Oracle8™ ConText® Cartridge

Administrator’s Guide

Release 2.4

July 1998

Part No. A63820-01

ORACLE®

Enabling the Information Age™

Oracle8 ConText Cartridge Administrator’s Guide, Release 2.4

Part No. A63820-01

Release 2.4

Primary Author: D. Yitzik Brenman

Contributors: Dave Allewell, Paul Anderson, Peter Bell, Steve Buxton, Chandu Bhavsar, Jack Chen, Chung-Ho Chen, Yun Cheng, Roy Clarke, Franco Cravero, Paul Dixon, Mohammad Faisal, Elena Huang, Garret Kaminaga, Hassan Karraby, Jeff Krauss, Jacqueline Kud, Kavi Mahesh, Yasuhiro Matsuta ,Colin mcGregor, Josh Powers, Gerda Shank, Dipti Sonak, and Steve Yang.

Contents

Send Us Your Comments

Oracle8 ConText Cartridge Administrator’s Guide, Release 2.4

Part No. A63820-01

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: Oracle8 ConText Cartridge Documentation
- Postal service:
  Oracle Corporation
  Oracle8 ConText Cartridge Documentation
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

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

____

____

____

If you have problems with the software, please contact your local Oracle World Wide Support Center.

Preface

This manual explains how to administer Oracle8 ConText Cartridge and perform the necessary setup and maintenance to allow application developers to develop ConText-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 administer. It could also be an application designer / 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 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 Oracle8, see:

- *Oracle8 Concepts*
- *Oracle8 Administrator's Guide*
- *Oracle8 Utilities*
- *Oracle8 Tuning*

- *Oracle8 SQL Reference*
- *Oracle8 Reference Manual*
- *Oracle8 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

Chapter 2, "Administration Concepts"

This chapter introduces the administration concepts, such as ConText users, roles, servers, and queues, that are necessary for understanding ConText administration.

Chapter 3, "Administering ConText"

This chapter provides detailed instructions and examples for performing administration tasks, such as creating ConText users, assigning ConText roles, starting and stopping ConText servers, and managing the ConText queues, from the command-line.

Chapter 4, "ConText Server Executable and Utility"

This chapter describes the ConText server executable, ctxsrv, and the control utility, ctxctl, and lists the command-line options for both. It also provides syntax examples for ctxsrv and ctxctl.

Chapter 5, "PL/SQL Packages - Administration"

This chapter describes the stored procedures and functions in the PL/SQL packages provided for performing ConText administration.

PART II: TEXT SETUP AND MANAGEMENT

Chapter 6, "Text Concepts"

This chapter introduces the text concepts, such as text columns, text loading, ConText indexes, thesauri, and document sections, that are necessary for understanding text setup and management for enabling ConText queries.

Chapter 7, "Automated Text Loading"

This chapter introduces the concepts required for understanding automated text loading in ConText. It also provides detailed descriptions of the ConText data dictionary objects, such as sources, text loading preferences, and Tiles, that are required for enabling automated text loading.

Chapter 8, "ConText Indexing"

This chapter introduces the concepts required for understanding ConText indexing. It also provides detailed descriptions of the ConText data dictionary objects, such as policies, indexing preferences, and Tiles, that are required for creating ConText indexes.

Chapter 9, "Setting Up and Managing Text"

This chapter provides detailed instructions and examples for performing text setup and management tasks, such as loading text, creating/managing ConText indexes, creating/managing thesauri, and creating/managing document sections, from the command line.

Chapter 10, "Text Loading Utility"

This chapter describes the ConText text loading utility, ctxload, and lists the command-line options for the utility. It also provides syntax examples for the utility.

Chapter 11, "PL/SQL Packages - Text Management"

This chapter lists the stored procedures and functions in the PL/SQL packages provided for setting up and managing text in ConText.

PART III: APPENDICES

Appendix A, "Supplied Stoplists"

This chapter lists the stop words in the language-specific stoplists provided by ConText. It also describes how to create the stoplists.

Appendix B, "ConText Views"

This chapter lists the views that ConText provides for managing ConText servers and queues, as well as the views for the ConText data dictionary.

Appendix C, "ConText Index Tables and Indexes"

This chapter lists the database tables and views that comprise a ConText index.

Appendix D, "External Filter Specifications"

This chapter lists the external filter formats supported by ConText for mixed-format columns, as well as provides installation, setup, and usage details for the external filters supplied by ConText.

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.
lowercase italics	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.
`monospace`	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
Oracle8 ConText Cartridge
Oracle Corporation
500 Oracle Parkway
Redwood Shores, California 94065
Phone: 1.415.506.7000 FAX: 1.415.506.7200

1 Introduction

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

The topics covered in this chapter are:

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

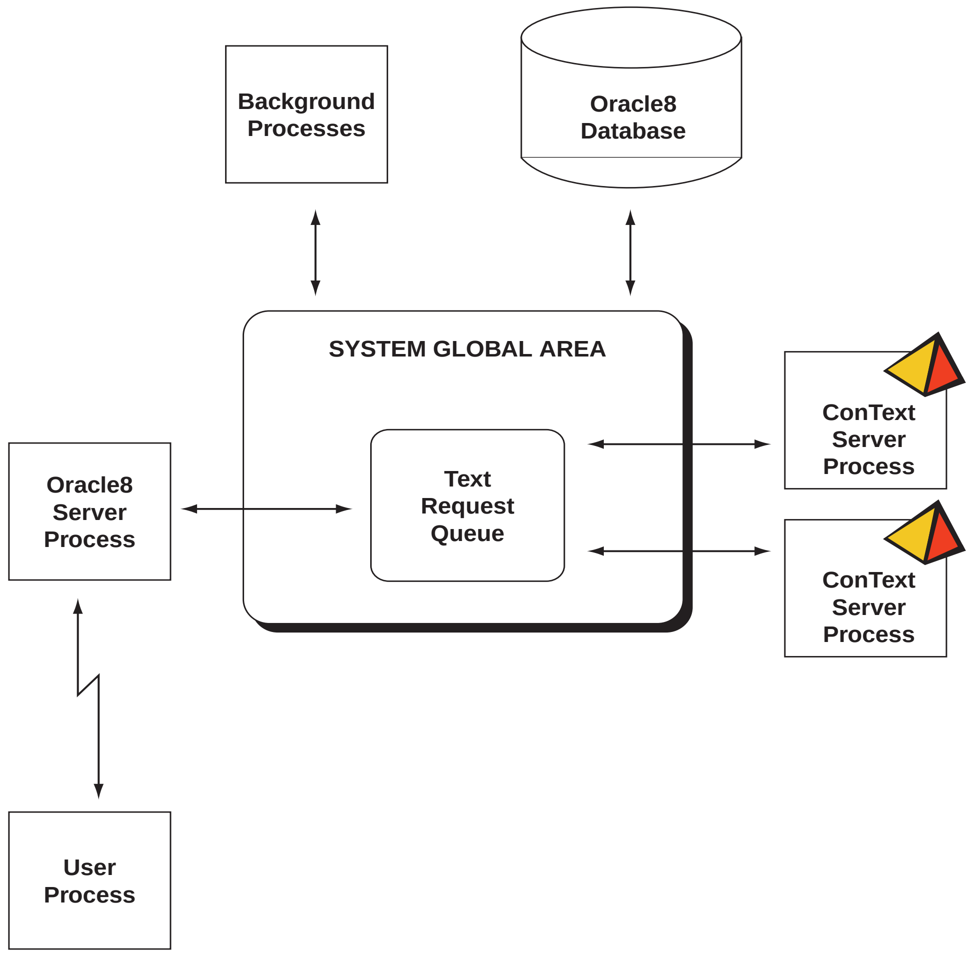
ConText provides advanced indexing, analysis, retrieval, and viewing functionality that can be integrated into any text applications that use Oracle8. The list of features include:

- 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
- support for indexing and querying text stored in columns of all datatypes, including LONG, LONG RAW, and LOBs
- indexing and querying relational tables/views and object tables/views
- 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

Figure 1–1



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 Concepts.*

Overview of ConText Functions

Figure 1–2

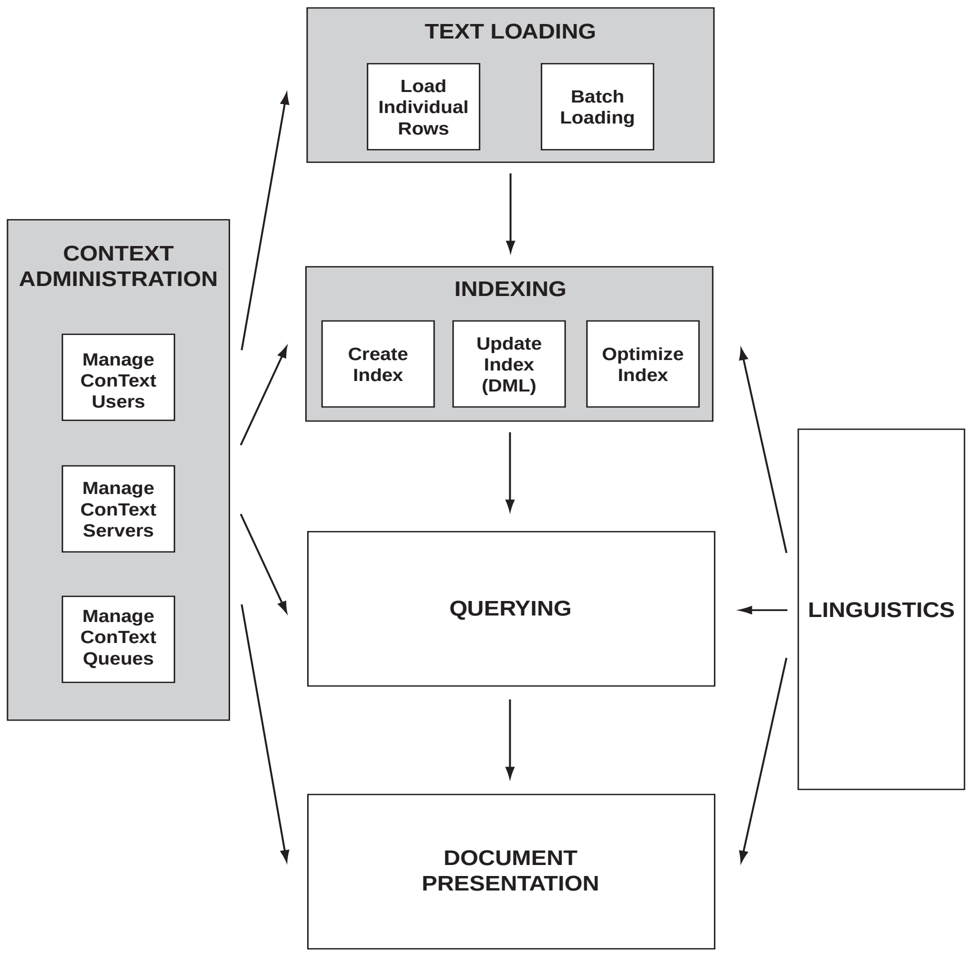


Figure 1–1 illustrates the six main functional areas of ConText:

- ConText Administration
- Text Loading

- Indexing
- Querying
- Document Presentation
- Linguistics

Oracle8 ConText Cartridge Administrator's Guide provides details about and instructions for the tasks that can be performed in ConText administration, text loading, and indexing. These tasks may be performed by a single user or may be divided between different users/responsibilities.

