name of an input file, which stores the document to be filtered
- name of an output file, which stores the filtered, ASCII text of the document

The external filter does not need to provide the values for these arguments; Context provides the values as part of its external filter processing.

---

---

**Note:** The name of the executable cannot be larger than 64 bytes. In addition, the name cannot contain blank spaces or any of the following illegal characters:

! @ # \$ % ^ & \* ( ) ~ \ Q ' , ^ : " ; ,

---

---

## Performance Issues

Performance is dependent on the external filter; ConText cannot begin processing a document until the entire document has been filtered. The external filter that performs the filtering should be tuned/optimized accordingly.

## Using External Filters

The process model for using external filters is:

1. Create a filter in the form of a command-line executable.
2. Store the executable on the server machine where ConText is installed.

---

---

**Note:** The filter executable must be located in the appropriate directory for your environment.

For example, in a UNIX-based environment, the filter executables must be stored in `$ORACLE_HOME/ctx/bin`.

In a Windows NT environment, the executables must be stored in `\BIN` in the Oracle home directory.

For more information about the required location for the external filter executables, see the Oracle8 installation documentation specific to your operating system.

---

---

3. Create a Filter preference that calls the filter executable.

The Tile you use to create the preference depends on whether you use the column to store documents in a single format or mixed formats.

4. Create a policy that includes the Filter preference for the external filter.

**See Also:** For examples of creating Filter preferences for external filters, see “Creating Filter Preferences” in Chapter 9, “Setting Up and Managing Text”.

## Filters for Single-Format Columns

For columns that store documents in only one format, a single filter is specified in the Filter preference for the column policy. The filtering method for the column is determined by whether the format is supported by the internal or external filters:

- if the format is supported by the internal filters, the appropriate internal filter can be used
- if the format is not one of the supported internal filter formats, an external filter must be used

**See Also:** For examples of creating Filter preferences for single-format columns, see “Creating Filter Preferences” in Chapter 9, “Setting Up and Managing Text”.

## Filters for Mixed-Format Columns

For columns that store documents in mixed formats, the filtering method is determined by whether the formats are supported by the internal filters, external filters, or both:

- if the formats are all supported by the internal filters, the internal Autorecognize filter can be used in the Filter preference
- if none of the formats are supported by the internal filter formats, an external filter must be specified for *each* of the formats in the column
- if some, but not all of the formats are supported by the internal filter formats, an external filter must be specified for *each* of the unsupported formats

---

---

**Note:** In columns that use external filters, only those external filter formats supported by ConText for mixed-format columns can be used.

For a complete list, see “Supported External Filter Formats for Mixed-Format Columns” in this chapter.

---

---

**See Also:** For examples of creating Filter preferences for mixed-format columns, see “Creating Filter Preferences” in Chapter 9, “Setting Up and Managing Text”.



### **Autorecognize Filter (Internal)**

Autorecognize is an internal filter that automatically recognizes the document formats of all the supported internal filters, as well as plain text (ASCII) and HTML formats, and extracts the text from the document using the appropriate filters.

---

**Note:** Microsoft Word for Windows 7.0 documents are not recognized by Autorecognize. As a result, ConText does not support storing Microsoft Word for Windows 7.0 documents in mixed-format columns.

---

**See Also:** For a complete list of supported internal filters, see “Internal Filters” in this chapter.

### **External-Only Filters**

For mixed-format columns that use only external filters, each filter executable for the formats in the column must be explicitly named in the Filter preference for the column policy. In addition, a format ID must be specified for each filter executable.

The format ID is used by ConText to identify document formats in text columns that store multiple formats.

### **Internal and External Filters**

If the column uses both internal and external filters, each external filter executable must be explicitly named in the Filter preference for the column policy. In addition, a format ID must be specified for each filter executable.

The internal filters do *not* have to be specified.

During filtering, ConText recognizes whether a format uses the internal or external filters and calls the appropriate filter.

---

---

**Note:** If required, internal filters can be overridden in a Filter preference by explicitly calling an external filter for the format. This can be useful if you have an external filter that provides additional filtering not provided by the internal filters.

For example, you may have MS Word documents that you want spellchecked before indexing. You could create an external MS Word filter that performs the spellchecking and specify the external filter in the Filter preference for the column policy.

---

---

## Supported External Filter Formats for Mixed-Format Columns

The following table lists all of the document formats that ConText supports for columns that use external filters and store documents in more than one format.

For each format, the format ID is also listed. This is the value that must be specified when creating a Filter preference using the BLASTER FILTER Tile with the *executable* attribute.

---

---

**Note:** This list does *not* represent the complete list of formats that ConText is able to process. The external filter framework enables ConText to process *any* document format, provided the documents are stored in a single format and an external filter exists which can filter the format to plain text.

It also does *not* represent the list of formats for which Oracle provides external filters.

For the complete list of external filters supplied by Oracle, see “Supplied External Filters” in this chapter.

---

---

**See Also:** For an example of using format IDs in Filter preferences, see “Creating Filter Preferences” in Chapter 9, “Setting Up and Managing Text”.

Document Format	ID	t					
AmiPro 1.x - 3.1	19	Enable 1.1, 2.0, 2.15	11	Lotus 123 4.x; Lotus 123 3.0; Lotus 123 1A, 2.0, 2.1	20	MS RTF; MS RTF (ANSI Char Set)	17
AmiPro Graphics SDW Samna Draw	62	Encapsulated Post-Script Preview; Encapsulated PostScript Bitmap	66	Lotus Freelance	85	MS Word for DOS 6.0; MS Word for DOS 5.0, 5.5; MS Word for DOS 4.0; MS Word for DOS 3.0, 3.1	8
ASCII	90	First Choice 3.0 Data Base	13	Lotus Manuscript 2.0, 2.1	26	MS Word for Mac 5.0, 5.1; MS Word for Mac 4.0; MS Word for Mac 3.0	28
AT&T Crystal Writer	46	FrameMaker (MIF) 3.0; FrameMaker (MIF) 3.0 Win	42	Lotus PIC	67	MS Word for Windows 2.0; MS Word for Windows 1.x	18
AutoCAD (DXF, DXB)	53	Framework III, 1.0, 1.1	22	Macintosh Paint	88	MS Word for Windows 6.0; MS Word for Mac 6.0	68
CEOWrite 3.0	78	FullWrite Professional 1.0x	31	Microsoft Windows Paint 2.x	70	MS Works for Windows 3.0	69
Computer Graphics Metafile (CGM)	79	GIF (Graphical Interchange Format)	51	Macintosh QuickDraw (PICT)	64	MS Write for Windows 3.x	7
CorelDraw 2.x and 3.x	59	Harvard Graphics	87	MacWrite 4.5 - 5.0	29	MultiMate 4; MultiMate Advantage II; MultiMate Advantage; MultiMate 3.3	6
CTOS DEF	75	HP Graphics Language (HPGL)	83	MacWrite II 1.0 - 1.1	30	Navy DIF (GSA)	35
DBase IV 1.0; DBase III, III +	37	HTML Level 1, 2, 3	91	Mass 11, Version 8.0 - 8.33	36	OfficePower 7; OfficePower 6	44
DCA/FFT - Final Form Text	27	IBM Writing Assistant 1.0	16	MastSoft Graphics (MSG)	49	OfficeWriter 6.0 - 6.2; OfficeWriter 5.0; OfficeWriter 4.0	9
DCA/RFT - Revisable Form Text	0	IGES	52	Micrografx Designer (DRW)	60	OS/2 Bitmap; Windows Bitmap (BMP); Windows RLE	63
Digital DX	15	Interleaf 5.2; Interleaf 5.2 - 6.0	32	MS Access 2.0	39	Paradox 3.5, 4.0	38
Digital WPS-PLUS	47	JPEG (Joint Photographic Experts Group)	58	MS Excel 5.0 - 6.0; MS Excel 4.0; MS Excel 3.0; MS Excel 2.1	21	PC Paintbrush (PCX)	71
EBCDIC	89	Legacy 1.x, 2.0	41	MS Powerpoint for Windows 2, 3, 4	84	PDF (Adobe Acrobat)	57

Document Format	ID						
PeachText 5000 2.1.2	82	TIFF (Tagged Image File Format)	50	WiziDraw	86	WordPerfect 4.2; WordPerfect 4.1	80
PFS:First Choice 3.0; PFS:First Choice 2.0; PFS:First Choice 1.0; PFS:WRITE Ver C; Professional Write 2.0 - 2.2; Professional Write 1.0	12	Uniplex V7 - V8	77	WiziWord	56	WordPerfect Mac 1.0	81
Quattro Pro DOS; Quattro Pro Windows	45	Vokswriter 3, 4	74	Word For Word Intermediate Communications format (COM)	34	WordPerfect Mac 3.0; WordPerfect Mac 2.1; WordPerfect Mac 2.0	33
Q&A 4.0; Q&A Write 1.x, Q&A 3.0	10	Wang PC, Version 3	24	WordPerfect for Windows 6.1; WordPerfect for Windows 6.0; WordPerfect 6.0	1	WordStar 5.0, 5.5, 6.0, 7.0	40
Rapid File 1.0	23	Wang WITA	55	WordPerfect 5.1 (Mail Merge)	2	WordStar 2000, Rel 3.0	14
RGIP	61	Windows Clipboard	72	WordPerfect for Windows 5.x; WordPerfect 5.1; WordPerfect 5.0	3	WriteNow 3.0	54
Samna Word IV & IV + 1.0, 2.0	25	Windows ICON	73	WordPerfect Graphics 1 (WPG)	4	Xerox - XIF 5.0, 6.0	43
Sun Raster Graphics	65	Windows Metafile (WMF)	48	WordPerfect Graphics 2 (WPG)	5	XYWrite IV; XyWrite III Plus	76

## Supplied External Filters

ConText provides a number of external filters, licensed from MasterSoft (Inso Corporation), on a number of platforms.

These filters can be used for filtering documents in many of the popular desktop publishing and word processor formats; however, the executables for the filters do not provide ConText with the required arguments, so ConText also provides scripts which act as wrappers for the executables:

Document Format	Version	Format ID	Wrapper Name	DOS (Windows NT) Executable	Sun Solaris 2.x Executable	Other UNIX Platforms Executable
AmiPro for Windows	1, 2, 3	19	amipro	w033b16d.exe	w4w33b	w4w33b
Lotus Freelance for Windows	2	85	lotusfre	w114b16d.exe	w4w114b	w4w114b
Lotus 123	2, 3, 4	20	lotus123	w020b16d.exe	w4w20b	w4w20b
Lotus 123	5	N/A	lotus123	w020b16d.exe	w4w20b	w4w20b
MS Excel	5	21	msexcel	w021b16d.exe	w4w21b	w4w21b
MS Excel	7	N/A	msexcel	w021b16d.exe	w4w21b	w4w21b
MS PowerPoint for Windows	2, 3, 4	84	power234	w109b16d.exe	w4w109b	w4w109b
MS PowerPoint for Windows	7	N/A	power7	w116b16d.exe	w4w116b	w4w116b
MS Word for DOS	5.0, 5.5	8	worddos	w005b16d.exe	w4w05b	w4w05b
MS Word for Macintosh	3, 4, 5	28	wordmac	w054b16d.exe	w4w54b	w4w54b
MS Word for Windows	2	18	wordwn2	w044b16d.exe	w4w44b	w4w44b
MS Word for Windows	6	68	wordwn67	w049b16d.exe	w4w49b	w4w49b
MS Word for Windows	7	N/A	wordwn67	w049b16d.exe	w4w49b	w4w49b
PDF/Adobe Acrobat	N/A	57	acropdf	acront.exe	acrosol	w4w107b (BETA)
WordPerfect for DOS; WordPerfect for Windows	5.0, 5.1; 5.x	3	wp5	w007b16d.exe	w4w07b	w4w07b
WordPerfect for DOS; WordPerfect for Windows	6.0; 6.x	1	wp67	w048b16d.exe	w4w48b	w4w48b
WordPerfect for Windows	7.0	N/A	wp67	w048b16d.exe	w4w48b	w4w48b
Xerox XIF	5, 6	43	xeroxxif	w103b16d.exe	w4w103b	w4w103b

### Supplied External Filters Installation

The supplied external filter executables and their wrappers are installed automatically during installation of ConText. The location and format of the executable and wrapper files are operating system dependent.

---

**Note:** If you have upgraded from a release prior to release 2.3 of ConText, you may have existing external filters supplied by ConText. These external filters are no longer up-to-date and should be replaced by the external filters provided in this release.

You may also need to change your wrappers accordingly or use the wrappers provided by ConText in this release.

---

**See Also:** For more information about the location of the supplied external filters, see the Oracle8 installation documentation specific to your operating system.

### Supplied External Filter Setup

The supplied external filters do not require any setup, aside from creating preferences that call the wrappers for the filters; however, if you have upgraded from a previous release and already have wrappers for the external filters provided in the previous release, as well as preferences that call the wrappers, if you want to use the new wrappers as provided, you must drop your indexes, policies, and preferences, then create new preferences, policies, and indexes.

To avoid this situation, you can choose one of the following actions:

- modify the names of the new wrappers to match the names of the existing wrappers
- modify the names of the new external filters to match the names of the previous external filters
- modify your existing wrappers to call the new external filter executables

---

**Note:** If you modify your existing wrappers, in general, the only information that needs to change in the wrappers is the name of the filter executable.

If you are using the BETA PDF filter provided for Windows NT and Sun Solaris in the previous release and wish to use the new production PDF filter for these two platforms, you should drop any ConText indexes that you created with the BETA PDF filter, as well as the policies and preferences that called the filter. Then, create new preferences/policies that use the new filter and create new ConText indexes using the policies.

---

## Supplied External Filter Usage

The supplied external filters have the following three usage issues:

1. The wrapper name (e.g. 'amipro'), *not* the executable (e.g. 'w033b16d.exe' or 'w4w33b'), must be specified in the Filter preferences that you create for the supplied external filters.

Because the wrappers, and not the executables, are called in the Filter preferences that you create for the supplied external filters, you generally do not need to know the name of the filter executables; however, if you find it necessary to modify the wrappers, it may be useful to know the names of the filter executables.

2. If a format does not have a format ID (e.g. MS Word for Windows, version 7), the external filter cannot be used in text columns that store multiple formats. It can *only* be used in text columns that store a single format.
3. Wrapper names are operating system dependent and may be different than the names listed. In particular, the wrapper names may have suffixes (e.g. '.sh' or '.bat') that your operating system uses to identify scripts.

If your operating system requires specifying the complete name of a script in order to run the script, you must include any suffixes in the Filter preferences that you create using the supplied external filters.

**See Also:** For examples of using the supplied external filters, see “Creating Filter Preferences” in Chapter 9, “Setting Up and Managing Text”.

## Limitations

The PDF filter provided for the UNIX-based platforms has a status of BETA. It does not support filtering multi-column documents or documents over approximately 1 Megabyte in size.



## ConText Indexes

A ConText index is the construct that allows ConText servers to process queries and return information based on the content or themes of the text stored in a text column of an Oracle database. A ConText index is an inverted index consisting of all the tokens (words or themes) that occur in a text column and the documents (i.e. rows) in which the tokens are found.

This information is stored in database tables that are associated with the text column through a policy. A ConText index is created by calling `CTX_DDL.CREATE_INDEX` for the policy.

When an query is issued against a text column, rather than scan the actual text to find documents that satisfy the search criteria of the query, ConText searches the ConText index tables for the column to determine whether a document should be returned in the results of the query.

ConText supports two types of indexes, text and theme. This section discusses the following concepts relevant to both text and theme indexes:

- ConText Index Tables
- Stages of ConText Indexing
- Text Indexes
- Index Fragmentation
- Memory Allocation
- Parallel Indexing
- Index Updates
- Index Optimization
- Index Log

**See Also:** For more information about policies, see “Policies” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

For more information about the different types of indexes created by ConText, see “Text Indexes” and “Theme Indexes” in this chapter.

## ConText Index Tables

The ConText index for a text column consists of the following internal tables:

- DR\_#####I1Tn (token table)
- DR\_#####KTB (textkey mapping table)
- DR\_#####LST (control table)
- DR\_#####NLT (DOCID resolution table)
- DR\_#####I1W (Soundex wordlist table -- created only if Soundex is enabled)
- DR\_#####SQL (stored query expression result table)

The ##### string is an identifier (from 1000-99999) which indicates the policy of the text column for which the ConText index is created.

In addition, ConText automatically creates one or more Oracle indexes for each ConText index table.

The tablespaces, storage clauses, and other parameters used to create the ConText index tables and Oracle indexes are specified by the attributes set for the Engine preference (GENERIC ENGINE Tile) in the policy for the text column.

**See Also:** For a description of the ConText index tables, see Appendix C, “ConText Index Tables and Indexes”.

For more information about stored query expressions (SQEs), see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Creating Empty ConText Indexes

If you want to create the ConText index tables without populating the tables, ConText provides a parameter, *pop\_index*, for CTX\_DDL.CREATE\_INDEX, which specifies whether the ConText index tables are populated during indexing.

## Stages of ConText Indexing

ConText indexing takes place in three stages:

- Index Initialization
- Index Population
- Index Termination

### Index Initialization

During index initialization, the tables used to store the ConText index are created.

**See Also:** For more information about the tables used to store the ConText index, see “ConText Index Tables” in this chapter.

### Index Population

During index population, the ConText index entries for the documents in the text column are created in memory, then transferred to the index tables.

If the memory buffer fills up before all of the documents in the column have been processed, ConText writes the index entries from the buffer to the index tables and retrieves the next document from the text column to continue ConText indexing.

The amount of memory allocated for ConText indexing for a text column determines the size of the memory buffer and, consequently, how often the index entries are written to the index tables.

**See Also:** For more information about the effects of frequent writes to the index tables, see “Index Fragmentation” and “Memory Allocation” in this chapter.

### Index Termination

During index termination, the Oracle indexes are created for the ConText index tables. Each ConText index table has one or more Oracle indexes that are created automatically by ConText.

---

---

**Note:** The termination stage only starts when the population stage has completed for all of the documents in the text column.

---

---

## Columns with Multiple Indexes

A column can have more than one index by simply creating more than one policy for the column and creating a ConText index for each policy. This is useful if you want to specify different indexing options for the same column. In particular, this is useful if you want to create a text and theme index on a column.

When two indexes exist for the same column, one-step queries (theme or text) require the policy name, as well as the column name, to be specified for the CONTAINS function in the query. In this way, the correct index is accessed for the query.

This requirement is not enforced for two-step and in-memory queries, because they use policy name, rather than column name, to identify the column to be queried.

**See Also:** For more information about one-step queries and the CONTAINS function, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## Index Fragmentation

As ConText builds an index entry for each token (word or theme) in the documents in a column, it caches the index entries in memory. When the memory buffer is full, the index entries are written to the ConText index tables as individual rows.

If all the documents (rows) in a text column have not been indexed when the index entries are written to the index tables, the index entry for a token may not include all of the documents in the column. If the same token is encountered again as ConText indexing continues, a new index entry for the token is stored in memory and written to the index table when the buffer is full.

As a result, a token may have multiple rows in the index table, with each row representing an index fragment. The aggregate of all the rows for a word/theme represents the complete index entry for the word/theme.

**See Also:** For more information about resolving index fragmentation, see “Index Optimization” in this chapter.

## Memory Allocation

A machine performing ConText indexing should have enough memory allocated for indexing to prevent excessive index fragmentation. The amount of memory allocated depends on the capacity of the host machine doing the indexing and the amount of text being indexed.

If a large amount of text is being indexed, the index can be very large, resulting in more frequent inserts of the index text strings to the tables. By allocating more memory, fewer inserts of index strings to the tables are required, resulting in faster indexing and fewer index fragments.

**See Also:** For an example of allocating memory for ConText indexing, see “Creating an Engine Preference” in Chapter 9, “Setting Up and Managing Text”.

## Parallel Indexing

Parallel indexing is the process of dividing ConText indexing between two or more ConText servers. Dividing indexing between servers can help reduce the time it takes to index large amounts of text.

To perform indexing in parallel, you must start two or more ConText servers (each with the DDL personality) and you must correctly allocate indexing memory.

The amount of allocated index memory should not exceed the total memory available on the host machine(s) divided by the number of ConText servers performing the parallel indexing.

For example, you allocate 10 Mb of memory in the policy for the text column for which you want to create a ConText index. If you want to use two servers to perform parallel indexing on your machine, you should have at *least* 20 Mb of memory available during indexing.

---

---

**Note:** When using multiple ConText servers to perform parallel indexing, the servers can run on different host machines if the machines are able to connect via SQL\*Net to the database where the index is stored.

---

---

## Index Updates

When an existing document in a text column is deleted or modified such that the ConText index is no longer up-to-date, the index must be updated.

However, updating the index for modified/deleted documents affects every row that contains references to the document in the index. Because this can take considerable time, ConText utilizes a deferred delete mechanism for updating the index for modified/deleted documents.

In a deferred delete, the document references in the ConText index token table (DR\_nnnnn\_IITn) for the modified/deleted document are not actually removed. Instead, the status of the document is recorded in the ConText index control table (DR\_nnnnn\_LST), so that the textkey for the document is not returned in subsequent text queries that would normally return the document.

Actual deletion of the document references from the token table (IITn) takes place only during optimization of a index.

## Index Optimization

Optimization performs two functions for an index:

- Compaction of Index Fragments
- Removal of Document References (also known as actual deletion or garbage collection)

Compaction of index fragments results in fewer rows in the ConText index tables, which results in faster and more efficient queries. It also allows for more efficient use of tablespace.

Garbage collection updates the index strings to accurately reflect the status of deleted and modified documents.

### Compaction of Index Fragments

Compaction combines the index fragments for a token into longer, more complete strings, up to a maximum of 64 Kb for any individual string.

ConText provides two methods of index compaction:

- in-place compaction
- two-table compaction (default)

In-place compaction uses available memory to compact index fragments, then writes the compacted strings back into the original (existing) token table in the ConText index.

Two-table compaction creates a second token table into which the compacted index fragments are written. When compaction is complete, the original token table is deleted.

Two-table compaction is faster than in-place compaction; however, it requires enough tablespace to be available during compaction to accommodate the creation and population of the second token table.

### Removal of Document References

ConText provides optimization methods which can be used to perform the actual deletion of all references to modified/deleted documents in an index.

During an actual delete, the index references for all modified/deleted documents are removed from the ConText index token table (DR\_nnnnn\_I1Tn), leaving only references to existing, unchanged documents. In addition, in an actual delete, the ConText index control table (DR\_nnnnn\_LST) is cleared of the information which records the status of documents.

Similar to compaction, ConText supports in-place or two-table actual deletion.

### When to Optimize

Index optimization should be performed regularly, as the indexing process can create many rows in the database depending on the amount of memory allocated for indexing and the amount of text being indexed.

In general, optimize an index after:

- large amounts of text are indexed
- parallel indexing has been utilized
- large numbers of documents in a table have been modified or deleted

## Index Log

The ConText index log records all the indexing operations performed on a policy for a text column. Each time an index is created, optimized, or deleted for a text column, an entry is created in the index log.

### Log Details

Each entry in the log provides detailed information about the specified indexing operation, including:

- the policy for the text column on which the indexing operation was performed
- the indexing operation that was performed (creation, optimization, deletion)
- if the indexing operation was performed in parallel, the ID of the server that processed the operation
- whether the operation failed and, if it did, the stage at which it failed
- the number of documents selected for processing and the number of documents actually processed during the indexing operation

- the textkeys of the first and last documents processed

### **Accessing the Log**

The index log is stored in an internal table and can be viewed using the CTX\_INDEX\_LOG or CTX\_USER\_INDEX\_LOG views. The index log can also be viewed in the GUI administration tools (System Administration or Configuration Manager).



## Text Indexes

A text index is generated by one of the text lexers provided by ConText and consists of:

- a list of every unique token (word) in the collection of documents in a text column
- for each word, a string that identifies each document in which the word occurs and the location of each occurrence within each document

There is a one-to-one relationship between a text index and the text indexing policy for which it was created.

**See Also:** For more information about text indexing policies, see “Text Indexing Policies” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

## Text Lexers

The text lexer identifies tokens for creating text indexes. During text indexing, each document in the text column is retrieved and filtered by ConText. Then, the lexer identifies the tokens and extracts them from the filtered text and stores the tokens in memory, along with the document ID and locations for each word, until all of the documents in the column have been processed or the memory buffer is full.

The index entries, consisting of each token and its location string, are then written as rows to the token table for the ConText index and the buffer is flushed.

ConText provides a number of Lexer Tiles that can be used to create text indexes. For non-pictorial languages, such as English and the other Western European languages, ConText provides a single Tile named BASIC LEXER.

For pictorial languages, ConText provides a separate Tile for each of the languages supported by ConText (Japanese, Chinese, and Korean).

**See Also:** For more information about the text indexing lexers, see “Lexer Tiles” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

## What's in a Text Index?

Text index entries consist of each unique token in the text column and a location string for each token. The text index may be case-sensitive or case-insensitive and contain stop words.

### Tokens in Text Indexes

A token is the smallest unit of text that can be indexed.

In non-pictorial languages, tokens are generally identified as alphanumeric characters surrounded by white space and/or punctuation marks. As a result, tokens can be single words, strings of numbers, and even single characters.

In pictorial languages, tokens may consist of single characters or combinations of characters, which is why separate lexers are required for each pictorial language. The lexers search for character patterns to determine token boundaries.

**See Also:** For more information about token recognition, see “Lexer Tiles” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

### Token Location Information

The location information for a token is bit string that contains the location (offsets in ASCII) of each occurrence of the token in each document in the column. The location information also contains any stop words that precede and follow the token.

### Case-sensitivity

For non-pictorial languages, the BASIC LEXER Tile, by default, creates case-insensitive text indexes. In a case-insensitive index, tokens are converted to all uppercase in the index entries.

The Tile also provides an attribute, *mixed\_case*, for enabling case-sensitive text indexes. In a case-sensitive index, entries are created using the tokens exactly as they appear in the text, including those tokens that appear at the beginning of sentences.

For example, in a case-insensitive text index, the tokens *oracle* and *Oracle* are recorded as a single entry, *ORACLE*. In a case-sensitive text index, two entries, *oracle* and *Oracle*, are created.

As a result, case-sensitive indexes may be much larger than case-insensitive indexes and may have some effect on text queries; however, case-sensitive indexes allow for greater precision in text queries.

---

**Note:** The case-sensitivity of a text index affects the text queries performed against the index. If the text index is case-sensitive, text queries are also case-sensitive.

---

**See Also:** For more information about case-sensitivity in text queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Stop Words

A stop word is any combination of alphanumeric characters (generally a word or single character) for which ConText does not create an entry in the index. Stop words are specified in the Stoplist preference for a text indexing policy.

While stop words do not have entries in the text index, stop word information for tokens is stored as numbers in the token bit strings. The number corresponds to the sequence defined for the stop word. The token bit string stores up to eight of the contiguous stop words immediately preceding and following the token. Because the stop words are stored in the text index, stop words can be included phrases in text queries.

---

**Note:** Stoplists for case-sensitive indexes are automatically case-sensitive, meaning stop words in the text are only indexed as stop words if they exactly match the case of the stop words in the stoplist.

As a result, when creating a Stoplist preference for a column on which you want create a case-sensitive text index, you should specify a stoplist entry for each variation (i.e. lowercase, initial uppercase, uppercase) that may occur for a stop word.

---

**See Also:** For an example of creating a Stoplist preference, see “Creating a Stoplist Preference” in Chapter 9, “Setting Up and Managing Text”.

For more information about stoplists, see “Stoplist Tiles” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

For more information about stop words in text queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## DDL and DML

Text indexes are processed using ConText servers with the DDL and DML personalities. All requests for index creation and optimization are processed by any currently available DDL servers.

Text index updates are processed by the DML or DDL servers that are running at the time, depending on the DML index update method (immediate or batch) you are using.

**See Also:** For more information about DDL and DML operations, see “DDL” and “DML” in this chapter.

For more information about ConText server personalities, see “Personality Masks” in Chapter 2, “Administration Concepts”.

## Theme Indexes

Theme indexes are functionally identical to text indexes and are created in the same manner:

- a policy is created for a column
- a DDL request for index creation is submitted for the column
- once the theme index has been generated, the column is enabled for all three query methods

The key to generating a theme index is the lexer that you specify for the column policy. Instead of specifying the basic (default) lexer, the theme lexer is specified.

---

---

**Note:** Theme indexing is *only* supported for English-language text.

---

---

**See Also:** For more information about text indexes, see “Text Indexes” in this chapter.

For more information about theme queries and query methods, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## Theme Lexer

For theme indexing, ConText provides a Tile, THEME LEXER Tile, that bypasses the standard text parsing routines and, instead, accesses the linguistic core in ConText to generate themes for documents.

The theme lexer analyzes text at the sentence, paragraph, and document level to create a context in which the document can be understood. It uses a mixture of statistical methods and heuristics to determine the main topics that are developed throughout the course of the document.

It also uses the ConText Knowledge Catalog, a collection of over 200,000 words and phrases, organized into a conceptual hierarchy with over 2,000 categories, to generate its theme information.

**See Also:** For more information about the ConText Knowledge Catalog, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## What's in a Theme Index?

A theme index contains a list of all the tokens (themes) for the documents in a column and the documents in which each theme is found. Each document can have up to sixteen themes.

---

---

**Note:** Offset and frequency information are not relevant in a theme index, so this type of information is not stored.

---

---

### Tokens in Theme Indexes

Unlike the single tokens that constitute the entries in a text index, the tokens in a theme index often consist of phrases.

In addition, these phrases may be common terms or they may be the names of companies, products, and fields of study as defined in the Knowledge Catalog.

For example, a document about Oracle contains the phrase *Oracle Corp.* In a (case-sensitive) text index for the document, this phrase would have two entries, *ORACLE* and *CORP*, all in uppercase. In a theme index for the document, the entry would be *Oracle Corporation*, which is the canonical form of *Oracle Corp.*, as stored in the Knowledge Catalog.

### Theme Weights

Each document theme has a weight associated with it. The theme weight measures the strength of the theme relative to the other themes in the document. Theme weights are stored as part of the theme signature for a document and are used by ConText to calculate scores for ranking the results of theme queries.

### Case-sensitivity

Theme indexes are always case-sensitive. Tokens (themes) are recorded in uppercase, lowercase, and mixed-case in a theme index. The case for the entry is determined by how the theme is represented in the Knowledge Catalog. If the theme is not in the Knowledge Catalog, the case for the entry is identical to the theme as it appears in the text of the document.

## Linguistic Settings

ConText uses linguistic settings, specified as setting configurations, to perform special processing for text that is in all-uppercase or all-lowercase. The Linguistics also use the settings to determine the size of theme summaries and the size and generation method for Gists.

ConText provides two predefined setting configurations:

- GENERIC (mixed-case text)
- SA (all-uppercase or all-lowercase text)

GENERIC is the default predefined setting configuration and is automatically enabled for each ConText server at start up.

In addition, you can create your own custom setting configurations in either of the GUI administration tools provided in the ConText Workbench.

---

---

**Note:** Theme indexing is *not* currently supported for all-uppercase or all-lowercase text. In addition, the other linguistic settings only affect Gist and theme summary generation for the ConText Linguistics.

As such, you do *not* need to create custom setting configurations and should *always* use the default setting configuration, GENERIC, for theme indexing.

---

---

**See Also:** For more information about Linguistics, Gists, and theme summaries, as well as the linguistic settings, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## Index Fragmentation

Because the number of distinct themes in a collection of documents is usually fewer than the number of distinct tokens, theme indexes generally contain fewer entries than text index. As a result, index fragmentation is not as much of an issue in theme indexes as in text indexes; however, some fragmentation may occur during theme indexing.

Similar to text indexes, index fragments in theme indexes can be consolidated through the CTX\_DDL.OPTIMIZE\_INDEX procedure.

## DDL and DML

Theme indexes are processed identically to text indexes, meaning that requests for index creation and optimization are processed by any currently available DDL servers.

Similarly, theme index updates are processed by either the DML or DDL servers that are running at the time, depending on the DML index update method (immediate or batch) you are using.

In contrast, Linguistics requests, such as theme and Gist/theme summary generation, use ConText servers with the Linguistic personality.

**See Also:** For more information about DDL and DML operations, see “DDL” and “DML” in this chapter.



## Base-letter Conversion

For each text column in a table, you can specify whether the characters used in single-byte (8-bit), non-English languages are to be converted to their base-letter representation. This means that words with diacritical marks (accents, umlauts, etc.) are converted to their base form before their tokens are stored in the text index for the column.

## Text Indexing

Base-letter conversion is an attribute that you can set when creating a Lexer preference.

If base-letter conversion is enabled for the Lexer preference in a policy, during text indexing of the column for the policy, all characters containing diacritical marks are converted to their base form in the text index. The original text is not affected.

Base-letter conversion requires that the database character set is a subset of the NLS\_LANG character set.

For example, suppose the NLS\_LANG environment variable is set to *French\_France.WE8ISO8859P1* and the following piece of text is to be converted to its base-letter representation:

La référence de session doit être égale à 'name'

The sentence is indexed as:

la reference de session doit etre egale a name

---

---

**Note:** Base-letter conversion requires that the *language* component for NLS\_LANG is set to a single-byte language (e.g. *French*, *German*) that supports an extended (8-bit) character set. In addition, the *charset* component must be set to one of the 8-bit character sets (e.g. *WE8ISO8859P1*).

---

---

**See Also:** For more information about enabling base-letter conversion for a text column, see “BASIC LEXER Tile” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

For more information about National Language Support and the NLS\_LANG environment variable, see *Oracle8 Server Reference Manual*.

## Text Queries

In a text query on a column with base-letter conversion enabled, the query terms are automatically converted to match the base-letter conversion that was performed during text indexing.

---

---

**Note:** Base-letter conversion works with all of the query operators (logical, control, expansion, thesaurus, etc.), *except* the STEM expansion operator.

---

---

**See Also:** For more information about text queries and the query operators, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## Thesauri

Users looking for information on a given topic may not know which words have been used in documents that refer to that topic.

ConText enables users to create case-sensitive or case-insensitive thesauri which define relationships between lexically equivalent words and phrases. Users can then retrieve documents that contain relevant text by expanding queries to include similar or related terms as defined in a thesaurus.

The ConText thesauri formats and functionality are compliant with both ISO-2788 and ANSI Z39.19 (1993).

## Thesaural Maintenance

Thesauri are stored in internal tables owned by CTXSYS. Each thesaurus is uniquely identified by a name that is specified when the thesaurus is created.

### Thesaurus Creation and Modification

Thesauri can be created and modified by all ConText users with the CTXAPP role.

ConText supports thesaural maintenance through PL/SQL (CTX\_THES package) and the System Administration tool.

---

---

**Note:** Thesauri can be created, updated, and deleted by all users with the CTXAPP role.

---

---

In addition, the ctxload utility can be used for loading (creating) thesauri from a load file into the thesaurus tables, as well as dumping thesauri from the tables into output (dump) files.

The thesaurus dump files created by ctxload can be printed out or used as input for other applications. The dump files can also be used to load a thesaurus into the thesaurus tables. This can be useful for using an existing thesaurus as the basis for creating a new thesaurus.

**See Also:** For more information, see “CTX\_THES: Thesaurus Management” in Chapter 11, “PL/SQL Packages - Text Management”.

For more information about ctxload, see Chapter 10, “Text Loading Utility”.

### Default Thesaurus

*Before* the query operators can be used in a query expression, a thesaurus named 'DEFAULT' *must* be created either through the GUI administration tools, CTX\_THES.CREATE\_INDEX or through ctxload.

The reason for this is because the thesaurus that is automatically used by the thesaurus operators is named DEFAULT, unless a different thesaurus is explicitly called by name in the query expression.

### Case-sensitivity

Thesauri support case-sensitivity. In other words, terms are stored in a case-sensitive thesaurus exactly as entered. In addition, for terms that are expanded using thesaurus operators in a text query, the case of the terms is retained and used for thesaural look-up.

To support case-sensitive thesauri, the *case\_sensitive* parameter is provided for the CTX\_THES.CREATE\_THESAURUS procedure. In addition, ctxload provides an argument, *-thescase*, to support importing case-sensitive thesauri.

### Query Expansion

The expansions returned by the thesaurus operators are combined using the ACCUMULATE operator ( , ) in the query expression.

---

---

**Note:** ConText supports creating multiple thesauri; however, only one thesaurus can be used at a time in a query.

---

---

**See Also:** For more information about using thesauri and the thesaurus operators to expand queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Text and Theme Queries

Thesauri are primarily used for expanding text queries, but can be used for expanding theme queries, provided a thesaurus has been created for the themes that can be generated by ConText.

### Limitations

In a query, the expansions generated by the thesaurus operators don't follow nested thesaural relationships. In other words, only one thesaural relationship at a time is used to expand a query.

For example, B is a narrower term for A. B is also in a synonym ring with terms C and D, and has two related terms, E and F. In a narrower term query for A, the following expansion occurs:

NT(A) query is expanded to {A}, {B}

---

---

**Note:** The query expression is *not* expanded to include C and D (as synonyms of B) or E and F (as related terms for B).

---

---

## Types of Thesaural Relationships

Three types of relationships can be defined for terms (words and phrases) in a thesaurus:

- Synonyms
- Hierarchical Relationships
- Related Terms

In addition, each entry in a thesaurus can have Scope Notes associated with it.

---

---

**Note:** ConText supports creating multiple thesauri; however, only one thesaurus can be used at a time in a query.

---

---

**See Also:** For more information about using thesauri to expand queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## Synonyms

car <b>SYN</b> auto	main <b>SYN</b> principal
auto <b>SYN</b> automobile	main <b>SYN</b> major
	main <b>SYN</b> predominant

Support for synonyms is implemented through synonym entries in a thesaurus. The collection of all of the synonym entries for a term and its associated terms is known as a synonym ring.

Synonyms support the following entries:

- Synonym Rings
- Preferred Terms

### Synonym Rings

Synonym rings are transitive. If term A is synonymous with term B and term B is synonymous with term C, term A and term C are synonymous. Similarly, if term A is synonymous with both terms B and C, terms B and C are synonymous. In either case, the three terms together form a synonym ring.

For example, in the synonym rings shown in this example, the terms *car*, *auto*, and *automobile* are all synonymous. Similarly, the terms *main*, *principal*, *major*, and *pre-dominant* are all synonymous.

---

---

**Note:** A thesaurus can contain multiple synonym rings; however, synonym rings are not named. A synonym ring is created implicitly by the transitive association of the terms in the ring.

As such, a term cannot exist twice within the same synonym ring or within more than one synonym ring in a thesaurus.

---

---

Synonym rings are not named, but they have an ID associated with them. The ID is assigned when the synonym entry is first created.

### Preferred Terms

Each synonym ring can have one, and only one, term that is designated as the preferred term. A preferred term is used in place of the other terms in a synonym ring when one of the terms in the ring is specified with the PT operator in a query.

---

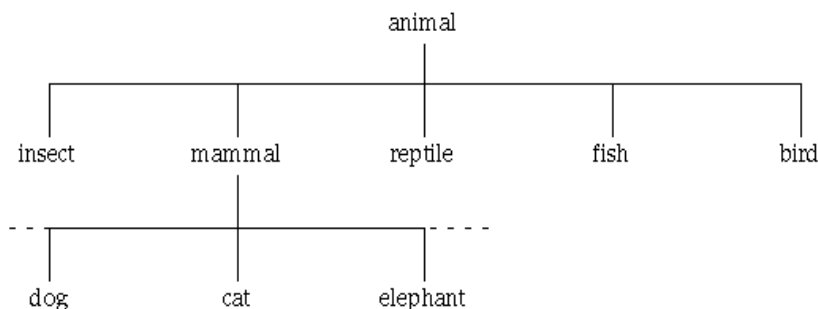
---

**Note:** A term in a preferred term (PT) query is replaced by, rather than expanded to include, the preferred term in the synonym ring.

---

---

## Hierarchical Relationships



Hierarchical relationships consist of broader and narrower terms represented as an inverted tree. Each entry in the hierarchy is a narrower term for the entry immediately above it and to which it is linked. The term at the root of each tree is known as the top term.

For example, in the tree structure shown in the following example, the term *elephant* is a narrower term for the term *mammal*. Conversely, *mammal* is a broader term for *elephant*. The top term is *animal*.

ConText also supports the following hierarchical relationships in thesauri:

- Generic Hierarchy
- Partitive Hierarchy
- Instance Hierarchy

Each of the three hierarchical relationships supported by ConText represents a separate branch of the hierarchy and are accessed in a query using different thesaurus operators.

---

---

**Note:** The three types of hierarchical relationships are optional. Any of the three hierarchical relationships can be specified for a term.

---

---



**Generic Hierarchy**

The generic hierarchy represents relationships between terms in which one term is a generic name for the other.

For example, the terms *rat* and *rabbit* could be specified as generic narrower terms for *rodent*.

**Partitive Hierarchy**

The partitive hierarchy represents relationships between terms in which one term is part of another.

For example, the provinces of *British Columbia* and *Quebec* could be specified as partitive narrower terms for *Canada*.

**Instance Hierarchy**

The instance hierarchy represents relationships between terms in which one term is an instance of another.

For example, the terms *Cinderella* and *Snow White* could be specified as instance narrower terms for *fairy tales*.

**Multiple Occurrences of the Same Term**

Because the four hierarchies are treated as separate structures, the same term can exist in more than hierarchy. In addition, a term can exist more than once in a single hierarchy; however, each occurrence of the term must be accompanied by a qualifier.

If a term exists more than once as a narrower term in one of the hierarchies, broader term queries for the term are expanded to include all of the broader terms for the term.

If a term exists more than once as a broader term in one of the hierarchies, narrower term queries for the term are expanded to include the narrower terms for each occurrence of the broader term.

For example, C is a generic narrower term for both A and B. D and E are generic narrower terms for C. In queries for terms A, B, or C, the following expansions take place:

NTG(A) expands to {C}, {A}

NTG(B) expands to {C}, {B}

NTG(C) expands to {C}, {D}, {E}

BTG(C) expands to {C}, {A}, {B}

---

---

**Note:** The same expansions hold true for standard, partitive, and instance hierarchical relationships.

---

---

### Qualifiers

For homographs (terms that are spelled the same way, but have different meanings) in a hierarchy, a qualifier must be specified as part of the entry for the word. When homographs that have a qualifier for each occurrence appear in a hierarchy, each term is treated as a separate entry in the hierarchy.

For example, the term *spring* has different meanings relating to seasons of the year and mechanisms/machines. The term could be qualified in the hierarchy using the terms *season* and *machinery*.

To differentiate between the terms during a query, the qualifier must be specified. Then, only the terms that are broader terms, narrower terms, or related terms for the term *and* its qualifier are returned. If no qualifier is specified, all of the related, narrower, and broader terms for the terms are returned.

---

---

**Note:** In thesaural queries that include a term and its qualifier, the qualifier must be escaped, because the parentheses required to identify the qualifier for a term will cause the query to fail.

---

---

## Related Terms

Each entry in a thesaurus can have one or more related terms associated with it. Related terms are terms that are close in meaning to, but not synonymous with, their related term. Similar to synonyms, related terms are reflexive; however, related terms are not transitive.

If a term that has one or more related terms defined for it is specified in a related term query, the query is expanded to include all of the related terms.

For example, B and C are related terms for A. In queries for A, B, and C, the following expansions take place:

RT(A) expands to {A}, {B}, {C}

RT(B) expands to {A}, {B}

RT(C) expands to {C}, {A}

---

---

**Note:** Terms B and C are not related terms and, as such, are not returned in the expansions performed by ConText.

---

---

## Scope Notes

Each entry in the hierarchy, whether it is a main entry or one of the synonymous, hierarchical, or related entries for a main entry, can have scope notes associated with it.

Scope notes can be used to provide descriptions or comments for the entry.

## Document Sections

Text queries tend toward low precision. Structured or meta-data can be used to create more precise text queries, thus increasing query precision.

Often, this data is embedded in the document itself. For example, HTML documents have title information, paragraph offsets, and other document attributes as part of the content. E-mail messages are another example in which the text of the message may contain fields with consistent, regularly-occurring headers such as *subject:* and *date:*.

With document sections, ConText allows users to leverage the structure of documents to increase text query precision. Users define rules for dividing documents into sections. ConText includes the section information in the ConText text index for a column so that text queries on the column can be restricted to a specified section.

---

---

**Note:** Section searching does not apply to theme queries. As such, defining document sections for theme indexes is not supported.

In addition, because section information is stored in the text index during indexing, if you want to use section searching sections for columns with existing text indexes, you must drop the indexes, create the required sections, section groups, and preferences, then reindex the columns.

---

---

## Sections

A section is a user-defined body of text, delimited by tags, within a document. Sections are named and grouped into section groups.

---

---

**Note:** A section is not created as an individual object. Instead, a section is created by adding the section to an existing section group.

For examples of adding, as well as removing, sections in section groups, see “Managing Document Sections” in Chapter 9, “Setting Up and Managing Text”.

---

---

## Start and End Tags

The beginning of a section is explicitly identified by a start tag, which can be any token in the text, provided the token can be recognized by the lexer for the text column. Each section *must* have a start tag.

The end of a section can be identified explicitly by an end tag or implicitly by the occurrence of the next occurring start tag, depending on whether the section is defined as a top-level or self-enclosing section. As a result, end tags can be optional. Similar to start tags, end tags can be any token in the text, provided the token can be recognized by the lexer.

---

---

**Note:** Start and end tags are *not* case-sensitive. The tag '`<head>`' is identical to the tag '`<HEAD>`'.

For documentation purposes, all references to start and end tags in this section are presented in UPPERCASE.

---

---

Start and end tags are stored as part of the ConText index, but do not take up space in the index. For example, a document contains the following string, where `<TITLE>` and `</TITLE>` are defined as start and end tags:

```
<TITLE>cats</TITLE> make good pets
```

The string is indexed by ConText as:

```
cats make good pets
```

which enables searching on phrases such as *cats make*.

In addition, start and end tags do not produce hits if searched upon.

---

---

**Suggestion:** Because each occurrence of a token specified as a start/end tag indicates the beginning/end of a section, specify tokens for start and end tags that are as distinctive as possible. Include any non-alphanumeric characters such as colons ':' or angle brackets '<>' which help to uniquely identify the tokens.