The remaining three areas, querying, document presentation, and Linguistics, are typically part of a ConText application and are described in full in *Oracle8 ConText Cartridge Application Developer's Guide.*

ConText Administration

ConText administration includes the following tasks, which are covered in Part 1 of this manual:

Task	Supported from Command-line?	Supported in GUI Admin Tools?
Enabling one-step queries	Yes	No
Managing ConText users	Yes	Yes
Monitoring and managing ConText servers	Yes	Yes
Monitoring and managing the ConText queues	Yes‘	Yes

See Also: For examples of performing the ConText administration tasks from the command-line, 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 Loading

Text loading includes the following tasks, which are covered in Part II of this manual:

Task	Supported from Command-line?	Supported in GUI Admin Tools?
Loading, updating, and deleting text for individual rows	Yes	No
Exporting text for individual rows	Yes	No
Batch text loading	Yes	No
Enabling automated batch loading through sources	Yes	Yes

See Also: For examples of performing the text loading tasks from the command-line, see Chapter 9, "Setting Up and Managing Text".

For more information about loading text, see Chapter 6, "Text Concepts".

For more information about sources and other text loading objects in the ConText data dictionary, see Chapter 7, "Automated Text Loading".

Who Performs Text Loading?

Text loading may be performed by the ConText administrator because it requires access to system and database resources or it may be performed by application developers.

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

In addition, automated text loading requires ConText servers to be running with the appropriate designation and only ConText administrators can manage ConText servers.

However, an application may provide users with the ability to load/update text for an individual file, which requires the application developer to build the functionality into the application.

Indexing

Indexing includes the following tasks, which are covered in Part II of this manual:

Task	Supported from Command-line?	Supported in GUI Admin Tools?
Defining text storage and indexing options through preferences	Yes	Yes
Identifying text columns through policies	Yes	Yes
Creating ConText text and theme indexes	Yes‘	Yes
Creating and managing thesauri for use in queries	Yes	Yes
Creating and managing document sections to enable section searching in queries	Yes	Yes

See Also: For examples of performing the indexing tasks from the command-line, see Chapter 9, "Setting Up and Managing Text".

For more information about text columns and ConText indexes, see Chapter 6, "Text Concepts".

For more information about preferences, policies, and other text loading objects in the ConText data dictionary, see Chapter 8, "ConText Indexing".

Who Performs Indexing?

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

For example, 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 a index have an effect on how an application retrieves text.

Administration Methods

ConText provides two different methods for performing the administration tasks associated with ConText administration, text loading, and indexing:

- 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 GUI 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 a Microsoft Windows 32-bit environment, such as Windows NT or 95.

> **See Also:** For more information about the GUI administration tools and the other components 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 environment, 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 the CTXSYS user to manage various administration tasks quickly and easily. It also incorporates a simple ad-hoc query tool and a utility for generating Web-based prototype ConText applications.

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

This part provides information specific to ConText administration. It introduces administration concepts such as ConText users, queues, and servers, as well as provides instructions and examples for setting up and administering ConText. It also includes reference information for the ConText executables, utilities, and PL/SQL packages provided for performing these tasks.

This part contains the following chapters:

2 Administration Concepts

This chapter describes the concepts necessary for understanding ConText administration.

The following concepts are discussed in this chapter:

- Administrator Responsibilities
- ConText Roles
- Predefined ConText Users
- ConText Data Dictionary
- ConText Servers
- Personalities
- Text Request 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

ConText provides database roles for performing the following tasks with ConText:

- managing ConText servers, including shut down, monitoring ConText server status, and changing the personality masks for ConText servers
- administering ConText queues, including removing entries from the queues changing the status of the queues, and 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 ConText queries

Three database roles are provided for performing these tasks:

- 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 all the privileges of the lower-level roles, as well as additional privileges.

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

For more information about Oracle users and database roles, see *Oracle8 SQL Reference*.

CTXADMIN Role

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

CTXADMIN should be assigned to Oracle 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 Oracle users who develop ConText applications.

CTXUSER Role

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

Predefined ConText Users

ConText provides two predefined Oracle users:

- CTXSYS User
- CTXDEMO User

CTXSYS User

CTXSYS is created automatically during installation and 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 created automatically if you choose to enable the ConText demos during installation. It 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, which illustrate some typical uses of ConText, can be divided into three groups:

- Sample Populated Database Tables
- Sample SQL Scripts
- Sample Oracle Forms Application

Sample Populated Database Tables

This sample consists of two tables, EMP and DEPT, owned by CTXDEMO. Each table has a text column containing sample plain (i.e. ASCII) English text and a text index associated with the column. These tables can be used to perform ad-hoc text queries to familiarize yourself with ConText's basic features.

The tables and their associated ConText objects are created automatically if you choose to enable the ConText demos when prompted during ConText installation.

> **See Also:** For more information about text columns and text indexes, see Chapter 6, "Text Concepts".
>
> For more information about text queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

Sample SQL Scripts

This sample consists of a populated table, ARTICLES, and two sets of SQL scripts, CTXPLUS and CTXLING, for performing the following tasks:

- creating a text index for ARTICLES
- creating result/output tables for highlighting and ConText Linguistics
- generating Linguistic output for documents in ARTICLES
- performing text queries using different query methods
- viewing documents and highlighted documents
- viewing Linguistic output for documents

The table and its associated ConText objects are *not* created automatically during ConText installation. They must be created after ConText installation using setup scripts provided with the sample scripts.

> **See Also:** For more information about text indexes, see Chapter 6, "Text Concepts".
>
> For instructions on setting up the sample scripts, as well as information about text queries, highlighting, and ConText Linguistics, see *Oracle8 ConText Cartridge Application Developer's Guide*.

Sample Oracle Forms Application

This sample consists of an Oracle Forms application that uses the ARTICLES table (from the SQL scripts sample) to illustrate how to incorporate text queries and document highlighting/viewing into a ConText application.

The Oracle Forms sample is distributed with the ConText Workbench and requires the same setup as the SQL scripts for CTXQUERY.

Note: The Oracle Forms sample uses the same ARTICLES table and ConText objects as the CTXQUERY sample; once setup has been performed for CTXQUERY, you do not need to perform any setup for the Oracle Forms sample.

See Also: For instructions on setting up the CTXQUERY and Oracle Forms samples, as well as information about text queries and highlighting, see *Oracle8 ConText Cartridge Application Developer's Guide*.

For a description of the Oracle Forms sample, including the source code for the sample, see *Oracle8 ConText Cartridge Workbench User's Guide*

ConText Data Dictionary

ConText utilizes a data dictionary, separate from the Oracle data dictionary, to store the ConText objects required for index creation, Linguistic output generation, and automated text loading:

- template policies
- column policies and the table/column to which each column policy is assigned
- indexing preferences (predefined and user-defined) and the Tiles used to create them
- sources for automated batch-loading of text into database columns
- text loading preferences (predefined and user-defined) and the Tiles used to create them

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.

> **See Also:** For more information about policies and indexing preferences, see Chapter 8, "ConText Indexing".
>
> For more information about sources and text loading preferences, see Chapter 7, "Automated Text Loading".

ConText Servers

ConText servers are shared 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 be started *only* from the command-line of the machine on which the ctxsrv executable 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 Executable and Utility".

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 personalities assigned to the server.

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 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 "Personalities" in this chapter.

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

Personalities

A personality for a ConText server indicates the type of text operation that the server can process. A ConText server can be assigned one or more of the following 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

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

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

Personality Masks

The collection of all the personalities for a server is called the personality mask. When a ConText server is started, it is assigned a default personality mask consisting of the DDL (D), DML (M), and Query (Q) personalities. The DBA personality does not appear as part of the personality mask because it is automatically assigned to all ConText servers.

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 "Overview of Automated Loading" in Chapter 7, "Automated Text Loading".

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

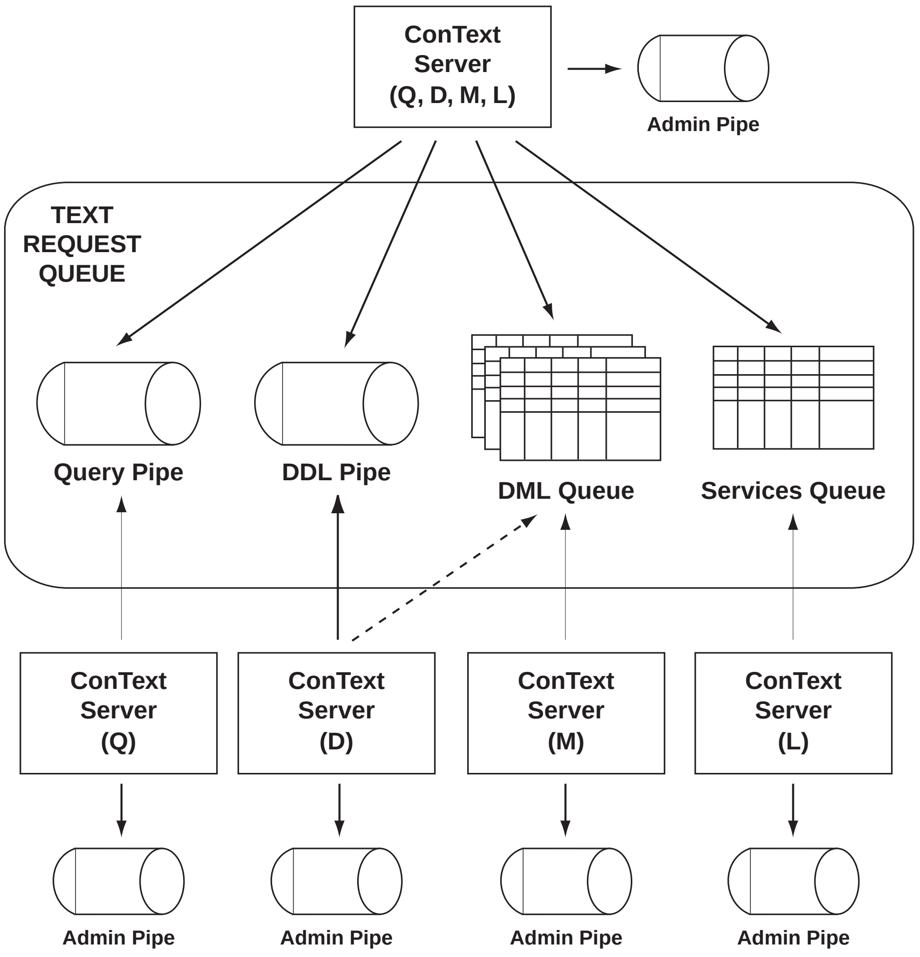
When a table has a text column with an existing ConText index and the table is dropped without first dropping the index and policy for the column, the index tables and policy for the column remain in the system until recovery is performed.

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

System cleanup can also be performed manually using CTX_ADM.RECOVER.

Text Request Queue

Figure 2–1



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

When a client submits a request for a text operation, such as a query, the request is sent to the Text Request Queue. All available ConText servers regularly scan the

Text Request Queue, retrieve pending requests if they have the appropriate personality, and perform the requested operations. Figure 2–1 illustrates how ConText servers with different personalities access the Text Request Queue.

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

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

- Query Pipe
- DDL Pipe
- DML Queue
- Services Queue

In addition, each ConText server has a dedicated administration pipe for processing the administrative tasks that control its operation (e.g. server shutdown).

Query Pipe

When a SQL statement includes a ConText query (text or theme), the system dispatches the query to the query pipe.

To help regulate the flow of query requests, ConText provides a stored procedure, CTX_ADM.SET_QUERY_BUFFER_SIZE, which allocates the amount of memory used by the query pipe. In addition, the pipe can be disabled / enabled using CTX_ADM.UPDATE_QUEUE_STATUS.

DDL Pipe

ConText dispatches all requests for DDL operations (e.g. CREATE_INDEX, DROP_INDEX, and OPTIMIZE_INDEX) to the DDL pipe. The processing of DDL requests is controlled through CTX_ADM.UPDATE_QUEUE_STATUS, which can be used to disable / enable the DDL pipe.

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

When a DML operation is performed (i.e. data in a text table is modified, deleted, or added), an index update request is automatically recorded in the DML Queue. Requests are placed in the DML Queue by internal database triggers that are created during the initial creation of a ConText index for a 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 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".

See Also: For more information about DML operations, see "DML" in Chapter 6, "Text Concepts".

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 stored 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, the corresponding row is deleted from the pending table and the server beginsto update the ConText index. In addition, a new 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 timestamp. 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 timestamp. 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

To control the processing of DML requests, CTX_ADM.UPDATE_QUEUE_STATUS can be used to disable/enable the DML Queue. 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 timestamp. 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

To control the processing of Linguistic requests, CTX_ADM.UPDATE_QUEUE_STATUS can be used to disable/enable the Services Queue. 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.

3 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 init*sid*.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 init*sid*.ora file, see the Oracle8 installation documentation specific to your operating system.

For more information about setting initialization parameters, see *Oracle8 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 init*sid*.ora file:

   ```
   text_enable=true
   ```

3. Restart the database

Note: Once you set TEXT_ENABLE in the init*sid*.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:

Task	Supported in Sys. Admin. Tool?	Supported in Config. Manager?
Creating ConText Users	No	No
Granting ConText Roles to Users	Yes	Yes
Granting EXECUTE Privileges to Application Developers	No	No

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 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 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 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:

Task	Supported in Sys. Admin. Tool?	Supported in Config. Manager?
Starting ConText Servers	No	No
Viewing the Status of ConText Servers	Yes	Yes
Shutting Down ConText Servers	Yes	Yes
Changing the Personality Masks of ConText Servers	Yes	Yes

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 Executable and Utility".
>
> 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 Executable and Utility".

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 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:

Task	Supported in Sys. Admin. Tool?	Supported in Config. Manager?
Viewing the DML Queue	Yes	Yes
Viewing the Services Queue	Yes	Yes
Removing Requests from the Services Queue	Yes	Yes
Enabling and Disabling Queues	No	No

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 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.

4

ConText Server Executable and Utility

This chapter provides reference information for using the ConText server executable and control utility provided with ConText.

The following topics are discussed in this chapter:

- 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

```
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 init*sid*.ora file.

See Also: For more information about SQL trace and the USER_DUMP_DEST initialization parameter, see *Oracle8 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, 1998.  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.

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

```
ctxctl
```

Commands

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
```

5 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

```
set serveroutput on

declare
  ver varchar2(20);
  stat varchar2(20);
begin
  ctx_info.get_info('OCO', ver, stat);
  dbms_output.put_line ('Version is '||ver);
  dbms_output.put_line ('Status is '||stat);
end;
```

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

```
set serveroutput on

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

GET_VERSION

The GET_VERSION function returns the version number for 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