For example, the term *TITLE* by itself does not make a good start tag, because it is a common word and ConText would record the start of a new section each time the term was encountered in the text. A better start tag would be the string `<TITLE>` or *TITLE:*.

---

---

### Top-level Sections

A top-level section is only closed (implicitly) by the next occurring top-level section or (explicitly) by the occurrence of the end tag for the section. End tags are *not* required for top-level sections. In addition, a top-level section implicitly closes all sections that are not defined as top-level.

Top-level sections cannot enclose themselves or each other. As a result, if a section is defined as top-level, it cannot also be defined as self-enclosing.

### Self-Enclosing Sections

A self-enclosing section is only closed (explicitly) when the end tag for the section is encountered or (implicitly) when a top-level section is encountered. As a result, end tags are required for sections that are defined as self-enclosing.

Self-enclosing sections support defining tags such as the table tag `<TD>` in HTML as a start tag. Table data in HTML is always explicitly ended with the `</TD>` tag. In addition, tables in HTML can have embedded or nested tables.

If a section is not defined as self-enclosing, the section is implicitly closed when another start tag is encountered. For example, the paragraph tag `<P>` in HTML can be defined as a start tag for a section that is not self-enclosing, because paragraphs in HTML are sometimes explicitly ended with the `</P>` tag, but are often ended implicitly with the start of another tag.

### Limitations

Sections have the following limitations:

**Implicit Start of Body Sections** ConText does not recognize the start of a body section after the implicit end of a header section.

For example, consider the following e-mail message in which *FROM:*, *SUBJECT:*, and *NEWSGROUPS:* are defined as start tags for three different sections:

```
From: jsmith@ABC.com
Subject: New teams
Newsgroups: arts.recreation, alt.sports
```

New teams have been added to the league.

All of the text following the *NEWSGROUPS:* header tag is included in the header section, including the body of the message.

**Multi-word Start and End Tags** ConText does not support start and end tags consisting of more than one word. Each start and end tag for a section can contain only a single word and the word must be unique for each tag within the section group.

For example:

```
problem description: Insufficient privileges
problem solution: Grant required privileges to file
```

The strings *PROBLEM DESCRIPTION:* and *PROBLEM SOLUTION:* cannot be specified as start tags.

**Identical Start and End Tags** ConText does not recognize sections in which the start and end tags are the same.

For example:

```
:Author:
Joseph Smith
:Author:
:Title:
Guide to Oracle
:Title:
```

The strings *:AUTHOR:* and *:TITLE:* cannot be specified as both start and end tags.

## Section Groups

A section group is the collection of all the sections for a text column. Section groups are assigned by name to a text column through the Wordlist preference in the policy for the column.

### Sections in Section Groups

The start and end tags for a particular section must be unique within the section group to which the section belongs. In addition, within a section group, no start tag can also be an end tag.

Section names do not have to be unique within a section group. This allows defining multiple start and end tags for the same logical section, while making the section details transparent to queries.

## Section Group Management

Section groups can be created and deleted by ConText users with the CTXADMIN or CTXAPP roles. In addition, users with CTXADMIN or CTXAPP can add and remove sections from section groups. Section group names must be unique for the user who creates the section group.

**See Also:** For examples of creating and deleting section groups, as well as adding and removing sections in section groups, see “Managing Document Sections” in Chapter 9, “Setting Up and Managing Text”.

## Startjoin and Endjoin Characters

To enable defining document sections, ConText supports specifying non-alphanumeric characters (e.g. hyphens, colons, periods, brackets) using the *startjoins* and *endjoins* attribute for the BASIC LEXER Tile.

When a character defined as a *startjoins* appears at the beginning of a word, it explicitly identifies the word as a new token and end the previous token. When an character specified as an *endjoins* appears at the end of a word, it explicitly identifies the end of the token.

---

**Note:** Characters that are defined as startjoins and endjoins are included as part of the entry for the token in the ConText index.

---

## Text Filtering

Section searching requires the start and end tags for the document sections to be included in the ConText index. This is accomplished through the use of ConText filters and the (optional) definition of *startjoins* and *printjoins* for the BASIC LEXER Tile.

For HTML text that uses the internal HTML filter, document sections have an additional requirement. Because the internal HTML filter removes all HTML markup during filtering, you must explicitly specify the HTML tags that serve as section start and end tags and, consequently, must not be removed by the filter.

This is accomplished through the *keep\_tag* attribute for the HTML FILTER Tile. The *keep\_tag* attribute is a multi-value attribute that lets users specify the HTML tags to keep during filtering with the internal HTML filter.

For HTML filter that is filtered using an external HTML filter, the filter must provide some mechanism for retaining HTML tags used as section start and end tags.



## Document Section Setup

The process model for creating sections and enabling section searching is as follows:

1. Use `CTX_DDL.CREATE_SECTION_GROUP` to create a section group for the sections.

When you call `CREATE_SECTION_GROUP`, you specify the name of the section group to create.

2. Call `CTX_DDL.ADD_SECTION` for each section that you want to create in your section group.

When you call `ADD_SECTION`, you specify the name of the section, the start and end tags for the section, and whether the section is top-level or self-enclosing.

3. If you are creating sections for HTML documents and you use the internal HTML filter, set the *keep\_tag* attribute (HTML FILTER Tile) once for each of the HTML tags that the filter retains for use as section start and end tags.

Then create a Filter preference for the Tile.

4. If necessary, specify values for the *startjoins* and *endjoins* attributes (BASIC LEXER Tile) and create a Lexer preference for the Tile.
5. Use the *section\_group* attribute (GENERIC WORD LIST Tile) to specify the name of the your section group and create a Wordlist preference for the Tile.
6. Create a policy that includes the section-enabled preferences (Filter, Lexer, and Wordlist) that you created.

When you create the policy, you specify the text column where your HTML text is stored.

**See Also:** For examples of creating section groups and sections, as well as creating a section-enabled Wordlist preference, see “Managing Document Sections” in Chapter 9, “Setting Up and Managing Text”.

For examples of specifying attributes for the HTML FILTER and BASIC LEXER Tiles, see “Filter Examples” and “Lexer Examples” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

## Predefined HTML Section Group and Sections

ConText provides a predefined section group, BASIC\_HTML\_SECTION, which enables section searching in basic HTML documents.

BASIC\_HTML\_SECTION contains the following section definitions:

Section Name	Start Tag	End Tag	Top Level	Self-Enclosing
HEAD	<HEAD>	</HEAD>	Yes	No
TITLE	<TITLE>	</TITLE>	No	No
BODY	<BODY>	</BODY>	Yes	No
PARA	<P>	</P>	No	No
HEADING	<H1>	</H1>	No	No
	<H2>	</H2>	No	No
	<H3>	</H3>	No	No
	<H4>	</H4>	No	No
	<H5>	</H5>	No	No
	<H6>	</H6>	No	No

In addition, the following predefined preferences have been created to support ready-to-use basic HTML section searching:

- Filter preference - BASIC\_HTML\_FILTER
- Lexer preference - BASIC\_HTML\_LEXER
- Wordlist preference - BASIC\_HTML\_WORDLIST

## Section Searching

A query expression operator, WITHIN, is provided for restricting a text query to a particular section.

**See Also:** For more information about the WITHIN operator and performing text queries using document sections, see *Oracle8 ConText Cartridge Application Developer's Guide*.

---

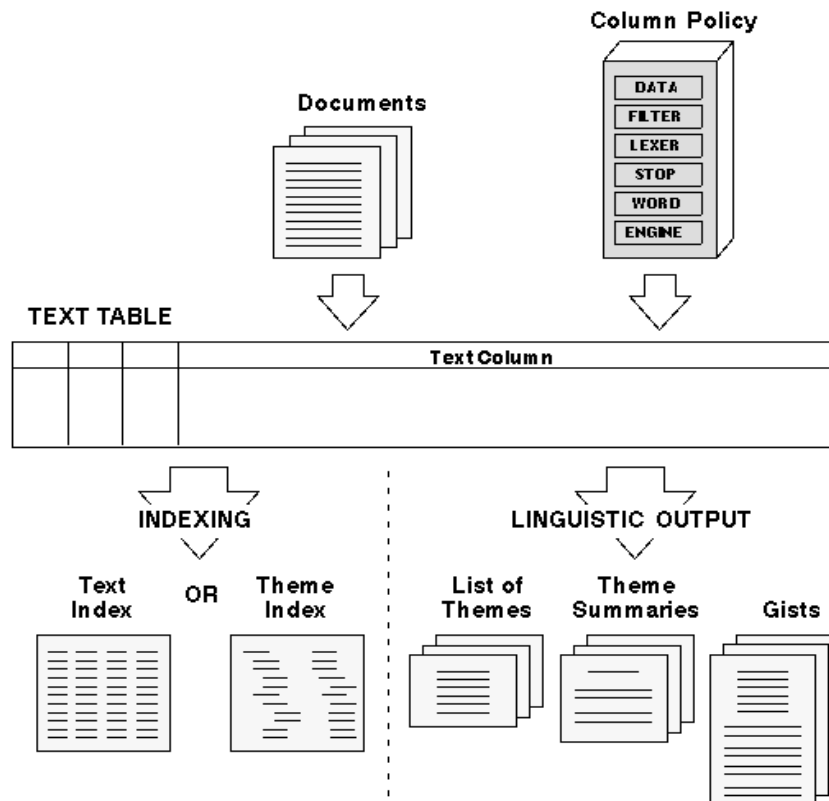
# Understanding the ConText Data Dictionary: Indexing

This chapter introduces the concepts necessary for understanding the objects in the ConText data dictionary.

The following topics are discussed in this chapter:

- Policies
- Preferences for Indexing
- Data Store Tiles
- Filter Tiles
- Lexer Tiles
- Engine Tiles
- Wordlist Tiles
- Stoplist Tiles

## Policies

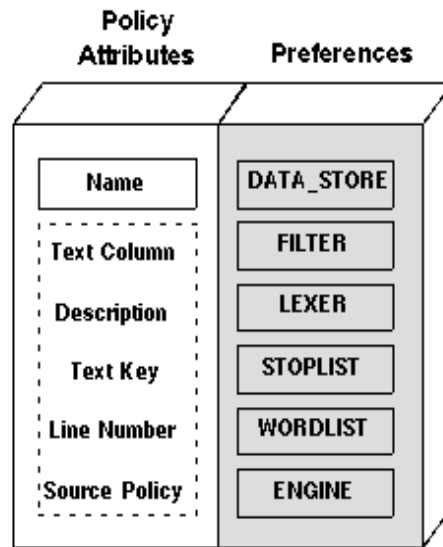


This section provides conceptual, as well as reference, information about policies, which are stored in the ConText data dictionary:

- What is a Policy?
- Column Policies
- Template Policies
- Text Indexing Policies
- Theme Indexing Policies
- Policy Examples

- Preferences in Policies
- Predefined Template Policies

## What is a Policy?



A policy is a logical grouping of six indexing preferences (one preference for each of the supported categories), assigned to a column in the database. A policy specifies the options used by ConText to create the index for the text in the column. It is also used to generate linguistic information for use in ConText applications.

---

**Note:** A policy must exist for a column before a ConText server can create a index or generate linguistic output for the column.

---

Policies can be created by any ConText user with the CTXAPP role. Policies are stored in the ConText data dictionary. In addition to the preferences for a policy, users specify a name for the policy and the text column for the policy, and a number of other policy attributes.

The policies created by a user must be unique for the user. As such, the same policy for a user cannot be assigned to more than one column.

## Column Policies

A column policy is a policy that has a text column assigned to it. Only column policies can be used to create ConText indexes or generate linguistic output.

### Multiple Policies on a Column

Multiple policies, as long as they are unique for the user, can be assigned to a column. As a result, a column can have more than one index.

When a query is performed, you can specify a policy name to indicate the index that is used to process the query.

This feature is particularly useful if you have English-language documents for which you want to enable both text and theme queries. To enable text and theme queries, you must create both a text indexing policy and a theme indexing policy on the column containing the documents and create a ConText index for each policy.

## Template Policies

A template policy is a policy that does not have a text column. Template policies are stored in the ConText data dictionary and are owned by the user who created them.

A template policy can be used by the policy owner as a source policy when creating new column or template policies. When a template policy is used as a source policy in a new policy, all of the preferences for the template policy are copied to the new policy. Any preference from the template policy can be overridden by explicitly naming a preference (for the same category) during the creation of the new policy.

## Text Indexing Policies

A text indexing policy is any policy created with a Lexer preference that uses the BASIC LEXER Tile or one of the Tiles provided for the pictorial languages supported by ConText.

Once a text index is created for the policy, any text requests, including text queries, on the policy will result in the text index being accessed.

**See Also:** For an example of creating a text indexing policy, see “Creating a Column Policy” in Chapter 9, “Setting Up and Managing Text”.

For more information about text indexes, see “Text Indexes” in Chapter 6, “Text Concepts”.

For more information about text queries, see *Oracle8 ConText Cartridge Application Developer’s Guide*.

## Theme Indexing Policies

By specifying the THEME LEXER Tile in the Lexer preference used in a column policy, you designate the policy as a theme indexing policy.

In addition, stoplists are not used by the theme lexer, so a NULL Stoplist preference can be specified for the policy.

Once a theme index is created for a theme indexing policy, any text requests, including queries, on the policy will result in the theme index being accessed.

**See Also:** For an example of creating a theme indexing policy, see “Creating a Theme Indexing Policy” in Chapter 9, “Setting Up and Managing Text”.

For more information about theme indexes, see “Theme Indexes” in Chapter 6, “Text Concepts”.

For more information about theme queries, see *Oracle8 ConText Cartridge Application Developer’s Guide*.

## Policy Examples

Consider a table with two text columns: one holds Microsoft Word documents and the other holds (plain text) comments for the documents. The table structure is:

Table name	Column Name	Datatype	Description
DOC_AND_COMMENT	TEXTKEY	NUMBER	Primary key column
	DATE	DATE	Publishing date of document
	AUTHOR	VARCHAR2(50)	Name of document author
	COMMENTS	VARCHAR2(2000)	Text column storing comments (ASCII text) for documents
	TEXT	LONG RAW	Text column storing MS Word documents

To create a text index for both the *comment* and *doc* columns in *doc\_and\_comment*, a policy must be defined for each column. The following example illustrates two policies named *i\_doc* and *i\_comments* that could be created:

Policy Name	Indexing Option	Indexing Option Value
I_DOC	Text Column	DOC_AND_COMMENT.DOC
	Data Store	Direct (text in column)
	Filter	MS Word
	Lexer	General purpose text lexer
	Engine	General purpose indexing engine
	Stoplist	Default English stoplist
	Wordlist	Soundex and stemming

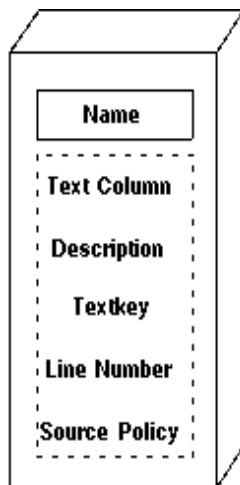


Policy Name	Indexing Option	Indexing Option Value
I_COMMENTS	Text Column	DOC_AND_COMMENT.COMMENTS
	Data Store	Direct (text in column)
	Filter	None (ASCII text)
	Lexer	General purpose lexer
	Engine	General purpose indexing engine
	Stoplist	Default English stoplist
	Wordlist	None

To create a theme index for the *doc* column, a theme indexing policy must be defined. The following example illustrates a policy named *i\_theme* that could be created for the table:

Policy Name	Indexing Option	Indexing Option Value
I_THEME	Text Column	DOC_AND_COMMENT.DOC
	Data Store	Direct (text in column)
	Filter	MS Word
	Lexer	Theme lexer
	Engine	General purpose indexing engine
	Stoplist	Not applicable
	Wordlist	Not applicable

## Policy Attributes



**POLICY ATTRIBUTES**

To define a policy, a user specifies a name for the policy and a number of optional attributes.

### Policy Name

Because a policy is owned by the user who creates it, the policy name must be unique for a user; however, different users can have policies with the same name.

### Optional Attributes

The following policy attributes are optional:

**Text Column** specifies the column in a table to which a policy is assigned. It is the column used to store text in the table.

---

---

**Note:** If the policy does not include a text column, the policy is a template policy, which can be used as a source policy in another policy.

---

---

**Description** specifies a description of the policy.

**Textkey** specifies the primary key column or columns (up to sixteen) for the table. This attribute is required if the policy is being assigned to a column.

**Line Number** specifies the column storing the unique identifier for the text column in a master-detail table. A master-detail table does not store a document as a single row, but rather breaks the document (identified by the textkey) into sections and stores each section in a separate row in the table. The collection of rows with the same textkey represents the whole document.

---

---

**Note:** This attribute is used *only* for policies that include a preference for the MASTER DETAIL Tile.

---

---

**Source Policy** specifies an existing template policy that you want to use as the basis for a new policy. When you specify a source policy in a policy, all of the preferences for the template (source) policy are copied into the new policy. The preferences from the source policy can be overwritten by explicitly specifying a preference for the category.

---

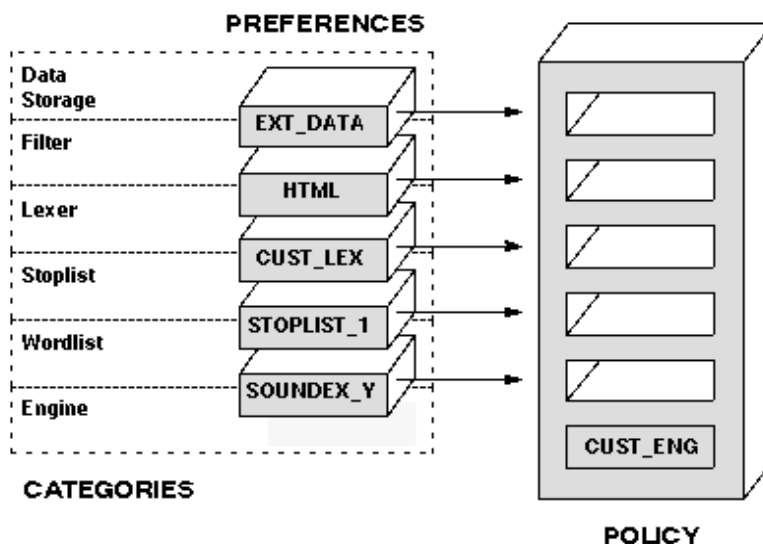
---

**Note:** When specifying a source policy in a policy, a user can specify either their template policies or CTXSYS-owned template policies.

---

---

## Preferences in Policies



To define a policy, the user specifies a preference for each of the six supported categories. ConText does not require the user to specify a preference for the seventh category, Compressor, because data compression is not currently supported.

A preference can be used in more than one policy; however, two preferences from the same category cannot be used in the same policy.

---

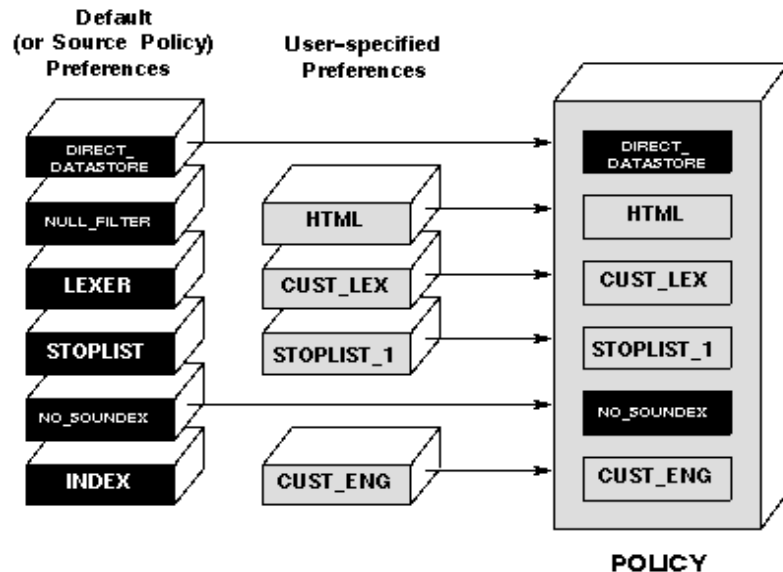
---

**Note:** If you want to use the same preferences for two text columns, you *must* create two separate policies. The policies will be identical (having all of the same preferences), but they must have *unique* names and be attached to *different* columns. This is true whether the columns are in the same table or in different tables.

---

---

## Preference Defaults



In a policy, if a user does not specify a preference for one of the preference categories, ConText uses the default preference for the category.

The above figure illustrates how the default preferences and user-specified preferences work together to create a complete policy.

## Predefined Template Policies

ConText provides the following template policies (listed in alphabetical order):

- DEFAULT\_POLICY (Default)
- TEMPLATE\_AUTOB
- TEMPLATE\_BASIC\_WEB
- TEMPLATE\_DIRECT
- TEMPLATE\_LONGTEXT\_STOPLIST\_OFF
- TEMPLATE\_LONGTEXT\_STOPLIST\_ON
- TEMPLATE\_MD

- TEMPLATE\_MD\_BIN
- TEMPLATE\_WW6B

**DEFAULT\_POLICY**

This template policy uses all of the default preferences. It can be used to create a policy with the following characteristics:

Preferences	Characteristics
DEFAULT_DIRECT_DATASTORE	Text stored in database
DEFAULT_NULL_FILTER	No filter (text stored in plain, ASCII format)
DEFAULT_LEXER	Basic lexer (standard punctuation and continuation characters, no <i>printjoins</i> or <i>skipjoins</i> characters)
DEFAULT_INDEX	Indexing memory = 12582912 bytes, default storage/other clauses for ConText index tables and indexes
NO_SOUNDEX	No Soundex word mappings stored during text indexing
DEFAULT_STOPLIST	Stoplist is active, default list of stop words

---

---

**Note:** DEFAULT\_POLICY is the default for *source\_policy* in both CTX\_DDL.CREATE\_POLICY and CTX\_DDL.CREATE\_TEMPLATE\_POLICY.

---

---

**TEMPLATE\_AUTOB**

This template policy uses the AUTOB predefined preference and all the remaining default preferences. It can be used to create a column policy for a text column that contains documents in mixed formats.

### TEMPLATE\_BASIC\_WEB

This template uses the following predefined preferences and can be used to create a column policy which enables basic section searching for a text column containing HTML documents:

Preferences	Characteristics
DEFAULT_URL	Text stored in external files, URLs to external files stored in text column
BASIC_HTML_FILTER	HTML filter with certain HTML tags specified for <i>keep_tag</i>
BASIC_HTML_LEXER	Basic lexer with characters specified for <i>startjoins</i> and <i>endjoins</i>
DEFAULT_LEXER	Indexing memory = 12582912 bytes, default storage/other clauses for ConText index tables and indexes
BASIC_HTML_WORDLIST	No Soundex word mappings stored during text indexing; HTML section group specified for <i>section_group</i>
DEFAULT_STOPLIST	Stoplist is active, default list of stop words

### TEMPLATE\_DIRECT

This template policy uses all the default preferences. It can be used to create a policy for indexing basic text stored in a text column.

### TEMPLATE\_LONGTEXT\_STOPLIST\_OFF

This template policy uses the NO\_STOPLIST predefined preference and all the remaining default preferences. It can be used to create a policy that does not use a stoplist during indexing.

### TEMPLATE\_LONGTEXT\_STOPLIST\_ON

This template policy uses the DEFAULT\_STOPLIST predefined preference and all the remaining default preferences. It can be used to create a policy that uses the supplied English stoplist during indexing.

### **TEMPLATE\_MD**

This template policy use the MD\_TEXT predefined preference and all the remaining default preferences. It can be used to create a policy for indexing text stored in the detail column in a master-detail table.

### **TEMPLATE\_MD\_BIN**

This template policy uses the MD\_BINARY predefined preference and all the remaining default preferences. It can be used to create a policy for indexing text stored in the detail column in a master-detail table.

### **TEMPLATE\_WW6B**

This template policy uses the WW6B predefined preference and all the remaining default preferences. It can be used to create a policy for indexing text in Microsoft Word for Windows 6 format.



## Preferences for Indexing

This section provides conceptual, as well as reference, information for indexing preferences, which are stored in the ConText data dictionary:

- What is an Indexing Preference?
- What is a Tile?
- Data Store Predefined Preferences
- Filter Predefined Preferences
- Lexer Predefined Preferences
- Engine Predefined Preferences
- Wordlist Predefined Preferences
- Stoplist Predefined Preferences

### What is an Indexing Preference?

Indexing preferences specify the options that ConText uses to create ConText indexes. Each preference represents one (and only one) indexing option.

A preference consists of a ConText Tile and one or more attributes (and their corresponding values) for the Tile.

In addition, each preference is grouped into one of six types or categories, which determine the indexing operation that the preference controls. While a category is not explicitly assigned to a preference, it is implied through the association of the Tile with the preference.

When creating a policy, six preferences are specified for the policy, one for each of the six categories.

#### User-defined Preferences

A ConText user with the CTXAPP role can create their own preferences by setting the required attributes for one of the Tiles provided by ConText, then calling `CTX_DDL.CREATE_PREFERENCE` and specifying the name of the Tile.

---

---

**Note:** When creating a policy, users can use all preferences that have been defined in the ConText data dictionary, including their own preferences, preferences created by other users, or the pre-defined preferences provided by ConText.

---

---

### **Predefined Preferences**

ConText provides a number of predefined preferences (owned by CTXSYS) for each category. These predefined preferences can be used by any ConText user with the CTXAPP role to create policies without having to first create preferences.

## **What is a Tile?**

Tiles are the objects in the ConText data dictionary that provide ConText servers with information about how text is managed in the system, as well as indexing instructions. Each Tile specifies a distinct indexing option within the ConText framework.

A Tile is the main component of a preference. When you define a preference, you specify a Tile and attributes for the Tile, as well as a value for each attribute.

### **Tile Attributes**

Each Tile may have none, one, or many attributes that are specified to define a preference. The attributes identify which indexing options are active for the Tile in a preference.

Each Tile attribute has a value (either a number or a string) that you assign when you specify attributes in a preference.

### **Tile Categories**

The indexing options that must be specified for ConText are divided into seven functional categories or classes.

Each category contains one or more Tiles for which you specify attributes when creating preferences. The Tiles in the categories essentially provide answers to the questions necessary for ConText to generate an index for a text column:

- Where and how is the text stored? (Data Store Tiles)
- What format is the text in? (Filter Tiles)
- Is text compressed? (Compressor Tiles -- Not currently implemented)
- How are tokens in the text identified? (Lexer Tiles)
- How is the index generated and where is it stored? (Engine Tiles)
- Are any special querying functions enabled? (Wordlist Tiles)
- Which words should not have entries in the index? (Stoplist Tiles)

## Data Store Predefined Preferences

ConText provides the following predefined Data Store preferences:

- DEFAULT\_DIRECT\_DATASTORE (Default)
- DEFAULT\_OSFILE
- DEFAULT\_URL
- MD\_BINARY
- MD\_TEXT

### DEFAULT\_DIRECT\_DATASTORE

This preference calls the DIRECT Tile, which is used to indicate that text is stored directly in the text column of a text table.

### DEFAULT\_OSFILE

This preference calls the OSFILE Tile, which is used to indicate that text is stored as files in a file system,

DEFAULT\_OSFILE uses the *path* attribute and a hardcoded set of dummy directory paths to indicate the directories in which the text files are located.

The hard-coded paths, delimited by colons are: /oracle/data, /oracle/data2, /oracle/data3.

---

---

**Note:** If the locations of your files do not match the hard-coded paths, do not use the DEFAULT\_OSFILE preference in a policy.

---

---

### DEFAULT\_URL

This preference calls the URL Tile which is used to indicate that text is stored as URLs.

DEFAULT\_URL uses all of the attribute defaults for the URL Tile:

- timeout of 30 seconds
- up to 8 HTTP threads handled simultaneously
- up to 256 HTML documents can be accessed simultaneously
- the maximum length of a URL stored in the text column is 256 bytes

- the maximum size of an HTML file that the URL data store will access without error is 2 megabytes
- no proxy server

### **MD\_BINARY**

This preference calls the MASTER DETAIL Tile which is used to indicate text is stored in a master detail table.

MD\_BINARY uses the *binary* attribute and a value of YES to indicate that the text in the table is stored in binary format (newline characters do not indicate end of line).

### **MD\_TEXT**

This preference calls the MASTER DETAIL Tile which is used to indicate text is stored in a master detail table.

MD\_TEXT uses the *binary* attribute and a value of NO to indicate that the text in the table is stored in plain text format (newline characters indicate end of line).

## Filter Predefined Preferences

ConText provides the following predefined Filter preferences:

- AUTOB
- BASIC\_HTML\_FILTER
- DEFAULT\_NULL\_FILTER (Default)
- HTML\_FILTER
- WW6B

### AUTOB

This preference calls the BLASTER FILTER Tile which specifies an internal filter used to extract text from formatted documents in a text column.

AUTOB uses the *format* attribute and a value of 997 to indicate that ConText uses the autorecognize filter to extract text. It can be used to filter text in a column that contains the following document formats:

Document Format	Version
AmiPro for Windows	1, 2, 3
ASCII	N/A
HTML	1, 2, 3
Lotus 123 for DOS	4, 5
Lotus 123 for Windows	2, 3, 4, 5
Microsoft Word for Windows	2, 6.x
Microsoft Word for DOS	5.0, 5.5
Microsoft Word for MAC	3, 4, 5.x
Word Perfect for Windows	5.x, 6.x
WordPerfect for DOS	5.0, 5.1, 6.0
Xerox XIF for UNIX	5, 6

### **BASIC\_HTML\_FILTER**

This preference is identical to the `HTML_FILTER` predefined preference, except the *keep\_tag* attribute is set with the following values to support basic section searching in HTML documents:

- 'P'
- 'TITLE'
- 'H1','H2','H3','H4','H5','H6'
- 'HEAD'
- 'BODY'

### **DEFAULT\_NULL\_FILTER**

This preference calls the `FILTER NOP` Tile which indicates that the text column in a text table contains plain, unformatted (ASCII) text and does not require filtering for indexing and highlighting.

### **HTML\_FILTER**

This preference calls the `HTML FILTER` Tile and can be used to filter documents in a column that contains only HTML-formatted documents.

### **WW6B**

This preference calls the `BLASTER FILTER` Tile and specifies a value of 11 for the *format* attribute to indicate ConText uses the Word for Windows 6 filter to extract text. It can be used in a column that contains only Word for Windows 6-formatted documents.

## Lexer Predefined Preferences

ConText provides the following predefined Lexer preferences:

- BASIC\_HTML\_LEXER
- DEFAULT\_LEXER (Default)
- KOREAN
- THEME\_LEXER
- VGRAM\_CHINESE\_1 and VGRAM\_CHINESE\_2
- VGRAM\_JAPANESE\_1 and VGRAM\_JAPANESE\_2

### BASIC\_HTML\_LEXER

This preference is identical to DEFAULT\_LEXER, except the *startjoins* and *endjoins* attributes for the BASIC LEXER Tile are set with '</' and '>' respectively to support basic section searching in HTML documents.

### DEFAULT\_LEXER

This preference calls the BASIC LEXER Tile, which indicates the lexer settings used to identify word and sentence boundaries for text indexing and text queries.

DEFAULT\_LEXER uses the following Tile attributes and values to indicate the lexer settings:

Attribute	Values
<i>punctuations</i>	. ? !
<i>printjoins</i>	NULL (indicates no characters defined as <i>printjoins</i> for the BASIC LEXER)
<i>skipjoins</i>	NULL (indicates <i>no</i> characters defined as <i>skipjoins</i> for the BASIC LEXER)
<i>continuation</i>	- \

### KOREAN

This preference calls the KOREAN LEXER Tile and can be used for parsing Korean text. It has no attributes.

### **THEME\_LEXER**

This preference calls the THEME LEXER Tile, which indicates the preference can be used in a column policy to create theme indexes for a column.

The THEME\_LEXER preference does not set any attributes because the THEME LEXER preference doesn't have any attributes.

### **VGRAM\_CHINESE\_1 and VGRAM\_CHINESE\_2**

This preference call the CHINESE V-GRAM LEXER Tile, which indicates the preferences can be used for parsing Chinese text.

The 1 or 2 indicates that the preference uses either method 1 or 2 for identifying tokens in Chinese text (*hanzi\_indexing* attribute).

### **VGRAM\_JAPANESE\_1 and VGRAM\_JAPANESE\_2**

This preference call the JAPANESE V-GRAM LEXER Tile which indicates the preferences can be used for parsing Japanese text.

The 1 or 2 indicates that the preference uses either method 1 or 2 for identifying tokens in Japanese text (*kanji\_indexing* attribute).

## **Engine Predefined Preferences**

ConText supplies a single predefined Engine preference, DEFAULT\_INDEX.

### **DEFAULT\_INDEX**

This preference calls the GENERIC ENGINE Tile which is used to specify the amount of memory reserved for indexing.

DEFAULT\_INDEX uses the *index\_memory* attribute and specifies the amount of memory allocated for indexing: 12582912 bytes.



## Wordlist Predefined Preferences

ConText provides the following predefined Wordlist preferences:

- BASIC\_HTML\_WORDLIST
- NO\_SOUNDEX (Default)
- SOUNDEX
- VGRAM\_CHINESE\_WORDLIST
- VGRAM\_CHINESE\_WORDLIST

---

---

**Note:** All of the Wordlist preferences call the GENERIC WORD LIST Tile.

---

---

### BASIC\_HTML\_WORDLIST

This preference is identical to the NO\_SOUNDEX preference, except the *section\_group* attribute has a value of 'BASIC\_HTML\_SECTION', which is a pre-defined section group provided by ConText for basic section searching of HTML text.

### NO\_SOUNDEX

This preference specifies a value of 0 for the *soundex\_at\_index* attribute to indicate that ConText does not generate Soundex word mappings during text indexing.

### SOUNDEX

This preference specifies a value of 1 for the *soundex\_at\_index* attribute to indicate that ConText generates Soundex word mappings during text indexing.

### KOREAN\_WORDLIST

This preference specifies a value 3 for the *fuzzy\_match* attribute to ensure fuzzy matching is not enabled for Korean.

### VGRAM\_CHINESE\_WORDLIST

This preference specifies a value 4 for the *fuzzy\_match* attribute to ensure fuzzy matching is not enabled for Chinese.

## VGRAM\_JAPANESE\_WORDLIST

This preference specifies a value 2 for the *fuzzy\_match* attribute to enable fuzzy matching for Japanese.

## Stoplist Predefined Preferences

ConText provides the following predefined Stoplist preferences for creating text indexes:

- DEFAULT\_STOPLIST (Default)
- FRENCH\_STOPLIST
- GERMAN\_STOPLIST
- ITALIAN\_STOPLIST
- NO\_STOPLIST
- SPANISH\_STOPLIST

---

---

**Note:** All of the Stoplist preferences call the GENERIC STOP LIST Tile.

---

---

**See Also:** For a complete list of the stop words for each of the predefined Stoplist preferences, see Appendix A, “Supplied Stoplists”.

## DEFAULT\_STOPLIST

This preference defines a list of English terms treated as stop words during indexing.

---

---

**Note:** The stop words in DEFAULT\_STOPLIST are in all-uppercase. As a result, DEFAULT\_STOPLIST should be used *only* when creating case-insensitive text indexes.

If you are creating case-sensitive text indexes and want to use a stoplist during indexing, first create a Stoplist preference that includes all forms (i.e. uppercase, initial uppercase, lowercase) of the words to be processed as stop words.

For more information about case-sensitivity in text indexes, see “What’s in a Text Index?” in Chapter 6, “Text Concepts”

---

---

**NO\_STOPLIST**

This preference specifies that no list of stop words is used during text indexing. All words that ConText encounters are stored in the text index.

**FRENCH\_STOPLIST**

This preference defines a list of French terms treated as stop words during indexing.

**GERMAN\_STOPLIST**

This preference defines a list of German terms treated as stop words during indexing.

**ITALIAN\_STOPLIST**

This preference defines a list of Italian terms treated as stop words during indexing.

**SPANISH\_STOPLIST**

This preference defines a list of Spanish terms treated as stop words during indexing.

---

**Note:** The stop words in FRENCH\_STOPLIST, GERMAN\_STOPLIST, ITALIAN\_STOPLIST, and SPANISH\_STOPLIST are in all-lowercase. As a result, these preferences can be used in case-sensitive and case-insensitive text indexes; however, if they are used in case-sensitive indexes, stop words that appear in all-uppercase or initial uppercase in your text will not be processed as stop words.

To ensure all stop words are processed correctly in a case-sensitive text index, create a Stoplist preference that includes all forms (i.e. uppercase, initial uppercase, lowercase) of the words to be processed as stop words.

For more information about case-sensitivity in text indexes, see “What’s in a Text Index?” in Chapter 6, “Text Concepts”

---

# Data Store Tiles

Data Store Tiles are used to create preferences which specify how text (data) is stored in the database. ConText supports the following methods of storing text in the database:

- direct
- master/detail
- external operating-system files
- external Web files

**See Also:** For more information about text storage, see “Text Storage” in Chapter 6, “Text Concepts”.

## List of Data Store Tiles and Attributes

ConText provides the following Data Store Tiles:

Tile	Attributes	Attribute Values
DIRECT	** none **	N/A
MASTER DETAIL	binary	0 (plain text) 1 (binary text)
MASTER DETAIL NEW	binary	0 (plain text) 1 (binary text)
	detail_table	<i>name of the detail table</i> (string)
	detail_key	<i>name of the foreign key column in the detail table</i> (string)
	detail_lineno	<i>name of the line number column in the detail table</i> (string)
	detail_text	<i>name of the text column in the detail table</i> (string)

Tile	Attributes	Attribute Values
OSFILE	detail_text_size	Internal use only
	path	<i>path1:path2:...:pathn</i>
URL	timeout	<i>seconds</i> (0 to 3600, default 30)
	maxthreads	<i>number of threads</i> (0 to 1024, default 8)
	maxurls	<i>buffer length in bytes</i> (1 to 4294967295, default 256)
	urlsize	<i>URL length</i> (32 to 65535, default 256)
	maxdocsize	<i>document size</i> (256 to 4294967295, default 2000000)
	http_proxy	<i>host name</i>
	ftp_proxy	<i>host name</i>
	no_proxy	<i>string</i> (up to 16 strings, separated by commas)

## DIRECT Tile

The DIRECT Tile is used for text stored directly in the database. It has no attributes.

## MASTER DETAIL Tile

The MASTER DETAIL Tile is used for text stored directly in the database in master-detail tables, with the textkey column located in the detail table. The column policy is assigned to this column.

### Attributes

MASTER DETAIL has the following attribute(s):

#### binary

The *binary* attribute specifies whether the text in a master detail table is in plain text format (0) or binary format (1).

Text in plain text format uses newline characters at the end of each line to indicate the end of the line. In contrast, binary format does not use newline characters to indicate the end of the line.

## MASTER DETAIL NEW Tile

The MASTER DETAIL NEW Tile is used for text stored directly in the database in master-detail tables, with the *textkey* column located in the master table. The column policy is assigned to this column and all detail information is stored in the Data Store preference, rather than the column policy.

### Attributes

MASTER DETAIL NEW has the following attribute(s):

#### **binary**

The *binary* attribute specifies whether the text in a master detail table is in plain text format (0) or binary format (1).

#### **detail\_table**

The *detail\_table* attribute specifies the name of the detail table in the master-detail relationship.

#### **detail\_key**

The *detail\_key* attribute specifies the name of the foreign key column in the detail table.

#### **detail\_lineno**

The *detail\_lineno* attribute specifies the name of the column in the detail table that identifies rows in the table.

#### **detail\_text**

The *detail\_text* attribute specifies the name of the text column in the detail table.

## OSFILE Tile

The OSFILE Tile is used for text stored in files accessed through the local file system.

### Attributes

OSFILE has the following attribute(s):

#### **path**

The *path* attribute specifies the location of text files that are stored externally in a file system.

Multiple paths can be specified for *path*, with each path separated by a colon (:). File names are stored in the text column in the text table. If *path* is not used to specify a path for external files, ConText requires the path to be included in the file names stored in the text column.

---

---

**Note:** If text is stored in external files rather than in a database, the files must be accessible from the host machine on which the ConText server is running.

This can be accomplished by storing the files in the file system for the host machine or by mounting the file system where the files are stored to the host machine.

---

---

## URL Tile

The URL Tile is used for text stored:

- in files in the local file system (accessed through the file protocol)
- in files on the World Wide Web (accessed through HTTP or FTP)

### Attributes

URL has the following attribute(s):

#### **timeout**

The *timeout* attribute specifies the length of time, in seconds, that a network operation such as 'connect' or 'read' waits before timing out and returning a timeout error to the application. The valid range for *timeout* is 0 to 3600 and the default is 30.

---

---

**Note:** Since timeout is at the network operation level, the total timeout may be longer than the time specified for *timeout*.

---

---

### **maxthread**

The *maxthreads* attribute specifies the maximum number of threads that can be running at the same time. The valid range for *maxthreads* is 1 to 1024 and the default is 8.

---

---

**Note:** The upper range of *maxthreads* corresponds to the number of file descriptors that the operating system can process at one time. If the number of files the operating system can process at one time is less than the value set, an invalid socket error may be returned.

---

---

### **maxurls**

The *maxurls* attribute specifies the maximum number of rows that the internal buffer can hold for HTML documents (rows) retrieved from the text table. The valid range for *maxurls* is 1 to 4294967295 and the default is 256.

### **urlsize**

The *urlsize* attribute specifies the maximum length, in bytes, that the URL data store supports for URLs stored in the database. If a URL is over the maximum length, an error is returned. The valid range for *urlsize* is 32 to 65535 and the default is 256.

---

---

**Note:** The values specified for *maxurls* and *urlsize*, when multiplied, cannot exceed 5000000.

In other words, the maximum size of the memory buffer (*maxurls* \* *urlsize*) for the URL Tile is approximately 5 Megabytes.

---

---

### **maxdocsize**

The *maxdocsize* attribute specifies the maximum size, in bytes, that the URL data store supports for accessing HTML documents whose URLs are stored in the database. The valid range for *maxdocsize* is 1 to 4294967295 and the default is 200000 (2 Mb).



**http\_proxy**

The *http\_proxy* attribute specifies the fully-qualified name of the host machine that serves as the HTTP proxy (gateway) for the machine on which ConText is installed. This attribute must be set if the machine is in an intranet that requires authentication through a proxy server to access Web files located outside the firewall.

**ftp\_proxy**

The *ftp\_proxy* attribute specifies the fully-qualified name of the host machine that serves as the FTP proxy (gateway) for the machine on which ConText is installed. This attribute must be set if the machine is in an intranet that requires authentication through a proxy server to access Web files located outside the firewall.

**no\_proxy**

The *no\_proxy* attribute specifies a string of domains (up to sixteen, separate by commas) which are found in most, if not all, of the machines in your intranet. When one of the domains is encountered in a host name, no request is sent to the machine(s) specified for *ftp\_proxy* and *http\_proxy*. Instead, the request is processed directly by the host machine identified in the URL.

For example, if the string 'us.oracle.com, uk.oracle.com' is entered for *no\_proxy*, any URL requests to machines that contain either of these domains in their host names are not processed by your proxy server(s).

## Data Store Example

The following example creates a preference named *doc\_ref* for the OSFILE Tile:

```
begin
  ctx_ddl.set_attribute ('PATH', '/private/mydocs');
  ctx_ddl.create_preference ('DOC_PREF', 'Path my for my documents' 'OSFILE');
end;
```

---

---

**Note:** This example illustrates usage of OSFILE for documents stored in a UNIX-based environment.

The directory path syntax may be different for other environments.

---

---

# Filter Tiles

Filter Tiles are used to create preferences which determine how text is filtered for indexing and highlighting. Filters allow word processor and formatted documents, as well as ASCII and HTML text documents, to be indexed and highlighted by ConText.

For formatted documents, ConText stores documents in their native format and uses filters to build temporary ASCII versions of the documents. ConText indexes the temporary ASCII text of the formatted document. ConText also uses the ASCII version to highlight query terms.

ConText provides internal filters for processing many of the popular document formats, including Microsoft Word, WordPerfect, and AmiPro.

In addition, ConText allows users to specify external filters for filtering documents in formats not supported by the internal filters provided with ConText.

External filters can also be used to perform operations, such as cleaning up or converting text, before the text is filtered for indexing and highlighting.

**See Also:** For examples of creating Filter preferences, see “Creating Filter Preferences” in Chapter 9, “Setting Up and Managing Text”.

For more information about internal and external filters, see “Text Filtering” in Chapter 6, “Text Concepts”.

## List of Filter Tiles and Attributes

ConText provides the following Filter Tiles:

Tile	Attributes	Attribute Values
BLASTER FILTER	executable	<i>format id</i> (number), <i>filter executable</i> , <i>sequence</i> (number)
	format	0 or 999 (No filter -- plain/ASCII text)
		1 or 4 (Word Perfect for Windows 5.x; Word Perfect for DOS 5.0, 5.1)
		2 (MS Word for DOS 5.0, 5.5)
		5 (Word Perfect for Windows 6.x; Word Perfect for DOS 6.0)
		6 (MS Word for Mac 3, 4, 5.x)

Tile	Attributes	Attribute Values
		7 (MS Word for Windows 2)
		8 (AMIPRO for Windows 1, 2, 3)
		9 (Lotus 1-2-3 for Windows 2, 3, 4, 5; Lotus 1-2-3 for DOS 4, 5)
		11 (MS Word for Windows 6.x, 7.0)
		13 (Xerox XIF for UNIX 5, 6)
		997 (Autorecognize)
FILTER NOP	** none **	N/A
HTML FILTER	code_conversion	0 (disabled) 1(enabled)
	keep_tag	tag (string), sequence (number)
USER FILTER	command	filter executable

## BLASTER FILTER Tile

The BLASTER FILTER Tile is used to specify that the internal filters are used to filter documents. It can also be used to specify that multiple external filters are used to filter documents in a mixed-format column.

### Attributes

BLASTER FILTER has the following attribute(s):

#### format

The *format* attribute specifies the internal filter used for filtering text stored in a text column.