```
set serveroutput on

declare
  ver varchar2(20);
begin
  ver := ctx_info.get_version('OCO');
  dbms_output.put_line ('Version is '||ver);
end;
```

Part II

Text Setup and Management

This part provides information specific to setting up and managing text for enabling ConText queries. It introduces text concepts, such as text loading, ConText indexes, and thesauri, as well as provides instructions and examples for setting up and managing text from the command line. It also includes reference information for the ConText utilities and PL/SQL packages provided for performing these tasks.

This part contains the following chapters:

6 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:

Text Operations

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

- Automated 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 "Personalities" in Chapter 2, "Administration Concepts".

Automated Text Loading

Automated text loading is performed by ConText servers running with the Loader (R) 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 R 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 "Overview of Automated Loading" in Chapter 7, "Automated Text Loading".

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, the 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 determine 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 textkeys 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 8, "ConText Indexing".
>
> For more information about Oracle datatypes, see *Oracle8 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. If the textkey is not specified, ConText uses the first primary key or unique key that it encounters for the table.

Note: ConText fully supports creating indexes on text columns in object tables; however, the object table must have a primary key that was explicitly defined during creation of the table.

For more information about object tables, see *Oracle8 Concepts*.

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 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 (number 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 (number 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 creating ConText indexes and generating 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.*

Individual Row Insert/Update/Export

The method you can use for inserting, updating, or exporting text for individual rows depends on the amount of text to be manipulated and whether the text is formatted.

SQL

For inserting small amounts of plain (ASCII) text into individual rows, you can use the INSERT command in SQL.

For updating individual rows containing small amounts of plain text, you can use the UPDATE command in SQL.

> **See Also:** For more information about the INSERT and UPDATE commands, see *Oracle8 SQL Reference*.

ctxload Utility

For updating individual rows from server-side files containing plain or formatted, you can use the ctxload command-line utility provided by ConText. ctxload is especially well-suited for loading large amounts of text contained in server-side files.

ctxload also allows you to export the contents (plain or formatted text) of the text column for a single row to a server-side file.

Note: If your server environment is Windows NT, you can also use the Input/Output utility for manipulating text in individual rows.

For more information, see "Client-side Insert/Update/Export" in this chapter.

See Also: For an example of updating/exporting an individual row using ctxload, see "Updating/Exporting a Document" in Chapter 9, "Setting Up and Managing Text".

Batch Load

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 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 the LONG or LONG RAW column of 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 consider using 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 Text Load

Automated text loading uses ctxload and ConText servers running with a Loader personality to automatically load text from ctxload load files into text columns of datatype LONG or LONG RAW.

See Also: For more information, see Chapter 7, "Automated Text Loading".

Client-side Insert/Update/Export

Context supports inserting/updating text from files residing on a PC running in a Microsoft Windows 32-bit environment, such as Windows NT or 95. In addition, ConText supports exporting text from individual rows into files on a PC.

ConText provides support for these functions through the Input/Output command-line utility provided with the ConText Workbench.

Note: Client-side text insert/update/export is supported only from 32-bit Windows environments, such as Windows NT or 95.

In addition, the Input/Output utility must be installed on each PC from which text loading/exporting is performed.

See Also: For more information, see *Oracle8 ConText Cartridge Workbench User's Guide*.

ConText Indexes

A ConText index is an inverted index containing entries for all the tokens (words or themes) that occur in a text column and the documents (i.e. rows) in which the tokens are found. The index entries are stored in database tables that are associated with the text column through a policy.

ConText supports creating indexes on text columns in relational tables and views, as well as text columns in object tables. In addition, ConText supports creating two types of indexes, text and theme.

This section discusses the following concepts relevant to ConText indexes:

- Text Indexes
- Theme Indexes
- Text Indexes
- Columns with Multiple Indexes
- Index Creation
- Index Fragmentation
- Memory Allocation
- Thesauri

See Also: For examples of creating policies and indexes, see "Creating a Column Policy" and "Creating an Index" in Chapter 9, "Setting Up and Managing Text".

For more information about policies, see "Policies" in Chapter 8, "ConText Indexing".

Text Indexes

A text index is generated by the text lexers provided by ConText and consists 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 offsets for each occurrence within each document

In addition, if section searching has been enabled for the column, the index stores the section names, as well as the documents in which the section occurs and the location offsets for 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 8, "ConText Indexing".
>
> For more information about section searching, see "Document Sections" in this chapter.

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.

> **See Also:** For more information about the lexers used for text indexing, see "Text Lexers" in Chapter 8, "ConText Indexing".

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 "Text Lexers" in Chapter 8, "ConText 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.

However, the Tile also provides an attribute, *mixed_case*, for creating 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 query performance; however, case-sensitive indexes allow for greater precision in text queries.

Note: The case-sensitivity of a text index determines whether the text queries performed against the index are case-sensitive. If the text index is case-sensitive, text queries are automatically 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.

See Also: For more information about stop words and stoplists, see "Stop Words" in Chapter 8, "ConText Indexing".

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 stop words in text queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

Theme Indexes

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 fifty themes.

Note: Theme indexing is *only* supported for English text.

In addition, offset and frequency information are not relevant in a theme index, so this type of information is not stored.

See Also: 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, 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.*

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, the entry would be *Oracle Corporation*, which is the canonical form of *Oracle Corp.*, as stored in the Knowledge Catalog.

> **See Also:** For more information about themes and the Knowledge Catalog, see *Oracle8 ConText Cartridge Application Developer's Guide*.

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 whether the theme is found in the Knowledge Catalog:

- if the theme is in the Knowledge Catalog, the case for the index entry matches the canonical form of the theme in the Knowledge Catalog
- if the theme is not in the Knowledge Catalog, the case for the index 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. 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.

You can create your own custom setting configurations in either of the GUI administration tools provided in the ConText Workbench.

> **See Also:** For more information about Linguistics, see *Oracle8 ConText Cartridge Application Developer's Guide*.

ConText Index Tables

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

- DR_*nnnnn*_I1T*n* (token table)
- DR_*nnnnn*_KTB (textkey mapping table)
- DR_*nnnnn*_LST (DOCID generation table)
- DR_*nnnnn*_NLT (DOCID control table)
- DR_*nnnnn*_I1W (Soundex wordlist table -- created only if Soundex is enabled)
- DR_*nnnnn*_SQR (stored query expression result table)

The *nnnnn* 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.*

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 Creation

A ConText index is created for a column by calling CTX_DDL.CREATE_INDEX for the column policy; however, before calling CREATE_INDEX, a ConText server must be running with the DDL (D) personality.

> **See Also:** For more information, see "ConText Servers" in Chapter 2, "Administration Concepts".

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 a list of the tables used to store the ConText index, see "Text Indexes" 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.

Creating Empty ConText Indexes

If you want to create a ConText index 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.

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 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 a index fragment. The aggregate of all the rows for a word/theme represents the complete index entry for the word/theme.

Note: 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 a concern in theme indexes as in text indexes; however, some fragmentation may occur during theme indexing and subsequent DML.

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 more information about allocating memory for ConText indexing, see "Creating an Engine Preference" in Chapter 9, "Setting Up and Managing Text".

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).

Index Updates (DML)

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

Text index updates are processed by ConText servers with the DML or DDL personality, depending on the DML index update method (immediate or batch) that is currently enabled.

Note: In contrast to requests for theme and Gist/theme summary generation, which are processed by ConText servers with the Linguistic personality, updates to theme indexes are processed identically to text indexes, using ConText servers with the DML personality.

Immediate Vs. Batch Update

If immediate index update is enabled, ConText servers with a DML personality regularly scan the DML Queue and process update requests as they come into the queue.

If batch index update is enabled, no ConText servers with a DML personality are running and update requests in the DML Queue are processed by ConText servers with a DDL personality only when explicitly requested.

See Also: For more information about DML index update methods, see "DML" in this chapter.

For more information about ConText servers, see "Personalities" in Chapter 2, "Administration Concepts".

Deferred Deletion

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_I1Tn) for the modified / deleted document are not actually removed. Instead, the status of the document is recorded in the ConText index DOCID control table (DR_nnnnn_NLT), 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 (I1T*n*) takes place only during optimization of a index.

> **See Also:** For more information, see "Removal of Obsolete Document References" in "Index Optimization" in this chapter.

Index Optimization

ConText supports index optimization for improving query performance. Optimization performs two functions for an index:

- Compaction of Index Fragments
- Removal of Obsolete Document References

ConText supports index optimization through the CTX_DDL.OPTIMIZE_INDEX procedure.

Note: Optimization cannot be performed for an index while any other DDL or DML operation is being performed on the index. Likewise, while optimization is being performed on an index, no DML operations can be performed on the index.

As such, it may not always be practical to optimize an entire index. ConText also supports piecewise optimization for individual index entries (words and sections) that are stored in the index

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. 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.

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 Obsolete Document References

ConText provides optimization methods which can be used to actually delete all references to modified / deleted documents from an index.

During an actual delete (also referred to as garbage collection), 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, the ConText index DOCID control table (DR_nnnnn_NLT) is cleared of the information which records the status of documents.

Similar to compaction, ConText supports both in-place or two-table garbage collection.

Piecewise Optimization

Index optimization can be performed piecewise for individual words in a ConText index (text or theme). Because it is generally faster than optimizing an entire index, piecewise optimization is useful when an index has a large number of index fragments or obsolete document references and it is not practical to block DML on the index while optimization is performed.

Note: While text index entries consist only of single words, theme index entries often consist of phrases as well as single words.

For documentation purposes, the term *word* refers to words and phrases with regards to piecewise optimization for theme indexes.

Piecewise optimization is specified for a word using arguments in CTX_DDL.OPTIMIZE_INDEX. Piecewise optimization supports only one type of optimization: combined compaction / garbage collection performed in-place.

Note: Piecewise garbage collection for a word only removes obsolete document references from the corresponding entries (rows) in the token table; obsolete references are retained in the entries for other words in the index, as well as in DR_nnnnn_NLT, which ensures that the other entries are not affected by the piecewise optimization.

To remove obsolete document references from all the entries in an index, garbage collection must be performed for the entire index.

See Also: For an example of piecewise optimization, see "Optimizing an Index" in Chapter 9, "Setting Up and Managing Text".

Optimizing Index Entries for Tokens and Sections

The word to be optimized can have two types of entries in the index: token and section.

Token entries consist of a word (and its location information) that occurs in one or more documents in a text column. Section entries, found only in text indexes, consist of the name (and location information) for a section that occurs in one or more documents in the column.

If the word to be optimized has token entries in the index, all the token entries (rows) corresponding to the word are combined into as few rows as possible and all obsolete document references are removed from the location strings for the rows.

If the word to be optimized has section entries in the index, all the section entries (rows) corresponding to the word are combined into as few rows as possible and all obsolete document references are removed from the location strings for the rows.

If the word to be optimized has both types of entries in the index, ConText optimizes all the entries for both types in a single pass; however, ConText optimizes the different types of entries as separate, distinct entities.

Case-sensitivity

Piecewise optimization is case-sensitive regardless of the case of the index, meaning index entries for a word are optimized only if the entries exactly match the word specified for piecewise optimization.

This feature is of particular importance for piecewise optimization in theme indexes, because theme indexes are always case-sensitive and the index entries often consist of phrases in mixed case.

For example, a theme index contains separate token entries for the word *oracle* and the phrase *Oracle Corporation*. If piecewise optimization is specified for the phrase *Oracle Corporation*, only those entries that exactly match the phrase are optimized; entries for *oracle* are not optimized. In addition, if piecewise optimization is specified for the word *Oracle*, no entries are optimized.

Identifying Candidates for Piecewise Optimization

The *word_text* and *doclsize* columns in the index token table (DR_nnnnn_I1Tn) can be queried to identify words that are potential candidates for piecewise optimization. Also note that the *word_type* column in the table identifies whether the row serves as a token entry or a section entry.

In general, if *word_text* returns a large number of rows for a word and/or the *doclsize* for many of the rows is significantly less than 64 Kilobytes (the maximum size of the location string for an index entry), the word is a good candidate for compaction.

When to Optimize

Index optimization should be performed regularly, as index creation and frequent updating can result in excessive fragmentation and accumulation of obsolete document references. The level of fragmentation for an index depends on the amount of memory allocated for indexing and the amount of text being indexed. The number of obsolete document references in an index depends on the frequency of DML for documents in the column and the degree of DML changes for the documents.

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/deleted

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.

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.

Note: The ConText thesauri formats and functionality are compliant with both the ISO-2788 and ANSI Z39.19 (1993) standards.

See Also: For more information about the relationships you can define for terms in a thesaurus, see "Thesaurus Entries and Relationships" in this chapter.

Thesaurus Creation and Maintenance

Thesauri and thesaurus entries can be created, modified, and deleted by all ConText users with the CTXAPP role.

ConText supports thesaurus maintenance from the command line through the PL/SQL package, CTX_THES. ConText also supports GUI viewing and administration of thesauri in the System Administration tool.

Note: The CTX_THES package calls an internal package, CTX_THS, which should *not* be called directly.

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 about command line administration of thesauri, see "Managing Thesauri" in Chapter 9, "Setting Up and Managing Text".

For more information about GUI administration of thesauri, see the help system provided with the System Administration tool.

For more information about ctxload, see Chapter 10, "Text Loading Utility".

Thesauri in Queries

Thesauri are primarily used for expanding the query terms in text queries to include entries that have been defined as having relationships with the terms in the specified thesaurus.

Thesauri can be used for expanding theme queries; however, expansion of theme queries is generally not needed, because ConText uses an internal lexicon, called the Knowledge Catalog, to automatically expand theme queries.

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*.

Query Expansion

The expansions returned by the thesaurus operators in queries are combined using the ACCUMULATE operator (,).

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).

Case-sensitivity

ConText thesauri supports creating case-sensitive and case-insensitive thesauri.

Case-sensitive Thesauri

In a case-sensitive thesaurus, terms (words and phrases) are stored exactly as entered. For example, if a term is entered in mixed-case (using either CTX_THES, the System Administration tool, or a thesaurus load file), the thesaurus stores the entry in mixed-case.

In addition, when a case-sensitive thesaurus is specified in a query, the thesaurus lookup uses the query terms exactly as entered in the query. As a result, queries that use case-sensitive thesauri allow for a higher level of precision in the query expansion performed by ConText.

For example, a case-sensitive thesaurus is created with different entries for the distinct meanings of the terms *Turkey* (the country) and *turkey* (the type of bird). Using the thesaurus, a query for *Turkey* expands to include only the entries associated with *Turkey.*

Case-insensitive Thesauri

In a case-insensitive thesaurus, terms are stored in all-uppercase, regardless of the case in which they were entered.

In addition, when a case-insensitive thesaurus is specified in a query, the query terms are converted to all-uppercase for thesaurus lookup. As a result, ConText is unable to distinguish between terms that have different meanings when they are in mixed-case.

For example, a case-insensitive thesaurus is created with different entries for the two distinct meanings of the term *TURKEY* (the country or the type of bird). Using the thesaurus, a query for either *Turkey* or *turkey* is converted to *TURKEY* for thesaurus lookup and then expanded to include all the entries associated with both meanings.

Default Thesaurus

If you do not specify a thesaurus by name in a query, by default, the thesaurus operators use a thesaurus named *DEFAULT*; however, because the entries in a thesaurus may vary greatly depending on the subject matter of the documents for which the thesaurus is used, ConText does not provide a *DEFAULT* thesaurus.

As a result, if you want to use a default thesaurus for the thesaurus operators, you must create a thesaurus named *DEFAULT*. You can create the thesaurus through any of the thesaurus creation methods supported by ConText:

- System Administration tool (GUI)
- CTX_THES.CREATE_THESAURUS (PL/SQL)
- ctxload

Supplied Thesaurus

Although ConText does not provide a default thesaurus, ConText does supply a thesaurus, in the form of a ctxload load file, that can be used to create a general-purpose, English-language thesaurus.

The thesaurus load file can be used to create a default thesaurus for ConText or it can be used as the basis for creating thesauri tailored to a specific subject or range of subjects.

See Also: For more information about using ctxload to create the thesaurus, see "Creating the Supplied Thesaurus" in Chapter 9, "Setting Up and Managing Text".

Supplied Thesaurus Structure and Content

The supplied thesaurus is similar to a traditional thesaurus, such as Roget's Thesaurus, in that it provides a list of synonymous and semantically related terms, sorted into conceptual domains.

The supplied thesaurus provides additional value by organizing the conceptual domains into a hierarchy that defines real-world, practical relationships between narrower terms and their broader terms.

Additionally, cross-references are established between domains in different areas of the hierarchy. At the lower levels of the hierarchy, synonym rings are attached to domain names.

Supplied Thesaurus Location

The exact name and location of the thesaurus load file is operating system dependent; however, the file is generally named 'dr0thsus' (with an appropriate extension for text files) and is generally located in the following directory structure:

```
<Oracle_home_directory>
    <ConText_directory>
        thes
```

See Also: For more information about the directory structure for ConText, see the Oracle8 installation documentation specific to your operating system.

Thesaurus Entries and Relationships

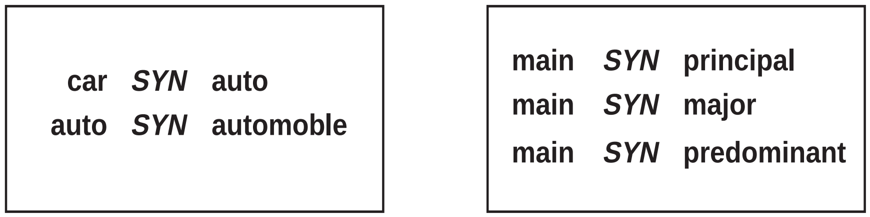
Three types of relationships can be defined for entries (words and phrases) in a thesaurus:

- Synonyms
- Hierarchical Relationships
- Related Terms

In addition, each entry in a thesaurus can have Scope Notes associated with it.

Synonyms

Figure 6–1



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.

Synonym entries support the following relationships:

- 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 *predominant* 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.

While synonym rings are not explicitly named, they have an ID associated with them. The ID is assigned when the synonym entry is first created.

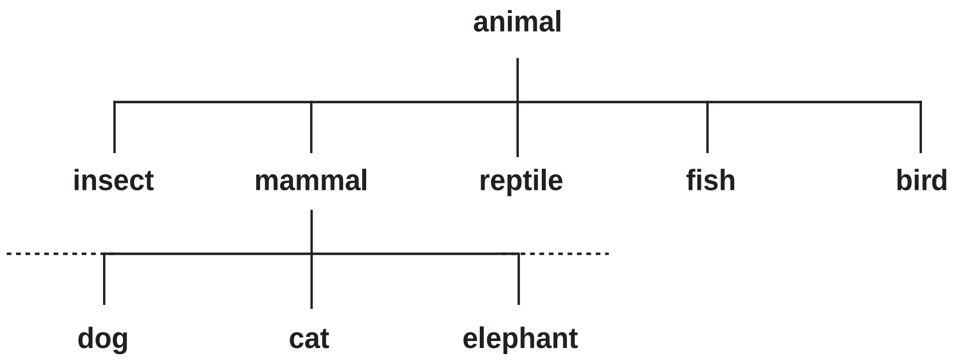
Preferred Terms

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

Figure 6–2



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*.

In addition to the standard hierarchy, ConText also supports the following specialized 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 narrower generic 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 narrower partitive 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 narrower instance 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, in this case, each occurrence of the term in the hierarchy 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: This example uses the generic hierarchy. The same expansions hold true for the standard, partitive, and instance hierarchies.

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. In particular, they can be used to provide information about the usage/function of the entry or to distinguish the entry from other entries with similar meanings.

Document Sections

ConText enables users to increase query precision using structure (i.e. sections) found in most documents. The most common structure found in documents is the grouping of text into sentences and paragraphs. In addition, many documents create structure through the use of tags or regularly-occurring fields delimited by strings of repeating text.

For example, World Wide Web documents use HTML, a defined set of tags and codes, to identify titles, headers, paragraph offsets, and other document meta-information as part of the document content. Similarly, e-mail messages often contains fields with consistent, regularly-occurring headers such as *subject:* and *date:*.

For each text column, users can choose to define rules for dividing the documents in the column into user-defined sections. In addition, for text columns that use the BASIC LEXER Tile, users can enable section searching for sentences and paragraphs. ConText includes section information as entries (rows) in the 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 sections and enabling section searching for theme indexes is not supported.

In addition, because section information is stored in the text index, sections must be defined, if desired, and section searching must be enabled before text index creation. If you want to use section searching for columns with existing text indexes, you must drop the indexes, define sections, if desired, enable section searching (through the preferences for the column policies), then reindex the columns.

Section Searching

A query expression operator, WITHIN, is provided for restricting a text query to a particular section.

The WITHIN operator can be used to restricts queries in two distinct ways:

- sentence and paragraph searching
- user-defined section searching

Note: Sentence / paragraph searching and user-defined section searching can be enabled concurrently for a text column; however, text queries can reference only a single section (sentence, paragraph, or user-defined) at a time.

In addition, if both sentence / paragraph searching and user-defined section searching are enabled for a text column, certain restrictions apply. For more information, see "User-Defined Sections" in this chapter.

See Also: For more information about the WITHIN operator and performing text queries using document sections, see *Oracle8 ConText Cartridge Application Developer's Guide*.

Sentence and Paragraph Searching

Sentence / paragraph searching returns documents in which two or more words occur within the same sentence or paragraph. In this way, sentence / paragraph searching is similar to proximity searching (NEAR operator), which returns documents in which two or more words occur within a user-specified distance.

For sentence / paragraph searching, the WITHIN operator takes *sentence* or *paragraph* as the value for the section name.

User-defined Section Searching

Section searching for user-defined sections returns documents in which one or more terms occur in a user-defined section.

For user-defined section searching, the WITHIN operator takes the name of a user-defined section.

Sentences and Paragraphs as Sections

ConText provides two system-level, predefined sections, sentence and paragraph, for sentence / paragraph searching; however, to enable ConText to identify sentence and paragraphs as sections, sentence and paragraph delimiters must be specified for the text lexer (BASIC LEXER Tile).

BASIC LEXER provides three attributes (*punctuations*, *whitespace*, and *newline*) for specifying sentence and paragraph delimiters.

Sentence Delimiters

Sentence delimiters are characters that, when they occur in the following sequence, indicate the end of a sentence and the beginning of a new sentence:

token -> punctuation character(s) -> whitespace character(s)

Paragraph Delimiters

Paragraph delimiters are characters that, when they occur in any of the following sequences, indicate the end of a paragraph and the beginning of a new paragraph:

token -> punctuation character(s) -> whitespace character(s) -> newline character(s)

token -> punctuation character(s) -> newline character(s) -> newline character(s)

By definition, paragraph delimiters also serve as sentence delimiters.

User-Defined Sections

A user-defined section is a body of text, delimited by user-specified start and end tags, within a document. ConText allows users to control the behavior / interaction of user-defined sections through the definition of sections as top-level or self-enclosing sections.

User-defined sections must be assigned a name and grouped into a section group. Sections are not created as individual, stand-alone objects. Instead, users create sections by adding them to an existing section group.

Note: If user-defined sections are used in conjunction with sentence / paragraph sections, *sentence* and *paragraph* are reserved words and cannot be used as section names.

See Also: For examples of creating section groups and adding, as well as removing, sections in section groups, see "Managing User-defined Document Sections" in Chapter 9, "Setting Up and Managing Text".

Start and End Tags

The beginning of a user-defined section is explicitly identified by a start tag, which can be any token in the text, as long as the token is a valid token 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, as long as the token can be recognized by the lexer.

Note: Start and end tags are case-sensitive if the text index for which they are defined is case-sensitive.

For documentation purposes, all references to start and end tags in this section are presented in uppercase.

For more information about case-sensitivity in text indexes, see "Text Indexes" in this chapter.

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 that is defined 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 token *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 token 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; however, 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.

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 for user-defined sections 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.