#### executable

The *executable* attribute specifies the external filters that are used to filter text stored in a mixed-format text column. It has three values that must be specified:

- *format\_id* (document format for the external filter)
- *filter\_executable* (name of executable that performs the filtering for the document format)
- *sequence\_num* (identifier for the executable and document format used in the preference)

---

---

**Note:** *format* and *executable* cannot both be set in the same preference.

---

---

**See Also:** For a list of the format IDs supported by the *executable* attribute, see “Supported External Filter Formats for Mixed-Format Columns” in Chapter 6, “Text Concepts”.

## FILTER NOP Tile

The FILTER NOP Tile is used to specify that plain text is stored in the text column and no filtering needs to be performed. It has no attributes.

## HTML FILTER Tile

The HTML FILTER Tile is used to specify that the internal HTML filter is used to filter plain text that contains HTML tags.

### Attributes

HTML\_FILTER has the following attribute(s):

#### **code\_conversion**

The *code\_conversion* attribute specifies whether code conversion is enabled for documents which contain Japanese ASCII text with HTML tags.

Code conversion is required for Japanese HTML documents if the documents use more than one of the three character sets supported for HTML text in Japanese. If code conversion is enabled, all Japanese HTML documents are converted to a single, common character set before indexing.

The default for *code\_conversion* is 0 (disabled).

---

---

**Note:** For mixed-format columns that use Autorecognize (BLASTER Tile, *format* attribute = 997) or use external filters (BLASTER Tile, *executable* attribute) for all formats except HTML, code conversion is *always* enabled.

---

---

**keep\_tag**

The *keep\_tag* attribute takes two values: the HTML tag to retain during indexing and a sequence number that uniquely identifies the tag.

The following rules apply to *keep\_tag*:

- the angle brackets '<>' that identify tags in HTML are not required when setting *keep\_tag*
- multiple tags can be specified for a Filter preference by calling CTX\_DDL.SET\_ATTRIBUTE once for each tag, then calling CTX\_DDL.CREATE\_PREFERENCE
- the sequence number specified for each tag must be unique within the preference
- if the tag specified for *keep\_tag* contains additional (i.e. meta) information, the additional information is filtered by the HTML filter

For example, *keep\_tag* is set to BODY and the following string occurs in a document:

```
<HTML><BODY BGCOLOR=#ffffff>hello</BODY></HTML>
```

ConText translates the string to:

```
<BODY>hello</BODY>
```

This string is passed to the HTML filter, which ignores the HTML tags, then to the lexer, which indexes the token *hello* as belonging to the BODY section.

## USER FILTER Tile

The USER FILTER Tile is used to specify an external filter for filtering documents in a column.

**Attributes**

USER FILTER has the following attribute(s):

**command**

The *command* attribute specifies the executable for the single external filter used to filter all text stored in a column. If more than one document format is stored in the column, the external filter specified for *command* must recognize and handle all such formats, otherwise the BLASTER FILTER Tile and the *executable* attribute should be used instead of the USER FILTER Tile.

## Filter Examples

The following section provides two Filter preference examples:

### Example 1 (MS Word 6 documents)

The following example creates a preference named *word6* for the BLASTER FILTER Tile:

```
begin
  ctx_ddl.set_attribute('FORMAT', '11');
  ctx_ddl.create_preference('WORD6', 'Microsoft Word docs', 'BLASTER FILTER');
end;
```

### Example 2 (HTML documents with document sections enabled)

The following example creates a preference named *sect\_filt\_pref* for the HTML FILTER Tile:

```
begin
  ctx_ddl.set_attribute('KEEP_TAG', 'TITLE', 1);
  ctx_ddl.set_attribute('KEEP_TAG', 'HEAD', 1);
  ctx_ddl.set_attribute('KEEP_TAG', 'BODY', 1);
  ctx_ddl.set_attribute('KEEP_TAG', 'H1', 1);
  ctx_ddl.create_preference('sect_filt_pref', 'sect search filt', 'HTML FILTER');
end;
```

In this example, the <TITLE>, </TITLE>, <HEAD>, </HEAD>, <BODY>, </BODY>, <H1>, and </H1> HTML tags are retained by the HTML filter during filtering, provided the *startjoins* and *endjoins* attributes for the BASIC LEXER Tile are set appropriately.

---

---

**Note:** When using *keep\_tag* to specify tags to be retained, you do not need to specify the angle bracket or forward slash characters in the tag strings.

---

---

## Lexer Tiles

The Tiles in the Lexer category are used to create preferences which specify the lexer used to perform indexing.

### Text Lexers

A text lexer parses text and identifies tokens for indexing. English and other single-byte languages, including most European languages, can use the same lexer because tokens (words) in those languages are delimited by blank spaces and standard punctuation (commas, periods, question marks, etc.).

Japanese, Chinese, and many other Asian languages are pictorial (multi-byte) languages that cannot be tokenized in the same manner as single-byte languages.

#### Single-Byte Languages

ConText includes a single Lexer Tile, BASIC LEXER Tile, for all of the single-byte languages, such as English (7-bit character set) and the other European languages (8-bit character sets), supported by ConText. The basic lexer also works with languages such as Greek, which have different alphabets, but still utilize blank spaces to delimit words.

#### Multi-Byte Languages

ConText includes three separate Lexer Tiles for processing Japanese, Chinese, and Korean text.

The CHINESE V-GRAM LEXER Tile and JAPANESE V-GRAM LEXER Tile do not rely on finding token boundaries within text; instead, they use a list of terms to match and index patterns of characters at user-specified, variable points of length.

The Japanese and Chinese lexers also work with languages that use a 7-bit character set, such as English. As a result, ConText supports indexing and querying Japanese and Chinese text that also contains English text.

---

---

**Note:** Languages that use an 8-bit character set, such as many of the European languages, are not supported by the Japanese and Chinese lexers.

---

---

The Korean lexer, KOREAN LEXER Tile, works similarly to the Japanese and Chinese lexers by finding character patterns in the text and matching the patterns to a dictionary of terms. However, due to the significant morphological transformations that Korean verbs undergo, the Korean lexer only indexes nouns and noun phrases.

### NLS Compliance

The BASIC LEXER Tile supports all NLS-compliant character sets, including the AL24UTFFSS (UTF-8) character set. UTF-8 is a character set that recognizes the characters from most single-byte and multi-byte character sets.

Users with multi-lingual environments, such as multi-national companies, can specify UTF-8 for a database and use the database to store documents that use any one of the character sets supported by UTF-8. ConText supports indexing all documents stored in a UTF-8 database and queries to the database from clients running any of the UTF-8 supported character sets.

**Supported Languages** The BASIC LEXER Tile currently supports the UTF-8 character set only for space-delimited, single-byte languages, which includes English and other Western European languages.

The BASIC LEXER Tile does not support UTF-8 for the multi-byte languages, nor do the Japanese, Chinese, and Korean lexers currently support UTF-8.

**Enabling the NLS-compliant Lexer** The BASIC LEXER Tile does not require any setup to enable it to handle UTF-8 or other NLS-compliant character sets; however, the NLS\_LANG environment variable must be set to the appropriate language/territory/character set. In addition, the ORA\_NLS32 and ORA\_NLS environment variables must be set to the directories containing the appropriate NLS data.

**Limitations** The lexer has the following limitations when UTF-8 is the character set specified for the database:

- base-letter conversion is *not* supported
- characters from 8-bit character sets are *not* supported in the BASIC LEXER Tile attributes (i.e. *printjoins*, *skipjoins*, *startjoins*, *endjoins*, *punctuations*, *numjoin*, *numgroup*, *continuation*)



## Composite Word Indexing

For German-language text, the BASIC LEXER Tile provides an attribute for enabling composite word indexing. With composite word indexing, tokens that are compound words (specifically, nouns in German text) are divided into their constituent (root) nouns, including inflected forms of the roots, and the roots are stored in the ConText index along with the entry for the compound word.

For example, if the word *Hauptbahnhof* is encountered in a document during composite word indexing, the following entries are created in the index: *HAUPTBAHNHOF*, *HAUPT*, *BAHN*, *BAHNEN*, *HOF*.

---

---

**Note:** Because each token that is encountered has to be processed through the ConText decompounding routines, composite indexing may affect indexing performance.

In addition, because composite word indexes may be substantially larger than standard text indexes, composite word indexing may affect query performance.

---

---

**Supported Character Sets** Composite word indexing supports both single-byte and multi-byte character sets, specifically WE8ISO8859P9 (extended, single-byte) and AL24UTFFSS (multi-byte).

**Limitations** Composite indexes have the following limitations:

- composite indexing must be enabled for text columns containing only German text. If the column contains text in other languages, composite indexing will fail
- composite word indexes do not support exact word searches (i.e. standard text queries). If you want to enable composite *and* exact word queries for a column, you must create a compound index and a standard index for the column
- case-sensitivity is not supported for composite indexes (all tokens are stored in all-uppercase)

---

---

**Note:** The uppercasing of all tokens in a composite index results in the composite routines not recognizing some compound nouns. As a result, those nouns are not divided into their root nouns and are indexed as regular tokens with a single entry only in the index.

---

---

**Word Queries** Composite word indexing enables text queries to return all documents that contain either the query term itself or the query term as a root of a compound word; however, queries for phrases that contain one or more compound words return only the documents that contain the *exact* phrase.

**Note:** For more information about composite word queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

Theme Lexer

For English-language text, a separate lexer, THEME LEXER Tile, is provided for creating theme indexes. This lexer breaks text into tokens; however, the tokens are not stored in the theme index. The tokens are passed to the ConText linguistic core where they are analyzed within the context of the sentences and paragraphs in which they appeared to determine whether they are content-bearing words. The linguistic core then generates themes, which are stored in the theme index.

The themes generated by ConText are based on, but are not identical to, the content-bearing tokens in the text.

**See Also:** For more information about the theme lexer and theme indexing, see “Theme Indexes” in Chapter 6, “Text Concepts”.

List of Lexer Tiles and Attributes

ConText provides the following Lexer Tiles for creating preferences for indexing:

Tile	Attributes	Attribute Values
BASIC LEXER	base_letter	0 (disabled) 1 (enabled)
	continuation	<i>characters</i> (string)
	numgroup	<i>characters</i> (string)
	numjoin	<i>characters</i> (string)
	printjoins	<i>characters</i> (string)
	punctuations	<i>characters</i> (string)
	skipjoins	<i>characters</i> (string)
	startjoins	<i>non-alphanumeric characters that occur at the beginning of a token</i> (string)

Tile	Attributes	Attribute Values
	endjoins	<i>non-alphanumeric characters that occur at the end of a token</i> (string)
	mixed_case	0 (disabled)
		1 (enabled)
	composite	0 (no composite word indexing)
		1 (German composite word indexing)
CHINESE V-GRAM LEXER	hanzi_indexing	1
		2
JAPANESE V-GRAM LEXER	kanji_indexing	1
		2
KOREAN LEXER	** none **	N/A
THEME LEXER	** none **	N/A

## BASIC LEXER Tile

The BASIC LEXER Tile is used to identify tokens for creating text indexes for English and all other supported single-byte languages. It is also used to enable base-letter conversion for single-byte languages that have extended character sets.

---

---

**Note:** Any changes made to tokens before text indexing (e.g. removing of characters, base-letter conversion) are also performed on the query terms in a text query. This ensures that the query terms match the form of the tokens in the text index entries.

---

---

### Attributes

BASIC LEXER has the following attribute(s):

---

---

**Note:** The character strings for the BASIC LEXER Tile attributes can contain multiple characters. Each character in the string serves as a punctuation, join, or continuation character.

For example, if the string '\*\_-' are specified for the *printjoins* attribute, each individual character ('\*', '\_', and '-') in the string is treated by ConText as a join character that is included in the index entry for a token in which the character occurs.

---

---

### **base\_letter**

*base\_letter* specifies whether characters that have diacritical marks (umlats, cedillas, acute accents, etc.) are converted to their base form before being stored in the text index. The default is 0 (base-letter conversion disabled).

### **continuation**

*continuation* specifies the characters that indicate a word continues on the next line and should be indexed as a single token. The most common continuation characters are a hyphen '-' and a backslash '\'.

### **numgroup**

*numgroup* specifies the characters that, when they appear in a string of digits, indicate that the digits are groupings within a larger single unit.

For example, comma ',' or period '.' may be defined as numgroup characters because they often indicate a grouping of thousands when they appear in a string of digits.

### **numjoin**

*numjoin* specifies the characters that, when they appear in a string of digits, cause ConText to index the string of digits as a single unit or word.

For example, period '.' or comma ',' may be defined as numjoin characters because they often serve as decimal points when they appear in a string of digits.

---

---

**Note:** The default values for *numjoin* and *numgroup* are determined by the NLS initialization parameters that are specified for the database.

In general, a value does not need to be specified for either *numjoin* or *numgroup* when creating a Lexer preference for the BASIC LEXER Tile.

---

---

## printjoins

*printjoins* specifies the non-alphanumeric characters that, when they appear anywhere in a word (beginning, middle, or end), are processed by ConText as alphanumeric and included with the token in the text index. This includes *printjoins* that occur consecutively.

For example, if the hyphen '-' and underscore '\_' characters are defined as *printjoins*, terms such as *pseudo-intellectual* and *\_file\_* are stored in the text index as *pseudo-intellectual* and *\_file\_*.

---

---

**Note:** If a *printjoins* character is also defined as a *punctuations* character, the character is only processed as an alphanumeric character if the character immediately following it is a standard alphanumeric character or has been defined as a *printjoins* or *skipjoins* character.

---

---

## punctuations

*punctuations* specifies the non-alphanumeric characters that, when they appear at the end of a word, indicate the end of a sentence or a grouping within a sentence.

Characters that are defined as *punctuations* are removed from a token before text indexing; however, if a *punctuations* character is also defined as a *printjoins* character, the character is only removed if it is the last character in the token and it is immediately preceded by the same character.

For example, if the period (.) is defined as both a *printjoins* and a *punctuations* character, the following transformations take place during indexing and querying as well:

Token	Indexed Token
.doc	.doc
dog.doc	dog.doc
dog..doc	dog..doc
dog.	dog
dog...	dog..

### **skipjoins**

*skipjoins* specifies the non-alphanumeric characters that, when they appear within a word, identify the word as a single token; however, the characters are not stored with the token in the text index.

For example, if the hyphen character '-' is defined as a *skipjoin*, the word *pseudo-intellectual* is stored in the text index as *pseudointellectual*.

---

---

**Note:** *printjoins* and *skipjoins* are mutually exclusive. The same characters cannot be specified for both attributes.

---

---

### **startjoins/endjoins**

*startjoins* specifies the characters that, when encountered as the first character in a token, explicitly identify the start of the token. The character, as well as any other *startjoins* characters that immediately follow it, is included in the ConText index entry for the token. In addition, the first *startjoins* character in a string of *startjoins* characters implicitly end the previous token.

*endjoins* specifies the characters that, when encountered as the last character in a token, explicitly identify the end of the token. The character, as well as any other *startjoins* characters that immediately follow it, is included in the ConText index entry for the token.

The following rules apply to both *startjoins* and *endjoins*:

- the characters specified for *startjoins/endjoins* cannot occur in any of the other attributes for BASIC LEXER.
- *startjoins/endjoins* characters can occur only at the beginning/end of tokens
- multiple, contiguous *startjoins/endjoins* characters are allowed at the beginning/end of a token; however, multiple occurrences of the same *startjoins/endjoins* character at the beginning/end of a token are not supported

---

---

**Note:** Defining *startjoins* and *endjoins* characters is particularly useful for creating document sections that enable section searching in a column.

For examples of creating sections and section groups, see “Managing Document Sections” in Chapter 9, “Setting Up and Managing Text”.

For more information about sections, see “Document Sections” in Chapter 6, “Text Concepts”.

For more information about section searching, see *Oracle8 ConText Cartridge Application Developer’s Guide*.

---

---

### **mixed\_case**

The *mixed\_case* attribute specifies whether the lexer converts the tokens in text index entries to all uppercase or stores the tokens exactly as they appear in the text. The default is 0 (tokens converted to all uppercase).

### **composite**

The *composite* attribute specifies whether composite word indexing is enabled. For the current release, composite indexing is supported for German-language text only. The default is 0 (no composite word indexing).

---

---

**Note:** In a Lexer preference that used the BASIC LEXER Tile, the *composite* and *mixed\_case* attributes cannot both be set. Composite indexes do not support case-sensitivity.

---

---

**See Also:** For more information, see “Composite Word Indexing” in this chapter.

## CHINESE V-GRAM LEXER Tile

The CHINESE V-GRAM LEXER Tile is used for identifying tokens for creating text indexes for Chinese text.

### Attributes

CHINESE V-GRAM LEXER has the following attribute(s):

#### **hanzi\_indexing**

The *hanzi\_indexing* attribute specifies the number of characters used for pattern matching while indexing.

A value of 1 for *hanzi\_indexing* indicates that the Chinese lexer examines each character individually to determine token boundaries.

A value of 2 for *hanzi\_indexing* indicates that the lexer examines characters in pairs to determine token boundaries. Pattern matching using pairs is generally faster than matching individual characters, resulting in faster index creation.

The default is 2.

## JAPANESE V-GRAM LEXER Tile

The JAPANESE V-GRAM LEXER Tile is used for identifying tokens for creating text indexes for Japanese text.

### Attributes

JAPANESE V-GRAM LEXER has the following attribute(s):

#### **kanji\_indexing**

The *kanji\_indexing* attribute specifies the number of characters used for pattern matching while indexing.

A value of 1 for *kanji\_indexing* indicates that the Japanese lexer examines each character individually to determine token boundaries.

A value of 2 for *kanji\_indexing* indicates that the lexer examines pairs of characters to determine token boundaries. Pattern matching using pairs is generally faster than matching individual characters, resulting in faster index creation.

The default is 2.



## **KOREAN LEXER Tile**

The KOREAN LEXER Tile is used for identifying tokens for creating text indexes for Korean text. It has no attributes.

## **THEME LEXER Tile**

The THEME LEXER Tile is used to create theme indexes for English-language text. It has no attributes.

## Lexer Examples

The following section provides two Lexer preference examples that both use the BASIC LEXER Tile.

### Example 1

The following example creates a preference named *doc\_link*:

```
begin
  ctx_ddl.set_attribute      ('PRINTJOINS', '.-@&$#/' );
  ctx_ddl.create_preference ('DOC_LINK', 'numerous joins', 'BASIC LEXER' );
end;
```

In this example, the '.', '-', '@', '&', '\$', '#', and '/' characters are all defined as *printjoins* characters.

Characters such as the dollar sign '\$' and number sign '#' are useful if you want to index tokens that may contain these characters, such as sums of money and numbers.

### Example 2 (startjoins and endjoins)

The following example creates a preference named *section\_pref*:

```
exec ctx_ddl.set_attribute('startjoins','</' );
exec ctx_ddl.set_attribute('endjoins','>');
exec ctx_ddl.set_attribute('printjoins','_@-&$#.');
...
exec ctx_ddl.create_preference('sect_lex_pref','basic lexing + sections','BASIC LEXER');
```

In this example, the characters '<' and '/' are defined as *startjoins* characters. The character '>' is defined as an *endjoins* character.

The open and closed angle brackets '<' >' and the forward slash '/' are useful for identifying HTML tags for document sections.

**See Also:** For more information about sections, see “Document Sections” in Chapter 6, “Text Concepts”

## Engine Tiles

Engine Tiles are used to create preferences which specify how ConText indexes are created by the ConText engine and where in the database the indexes are stored.

The engine is the ConText component that creates a ConText index for a text column. A ConText index is required before text in a column can be queried.

**See Also:** For an example of creating an Engine preference, see “Creating an Engine Preference” in Chapter 9, “Setting Up and Managing Text”.

## List of Engine Tiles and Attributes

ConText provides the following Engine Tiles:

Tile	Attributes	Attribute Values
ENGINE NOP (NOT USED)	** none **	N/A
GENERIC ENGINE	index_memory	<i>memory in bytes (integer)</i>
	optimize_default	<i>default ConText index optimization method</i>
	ilt_tablespace, ilt_storage, ilt_other_parms	<i>tablespace (string), STORAGE clause (string), and other table creation parameters (string) for token table</i>
	ili_tablespace, ili_storage, ili_other_parms	<i>tablespace (string), STORAGE clause (string), and other index creation parameters (string) for index on token table</i>
	ktb_tablespace, ktb_storage, ktb_other_parms	<i>tablespace (string), STORAGE clause (string), and other table creation parameters (string) for mapping table</i>
	kid_tablespace, kid_storage, kid_other_parms	<i>tablespace (string), STORAGE clause (string), and other index creation parameters (string) for indexes on mapping table</i>
	kik_tablespace, kik_storage, kik_other_parms	
	lst_tablespace, lst_storage, lst_other_parms	<i>tablespace (string), STORAGE clause (string), and other table creation parameters (string) for control table</i>

Tile	Attributes	Attribute Values
	lix_tablespace, lix_storage, lix_other_parms	<i>tablespace</i> (string), <i>STORAGE clause</i> (string), and <i>other table creation parameters</i> (string) for index on control table
	sqr_tablespace, sqr_storage, sqr_other_parms	<i>tablespace</i> (string), <i>STORAGE clause</i> (string), and <i>other table creation parameters</i> (string) for SQE results table
	sri_tablespace, sri_storage, sri_other_parms	<i>tablespace</i> (string), <i>STORAGE clause</i> (string), and <i>other table creation parameters</i> (string) for index on SQE results table

ENGINE NOP Tile

The ENGINE NOP Tile specifies that no engine is used for indexing. This Tile is currently not used and should *not* be used to create preferences for indexing.

GENERIC ENGINE Tile

The GENERIC ENGINE Tile specifies that the engine provided by ConText is used for indexing. ConText supplies a single engine that creates index entries for Context indexes, independent of the format, location, language, and character set of the text.

In particular, the GENERIC ENGINE Tile attributes specify the amount of memory allocated for indexing, and the tablespace(s) and creation parameters for the database tables and indexes that constitute a ConText index.

**See Also:** For descriptions of the ConText index tables and indexes, see “Appendix C, “ConText Index Tables and Indexes”.

Attributes

GENERIC ENGINE has the following attribute(s):

**index\_memory**

*index\_memory* specifies the amount of memory, in bytes, allocated for indexing.

---

---

**Note:** When specifying a value for *index\_memory* in a preference, specify as much real (not virtual) memory as is available on the machine which is running the ConText server that will be creating indexes.

For parallel indexing, the memory specified should be the amount of available memory divided evenly among the number of ConText servers that will perform the indexing in parallel.

---

---

### **optimize\_default**

*optimize\_default* specifies the type of optimization used when CTX\_DDL.OPTIMIZE\_INDEX is called without an optimization type. If no value is specified for *optimize\_default*, the default is DEFRAGMENT\_TO\_TWO\_TABLE.

### **xxx\_tablespace**

*ilt\_tablespace*, *kth\_tablespace*, and *lst\_tablespace* specify the tablespaces used for the ConText index tables created during indexing.

*sqr\_tablespace* specifies the tablespace used for the stored query expression result (SQR) table that is created, but not populated, during indexing. The SQR table for a policy stores the results of stored query expressions for the policy.

*ili\_tablespace*, *kid\_tablespace*, *kik\_tablespace*, and *lix\_tablespace* specify the tablespaces used for the Oracle indexes generated for each ConText index table.

*sri\_tablespace* specifies the tablespace used for the Oracle index generated for each SQR table.

---

---

**Note:** For each *xxx\_tablespace* attribute that is not specified when creating an Engine preference, the text table owner's default tablespace is used for storing the ConText index objects (tables and indexes).

---

---

### **xxx\_storage**

*ilt\_storage*, *kth\_storage*, and *lst\_storage* specify the STORAGE clauses used to create the ConText index tables during ConText indexing.

*sqr\_storage* specifies the STORAGE clause used to create the stored query expression result (SQR) table during ConText indexing.

*ili\_storage*, *kid\_storage*, *kik\_storage*, and *lix\_storage* specify the STORAGE clauses used to create the Oracle indexes for each ConText index table.

*sri\_storage* specifies the STORAGE clause used to create the Oracle index for each SQR table.

**See Also:** For more information about the STORAGE clause, see the CREATE TABLE and CREATE INDEX commands in *Oracle8 Server SQL Reference*.

#### **xxx\_other\_parms**

*ilt\_other\_parms*, *ktb\_other\_parms*, and *lst\_other\_parms* specify any additional parameters used to create the ConText index tables during ConText indexing.

*sqr\_other\_parms* specifies any additional parameters used to create the stored query expression result (SQR) table during ConText indexing.

*ili\_other\_parms*, *kid\_other\_parms*, *kik\_other\_parms*, and *lix\_other\_parms* specify any additional parameters used to create the Oracle indexes for each ConText index table.

*sri\_other\_parms* specifies any additional parameters used to create the Oracle index for each SQR table.

---

---

**Note:** In particular, the *xxx\_other\_parms* attributes are used to specify a value for the PARALLEL clause in the CREATE TABLE|INDEX command. The PARALLEL clause determines the degree of parallelism used by the Oracle parallel query option for operations such as generating Oracle indexes.

For more information about the PARALLEL clause in CREATE TABLE and CREATE INDEX, as well as the other parameters that can be used to create database tables and indexes, see *Oracle8 Server SQL Reference*.

For more information about the parallel query option in Oracle, see *Oracle8 Server Tuning*.

---

---

**See Also:** For more information about SQEs, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## Engine Example

The following example creates a preference named *doc\_engine* for the GENERIC ENGINE Tile:

```
begin
  ctx_ddl.set_attribute ('INDEX_MEMORY', 30000000 );
  ctx_ddl.set_attribute ('I1T_TABLESPACE', 'DOCUMENTS' );
  ctx_ddl.set_attribute ('I1T_STORAGE', ' initial 10M next 2M
                                maxextents 10');
  ctx_ddl.set_attribute ('I1T_OTHER_PARS', ' pctfree 20');
  ctx_ddl.set_attribute ('I1I_OTHER_PARS', ' parallel 2');
  ctx_ddl.create_preference ('DOC_ENGINE', 'Test case',
                                'GENERIC ENGINE' );
end;
```

## Wordlist Tiles

The Tiles in the Wordlist category are used to create preferences for enabling three of the ConText query expansion methods:

- Stemming
- Fuzzy Matching
- Soundex

**See Also:** For more information about expanding queries and the query expansion operators provided by ConText, see *Oracle8 Con-Text Cartridge Application Developer's Guide*.

### Stemming

Stemming expands a query by deriving variations (verb conjugation, noun, pronoun, and adjective inflections) of the search token(s) in the query.

For example, a stem search on the verb *buy* expands to include its alternate verb forms, such as *buys*, *buying*, and *bought*, but not on the noun *buyer*. A search on the noun *buyer* would expand only to include its plural form *buyers*.

Since different languages have different stemming rules, stemming is language-dependent and uses term lists that define the relationships between the words in a given language

ConText provides a stemmer, licensed from Xerox Corporation, that utilizes Xerox Lexical Technology to support inflectional and derivational stemming in English and inflectional stemming in a number of Western European languages.

### Fuzzy Matching

Fuzzy matching expands queries by including terms that are spelled similar to the search token in the query. This type of expansion can be useful in queries for text that contains frequent misspellings or has been scanned using OCR software.

For example, a fuzzy matching query for the term *cat* expands to include *cats*, *calc*, *case*.

The number of expansions generated by fuzzy matching depends on the tokens that ConText identified during indexing; results can vary significantly according to the tokens that were identified and indexed by ConText for the column. As such, fuzzy matching depends on how tokens are delimited in a given language.



---

---

**Note:** Fuzzy matching is designed primarily for English-language documents, but can be used, with varying degrees of success with many of the Western European languages.

---

---

## Soundex

During text indexing of a column, Soundex, if enabled, creates a list of all the words that sound alike and assigns one or more IDs to each word to identify the other words in the list that sound like the word.

---

---

**Note:** Soundex is designed primarily to look for matches in phonetic spellings used in English, but can be used, with varying degrees of success with many of the other Western European languages.

---

---

The Soundex wordlist is stored in the DR\_nnnnn\_I1W ConText index table, where *nnnnn* is the identifier of the policy for the text index.

If Soundex is enabled for a text column, users can call Soundex in a query to expand the query. Soundex expands a query by searching the I1W table for terms that sound similar to the specified query term.

For example, a Soundex search on the name *Smith* would also find the names *Smythe* and *Smit*.

---

---

**Note:** Soundex in ConText uses the same algorithm as the SOUNDDEX function in SQL.

For more information about the SOUNDDEX function in SQL, see *Oracle8 Server SQL Reference*.

---

---

## List of Wordlist Tiles and Attributes

ConText provides the following Wordlist Tiles:

Tile	Attributes	Attribute Values
GENERIC WORD LIST	stclause	<i>STORAGE clause</i> (string) for Soundex wordlist table
	instclause	<i>STORAGE clause</i> (string) for index on Soundex wordlist table
	soundex_at_index	0 (disabled)
		1 (enabled)
	stemmer	1 (English)
		2 (English -- derivational)
		3 (Dutch)
		4 (French)
		5 (German)
		6 (Italian)
		7 (Spanish)
	fuzzy_match	1 (English and other Western European languages)
		2 (Japanese)
		3 (Korean)
		4 (Chinese)
		12 (Soundex emulation)
		13 (Dutch)
		14 (French)
		15 (German)
		16 (Italian)
		17 (Spanish)
		18 (OCR text)
	section_group	<i>name of section group</i>

## GENERIC WORD LIST Tile

The GENERIC WORD LIST Tile is used to specify the advanced query options for ConText indexes. ConText provides a single Tile for handling stemming, fuzzy matching, Soundex, and named section searching.

**See Also:** For more information about expansion methods in queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

### Attributes

GENERIC WORD LIST has the following attribute(s):

#### **stclause**

The *stclause* attribute specifies the STORAGE clause used to create the Soundex wordlist table during ConText indexing. The Soundex wordlist table is only created if Soundex is enabled through the *soundex\_at\_index* attribute.

#### **instclause**

The *instclause* attribute specifies the STORAGE clause used to create the Oracle index for the Soundex wordlist table.

#### **soundex\_at\_index**

The *soundex\_at\_index* attribute specifies whether ConText generates Soundex word mappings and stores them in the Soundex wordlist table during text indexing. If Soundex word mappings are not generated and stored in the wordlist table during indexing, queries that use Soundex are not expanded.

#### **stemmer**

The *stemmer* attribute specifies the stemmer used for word stemming in text queries. For all the supported languages, the stemmers return standard inflected forms of a word, such as the plural form (e.g. *department* --> *departments*).

For English, an additional stemmer is provided which returns standard inflected forms and derived forms (e.g. *department* --> *departments*, *departmentalize*).

The default for *stemmer* is 1 (inflectional English)

#### **fuzzy\_match**

The *fuzzy\_match* attribute specifies which fuzzy matching routines are used for the column. Fuzzy matching is currently supported for English, Japanese, and, to a lesser extent, the Western European languages.

The default for *fuzzy\_match* is 1.

---

---

**Note:** The *fuzzy\_match* attribute values for Chinese and Korean are dummy attribute values that prevent the English and Japanese fuzzy matching routines from being used on Chinese and Korean text.

---

---

### **section\_group**

The *section\_group* attribute specifies the name of the section group to assign to a text column. The following rules apply to *section\_group*:

- no default value for *section\_group*
- all available section groups in the ConText data dictionary can be specified for *section\_group*; the section group owner does not need to be the same as the policy owner

**See Also:** For more information about section groups, see “Document Sections” in Chapter 6, “Text Concepts”.

## **Wordlist Example**

The following example creates a preference named *soundex\_yes* for the GENERIC WORD LIST Tile:

```
begin
  ctx_ddl.set_attribute('SOUNDEX_AT_INDEX', '1');
  ctx_ddl.create_preference('SOUNDEX_YES',
                           'Will build the soundex mapping during indexing',
                           'GENERIC WORDLIST');
end;
```

## Stoplist Tiles

The Stoplist Tiles are used to create Stoplist preferences. A stoplist is a list of common terms that ConText does not include in the text index for a text column.

Each stoplist can contain a maximum of 4095 words.

**See Also:** For an example of creating a Stoplist preference, see “Creating a Stoplist Preference” in Chapter 9, “Setting Up and Managing Text”.

## List of Stoplist Tiles and Attributes

The Stoplist category contains the following Tiles:

Tile	Attributes	Attribute Values
GENERIC STOP LIST	stop_word	word (string), sequence (number)

## GENERIC STOP LIST Tile

The GENERIC STOP LIST Tile specifies the terms that should not be included in the text index.

### Attributes

GENERIC STOP LIST has the following attribute(s):

#### stop\_word

The *stop\_word* attribute has two values that must be specified:

- the *word* for which ConText does not create an entry in the text index
- the *sequence* for the word

*sequence* is a value from 1 to 4095 and is used in a text index to record the stop words that proceed and follow an indexed term. ConText records up to eight preceding stop words and eight following stop words for each indexed term. This enables text queries for phrases which contain stop words.

For example, consider the sentence “he is at the top of the class” where *at*, *the*, *top*, and *of* are stop words. The sequences for each of the stop words are recorded as part of the text index entry for the term *class*, which allows users to include stop words in a query (e.g. ‘top of the class’).

## Stoplist Example

The following example creates a preference named *mini\_stoplist* for the GENERIC STOP LIST Tile:

```
begin
  ctx_ddl.set_attribute      ('STOP_WORD', 'a', 1);
  ctx_ddl.set_attribute      ('STOP_WORD', 'A', 2);
  ctx_ddl.set_attribute      ('STOP_WORD', 'the', 3);
  ctx_ddl.set_attribute      ('STOP_WORD', 'The', 4);
  ctx_ddl.set_attribute      ('STOP_WORD', 'and', 5);
  ctx_ddl.set_attribute      ('STOP_WORD', 'And', 6);
  ctx_ddl.create_preference  ('MINI_STOPLIST', 'minilist', 'GENERIC STOP LIST' );
end;
```

---

---

**Note:** This example illustrates a stoplist for a case-sensitive text index. If the stoplist is for a case-insensitive index, the stoplist requires only one entry for each stop word and the case of the entry has no effect.

---

---

---

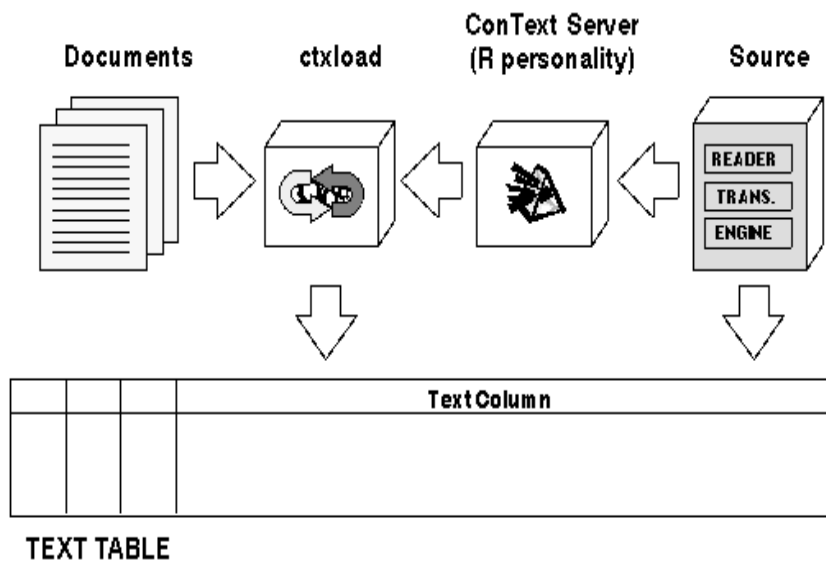
# Understanding the ConText Data Dictionary: Text Loading

This chapter describes the ConText data dictionary objects provided with ConText for text loading.

The topics discussed in this chapter are:

- Sources
- Preferences for Text Loading
- Reader Tiles
- Engine Tiles
- Translator Tiles

## Sources

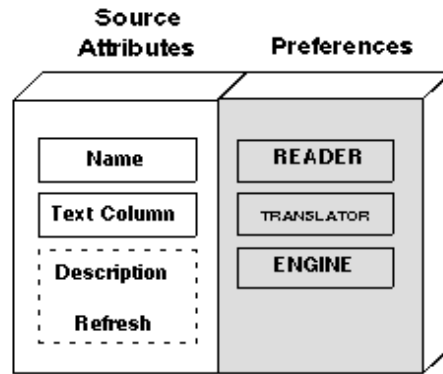


This section provides conceptual, as well as reference, information about text loading sources, which are stored in the ConText data dictionary:

- What is a Source?
- Source Attributes
- Preferences in Sources



## What is a Source?



A source is a logical grouping of three text loading preferences (one preference for each of the supported categories), assigned to a column in the database. A source specifies the options used by ConText to load text automatically into a column using `ctxload` and ConText servers with the Loader personality.

---

**Note:** A source must exist for a column before a ConText server with the Loader personality can load text from an operating system file into the column.

---

Sources can be created by any ConText user with the CTXAPP role. Sources are stored in the ConText data dictionary. In addition to the preferences for a source, users specify a name and text column for the source. Users can also specify a description and a refresh rate for directory scanning.

The sources created by a user must be unique for the user. As such, the same source for a user cannot be assigned to more than one column.

**See Also:** For an example of automated text loading with ConText servers, see “Loading Text” in “Setting Up and Managing Text”.

For more information about text loading preferences, see “Preferences for Text Loading” in this chapter.

## Source Attributes

To define a source, a user specifies a name and column for the source, and optionally a description and refresh rate for the source.

The column in the source indicates the column to which text is loaded by ConText servers.

---

---

**Note:** The column must be a LONG or LONG RAW column, because load servers only supports loading text columns with these datatypes.

---

---

## Preferences in Sources

To define a source, the user specifies a preference for each of the three supported categories.

A preference can be used in more than one source; however, two preferences from the same category cannot be used in the same source.

If a preference for one of the categories is not specified when the source is created, the default, predefined preference for the category is used in the source.

---

---

**Note:** All three of the loading categories have defaults; however, the default preference for the Reader category should not be used. This is because the directory specified in the default Reader preference is a generic directory specification and probably does not exist in your file system.

---

---

## Preferences for Text Loading

This section provides conceptual, as well as reference, information for text loading preferences, which are stored in the ConText data dictionary:

- What is a Text Loading Preference?
- Reader Predefined Preferences
- Engine Predefined Preferences
- Translator Predefined Preferences

### What is a Text Loading Preference?

Text loading preferences specify the options that ConText uses to load text (in batch mode). Each preference represents one (and only one) text loading option.

A preference consists of a ConText Tile and one or more attributes (and their corresponding values) for the Tile.

In addition, each preference is grouped into one of three types or categories, which determine the text loading operation that the preference controls.

When creating a source, three preferences are specified for the source, one for each of the three categories.

**See Also:** For more information about Tiles in sources, see “Reader Tiles”, “Engine Tiles”, or “Translator Tiles” in this chapter.

#### User-defined Preferences

A ConText user with the CTXAPP role can create their own preferences by setting the required attributes for one of the Tiles provided by ConText, then calling CTX\_DDL.CREATE\_PREFERENCE and specifying the name of the Tile.

---

---

**Note:** When creating a source, users can use all preferences that have been defined in the ConText data dictionary, including their own preferences, preferences created by other users, or the predefined preferences provided by ConText.

---

---

#### Predefined Preferences

ConText provides predefined preferences for each category. These predefined preferences can be used by any ConText user with the CTXAPP role to create sources without first creating preferences.

## Reader Predefined Preferences

ConText provides a single predefined Reader preference for text loading: `DEFAULT_READER`.

### **DEFAULT\_READER**

This preference calls the `DIRECTORY READER` Tile, which specifies a dummy directory for the Tile.

---

---

**Note:** Because it is unknown which directory contains the files to be loaded and path names are operating-system specific, this preference is provided as a default only and should *not* be used when creating a source.

Before creating a source, you should create your own Reader preference that specifies the directory where your files to be loaded are located.

---

---

## Engine Predefined Preferences

ConText provides a single predefined Engine preference, `DEFAULT_LOADER`, for text loading.

### **DEFAULT\_LOADER**

This preference calls the `GENERIC LOADER` Tile, which indicates the preference can be used to load text from files in a operating system directory.

## Translator Predefined Preferences

ConText provides a single predefined Translator preference, `DEFAULT_TRANSLATOR`, for text loading.

### **DEFAULT\_TRANSLATOR**

This preference calls the `NULL TRANSLATOR` Tile, which indicates no translation is performed on the files to be loaded; the files are in the format required by `ctxload`.

# Reader Tiles

The Reader Tiles are used to specify the location of the files to be loaded.

ConText provides the a single Reader Tile for creating preferences for text loading sources:

Tile	Attributes	Attribute Values
DIRECTORY READER	directories	<i>pathname for the directory where text loading files are located</i>

## DIRECTORY READER Tile

The DIRECTORY READER Tile is used to specify the location of files to be loaded when the files are located in the local operating system.

### Attributes

DIRECTORY READER has the following attribute(s):

#### directories

The *directories* attribute specifies the full pathname for the directory that the Con-Text server with the Loader personality scans when looking for new files to load into a column in a table or view.

The structure of the value for *directories* will vary depending on the directory naming conventions used by your operating system.

# Engine Tiles

ConText provides a single Engine Tile for creating preferences for text loading sources:

Tile	Attributes	Attribute Values
GENERIC LOADER	separate	Y (text stored in separate file(s), load file contains pointers to separate file(s))  N (text stored in load file, default)
	longsize	<i>maximum size, in kilobytes, of text to be loaded (default 64)</i>

## GENERIC LOADER Tile

The GENERIC LOADER Tile is used to specify the loader (ctxload) for loading text.

### Attributes

GENERIC LOADER has the following attribute(s).

#### separate

The *separate* attribute specifies whether the *-separate* option for ctxload is enabled. When the *-separate* option is enabled, the load files do not contain the actual text of the documents to be loaded, but, rather, contain pointers to separate files where the text of the documents is stored.

The default for *separate* is N.

#### longsize

The *longsize* attribute specifies a value for the *-longsize* option for ctxload. The *-longsize* option specifies the maximum size, in kilobytes, allowed for text loaded by ctxload.

**See Also:** For more information about how the *-separate* and *-longsize* options work in ctxload for loading text, see “Command-line Syntax” in Chapter 10, “Text Loading Utility”.

## Translator Tiles

ConText provides the following Translator Tiles for creating preferences for text loading sources:

Tile	Attributes	Attribute Values
NULL TRANSLATOR	** none **	N/A
USER TRANSLATOR	command	<i>executable for translator program</i>

### NULL TRANSLATOR Tile

The NULL TRANSLATOR Tile is used to specify that the load files for the loader (ctxload) are already in the format required by the loader. It has no attributes.

### USER TRANSLATOR Tile

The USER TRANSLATOR Tile is used to specify a translator program that converts load files into the format required by the loader (ctxload).

#### Attributes

USER TRANSLATOR has the following attribute(s):

#### command

The *command* attribute specifies the executable name of the translator program used to convert a load file into the format required by ctxload.

---

**Note:** The specified translator executable must be stored in the appropriate directory in the Oracle home directory.

For example, in a UNIX-based environment, all translator executables must be stored in \$ORACLE\_HOME/ctx/bin.

In a Windows NT environment, the translator executables must be stored in \BIN in the Oracle home directory.

For more information about the required location for executable files, see the Oracle8 installation documentation specific to your operating system.

---





---

## Setting Up and Managing Text

This chapter provides details on how to use the command-line to set up and maintain text in ConText.

The process of administering text in a ConText system comprises the following tasks:

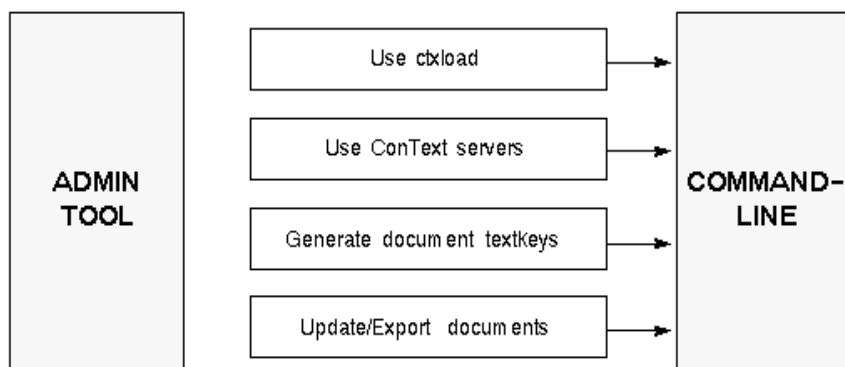
- Loading Text
- Managing Preferences
- Managing Policies
- Managing Indexes
- Managing Thesauri
- Managing Document Sections

---

**Note:** Most of the text setup and administration tasks can also be performed from the GUI administration tools (System Administration tool or Configuration Manager). These tasks are diagramed in each section of the chapter.

---

## Loading Text



This section provides instructions for loading text into database columns from the command-line:

- Using ctxload
- Using ConText Servers for Automated Text Loading
- Generating Document Textkeys
- Updating/Exporting a Document

---

**Note:** The ConText Workbench includes an input/output (I/O) utility for loading/updating text from client-side files in a Windows environment to database columns, as well as exporting text from database columns to client-side files.

For more information about using the ConText Workbench I/O utility, see *Oracle8 ConText Cartridge Application Developer's Guide*.

---

### Using ctxload

Use ctxload to load text from a load file or from separate text files into the database.

For example:

```
ctxload -user jsmith/welcome -name MY_DOCS -file docs.txt -log docload.log
```

In this example, the Oracle user's username/password is *jsmith/welcome*. Because the *-thes* argument for *ctxload* isn't specified, by default, *ctxload* loads text, rather than a thesaurus, into the specified database table. The table to which the documents are loaded is *my\_docs* and the load file being used is *docs.txt*.

In addition, this example generates a log file named *docload.log*.

**See Also:** For a complete description of *ctxload* requirements and options, as well as the structure and syntax of the text load file, see Chapter 10, "Text Loading Utility".

### Loading Text into External Data Store Columns

If you use the external data store (i.e. in your text column, you store pointers to documents in your file system), you can use *ctxload* to load the file pointers.

---

**Note:** The *ctxload* utility is best suited for loading text into columns that utilize the direct data store for storing text.