Limitations

User-defined 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: Insuffcent 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 user-defined sections for a text column. Section groups are assigned by name to a text column through the Wordlist preference in the column policy.

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 User-defined Document Sections" in Chapter 9, "Setting Up and Managing Text".

Predefined HTML Section Group

ConText provides a predefined section group, BASIC_HTML_SECTION, which enables user-defined 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

Setup Process for Section Searching

The process for setting up section searching differs depending on whether you are enabling section searching for sentences / paragraphs or user-defined sections.

Sentence and Paragraph Searching

The process model for enabling sentence / paragraph searching is as follows:

1. If necessary, specify values for the *whitespace*, *newline*, and *punctuations* attributes of the BASIC LEXER Tile.
2. Specify a value of '1' for the *sent_para* attribute (BASIC LEXER).
3. Create a Lexer preference for the Tile.
4. Create a policy that includes the Lexer preference you created.

User-defined Section Searching

The process model for defining sections and enabling section searching for these sections is as follows:

1. Use CTX_DDL.CREATE_SECTION_GROUP to create a section group for your user-defined 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 user-defined 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 must retain 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 of the BASIC LEXER Tile.

   Then, create a Lexer preference for the Tile.

5. Use the *section_group* attribute of the GENERIC WORD LIST Tile to specify the name of 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.

> **See Also:** For examples of defining section groups and sections, as well as creating a section-enabled Wordlist preference, see "Managing User-defined 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 Preference Examples" and "Lexer Preference Examples" in Chapter 8, "ConText Indexing".

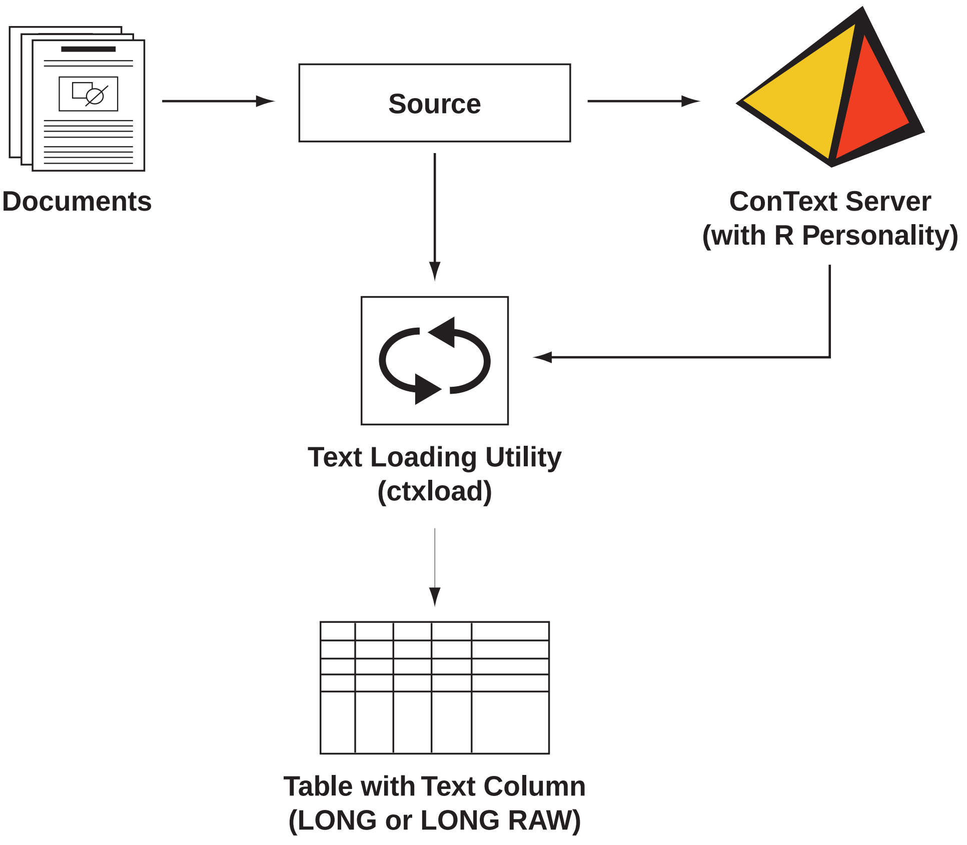
7 Automated Text Loading

This chapter describes the ConText data dictionary objects provided for automated text loading.

The topics discussed in this chapter are:

Overview of Automated Loading

Figure 7–1



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

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".

ConText Servers

If a ConText server is running with the R personality, it regularly checks all the sources that have been defined for columns in the database, then scans the 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.

Text Loading Utility (ctxload)

The text loading utility, ctxload, loads text from operating system files into the LONG or LONG RAW column in a table. ctxload requires the files to be in the load file format. If the files are not in the load file format, the files need 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.

See Also: For more information about ctxload and the required format for load files, see Chapter 10, "Text Loading Utility".

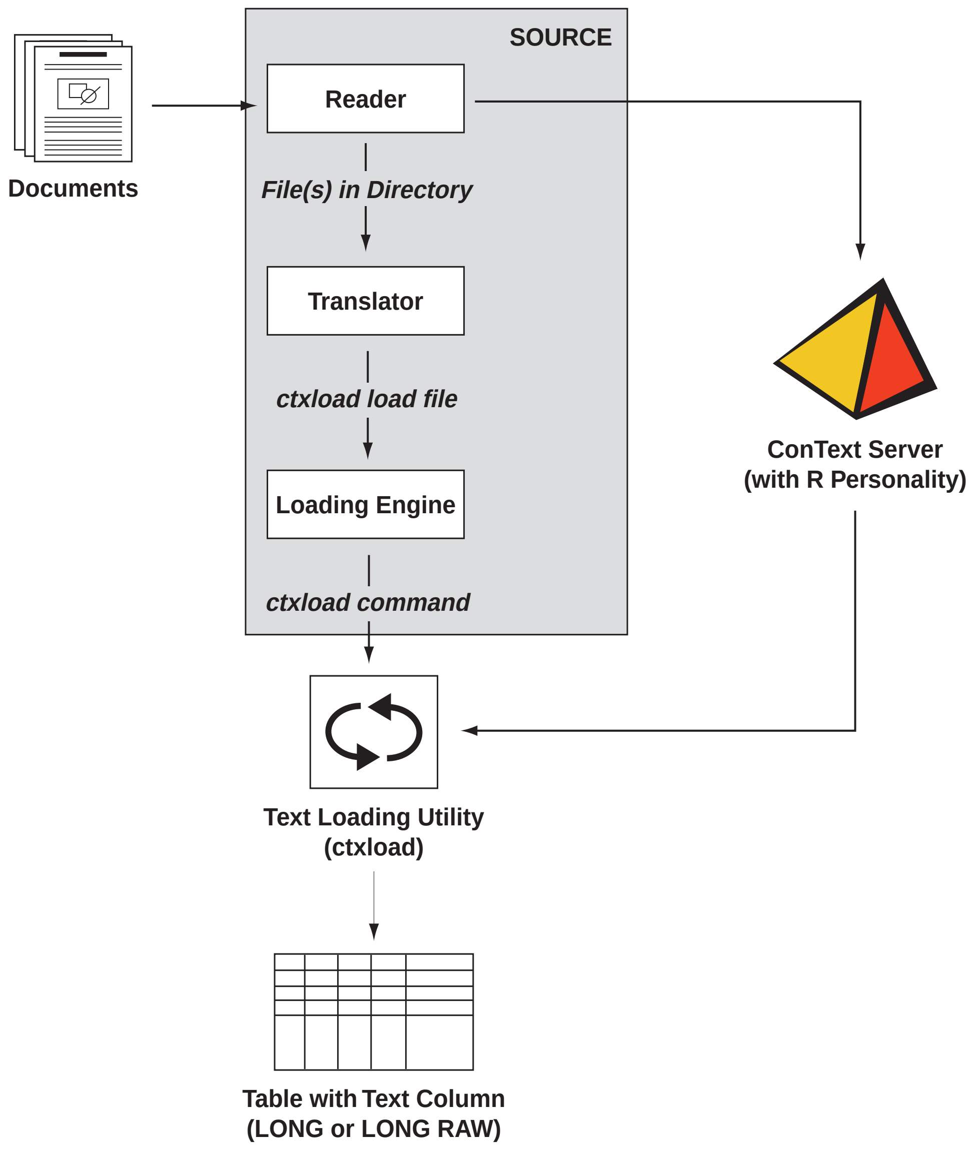
For more information about translators for text loading, see "Translator Tiles" in this chapter.

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.

Sources

Figure 7–2



To automate loading text from operating system files into a database column, ConText requires the following information:

- where are the files located in the local file system?
- are the files in the load file format required by ctxload?
- which command-line options to use when calling ctxload?

A source provides this information, in the form of text loading preferences (one preference for each of the requirements). Sources can be created by any ConText user with the CTXAPP role. Sources are stored in the ConText data dictionary.

Note: A source must exist for a column before a ConText server with the Loader personality can load text from operating system files into the column.

In addition to the preferences for a source, users specify a name and text column for the source. The text column in the source indicates the column to which text is loaded by ConText servers.

Note: The column datatype must be LONG or LONG RAW, because ctxload only supports loading text for these types.

Users can also choose to 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 Chapter 9, "Setting Up and Managing Text".

For more information about text loading preferences, see "Preferences for Text Loading" in this chapter.

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
- Translator Predefined Preferences
- Engine Predefined Preferences

What is a Text Loading Preference?

Text loading preferences specify the options that ConText uses to automatically load text. Each preference represents one (and only one) text loading option and is grouped into one of three categories or types, which correspond to the information ConText requires for automating text loading:

- Reader preferences
- Translator preferences
- Engine preferences

When creating a source, three preferences are specified for the source, one for each of the three types. If one of the types of preference is not specified when the source is created, the default, predefined preference for that type is used in the source.

A preference can be used in more than one source; however, two preferences of the same type cannot be used in the same source.

Tiles in Preferences

A text loading preference consists of a ConText Tile and one or more attributes (and their corresponding values) for the Tile.

See Also: For more information about the Tiles used in text loading preferences, see "Reader Tiles", "Translator Tiles", or "Engine Tiles" in this chapter.

Predefined Preferences

ConText provides predefined preferences for each type. These predefined preferences can be used by any ConText user with the CTXAPP role to create sources without first creating preferences.

Note: The predefined preference for the Reader category should not be used. The directory specified in the default Reader preference is a generic directory specified for default purposes only; the directory most likely does not exist in your file system.

User-defined Preferences

A ConText user with the CTXAPP role can create their own preferences by setting the required attributes for the appropriate Tile, 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.

Reader Predefined Preferences

ConText provides a single predefined Reader preference, DEFAULT_READER, for text loading.

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.

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.

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.

Reader Tiles

The Reader Tiles are used to specify the location of the files to be loaded.

ConText provides a single Tile, DIRECTORY READER, for creating Reader preferences for text loading sources.

DIRECTORY READER

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.

DIRECTORY READER has the following attribute(s):

Attribute	Attribute Values
directories	pathname for the directory where text loading files are located

directories

The *directories* attribute specifies the full pathname for the directory that the ConText 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.

Translator Tiles

ConText provides the following Tiles for creating Translator preferences for text loading sources:

Tile	Description
NULL TRANSLATOR	Files to be loaded are already in the load file format required by ctxload.
USER TRANSLATOR	Files to be loaded are converted into the required load file format using a translator provided and specified by the user.

NULL TRANSLATOR

The NULL TRANSLATOR Tile is used to specify that the load files for the loader (ctxload) are already in the format required by ctxload. It has no attributes.

USER TRANSLATOR

The USER TRANSLATOR Tile is used to specify a translator program that converts load files into the format required by ctxload.