---

To use *ctxload* with external data store columns, the following conditions must exist:

- the column that you are using to store the file pointers is a LONG column
- each file pointer is embedded in the load file as the text of the document; however do *not* use the *-separate* option in the *ctxload* command-line.

For example:

```
<TEXTSTART: EMPNO=1010, ENAME='Mary Jones'>
mjones.pdf
<TEXTEND>
```

In this example, the file name *mjones.pdf* will be loaded into the LONG column for the table and the structured employee information, such as employee number (1010) and name (Mary Jones), will be loaded into the specified columns.

---

**Suggestion:** Because a LONG column is an inefficient means of storing file pointers, you probably do not want to use *ctxload* to load file pointers into columns that use the external data store. You may want to consider using *SQL\*Loader* to load the file pointers instead.

---

## Using ConText Servers for Automated Text Loading

ConText servers can be used to automatically load text from files in an operating system directory as the directory is populated with files.

ConText uses ConText servers with the Loader personality to scan a specified directory for files and call `ctxload` to load all existing files in the directory into a specified column. The column and the directory to be scanned are specified in a text loading source created by the user.

To setup ConText servers for automated text loading, perform the following tasks:

---

---

**Note:** This example assumes that ConText is installed in a UNIX-based environment and the files to be loaded are stored in local directories on the operating system.

---

---

1. Use the `SET_ATTRIBUTE` and `CREATE_PREFERENCE` procedures in `CTX_DDL` to create a Reader preference. The Reader preference identifies the directory where the files are (or will be) located.

For example:

```
begin
  ctx_ddl.set_attribute('DIRECTORIES', '/product/docs');
  ctx_ddl.create_preference('PRODUCT_DOCS_READER',
                           'Directory scanner for /private/docs',
                           'DIRECTORY READER');
end;
```

In this example, the name of the preference created is *reader\_pref*. The *directories* attribute for the DIRECTORY READER Tile specifies the directory path and name for the directory to be scanned (*/private/docs*).

2. Use `SET_ATTRIBUTE` and `CREATE_PREFERENCE` to (optionally) create a Translator preference. The Translator preference converts incoming files into the format required by `ctxload`.

For example:

```
begin
  ctx_ddl.set_attribute('COMMAND', '/bin/convert.sh');
  ctx_ddl.create_preference('PRODUCT_DOCS_TRANSLATOR',
                           'script that converts files to ctxload format',
                           'USER TRANSLATOR');
end;
```

In this example, the name of the preference created is *reader\_pref*. The *command* attribute for the USER TRANSLATOR Tile specifies the location and the name of the translation program (a shell script named *convert.sh*).

---

**Note:** If the incoming files do not need to be converted, you can skip this step and use the predefined preference, DEFAULT\_TRANSLATOR, in your text loading source.

---

3. Use SET\_ATTRIBUTE and CREATE\_PREFERENCE to (optionally) create a Loader Engine preference. The Loader Engine preference specifies values for the required parameters in ctxload.

For example:

```
begin
  ctx_ddl.set_attribute('separate', 'Y');
  ctx_ddl.set_attribute('longsize',2000);
  ctx_ddl.create_preference('PRODUCT_DOCS_LOADER',
                           'text in separate files, max size of text 2MB',
                           'GENERIC LOADER');
end;
```

In this example, the *separate* attribute is set for the GENERIC LOADER Tile, which indicates that the text to be loaded by ctxload is stored in separate files and not in the load file. In addition, a value of 2000 Kilobytes (2 Megabytes) is specified for *longsize*, which sets the maximum size of the text to be loaded by ctxload.

---

**Note:** If the text to be loaded is stored directly in the load file and the amount of text for any given document is less than 64 Kilobytes, you can skip this step and use the predefined preference, DEFAULT\_LOADER, in your text loading source.

---

4. Use the `CREATE_SOURCE` procedure in `CTX_DDL` to create a source for the column into which you want to load text.

When you create a source, you specify the name of the source and the column to be loaded. You also specify the Reader, Translator, and Engine preferences that you created.

For example:

```
begin
  ctx_ddl.create_source('DOCS_SOURCE','DOCS.TEXT',
    'basic source for documents in /product/docs',
    reader_pref => 'PRODUCT_DOCS_READER',
    translator_pref => 'PRODUCT_DOCS_TRANSLATOR',
    engine_pref => 'PRODUCT_DOCS_LOADER');
end;
```

In this example, the source name is *docs\_source* and the column to be loaded is *text* in a table named *docs*.

**See Also:** For more information about sources and automated text loading, see “Sources” in Chapter 8, “Understanding the ConText Data Dictionary: Text Loading”.

5. Start a ConText server with the Loader (R) personality.

For example:

```
ctxsrv -user ctxsys/passwd -personality R &
```

**See Also:** For a complete description of `ctxsrv`, see “`ctxsrv` Executable” in Chapter 4, “ConText Server Executables and Utilities”.

## Generating Document Textkeys

Each document loaded into a table must be assigned a value in the primary key column of the table. This value serves as the textkey for the document.

Textkeys can be assigned using the following methods:

- Embedding Textkeys in Load File
- Generating Textkeys Using Sequences and Triggers

## Embedding Textkeys in Load File

To manually embed textkey values for documents in the load file, in each document header, create an entry which specifies the name of the primary key column in the table and the textkey value to be assigned to the document.

For example:

```
. . .
<TEXTSTART: PK=1000, TITLE='DOC 1000'>
doc1000.txt
<TEXTEND>
<TEXTSTART: PK=1001, TITLE='DOC 10001'>
doc1001.txt
<TEXTEND>
. . .
```

In this example, the load file contains pointers to separate text files (*doc1000.txt* and *doc1001.txt*), rather than the text for each document. The primary key column for the table is *pk* and the values specified are loaded into *pk* when *ctxload* is run.

**See Also:** For a complete description of the structure of the load file, see Chapter , “Structure of Text Load File” in Chapter 10, “Text Loading Utility”.

## Generating Textkeys Using Sequences and Triggers

To automatically generate textkey values for each document loaded into a table, use the SQL command *CREATE* to create a trigger and sequence for the table.

The sequence generates unique values for each document. The trigger calls the sequence each time a row (document) is loaded into the table and stores the value in the primary key column for the table.

For example:

```
create sequence doc_seq;
create trigger doc_trigger
before insert on DOCS
for each row
BEGIN
    select docs_seq.nextval into :new.pk from dual;
END;
```

In this example, a sequence named *doc\_seq* and a trigger named *doc\_trigger* are created for a table named *docs*, in which the primary key column is *pk*.

*doc\_trigger* specifies that the next available value generated by *doc\_seq* is inserted into the *docs* table before each new row is inserted into the table.

**See Also:** For more information about creating sequences and triggers, see *Oracle8 Server SQL Reference*.

## Updating/Exporting a Document

This section provides details for using *ctxload* to update the text column for an existing document (row in the table) from an operating-system file or to export the contents of the text column for a row to an operating-system file.

To use *ctxload* to update/export a document, the document must already exist as a row in the table. To create the row, you can use *ctxload* or other text loading methods supported by Oracle.

If you are using the ConText Workbench, you can use the I/O Utility, which provides the same functionality, except it uses client-side files to perform the update/export.

**See Also:** For a complete description of *ctxload*, see Chapter 10, “Text Loading Utility”.

For a description of the I/O Utility, see *Oracle ConText Workbench User's Guide*.

### Update a Document

To update an existing document, you can use *ctxload* in update mode. To specify update mode for *ctxload*, use the *-update* option and specify the name of the policy for the text column, the primary key of the row to be updated, and the file containing the updated text.

For example:

```
ctxload -user ctxdemo/passwd -update -name word_docs -pk 3452 -file /docs/resume1.doc
```

In this UNIX-based example, the row identified by primary key *3452*, in the table for a policy named *word\_docs*, is updated with the contents of *resume1.doc* located in */docs*.



## Export a Document

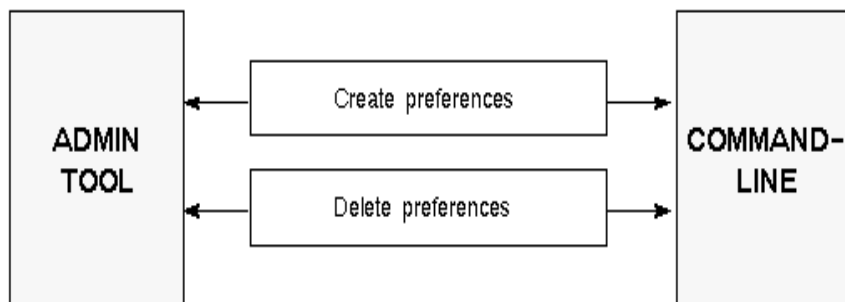
To export a document, you can use `ctxload` in export mode. To specify update mode for `ctxload`, use the *-export* option and specify the name of the policy for the text column, the primary key of the row to be exported, and the file to which the text is exported.

For example:

```
ctxload -user ctxdemo/passwd -export -name word_docs -pk 3452 -file /docs/new.doc
```

In this UNIX-based example, the contents of the text column for the row identified by primary key *3452*, in the table for a policy named *word\_docs*, are written to an operating system file named *new.doc* located in */docs*.

## Managing Preferences



This section provides details for using the CTX\_DDL PL/SQL package to perform the following administration tasks for preferences:

- Creating a Preference
- Deleting a Preference

In addition, this section describes the following tasks for specific types of preferences:

- Creating an Engine Preference
- Creating a Data Store Preference for a Master Table
- Creating Filter Preferences
- Creating a Theme Lexer Preference
- Creating a Stoplist Preference

---

**Note:** When creating preferences, do *not* use PL/SQL and SQL reserved words as the names of your preferences.

In addition, certain words, such as *ascii*, *html*, *blaster*, and *filter* are used internally by ConText and, consequently, should *not* be used by themselves as preference names; however, they can be combined with other words to create descriptive names, such as *html\_filter\_pref*.

---

## Creating a Preference

To create a preference in the ConText data dictionary, use the SET\_ATTRIBUTE and CREATE\_PREFERENCE procedures in CTX\_DDL.

For example:

```
begin
  ctx_ddl.set_attribute ('PATH',
                        '/public/doc1:/public/doc2');
  ctx_ddl.create_preference ('PUB_DOCS',
                           'Docs stored in files',
                           'OSFILE');
end;
```

---

---

**Note:** CREATE\_PREFERENCE must be called immediately after SET\_ATTRIBUTE to assign the specified attribute(s) to the preference that you are creating.

---

---

In this example, a Datastore preference named *pub\_docs* is created for text stored externally in operating system files in a UNIX-based environment.

The *path* attribute for the OSFILE Tile specifies the directory paths and names (*/pub/doc1* and */public/doc2*) where the files are stored. A colon is used to separate the multiple directory paths/names.

### Specifying Multiple Values for Attributes

To assign more than one value to the same Tile attribute, you must call SET\_ATTRIBUTE separately for each value that you want to set before calling CREATE\_PREFERENCE.

Currently, only the *stop\_word* (GENERIC STOP LIST Tile), *executable* (BLASTER FILTER Tile), and *keep\_tag* (GENERIC WORD LIST Tile) attributes support multiple values.

**See Also:** For examples of specifying multiple attribute values for preferences, see “Creating Filter Preferences” and “Creating a Stoplist Preference” in this chapter.

### Clearing Attributes from the Buffer

Each time CREATE\_PREFERENCE is called, the buffer used to store the attributes for preferences is automatically cleared. As a result, if the preference creation failed, all of the attributes must be entered again before calling CREATE\_PREFERENCE again.

If you enter an incorrect value for an attribute, you can override the attribute value by simply calling SET\_ATTRIBUTE again for the same attribute. If you need to remove all of the attributes from the buffer, use the CTX\_DDL.CLEAR\_ATTRIBUTES procedure.

## Creating an Engine Preference

One of the most important preferences you create is an Engine preference. In the Engine preference for a policy, you specify the amount of indexing memory allocated for the column in the policy, as well as the STORAGE clauses used for the automatically-generated tables and Oracle indexes that comprise a ConText index.

Because ConText index strings for indexed tokens are stored in memory before they are saved to the ConText index tables, it is vital that you allocate as much indexing memory as possible to avoid excessive index fragmentation.

When you create an Engine preference, you use the *index\_memory* attribute for the GENERIC ENGINE Tile to allocate indexing memory.

If you plan to use parallel indexing, the memory specified for the Engine preference should be the amount of real memory available divided evenly among the number of ConText servers that will perform the indexing in parallel.

For example, if you are going to use three ConText servers in parallel to create an index for a column and you have 100 Mb of memory available on the machine on which the servers will be running, you should create an Engine preference with *index\_memory* set to 33 Mb, then specify the preference in the policy for the column.

---

---

**Suggestion:** To ensure the best results for indexing, calculate the total amount of *real* memory (*not* virtual memory) available on the machine which will be used to create the index, then specify this amount when you create an Engine preference.

---

---

**See Also:** For an example of creating a policy, see “Creating a Column Policy” in this chapter.

## Creating a Data Store Preference for a Master Table

The following example illustrates creating a Data Store preference for two tables with the following master/detail relationship:

Table	Columns	Datatype	Description
DOCS	DOCID	NUMBER	Primary key for <i>docs_text.fk_docid</i>
	TITLE	VARCHAR2	Document title
	COMMENT	VARCHAR2	Text column
DOCS_TEXT	FK_DOCID	NUMBER	Foreign key to <i>docs.docid</i>
	CHAPTER	NUMBER	Detail information (combined with <i>docs.docid</i> uniquely identifies rows in <i>docs_text</i> )
	TEXT	VARCHAR2	Actual column containing text of documents

To create a Data Store preference for use in a policy on the master table, use the MASTER DETAIL NEW Tile.

For example:

```
exec ctx_ddl.set_attribute('BINARY','0');
exec ctx_ddl.set_attribute('DETAIL_TABLE','DOCS_TEXT');
exec ctx_ddl.set_attribute('DETAIL_KEY','FK_DOCID');
exec ctx_ddl.set_attribute('DETAIL_LINENO','CHAPTER');
exec ctx_ddl.set_attribute('DETAIL_TEXT','TEXT');
exec ctx_ddl.create_preference('MDN','','MASTER DETAIL NEW');
exec ctx_ddl.create_policy('DOCS_POL','DOCS.COMMENT',textkey=>'DOCID',dstore_pref=>'MDN');
```

In this example, the text column for the policy is the *comment* column in *master*; however, ConText does not index the contents of this column. The column simply serves as the place-holder for the policy.

As such, any column in the master table, except for the *textkey* column, can serve as the text column for the policy; however, the DML trigger for the table always includes the text column. Any changes to the text column result in a reindexing request sent to the DML queue.

You may wish to create a dummy column in your master table for use as the text column so that changes to the column do not trigger reindexing requests. In the example above, every time *comments* changes, reindexing is performed on the *text* column for each row in the *docs\_text* table that is associated with *comments*.

In addition, the dummy column, if named something appropriate (e.g. *text* or *detail*), makes one-step queries more intuitive to write.

For example:

```
alter table master add (text char(1));
exec ctx_ddl.create_policy('MY_POL','master.text',textkey => 'PK' dstore_pref=>'MY_MD')

select title
from master
where contains(text, 'Oracle')> 0;
```

## Creating Filter Preferences

When creating Filter preferences, the following considerations determine which Tiles and attributes you use, as well as the values that you specify for each attribute:

- internal filters or external filters
- single-format or mixed-format columns

**See Also:** For a complete list of the Tiles and attributes for Filter preferences, see the Filter section in “Filter Tiles” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

### Creating a Filter Preference Using Internal Filters

This section provides one example for internal filters in single-format columns and one example for mixed-format columns.

**Single-Format Columns:** For a single-format column using one of the internal filters, create a Filter preference that sets the *format* attribute (BLASTER FILTER Tile) to the format used in your column.

The following example illustrates creating a Filter preference for a column that contains documents only in MS Word for Windows format:

```
begin
  ctx_ddl.set_attribute('FORMAT','11')
  ctx_ddl.create_preference('WP6_FILT',
                           'WP6 filter',
                           'BLASTER FILTER');
end;
```

**Mixed-Format Columns:** For mixed-format columns using internal filters, create a Filter preference that sets the *format* attribute (BLASTER FILTER Tile) for the Autorecognize filter:

```
begin
  ctx_ddl.set_attribute('FORMAT','997')
  ctx_ddl.create_preference('MULTI_FILT',
                           'multiple internal filters',
                           'BLASTER FILTER');
end;
```

### Creating a Filter Preference Using External Filters

This section provides one example for external filters in single-format columns and one example for mixed-format columns.

---

---

**Note:** Before a Filter preference that uses external filters can be created, one or more external filters (executables) must be created and stored in the appropriate directory in your Oracle home directory.

You can choose to create your own external filters or use the external filters executables provided by ConText. The examples in this section use the external filters provided by ConText.

For a complete list of the external filters provided by ConText, see “Supplied External Filters” in Chapter 6, “Text Concepts”.

For the location of the directory for the external filter executables, see the Oracle8 installation documentation specific to your operating system.

---

---

**Single-Format Columns:** For a single-format column that uses external filters, create a Filter preference that uses the *command* attribute (USER FILTER Tile) to specify the filter executable for the format used in your column.

The following example illustrates creating a Filter preference for a column that contains documents in AmiPro format and uses the supplied external filter named *ami-pro*:

```
begin
  ctx_ddl.set_attribute('COMMAND', 'amipro')
  ctx_ddl.create_preference('AMIPRO_FILT',
                          'amipro external filter',
                          'USER FILTER');
end;
```



**Mixed-Format Columns:** For a mixed-format column that uses external filters only or external and internal filters, create a Filter preference that sets the *executable* attribute (BLASTER FILTER Tile) once for each of the external filters you want to use in your column.

---

**Note:** The *executable* attribute requires that you specify a format ID which identifies the document format supported by the filter executable.

For a complete list of format IDs for document formats, see “Supported External Filter Formats for Mixed-Format Columns” in Chapter 6, “Text Concepts”.

---

The following example illustrates creating a Filter preference for a column that contains documents in AmiPro, PDF (Adobe Acrobat), and WordPerfect 6.0 formats. It uses the supplied external filters *amipro* and *acropdf*, because these formats are not supported by the internal filters:

```
begin
  ctx_ddl.set_attribute('EXECUTABLE', 19, 'amipro', 1)
  ctx_ddl.set_attribute('EXECUTABLE', 57, 'acropdf', 2)
  ctx_ddl.create_preference('MULT_FILT',
                           'multiple filters, some external',
                           'BLASTER FILTER');
end;
```

---

**Note:** It is not necessary to explicitly specify the external filter for WordPerfect 6.0, because ConText provides an internal filter for WordPerfect 6.0.

When a Filter preference is created using the *executable* attribute, ConText always uses internal filters, unless an external filter is explicitly specified in the preference.

---

## Creating a Theme Lexer Preference

If you have English-language documents in a column and you want to create a theme index for the column to enable theme queries, you need to first create a Lexer preference that calls the theme lexer, then include the preference in the policy definition for the column.

To create your own theme lexer preference, call `CTX_DDL.CREATE_PREFERENCE` and specify the `THEME LEXER` Tile.

For example:

```
begin
  ctx_ddl.create_preference ('MY_THEME_PREF',
                           'Pref for theme indexes',
                           'THEME LEXER');
end;
```

---

---

**Note:** Because `THEME LEXER` does not require any attributes to be set, you do not generally need to create Theme Lexer preferences; instead, you can use the `THEME_LEXER` predefined preference provided by ConText.

---

---

**See Also:** For examples of creating a policy that uses the theme lexer preference, see “Creating a Theme Indexing Policy” in this chapter.

## Creating a Stoplist Preference

A Stoplist preference is created by calling CTX\_DDL.SET\_ATTRIBUTE separately for each stop word in the list.

To define a Stoplist:

1. Set the *stop\_word* attribute (GENERIC STOP LIST Tile) for each word that you do not want ConText to index. In addition, for each call to SET\_ATTRIBUTE for *stop\_word*, specify a sequence from 1 to 4095. The sequence is used in the text index to record the stop words that proceed and follow each indexed term. This enables queries for phrases that contain stop words.

---

**Note:** The maximum number of terms (stop words) that a Stoplist preference can contain is 4095.

---

2. Call CTX\_DDL.CREATE\_PREFERENCE and specify the GENERIC STOP LIST Tile to create the preference.

For example:

```
begin
  ctx_ddl.set_attribute('STOP_WORD', 'OF', 1);
  ctx_ddl.set_attribute('STOP_WORD', 'TO', 2);
  ctx_ddl.set_attribute('STOP_WORD', 'A', 3);
  . . .
  . . .
  . . .
  ctx_ddl.set_attribute('STOP_WORD', 'SO', 82);
  ctx_ddl.set_attribute('STOP_WORD', 'MOST', 83);
  ctx_ddl.set_attribute('STOP_WORD', 'MAY', 85);
  ctx_ddl.set_attribute('STOP_WORD', 'INTO', 86);
  ctx_ddl.set_attribute('STOP_WORD', 'ANY', 87);
  ctx_ddl.create_preference('MY_STOPLIST',
                          'My list of stop words',
                          'GENERIC STOP LIST');
end;
```

---

**Note:** This example illustrates a stoplist for a case-insensitive text index. To create a stoplist for a case-sensitive index, specify the stop words in the stoplist exactly as they would appear in the text of your documents (e.g. *Of, of, To, to*)

---

## Deleting a Preference

To delete a preference from the ConText data dictionary, use the PL/SQL procedure `CTX_DDL.DROP_PREFERENCE`.

For example:

```
exec ctx_ddl.drop_preference('PUB_DOCS')
```

To use `DROP_PREFERENCE`, you need to specify only the name (in this example, *pub\_docs*) of the preference that you want to drop.

---

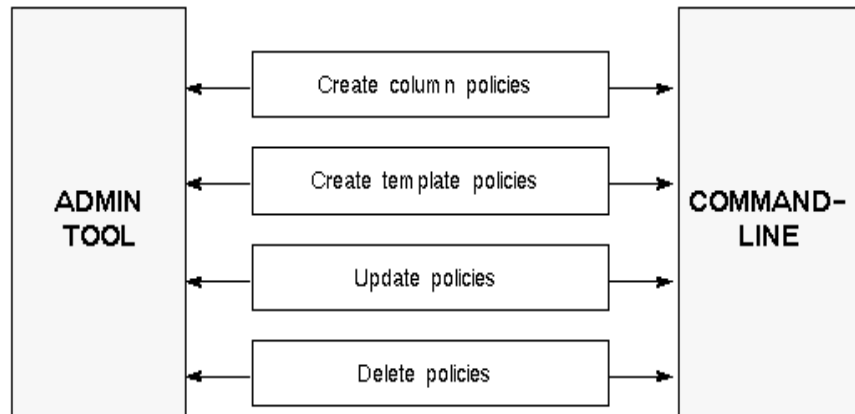
---

**Note:** If a preference is used in a policy, the policy must be deleted from the ConText data dictionary before the preference can be deleted.

---

---

## Managing Policies



This section provides details for using the CTX\_DDL PL/SQL package to perform the following policy administration tasks:

- Creating a Column Policy
- Creating a Template Policy
- Modifying a Policy
- Deleting a Policy

In addition, this section describes the following tasks for specific types of policies:

- Creating a Theme Indexing Policy
- Using Composite Textkeys in a Policy

---

**Note:** When creating policies, do *not* use PL/SQL and SQL reserved words as the names of your policies.

In addition, certain words, such as *ascii*, *html*, *blaster*, and *filter* are used internally by ConText and, consequently, should *not* be used by themselves as policy names; however, they can be combined with other words to create descriptive names, such as *ascii\_indexing\_policy*.

---

## Creating a Column Policy

To create a column policy for text indexing, use the PL/SQL procedure `CTX_DDL.CREATE_POLICY`.

For example:

```
begin
  ctx_ddl.create_policy (policy_name      => 'DOC_POL',
                        colspec          => 'DOCS.ARTICLES',
                        textkey          => 'PK',
                        dstore_pref      => 'PUB_DOCS',
                        engine_pref      => 'DOC_ENGINE',
                        filter_pref      => 'WORD6',
                        lexer_pref       => 'DOC_LINK',
                        wordlist_pref    => 'CTXSYS.SOUNDEX',
                        stoplist_pref    => 'MINI_STOP_LIST');
end;
```

In this example, the name of the policy is *doc\_pol*. The policy does not have a description, nor does it use a source policy. The text column (specified by *colspec*) is *articles* in a table named *docs*. The textkey for the table is *pk*.

The preferences used in the policy are all user-owned policies, except for the Wordlist preference, which uses the predefined preference named `SOUNDEX`.

### Usage Notes for *colspec* and *textkey*

The following conditions apply to the *colspec* and *textkey* parameters in `CREATE_POLICY`:

- the column name in *colspec* must include the table name, using the following syntax: *table\_name.column\_name*
- *textkey* and *colspec* must be distinct; a textkey column cannot also serve as the text column for the policy
- if a value is not specified for *textkey*, ConText does not, by default, always select the primary key column for the table identified in *colspec*. Instead, ConText selects the first primary key or unique column encountered in the table.

### Compressor Preferences

It is not necessary to specify a Compressor preference when creating a policy. ConText uses the predefined `NULL COMPRESSOR` preference as the default.

## Preferences and Policies in Other Schemas

In a policy, you can use preferences owned by other users; however, you must specify the fully qualified name of the preference. For example, to specify a preference owned by CTXSYS, such as the SOUNDEX preference, use the following syntax: `CTXSYS.pref_name` (e.g. `CTXSYS.soundex`).

In addition, if you use a source policy in a policy, you can specify either your template policies or the CTXSYS-owned template policies; however, you must specify the fully-qualified name of the template policy.

## Creating a Theme Indexing Policy

To create a theme indexing policy, use the PL/SQL procedure `CTX_DDL.CREATE_POLICY` and specify either your own theme lexer preference or the predefined preference (`THEME_LEXER`) provided by ConText.

The following example illustrates how to create a policy identical to the previous policy example, except that *my\_theme\_pref*, the theme lexer preference created in the theme lexer example, is used in place of *doc\_link*, which is a preference that calls the basic (indexing) lexer:

```
begin
  ctx_ddl.create_policy (policy_name => 'DOC_POL',
                        colspec      => 'DOCS.ARTICLE',
                        textkey      => 'PK',
                        dstore_pref  => 'PUB_DOCS',
                        engine_pref  => 'DOC_ENGINE',
                        filter_pref  => 'WORD6',
                        lexer_pref   => 'MY_THEME_PREF',
                        wordlist_pref => 'CTXSYS.SOUNDEX',
                        stoplist_pref => 'MINI_STOP_LIST');
end;
```

When index creation is requested for this theme indexing policy, the theme lexer generates a theme index that can be used to perform theme queries.

**See Also:** For an example of creating a theme lexer preference, see “Creating a Theme Lexer Preference” in this chapter.

For more information about theme indexes and queries, see “Theme Indexes” in Chapter 6, “Text Concepts”.

## Using Composite Textkeys in a Policy

To create a policy that uses a composite textkey, use the PL/SQL procedure `CTX_DDL.CREATE_POLICY`; however, when you specify the textkey for the column, reference each of the primary or unique key columns (up to 16) that constitute the composite textkey for the column.

For example:

```
begin
  ctx_ddl.create_policy (policy_name => 'DOC_POL',
                        colspec      => 'DOCS.ARTICLE',
                        textkey      => 'AUTH,TITLE',
                        dstore_pref  => 'PUB_DOCS',
                        engine_pref  => 'DOC_ENGINE',
                        filter_pref  => 'WORD6',
                        lexer_pref   => 'DOC_LINK',
                        wordlist_pref => 'CTXSYS.SOUNDEX',
                        stoplist_pref => 'MINI_STOP_LIST');
end;
```

In this example, the textkey for the *articles* column is a composite textkey consisting of the columns *auth* and *title* in the *docs* table. The names of the textkey columns are separated by commas and are registered in the ConText data dictionary in the order in which they are specified.

---

---

**Note:** There is a 256 character limit, including the comma separators, on the combined length of the column names in a composite textkey.

Also, there is a 256 character limit on the combined length of the columns in a composite textkey.

For more information about these limits, see “Composite Textkeys” in Chapter 6, “Text Concepts”.

---

---



## Creating a Template Policy

To create a template policy, use the PL/SQL procedure `CTX_DDL.CREATE_TEMPLATE_POLICY`.

For example:

```
begin
  ctx_ddl.create_template_policy (policy_name => 'TEMPLATE_POL',
                                dstore_pref  => 'PUB_DOCS',
                                engine_pref  => 'DOC_ENGINE',
                                filter_pref   => 'WORD6',
                                lexer_pref    => 'DOC_LINK',
                                wordlist_pref => 'SOUNDEX_YES',
                                stoplist_pref => 'MINI_STOP_LIST');
end;
```

In this example, the name of the policy is *template\_pol*. The preferences for the policy are as specified above. If *template\_pol* is specified as a source policy when creating a new policy, its preferences are copied to the new policy.

---

---

**Note:** You can also use `CTX_DDL.CREATE_POLICY` to create a template policy. When you call `CREATE_POLICY`, do *not* specify a value for *colspec*.

---

---

## Modifying a Policy

To modify a policy, use the PL/SQL procedure `CTX_DDL.UPDATE_POLICY`.

---

---

**Note:** If a policy has been used to create a index for the text column in the policy, the index must be dropped before the policy can be updated.

In addition, you cannot modify the attributes for a policy; you can only modify the description and preferences for a policy.

---

---

For example:

```
begin
  ctx_ddl.update_policy (policy_name => 'DOC_POL',
                        filter_pref  => 'HTML_DOC',
                        wordlist_pref => 'CTXSYS.NO_SOUNDEX');
end;
```

In this example, a Filter preference named *html\_doc* replaces the existing preference for the Filter category, while the predefined Wordlist preference named NO\_SOUNDEX replaces the existing preference for the Wordlist category.

## Deleting a Policy

To delete a policy from the ConText data dictionary, use the PL/SQL procedure CTX\_DDL.DROP\_POLICY.

For example:

```
execute ctx_ddl.drop_policy ('DOC_POL')
```

To use DROP\_POLICY, you only need to specify the name (in this example, *doc\_pol*) of the policy that you want to drop.

---

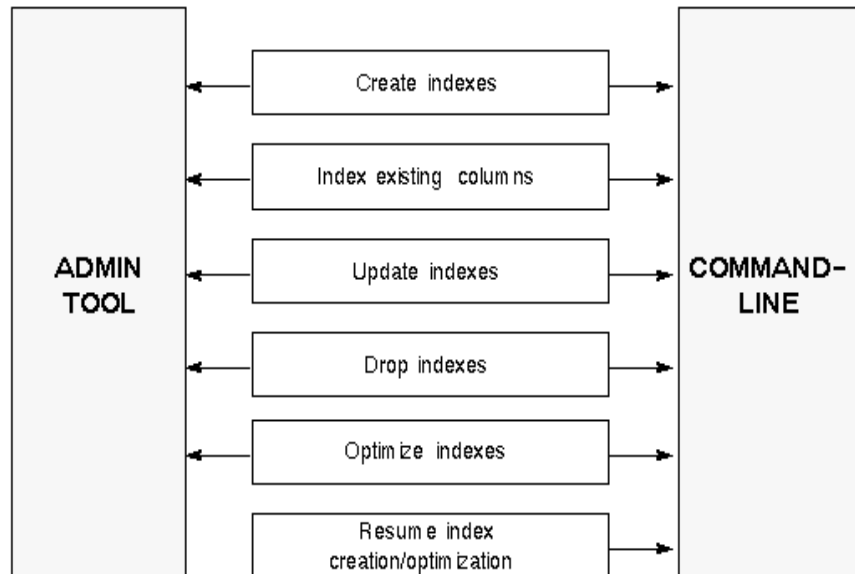
---

**Note:** If a policy has been used to create a index for the text column in the policy, the index must be dropped before the policy can be deleted.

---

---

## Managing Indexes



This section provides details for using the CTX\_DDL PL/SQL package to perform the following indexing tasks:

- Creating an Index and ConText Indexing in Parallel
- Indexing Existing Columns (Hot Upgrade)
- Updating an Index
- Dropping an Index
- Optimizing an Index
- Resuming Index Creation/Optimization

## Creating an Index

To create a ConText index (theme or text), use the CTX\_DDL.CREATE\_INDEX procedure.

The only argument required for CREATE\_INDEX is the name of the policy for the text column to be indexed.

For example:

```
execute ctx_ddl.create_index('DOC_POL')
```

In this example, CREATE\_INDEX creates an index for the text column defined in a policy named *doc\_pol*.

---

---

**Note:** During indexing, ConText creates Oracle indexes for the index tables using the temporary tablespace for CTXSYS. To ensure successful creation of the Oracle indexes, the temporary tablespace for CTXSYS must have enough space to store the temporary segments used in creating the Oracle indexes.

The temporary tablespace for CTXSYS is defined during installation of ConText.

For more information about defining the temporary tablespace for CTXSYS, see the Oracle8 installation documentation specific to your operating system.

---

---

### Creating a Non-populated Index

To create the ConText index tables without populating the tables, use the *pop\_index* parameter in CTX\_DDL.CREATE\_INDEX.

For example:

```
execute ctx_ddl.create_index('DOC_POL', pop_index => FALSE)
```

This example creates the ConText index tables for the *doc\_pol* policy without populating the tables with index entries.

To populate the tables, the CTX\_DML.REINDEX procedure can be called for each of the rows (documents) in the table for *doc\_pol* or, if automatic DML is enabled, update each of the rows in the table.

## ConText Indexing in Parallel

You can optionally include a numeric value in the argument string for CTX\_DDL.CREATE\_INDEX to specify the number of ConText servers used for parallel indexing.

For example:

```
execute ctx_ddl.create_index('DOC_POL', 4)
```

In this example, CREATE\_INDEX uses the first four available ConText servers with the DDL personality to create an index in parallel for the text column defined in the *doc\_pol* policy.

---

---

**Note:** The value you specify for parallel creation of ConText indexes cannot exceed the number of ConText servers currently running with the DDL personality. If you specify more ConText servers than the number of servers running, CREATE\_INDEX will not execute.

---

---

**See Also:** For more information about ConText indexing in parallel, see “Parallel Indexing” in Chapter 6, “Text Concepts”.

### Parallel Creation of Oracle Indexes

ConText indexing in parallel does not automatically cause the Oracle indexes on the ConText index tables to be created in parallel.

To have Oracle8 create Oracle indexes in parallel, the parallel query option for Oracle8 must be installed. In addition, a value must be specified for the PARALLEL clause used in the CREATE INDEX command.

To specify a value for the PARALLEL clause used in the CREATE INDEX command for the Oracle index created on the token table in the ConText index, use the *ili\_other\_params* attribute (GENERIC ENGINE Tile) in the Engine preference for the column policy.

To set the PARALLEL clause for the Oracle indexes created on the other tables in the ConText index, use the *kid\_other\_params*, *kik\_other\_params*, *lix\_other\_params*, and *sri\_other\_params* attributes.

For example:

```
begin
  ctx_ddl.set_attribute('I1I_OTHER_PARAMS', ' PARALLEL 4');
  ctx_ddl.set_attribute('KID_OTHER_PARAMS', ' PARALLEL 4');
  ctx_ddl.set_attribute('KIK_OTHER_PARAMS', ' PARALLEL 4');
  ctx_ddl.create_preference('PAR_INDEX',
                           'Parallel indexing x 4',
                           'GENERIC ENGINE');
end;
```

In this example, an Engine preference named *par\_index* is created with a PARALLEL value of 4 for the Oracle indexes created on the token and document mapping tables in ConText indexes.

If the *par\_index* preference is used in a column policy, when a ConText index is created for the policy, four Oracle8 server processes create the indexes in parallel for the token and document mapping tables.

---

---

**Note:** If you do not set the *other\_params* attributes for the indexes on a particular ConText index table, the value for PARALLEL is derived from the PARALLEL value specified for the CREATE TABLE command used to create the ConText index table.

If no PARALLEL value is specified for the ConText index table, the default is 1.

---

---

**See Also:** For more information about the PARALLEL clause in the CREATE INDEX and CREATE TABLE commands, see *Oracle8 Server SQL Reference*.

For more information about the parallel query option, see *Oracle8 Server Tuning*.

## Indexing Existing Columns (Hot Upgrade)

ConText does not require you to create new tables or modify existing tables to create indexes for text already stored in a database. If you already have text stored in a column in an existing database, you can use ConText to index the text in the column without changing the structure of the table itself. Once the column has an index, queries can be submitted against the column.

The only requirements are:

- the table must have a primary key or unique column that can serve as a textkey column for identifying the documents stored in the table
- if the text in the column is formatted, it must be in a format supported by ConText
- if the text in the column is stored in mixed formats, the policy for the column must include a preference that uses the Autorecognize filter

The procedure for indexing an existing text column is identical to the procedure for indexing a new text column:

1. Create preferences (optional)
2. Create a policy for the column
3. Create an index using the policy for the column

**See Also:** For examples of creating preferences and policies, see “Creating a Preference” and “Creating a Column Policy” in this chapter.

For an example of creating a ConText index, see “Creating an Index” in this chapter.

## Updating an Index

Once an index is created for a text column, ConText automatically updates the index each time a document (row) is added, deleted, or modified in the table.

In addition, the index can be manually updated for a single document using `CTX_DML.REINDEX`.

## Dropping an Index

To drop an existing index from the data dictionary, use the PL/SQL procedure `CTX_DDL.DROP_INDEX`.

For example:

```
execute ctx_ddl.drop_index ('DOC_POL')
```

In this example, the index and associated tables for *doc\_pol* are deleted from the database. If you wanted to perform subsequent text queries against the text column for *doc\_pol*, the index for the column in *doc\_pol* must be recreated using `CTX_DDL.CREATE_INDEX`.

## Optimizing an Index

Index optimization can be used to help reduce the size of ConText indexes, as well as update the indexes to reflect deleted/modified documents.

To optimize an index in the data dictionary, use the PL/SQL procedure, `CTX_DDL.OPTIMIZE_INDEX`.

For example:

```
execute ctx_ddl.optimize_index('DOC_POL', ctx_ddl.defragment_to_new_table);
```

In this example, the optimization method used for the ConText index for *doc\_pol* is *defragment\_to\_new\_table*. This method uses a second, mirror ConText index table to compact the index fragments for all indexed terms with multiple fragments and remove references from the index strings for all deleted/modified documents.

**See Also:** For more information about ConText index optimization, see “Index Optimization” in Chapter 6, “Text Concepts”

### Parallel Optimization

Similar to index creation, index optimization can be performed in parallel. To perform parallel index optimization, specify a degree of parallelism when calling the `OPTIMIZE_INDEX` procedure.

For example:

```
begin
  ctx_ddl.optimize_index(policy_name => 'DOC_POL',
                        optyp        => ctx_ddl.defragment_to_new_table,
                        parallel     => 4);
end;
```



In this example, `OPTIMIZE_INDEX` is called for *doc\_pol* with an optimization method of *defragment\_to\_new\_table* and degree of parallelism of 4.

---

---

**Note:** The parallel issues for Oracle index creation on ConText index tables apply to ConText index optimization as well.

For more information about the issues regarding parallel index creation, see “Parallel Creation of Oracle Indexes” in this chapter.

---

---

## Resuming Index Creation/Optimization

If an index operation (creation or optimization) fails, you can use the PL/SQL procedure `CTX_DDL.RESUME_FAILED_INDEX` to resume the operation once the reason for the failure has been determined and corrected/removed.

For example:

```
execute ctx_ddl.resume_failed_index('DOC_POL')
```

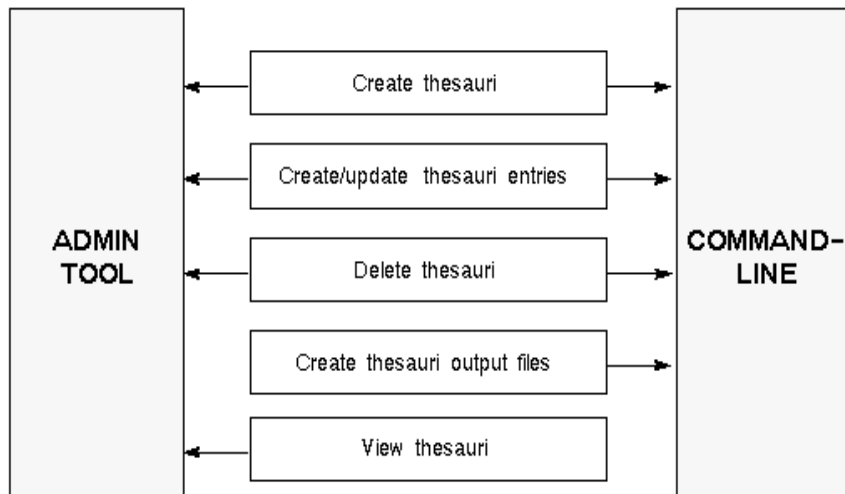
In this example, text DDL operation is the default (index creation) and is resumed for the text column for the *doc\_pol* policy.

You can also choose to start the index operation over from the beginning using `CTX_DDL.CREATE_INDEX` or `CTX_DDL.OPTIMIZE_INDEX`.

You can view the index log in the GUI administration tools or through the `CTX_INDEX_LOG` view to determine when and where the index operation failed.

The log also can be used to determine whether to resume the operation or simply start the operation over, based on the stage at which the operation failed and/or the percentage of the operation completed before failure.

## Managing Thesauri



This section provides details for using the CTX\_THES PL/SQL package and/or ctx-load to perform the following indexing tasks:

- Creating a Thesaurus and Creating a Case-sensitive Thesaurus
- Creating/Updating a Thesaurus Entry
- Deleting a Thesaurus
- Managing Document Sections

Thesauri viewing is not covered in this manual because viewing thesauri terms and relationships is best handled through a graphical interface, such as the System Administration tool, which is provided with the ConText Workbench.

## Creating a Thesaurus

To create a thesauri, use the PL/SQL function `CTX_THES.CREATE_THESAURUS` or use the `ctxload` command-line utility.

---

**Note:** `CREATE_THESAURUS` creates a thesaurus with no entries.

`ctxload` creates a thesaurus using a thesaurus import file. The file can contain thesaurus entries or can be empty.

To add entries to a thesaurus, you must use `CTX_THES.CREATE_PHRASE`.

---

### Using `CREATE_THESAURUS`

The following SQL\*Plus example creates an empty thesaurus named *tech\_thes* using `CREATE_THESAURUS`:

```
variable thesid number
execute :thesid := ctx_ddl.create_thesaurus('tech_thes')
```

### Using `ctxload`

The following command-line example creates a thesaurus named *science\_thes* using `ctxload`:

```
ctxload -user ctxdev/passwd -thes -name science_thes -file sci_terms.txt
```

In this example, the owner of the thesaurus is an Oracle user named *ctxdev*. The *-thes* argument specifies that `ctxload` is used to create/import a thesaurus. The name of the thesaurus import file is *sci\_terms.txt*.

**See Also:** For a complete description of `ctxload` requirements and options, as well as the structure and syntax of the thesaurus import file, see Chapter 10, “Text Loading Utility”.

## Creating a Case-sensitive Thesaurus

To create a case-sensitive thesaurus that contains no entries, use `CTX_THES.CREATE_THESAURUS` and specify `TRUE` for *case\_sensitive*.

To create a case-sensitive thesaurus with entries, use `ctxload` and the *-thescase* argument.

### Using CREATE\_THESAURUS

The following example creates an empty, case-sensitive thesaurus named *science\_terms*. ConText retains the case of all terms that are subsequently entered in the thesaurus:

```
variable thesoid number
execute :thesoid := ctx_ddl.create_thesaurus('science_terms',TRUE)
```

### Using ctxload

The following UNIX-based example creates a case-sensitive thesaurus named *science\_terms* and populates the thesaurus with entries from a file named *science.thes*:

```
ctxload -thes -thescase y -name science_terms -file science.thes
```

## Creating/Updating a Thesaurus Entry

To create a entry in an existing thesaurus or update an existing entry, use the PL/SQL function CTX\_THES.CREATE\_PHRASE. The only update allowed for an existing entry is the definition of a new relationship between the phrase in the entry and another phrase in the thesaurus.

---

---

**Suggestion:** Because the relationships between terms in a thesaurus entry can be complex, updating entries using CREATE\_PHRASE is not recommended.

If possible, use the System Administration tool to update entries. The System Administration tool provides a graphical representation of thesaurus entries and relationships.

---

---

The following SQL\*Plus example creates two new phrases (*intranet* and *world wide web*) in a thesaurus named *tech\_thes*:

```
variable phraseid number
execute :phraseid := ctx_ddl.create_phrase('tech_thes','intranet')
execute :phraseid := ctx_ddl.create_phrase('tech_thes','world wide web')
```

The following SQL\*Plus example establishes the phrase *intranet* as a narrower partitive term for *world wide web* in *tech\_thes*:

```
variable phraseid number
execute :phraseid := ctx_ddl.create_phrase('tech_thes','intranet','NTP','world wide web')
```

## Deleting a Thesaurus

To delete an existing thesaurus, use the PL/SQL procedure `CTX_THES.DROP_THESAURUS`.

For example:

```
execute ctx_ddl.drop_thesaurus('science_thes')
```

In this example, a thesaurus named *science\_thes* and all of its entries are deleted from the thesaurus tables.

## Creating a Thesaurus Output File

To create an output file containing all the entries for an existing thesaurus, use the `ctxload` command-line utility.