USER TRANSLATOR has the following attribute(s):

Attribute	Attribute Values
command	executable for translator program

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 directory structures for ConText, see the Oracle8 installation documentation specific to your operating system.

Engine Tiles

ConText provides a single Tile, GENERIC LOADER, for creating Engine preferences for text loading sources:

GENERIC LOADER

The GENERIC LOADER Tile is used to specify the command-line options for ctxload.

GENERIC LOADER has the following attribute(s):

Attribute	Attribute Values
separate	Y (text stored in separate file(s), load file contains pointers to separate file(s))
	N (text stored in load file, default)
longsize	maximum size, in kilobytes, of text to be loaded (default 64)

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".

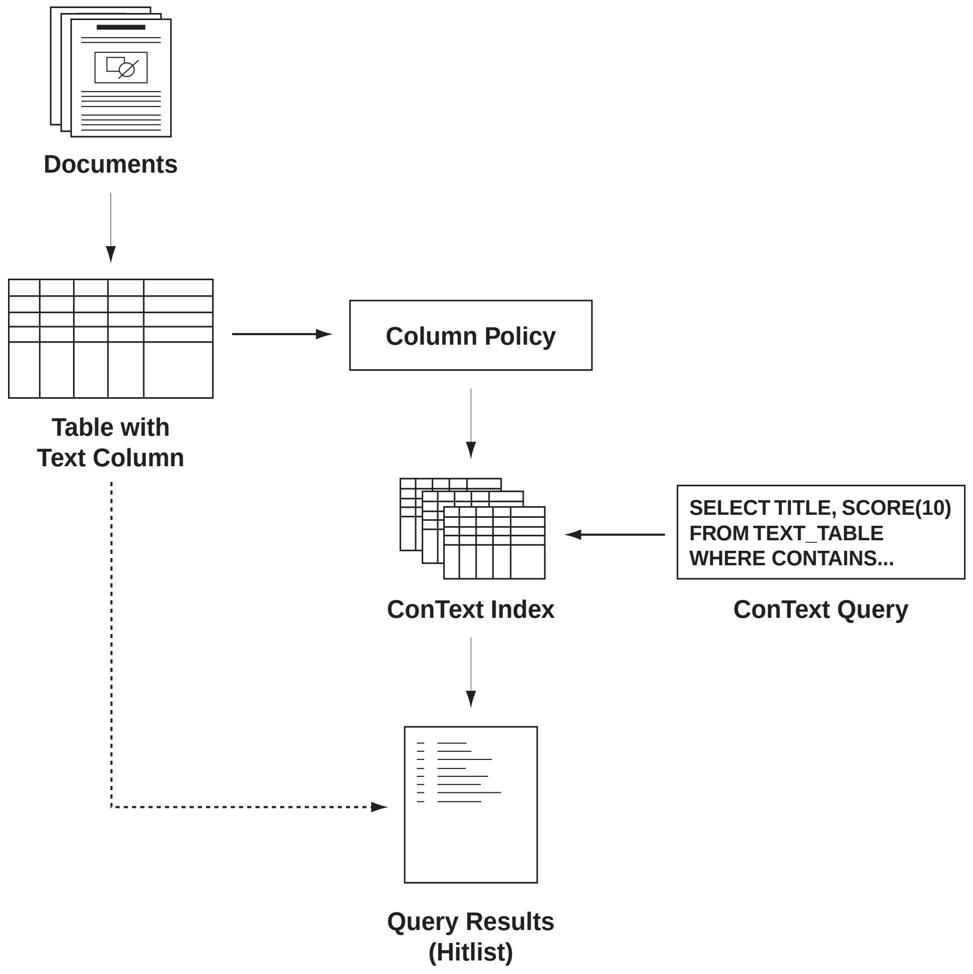
8 ConText Indexing

This chapter introduces the concepts necessary for understanding the indexing objects in the ConText data dictionary.

The following topics are discussed in this chapter:

Overview of Indexing

Figure 8–1



ConText indexes enable text and theme queries to be performed against text columns. Figure 8–1 illustrates the basic relationships between text tables, policies, ConText indexes, and ConText queries.

In a typical ConText system, text is loaded into a text column in a table, then a policy is created for the column.

The policy is used to create the ConText index, which resides in separate database tables associated with the text column through the policy. Once an index exists for a column, queries can be performed against the column using any of the query methods supported by ConText.

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

The query results are then returned, in the form of a hitlist, to the user that submitted the query. The query results can be returned directly or can be combined with structured data from the base table to refine the query or provide more information about the document that satisfy the query.

See Also: For more information about ConText indexes and the objects used to create them, see:

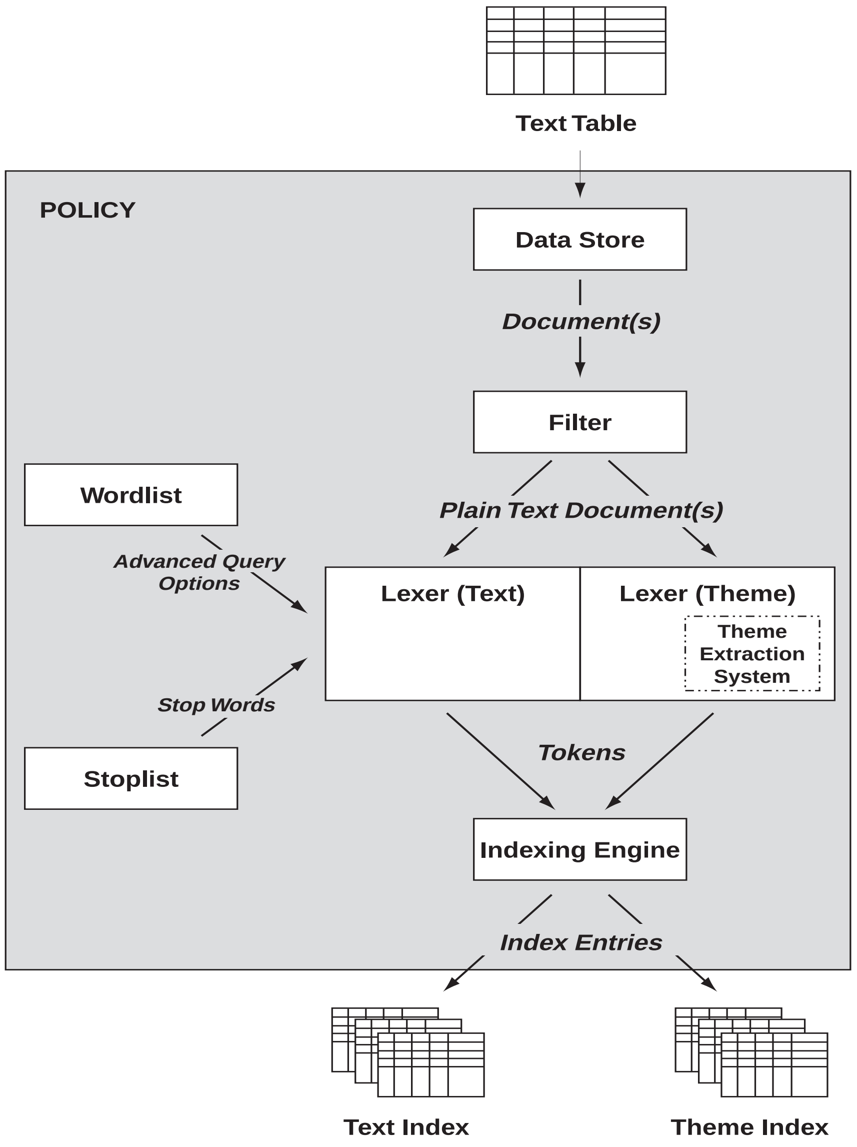
- "ConText Indexes" in Chapter 6, "Text Concepts"
- "Policies" in this chapter

For more information about text loading, see "Text Loading" in Chapter 6, "Text Concepts".

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

Policies

Figure 8–2



This section provides conceptual, as well as reference, information about policies:

- What is a Policy?
- Policy Examples
- Predefined Template Policies

What is a Policy?

To create a ConText index for text stored in a database column, ConText requires the following information about the text:

- how is the text stored in the column? - Data Storage
- what format(s) is the text in? - Filtering
- how should tokens in the text be identified? - Lexers
- how should the index be generated and where should it be stored? - Indexing Engine
- are any advanced query options going to be used? - Advanced Query (Wordlist) Options
- are there any words which should not have entries in the index? - Stop Words

Note: ConText also provides a facility for specifying whether the text is compressed; however, this facility is not currently implemented.

A policy provides this information for the column, in the form of indexing preferences (one preference for each of the requirements). Policies can be created by any ConText user with the CTXAPP role and are stored in the ConText data dictionary.

Note: A policy must exist for a column before a ConText server can create a index for the column.

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.

> **See Also:** For examples of creating policies, see "Creating a Column Policy" in Chapter 9, "Setting Up and Managing Text".

Template Policies

A template policy is a policy that does not have a text column assigned to it. Template policies are used as source policies when creating column policies or other template policies. The source policy for a policy specifies the preferences (one for each requirement) to be used as defaults in the policy.

For example, ConText provides a template policy, DEFAULT_POLICY, that is the default source policy for all column and template policies.

Any of the preferences provided in a template policy can be overwritten with other preferences (of the same type) by explicitly naming the preference during creation of the new policy.

ConText provides a number of predefined template policies, owned by CTXSYS. Users can create their own template policies or use the predefined template policies when creating policies.

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.

> **See Also:** For more information about text and theme queries, see "Text/Theme Queries" in Chapter 6, "Text Concepts".
>
> For more information about text indexing and theme indexing policies, see "Text Lexers" and "Theme Lexer" in this chapter.
>
> For a complete discussion of text and 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 stoplist (English)
	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 stoplist (English)
	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

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 *printjoins* or *skipjoins* 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	Default stoplist (English) is active

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 Lexer preference and all the remaining preferences from DEFAULT_POLICY. It can be used to create a column policy for a text column that contains documents in any of the formats supported by the ConText internal filters.

TEMPLATE_BASIC_WEB

This template policy 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 *keep_tag*
BASIC_HTML_LEXER	Basic lexer with characters specified for *startjoins* and *endjoins*
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 *section_group*
DEFAULT_STOPLIST	Default stoplist (English) is active

TEMPLATE_DIRECT

This template policy uses the same preferences as DEFAULT_POLICY. 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 Stoplist preference and all the remaining preferences from DEFAULT_POLICY. 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 Stoplist preference and all the remaining preferences from DEFAULT_POLICY. It can be used to create a policy that uses the default stoplist (English) during indexing.

TEMPLATE_MD

This template policy uses the MD_TEXT predefined Data Store preference and all the remaining preferences from DEFAULT_POLICY. 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 preferences from DEFAULT_POLICY. 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 preferences from DEFAULT_POLICY. 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:

- What is an Indexing Preference?
- 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 and is grouped into one of six categories or types, which correspond to the information ConText requires for creating indexes:

- Data Store preferences
- Filter preferences
- Lexer preferences
- Engine preferences
- Wordlist preferences
- Stoplist preferences

When creating a policy, six preferences are specified, one for each of the six types. If one of the preference is not specified when the policy is created, the preference (for that type) from the DEFAULT_POLICY template policy is used.

A preference can be used in more than one policy; however, two preferences of the same type 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.

Tiles in Preferences

Tiles are the objects in the ConText data dictionary that provide ConText 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. Each Tile may have none, one, or many attributes that are used to define preferences. The attributes identify which indexing options are active for the preference.

You define one of the types of preferences by setting the attributes with the desired values for the appropriate Tile, then creating the preference. While a type is not explicitly assigned to a preference, it is implied through the association of the Tile with the preference.