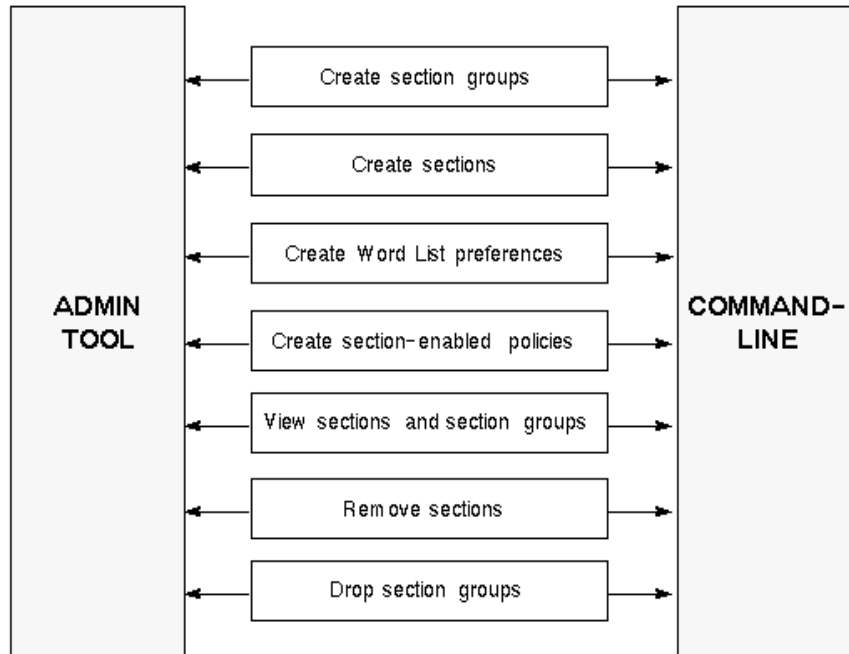
For example:

```
ctxload -user ctxdev/passwd -thesdump -name tech_thes -file tech_terms.out
```

In this example, the owner of the thesaurus is an Oracle user named *ctxdev*. The *-thesdump* argument specifies that `ctxload` is used to create/export a thesaurus output file. The name of the thesaurus import file is *tech\_terms.out* and it is created in the directory from which `ctxload` is run.

**See Also:** For a complete description of `ctxload` requirements and options, as well as the structure and syntax of the thesaurus import file, see Chapter 10, “Text Loading Utility”.

## Managing Document Sections



This section provides details for creating sections and section groups and assigning a section group to a text column via the policy for the column:

- Creating a Section Group
- Creating a Section
- Creating a Wordlist Preference with a Section Group
- Creating a Policy for a Section Group
- Viewing Sections and Section Groups
- Removing a Section from a Section Group
- Dropping a Section Group

## Creating a Section Group

To create a section group, use the `CTX_DDL.CREATE_SECTION_GROUP` procedure. The only argument required for `CREATE_SECTION_GROUP` is the name of the section group to be created.

For example:

```
exec ctx_ddl.create_section_group('HTML_SECTIONS')
```

## Creating a Section

To create a section and assign the section to a section group, use the `CTX_DDL.ADD_SECTION` procedure. The `ADD_SECTION` procedure requires you to enter a name for the section, the name of the section group to which the section is assigned, start and end tags for the section, and whether the section is a top-level section or self-enclosing.

For example:

```
exec ctx_ddl.add_section('HTML_SECTIONS','HEAD','<HEAD>','</HEAD>',true,false)
```

In this example, the name of the section is *head*, the start and end tags for the section are `<HEAD>` and `</HEAD>`, and the section is defined as a top-level section, meaning the section ends when an end tag for the section or a start tag for another top-level section is encountered.

---

**Note:** For the strings `<HEAD>` and `</HEAD>` to be treated as start and end tags in HTML documents, both strings must be specified for the *keep\_tag* attribute (HTML FILTER Tile) and the *startjoins* and *endjoins* attributes (BASIC LEXER Tile) must be set.

For examples of setting these attributes, see “Filter Examples” and “Lexer Examples” in Chapter 7, “Understanding the ConText Data Dictionary: Indexing”.

---

## Creating a Wordlist Preference with a Section Group

To create a Wordlist preference for sections in a text column, set the *section\_group* attribute (GENERIC WORD LIST Tile), then use CTX\_DDL.CREATE\_PREFERENCE to create a preference for the Tile.

For example:

```
exec ctx_ddl.set_attribute('SECTION_GROUP','HTML_SECTIONS');  
exec ctx_ddl.create_preference('html_sect','HTML sections','GENERIC WORD LIST');
```

## Creating a Policy for a Section Group

To use the preference in a policy, use CTX\_DDL.CREATE\_POLICY and specify the name of the preference.

For example:

```
ctx_ddl.create_policy('html_pol','docs.text',wordlist_pref=>'html_sect');
```

## Viewing Sections and Section Groups

To view all the sections and section groups that have been created in the ConText data dictionary, use the CTX\_ALL\_SECTIONS and CTX\_ALL\_SECTION\_GROUPS views.

To view only the sections and section groups that you have created, use the CTX\_USER\_SECTIONS and CTX\_USER\_SECTION\_GROUPS views.

## Removing a Section from a Section Group

To remove a section from a section group, use the CTX\_DDL.REMOVE\_SECTION procedure and specify the name of the section group to which the section belongs and the name of the section to remove.

For example:

```
exec ctx_ddl.remove_section('headers','heading1')
```

---

---

**Note:** A section can only be removed from a section group if the section group is not currently used in any existing preferences.

---

---

To remove all the sections from a section group, you must call REMOVE\_SECTION for each section in the group. You can also drop the section group, which automatically removes all sections defined for the group.



## Dropping a Section Group

To drop a section group (and all of its sections) from the ConText data dictionary, use the CTX\_DDL.DROP\_SECTION\_GROUP procedure and specify the name of the section group to drop.

For example:

```
exec ctx_ddl.remove_section('headers','heading1')
```

---

---

**Note:** A section group can only be dropped if it is not currently used in any existing preferences.

---

---



---

## Text Loading Utility

This chapter provides reference information for using the text loading utility, `ctxload`, provided with ConText.

The topics discussed in this chapter are:

- Overview of `ctxload`
- Command-line Syntax
- Command-line Examples
- Structure of Text Load File
- Structure of Thesaurus Import File

## Overview of ctxload

The ctxload utility can be used to perform the following operations:

- Text Loading
- Document Updating/Exporting
- Thesaurus Importing and Exporting

### Text Loading

Use ctxload to load text from a load file into a LONG or LONG RAW column in a table.

---

---

**Suggestion:** If the target table does not contain a LONG or LONG RAW column or you do not want to load text into a LONG or LONG RAW column, you may want to use SQL\*Loader to populate the table with text.

---

---

A load file is an ASCII flat file that contains the plain text, as well as any structured data (title, author, date, etc.), for documents to be stored in a text table; however, in place of the text for each document, the load file can store a pointer to a separate file that holds the actual text (formatted or plain) of the document.

---

---

**Note:** The ctxload utility does not support load files that contain both embedded text and file pointers. You must use one method or the other when creating a load file.

---

---

The ctxload utility creates one row in the table for each document identified by a header in the load file.

**See Also:** For examples of load files for text loading, see “Structure of Text Load File” in this chapter.

## Document Updating/Exporting

The ctxload utility supports updating database columns from operating system files and exporting database columns to files, specifically LONG RAW and LONG columns used as text columns for ConText.

---

---

**Note:** The updating/exporting of data is performed in sections to avoid the necessity of a large amount of memory (up to 2 Gigabytes) for the update/fetch buffer.

As a result, a minimum of 16 Kilobytes of memory is required for document update/export.

---

---

## Thesaurus Importing and Exporting

Use ctxload to load a thesaurus from an import file into the ConText thesaurus tables.

An import file is an ASCII flat file that contains entries for synonyms, broader terms, narrower terms, or related terms which can be used to expand queries.

ctxload can also be used to export a thesaurus by dumping the contents of the thesaurus into a user-specified operating-system file.

**See Also:** For examples of import files for thesaurus importing, see “Structure of Thesaurus Import File” in this chapter.

## Command-line Syntax

The syntax for running ctxload is:

```
ctxload -user username[/password][@sqlnet_address]
        -name object_name
        -file file_name
        -pk primary_key
        [-export]
        [-update]
        [-thes]
        [-thescase y|n]
        [-thesdump]
        [-separate]
        [-longsize n]
        [-date date_mask]
        [-log file_name]
        [-trace]
        [-commitafter n]
        [-verbose]
```

### Mandatory Arguments

#### **-user**

Specifies the username and password of the user running ctxload.

The username and password can be followed immediately by *@sqlnet\_address* to permit logon to remote databases. The value for *sqlnet\_address* is a database connect string. If the TWO\_TASK environment variable is set to a remote database, you do not have to specify a value for *sqlnet\_address* to connect to the database.

#### **-name**

If ctxload is used to load text, *-name* specifies the name of the table to be loaded. The table must be accessible to the user specified in the command-line.

If ctxload is used to update/export a text column, *-name* specifies the policy for the column to be exported/updated.

If ctxload is used to import a thesaurus, *-name* specifies the name of the thesaurus to be imported. The thesaurus name is used to specify the thesaurus to be used for expanding query terms in queries.

---

---

**Note:** Thesaurus name must be unique. If the name specified for the thesaurus is identical to an existing thesaurus, ctxload returns an error and does not overwrite the existing thesaurus.

---

---

If ctxload is used to export a thesaurus, *-name* specifies the name of the thesaurus to be exported.

**-file**

If ctxload is used to load text, *-file* specifies the name of the load file which contains the document header markers, structured data, and text/file pointers (see the *-separate* argument).

If ctxload is used to update a row in a text column, *-file* specifies the file which stores the text to be inserted into the text column for the row specified by *-pk*.

If ctxload is used to export a row in a text column, *-file* specifies the file which stores the text to be exported from the text column for the row specified by *-pk*.

If ctxload is used to load a thesaurus, *-file* specifies the name of the import file which contains the thesaurus entries.

If ctxload is used to export a thesaurus, *-file* specifies the name of the export file created by ctxload.

---

---

**Note:** If the name specified for the thesaurus dump file is identical to an existing file, ctxload *overwrites* the existing file.

---

---

**-pk**

Specifies the primary key for the row in which the text column (LONG or LONG RAW) to be exported/updated is located.

---

---

**Note:** A value is required for *-pk* only when ctxload is used to update/export the contents of a text column for a row.

---

---

For tables that contain composite primary keys, enter the multiple primary key values as a string, with each primary key value separated by a comma.

---

---

**Note:** For composite textkeys, the string must be entered in the same order in which the primary key columns were defined as textkeys for the policy.

If the primary key value(s) contain blank spaces, the entire value for `-pk` must be enclosed in double quotation marks (" ").

For example:

```
...-pk "3452,Joe Smith,500 Oracle Parkway"...
```

If the primary key values contain commas (,) or backslashes (\), each comma/backslash must be preceded by a backslash.

For example:

```
...-pk "3452,Smith\, Joe"...
```

In this example, the second value 'Smith, Joe' contains a blank space, so the entire primary key value is enclosed in double quotes.

---

---

## Optional Arguments

### **-export**

Specifies that `ctxload` exports the contents of a cell in a database table into the operating system file specified by `-file`. The cell is identified as the LONG RAW or LONG column for the row specified by `-pk` in the table for the policy specified by `-name`.

---

---

**Note:** If the file specified by `-file` already exists, `ctxload` *overwrites* the contents of the file with the contents of the LONG/LONG RAW column.

---

---

### **-update**

Specifies that `ctxload` updates the contents of a cell in a database table with the contents of the operating system file specified by `-file`. The cell is identified as the LONG RAW or LONG column for the row specified by `-pk` in the table for the policy specified by `-name`.



**-thes**

Specifies that ctxload imports a thesaurus. The file from which it loads the thesaurus is specified by the *-file* argument. The name of the thesaurus to be imported is specified by the *-name* argument.

**-thescase**

Specifies whether ctxload create a case-sensitive thesaurus with the name specified by *-name* and populate the thesaurus with entries from the thesaurus import file specified by *-file*. If *-thescase* is 'y' (the thesaurus is case-sensitive), ConText enters the terms in the thesaurus exactly as they appear in the import file.

The default for *-thescase* is 'n' (case-insensitive thesaurus)

---

---

**Note:** *-thescase* is only valid for use with the *-thes* argument.

---

---

**-thesdump**

Specifies that ctxload exports a thesaurus. The name of the thesaurus to be exported is specified by the *-name* argument. The file into which the thesaurus is dumped is specified by the *-file* argument.

**-separate**

For text loading, specifies that the text of each document in the load file is actually a pointer to a separate text file. During processing, ctxload loads the contents of each text file into the LONG or LONG RAW column for the specified row.

**-longsize**

For text loading, specifies the maximum number of kilobytes to be loaded into the LONG or LONG RAW column. This argument may be necessary for loading separate data and to help reduce memory usage when loading smaller embedded data.

The minimum value is 1 (Kb, i.e. 1024 bytes) and the maximum value is machine-dependent. The default is 64 (Kb).

---

---

**Note:** The value for *-longsize* must be entered as a numeric value. Do *not* include a 'K' or 'k' to indicate Kilobytes.

---

---

**-date**

Specifies the TO\_CHAR date format for any date columns loaded using ctxload.

**See Also:** For more information about the available date format models, see *Oracle8 Server SQL Reference*.

**-log**

Specifies the name of the log file to which ctxload writes any national-language supported (NLS) messages generated during processing. If you do not specify a log file name, the messages appear on the standard output.

**-trace**

Specifies that a server process trace file is enabled using 'ALTER SESSION SET SQL\_TRACE TRUE'. This command captures all processed SQL statements in a trace file, which can be used for debugging purposes. The location of the trace file is operating-system dependent and may be modified using the USER\_DUMP\_DEST initialization parameter.

**-commिताfter**

Specifies the number of rows (documents) that are inserted into the table before a commit is issued to the database. The default is 1.

**-verbose**

Specifies that non-NLS messages can appear on standard output.

## Usage Notes

The following conditions apply to the command-line syntax:

- if you do not specify -thes or -thesdump, by default ctxload loads text into the specified table.
- for text loading, you do not need to specify a column name because ctxload automatically loads text to the LONG or LONG RAW column in a table and a table can contain only one such column
- if you use embedded text instead of separate file pointers in the text load file, *do not* use the *-separate* option
- loading text from separate files (using the *-separate* option) is faster, in general, than loading text embedded in the load file

## Command-line Examples

This section provides examples for each of the operations that `ctxload` can perform:

- Text Load Example
- Document Update Example
- Document Export Examples
- Thesaurus Import Example
- Thesaurus Export Example

### Text Load Example

The following example loads documents from the *reviews.txt* load file into table *docs* for user *jsmith*. It also writes log information to a file called *log2.out*. Because *-commitafter* was not specified, each row (document) is committed to the database after it is inserted into the *docs* table.

Also, because *-separate* was not specified, `ctxload` expects the text for each document to be embedded in the *reviews.txt* file.

```
ctxload -user jsmith/123abc -name docs -file review.txt -log log2.out
```

### Document Update Example

The following UNIX-based example illustrates updating the LONG RAW column for the row identified by primary key *3452* in the table for a policy named *word\_docs*. The column is updated with the contents of *resume1.doc* located in */docs*:

```
ctxload -user ctxdemo/passwd -update -name word_docs -pk 3452 -file /docs/resume1.doc
```

## Document Export Examples

The following UNIX-based example illustrates exporting the LONG RAW column for the row identified by primary key *3452* in the table for a policy named *word\_docs*. The contents of the cell in the column are copied to a file named *new.doc* located in */docs*:

```
ctxload -user ctxdemo/passwd -export -name word_docs -pk 3452 -file /docs/new.doc
```

The following example is identical to the preceding example, except the row is identified by a compound primary key consisting of a name and location. The name and location values are separate by a comma and the entire primary key string is enclosed in double quotation marks because the location value includes a space:

```
ctxload -user ctxdemo/passwd -export -name word_docs -pk "Smith,HQ 1" -file /docs/new.doc
```

## Thesaurus Import Example

The following example imports a thesaurus named *tech\_doc* from an import file named *tech\_thesaurus.txt*:

```
ctxload -user jsmith/123abc -thes -name tech_doc -file tech_thesaurus.txt
```

## Thesaurus Export Example

The following example dumps the contents of a thesaurus named *tech\_doc* into a file named *tech\_thesaurus.out*:

```
ctxload -user jsmith/123abc -thesdump -name tech_doc -file tech_thesaurus.out
```

## Structure of Text Load File

The load file must use the following format for each document, as well as any structured data associated with the document:

```
<TEXTSTART: col_name1=doc_data, col_name2=doc_data,...col_nameN=doc_data>
text. . .
<TEXTEND>
```

where:

**<TEXTSTART: ... >**

is a header marker that indicates the beginning of a document. It also may contain one or more of the following fields used to specify structured data for a document:

***col\_name***

is the name of a column that will store structured data for the document.

***doc\_data***

is the structured data that will be stored in the column specified in *col\_name*.

***text***

is the text of the document to be loaded or the name (and location, if necessary) of an operating system file containing the text to be loaded.

---

---

**Note:** The data in *text* (either a string of text or a file name pointer) is treated by ctxload as literal data, including any non-alphanumeric characters or blank spaces that may occur. As a result, you must ensure that *text* exactly represents the data you wish ctxload to process.

For example, if you use ctxload to load text from separate files, the file names in the load file must exactly represent the name(s) of the operating-system file(s) containing the text. If any blank spaces are included in a file name, ctxload cannot locate the file and the text is not loaded.

---

---

**<TEXTEND>**

indicates the end of the document.

## Load File Structure

The following conditions apply to the structure of the load file:

- for each document to be loaded, either the text of the document or a pointer to a separate file must be in the load file.
- embedded text and separate file pointers cannot be used together in the same load file
- if the text for your documents is embedded in the load file, the text must be in ASCII format
- if pointers to separate files are used, the text in the files can be in plain (ASCII) format or a proprietary format (e.g. MS Word)
- if the text in a separate file is in a proprietary format, the format must be supported by ConText and it must be loaded into a LONG RAW column
- each separate file must contain a single document (the contents of a separate file are stored as a single row in the table)

## Load File Syntax

The following conditions apply to the syntax utilized in the text load file:

- <TEXTSTART: ... > and <TEXTEND> *must* each start on a new line
- the structured data parameters within the <TEXTSTART: ... > string do not have to be in any particular order
- a newline character (either hard or soft return) cannot occur between a *col\_name* and the beginning of its associated *doc\_data*

---

---

**Note:** The entire value for *doc\_data* does not have to be on the same line as the *col\_name*; only the beginning of the value and the *col\_name* must share the same line.

---

---

- the first *col\_name* should be on the same line as the 'TEXTSTART:'
- the '>' character which indicates the end of the <TEXTSTART: ... > string must be on the same line as the last *doc\_data* field for the document
- structured and LONG data may span more than one line
- single quote-marks must be escaped in *doc\_data* (e.g. *don't* must be entered as *don"t*)

- each <TEXTSTART: ... > string *must* be followed by the text of a document or a pointer to a separate file
- the text or file pointer must be placed after the complete <TEXTSTART: ... > string and *should* start on a new line
- the *last* character in the load file *should* be a newline character

## Example of Embedded Text in Load File

The following example illustrates a correctly formatted text load file containing structured employee information, such as employee number (1000, 1024) and name (Joe Smith, Mary Jones), and the text for each document:

```
<TEXTSTART: EMPNO=1000, ELNAME='Smith', EFNAME='Joe'>
Joe has an interesting resume, includes...cliff-diving.
<TEXTEND>
<TEXTSTART: EMPNO=1024, EFNAME='Mary', ELNAME='Jones'>
Mary has many excellent skills, including...technical,
marketing, and organizational. Team player.
<TEXTEND>
```

## Example of File Name Pointers in Load File

The following example illustrates a correctly formatted text load file containing structured employee information, such as employee number (1000, 1024) and name (Joe Smith, Mary Jones), and a file name pointer for each document.

```
<TEXTSTART: EMPNO=1024, EFNAME='Mary', ELNAME='Jones'>
mjones.doc
<TEXTEND>
<TEXTSTART: EMPNO=1000, EFNAME='Joe', EFNAME='Smith'>
jsmith.doc
<TEXTEND>
```

---

---

**Note:** To use the load file in this example, you would have to specify the *-separate* argument when executing ctxload.

---

---

## Structure of Thesaurus Import File

The import file must use the following format for entries in the thesaurus:

```
phrase
BT broader_term
NT narrower_term1
NT narrower_term2
. . .
NT narrower_termN

BTG broader_term
NTG narrower_term1
NTG narrower_term2
. . .
NTG narrower_termN

BTP broader_term
NTP narrower_term1
NTP narrower_term2
. . .
NTP narrower_termN

BTI broader_term
NTI narrower_term1
NTI narrower_term2
. . .
NTI narrower_termN

SYN synonym1
SYN synonym2
. . .
SYN synonymN
USE|SEE synonym1

RT related_term1
RT related_term2
. . .
RN related_termN

SN text
```

where:



***phrase***

is a word or phrase that is defined as having synonyms, broader terms, narrower terms, and/or related terms.

---

**Note:** In compliance with ISO-2788 standards, a TT marker can be placed before a phrase to indicate that the phrase is the top term in a hierarchy; however, the TT marker is not required. In fact, ctx-load ignores TT markers during import.

In ConText, a top term is identified as any phrase that does not have a broader term (BT, BTG, BTP, or BTI).

---

**BT, BTG, BTP, BTI**

are the markers that indicate *broader\_termN* is a broader (generic | partitive | instance) term for *phrase*.

**NT, NTG, NTP, NTI**

are the markers that indicate *narrower\_termN* is a narrower (generic | partitive | instance) term for *phrase*.

If *phrase* does not have a broader (generic | partitive | instance) term, but has one or more narrower (generic | partitive | instance) terms, *phrase* is created as a top term in the respective hierarchy (in a ConText thesaurus, the BT/NT, BTG/NTG, BTP/NTP, and BTI/NIT hierarchies are separate structures).

**SYN**

is a marker that indicates *phrase* and *synonymN* are synonyms within a synonym ring.

---

**Note:** Synonym rings are not defined explicitly in ConText thesauri. They are created by the transitive nature of synonyms.

---

**USE | SEE**

are markers that indicate *phrase* and *synonymN* are synonyms within a synonym ring (similar to SYN); however, USE | SEE also indicates *synonymN* is the preferred term for the synonym ring. Either marker can be used to define the preferred term for a synonym ring.

**RT**

is the marker that indicates *related\_termN* is a related term for *phrase*.

**SN**

is the marker that indicates the following *text* is a scope note (i.e. comment) for the preceding entry.

***broader\_termN***

is a word or phrase that conceptually provides a more general description or category for *phrase*. For example, the word *elephant* could have a broader term of *land mammal*.

***narrower\_termN***

is a word or phrase that conceptually provides a more specific description for *phrase*. For example, the word *elephant* could have a narrower terms of *indian elephant* and *african elephant*.

***synonymN***

is a word or phrase that has the same meaning for *phrase*. For example, the word *elephant* could have a synonym of *pachyderm*.

***related\_termN***

is a word or phrase that has a meaning related to, but not necessarily synonymous with *phrase*. For example, the word *elephant* could have a related term of *wooly mammoth*.

---

---

**Note:** Related terms are not transitive. If a phrase has two or more related terms, the terms are related only to the parent phrase and not to each other.

---

---

## Alternate Hierarchy Structure

In compliance with thesauri standards, the load file supports formatting hierarchies (BT/NT, BTG/NTG, BTP, NTP, BTI/NTI) by indenting the terms under the top term and using NT (or NTG, NTP, NTI) markers that indicate the level for the term:

*phrase*

```
NT1 narrower_term1
  NT2 narrower_term1.1
  NT2 narrower_term1.2
    NT3 narrower_term1.2.1
    NT3 narrower_term1.2.2
NT1 narrower_term2
. . .
NT1 narrower_termN
```

Using this method, the entire branch for a top term can be represented hierarchically in the load file.

## Import File Structure for Terms

The following conditions apply to the structure of the entries in the import file:

- each entry (*phrase*, BT, NT, or SYN) must be on a single line followed by a new-line character
- entries can consist of a single word or phrases
- the maximum length of an entry (*phrase*, BT, NT, or SYN) is 255 characters, not including the BT, NT, and SYN markers or the newline characters
- entries cannot contain parentheses or plus signs.
- each line of the file that does not start with the BT, NT, and SYN markers indicates a *phrase*
- a *phrase* can occur more than once in the file
- each *phrase* can have one or more narrower term entries (NT, NTG, NTP), broader term entries (BT, BTG, BTP), synonym entries, and related term entries
- each broader term, narrower term, synonym, and preferred term entry must start with the appropriate marker and the markers must be in capital letters
- the broader terms, narrower terms, and synonyms for a *phrase* can be in any order
- holographs must be followed by parenthetical disambiguators everywhere they are used

For example: cranes (birds), cranes (lifting equipment)

- compound terms are signified by a plus sign between each factor (e.g. buildings + construction)
- compound terms are allowed only as synonyms or preferred terms for other terms -- never as terms by themselves, or in hierarchical relations.
- terms can be followed by a scope note (SN), total maximum length of 2000 characters, on subsequent lines

- multi-line scope notes are allowed, but require an SN marker on each line of the note

**Example of Incorrect SN usage:**

```
VIEW CAMERAS
SN Cameras with through-the lens focusing and a
range of movements of the lens plane relative to
the film plane
```

**Example of Correct SN usage:**

```
VIEW CAMERAS
SN Cameras with through-the lens focusing and a
SN range of movements of the lens plane relative
SN to the film plane
```

- Multi-word terms cannot start with reserved words (e.g. *use* is a reserved word, so *use other door* is not an allowed term; however, *use* is an allowed term)

## Import File Structure for Relationships

The following conditions apply to the relationships defined for the entries in the import file:

- related term entries must follow a phrase or another related term entry
- related term entries start with the RT marker, followed by white space, then the related term on the same line
- multiple related terms require multiple RT markers

**Example of incorrect RT usage:**

```
MOVING PICTURE CAMERAS
NT CINE CAMERAS
TELEVISION CAMERAS
```

**Example of correct RT usage:**

```
MOVING PICTURE CAMERAS
RT CINE CAMERAS
RT TELEVISION CAMERAS
```

- Terms are allowed to have multiple broader terms, narrower terms, and related terms

## Examples of Import Files

This section provides three examples of correctly formatted thesaurus import files.

### Example 1 (Flat Structure)

```
cat
SYN feline
NT domestic cat
NT wild cat
BT mammal
mammal
BT animal
domestic cat
NT Persian cat
NT Siamese cat
wild cat
NT tiger
tiger
NT Bengal tiger
dog
BT mammal
NT domestic dog
NT wild dog
SYN canine
domestic dog
NT German Shepard
wild dog
NT Dingo
```

### Example 2 (Hierarchical)

```
animal
  NT1 mammal
    NT2 cat
      NT3 domestic cat
        NT4 Persian cat
        NT4 Siamese cat
      NT3 wild cat
        NT4 tiger
          NT5 Bengal tiger
    NT2 dog
      NT3 domestic dog
        NT4 German Shepard
      NT3 wild dog
        NT4 Dingo
cat
SYN feline
dog
SYN canine
```

### Example 3

35MM CAMERAS

BT MINIATURE CAMERAS

CAMERAS

BT OPTICAL EQUIPMENT

NT MOVING PICTURE CAMERAS

NT STEREO CAMERAS

LAND CAMERAS

USE VIEW CAMERAS

VIEW CAMERAS

SN Cameras with through-the lens focusing and a range of  
SN movements of the lens plane relative to the film plane

UF LAND CAMERAS

BT STILL CAMERAS

---

## PL/SQL Packages - Text Management

This chapter provides reference information for using the PL/SQL packages provided with ConText to manage text.

The topics covered in this chapter are:

- CTX\_DDL: Text Setup and Management
- CTX\_DML: ConText Index Update
- CTX\_THES: Thesaurus Management

## CTX\_DDL: Text Setup and Management

The CTX\_DDL PL/SQL package is used to create preferences and policies for ConText and to perform DDL actions such as index creation and optimization.

CTX\_DDL contains the following stored procedures and functions:

Name	DESCRIPTION
ADD_SECTION	Creates a section and assigns the section to the specified section group
CLEAR_ATTRIBUTES	Clears the buffer for any attributes that have been set
CREATE_INDEX	Creates an index for the text column using the specified policy
CREATE_POLICY	Creates a policy in the ConText data dictionary
CREATE_PREFERENCE	Creates a preference in the ConText data dictionary
CREATE_SECTION_GROUP	Creates a section group in the ConText data dictionary
CREATE_SOURCE	Creates a text loading source in the ConText data dictionary
CREATE_TEMPLATE_POLICY	Creates a policy that has no text column defined
DROP_INDEX	Deletes the ConText index for the specified policy
DROP_INTRIG	Deletes the DML trigger for the specified table
DROP_POLICY	Deletes a policy from the ConText data dictionary
DROP_PREFERENCE	Deletes a preference from the ConText data dictionary
DROP_SECTION_GROUP	Deletes a section group from the ConText data dictionary
DROP_SOURCE	Deletes a text loading source from the ConText data dictionary
OPTIMIZE_INDEX	Combines index fragments into complete strings and updates index strings for deleted documents
REMOVE_SECTION	Deletes a section from a section group
RESUME_FAILED_INDEX	Resumes creation/optimization of a failed ConText index
SET_ATTRIBUTE	Specifies the Tile attribute and corresponding value for a preference
UPGRADE_INDEX	Converts ConText indexes from Release 2.0 or earlier to the current release



Name	DESCRIPTION
UPDATE_POLICY	Changes the description and/or the preferences in a policy
UPDATE_SOURCE	Changes the description and/or the preferences in a source

---

## ADD\_SECTION

The ADD\_SECTION procedure creates a section and adds the section to an existing section group.

### Syntax

```
CTX_DDL.ADD_SECTION(group_name    IN VARCHAR2,  
                    section_name  IN VARCHAR2,  
                    start_tag     IN VARCHAR2,  
                    end_tag       IN VARCHAR2,  
                    top_level     IN BOOLEAN DEFAULT FALSE,  
                    enclose_self  IN BOOLEAN DEFAULT FALSE);
```

**group\_name**

Specify the name of the section group to which ConText adds the section.

**section\_name**

Specify the name of the section ConText adds to the section group.

**start\_tag**

Specify the token, including any characters that appear at the beginning or end of the token, which marks the start of a section. For example: <HTML>

**end\_tag**

specify the token, including any characters that appear at the beginning or end of the token, which marks the end of a section. For example: </HTML>

**top\_level**

Specify that the section implicitly closes non-top-level sections and is implicitly closed by the start of other top-level sections.

**enclose\_self**

Specify that the section can enclose itself. If this parameter is not set, the section is implicitly closed when the next start tag is encountered.

If *enclose\_self* is TRUE, the end of the section is identified by either:

1. the end tag for the section
2. the end of the document

If *enclose\_self* is FALSE, the end of the section is identified by either:

1. the end tag (if any) for the section
2. the next start or end tag encountered (if *top\_level* is FALSE)
3. the end of the document

**See Also:** For more information about top-level sections and self-enclosing sections, see “Sections” in Chapter 6, “Text Concepts”.

## Examples

Examples are provided for four different types of sections you can create.

### Example 1: Non-enclosed, repeating sections

Title: Guide to Oracle  
 Author: Joseph Smith  
 Review: Very well written  
 Review: Interesting and exciting

Section Name	Start Tag	End Tag	Top Level	Enclose Self
TITLE	Title:		Y	N
AUTHOR	Author:		Y	N
REVIEW	Review:		Y	N

```
exec ctx_ddl.add_section('doc_section','title','Title:', top_level=>TRUE)
exec ctx_ddl.add_section('doc_section','author','Author:', top_level=>TRUE)
exec ctx_ddl.add_section('doc_section','review','Review:', top_level=>TRUE)
```

Example 2: Enclosed and non-enclosed repeating sections

```
<BODY>
<P> This is the first <B>paragraph</B>
<P> This is the second paragraph
</BODY>
```

Section Name	Start Tag	End Tag	Top Level	Enclose Self
BODY	<BODY>	</BODY>	Y	N
PARA	<P>	</P>	N	N
BOLD	<B>	</B>	N	N

```
exec ctx_ddl.add_section('html_section','BODY','<BODY>', '</BODY>', top_level=>TRUE)
exec ctx_ddl.add_section('html_section','PARA','<P>','</P>')
exec ctx_ddl.add_section('html_section','BOLD','<B>','</B>')
```

Example 3: Enclosed, overlapping sections

```
<CODE>
<OLD>
a := 9;
<NEW>
c := 14;
</OLD>
d := 15;
</NEW>
</CODE>
```

Section Name	Start Tag	End Tag	Top Level	Enclose Self
CODE	<CODE>	</CODE>	Y	N
OLD	<OLD>	</OLD>	N	N
NEW	<NEW>	</NEW>	N	N

```
exec ctx_ddl.add_section('html_sections','CODE','<CODE>', '</CODE>', top_level=>TRUE)
exec ctx_ddl.add_section('html_sections','OLD','<OLD>', '</OLD>')
exec ctx_ddl.add_section('html_sections','NEW','<NEW>', '</NEW>')
```

Example 4: Enclosed, self enclosing, repeating sections

```
<TABLE>
<TR>
<TD>March</TD>
<TD>
<TABLE>
<TR>
<TD>14</TD>
</TR>
</TABLE>
</TD>
</TR>
</TABLE>
```

Section Name	Start Tag	End Tag	Top Level	Enclose Self
TABLE	<TABLE>	</TABLE>	N	Y
ROW	<TR>	</TR>	N	Y
DATA	<TD>	</TD>	N	Y

```
exec ctx_ddl.add_section('html_sections','TABLE','<TABLE>','</TABLE>', enclose_self=>TRUE)
exec ctx_ddl.add_section('html_sections','ROW','<TR>','</TR>', enclose_self=>TRUE)
exec ctx_ddl.add_section('html_sections','DATA','<TD>','</TD>', enclose_self=>TRUE)
```

Notes

If the section group specified in *group\_name* is currently used in a preference, the preference must be dropped using CTX\_DDL.DROP\_PREFERENCE before sections can be added to the section group.

---

## CLEAR\_ATTRIBUTES

The `CLEAR_ATTRIBUTES` procedure clears the buffer of all attributes that have been set using `CTX_DDL.SET_ATTRIBUTE`.

### Syntax

```
CTX_DDL.CLEAR_ATTRIBUTES;
```

### Examples

```
execute ctx_ddl.clear_attributes
```

## CREATE\_INDEX

The `CREATE_INDEX` procedure creates an index for the column defined in the specified policy.

### Syntax

```
CTX_DDL.CREATE_INDEX(policy_name IN VARCHAR2,  
                      parallel     IN VARCHAR2 DEFAULT 1  
                      create_trig  IN BOOLEAN  DEFAULT TRUE  
                      pop_index    IN BOOLEAN  DEFAULT TRUE);
```

#### **policy\_name**

Specify the name of the policy for which the index is created.

#### **parallel**

Specify the number of ConText servers to be used in parallel to create the index for a column.

The default is 1.

#### **create\_trig**

Specify whether to create a DML trigger for the table or update the existing trigger to include the text column for the specified policy:

- `TRUE` (create/update trigger)
- `FALSE` (do not create/update trigger)

The default is *TRUE*.

#### **pop\_index**

Specify whether to populate the ConText index tables with index entries during ConText indexing:

- `TRUE` (create and populate ConText index tables)
- `FALSE` (create ConText index tables, but do not populate tables)

The default is *TRUE*.

## Examples

Examples are provided for parallel indexing, DML trigger control, and table population during indexing.

### Example 1: Parallel Indexing

In the following example, a ConText index is created with a parallelism level of 2 for the text column in *my\_policy*.

```
execute ctx_ddl.create_index('MY_POLICY', 2)
```

### Example 2: DML Trigger and Index Population Control

In the following example, a table has policies *pol1*, *pol2*, *pol3* for text columns *text1*, *text2*, *text3* respectively. ConText indexes are created for each policy:

```
ctx_ddl.create_index('P1', create_trig=>FALSE, pop_index=>FALSE);  
ctx_ddl.create_index('P2', create_trig=>TRUE, pop_index=>TRUE);  
ctx_ddl.create_index('P3', create_trig=>FALSE, pop_index=>FALSE);
```

The DML trigger is created for the table; however, only the text column (*text2*) for policy *pol2* is included in the trigger. As a result, only an update to the textkey or text column for policy *pol2* will cause a request to be inserted into the DML Queue.

In addition, during ConText indexing, only the ConText index tables for policy *pol2* are populated. To populate the ConText index tables for *pol1* and *pol3*, CTX\_DML.REINDEX must be called for each document in text columns *text1* and *text3*.

### Example 3: DML Trigger Control

In the following example, the same three policies and tables are used from before. The *create\_trig* parameter is set to *FALSE* for all three, so no DML trigger is created for the table. The *pop\_index* parameter is set to *TRUE* for all three, so the ConText index tables for all three policies are populated.

```
ctx_ddl.create_index('P1', create_trig=>FALSE, pop_index=>TRUE);  
ctx_ddl.create_index('P2', create_trig=>FALSE, pop_index=>TRUE);  
ctx_ddl.create_index('P3', create_trig=>FALSE, pop_index=>TRUE);
```

## Notes

If a DML trigger is not created for a table during ConText indexing, changes to the table will not result in the ConText index being updated. Changes to a document in the table can be recorded in the DML Queue using the CTX\_DML.REINDEX procedure; however, REINDEX must be called each time a document changes.



Automated DML notification can be enabled for the table by creating a trigger that calls CTX\_DML.REINDEX.

For example:

```
create or replace trigger resume_update
before delete or insert or update of empno,resume on ctxdev.emp
for each row
declare
    newkey varchar2(1000) := :new.empno;
    oldkey varchar2(1000) := :old.empno;
begin
    if inserting then
        ctx_dml.reindex('resume_pol',newkey);
    else if updating then
        ctx_dml.reindex('resume_pol',oldkey);
        ctx_dml.reindex('resume_pol',newkey);
    else
        ctx_dml.reindex('resume_pol',oldkey);
    end if;
end;
```

In this example, a trigger named *resume\_update* is created on a table named *emp* in the database schema for user *ctxdev*. *empno* is the primary key (and textkey) and *resume* is the text column for *emp*. A policy named *resume\_pol* has been created for the text column and an index created for the policy.

Each time a row is inserted or deleted from *emp*, or *empno* or *resume* is updated for an existing row, *resume\_update* places a DML request for the row (document) in the DML queue.

---

## CREATE\_POLICY

The CREATE\_POLICY procedure creates a policy for a column.

### Syntax

```
CTX_DDL.CREATE_POLICY(  
    policy_name      IN VARCHAR2,  
    colspec          IN VARCHAR2 DEFAULT NULL,  
    source_policy     IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_POLICY',  
    description      IN VARCHAR2 DEFAULT NULL,  
    textkey          IN VARCHAR2 DEFAULT NULL,  
    lineno           IN VARCHAR2 DEFAULT NULL,  
    dstore_pref      IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_DIRECT_DATASTORE',  
    compressor_pref  IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_NULL_COMPRESSOR',  
    filter_pref      IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_NULL_FILTER',  
    lexer_pref       IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_LEXER',  
    wordlist_pref    IN VARCHAR2 DEFAULT 'CTXSYS.NO_SOUNDSEX',  
    stoplist_pref    IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_STOPLIST',  
    engine_pref      IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_INDEX' );
```

#### **policy\_name**

Specify the name of the policy to be created.

#### **colspec**

Specify the column and table to which the policy is assigned. This is the column that contains the text to be indexed. If no value is specified for *colspec*, a template policy is created.

#### **source\_policy**

Specify the name of a template policy on which the column policy to be created is based.

The default is DEFAULT\_POLICY.

#### **description**

Specify the description of the policy.

**textkey**

Specify the column or columns (up to sixteen) that represent the unique identifier (textkey) for each document. This is usually the primary key(s) for the table, but can also be any column(s) for which a UNIQUE constraint has been defined.

---

---

**Note:** If no value is specified for *textkey* in CREATE\_POLICY, ConText does not, by default, always select the primary key column for the table identified in *colspec*.

ConText selects the first primary key or unique column encountered in the table. To ensure that the desired column(s) are defined as the textkey for a text column, always specify a *textkey* value when creating a policy for the column.

---

---

**lineno**

Specify the column that stores the unique ID for each document section in a master-detail table.

---

---

**Note:** This attribute is used *only* if the Data Store preference for the policy calls the MASTER DETAIL Tile.

If the Data Store preference calls the MASTER DETAIL NEW Tile, the line number column name is specified in the preference.

---

---

**dstore\_pref**

Specify the name of the Data Store preference assigned to the policy.

**compressor\_pref**

Specify the name of the Compressor preference assigned to the policy (Compressors are not currently provided or supported by ConText).

**filter\_pref**

Specify the name of the Filter preference assigned to the policy.

**lexer\_pref**

Specify the name of the Lexer preference assigned to the policy.

**wordlist\_pref**

Specify the name of the Wordlist preference assigned to the policy.

**stoplist\_pref**

Specify the name of the Stoplist preference assigned to the policy.

**engine\_pref**

Specify the name of the Engine preference assigned to the policy.

## Examples

```
begin
  ctx_ddl.create_policy(policy_name => 'MY_POLICY',
                        colspec     => 'DOCS.TEXT',
                        description => 'This is my policy',
                        textkey     => 'AUTH,TITLE',
                        dstore_pref => 'INTERNAL_STORE',
                        filter_pref => 'ASCII_TXT',
                        lexer_pref  => 'ENGLISH_BASIC',
                        wordlist_pref => 'CTXSYS.NO_SOUNDEX',
                        stoplist_pref => 'MY_LIST',
                        engine_pref  => 'BASIC_INDEX', );
end;
```

In this example, the textkey for *docs.text* is a composite textkey consisting of two columns named *auth* and *title* in *docs*.

## Notes

All of the arguments are optional, except for *policy\_name*. If you do not specify a preference for one of the categories, the default preference for the category is automatically used.

The values for *colspec* and *textkey* cannot be the same. In other words, a column that serves as a text column cannot also be the (only) column that uniquely identifies rows in the table.

For a composite textkey, each column name specified in *textkey* must be separated by a comma from the other column names. In addition, the string of column names is limited to 256 characters, including the comma.

If a preference belonging to another user is specified in a policy, the fully-qualified name of the preference must be used. For example, if you want to include the NO\_SOUNDEX predefined preference in a policy, the syntax would be:

```
exec ctx_ddl.create_policy(...,wordlist_pref => CTXSYS.NO_SOUNDEX,...)
```

## CREATE\_PREFERENCE

The CREATE\_PREFERENCE procedure creates a preference in the ConText data dictionary for a Tile. All Tile attributes and their values that have been set using CTX\_DDL.SET\_ATTRIBUTE are applied to the preference created by CREATE\_PREFERENCE.

The preference can then be used in a policy (indexing/linguistic generation) or a source (text loading).

### Syntax

```
CTX_DDL.CREATE_PREFERENCE(preference_name IN VARCHAR2,  
                           description      IN VARCHAR2,  
                           object_name     IN VARCHAR2);
```

#### **preference\_name**

Specify the name of the preference to be created.

#### **description**

Specify the description for the preference.

#### **object\_name**

Specify the Tile for the preference.

### Examples

```
begin  
  ctx_ddl.create_preference('NO_JOIN',  
                           'Lexer that does not use any printjoins',  
                           'BASIC LEXER');  
end;
```

### Notes

CREATE\_PREFERENCE *must* always be preceded by one or more SET\_ATTRIBUTE calls, which set the attribute values for the specified Tile.

Once CREATE\_PREFERENCE is called, the buffer used to store the attributes that were set for the preference is cleared. If the preference creation failed, all of the attributes must be entered again before calling CREATE\_PREFERENCE.

---

## CREATE\_SECTION\_GROUP

The `CREATE_SECTION_GROUP` procedure creates a section group for defining sections for a text column.

### Syntax

```
CTX_DDL.CREATE_SECTION_GROUP(group_name IN VARCHAR2);
```

**group\_name**

Specify the name of the section group to create.

### Examples

The following example creates a section group named *html\_sections*:

```
exec ctx_ddl.create_section_group('html_sections')
```

---

## CREATE\_SOURCE

The CREATE\_SOURCE procedure creates a text loading source for a column.

### Syntax

```
CTX_DDL.CREATE_SOURCE(name          IN VARCHAR2,
                      colspec       IN VARCHAR2 DEFAULT NULL,
                      description    IN VARCHAR2 DEFAULT NULL,
                      refresh        IN NUMBER   DEFAULT NULL,
                      engine_pref    IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_LOADER',
                      translator_pref IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_TRANSLATOR',
                      reader_pref    IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_READER' );
```

**name**

Specify the name of the source to be created.

**colspec**

Specify the column (and table) to which the source is assigned.

**description**

Specify the description of the source.

**refresh**

Specify the elapsed time, in minutes, before a ConText server checks the specified directory for new files to be loaded.

**engine\_pref**

Specify the name of the Loader Engine preference assigned to the source.

**translator\_pref**

Specify the name of the Translator preference assigned to the policy.

**reader\_pref**

Specify the name of the Reader preference assigned to the source.

## Examples

```
begin
  ctx_ddl.create_source(name      => 'MY_SOURCE',
                        colspec   => 'DOCS.TEXT',
                        desrcption => 'Source for loading',
                        reader_pref => 'DOCS_DIRECTORY');
end;
```

In this example, the default, predefined Loader Engine and Translator preferences are used.

## Notes

*colspec* must be a LONG or LONG RAW column, because load servers only support loading text into LONG or LONG RAW columns.

If a Loader Engine, Reader, or Translator preference belonging to another user is used to create a source, the fully-qualified name of the preference must be used.

The first time the source directory is scanned for files to load is SYSDATE (of source creation) + *refresh*. Subsequent scans occur at regular intervals specified by *refresh*.



## CREATE\_TEMPLATE\_POLICY

The `CREATE_TEMPLATE_POLICY` procedure creates a policy that does not have a reference to a text column. It is identical to `CTX_DDL.CREATE_POLICY`, except the *colspec* argument is not included.

The template policy can be used as a source policy for other policies in the user's schema. If CTXSYS creates a template policy, the policy is available to all ConText users.

### Syntax

```
CTX_DDL.CREATE_TEMPLATE_POLICY(
    policy_name      IN VARCHAR2,
    source_policy    IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_POLICY',
    description      IN VARCHAR2 DEFAULT NULL,
    textkey         IN VARCHAR2 DEFAULT NULL,
    lineno          IN VARCHAR2 DEFAULT NULL,
    dstore_pref     IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_DIRECT_DATASTORE',
    compressor_pref IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_NULL_COMPRESSOR',
    filter_pref     IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_NULL_FILTER',
    lexer_pref      IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_LEXER',
    wordlist_pref   IN VARCHAR2 DEFAULT 'CTXSYS.NO_SOUNDINDEX',
    stoplist_pref   IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_STOPLIST',
    engine_pref     IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_INDEX');
```

#### **policy\_name**

Specify the name of the template policy to be created.

#### **source\_policy**

Specify the name of another template policy on which the template policy to be created is based.

The default is `DEFAULT_POLICY`.

#### **description**

Specify the description of the template policy.

#### **textkey**

Specify the column or columns (up to sixteen) that represent the unique identifier (textkey) for each document.

**lineno**

Specify the column that stores the unique ID for each document section in a master-detail table.

**dstore\_pref**

Specify the name of the Data Store preference assigned to the template policy.

**compressor\_pref**

Specify the name of the Compressor preference assigned to the template policy (Compressors are not currently provided or supported by ConText).

**filter\_pref**

Specify the name of the Filter preference assigned to the template policy.

**lexer\_pref**

Specify the name of the Lexer preference assigned to the template policy.

**wordlist\_pref**

Specify the name of the Wordlist preference assigned to the template policy.

**stoplist\_pref**

Specify the name of the Stoplist preference assigned to the template policy.

**engine\_pref**

Specify the name of the Engine preference assigned to the template policy.

## Examples

See CTX\_DDL.CREATE\_POLICY

---

## DROP\_INDEX

The DROP\_INDEX procedure deletes the index for the column defined in the specified policy.

### Syntax

```
CTX_DDL.DROP_INDEX(policy_name IN VARCHAR2);
```

#### **policy\_name**

Specify the name of the policy for which the index is deleted.

### Examples

```
execute ctx_ddl.drop_index('MY_POLICY')
```

---

## DROP\_INTTRIG

The DROP\_INTTRIG procedure deletes the DML trigger for a specified table. A DML trigger is created/updated automatically for a table when a ConText index is created for a text column in the table.

### Syntax

```
CTX_DDL.DROP_INTTRIG(tablename IN VARCHAR2);
```

**tablename**

Specify the name of the table for which the DML trigger is dropped.

### Examples

```
execute ctx_ddl.drop_inttrig('DOCS')
```

### Notes

DROP\_INTTRIG deletes the trigger for the table; it cannot be used to selectively disable automatic DML for a text column in a table. If the table contains more than one text column with existing ConText indexes, automatic DML is disabled for *all* the text columns.

---

## DROP\_POLICY

The DROP\_POLICY procedure deletes the specified policy from the ConText data dictionary.

### Syntax

```
CTX_DDL.DROP_POLICY(policy_name IN VARCHAR2);
```

#### **policy\_name**

Specify the name of the policy to be dropped.

### Examples

```
execute ctx_ddl.drop_policy('MY_POLICY')
```

### Notes

If the specified policy has an existing index, the index must be dropped using CTX\_DDL.DROP\_INDEX before the policy can be dropped.

---

## DROP\_PREFERENCE

The DROP\_PREFERENCE procedure deletes the specified preference from the Con-Text data dictionary.

### Syntax

```
CTX_DDL.DROP_PREFERENCE(preference_name IN VARCHAR2);
```

#### **preference\_name**

Specify the name of the preference to be dropped.

### Examples

```
execute ctx_ddl.drop_preference('MY_ENGINE')
```

### Notes

If the specified preference is currently used in a policy, the policy must be dropped, using CTX\_DDL.DROP\_POLICY, before the preference can be dropped.

---

## DROP\_SECTION\_GROUP

The `DROP_SECTION_GROUP` deletes the specified section group, as well as all the sections in the group, from the ConText data dictionary.

### Syntax

```
CTX_DDL.DROP_SECTION_GROUP(group_name IN VARCHAR2);
```

**group\_name**

Specify the name of the section group to delete.

### Examples

```
exec ctx_ddl.drop_section_group('html_sections')
```

### Notes

If the specified section group is used in an existing Wordlist preference, the preference must be dropped, using `CTX_DDL.DROP_PREFERENCE`, before the section can be dropped from the section group.

---

## DROP\_SOURCE

The DROP\_SOURCE procedure deletes the specified text loading source from the ConText data dictionary. A source can be dropped at any time.

### Syntax

```
CTX_DDL.DROP_SOURCE(source_name IN VARCHAR2);
```

#### **source\_name**

Specify the name of the source to be dropped.

### Examples

```
execute ctx_ddl.drop_source('MY_LOADER')
```



## OPTIMIZE\_INDEX

The OPTIMIZE\_INDEX procedure optimizes the index for the column defined in the specified policy.

### Syntax

```
CTX_DDL.OPTIMIZE_INDEX(policy_name IN VARCHAR2,
                        opttyp      IN NUMBER  DEFAULT NULL,
                        threshold    IN NUMBER  DEFAULT 50,
                        parallel     IN NUMBER  DEFAULT 1,
                        switch_new   IN BOOLEAN DEFAULT TRUE,
                        drop_old     IN BOOLEAN DEFAULT TRUE);
```

#### **policy\_name**

Specify the name of the policy for the index to be optimized.

#### **opttyp**

Specify the type of optimization performed for the index:

- 1 (DR\_OPTIMIZE\_LAZY\_DELETES) - use original token table to remove references for deleted/modified documents
- 2 (DR\_OPTIMIZE\_COMPACT\_INDEXES) - use original token table to compact index fragments
- 3 (DR\_OPTIMIZE\_COMPACT\_NEW) - use mirror token table to compact index fragments
- 4 (DEFRAGMENT\_TO\_NEW\_TABLE) - use mirror token table to compact index fragments and remove deleted/modified document references
- 5 (DEFRAGMENT\_IN\_PLACE) - use original token table to compact index fragments and remove deleted document references

The default depends on the value set for the *default\_optimize* attribute in the BASIC ENGINE Tile (see “Notes” for this procedure).

#### **threshold**

Specify the threshold, as a percentage, under which a term’s index strings are not compacted during in-place compaction.

The default is 50.

**parallel**

Specify the number of ConText servers to be used in parallel to perform two-table optimization.

The default is 1.

**switch\_new**

For internal use only.

**drop\_old**

For internal use only.

## Examples

```
begin
  ctx_ddl.optimize_index('MY_POLICY',
                        opttyp => ctx_ddl.defragment_in_place,
                        parallel => 2);
end;
```

## Notes

Optimization cannot be performed for an index while any other operation (i.e. creation, updating, deletion) is being performed on the index.

*opttyp* must be fully qualified with the PL/SQL package name (CTX\_DDL) as shown in the examples.

The default for *opttyp* is the value specified for the DEFAULT\_OPTIMIZE attribute (BASIC ENGINE Tile) in the Engine preference of the policy for the text column to be optimized. If no value was specified for DEFAULT\_OPTIMIZE when the Engine preference for the policy was created, the default is DEFRACTIONMENT\_TO\_NEW\_TABLE.

DEFRACTIONMENT\_IN\_PLACE does not use *threshold*. If *opttyp* is DEFRACTIONMENT\_IN\_PLACE, OPTIMIZE\_INDEX ignores any value specified for *threshold*.

*parallel* is used only for two-table compaction and two-table combined reference deletion and compaction.

*threshold* is used only for in-place compaction. It specifies the percentage under which ConText compacts a term's index fragments (rows) if the compaction will result in the number of fragments for the term being reduced to more than or equal to the percentage specified.

For example, a value of 60 for *threshold* indicates the number of fragments for a given term must be reduced to 60% or more of the total number of pre-optimization fragments for in-place compaction to take place.

---

## REMOVE\_SECTION

The REMOVE\_SECTION procedure removes the specified section from the specified section group.

### Syntax

```
CTX_DDL.REMOVE_SECTION(group_name  IN VARCHAR2,  
                        section_name IN VARCHAR2);
```

**group\_name**

Specify the name of the section group from which ConText deletes the section.

**section\_name**

Specify the name of the section ConText deletes from the section group.

### Examples

```
exec ctx_ddl.remove_section('html_sections', 'H1')
```

### Notes

If the specified section is part of a section group used in an existing Wordlist preference, the preference must be dropped, using CTX\_DDL.DROP\_PREFERENCE, before the section can be dropped from the section group.

## RESUME\_FAILED\_INDEX

The RESUME\_FAILED\_INDEX procedure resumes an unsuccessful text DDL operation (index creation/optimization).

### Syntax

```
CTX_DDL.RESUME_FAILED_INDEX(policy_name IN VARCHAR2,
                             operation    IN NUMBER  DEFAULT 1,
                             parallel    IN NUMBER  DEFAULT 1,
                             opttyp      IN NUMBER  DEFAULT 3,
                             switch_new  IN BOOLEAN DEFAULT TRUE,
                             drop_old    IN BOOLEAN DEFAULT TRUE);
```

#### **policy\_name**

Specify the index (through the policy) that requires an Oracle index.

#### **operation**

Specify the operation that was being performed on the index at the time of failure:

- 1 (OPERATION\_CREATE)
- 2 (OPERATION\_OPTIMIZE)

The default is 1.

#### **parallel**

If *operation* is 1 (index creation), then use this argument to specify the degree of parallelism used for creating the index.

The default is 1.

#### **opttyp**

If *operation* is 2 (OPERATION\_OPTIMIZE), use this argument to specify the method of two-table optimization to use:

- 3 (DR\_OPTIMIZE\_COMPACT\_NEW) - use mirror index table to compact index fragments)
- 4 (DEFRAGMENT\_TO\_NEW\_TABLE) - use mirror token table to compact index fragments and remove deleted/modified document references)

The default depends on the value set for the *default\_optimize* attribute in the BASIC ENGINE Tile (see notes).

## Examples

```
begin
  ctx_ddl.resume_failed_index('MY_POLICY',
                             operation => 2,
                             parallel  => 2,
                             opttyp    => ddl.defragment_to_new_table);
end;
```

In this example, optimization (*operation* => 2) is resumed with a parallelism level of 2 for the index for *my\_policy*. The type of optimization performed is compaction and garbage collection combined.

## Notes

RESUME\_FAILED\_INDEX should be called only after the problem that caused the failure has been corrected or removed.

Only the owner of the policy or CTXSYS can resume creation of a ConText index.

RESUME\_FAILED\_INDEX uses the ConText index log to determine the point of failure for the index and the point from which to proceed with indexing/optimization.

Depending on the stage at which the text DDL operation failed, RESUME\_FAILED\_INDEX may start the operation from the beginning, in which case, CREATE\_INDEX or OPTIMIZE\_INDEX serves the same purpose as RESUME\_FAILED\_INDEX and can be called in its place.

Because RESUME\_FAILED\_INDEX automatically determines where to resume a failed DDL operation, the user should consult the index log before calling RESUME\_FAILED\_INDEX to decide whether to call CREATE\_INDEX/OPTIMIZE\_INDEX instead.

*opttyp* must be fully qualified with the PL/SQL package name (CTX\_DDL) as shown in the examples.

The default for *opttyp* is the value specified for the *default\_optimize* attribute (BASIC ENGINE Tile) in the Engine preference of the policy for the text column to be optimized. If no value was specified for *default\_optimize* when the Engine preference for the policy was created, the default is 3 (DR\_OPTIMIZE\_COMPACT\_NEW).

## SET\_ATTRIBUTE

The SET\_ATTRIBUTE procedure assigns values to Tile attributes used in the CTX\_DDL.CREATE\_PREFERENCE procedure.

### Syntax

```
CTX_DDL.SET_ATTRIBUTE(name  IN VARCHAR2,  
                      value  IN VARCHAR2,  
                      seq    IN NUMBER DEFAULT 1);
```

```
CTX_DDL.SET_ATTRIBUTE(name  IN VARCHAR2,  
                      value1 IN VARCHAR2,  
                      value2 IN VARCHAR2,  
                      seq    IN NUMBER);
```

#### **name**

Specify the attribute to which a value is assigned.

#### **value**

Specify the value assigned to the attribute. This argument is not used when *value1* and *value2* are used.

#### **value1**

Specify the first value assigned to the attribute (used only with the *executable* attribute for the BLASTER FILTER Tile).

#### **value2**

Specify the second value assigned to attribute (used only with the *executable* attribute for the BLASTER FILTER Tile).

#### **seq**

Specify the sequence number assigned to the attribute (only required for creating preferences that use Tiles which support multiple values for the same attribute).

The default is 1.

## Examples

Examples are provided for setting attributes for Engine, Stoplist, and Filter Tiles.

### Example 1: Engine Tile Attribute

In this example, the *index\_memory* attribute is assigned approximately 3 megabytes of memory. The *index\_memory* attribute belongs to the GENERIC ENGINE Tile and is used for allocating indexing memory.

```
execute ctx_ddl.set_attribute('INDEX_MEMORY', '3000000')
```

### Example 2: Stoplist Tile Attributes

In this example, the *stop\_word* attribute (GENERIC STOP LIST Tile) is set twice, once for the stop word *of* and once for the stop word *and*. The stop words are assigned sequences of 1 and 2 respectively.

```
execute ctx_ddl.set_attribute('STOP_WORD', 'of', 1)
execute ctx_ddl.set_attribute('STOP_WORD', 'and', 2)
```

### Example 3: Filter Tile Attributes

In example 3, the *executable* attribute (BLASTER FILTER Tile) is set twice to register external filter executables (*amipro.sh* and *acrobat.sh*) for AmiPro and Adobe Acrobat (PDF) documents. AmiPro has a format code of 19 and Acrobat has a format code of 57. The executables are assigned sequences of 1 and 2 respectively.

```
execute ctx_ddl.set_attribute('EXECUTABLE', 19, 'amipro.sh', 1)
execute ctx_ddl.set_attribute('EXECUTABLE', 57, 'acrobat.sh', 2)
```

## Notes

SET\_ATTRIBUTE writes the specified attribute values to an internal buffer. Once all of the attributes for a particular Tile have been set, CTX\_DDL.CREATE\_PREFERENCE is called to create a preference for the Tile.

Any errors that may occur from entering incorrect values for SET\_ATTRIBUTE are not reported until CREATE\_PREFERENCE is called.

When CREATE\_PREFERENCE is called, the buffer used to store the attributes for the preference is automatically cleared. As a result, if the preference creation failed, all of the attributes must be entered again before calling CREATE\_PREFERENCE.

CTX\_DDL.CLEAR\_ATTRIBUTES can be used to manually clear all attributes in the buffer.



*seq* is only used with the Tiles that have attributes that support multiple values for the same attribute (i.e. BLASTER FILTER, GENERIC STOP LIST, and GENERIC WORD LIST). For all the other Tiles, *seq* is not required and should *not* be set.

A call to SET\_ATTRIBUTE that uses the same *seq* value as a previous call to SET\_ATTRIBUTE overrides the previously attribute that was set in the buffer.

---

## UPGRADE\_INDEX

The UPGRADE\_INDEX procedure upgrades the ConText index for a policy from the format used in ConText, Release 2.0 and earlier, to the current format.

### Syntax

```
CTX_DDL.UPGRADE_INDEX(policy_name IN VARCHAR2);
```

#### **policy\_name**

The name of the policy for which the index is upgraded.

### Examples

In the following example, UPGRADE\_INDEX is called for a policy named *doc\_pol1* that is owned by *ctxdemo*.

```
connect ctxdemo/passwd
SQL> exec ctx_ddl.upgrade_index('doc_pol1')
```

In the following example, UPGRADE\_INDEX is called by CTXSYS for a policy named *doc\_pol2* that is owned by *ctxdemo*.

```
connect ctxsys/passwd
SQL> exec ctx_ddl.upgrade_index('ctxdemo.doc_pol2')
```

### Notes

You only need to run UPGRADE\_INDEX for ConText indexes that were created in Release 2.0 or earlier. In addition, you only need to run UPGRADE\_INDEX once for each ConText index.

If CTXSYS is used to upgrade the indexes for other users, the policy name specified for UPGRADE\_INDEX must be fully qualified with the username for the user.

The CTX\_INDEX\_LOG view can be used by users with the CTXADMIN role to view the status of all ConText indexes.

The CTX\_USER\_INDEX\_LOG view can be used by users with the CTXAPP role to view the status of all ConText indexes for the user.

## UPDATE\_POLICY

The UPDATE\_POLICY procedure updates the description and/or the preferences for an existing column or template policy. For column policies, it can only be used to update a column policy if ConText has not yet generated a ConText index for the policy.

### Syntax

```
CTX_DDL.UPDATE_POLICY(
    policy_name      IN VARCHAR2,
    description      IN VARCHAR2 DEFAULT NULL,
    dstore_pref      IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_DIRECT_DATASTORE',
    compressor_pref  IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_NULL_COMPRESSOR',
    filter_pref      IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_NULL_FILTER',
    lexer_pref       IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_LEXER',
    wordlist_pref    IN VARCHAR2 DEFAULT 'CTXSYS.NO_SOUNDINDEX',
    stoplist_pref    IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_STOPLIST',
    engine_pref      IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_INDEX');
```

#### **policy\_name**

Specify the name of the policy to be updated.

#### **description**

Specify the new description of the policy.

#### **dstore\_pref**

Specify the name of the new Data Store preference for the policy.

#### **compressor\_pref**

Specify the name of the new Compressor preference (Compressors are not currently provided or supported by ConText).

#### **filter\_pref**

Specify the name of the new Filter preference for the policy.

#### **lexer\_pref**

Specify the name of the new Lexer preference for the policy.

#### **wordlist\_pref**

Specify the name of the new Wordlist preference for the policy.

**stoplist\_pref**

Specify the name of the new Stoplist preference for the policy.

**engine\_pref**

Specify the name of the new Engine preference for the policy.

**Examples**

```
begin
  ctx_ddl.update_policy(policy_name => 'MY_POLICY',
                        dstore_pref => 'CTXSYS.MD_BINARY');
end;
```

**Notes**

If a preference belonging to another user is used to update a policy, the fully-qualified name of the preference must be used.

## UPDATE\_SOURCE

The UPDATE\_SOURCE procedure updates the description, text column, refresh rate, and preferences for the text loading source specified in the argument string. UPDATE\_SOURCE can be called at any time for any existing source.

### Syntax

```
CTX_DDL.UPDATE_SOURCE(name          IN VARCHAR2,
                        colspec      IN VARCHAR2 DEFAULT NULL,
                        description   IN VARCHAR2 DEFAULT NULL,
                        refresh       IN NUMBER   DEFAULT NULL,
                        next           IN DATE     DEFAULT NULL,
                        engine_pref   IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_LOADER',
                        translator_pref IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_TRANSLATOR',
                        reader_pref   IN VARCHAR2 DEFAULT 'CTXSYS.DEFAULT_READER' );
```

#### **name**

Specify the name of the source to be updated.

#### **colspec**

Specify the new text column (and table) to which the source is assigned.

#### **description**

Specify the new description for the source.

#### **refresh**

Specify the new elapsed time, in minutes, before a ConText server checks the directory (specified in the Reader preference) for new files to be loaded.

#### **next**

Specify the date and time for the initial scan of the updated source by available Loader servers.

#### **engine\_pref**

Specify the name of the new Loader Engine preference assigned to the source.

#### **translator\_pref**

Specify the name of the new Translator preference assigned to the source.

#### **reader\_pref**

Specify the name of the new Reader preference assigned to the source.

## Examples

```
begin
  ctx_ddl.update_policy(policy_name => 'MY_POLICY',
                        dstore_pref => 'CTX.MD_BINARY' );
end;
```

## Notes

If a Loader Engine, Reader, or Translator preference belonging to another user is used to update a source, the fully-qualified name of the preference must be used.

*next* specifies the date and time that an updated source is initially scanned by ConText servers running with the Loader (R) personality.

The next scan of the source occurs at *next* + *refresh*, then all subsequent scans occur at regular intervals specified by *refresh*.

## CTX\_DML: ConText Index Update

The CTX\_DML PL/SQL package is used to manage DML Operations.

CTX\_DML contains the following stored procedures and functions:

Name	Description
REINDEX	Specify reindexing for a document
SYNC	Batches all pending requests in DML Queue and enables ConText servers with DDL personality to process the batches
SYNC_QUERY	Returns a time-stamp in the form of a date for the batches generated by SYNC

## REINDEX

The REINDEX procedure is used to write a row to the DML Queue for a specified document. The index for the document is then created/updated according to the DML method being used (immediate or batch).

REINDEX can be used to reindex documents that have errored during DDL or DML. It can also be used to provide automatic DML processing on a view. Views cannot have a trigger assigned, meaning that DML operations on a view cannot be sent to the DML Queue by way of the trigger that is automatically created when a table is indexed. Instead, a trigger that calls REINDEX must be created on the base table containing the text column for the view.

Finally, it can be used to notify the system of updates to documents stored externally. If a document uses the OSFILE Tile, REINDEX can be called when the document is updated to ensure that the update is recorded in the DML Queue.

### Syntax

```
CTX_DML.REINDEX(policy IN VARCHAR2,  
                 pk      IN VARCHAR2);
```

```
CTX_DML.REINDEX(cid IN NUMBER,  
                 pk  IN VARCHAR2);
```

#### **policy**

Specify name of policy for text column where document to be reindexed is stored. If *policy* is used, *cid* is not used.

#### **cid**

Specify the identifier for the text column where document to be reindexed is stored. If *cid* is used, *policy* is not used.

#### **pk**

Specify the identifier for the document to be reindexed.

### Examples

```
execute ctx_dml.reindex('MY_POLICY', '1')
```

```
execute ctx_dml.reindex(3451, '1')
```



**Notes**

REINDEX uses either the policy name or the column ID to identify the column where the document to be reindexed is stored.

REINDEX does not perform a COMMIT. After REINDEX is called for a document, COMMIT must be performed to save the request in the DML Queue.

---

## SYNC

The SYNC procedure bundles all pending rows in the DML Queue at the time it is called and enables ConText servers with the DDL personality to process the rows as a single batch (if parallelism is not specified) or as a group of batches (if parallelism is specified).

### Syntax

```
CTX_DML.SYNC(timestamp IN DATE      DEFAULT NULL,
               pol      IN VARCHAR2  DEFAULT NULL,
               parallel IN NUMBER    DEFAULT 1,
               testing  IN NUMBER    DEFAULT 0,
               timeout  IN NUMBER    DEFAULT 0);
```

**timestamp**

Specify the time at which you want the batch DML to start.

The default is SYSDATE.

**pol**

Specify the policy for the text column for which SYNC is performed.

**parallel**

Specify the number of ConText servers used to process the operation.

The default is 1.

**testing**

For internal use only.

**timeout**

For internal use only.

### Examples

```
execute ctx_dml.sync(PARALLEL=>2)
```

## Notes

*timestamp* limits the rows in the batch to those rows with a date equal to or less than the date specified.

*pol* limits SYNC to a particular text column. If a value is not specified for *pol*, SYNC is performed for every text column in the database.

---

## SYNC\_QUERY

The SYNC\_QUERY function returns a DATE which is the lower bound to which rows in the DML Queue have been indexed.

### Syntax

```
CTX_DML.SYNC_QUERY(cid      IN NUMBER DEFAULT NULL,  
                    cur_date IN DATE   DEFAULT SYSDATE)  
RETURN DATE;
```

#### **cid**

Specify the text column for which SYNC\_QUERY is called.

#### **cur\_date**

Specify the date from which to perform the query synchronization.

### Returns

The timestamp (date and time) for the reindexed rows.

### Examples

```
select ctx_dml.sync_query(3) from dual;
```

### Notes

*cid* can be used to limit SYNC\_QUERY to a particular text column. Otherwise, SYNC\_QUERY returns the DATE value for all text columns.

## CTX\_THES: Thesaurus Management

The CTX\_THES PL/SQL package is used to manage thesauri in the ConText thesaurus tables.

CTX\_THES contains the following stored procedures and functions:

Name	Description
CREATE_PHRASE	Adds a phrase to the specified thesaurus or modifies the information about the phrase in the thesaurus and returns the ID for the phrase
CREATE_THESAURUS	Creates the specified thesaurus and returns the ID for the thesaurus
DROP_THESAURUS	Drops the specified thesaurus from the thesaurus tables

---

**Note:** The remaining procedures and functions in CTX\_THES are used to enable the thesaurus operators in query expressions.

For more information about the thesaurus operators, see *Oracle8 ConText Cartridge Application Developer's Guide*.

---

---

## CREATE\_PHRASE

The CREATE\_PHRASE function adds a new phrase to the specified thesaurus or creates a relationship between two existing phrases.

### Syntax

```
CTX_THES.CREATE_PHRASE(tname    IN VARCHAR2,  
                        phrase    IN VARCHAR2,  
                        rel       IN VARCHAR2 DEFAULT NULL,  
                        relname   IN VARCHAR2 DEFAULT NULL)  
  
RETURN NUMBER;
```

#### **tname**

Specify the name of the thesaurus in which the new phrase is added or the existing phrase is located.

#### **phrase**

Specify the phrase to be added to a thesaurus or the phrase for which a new relationship is created.

#### **rel**

Specify the new relationship between *phrase* and *relname*:

- SYN
- BT
- NT
- BTG
- NTG
- BTP
- NTP
- RT
- TT

**See Also:** For more information about the relationships you can define for thesaurus entries, see “Thesauri” in Chapter 6, “Text Concepts”.

**relname**

Specify the existing phrase that is related to *phrase*.

**Returns**

The ID for the entry.

**Examples**

Examples are provided for creating entries for two phrases and defining a relationship between the phrases.

**Example 1: Creating Entries for Phrases**

In this example, two new phrases (*os* and *operating system*) are created in a thesaurus named *tech\_thes*.

```
declare phraseid number;
begin
    phraseid := ctx_thes.create_phrase('tech_thes','os');
    phraseid := ctx_thes.create_phrase('tech_thes','operating system');
end;
```

**Example 2: Creating a Relationship**

In this example, the two phrases (*os* and *operating system*) in *tech\_thes* are recorded as synonyms (*syn*).

```
declare phraseid number;
begin
    phraseid := ctx_thes.create_phrase('tech_thes','os','syn','oprating system');
end;
```

**Notes**

*rel* and *relname* can only be used in CREATE\_PHRASE if the phrases specified for both *phrase* and *relname* already exist in the thesaurus.

CREATE\_PHRASE cannot be used to update the relationship between two existing phrases. It can only be used to create a new relationship between two existing phrases.

---

## CREATE\_THESAURUS

The `CREATE_THESAURUS` function creates an empty thesaurus with the specified name in the thesaurus tables.

### Syntax

```
CTX_THES.CREATE_THESAURUS(thes_name      IN VARCHAR2
                           case_sensitive IN BOOLEAN DEFAULT FALSE)
RETURN NUMBER;
```

#### **thes\_name**

Specify the name of the thesaurus to be created.

#### **case\_sensitive**

Specify whether the thesaurus to be created is case-sensitive. If `case_sensitive` is *TRUE*, ConText retains the cases of all terms entered in the specified thesaurus. As a result, queries that use the thesaurus are case-sensitive.

### Returns

The ID for the thesaurus.

### Examples

```
declare thesid number;
begin
    thesid := ctx_thes.create_phrase('tech_thes');
end;
```

### Notes

The name of the thesaurus must be unique. If a thesaurus with the specified name already exists, `CREATE_THESAURUS` returns an error and does not create the thesaurus.

To enter phrases in the thesaurus, use `CTX_THES.CREATE_PHRASE` or use the Thesaurus Maintenance screen in the ConText System Administration tool.



---

## DROP\_THESAURUS

The DROP\_THESAURUS procedure deletes the specified thesaurus and all of its entries from the thesaurus tables.

### Syntax

```
CTX_THES.DROP_THESAURUS(name IN VARCHAR2);
```

#### **name**

Specify the name of the thesaurus to be dropped.

### Examples

```
execute ctx_thes.drop_thesaurus('tech_thes');
```



# Part III

---

## Appendices



---

## Supplied Stoplists

ConText provides predefined Stoplist preferences for English, French, German, Italian, and Spanish.

This appendix lists the stop words included in each of the stoplists:

- English Stoplist
- French Stoplist
- German Stoplist
- Italian Stoplist
- Spanish Stoplist

# English Stoplist

The DEFAULT\_STOPLIST preference defines the following English terms as stop words

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
A	BE	HAD	IT	ONLY	SHE	WE
ABOUT	BECAUSE	HAS	ITS	OF	SOME	WERE
AFTER	BEEN	HAVE	LAST	ON	SUCH	WHEN
ALL	BUT	HE	MORE	ONE	THAN	WHICH
ALSO	BY	HER	MOST	OR	THAT	WHO
AN	CAN	HIS	MR	OTHER	THE	WILL
ANY	CO	IF	MRS	OUT	THEIR	WITH
AND	CORP	IN	MS	OVER	THERE	WOULD
ARE	COULD	INC	MZ	S	THIS	UP
AS	FOR	INTO	NO	SO	TO	
AT	FROM	IS	NOT	SAYS	WAS	

---

**Note:** Because the stop words in DEFAULT\_STOPLIST are in all-uppercase, DEFAULT\_STOPLIST should be used *only* when creating case-insensitive text indexes.

For more information about case-sensitivity in text indexes, see “What’s in a Text Index?” in Chapter 6, “Text Concepts”.

---

## French Stoplist

The FRENCH\_STOPLIST preference defines the following terms as stop words.

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	celles	dessus	jusque	ne	quelle	tant
afin	celui	dès	la	ni	quelqu'un	te
ailleurs	cependant	donc	laquelle	non	quelqu'une	telle
ainsi	certain	donné	là	nos	quelque	telles
alors	certaine	dont	le	notamment	quelques-unes	tes
après	certaines	du	lequel	notre	quelques-uns	tienne
attendant	certains	duquel	les	notres	quels	tiennes
au	ces	durant	lesquelles	nôtre	qui	tiens
aucun	cet	elle	lesquels	nôtres	quiconque	toi
aucune	cette	elles	leur	nous	quoi	ton
au-dessous	ceux	en	leurs	nulle	quoique	toujours
au-dessus	chacun	encore	lors	nulles	sa	tous
auprès	chacune	entre	lorsque	on	sans	toute
auquel	chaque	et	lui	ou	sauf	toutes
aussi	chez	étaient	ma	où	se	très
aussitôt	combien	était	mais	par	selon	trop
autant	comme	étant	malgré	parce	ses	tu
autour	comment	etc	me	parmi	sien	un
aux	concernant	eux	même	plus	sienne	une
auxquelles	dans	furent	mêmes	plusieurs	siennes	vos
auxquels	de	grâce	mes	pour	siens	votre
avec	dedans	hormis	mien	pourquoi	soi	vôtre
à	dehors	hors	mienne	près	soi-même	vôtres
beaucoup	déjà	ici	miennes	puis	soit	vous
ça	delà	il	miens	puisque	sont	vu
ce	depuis	ils	moins	quand	suis	y
ceci	des	jadis	moment	quant	sur	
cela	desquelles	je	mon	que	ta	
celle	desquels	jusqu	moyennant	quel	tandis	

## German Stoplist

This section lists the stop words in the predefined preference, GERMAN\_STOPLIST:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
ab	darauf	die	eurem	Ihrer	meiner	seit	vom	zur
aber	daraus	dies	euren	ihrer	meines	seitdem	von	zwar
allein	darin	diese	eurer	ihres	mich	selbst	vor	zwischen
als	darüber	dieselbe	eures	Ihres	mir	sich	während	zwichens
also	darum	dieselben	für	im	mit	Sie	war	
am	darunter	diesem	fürs	in	nach	sie	wäre	
an	das	diesen	ganz	ist	nachdem	sind	wären	
auch	dasselbe	dieser	gar	ja	nämlich	so	warum	
auf	daß	dieses	gegen	je	neben	sogar	was	
aus	davon	dir	genau	jedesmal	nein	solch	wegen	
außer	davor	doch	gewesen	jedoch	nicht	solche	weil	
bald	dazu	dort	her	jene	nichts	solchem	weit	
bei	dazwischen	du	herein	jenem	noch	solchen	welche	
beim	dein	ebenso	herum	jenen	nun	solcher	welchem	
bin	deine	ehe	hin	jener	nur	solches	welchen	
bis	deinem	ein	hinter	jenes	ob	sondern	welcher	
bißchen	deinen	eine	hintern	kaum	ober	sonst	welches	
bist	deiner	einem	ich	kein	obgleich	soviel	wem	
da	deines	einen	ihm	keine	oder	soweit	wen	
dabei	dem	einer	ihn	keinem	ohne	über	wenn	
dadurch	demselben	eines	Ihnen	keinen	paar	um	wer	
dafür	den	entlang	ihnen	keiner	sehr	und	weshalb	
dagegen	denn	er	ihr	keines	sei	uns	wessen	
dahinter	der	es	ihre	man	sein	unser	wie	
damit	derselben	etwa	Ihre	mehr	seine	unsre	wir	
danach	des	etwas	ihrem	mein	seinem	unsrem	wo	
daneben	desselben	euch	Ihrem	meine	seinen	unsren	womit	
dann	dessen	euer	ihren	meinem	seiner	unsrer	zu	
daran	dich	eure	Ihren	meinen	seines	unsres	zum	



## Italian Stoplist

The ITALIAN\_STOPLIST preference defines the following terms as stop words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	da	durante	lo	o	seppure	un
affinchè	dachè	e	loro	onde	si	una
agl'	dagl'	egli	ma	oppure	siccome	uno
agli	dagli	eppure	mentre	ossia	sopra	voi
ai	dai	essere	mio	ovvero	sotto	vostro
al	dal	essi	ne	per	su	
all'	dall'	finché	neanche	perchè	subito	
alla	dalla	fino	negl'	perciò	sugl'	
alle	dalle	fra	negli	però	sugli	
allo	dallo	giacchè	nei	poichè	sui	
anzichè	degl'	gl'	nel	prima	sul	
avere	degli	gli	nell'	purchè	sull'	
bensi	dei	grazie	nella	quand'anche	sulla	
che	del	I	nelle	quando	sulle	
chi	dell'	il	nello	quantunque	sullo	
cioè	delle	in	nemmeno	quasi	suo	
come	dello	inoltre	neppure	quindi	talchè	
comunque	di	io	noi	se	tu	
con	dopo	l'	nonchè	sebbene	tuo	
contro	dove	la	nondimeno	sennonchè	tuttavia	
cosa	dunque	le	nostro	senza	tutti	

## Spanish Stoplist

The SPANISH\_STOPLIST preference defines the following terms as stop words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	con	él	mismos	otros	suyos	vuestra
acá	conmigo	esa	mucha	para	tal	vuestras
ahí	consigo	esas	muchas	parecer	tales	vuestro
ajena	contigo	ese	muchísima	poca	tan	vuestros
ajenas	cualquier	esos	muchísimas	pocas	tanta	y
ajeno	cualquiera	esta	muchísimo	poco	tantas	yo
ajenos	cualquieras	estar	muchísimos	pocos	tanto	
al	cuan	estas	mucho	por	tantos	
algo	cuanta	este	muchos	porque	te	
alguna	cuantas	estos	muy	que	tener	
algunas	cuánta	hacer	nada	querer	ti	
alguno	cuántas	hasta	ni	qué	toda	
algunos	cuanto	jamás	ninguna	quien	todas	
algún	cuantos	junto	ningunas	quienes	todo	
allá	cuán	juntos	ninguno	quienesquiera	todos	
allí	cuánto	la	ningunos	quienquiera	tomar	
aquel	cuántos	las	no	quién	tuya	
aquella	de	lo	nos	ser	tuyo	
aquellas	dejar	los	nosotras	si	tú	
aquello	del	mas	nosotros	siempre	un	
aquellos	demasiada	más	nuestra	sí	una	
aquí	demasiadas	me	nuestras	sín	unas	
cada	demasiado	menos	nuestro	Sr	unos	
cierta	demasiados	mía	nuestros	Sra	usted	
ciertas	demás	mientras	nunca	Sres	ustedes	
cierto	el	mío	os	Sta	varias	
ciertos	ella	misma	otra	suya	varios	
como	ellas	mismas	otras	suyas	vosotras	
cómo	ellos	mismo	otro	suyo	vosotros	

---

## ConText Views

This appendix lists all of the views provided by ConText.

The views are divided into the following three groups:

- ConText Server Views
- ConText Queue Views
- ConText Data Dictionary Views

# ConText Server Views

This section describes the views provided with ConText for monitoring the status of ConText servers.

## CTX\_ALL\_SERVERS

This view displays all the ConText servers that have been started, including idle servers and inactive servers. Only users assigned the CTXAPP or CTXADMIN roles can query CTX\_ALL\_SERVERS.

Column Name	Type	Description
SER_NAME	VARCHAR2(60)	ConText server identifier
SER_STATUS	VARCHAR2(8)	ConText server status (IDLE, RUN, EXIT)
SER_ADMMBX	VARCHAR2(60)	Admin pipe mailbox name for ConText server
SER_OOBBMX	VARCHAR2(60)	Out-of-bound mailbox name for ConText server
SER_SESSION	NUMBER	ConText server session ID
SER_AUDSID	NUMBER	ConText server audit session ID
SER_DBID	NUMBER	ConText server database ID
SER_PROCID	VARCHAR2(10)	ConText server process ID
SER_PERSON_MASK	VARCHAR2(30)	Personality mask for ConText server
SER_STARTED_AT	DATE	Date on which ConText server was started
SER_IDLE_TIME	NUMBER	Idle time, in seconds, for ConText server
SER_DB_INSTANCE	VARCHAR2(10)	Database instance ID
SER_MACHINE	VARCHAR2(64)	Name of host machine on which ConText server is running

## CTX\_SERVERS

This view displays only ConText servers that are currently active. Only DBA users and users assigned CTXADMIN can query CTX\_SERVERS.

Column Name	Type	Description
SER_NAME	VARCHAR2(60)	ConText server identifier
SER_STATUS	VARCHAR2(8)	ConText server status (IDLE, RUN, EXIT)
SER_ADMMBX	VARCHAR2(60)	Admin pipe mailbox name for ConText server
SER_OOBMBX	VARCHAR2(60)	Out-of-bound mailbox name for ConText server
SER_SESSION	NUMBER	ConText server session ID
SER_AUDSID	NUMBER	ConText server audit session ID
SER_DBID	NUMBER	ConText server database ID
SER_PROCID	VARCHAR2(10)	ConText server process ID
SER_PERSON_MASK	VARCHAR2(30)	Personality mask for ConText server
SER_STARTED_AT	DATE	Date on which ConText server was started
SER_IDLE_TIME	NUMBER	Idle time, in seconds, for ConText server
SER_DB_INSTANCE	VARCHAR2(10)	Database instance ID
SER_MACHINE	VARCHAR2(64)	Name of host machine on which ConText server is running

# ConText Queue Views

This section describes the views provided with ConText for monitoring the status of all the ConText queues.

## CTX\_ALL\_DML\_QUEUE

This view displays a row for each entry in the DML Queue. Only users assigned CTXADMIN can query CTX\_ALL\_DML\_QUEUE.

Column Name	Type	Description
CID	NUMBER	Text column ID
POL_OWNER	VARCHAR2(30)	Owner of policy for text column
POL_NAME	VARCHAR2(30)	Name of policy for text column
SID	VARCHAR2(10)	Identifier for ConText server working on row (value is PENDING if a ConText server is not yet assigned)
PKEY	VARCHAR2(256)	Primary key of row being processed
TIME	DATE	Lower bound of time row was last updated

## CTX\_ALL\_DML\_SUM

This view displays the total number of entries in the DML Queue for each policy. Only users assigned CTXADMIN can query CTX\_ALL\_DML\_QUEUE.

Column Name	Type	Description
CID	NUMBER	Text column ID
POL_OWNER	VARCHAR2(30)	Owner of policy for text column
POL_NAME	VARCHAR2(30)	Name of policy for text column
CNT	NUMBER	Count of rows in DML Queue
TSTAMP	DATE	Minimum time stamp for rows

## CTX\_ALL\_QUEUE

This view displays all of the rows (pending and in progress requests) in the DML Queue. Only users assigned CTXADMIN can query CTX\_ALL\_QUEUE.

Column Name	Type	Description
CID	NUMBER	Policy ID
SID	VARCHAR2(10)	ID (name) of server processing the request (if the request is pending, this column is NULL)
PKEY	VARCHAR2(256)	Textkey (primary key) of column policy
TIME	DATE	Lower boundary of time row was last updated
BATCH	NUMBER	ID of batch in which request is being processed (used for batch DML)

## CTX\_INDEX\_ERRORS

This view displays a row for each document for which ConText indexing failed during a DDL or DML operation. Only users assigned CTXADMIN or CTXAPP can query CTX\_INDEX\_ERRORS.

When the error that caused the indexing for the document to fail is corrected and automatic DML is enabled, once the table that contains the document has been updated, the document is automatically reindexed.

You can also use the CTX\_DML.REINDEX procedure to manually reindex the errored document once the error condition has been corrected.

---

**Note:** Rows in CTX\_INDEX\_ERRORS are not automatically deleted when the errored documents have been corrected and reindexed. The rows must be manually cleared using CTX\_SVC.CLEAR\_ERROR.

---

Column Name	Type	Description
HANDLE	NUMBER	Handle ID for errored document
TSTAMP	DATE	Time stamp
POL_OWNER	VARCHAR2(30)	Username of Policy owner for text column in which errored document is stored
POL_NAME	VARCHAR2(30)	Name of Policy for text column in which errored document is stored
PK	VARCHAR2(64)	Textkey for errored document
ERRORS	VARCHAR2(2000)	Error text for errored document



## CTX\_INDEX\_STATUS

This view displays the reindexing status of requests (rows) in the DML Queue. Only users assigned CTXADMIN or CTXAPP can query CTX\_INDEX\_STATUS.

Column Name	Type	Description
STATUS	VARCHAR2(1)	Status of DML request
POL_OWNER	VARCHAR2(30)	Username of policy owner for DML request
POL_NAME	VARCHAR2(30)	Name of policy for DML request
PK	VARCHAR2(256)	ID for the document being processed by the DML request

## CTX\_LING\_ERRORS

This view displays all the Linguistics requests that have a status of ERROR. Only users assigned CTXADMIN or CTXAPP can query CTX\_LING\_ERRORS.

**Note:** Rows in CTX\_LING\_ERRORS are not automatically deleted when the errored documents have been corrected and reprocessed through the Linguistics. The rows must be manually cleared using CTX\_SVC.CLEAR\_ERROR.

Column Name	Type	Description
HANDLE	NUMBER	Handle ID
TSTAMP	DATE	Time stamp
POL_OWNER	VARCHAR2(2000)	Policy owner for column in which document with errored request is stored
POL_NAME	VARCHAR2(2000)	Policy name for column in which document with errored request is stored
PK	VARCHAR2(64)	Textkey for document with errored request
ERRORS	VARCHAR2(2000)	Error text for errored request

## CTX\_USER\_DML\_QUEUE

This view displays a row for each of the user's entries in the DML Queue. All users can query CTX\_INDEX\_STATUS.

Column Name	Type	Description
CID	NUMBER	Text column ID
POL_NAME	VARCHAR2(30)	Name of policy for text column
SID	VARCHAR2(10)	Identifier for ConText server working on row (value is PENDING if a ConText server is not yet assigned)
PKEY	VARCHAR2(256)	Primary key of row being processed
TIME	DATE	Lower bound of time row was last updated

## CTX\_USER\_DML\_SUM

This view displays the total number of user's entries in the DML Queue for each text column. All users can query CTX\_INDEX\_STATUS.

Column Name	Type	Description
CID	NUMBER	Text column ID
POL_NAME	VARCHAR2(30)	Name of policy for text column
CNT	NUMBER	Count of rows in DML Queue
TSTAMP	DATE	Minimum time stamp for rows

## CTX\_USER\_QUEUE

This view displays all of the rows (pending and in progress requests) in the DML Queue for policies owned by the current user. Only users assigned CTXADMIN or CTXAPP can query CTX\_USER\_QUEUE.

Column Name	Type	Description
CID	NUMBER	Policy ID
SID	VARCHAR2(10)	ID (name) of server processing the request (if the request is pending, this column is NULL)
PKEY	VARCHAR2(256)	Textkey (primary key) of column policy
TIME	DATE	Lower boundary of time row was last updated
BATCH	NUMBER	ID of batch in which request is being processed (for batch DML)

**CTX\_USER\_SVCQ**

This view displays a row for each Linguistics request and each reindexing request (DML or DDL) that has a status of ERROR. Only users assigned CTXADMIN or CTXAPP can query CTX\_USER\_SVCQ.

Column Name	Type	Description
SVCNO	NUMBER	Services Queue number
STATUS	VARCHCHAR2(1)	Request status
SID	NUMBER	Identifier of server that processed request
SERVICE	NUMBER	Requested service
PRIORITY	NUMBER	Priority of request
TSTAMP	DATE	Lower boundary of time row was last updated
USERNAME	VARCHAR2(30)	Policy owner who submitted request
PAR1-10	VARCHAR2(2000)	Parameters (internal use only)
ERRORS	VARCHAR2(2000)	Error text

## ConText Data Dictionary Views

This section describes the views that can be used to query the Oracle ConText objects in the ConText data dictionary.

### CTX\_ALL\_PREFERENCES

This view displays preferences created by ConText users, as well as all the pre-defined preferences included with ConText. The view contains one row for each preference. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
PRE_ID	NUMBER	Preference identifier
PRE_NAME	VARCHAR2(30)	Preference name
PRE_DESC	VARCHAR2(240)	Preference description
PRE_OBJ_NAME	VARCHAR2(30)	Tile specified in preference
PRE_CLA_NAME	VARCHAR2(30)	Category for Tile
PRE_OBJ_ID	NUMBER	Identifier for Tile in preference
PRE_CLA_ID	NUMBER	Identifier for category of preference
PRE_OWNER	VARCHAR2(30)	Username of preference owner

## CTX\_ALL\_SECTIONS

This view displays information about all the sections that have been created in the ConText data dictionary. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
SEC_ID	NUMBER	Identifier for the section
SEC_NAME	VARCHAR2(30)	Name of the section
SGRP_ID	NUMBER	Identifier for the section group to which the section belongs
SGRP_NAME	VARCHAR2(30)	Name of the section group to which the section belongs
SGRP_OWNER	VARCHAR2(30)	Owner of the section group to which the section belongs
START_TAG	VARCHAR2(64)	String of characters that identify the start of the section
END_TAG	VARCHAR2(64)	String of characters that identify the end of the section
TOP_LEVEL	CHAR(1)	Indicates whether the section is a top-level section
ENCLOSE_SELF	CHAR(1)	Indicates whether the section is self-enclosing

## CTX\_ALL\_SECTION\_GROUPS

This view displays information about all the section groups that have been created in the ConText data dictionary. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
SGRP_ID	NUMBER	Identifier for the section group
SGRP_NAME	VARCHAR2(30)	Name of the section group
SGRP_OWNER	VARCHAR2(30)	Owner of the section group

## CTX\_ALL\_THESAURI

This view displays information about all the thesauri that have been created in the ConText data dictionary. It can be viewed by all ConText users.

Column Name	Type	Description
THS_OWNER	VARCHAR2(30)	Username for thesaurus owner
THS_NAME	VARCHAR2(30)	Thesaurus name

## CTX\_CLASS

This view displays all the preference categories registered in the ConText data dictionary. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
CLA_ID	NUMBER(38)	Category identifier
CLA_NAME	VARCHAR2(30)	Category name
CLA_DESC	VARCHAR2(240)	Category description

**CTX\_COLUMN\_POLICIES**

This view displays all policies that have been assigned to a column (non-template policies). It can be viewed only by users with the CTXADMIN roll.

Column Name	Type	Description
POL_ID	NUMBER(38)	Policy identifier
POL_NAME	VARCHAR2(30)	Policy name
POL_DESC	VARCHAR2(240)	Policy description
POL_TABLENAME	VARCHAR2(30)	Name of table to which policy is attached
POL_OWNER	VARCHAR2(30)	Username of policy owner
POL_KEY_NAME	VARCHAR2(256)	Name of textkey column in table containing text column
POL_LN_NAME	VARCHAR2(60)	Name of line number column in table containing text column (only used when master/ detail data store used)
POL_TEXT_EXPR	VARCHAR2(2000)	Name of text column
POL_STATUS	VARCHAR2(12)	Status of policy (VALID or INVALID)



## CTX\_INDEX\_LOG

This view displays all of the details for each indexing operation (index creation, optimization, resumption, etc.) that has been requested. It can be viewed only by users with the CTXADMIN role.

Column Name	Type	Description
TXIL_POL_ID	NUMBER	Policy identifier
TXIL_OPERATION	VARCHAR2(1)	Code which identifies the index operation performed for the policy
TXIL_OPER_STAGE	VARCHAR2(1)	Code which identifies the current stage of the indexing operation
TXIL_RUN_SEQ	NUMBER	Identifier for run in which the index operation was processed
TXIL_SERVER_ID	NUMBER	Identifier for ConText server processing the indexing request
TXIL_START_TIME	DATE	Time and date the index operation was started
TXIL_FIRST_DOC	VARCHAR2(256)	Textkey for first document processed
TXIL_DOC_SELECTED_CNT	NUMBER	Number of documents selected for processing
TXIL_LAST_DOC	VARCHAR2(256)	Textkey for last document processed
TXIL_DOC_PROCESSED_CNT	NUMBER	Number of documents processed
TXIL_MID_FLUSH_FLAG	VARCHAR2(1)	Flag indicating whether a halted index operation is resumed or started over
TXIL_END_TIME	DATE	Time and date the index operation ended
TXIL_CURR_RUN	VARCHAR2(1)	Identifier for current run (internal use only)

**CTX\_OBJECTS**

This view displays all of the ConText Tiles registered in the ConText data dictionary. Only users assigned the CTXAPP and CTXADMIN roles can query CTX\_OBJECTS. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
OBJ_NAME	VARCHAR2(30)	Tile name
CLA_NAME	VARCHAR2(30)	Preference category for Tile (Data Store, Filter, Lexer, etc.)
OBJ_DESC	VARCHAR2(240)	Tile description
OBJ_VALIDATE_ PROC	VARCHAR2(240)	Procedure to call to validate preference settings for Tile
OBJ_IS_ DEFAULT	VARCHAR2(1)	Default Tile for the preference category (Y or N)
OBJ_ID	NUMBER(38)	Tile identifier
CLA_ID	NUMBER(38)	Preference category identifier for Tile

## CTX\_OBJECT\_ATTRIBUTES

This view displays the attributes that can be assigned to each Tile. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
CLA_NAME	VARCHAR2(30)	Preference category for Tile (Data Store, Filter, Lexer, etc.)
CLA_ID	NUMBER(38)	Preference category identifier for Tile
OBJ_NAME	VARCHAR2(30)	Tile name
OBJ_ID	NUMBER(38)	Tile identifier
OAT_NAME	VARCHAR2(64)	Attribute name
OAT_ORDINAL	NUMBER(4)	Order number for attribute
OAT_DATATYPE	VARCHAR2(64)	Attribute datatype
OAT_OPTIONAL	VARCHAR2(2)	Attribute optional (Y or N)
OAT_CARDINALITY	NUMBER	Number of times attribute can be specified in the same preference (4095 for STOP_WORD attribute, 1 for all other attributes)
OAT_DEFAULT_VAL	VARCHAR2(1000)	Default value for attribute
OAT_DESCRIPTION	VARCHAR2(1000)	Description of attribute

**CTX\_OBJECT\_ATTRIBUTES\_LOV**

This view displays the values for the Tile attributes provided by ConText. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
CLA_ID	NUMBER(38)	Preference category identifier
CLA_NAME	VARCHAR2(30)	Preference category for Tile
OBJ_ID	NUMBER(38)	Tile identifier
OBJ_NAME	VARCHAR2(30)	Tile name
OAT_NAME	VARCHAR2(64)	Attribute name
OAL_VALUE	VARCHAR2(64)	Attribute value
OAL_DESCRIPTION	VARCHAR2(64)	Attribute value description

## CTX\_POLICIES

This view displays all of the policies created by ConText users, as well as the template policies included with ConText. The view may contain policies with the same name, because a policy must be unique only for the owner. It can be viewed only by users with the CTXADMIN role.

Column Name	Type	Description
POL_ID	NUMBER(38)	Policy identifier
POL_NAME	VARCHAR2(30)	Policy name
POL_DESC	VARCHAR2(240)	Policy description
POL_TABLENAME	VARCHAR2(30)	Name of table to which policy is attached
POL_OWNER	VARCHAR2(30)	Username of policy owner
POL_KEY_NAME	VARCHAR2(256)	Name of textkey column in table containing text column
POL_LN_NAME	VARCHAR2(60)	Name of line number column in table containing text column (only used when master/ detail data store used)
POL_TEXT_EXPR	VARCHAR2(2000)	Name of text column
POL_STATUS	VARCHAR2(12)	Status of policy (VALID or INVALID)

## CTX\_PREFERENCES

This view displays preferences created by ConText users, as well as all the pre-defined preferences included with ConText. The view contains one row for each preference. It can be viewed only by users with the CTXADMIN role.

Column Name	Type	Description
PRE_ID	NUMBER(38)	Preference identifier
PRE_NAME	VARCHAR2(30)	Preference name
PRE_OWNER	VARCHAR2(30)	Username of preference owner
PRE_DESC	VARCHAR2(240)	Preference description
PRE_OBJ_NAME	VARCHAR2(30)	Tile specified in preference
PRE_CLA_NAME	VARCHAR2(30)	Category for Tile
PRE_OBJ_ID	NUMBER(38)	Identifier for Tile in preference
PRE_CLA_ID	NUMBER(38)	Identifier for category of preference

## CTX\_PREFERENCE\_ATTRIBUTES

This view displays the attributes assigned to all the preferences in the ConText data dictionary. The view contains one row for each attribute. It can be viewed only by users with the CTXADMIN role.

Column Name	Type	Description
PRE_OWNER	VARCHAR2(30)	Username of preference owner
PRE_NAME	VARCHAR2(30)	Preference name
ATT_NAME	VARCHAR2(64)	Name of Tile attribute specified in preference
ATT_VALUE	VARCHAR2(1000)	Value of Tile attribute
ATT_SEQ	NUMBER(38)	Sequence assigned to Tile attribute
ATT_PRE_ID	NUMBER(38)	Identifier for preference
ATT_DESC	VARCHAR2(1000)	Attribute description
ATT_TYPE	VARCHAR2(64)	Attribute type

**CTX\_PREFERENCE\_USAGE**

This view displays the relationship between preferences and policies. The view contains one row for each preference attached to a policy. It can be viewed only by users with the CTXADMIN role.

Column Name	Type	Description
POL_TYPE	VARCHAR2(6)	Policy type
POL_ID	NUMBER(38)	Policy identifier
POL_OWNER	VARCHAR2(30)	Username of policy owner
POL_NAME	VARCHAR2(30)	Policy name
PRE_ID	NUMBER(38)	Preference identifier
PRE_OWNER	VARCHAR2(30)	Username of preference owner
PRE_NAME	VARCHAR2(30)	Preference name

**CTX\_SOURCE**

This view displays all the sources that have been created by ConText users. The view may contain sources with the same name, because a source must be unique only for the owner. It can be viewed by users only with the CTXADMIN role.

Name	Null?	Type
SRC_ID	NUMBER(38)	Source identifier
SRC_NAME	VARCHAR2(30)	Source name
SRC_DESC	VARCHAR2(240)	Source description
SRC_OWNER	VARCHAR2(30)	Username of source owner
SRC_CREATED_BY	VARCHAR2(30)	Username of user who created source
SRC_REFRESH	NUMBER	Refresh rate, in minutes, before a ConText server with the Reader personality checks for new files to be loaded.
SRC_NEXT	DATE	Date and time of the next load (calculated using SYSDATE of last load plus refresh rate)
SRC_SID	NUMBER	Identifier for ConText server processing the source, NULL if no ConText servers are currently processing the source
SRC_CURRENT	VARCHAR2(2000)	Not currently used
SRC_TABLE_NAME	VARCHAR2(30)	Name of the table for the source
SRC_COLUMN_NAME	VARCHAR2(30)	Name of the text column for the source



## CTX\_SQES

This view displays the definitions for all system and session SQEs that have been created by users. It can be viewed only by users with the CTXADMIN role.

Column Name	Type	Description
POL_NAME	VARCHAR2(30)	Policy name for SQE
POL_OWNER	VARCHAR2(30)	Username of policy owner for SQE
POL_ID	NUMBER(38)	Policy identifier for SQE
QUERY_NAME	VARCHAR2(32)	Name of SQE
SESSION_ID	VARCHAR2(32)	Session identifier for SQE ('SYSTEM' for System SQEs)
QUERY_TEXT	VARCHAR2(2000)	Query expression for SQE
TSTAMP	DATE	Date and time SQE was created or last updated

## CTX\_SYSTEM\_PREFERENCES

This view displays all the system-wide preferences (preferences owned by user CTXSYS) registered in the ConText data dictionary. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
PRE_ID	NUMBER(38)	Preference identifier
PRE_NAME	VARCHAR2(30)	Preference name
PRE_DESC	VARCHAR2(240)	Preference description
PRE_OBJ_NAME	VARCHAR2(30)	Tile specified in preference
PRE_CLA_NAME	VARCHAR2(30)	Category for Tile
PRE_OBJ_ID	NUMBER(38)	Identifier for Tile in preference
PRE_CLA_ID	NUMBER(38)	Identifier for category of preference
PRE_OWNER	VARCHAR2(30)	Username for preference owner

## CTX\_SYSTEM\_PREFERENCE\_USAGE

This view displays the relationship between system-wide preferences (preferences owned by CTXSYS) and policies. The view contains one row for each preference owned by CTXSYS attached to a policy. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
POL_ID	NUMBER	Policy identifier
POL_NAME	VARCHAR2(30)	Policy name
PRE_ID	NUMBER	Preference identifier
PRE_NAME	VARCHAR2(30)	Preference name

## CTX\_SYSTEM\_TEMPLATE\_POLICIES

This view displays all the system-wide template policies (template policies owned by user CTXSYS) registered in the ConText data dictionary. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
POL_ID	NUMBER(38)	Policy identifier
POL_NAME	VARCHAR2(30)	Policy name
POL_DESC	VARCHAR2(240)	Policy description
POL_OWNER	VARCHAR2(30)	Username of policy owner

## CTX\_TEMPLATE\_POLICIES

This view displays all the template policies registered in the ConText data dictionary. It can be viewed only by users with the CTXADMIN role.

Column Name	Type	Description
POL_ID	NUMBER(38)	Policy identifier
POL_NAME	VARCHAR2(30)	Policy name
POL_OWNER	VARCHAR2(30)	Username of policy owner
POL_DESC	VARCHAR2(240)	Policy description

## CTX\_USER\_COLUMN\_POLICIES

This view displays all policies that have been assigned to a column (non-template policies) and are owned by the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
POL_ID	NUMBER(38)	Policy identifier
POL_NAME	VARCHAR2(30)	Policy name
POL_DESC	VARCHAR2(240)	Policy description
POL_TABLENAME	VARCHAR2(30)	Name of table to which policy is attached
POL_KEY_NAME	VARCHAR2(256)	Name of textkey column in table containing text column
POL_LN_NAME	VARCHAR2(60)	Name of line number column in table containing text column (only used when master/ detail data store used)
POL_TEXT_EXPR	VARCHAR2(2000)	Name of text column
POL_STATUS	VARCHAR2(12)	Status of policy (VALID or INVALID)

CTX\_USER\_INDEX\_LOG

This view displays all of the details for each indexing operation (index creation, optimization, resumption, etc.) that has been requested by the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
TXIL_POL_ID	NUMBER	Policy identifier
TXIL_OPERATION	VARCHAR2(1)	Code which identifies the index operation performed for the policy
TXIL_OPER_STAGE	VARCHAR2(1)	Code which identifies the current stage of the indexing operation
TXIL_RUN_SEQ	NUMBER	Identifier for run in which the indexing operation was processed
TXIL_SERVER_ID	NUMBER	Identifier for ConText server processing the indexing request
TXIL_START_TIME	DATE	Time and date the index operation was started
TXIL_FIRST_DOC	VARCHAR2(256)	Textkey for first document processed
TXIL_DOC_SELECTED_CNT	NUMBER	Number of documents selected for processing
TXIL_LAST_DOC	VARCHAR2(256)	Textkey for last document processed
TXIL_DOC_PROCESSED_CNT	NUMBER	Number of documents processed
TXIL_MID_FLUSH_FLAG	VARCHAR2(1)	Flag indicating whether a halted index operation is resumed or started over
TXIL_END_TIME	DATE	Time and date the index operation ended
TXIL_CURR_RUN	VARCHAR2(1)	Identifier for current run (internal use only)

## CTX\_USER\_POLICIES

This view displays all policies that are registered in the ConText data dictionary for the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
POL_ID	NUMBER(38)	Policy identifier
POL_NAME	VARCHAR2(30)	Policy name
POL_DESC	VARCHAR2(240)	Policy description
POL_TABLENAME	VARCHAR2(30)	Name of table to which policy is attached
POL_KEY_NAME	VARCHAR2(256)	Name of textkey column in table containing text column
POL_LN_NAME	VARCHAR2(60)	Name of line number column in table containing text column (only used when master/ detail data store used)
POL_TEXT_EXPR	VARCHAR2(2000)	Name of text column
POL_STATUS	VARCHAR2(12)	Status of policy (VALID or INVALID)

## CTX\_USER\_PREFERENCES

This view displays all preferences defined by the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
PRE_ID	NUMBER(38)	Preference identifier
PRE_NAME	VARCHAR2(30)	Preference name
PRE_DESC	VARCHAR2(240)	Preference description
PRE_OBJ_NAME	VARCHAR2(30)	Tile specified in preference
PRE_CLA_NAME	VARCHAR2(30)	Category for Tile
PRE_OBJ_ID	NUMBER(38)	Identifier for Tile in preference
PRE_CLA_ID	NUMBER(38)	Identifier for category of preference
PRE_OWNER	VARCHAR2(30)	Username for preference owner

## CTX\_USER\_PREFERENCE\_ATTRIBUTES

This view displays all the attributes for preferences defined by the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
PRE_OWNER	VARCHAR2(30)	Username of preference owner
PRE_NAME	VARCHAR2(30)	Preference name
ATT_NAME	VARCHAR2(64)	Name of Tile attribute specified in preference
ATT_VALUE	VARCHAR2(1000)	Value of Tile attribute
ATT_SEQ	NUMBER(38)	Sequence assigned to Tile attribute
ATT_PRE_ID	NUMBER(38)	Identifier for preference
ATT_DESC	VARCHAR2(1000)	Attribute description
ATT_TYPE	VARCHAR2(64)	Attribute type

## CTX\_USER\_PREFERENCE\_USAGE

This view displays the preferences that are attached to the policies defined for the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
POL_ID	NUMBER	Policy identifier
POL_NAME	VARCHAR2(30)	Policy name
PRE_ID	NUMBER	Preference identifier
PRE_NAME	VARCHAR2(30)	Preference name
PRE_OWNER	VARCHAR2(30)	Username of preference owner

## CTX\_USER\_SECTIONS

This view displays information about the sections that have been created in the ConText data dictionary for the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
SEC_ID	NUMBER	Identifier for section
SEC_NAME	VARCHAR2(30)	Name of section
SGRP_ID	NUMBER	Identifier for section group to which the section belongs
SGRP_NAME	VARCHAR2(30)	Name of the section group to which the section belongs
START_TAG	VARCHAR2(64)	String of characters that identify the start of the section
END_TAG	VARCHAR2(64)	String of characters that identify the end of the section
TOP_LEVEL	CHAR(1)	Indicates whether the section is a top-level section
ENCLOSE_SELF	CHAR(1)	Indicates whether the section is self-enclosing

## CTX\_USER\_SECTION\_GROUPS

This view displays information about the section groups that have been created in the ConText data dictionary for the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
SGRP_ID	NUMBER	Identifier for the section group
SGRP_NAME	VARCHAR2(30)	Name of the section group

**CTX\_USER\_SOURCES**

This view displays all sources that are registered in the ConText data dictionary for the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Name	Null?	Type
SRC_ID	NUMBER(38)	Source identifier
SRC_NAME	VARCHAR2(30)	Source name
SRC_DESC	VARCHAR2(240)	Source description
SRC_OWNER	VARCHAR2(30)	Username of source owner
SRC_CREATED_BY	VARCHAR2(30)	Username of user who created source
SRC_REFRESH	NUMBER	Refresh rate, in minutes, before a ConText server with the Reader personality checks for new files to be loaded.
SRC_NEXT	DATE	Date and time of the next load (calculated using SYSDATE of last load plus refresh rate)
SRC_SID	NUMBER	Identifier for ConText server processing the source, NULL if no ConText servers are currently processing the source
SRC_CURRENT	VARCHAR2(2000)	Not currently used
SRC_TABLE_NAME	VARCHAR2(30)	Name of the table for the source
SRC_COLUMN_NAME	VARCHAR2(30)	Name of the text column for the source



## CTX\_USER\_SQES

This view displays the definitions for all system and session SQEs that have been created by the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
POL_NAME	VARCHAR2(30)	Policy name for SQE
POL_OWNER	VARCHAR2(30)	Username of policy owner for SQE
POL_ID	NUMBER	Policy identifier for SQE
QUERY_NAME	VARCHAR2(32)	Name of SQE
SESSION_ID	VARCHAR2(32)	Session identifier for SQE ('SYSTEM' for System SQEs)
QUERY_TEXT	VARCHAR2(2000)	Query expression for SQE
TSTAMP	DATE	Date and time SQE was created or last updated

## CTX\_USER\_TEMPLATE\_POLICIES

This view displays all the template policies defined by the current user. It can be viewed only by users with the CTXADMIN or CTXAPP role.

Column Name	Type	Description
POL_ID	NUMBER(38)	Policy identifier
POL_NAME	VARCHAR2(30)	Policy name
POL_DESC	VARCHAR2(240)	Policy description
POL_OWNER	VARCHAR2(30)	Username of policy owner

## CTX\_USER\_THESAURI

This view displays the information about all of the thesauri that have been created in the system by the current user. It can be viewed by all ConText users.

Column Name	Type	Description
THS_OWNER	VARCHAR2(30)	Username for thesaurus owner
THS_NAME	VARCHAR2(30)	Thesaurus name



---

## ConText Index Tables and Indexes

This appendix contains detailed information about the database tables and Oracle indexes that are created by ConText when a text or theme index is created for a column.

The following topics are covered in this appendix:

- ConText Index Tables
- Oracle Indexes for ConText Index Tables
- SQR Table

# ConText Index Tables

ConText index tables are created automatically by ConText during text and theme indexing. The five digit number *nnnnn* is the identifier for the policy that owns the index. Each of the ConText index tables for a policy has the same five digit identifier.

---

**Note:** The ConText index tables are internal tables and should *not* be accessed directly. To perform administrative tasks on a ConText index, use the CTX\_DDL and CTX\_DML packages or the System Administration tool.

---

## DR\_ *nnnnn* \_I1T*n*

This is the main table of a ConText index. It stores each indexed token from the text column, as well as a reference to the documents in which the word occurs and the location of each occurrence.

The *n* appended to the end of the table name is an internal identifier (value of 1 or 2) which ConText uses to prevent table name collisions when two-table compaction or two-table combined real deletion and compaction are used to optimize the ConText index for a table.

---

**Note:** The appended *n* is transparent to users because a synonym called DR\_ *nnnnn* \_I1T, which points to DR\_ *nnnnn* \_I1T*n*, is automatically updated after two-table index optimization.

For more information about two-table index optimization, see “Index Optimization” in Chapter 6, “Text Concepts”.

---

Name	Type	Description
WORD_TEXT	VARCHAR2(64)	Indexed word
FIRST_DOC	NUMBER(38)	DOCID of first document in WORD_INFO
DOCLSIZE	NUMBER(38)	Size, in bytes, of WORD_INFO string
WORD_TYPE	NUMBER(3)	Identifier for type of word (internal use only)
WORD_INFO	LONG RAW	String identifying DOCIDs for all documents in which the indexed word occurs and location of each occurrence.

**DR\_nnnnn\_KTB**

This internal table maps the textkey for each indexed document to a document identifier (DOCID). ConText indexes use DOCIDs internally to identify the documents in which indexed words occur. ConText indexes also use DOCIDs to track documents that have been deleted or modified through a DML action.

Name	Type	Description
TEXTKEY	VARCHAR2(256)	ID for document in text table
DOCID	NUMBER(38)	ID for document in index

**DR\_nnnnn\_LST**

This internal table generates the unique document IDs (DOCID) used in a ConText index. It also stores the next available DOCID and the DOCIDs of documents that have been modified or deleted from the text table.

Name	Type	Description
SID	NUMBER(38)	Audit session ID of the ConText server which is currently creating an index
IDCOUNT	NUMBER(38)	Maximum value for DOCIDs
LTYPE	VARCHAR2(32)	Status of DOCID in index: F = Free, D = Deleted
CONTIGUOUS	NUMBER(2)	Indicates the range of DOCIDs is contiguous for the current indexing (ensures that no overlapping DOCIDs are used in index)
DATA	VARCHAR2(1024)	If LTYPE = F, DOCID for next insert; If LTYPE = D, DOCID for deleted document

**DR\_nnnnn\_NLT**

This internal table is used to optimize DOCID resolution during queries.

Name	Type	Description
FIRST_DOC	NUMBER(38)	Internal use only
IDLIST	LONG RAW	Internal use only

**DR\_nnnnn\_l1W**

This internal table stores each word identified by the Soundex function and the groups to which the word belongs. This table is only created when you index a table with a policy that includes Soundex (*soundex\_at\_index* attribute enabled for the GENERIC WORD LIST Tile).

Name	Type	Description
WORD	VARCHAR2(15)	Word identified by Soundex
GROUP1	VARCHAR2(15)	ID for 1st Soundex group to which word belongs
GROUP2	VARCHAR2(15)	Reserved for future use
GROUP3	VARCHAR2(15)	Reserved for future use

## Oracle Indexes for ConText Index Tables

The Oracle indexes for a ConText index are created automatically by ConText after the index tables have been populated with the ConText index information.

ConText creates a total of five Oracle indexes for the three ConText index tables created during indexing. The Oracle indexes follow the naming conventions used to name the index tables, where the five digit number *nnnnn* is the internal identifier for the policy that owns the ConText index.

ConText Index Table	Oracle Index
DR_#####I1Tn	DR_#####I1In
DR_#####KTB	DR_#####KID
	DR_#####KIK
	DR_#####KSQ
DR_#####LST	DR_#####LIX

# SQR Table

The SQR table is created automatically by ConText during text and theme indexing of a text column; however, the table is not populated until a stored query expression (SQE) is created (stored) for the policy of the text column.

The five digit number *nnnnn* is the identifier for the policy that owns the SQE.

**Note:** The SQR table is an internal table and should *not* be accessed directly. To perform administration tasks on an SQE, use the CTX\_QUERY package or the System Administration tool.

For more information about creating SQEs and using the CTX\_QUERY package, see *Oracle8 ConText Cartridge Application Developer's Guide*.

## DR\_nnnnn\_SQR

This internal table stores the results of an SQE. The definition of the SQE is stored in an internal table owned by CTXSYS.

Name	Type	Description
QUERY_NAME	VARCHAR2(32)	Name of SQE
SESSION_ID	VARCHAR2(32)	SQE type (session or system)
FIRST_DOC	VARCHAR2(38)	DOCID for the first document retrieved by SQE
QUERY_RESULT	LONG RAW	Binary string containing results of SQE

## Oracle Index for DR\_nnnnn\_SQR

During creation of the SQR table, an Oracle index, DR\_nnnnn\_SRI, is created on the table.



---

# Index

## Symbols

---

, query operator, 6-60

## Numerics

---

7-bit character sets  
    using V-Gram lexers with, 7-37  
8-bit character sets, 6-57  
    using V-Gram lexers with, 7-37

## A

---

ACCUMULATE query operator  
    in thesaural expansions, 6-60  
actual deletion  
    of ConText index entries, 6-45, 6-47  
ADD\_SECTION procedure, 11-4  
administration  
    ConText, 1-7  
    ConText servers, 4-5  
    overview of, 1-6  
    Services Queue, 2-18  
    using command-line, 1-10, 3-1, 9-1  
    using GUI tools, 1-11, 3-1, 9-1  
administration tools, 1-11  
    *See also* Configuration Manager, System  
        Administration tool  
Adobe Acrobat format. *See* PDF format  
allocating  
    indexing memory, 7-50  
AmiPro format  
    internal filter, 6-29, 7-19  
    supplied external filter, 6-38

ANSI Z39.19 thesauri, 6-59  
applications  
    sample, 2-5  
ASCII text. *See* plain text  
attributes  
    base\_letter, 7-42  
    binary, 7-27, 7-28  
    code\_conversion, 7-34  
    command, 7-35, 8-9  
    composite, 7-45  
    continuation, 7-42  
    detail\_key, 7-28  
    detail\_lineno, 7-28  
    detail\_table, 7-28  
    detail\_text, 7-28  
    directories, 8-7  
    endjoins, 6-72, 7-44  
    executable, 7-33  
    format, 7-33  
    ftp\_proxy, 7-31  
    fuzzy\_match, 7-57  
    hanzi\_indexing, 7-46  
    http\_proxy, 7-31  
    index\_memory, 7-50  
    instclause, 7-57  
    kanji\_indexing, 7-46  
    keep\_tag, 7-35  
    longsize, 8-8  
    maxdocsize, 7-30  
    maxthread, 7-30  
    maxurls, 7-30  
    no\_proxy, 7-31  
    numgroup, 7-42  
    numjoin, 7-42

- optimize\_default, 7-51
- other\_parms, 7-52
- path, 7-29
- printjoins, 7-43
- punctuations, 7-43
- section\_group, 7-58
- separate, 8-8
- skipjoins, 7-44
- soundex\_at\_index, 7-57
- startjoins, 6-72, 7-44
- stclause, 7-57
- stemmer, 7-57
- stop\_word, 7-59
- storage, 7-51
- tablespace, 7-51
- timeout, 7-29
- urlsize, 7-30
- viewing values for, B-20
- AUTOB predefined indexing preference, 7-19
- automated
  - batch text loading, 6-14, 8-3
  - DML Queue notification, 6-4
- Autorecognize filter, 6-33, 7-19
  - filtering mixed-format columns, 6-32
  - formats supported by, 6-33
  - in Filter preferences, 9-15

## B

---

- base\_letter attribute, 7-42
- base-letter conversion, 6-57
  - in text queries, 6-58
  - in UTF-8 databases, 7-38
  - setting NLS\_LANG for, 6-57
- BASIC LEXER Tile
  - attributes for, 7-41
  - in text indexing policies, 7-4
  - NLS compliance for, 7-38
- BASIC\_HTML\_FILTER predefined indexing
  - preference, 7-20
- BASIC\_HTML\_LEXER predefined indexing
  - preference, 7-21
- BASIC\_HTML\_SECTION predefined section
  - group, 6-74
- BASIC\_HTML\_WORDLIST predefined indexing

- preference, 7-23
- batch mode
  - DML operations, 6-5
  - processing in DML Queue, 2-16
  - text loading using ConText servers, 6-14
  - text loading using ctxload, 6-13
- BFILE datatype, 6-10
- binary attribute
  - for MASTER DETAIL NEW Tile, 7-28
  - for MASTER DETAIL Tile, 7-27
- BLASTER FILTER Tile, 9-15
  - attributes for, 7-33
- BLOB datatype, 6-10
- broader terms
  - generic hierarchy, 6-65
  - in thesauri, 6-64
  - instance hierarchy, 6-65
  - multiple occurrences in a hierarchy, 6-65
  - partitive hierarchy, 6-65

## C

---

- calling
  - procedures within stored procedures, 3-5
  - procedures within triggers, 3-5
- CANCEL procedure, 5-12
  - using, 3-13
- CANCEL\_ALL procedure, 5-13
- CANCEL\_USER procedure, 5-14
- case-sensitivity
  - in composite word indexes, 7-39
  - in stoplists, 6-51, 7-24, 7-25
  - in text indexes, 6-50
  - in theme indexes, 6-54
  - in thesauri, 6-60
- categories
  - See also* indexing Tiles, text loading Tiles
  - Data Store, 7-26
  - Engine (indexing), 7-49
  - Engine (text loading), 8-8
  - Filter, 7-32
  - Lexer, 7-37
  - Reader, 8-7
  - Stoplist, 7-59
  - Translator, 8-9

- Wordlist, 7-54
- CHANGE\_MASK procedure, 5-3
  - using, 3-10
- changing
  - personality masks for ConText servers, 3-10
  - Query pipe buffer size, 5-7
- CHAR datatype, 6-10
- character sets
  - 7-bit, 7-37
  - 8-bit, 6-57, 7-37
  - supported for German composite word indexing, 7-39
- characters
  - base-letter conversion for, 6-57
  - continuation, 7-42
  - in multi-byte languages, 7-37
  - in single-byte languages, 7-37
  - numgroup, 7-42
  - numjoin, 7-42
  - printjoin, 7-43
  - punctuation, 7-43
  - skipjoin, 7-44
  - startjoin and endjoin, 6-72, 7-44
- Chinese language
  - lexers for, 6-49, 7-37
  - pattern matching, 7-46
- CHINESE V-GRAM LEXER Tile
  - attributes for, 7-46
- classes. *See* categories
- CLEAR\_ALL\_ERRORS procedure, 5-15
- CLEAR\_ATTRIBUTES procedure, 11-8
- CLEAR\_ERROR procedure, 5-16, B-6
  - using, 3-13
- CLEAR\_INDEX\_ERRORS procedure, 5-17
- CLEAR\_LING\_ERRORS procedure, 5-18
- CLOB datatype, 6-10
- code\_conversion attribute, 7-34
- column policies
  - See also* policies
  - creating, 11-12
  - viewing, B-25
- columns
  - See also* text columns
  - defining as text columns, 7-3
  - loading text into, 6-3
  - requirements for automated batch loading, 8-3
  - textkey, 9-31
  - with multiple indexes, 6-43
  - with multiple policies, 7-4
- command attribute, 7-35, 8-9
- command-line
  - See also* utilities
  - administration, 1-10, 3-1, 9-1
- compaction
  - in-place, 6-46
  - two-table, 6-46
- composite attribute, 7-45
- composite textkeys, 6-11
  - column length limitations, 6-11
  - column name limitations, 6-11
  - creating policies with, 9-24
  - updating documents containing, 10-6
- composite word indexing
  - enabling for German text, 7-45
  - for German text, 7-39
- compound words
  - in German text, 7-39
- Configuration Manager, 1-11, 3-1, 9-1
- CONTAINS
  - cursor (in-memory queries), 6-8
  - PL/SQL procedure (two-step queries), 6-7
  - SQL function (one-step queries), 6-8, 6-43
- ConText
  - administration, 1-7
  - description, 1-2
  - features, 1-3
  - related publications, xx
- ConText data dictionary
  - creating a preference for a Tile, 11-15
  - description, 6-2
  - querying objects, B-11
  - storing policies, 7-3
  - storing sources, 8-3
  - viewing preference categories, B-13
  - viewing Tiles, B-16
- ConText index tables
  - multiple rows for tokens, 6-44
  - Oracle indexes for, C-5
  - specifying additional parameters for, 7-52
  - specifying STORAGE clauses for, 7-51

- specifying tablespaces for, 7-51
- ConText indexes
  - See also* text indexes, theme indexes
  - compaction, 6-46
  - creating, 9-28, 11-9
  - DDL operations, 6-3, 9-33
  - description, 6-41
  - dropping, 9-25, 9-26, 9-32, 11-21
  - Engine preferences for, 7-49
  - fragmentation, 6-5, 6-44, 6-46, 9-12
  - initialization stage for, 6-43
  - managing, 9-27
  - optimizing, 6-46, 9-32
  - population stage for, 6-43
  - recording index operations, 6-47
  - tables for, 6-42, C-2
  - termination stage for, 6-43
  - themes in, 6-53
  - tuning, 6-44
  - updating, 6-45, 9-31
  - updating using DDL ConText servers, 6-5
- ConText indexing
  - in parallel, 6-45, 9-12, 9-29
  - memory allocation for, 6-43, 6-44, 7-50, 9-12
  - Oracle indexes created during, C-5
  - policy requirements for, 7-3
  - Tiles for, 7-16
- ConText Linguistics. *See* Linguistics
- ConText queues. *See* queues
- ConText roles, 2-3
  - granting to users, 3-4
- ConText server processes. *See* ConText servers
- ConText servers, 1-4, 2-7
  - See also* ctxsrv executable
  - accessing external text files, 6-21
  - accessing the Text Request Queue, 2-13
  - assigning personalities, 4-3
  - checking status of, 4-5
  - failed, 2-11
  - loading text using, 6-14
  - managing, 3-6
  - masking CTXSYS password for, 3-8
  - monitoring status of, B-2
  - personality masks for, 2-9
  - running as background processes, 3-7
  - running multiple servers, 3-7
  - server log, 2-8
  - shutting down, 3-10, 4-2, 4-5, 5-8
  - specifying directories for text loading, 8-7
  - starting, 3-7, 4-2, 4-5
  - starting using ctxctl, 3-8
  - using for parallel indexing, 6-45
  - viewing the status of, 3-9
- ConText users, 2-5
  - See also* users
  - creating, 3-4
  - managing, 3-3
- ConText Workbench
  - administration tools, 1-11
  - defining custom setting configurations, 1-9
  - loading/updating/exporting text using, 9-2
- continuation attribute, 7-42
- control tables, C-3
  - in ConText indexes, 6-42, 6-45
- conventions
  - notational, xxii
- conversion
  - base-letter, 6-57
- CREATE\_INDEX procedure, 11-9
  - using, 9-28
- CREATE\_PHRASE function, 11-48
  - using, 9-36
- CREATE\_POLICY procedure, 11-12
  - creating template policies, 9-25
  - using, 9-22, 9-23
- CREATE\_PREFERENCE procedure, 11-15
  - using, 9-19
- CREATE\_SECTION\_GROUP procedure, 11-16
- CREATE\_SOURCE procedure, 11-17
- CREATE\_TEMPLATE\_POLICY procedure, 11-19
  - using, 9-25
- CREATE\_THESAURUS function, 11-50
  - using, 9-35, 9-36
- creating
  - case-sensitive thesauri, 9-35
  - column policies, 9-22
  - ConText indexes, 9-28
  - ConText users, 3-4
  - Engine preferences (indexing), 9-12
  - Engine preferences (text loading), 9-5

- Filter preferences for external filters, 9-16
- Filter preferences for internal filters, 9-15
- indexes in parallel, 9-29
- Oracle indexes in parallel, 9-29
- phrases in thesauri, 11-48
- policies for detail tables, 6-20
- policies for master tables, 6-19
- policies using composite textkeys, 9-24
- preferences, 9-11
- Reader preferences, 9-4
- section groups, 9-39
- sections, 9-39
- sources, 9-6
- stoplists, 7-59, 9-19
- template policies, 9-25
- text indexing policies, 9-22
- theme indexing policies, 9-23
- Theme Lexer preferences, 9-18
- thesauri, 6-59, 9-35
- thesaurus entries, 9-36
- thesaurus output files, 9-37
- Translator preferences, 9-4, 9-5
- CTX\_ADM package
  - CHANGE\_MASK, 5-3
  - GET\_QUEUE\_STATUS, 5-4
  - RECOVER, 5-6
  - SET\_QUERY\_BUFFER\_SIZE, 5-7
  - SHUTDOWN, 5-8
  - UPDATE\_QUEUE\_STATUS, 5-9
- CTX\_ALL\_DML\_QUEUE view, 3-12, B-4
- CTX\_ALL\_DML\_SUM view, 3-12, B-4
- CTX\_ALL\_PREFERENCES view, B-11
- CTX\_ALL\_QUEUE view, 3-12, B-5
- CTX\_ALL\_SECTION\_GROUPS view, B-12
- CTX\_ALL\_SECTIONS view, B-12
- CTX\_ALL\_SERVERS view, B-2
  - using, 3-9
- CTX\_ALL\_THESAURI view, B-13, B-31
- CTX\_CLASS view, B-13
- CTX\_COLUMN\_POLICIES view, B-14
- CTX\_DDL package, 11-2
  - ADD\_SECTION, 11-4
  - CLEAR\_ATTRIBUTES, 11-8
  - CREATE\_INDEX, 11-9
  - CREATE\_POLICY, 11-12
  - CREATE\_PREFERENCE, 11-15
  - CREATE\_SECTION\_GROUP, 11-16
  - CREATE\_SOURCE, 11-17
  - CREATE\_TEMPLATE\_POLICY, 11-19
  - DROP\_INDEX, 11-21
  - DROP\_INTTTRIG, 11-22
  - DROP\_POLICY, 11-23
  - DROP\_PREFERENCE, 11-24
  - DROP\_SECTION\_GROUP, 11-25
  - DROP\_SOURCE, 11-26
  - OPTIMIZE\_INDEX, 11-27
  - REMOVE\_SECTION, 11-30
  - RESUME\_FAILED\_INDEX, 11-31
  - SET\_ATTRIBUTE, 11-33
  - UPDATE\_POLICY, 11-37
  - UPDATE\_SOURCE, 11-39
  - UPGRADE\_INDEX, 11-36
- CTX\_DML package, 11-41
  - REINDEX, 11-42
  - SYNC, 11-44
  - SYNC\_QUERY, 11-46
- CTX\_INDEX\_ERRORS view, B-6
  - viewing errors for automated batch loading, 6-15
  - viewing URL indexing errors, 6-26
- CTX\_INDEX\_LOG view, 6-48, B-15
- CTX\_INDEX\_STATUS view, B-7
- CTX\_INFO package, 5-21
  - GET\_INFO, 5-22
  - GET\_STATUS, 5-23
  - GET\_VERSION, 5-24
- CTX\_LING package
  - for requesting Linguistics, 3-12
- CTX\_LING\_ERRORS view, 2-19, B-7
- CTX\_OBJECT\_ATTRIBUTES view, B-17
- CTX\_OBJECT\_ATTRIBUTES\_LOV view, B-18
- CTX\_OBJECTS view, B-16
- CTX\_POLICIES view, B-19
- CTX\_PREFERENCE\_ATTRIBUTES view, B-20
- CTX\_PREFERENCE\_USAGE view, B-21
- CTX\_PREFERENCES view, B-20
- CTX\_SERVERS view, B-3
- CTX\_SOURCE view, B-22
- CTX\_SQES view, B-23
- CTX\_SVC package, 3-13, 11-2

- CANCEL, 5-12
- CANCEL\_ALL, 5-13
- CANCEL\_USER, 5-14
- CLEAR\_ALL\_ERRORS, 5-15
- CLEAR\_ERROR, 5-16
- CLEAR\_INDEX\_ERRORS, 5-17
- CLEAR\_LING\_ERRORS, 5-18
- REQUEST\_STATUS, 5-19
- CTX\_SYSTEM\_PREFERENCE\_USAGE view, B-24
- CTX\_SYSTEM\_PREFERENCES view, B-23
- CTX\_SYSTEM\_TEMPLATE\_POLICIES view, B-24
- CTX\_TEMPLATE\_POLICIES view, B-24
- CTX\_THES package, 11-47
  - CREATE\_PHRASE, 11-48
  - CREATE\_THESAURUS, 11-50
  - DROP\_THESAURUS, 11-51
    - for thesauri maintenance, 6-59
- CTX\_USER\_COLUMN\_POLICIES view, B-25
- CTX\_USER\_DML\_QUEUE view, 3-12, B-8
- CTX\_USER\_DML\_SUM view, 3-12, B-8
- CTX\_USER\_INDEX\_LOG view, 6-48, B-26
- CTX\_USER\_POLICIES view, B-27
- CTX\_USER\_PREFERENCE\_ATTRIBUTES view, B-28
- CTX\_USER\_PREFERENCE\_USAGE view, B-28
- CTX\_USER\_PREFERENCES view, B-27
- CTX\_USER\_QUEUE view, 3-12, B-9
- CTX\_USER\_SECTION\_GROUPS view, B-29
- CTX\_USER\_SECTIONS view, B-29
- CTX\_USER\_SOURCES view, B-30
- CTX\_USER\_SQES view, B-31
- CTX\_USER\_SVCQ view, B-10
- CTX\_USER\_TEMPLATE\_POLICIES view, B-31
- CTXADMIN role, 2-4
  - granting to users, 3-4
- CTXAPP role, 2-4
  - granting to users, 3-4
- ctxctl utility, 4-5
  - examples, 4-6
  - shutting down ConText servers, 3-10
  - starting ConText servers, 3-8
  - starting the utility, 4-5
  - syntax, 4-5
  - viewing ConText server status, 3-9
- CTXDEMO user

- privileges, 2-5
- ctxload utility, 6-13, 10-2
  - command-line restrictions, 10-8
  - mandatory arguments, 10-4
  - optional arguments, 10-6
  - syntax, 10-4
  - text load example, 10-9
  - thesaurus export example, 10-10
  - thesaurus import example, 10-10
  - using to create thesauri, 9-35, 9-36
  - using to export thesauri, 9-37
  - using to load file pointers, 9-3
  - using with external data store columns, 9-3
- ctxsrv executable, 4-2
  - See also* ConText servers
  - masking CTXSYS password in, 3-8, 4-2
  - syntax, 4-2
  - using, 3-7
- CTXSYS user, 3-4
  - ConText data dictionary, 6-2
  - masking password in ctxsrv, 3-8, 4-2
  - privileges, 2-5
  - starting ConText servers, 4-2
  - temporary tablespaces, 9-28
  - viewing preferences owned by, B-23
- CTXUSER role, 2-4
  - granting to users, 3-4
- customer support
  - contacting, xxiii

## D

---

- data dictionary. *See* ConText data dictionary
- Data Store Tiles, 7-26
  - list of, 7-26
  - preference example, 7-31
- database administrator. *See* DBA
- database indexes. *See* Oracle indexes
- database pipes. *See* pipes
- database schema objects
  - deleting, 5-6
  - limitations on importing/exporting, 3-3
- database triggers. *See* triggers
- database views. *See* views, 1-1
- datatypes

- for text columns, 6-10
- date format
  - specifying for ctxload, 10-8
- DBA
  - personality, 2-11
  - responsibilities, 2-2
- DDL
  - description, 6-3
  - personality, 2-10, 2-15, 4-3
- DDL operations, 6-3
  - for text indexes, 6-52
  - for theme indexes, 6-56
- DDL pipe
  - disabling/enabling, 3-14
  - requests for DDL operations, 6-3
- default
  - personality mask, 4-3
  - preferences, 7-12
  - stoplist, 7-24
  - thesaurus, 6-60
- DEFAULT\_DIRECT\_DATASTORE predefined
  - indexing preference, 7-17
- DEFAULT\_INDEX predefined indexing
  - preference, 7-22
- DEFAULT\_LEXER predefined indexing
  - preference, 7-21
- DEFAULT\_LOADER predefined text loading
  - preference, 8-6
- DEFAULT\_NULL\_FILTER predefined indexing
  - preference, 7-20
- DEFAULT\_OSFILE predefined indexing
  - preference, 7-17
- DEFAULT\_POLICY template policy, 7-12
- DEFAULT\_READER predefined text loading
  - preference, 8-6
- DEFAULT\_STOPLIST predefined indexing
  - preference, 7-24
- DEFAULT\_TRANSLATOR predefined text loading
  - preference, 8-6
- DEFAULT\_URL predefined indexing
  - preference, 7-17
- deferred deletion
  - of ConText index entries, 6-45
- defining
  - columns as text columns, 7-3
  - thesaurus relationships between terms, 6-59
- deleting
  - See also* dropping
  - database schema objects, 5-6
  - document references in ConText indexes, 6-45, 6-47
  - rows in CTX\_INDEX\_ERRORS, B-6
  - sections, 9-40
  - triggers, 11-22
- derivational stemming
  - enabling for English, 7-57
- detail\_key attribute, 7-28
- detail\_lineno attribute, 7-28
- detail\_table attribute, 7-28
- detail\_text attribute, 7-28
- diacritical marks
  - conversion during text indexing, 6-57
- direct data store, 6-14, 6-16
  - loading text using ctxload, 9-3
- DIRECT Tile, 7-27
- directories
  - ConText servers scanning for text loading, 6-3
  - specifying for text loading, 8-7
  - specifying in external data store, 6-21
- directories attribute, 8-7
- DIRECTORY READER Tile
  - attributes for, 8-7
- disabling
  - pipes, 3-14
  - queues, 3-14, 5-10
- DML
  - description, 6-4
  - personality, 2-10, 2-15, 4-3, 6-5
  - processing, 6-5
- DML operations, 2-16, 11-41
  - automated processing of, 6-4
  - automatic processing of, 6-4
  - batch mode, 2-15, 11-44
  - batch mode processing of, 6-5
  - for text indexes, 6-52
  - for theme indexes, 6-56
  - immediate mode, 2-15, 6-5
  - manual processing of, 6-5
- DML Queue, 2-15
  - batch processing of pending rows, 2-16, 11-44

- disabling/enabling, 2-17, 3-14
- error handling, 2-17
- requests for DML operations, 6-4
- tables, 2-16
- timestamps, 2-17
- updating status, 5-9
- viewing, 2-15, 3-12
- viewing status of, 5-4
- views, B-4, B-5, B-7, B-8, B-9
- documents
  - See also* rows, text
  - accessing through HTTP/FTP, 6-24
  - changes affecting ConText indexes, 6-45
  - exporting, 9-9, 10-3
  - exporting example, 10-10
  - native format storage, 7-32
  - sections in, 6-68
  - stored inside database, 6-17
  - stored outside database, 6-17
  - uniquely identifying using textkeys, 6-10
  - updating, 9-8, 10-3
  - updating example, 10-9
  - with indexed words, C-3
- DR\_nnnnn\_IITn table, 6-45, C-2
- DR\_nnnnn\_IIW table, C-4
- DR\_nnnnn\_KTB table, C-3
- DR\_nnnnn\_LST table, 6-45, C-3
- DR\_nnnnn\_NLT table, C-3
- DR\_nnnnn\_SQR table, C-6
  - Oracle index for, C-6
- DROP\_INDEX procedure, 11-21
  - using, 9-32
- DROP\_INTTRIG procedure, 11-22
- DROP\_POLICY procedure, 11-23
  - using, 9-26
- DROP\_PREFERENCE procedure, 11-24
  - using, 9-20
- DROP\_SECTION\_GROUP procedure, 11-25
- DROP\_SOURCE procedure, 11-26
- DROP\_THESAURUS procedure, 11-51
  - using, 9-37
- dropping
  - ConText indexes, 9-32
  - policies, 9-26
  - preferences, 9-20

- section groups, 9-41
- thesauri, 9-37

## E

---

- embedded text, 10-2
  - in ctxload load files, 10-8, 10-13
- enabling
  - base-letter conversion, 6-57
  - fuzzy matching, 7-54
  - German composite word indexing, 7-45
  - pipes, 3-14
  - queues, 3-14
  - Soundex, 7-55, 7-57
  - stemming, 7-54
- end markers
  - in ctxload load files, 10-11
- end tags
  - for sections, 6-69
- endjoins attribute, 6-72, 7-44
- ENGINE NOP Tile, 7-50
- Engine preferences (indexing)
  - creating, 9-12
  - setting attributes for PARALLEL, 9-29
- Engine preferences (text loading)
  - creating, 9-5
- Engine Tiles (indexing), 7-49
  - list of, 7-49
  - preference example, 7-53
- Engine Tiles (text loading)
  - list of, 8-8
- English language
  - lexers for, 6-49, 7-37
  - predefined stoplist, A-2
- environment variables
  - NLS\_LANG, 6-57
  - TWO\_TASK, 4-2, 10-4
- error handling
  - clearing disabled queues, 5-9
  - DDL operations, 2-14
  - DML Queue, 2-17
  - for automated batch loading, 6-15
  - for URL data store, 6-23, 6-26
  - Services Queue, 2-19
- errored requests



- removing from Services Queue, 3-13
- examples
  - Data Store preference, 7-31
  - Data Store preference for master table, 9-13
  - document exporting, 9-9, 10-10
  - document importing, 9-8
  - document updating, 10-9
  - Engine preference (indexing), 7-53
  - Filter preferences, 7-36, 9-15
  - import file, 10-19
  - Lexer preferences, 7-48
  - section, 9-39, 11-5
  - section group, 9-39
  - Stoplist preference, 7-60, 9-19
  - text loading, 10-9
  - Theme Lexer preference, 9-18
  - thesaurus exporting, 10-10
  - thesaurus importing, 10-10
  - trigger for automated DML notification, 11-11
  - Wordlist preference, 7-58
  - Wordlist preference with section group, 9-40
- executable attribute, 7-33
- executables
  - ctxsrv, 4-2
- EXECUTE privileges
  - granting to users, 3-5
  - packages requiring, 3-5
- expanding
  - queries, 6-60
- exporting
  - documents to OS files, 9-9, 10-3
  - schema objects, 3-3
  - thesauri, 6-59, 9-37, 10-3
- external data store
  - loading text using ctxload, 9-3
- external filters, 6-35
  - executables, 6-30
  - filtering mixed-format columns, 6-32, 6-33
  - indexing text in unsupported formats, 6-30
  - performance, 6-31
  - specifying for mixed-format columns, 7-33
  - specifying for single-format columns, 7-35
  - supplied, 6-38
  - supplied by ConText, 6-38
  - using to perform document cleanup, 6-30

- external text, 6-17
  - directory paths in, 6-21
  - file accessibility for indexing, 6-21
  - file permissions for indexing, 6-21
  - storing as file names, 6-21
  - storing as URLs, 6-22
  - Tile for, 7-29

## F

---

- features
  - ConText, 1-3
- file names
  - in ctxload load file, 10-13
- file pointers, 10-2
  - in load files, 6-13, 10-7
  - storing, 10-2
- file protocol
  - in URL data store, 6-23
- File Transfer Protocol. *See* FTP
- FILTER NOP Tile, 7-34
- Filter Tiles, 7-32
  - list of, 7-32
  - preference example, 7-36
- filtering
  - formatted text, 6-28
  - HTML text, 6-28
  - mixed-format columns, 6-32
  - plain text, 6-28
  - single-format columns, 6-32
- filters
  - external, 6-30, 6-35, 7-33, 7-35
  - internal, 6-28, 7-33
- firewalls. *See* gateway proxies
- format attribute, 7-33
- format ID
  - for document formats, 6-35
  - using in Filter preferences, 9-17
- formatted text
  - filtering, 6-28
  - supported formats for external filters, 6-35
  - supported formats for internal filters, 6-28
- formatting
  - text load files, 10-11
  - thesaurus import files, 10-14

- fragmentation
  - in ConText indexes, 6-44
  - in theme indexes, 6-55
  - through immediate DML processing, 6-5
- French language
  - predefined stoplist, A-3
- FRENCH\_STOPLIST predefined indexing
  - preference, 7-25
- FTP
  - in URLs, 6-23
- ftp\_proxy attribute, 7-31
- functions
  - CREATE\_PHRASE, 11-48
  - CREATE\_THESAURUS, 11-50
  - GET\_QUEUE\_STATUS, 5-4
  - GET\_STATUS, 5-23
  - GET\_VERSION, 5-24
  - REQUEST\_STATUS, 5-19
  - SYNC\_QUERY, 11-46
- fuzzy matching
  - enabling, 7-54
  - specifying a method, 7-57
- fuzzy\_match attribute, 7-57

## G

---

- garbage collection. *See* actual deletion
- gateway proxies
  - accessing the World Wide Web, 6-24
  - for URL data store, 6-24
- generating
  - Gists, 6-9
  - textkeys, 9-7
  - themes, 6-9
- GENERIC ENGINE Tile
  - attributes for, 7-50
- generic hierarchies
  - in thesauri, 6-65
- GENERIC LOADER Tile
  - attributes for, 8-8
- GENERIC STOP LIST Tile, 9-19
  - attributes for, 7-59
- GENERIC WORD LIST Tile
  - attributes for, 7-57
- German language

- composite word indexing, 7-39
  - predefined stoplist, A-4
  - queries for composite words in, 7-40
- GERMAN\_STOPLIST predefined indexing
  - preference, 7-25
- GET\_INFO procedure, 5-22
- GET\_QUEUE\_STATUS function, 5-4, 5-9
- GET\_STATUS function, 5-23
- GET\_VERSION function, 5-24
- Gists, 2-11, 6-9
- GRANT command, 3-5
- granting
  - ConText roles to users, 3-4
  - EXECUTE privileges to users, 3-5
- groups
  - section, 6-71

## H

---

- hanzi\_indexing attribute, 7-46
- header markers
  - in ctxload load file, 10-11
- hierarchical relationships
  - in thesauri, 6-64
  - in thesaurus import file, 10-16
- homographs
  - qualifiers for, 6-66
- HTML
  - filtering tags in, 7-35
  - internal filter, 7-19
  - text, 7-34
  - text filtering for sections, 6-72
  - text stored as URLs, 6-22
- HTML FILTER Tile
  - attributes for, 7-34
- HTML\_FILTER predefined indexing preference, 7-20
- HTTP
  - in URLs, 6-22
- http\_proxy attribute, 7-31
- HyperText Markup Language. *See* HTML
- HyperText Transfer Protocol. *See* HTTP

## I

---

I/O utility  
    in ConText Workbench, 9-2  
immediate DML processing, 6-5  
import files, 10-3, 10-5  
    alternate hierarchy structures in, 10-16  
    examples of, 10-19  
    formatting, 10-14  
    restrictions for thesaural relationships, 10-18  
    restrictions for thesaurus terms, 10-17  
importing  
    schema objects, 3-3  
    thesauri, 6-59, 10-3  
index\_memory attribute, 7-50  
indexes. *See* ConText indexes, Oracle indexes  
indexing  
    *See also* ConText indexing  
    existing columns, 9-31  
    HTML text, 6-28  
    text in unsupported formats, 6-30  
indexing Tiles  
    Data Store, 7-26  
    Engine, 7-49  
    Filter, 7-32  
    Lexer, 7-37  
    Stoplist, 7-59  
    Wordlist, 7-56  
inflectional stemming  
    enabling, 7-57  
initialization parameters  
    TEXT\_ENABLE, 3-2  
    USER\_DUMP\_DEST, 4-3  
initialization stage  
    for index creation, 6-43  
initsid.ora file, 4-3  
in-memory queries, 6-8  
    using composite textkeys, 6-11  
in-place  
    compaction, 6-46  
    deletion, 6-47  
input files  
    for external filters, 6-30  
installing  
    supplied external filters, 6-38

instance hierarchies  
    in thesauri, 6-65  
instclause attribute, 7-57  
internal filters, 6-28, 7-33  
    filtering mixed-format columns, 6-32  
    supported formats, 7-32  
internal tables  
    ConText index tables, 6-42, C-2  
    SQE results, C-6  
internal text  
    storage, 6-16  
    Tile for, 7-27  
intranets  
    indexing using URL data store, 6-24  
ISO-2788 thesauri, 6-59  
issues for external filters, 6-31  
Italian language  
    predefined stoplist, A-5  
ITALIAN\_STOPLIST predefined indexing  
    preference, 7-25

## J

---

Japanese language  
    HTML text conversion, 7-34  
    lexers for, 6-49, 7-37  
    pattern matching, 7-46  
JAPANESE V-GRAM LEXER Tile  
    attributes for, 7-46

## K

---

kanji\_indexing attribute, 7-46  
keep\_tag attribute, 7-35  
Knowledge Catalog, 6-53, 6-54  
Korean language  
    lexers for, 6-49, 7-38  
KOREAN LEXER Tile, 7-47  
KOREAN predefined indexing preference, 7-21  
KOREAN\_WORDLIST predefined indexing  
    preference, 7-23

## L

---

languages

- multi-byte, 7-37
- single-byte, 6-57, 7-37
- supported for fuzzy matching, 7-54
- supported for Soundex, 7-55
- supported for stemming, 7-54
- supported for text indexing, 7-37
- supported for theme indexing, 7-40
- Lexer Tiles, 7-37
  - list of, 7-40
  - preference examples, 7-48
- lexers
  - text indexing, 6-49
  - theme indexing, 6-53
- limitations
  - composite textkeys, 6-11
  - for importing/exporting schema objects, 3-3
  - German composite word indexing, 7-39
  - policies on master tables, 6-19
  - redirection for URL data store, 6-25
  - sections, 6-70
  - supplied external PDF filter, 6-40
  - thesaurus query expansion, 6-61
  - UTF-8 character set, 7-38
- linguistic output, 2-11, 6-9
  - generated by Linguistics, 6-9
  - used in theme indexes, 6-53
- linguistic settings, 6-55
- Linguistics
  - enabling, 3-7
  - personality for, 2-11, 3-7, 4-3
  - processing HTML text through, 6-28
  - requests for, 2-18, 6-9
  - setting up, 1-9
  - viewing errored requests, B-7, B-10
- list of, 7-59
- load files, 6-13, 10-2
  - and the separate option, 10-8
  - creating using translators, 6-14
  - example of embedded text in, 10-13
  - example of file pointers in, 10-13
  - formatting, 10-11
  - restrictions for structure of entries, 10-12
  - restrictions for syntax, 10-12
  - with embedded text, 10-8
- Loader

- personality, 2-9, 4-3, 6-3, 8-7
- loading
  - See also* importing
  - file pointers using ctxload, 8-8
  - text, 6-3, 6-13
  - text into direct data store columns, 9-3
  - text into LONG and LONG RAW columns, 10-2
  - text using ConText servers, 6-14, 9-4
  - text using ctxload, 9-2, 10-2
  - thesauri, 10-3
- LOBs
  - support for, 6-10
- log files
  - for ctxload, 9-3, 10-8
  - for ctxsrv, 4-3
- logs
  - for ConText indexes, 6-47
  - for ConText servers, 2-8
  - using index logs, 9-33
- LONG columns
  - in sources, 8-4
  - loading text into, 10-2
- LONG datatype, 6-10
- LONG RAW columns
  - in sources, 8-4
  - loading text into, 10-2
- LONG RAW datatype, 6-10
- longsize attribute, 8-8
- longsize option (ctxload), 10-7
- Lotus 123 format
  - internal filter, 7-19
  - internal filters, 6-29
  - supplied external filter, 6-38
- Lotus Freelance format
  - supplied external filter, 6-38

## M

---

- maintaining
  - thesauri, 6-59
- managing
  - ConText indexes, 9-27
  - ConText servers, 3-6, 4-5
  - ConText users, 3-3
  - policies, 9-21

- preferences, 9-10
  - queues, 3-11
  - text, 1-8
  - thesauri, 9-34, 9-38
- manual
  - DML Queue notification, 6-5
- mapping tables, C-3
  - in ConText indexes, 6-42
- markers
  - in ctxload load file, 10-11
- MASTER DETAIL NEW Tile, 6-19
  - attributes for, 7-28
- MASTER DETAIL Tile, 6-20
  - attributes for, 7-27
- master-detail
  - in policies, 7-9
  - tables, 6-18
  - text storage, 6-18
- maxdocsize attribute, 7-30
- maximum file size
  - specifying for URL Tile, 7-30
- maximum length
  - specifying for URL Tile, 7-30
- maximum number of rows
  - specifying for URL Tile, 7-30
- maxthread attribute, 7-30
- maxurls attribute, 7-30
- MD\_BINARY predefined indexing preference, 7-18
- MD\_TEXT predefined indexing preference, 7-18
- memory
  - allocating for ConText indexing, 6-43, 6-44
  - allocating for parallel ConText indexing, 6-45
  - in GENERIC ENGINE Tile, 7-50
- Microsoft Excel format
  - supplied external filter, 6-38
- Microsoft PowerPoint format
  - supplied external filters, 6-38
- Microsoft Word format
  - internal filters, 6-29, 7-19
  - supplied external filters, 6-38
- mixed-format columns, 6-32
  - filtering using external filters, 6-33, 9-17
  - filtering using internal filters, 9-15
  - supported formats for, 6-35

- modifying
    - policies, 9-25
- multi-byte languages
  - lexers for, 7-37
- multi-lingual environments
  - using UTF-8 in, 7-38
- multi-threading
  - specifying for URL Tile, 7-30
  - with URL data store, 6-25

## N

---

- narrower terms
  - generic hierarchy, 6-65
  - in thesauri, 6-64
  - instance hierarchy, 6-65
  - multiple occurrences in a hierarchy, 6-65
  - partitive hierarchy, 6-65
- National Language Support (NLS), 6-57
  - compliance with, 7-38
  - enabling support for, 7-38
- newline characters
  - in ctxload load file, 10-12
- NLS\_LANG environment variable
  - base-letter conversion, 6-57
- no\_proxy attribute, 7-31
- NO\_SOUNDEX predefined indexing preference, 7-23
- NO\_STOPLIST predefined indexing preference, 7-25
- nouns
  - decompounding in German text, 7-39
- NULL TRANSLATOR Tile, 8-9
- numgroup attribute, 7-42
- numjoin attribute, 7-42

## O

---

- objects. *See* Tiles
- offsets
  - for tokens in text indexes, 6-50
- one-step queries, 6-8
  - on columns with multiple indexes, 6-43
- operations
  - See also* text operations

- optimize\_default attribute, 7-51
- OPTIMIZE\_INDEX procedure, 11-27
  - using, 9-32
- optimizing
  - ConText indexes, 9-32
  - guidelines for, 6-47
  - indexes, 6-46
- optional attributes
  - for policies, 7-8
- Oracle Corporation
  - customer support, xxiii
- Oracle indexes
  - creating in parallel, 9-29
  - for ConText index tables, C-5
  - for SQE results tables, C-6
  - specifying additional parameters, 7-52
  - specifying PARALLEL clauses for, 9-29
  - specifying STORAGE clauses for, 7-52, 7-57
  - specifying tablespaces for, 7-51
- Oracle servers, 1-4
  - See also* ConText servers
- Oracle8 ConText Cartridge. *See* ConText
- OSFILE Tile, 6-21
  - attributes for, 7-29
- other\_parms attributes, 7-52
- output files
  - for external filters, 6-30

## P

---

- packages. *See* PL/SQL packages
- PARALLEL clause
  - for Oracle indexes, 7-52
  - setting for Oracle indexes, 9-29
- parallel ConText indexing, 6-45, 9-29
  - memory allocation for, 9-12
- parallel creation of Oracle indexes, 9-29
- parallel processing
  - using SYNC, 6-5
- parallel query option, 7-52, 9-29
- parameters
  - for ConText index tables, 7-52
  - for Oracle indexes, 7-52
- partitive hierarchies
  - in thesauri, 6-65
- passwords
  - masking in ctxsrv, 3-8, 4-2
  - supplying through files, 3-8
- path attribute, 7-29
- pattern matching
  - Chinese language, 7-46
  - Japanese language, 7-46
- PDF format
  - limitations, 6-40
  - supplied external filter, 6-38
- pending requests
  - processing using SYNC, 6-5
  - removing from Services Queue, 3-13
- performance, 6-31
- permissions
  - granting permissions for external text files, 6-21
- personalities
  - DBA, 2-11
  - DDL, 2-10
  - DML, 2-10
  - Linguistic, 2-11
  - Loader, 2-9
  - Query, 2-10
- personality masks, 2-9
  - changing, 3-10
  - default, 4-3
- phrases
  - creating in thesauri, 9-36
  - creating relationships for, 9-36
  - in theme indexes, 6-54, 7-40
- pictorial languages
  - lexers for, 6-49, 7-37
  - token recognition for, 6-50
- pipes
  - administration, 2-14
  - DDL, 2-14
  - increasing the size of, 5-7
  - query, 2-14
  - Text Request Queue, 2-14
- PL/SQL
  - reserved words, 3-4, 9-21
- PL/SQL packages
  - CTX\_DDL, 11-2
  - CTX\_DML, 11-41
  - CTX\_INFO, 5-21

- CTX\_SVC, 11-2
- CTX\_THES, 11-47
- granting EXECUTE privileges for, 3-5
- plain text
  - filtering, 6-28
- policies
  - creating, 9-22, 9-23, 11-12
  - creating on detail tables, 6-20
  - creating on master tables, 6-19
  - default preferences, 7-11
  - description, 7-3
  - dropping, 9-26, 11-23
  - for text indexing, 7-4
  - modifying, 9-25
  - multiple on a single column, 7-4
  - names for, 7-8
  - naming restrictions, 9-21
  - optional attributes, 7-8
  - required attributes, 7-8
  - source, 7-9
  - specifying preferences, 7-10, 9-23
  - template, 7-4, B-19
  - theme indexing, 7-5
  - unique names for, 7-10
  - updating, 11-37
  - user-specified preferences, 7-11
  - viewing, B-14, B-19
- population stage
  - for index creation, 6-43
- predefined ConText users, 2-5
- predefined preferences, 7-16, 8-5
  - See also* preferences
- predefined section groups, 6-74
- predefined template policies, 7-11
- preferences
  - creating, 9-11
  - Data Store, 7-26
  - default, 7-11
  - dropping, 9-20, 11-24
  - enabling base-letter conversion, 6-57
  - Engine (indexing), 7-49
  - Engine (text loading), 8-8
  - Filter, 7-32
  - for indexing, 7-15
  - for mixed-format columns, 6-32
  - in policies, 7-10
  - in sources, 8-4
  - Lexer, 7-37
  - predefined, 7-16, 8-6
  - Reader, 8-7
  - Stoplist, 7-59
  - text loading, 8-5
  - Tiles in, 7-16
  - Translator, 8-9
  - viewing, B-20, B-27
  - viewing attributes for, B-20
  - viewing policies for, B-21
  - Wordlist, 7-54
- preferred terms
  - in synonym rings, 6-63
- primary keys, 6-10, 9-31
- printjoins attribute, 7-43
- procedures
  - ADD\_SECTION, 11-4
  - CANCEL, 5-12
  - CANCEL\_ALL, 5-13
  - CANCEL\_USER, 5-14
  - CHANGE\_MASK, 5-3
  - CLEAR\_ALL\_ERRORS, 5-15
  - CLEAR\_ATTRIBUTES, 11-8
  - CLEAR\_ERROR, 5-16
  - CLEAR\_INDEX\_ERRORS, 5-17
  - CLEAR\_LING\_ERRORS, 5-18
  - CREATE\_INDEX, 11-9
  - CREATE\_POLICY, 11-12
  - CREATE\_PREFERENCE, 11-15
  - CREATE\_SECTION\_GROUP, 11-16
  - CREATE\_SOURCE, 11-17
  - CREATE\_TEMPLATE\_POLICY, 11-19
  - DROP\_INDEX, 11-21
  - DROP\_INTRIG, 11-22
  - DROP\_POLICY, 11-23
  - DROP\_PREFERENCE, 11-24
  - DROP\_SECTION\_GROUP, 11-25
  - DROP\_SOURCE, 11-26
  - DROP\_THESAURUS, 11-51
  - GET\_INFO, 5-22
  - OPTIMIZE\_INDEX, 11-27
  - RECOVER, 5-6
  - REINDEX, 11-42

- REMOVE\_SECTION, 11-30
- RESUME\_FAILED\_INDEX, 11-31
- SET\_ATTRIBUTE, 11-33
- SET\_QUERY\_BUFFER\_SIZE, 5-7
- SHUTDOWN, 5-8
- SYNC, 11-44
- UPDATE\_POLICY, 11-37
- UPDATE\_QUEUE\_STATUS, 5-9
- UPDATE\_SOURCE, 11-39
- UPGRADE\_INDEX, 11-36
- processes
  - See also* ConText servers
- processing
  - DML requests in parallel, 6-5
  - HTML text through Linguistics, 6-28
  - text before indexing, 6-30
- protocols
  - URL support for, 6-22, 6-23
- proxies
  - gateway, 6-24
  - specifying for URL Tile, 7-31
- punctuation marks
  - indexing, 6-50
  - lexing, 7-37
- punctuations attribute, 7-43

## Q

---

- qualifiers
  - in thesauri, 6-66
- queries
  - in Text Request Queue, 2-14
  - in-memory, 6-8
  - iterative, 6-8
  - one-step, 6-8
  - SQEs, 6-8
  - two-step, 6-7
- Query
  - personality, 2-10, 4-3
- query expansion
  - using fuzzy matching, 7-54
  - using Soundex, 7-55
  - using stemming, 7-54
  - using thesauri, 6-60
- query methods, 6-7

- query operators
  - ACCUMULATE, 6-60
  - WITHIN, 6-74
- Query pipe
  - disabling/enabling, 3-14
  - increasing buffer size, 5-7
- queues
  - disabling/enabling, 3-14, 5-10
  - managing, 3-11
  - updating, 5-9
  - viewing entries in, B-4
  - viewing status of, 5-4
- quotation marks
  - in ctxload load file, 10-12

## R

---

- Reader Tiles
  - list of, 8-7
- real-time
  - ConText index updates in, 6-5
- RECOVER procedure, 2-12, 5-6
- recovery
  - system, 2-12
- redirection
  - with URL data store, 6-25
- REINDEX procedure, 11-42, B-6
  - manually reindexing documents, 6-5
  - using, 9-31
- related terms
  - in thesauri, 6-67
- relationships
  - defining in thesauri, 9-36
- relevance ranking
  - in theme queries, 6-54
- REMOVE\_SECTION procedure, 11-30
- removing
  - errored requests from Services Queue, 3-13
  - pending requests from Services Queue, 3-13
  - sections from section groups, 9-40
- REQUEST\_STATUS function, 5-19
  - using, 3-12
- requests
  - visible in pending table, 2-16
- reserved words, 3-4, 9-21



- responsibilities
  - DBA, 2-2
  - system administrator, 2-2
- RESUME\_FAILED\_INDEX procedure, 11-31
  - using, 9-33
- resuming
  - failed index operations, 9-33
- roles. *See* ConText roles
- rows
  - deleting from text columns, 6-4
  - deleting in CTX\_INDEX\_ERRORS, B-6
  - inserting into text columns, 6-4
  - multiple entries in index tables, 6-44
  - updating in text columns, 6-4

## S

---

- sample applications, 2-5
- scanning
  - directories, 6-3
- scope notes
  - in thesauri, 6-67
- scoring
  - in queries, 6-7
  - in theme queries, 6-54
- section groups
  - creating, 9-39
  - dropping, 9-41
  - for text queries, 6-71
  - predefined, 6-74
  - specifying for text columns, 7-58
  - viewing, 9-40
- section searching
  - process for enabling, 6-73
  - template policy for, 7-13
  - using document sections, 6-68
  - WITHIN operator for, 6-74
- section\_group attribute, 7-58
- sections, 6-68
  - adding to section groups, 9-39
  - creating, 9-39
  - deleting, 9-40
  - filtering HTML text for, 6-72
  - limitations for, 6-70
  - removing from a section group, 9-40
  - section groups for, 6-71
  - self-enclosing, 6-70
  - start and end tags for, 6-69
  - template policy for, 7-13
  - top-level, 6-70
  - viewing, 9-40
- security
  - masking CTXSYS passwords, 3-8
- self-enclosing sections, 6-70
- separate attribute, 8-8
- separate option
  - for ConText servers with Loader personality, 8-8
  - for ctxload, 10-7
- sequences
  - for external filters, 9-17
  - for generating textkeys, 9-7
  - for stop words, 6-51, 9-19
- servers. *See* ConText servers
- Services Queue, 2-18
  - enabling/disabling, 2-19, 3-14
  - error handling, 2-19
  - querying requests, 11-2
  - removing requests, 5-12, 5-13, 5-14
  - requesting status, 5-19
  - tables, 2-18
  - updating status, 5-9
  - viewing requests in, 3-12
  - viewing status of, 3-12, 5-4
- sessions
  - errors in ctxsrv log, 4-3
  - information in ctxsrv log, 4-3
- SET\_ATTRIBUTE procedure, 11-33
- SET\_QUERY\_BUFFER\_SIZE procedure, 5-7
- setting
  - environment variables for ctxsrv, 4-2, 10-4
  - Query pipe buffer size, 5-7
- setting configurations
  - defining custom, 1-9
- setting up
  - Linguistics, 1-9
  - text, 1-8
- settings
  - linguistic, 6-55
- SHUTDOWN procedure, 5-8

- using, 3-10
- shutting down
  - ConText servers, 3-10
  - ConText servers using ctxctl, 4-6
- single-byte languages, 6-57
  - lexers for, 7-37
- single-format columns, 6-32
  - filtering using external filters, 9-16
  - filtering using internal filters, 9-15
- skipjoins attribute, 7-44
- SOUNDEX
  - predefined indexing preference, 7-23
  - SQL function, 7-55
- Soundex, 7-55
  - enabling, 7-55, 7-57
  - wordlist table, 6-42, C-4
- soundex\_at\_index attribute, 7-57
- source policies. *See* template policies
- sources, 2-9, 6-14
  - attributes, 8-4
  - creating, 8-4, 9-6, 11-17
  - definition, 8-3
  - dropping, 11-26
  - preferences in, 8-4
  - viewing, B-22, B-30
- Spanish language
  - predefined stoplist, A-6
- SPANISH\_STOPLIST predefined indexing preference, 7-25
- specifying
  - additional parameters for Oracle indexes, 9-29
  - directories to scan for text loading, 8-7
  - fuzzy matching method, 7-57
  - language for stemming, 7-57
  - personality mask for ctxsrv, 4-3
  - stop words, 7-59
  - STORAGE clauses for tables/indexes, 7-57
  - tablespaces for Oracle indexes, 7-51
  - translator for text loading, 8-9
- SQE result tables, 6-8, C-6
  - in ConText indexes, 6-42
- SQEs
  - storing results, 6-8
  - viewing, B-31
  - viewing results, 6-8

- SQL
  - ALTER SESSION command, 3-2
  - CREATE USER command, 3-4
  - GRANT command, 3-4
  - reserved words, 3-4, 9-21
- SQL\*Loader, 6-13, 9-3
- start tags
  - for sections, 6-69
- starting
  - ConText servers, 3-7
  - ConText servers on remote machines, 4-2
  - ConText servers using ctxctl, 4-5
  - ConText servers with Loader personality, 9-6
- startjoins attribute, 6-72, 7-44
- status
  - monitoring ConText servers, B-2
  - queues, 5-4, 5-9
  - Services Queue, 5-19
- stclause attribute, 7-57
- stemmer attribute, 7-57
- stemming
  - derivational, 7-57
  - enabling, 7-54
  - inflectional, 7-57
  - specifying a language for, 7-57
  - supported languages, 7-54
- stop words
  - case-sensitivity for, 6-51
  - in text indexes, 6-51
  - in text queries, 6-51
  - maximum number in stoplists, 7-59
  - specifying, 7-59
- stop\_word attribute, 7-59
- Stoplist Tiles, 7-59
  - preference example, 7-60
- stoplists
  - case-sensitivity in, 6-51
  - creating, 7-59, 9-19
  - default, A-2
  - English, A-2
  - French, A-3
  - German, A-4
  - in text indexes, 6-51
  - Italian, A-5
  - maximum size, 9-19

- Spanish, A-6
- supported number of stop words, 7-59
- storage attributes, 7-51
- STORAGE clauses
  - for ConText index tables, 7-52, 9-12
  - for Oracle indexes, 7-52, 7-57
  - for wordlist tables, 7-57
- stored procedures. *See* procedures
- stored query expressions. *See* SQEs
- storing
  - file names in text columns, 6-21
  - internal text, 6-16
  - text as multiple rows, 6-18
  - text in operating system files, 6-17
  - text using MASTER DETAIL, 7-27, 7-28
  - text using OSFILE, 7-29
  - text using URL, 7-29
  - URLs in text columns, 6-22
- SUBMIT procedure
  - in CTX\_LING package, 3-12
- supplied external filters, 6-38
  - installing, 6-38
  - limitations for, 6-40
  - setting up, 6-39
  - using, 9-16, 9-17
  - wrappers for, 6-40
- supported formats
  - for mixed-format columns, 6-35
  - internal filters, 6-28
  - viewing in Windows 32-bit viewer, 6-29
- supported languages
  - for fuzzy matching, 7-54
  - for Soundex, 7-55
  - for stemming, 7-54
  - for text indexing, 7-37
  - for theme indexing, 7-40
- SYNC procedure, 11-44
  - initiating DML processing, 6-5
- SYNC\_QUERY function, 11-46
- synonym rings
  - in thesauri, 6-63
  - preferred terms in, 6-63
  - synonyms in, 6-62
- System Administration tool, 1-11, 3-1, 9-1
  - maintaining thesauri in, 6-59, 9-36

- system administrator
  - responsibilities, 2-2
- system recovery
  - automatic, 2-12
  - manual, 2-12, 5-6

## T

---

- table names
  - collisions, C-2
  - in policies, 9-22
- tables
  - ConText index, C-2
  - DML Queue, 2-16
  - master-detail, 6-18
  - Services Queue, 2-18
  - text columns, 6-10
- tablespace attributes, 7-51
- tablespaces
  - for ConText index tables, 7-51
  - specifying for Oracle indexes, 7-51
  - temporary, 9-28
- tags
  - retaining in HTML text, 7-35
  - start and end, 6-69
- template policies, 7-4
  - See also* policies
  - creating, 9-25, 11-19
  - DEFAULT\_POLICY, 7-12
  - TEMPLATE\_AUTOB, 7-12
  - TEMPLATE\_BASIC\_WEB, 7-13
  - TEMPLATE\_DIRECT, 7-13
  - TEMPLATE\_LONGTEXT\_STOPLIST\_OFF, 7-13
  - TEMPLATE\_LONGTEXT\_STOPLIST\_ON, 7-13
  - TEMPLATE\_MD, 7-14
  - TEMPLATE\_MD\_BIN, 7-14
  - TEMPLATE\_WW6B, 7-14
  - using in policies, 7-9
  - viewing, B-24
- TEMPLATE\_AUTOB template policy, 7-12
- TEMPLATE\_BASIC\_WEB template policy, 7-13
- TEMPLATE\_DIRECT template policy, 7-13
- TEMPLATE\_LONGTEXT\_STOPLIST\_OFF template policy, 7-13
- TEMPLATE\_LONGTEXT\_STOPLIST\_ON template

- policy, 7-13
- TEMPLATE\_MD template policy, 7-14
- TEMPLATE\_MD\_BIN template policy, 7-14
- TEMPLATE\_WW6B template policy, 7-14
- temporary tablespaces
  - CTXSYS user, 9-28
- termination stage
  - for index creation, 6-43
- text
  - examples of loading, 10-9
  - external storage, 6-21, 7-29
  - external storage in operating system files, 6-17
  - external storage with URL pointers, 6-17
  - filtering, 6-27
  - indexing, 6-49
  - internal storage, 6-16, 7-27
  - loading, 2-9, 6-13, 8-7, 8-9, 10-2
  - loading using ConText servers, 6-3
  - preprocessing using external filters, 6-30
  - setting up and managing, 1-8
  - storing as multiple rows, 6-18
- text columns
  - datatypes for external storage, 6-17
  - description, 6-10
  - in policies, 7-8
  - length limitations for composite textkeys, 6-11
  - LOB datatypes in, 6-10
  - naming limitations for composite textkeys, 6-11
  - specifying section groups for, 7-58
  - storing data, 6-16
  - textkeys for, 6-10
  - updating, 6-6
- text indexes
  - See also* ConText indexes
  - case-sensitivity in, 6-50
  - creating, 11-9
  - definition, 6-49
  - DML and DDL processing for, 6-52
  - dropping, 11-21
  - for columns with theme indexes, 6-43
  - stop words in, 6-51
  - tables for, C-2
  - tokens in, 6-50
- text indexing policies, 7-4
- text load files. *See* load files
- text loading Tiles
  - Engine, 8-8
  - Reader, 8-7
  - Translator, 8-9
- text operations
  - DDL, 2-14, 6-3
  - DML, 2-15, 6-4
  - Linguistics requests, 6-9
  - Query, 2-14, 6-7
  - Text Loading, 6-3
  - types of, 2-7
- text queries, 6-7
  - using base-letter conversion, 6-58
- Text Queue. *See* Text Request Queue
- Text Request Queue, 1-4, 2-8, 2-13
- text requests
  - controlling, 5-9
  - pending, 6-5
  - precedence for processing, 2-13
- TEXT\_ENABLE initialization parameter, 3-2
- textkeys, 6-10, 9-31
  - composite, 6-11, 9-24, 10-6
  - generating, 9-6
  - generating using sequences and triggers, 9-7
  - in ctxload load file, 9-7
  - in policies, 7-9
  - mapping to document IDs in ConText indexes, C-3
- theme indexes
  - See also* ConText indexes
  - creating, 11-9
  - DDL and DML processing, 6-56
  - definition, 6-53
  - dropping, 11-21
  - for columns with text indexes, 6-43
  - fragmentation in, 6-55
  - policies for, 7-5
  - tables for, C-2
  - tokens in, 6-54
  - weights in, 6-54
- theme indexing policies, 7-5
- Theme Lexer preferences
  - creating, 9-18
- THEME LEXER Tile, 6-53, 7-40, 7-47, 9-18
- theme queries, 6-7

- theme weights in, 6-54
- theme summaries, 2-11, 6-9
- theme weights
  - in theme indexes, 6-54
- THEME\_LEXER predefined indexing
  - preference, 7-22
- themes, 2-11
  - in linguistic output, 6-9
  - indexing, 6-53, 7-40
- thesauri, 6-59
  - broader term hierarchies, 6-64
  - case-sensitive, 6-60
  - creating, 6-59, 9-35
  - creating case-sensitive, 9-35
  - creating using CREATE\_THESAURUS, 9-35
  - creating using ctxload, 9-35
  - DEFAULT, 6-60
  - dropping, 9-37
  - export example, 10-10
  - exporting, 9-37, 10-3
  - hierarchical relationships, 6-64
  - import example, 10-10
  - importing, 10-3
  - maintaining, 6-59
  - managing, 9-34, 9-38
  - narrower term hierarchies, 6-64
  - qualifiers for entries, 6-66
  - query expansion limitations, 6-61
  - related terms in, 6-67
  - scope notes in, 6-67
  - synonyms in, 6-62
  - synonyms rings in, 6-63
  - types of relationships in, 6-62
  - viewing, B-13
- thesaurus entries
  - creating, 9-36
  - exporting to files, 9-37
  - updating, 9-36
- thesaurus import files. *See* import files
- thesaurus output files
  - creating, 9-37
- Tiles, 7-16
  - assigning multiple attribute values, 9-11
  - BASIC LEXER, 7-41
  - BLASTER FILTER, 7-33
  - CHINESE V-GRAM LEXER, 7-46
  - DIRECT, 7-27
  - DIRECTORY READER, 8-7
  - ENGINE NOP, 7-50
  - FILTER NOP, 7-34
  - GENERIC ENGINE, 7-50
  - GENERIC LOADER, 8-8
  - GENERIC STOP LIST, 7-59
  - GENERIC WORD LIST, 7-57
  - HTML FILTER, 7-34
  - JAPANESE V-GRAM LEXER, 7-46
  - KOREAN LEXER, 7-47
  - MASTER DETAIL, 7-27
  - MASTER DETAIL NEW, 7-28
  - NULL TRANSLATOR, 8-9
  - OSFILE, 7-29
  - THEME LEXER, 7-47
  - URL, 7-29
  - USER FILTER, 7-35
  - USER TRANSLATOR, 8-9
  - viewing attribute values for, B-18
  - viewing attributes for, B-17
- timeout attribute, 7-29
- timeouts
  - with URL data store, 6-25
- timestamps
  - DML Queue, 2-17
- TO\_CHAR date format
  - specifying for ctxload, 10-8
- token tables, C-2
  - in ConText indexes, 6-42, 6-45
- tokens
  - in ConText indexes, C-2
  - in text indexes, 6-50, 7-37
  - in theme indexes, 6-54, 7-40
  - location information in text indexes, 6-50
- tools
  - for GUI administration, 1-11
- top terms
  - in ctxload import file, 10-15
  - in thesauri, 6-64
- top-level sections, 6-70
- trace files
  - for ctxload, 10-8
  - for ctxsrv, 4-3

## Translator Tiles

- list of, 8-9

## translators

- for automated text loading, 6-14
- specifying for text loading, 8-9

## triggers

- creating DML trigger, 11-11
- deleting, 11-22
- for DML operations, 2-15, 6-4
- for generating textkeys, 9-7

## tuning

- ConText indexes, 6-44

TWO\_TASK environment variable, 4-2, 10-4

## two-step queries, 6-7

- using composite textkeys, 6-11

## two-table

- compaction, 6-46, C-2
- deletion, 6-47, C-2

# U

---

Unicode. *See* UTF-8 character set

uniform resource locators. *See* URLs

unique keys, 6-10

UPDATE\_POLICY procedure, 11-37

- using, 9-25

UPDATE\_QUEUE\_STATUS procedure, 5-9

- using, 3-14

UPDATE\_SOURCE procedure, 11-39

## updating

- ConText indexes, 6-5, 9-31
- document references in ConText indexes, 6-45
- documents from OS files, 9-8, 10-3
- text columns during indexing, 6-6

UPGRADE\_INDEX procedure, 11-36

## URL Tile

- attributes for, 7-29
- using, 6-22

## URLs

- data store for, 6-17
- file accessibility, 6-23
- in text columns, 6-22
- support for redirection, 6-25
- using file protocol in, 6-23
- using FTP in, 6-23

- using HTTP in, 6-22, 6-24

urlsize attribute, 7-30

USER\_FILTER Tile, 9-16

- attributes for, 7-35

USER\_TRANSLATOR Tile

- attributes for, 8-9

USER\_DUMP\_DEST initialization parameter

- for ctxload, 10-8

- for ctxsrv, 4-3

user-defined translators

- for automated text loading, 6-14

## usernames

- restrictions, 3-4

## users

- creating, 3-4

- granting ConText roles to, 3-4

- viewing policies for, B-27

- viewing preferences for, B-27

## using

- ctxctl to manage ConText servers, 4-5

- ctxload to import/export thesauri, 10-3

- ctxload to load text, 10-2

- ctxsrv to start ConText servers, 4-2

- external filters, 6-31

- index logs to resume index operations, 9-33

- log files with ctxload, 10-8

- log files with ctxsrv, 4-3

- trace files with ConText servers, 4-3

- trace files with ctxload, 10-8

UTF-8 character set

- limitations, 7-38

- supported languages for, 7-38

## utilities

- ctxctl, 4-5

- ctxload, 10-2

# V

---

VARCHAR datatype, 6-10

variable grammar lexer. *See* V-Gram lexer

V-Gram lexers, 7-37

VGRAM\_CHINESE predefined indexing

- preferences, 7-22

VGRAM\_CHINESE\_WORDLIST predefined

- indexing preference, 7-23

- VGRAM\_JAPANESE predefined indexing preferences, 7-22
- VGRAM\_JAPANESE\_WORDLIST predefined indexing preference, 7-24
- viewing
  - column policies, B-25
  - column policies for a user, B-27
  - ConText server status, 3-9
  - DML Queue, 3-12
  - preference attributes defined by current user, B-28
  - preferences attached to policies, B-28
  - sections and sections groups, 9-40
  - Services Queue, 3-12
  - SQEs, B-31
  - template policies, B-24
  - template policies for current user, B-31
- views, B-2
  - CTX\_ALL\_DML\_QUEUES, B-4
  - CTX\_ALL\_DML\_SUM, B-4
  - CTX\_ALL\_PREFERENCES, B-11
  - CTX\_ALL\_QUEUE, B-5
  - CTX\_ALL\_SECTION\_GROUPS, B-12
  - CTX\_ALL\_SECTIONS, B-12
  - CTX\_ALL\_SERVERS, B-2
  - CTX\_ALL\_THESAURI, B-13
  - CTX\_CLASS, B-13
  - CTX\_COLUMN\_POLICIES, B-14
  - CTX\_INDEX\_ERRORS, B-6
  - CTX\_INDEX\_STATUS, B-7
  - CTX\_INEX\_LOG, B-15
  - CTX\_LING\_ERRORS, B-7
  - CTX\_OBJECT\_ATTRIBUTES, B-17
  - CTX\_OBJECT\_ATTRIBUTES\_LOV, B-18
  - CTX\_OBJECTS, B-16
  - CTX\_PREFERENCE\_ATTRIBUTES, B-20
  - CTX\_POLICIES, B-19
  - CTX\_PREFERENCE\_USAGE, B-21
  - CTX\_PREFERENCES, B-20
  - CTX\_SERVERS, B-3
  - CTX\_SOURCE, B-22
  - CTX\_SQES, B-23
  - CTX\_SYSTEM\_PREFERENCE\_USAGE, B-24
  - CTX\_SYSTEM\_PREFERENCES, B-23
  - CTX\_SYSTEM\_TEMPLATE\_POLICIES, B-24

- CTX\_TEMPLATE\_POLICIES, B-24
- CTX\_USER\_COLUMN\_POLICIES, B-25
- CTX\_USER\_DML\_QUEUE, B-8
- CTX\_USER\_DML\_SUM, B-8
- CTX\_USER\_INDEX\_LOG, B-26
- CTX\_USER\_POLICIES, B-27
- CTX\_USER\_PREFERENCE\_ATTRIBUTES, B-28
- CTX\_USER\_PREFERENCE\_USAGE, B-28
- CTX\_USER\_PREFERENCES, B-27
- CTX\_USER\_QUEUE, B-9
- CTX\_USER\_SECTION\_GROUPS, B-29
- CTX\_USER\_SECTIONS, B-29
- CTX\_USER\_SOURCES, B-30
- CTX\_USER\_SQES, B-31
- CTX\_USER\_SVCQ, B-10
- CTX\_USER\_TEMPLATE\_POLICIES, B-31
- CTX\_USER\_THESAURI, B-31

## W

---

- Web files
  - formats supported by URL data store, 6-22
- Web servers, 6-22, 6-23
- Windows 32-bit viewer
  - supported formats for, 6-29
- WITHIN query operator
  - for section searching, 6-74
- word processing formats, 6-28
- wordlist tables, C-4
  - See also* Soundex
  - in ConText indexes, 6-42
  - specifying STORAGE clauses for, 7-57
- Wordlist Tiles, 7-54
  - list of, 7-56
  - preference example, 7-58
- WordPerfect format
  - internal filters, 6-29, 7-19
  - supplied external filters, 6-38
- World Wide Web
  - administering ConText using, 1-11
  - indexing using URL data store, 6-22
- wrappers
  - for supplied external filters, 6-38
  - in Filter preferences, 6-40
- writing

ConText server output to trace file, 4-3  
WW6B predefined indexing preference, 7-20  
WWW. *See* World Wide Web  
WYSIWYG document viewing, 6-29

## **X**

---

Xerox XIF format  
    internal filter, 6-29, 7-19  
    supplied external filter, 6-38