Predefined Preferences

ConText provides a number of predefined preferences (owned by CTXSYS) for each type. These predefined preferences can be used by any ConText user with the CTXAPP role to create policies without having to first create preferences.

User-defined Preferences

ConText users 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 predefined preferences provided by ConText.

Data Store Predefined Preferences

ConText provides the following predefined Data Store preferences:

- DEFAULT_DIRECT_DATASTORE (Used in DEFAULT_POLICY)
- 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 hardcoded paths, delimited by colons are: /oracle/data, /oracle/data2, /oracle/data3.

Note: If the locations of your files do not match the hardcoded 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 (Used in DEFAULT_POLICY)
- 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 (Used in DEFAULT_POLICY)
- 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
punctuations	. ? !
printjoins	NULL (indicates no characters defined as *printjoins* for the BASIC LEXER)
skipjoins	NULL (indicates *no* characters defined as *skipjoins* for the BASIC LEXER)
continuation	- \

KOREAN

This preference calls the KOREAN LEXER Tile and can be used for parsing Korean text. Because the KOREAN LEXER Tile does not have any attributes, no attributes are set for this preference.

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 to allocate the following amount of memory for indexing: 12582912 bytes.

Wordlist Predefined Preferences

ConText provides the following predefined Wordlist preferences, which all use the GENERIC WORD LIST Tile:

- BASIC_HTML_WORDLIST
- NO_SOUNDEX (Used in DEFAULT_POLICY)
- SOUNDEX
- VGRAM_CHINESE_WORDLIST
- VGRAM_CHINESE_WORDLIST

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 predefined 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 (Used in DEFAULT_POLICY)
- NO_STOPLIST

Note: All of the Stoplist preferences call the GENERIC STOP LIST Tile.

DEFAULT_STOPLIST

This preference defines a list of English terms treated as stop words during indexing.

In addition to the English stoplist in DEFAULT_STOPLIST, ConText supplies stoplists for many European languages. These stoplists are not provided as predefined Stoplist preferences; they are provided as SQL scripts which can be used to create Stoplist preferences for the languages.

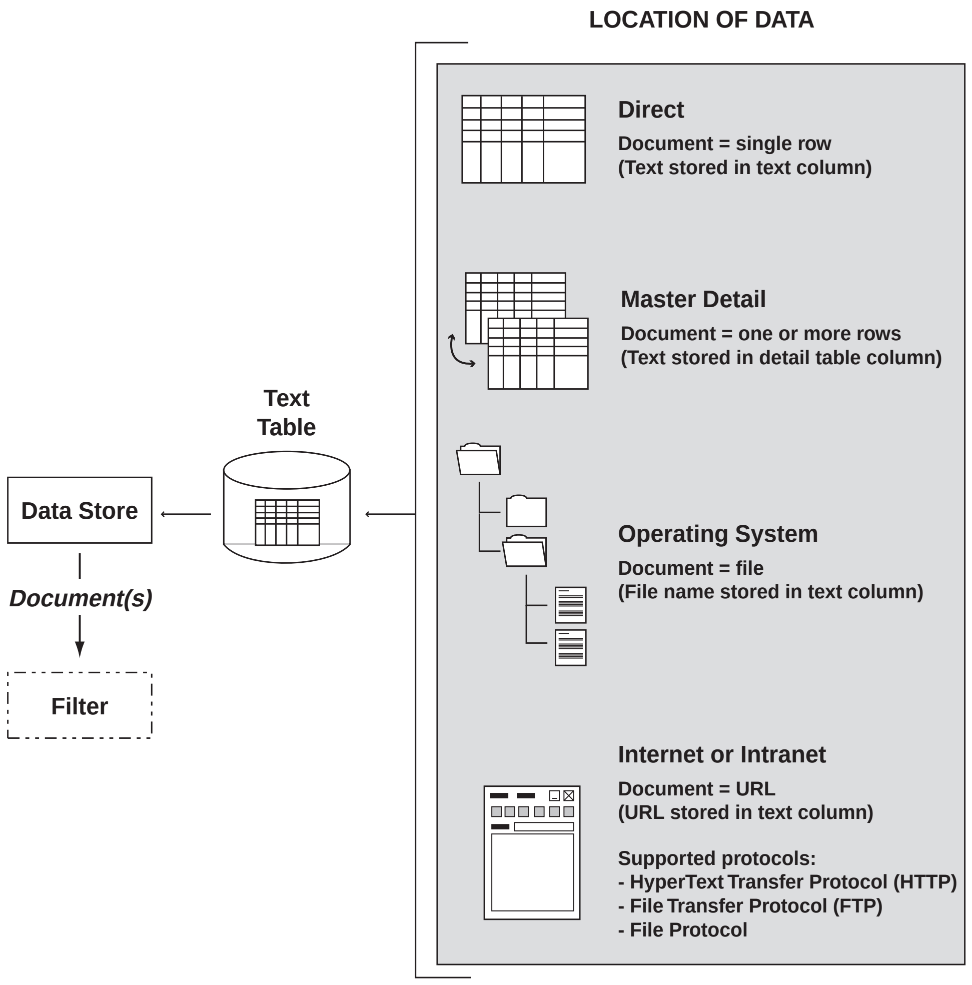
See Also: For a complete list of the stop words in DEFAULT_STOPLIST, as well as the list of stop words for each supplied stoplist, see Appendix A, "Supplied Stoplists".

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.

Data Storage

Figure 8–3


LOCATION OF DATA
Direct
Document = single row
(Text stored in text column)
Master Detail
Document = one or more rows
(Text stored in detail table column)
Operating System
Document = file
(File name stored in text column)
Internet or Intranet
Document = URL
(URL stored in text column)
Supported protocols:
- HyperText Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)
- File Protocol
Text Table
Data Store
Document(s)
Filter

ConText supports four methods of storing text in a column:

- Direct Storage
- Master-Detail Storage
- External Storage (Operating System Files)
- External Storage (URLs)

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.

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 *master.pk*
	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 NEW 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 Storage (Operating System Files)

With operating system storage, the text column does not contain the actual text of the document, but rather stores a pointer (file name) to the operating-system file that contains the text of the document. The Data Store preference for the column policy uses the OSFILE Tile and specifies the location of the file.

Suggestion: If text is stored in operating system files, the column containing the file names should be either a CHAR or VARCHAR2 column. LONG and LONG RAW columns are best suited for long documents stored directly in the database.

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).

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.

External Storage (URLs)

For text stored in external World Wide Web files, the complete address for each file must be stored as a Uniform Resource Locator (URL) in the text column and the URL Tile must be utilized in the Data Store preference for the column policy.

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 "External Storage (URLs)" 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*.

Data Store Tiles

ConText provides the following Tile(s) for creating Data Store preferences:

Tile	Description
DIRECT	Data stored internally in the text column. Each row is indexed as a single document
MASTER DETAIL	Data stored internally in the text column. Document consists of one or more rows in a detail table, with header information stored in a master table. The policy is created on the text column in the detail table. As a result, queries return detail information from the detail table. Header information must be queried explicitly.
MASTER DETAIL NEW	Data stored internally in the text column. Document consists of one or more rows in a detail table, with header information stored in a master table. The policy is created on a designated text column in the master table. As a result, queries return header information from the master table. Detail information must be queried explicitly.
OSFILE	Data stored externally in operating system files. File names stored in the text column.
URL	Data stored externally in files located on an intranet or the Internet. Uniform Resource Locators (URLs) stored in the text column.

DIRECT

The DIRECT Tile is used for text stored directly in the database. It has no attributes.

MASTER DETAIL

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.

The MASTER DETAIL Tile has the following attribute(s):

Attribute	Attribute Values
binary	0 (plain text)
	1 (binary text)

binary

The *binary* attribute specifies whether text is in plain text format (0) or binary format (1) in the detail table in a master-detail relationship.

Text in plain text format uses newline characters at the end of each line to indicate the end of the line. Text in binary format does not use newline characters to indicate the end of the line.

MASTER DETAIL NEW

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.

MASTER DETAIL NEW has the following attribute(s):

Attribute	Attribute Values
binary	0 (plain text)
	1 (binary text)
detail_table	*name of the detail table* (string)
detail_key	*name of the foreign key column in the detail table* (string)
detail_lineno	*name of the line number column in the detail table* (string)
detail_text	*name of the text column in the detail table* (string)
detail_text_size	Internal use only

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

The OSFILE Tile is used for text stored in files accessed through the local file system.

OSFILE has the following attribute(s):

Attribute	Attribute Values
path	*path1*:*path2*:...:*pathn*

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

The URL Tile is used for text stored:

- in files on the World Wide Web (accessed through HTTP or FTP)
- in files in the local file system (accessed through the file protocol)

URL has the following attribute(s):

Attribute	Attribute Values
timeout	*seconds* (0 to 3600, default 30)
maxthreads	*number of threads* (0 to 1024, default 8)
maxurls	*buffer length in bytes* (1 to 4294967295, default 256)
urlsize	*URL length* (32 to 65535, default 256)
maxdocsize	*document size* (256 to 4294967295, default 2000000)
http_proxy	*host name*
ftp_proxy	*host name*
no_proxy	*string* (up to 16 strings, separated by commas)

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 Preference Example

The following example creates a preference named *doc_ref* for the OSFILE Tile:

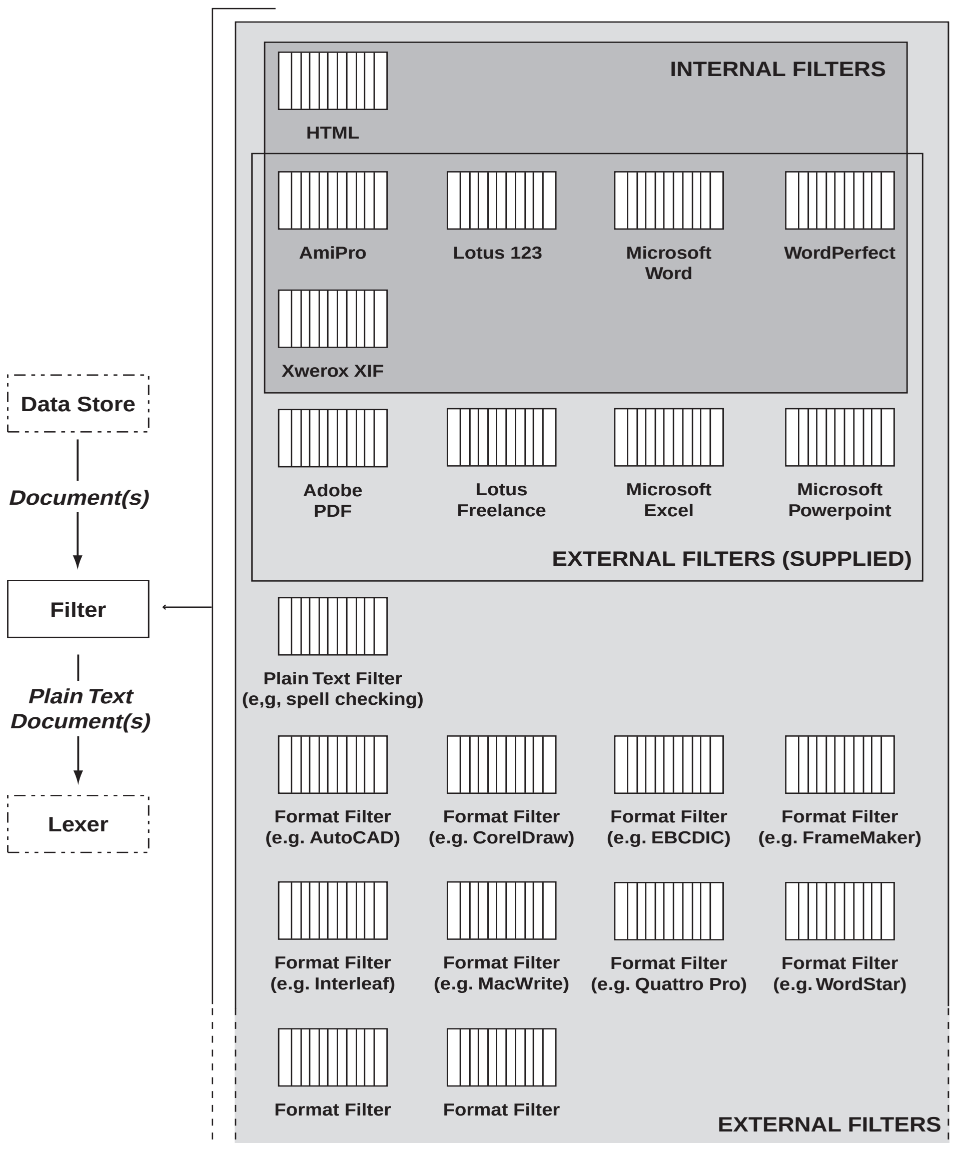
```
begin
  ctx_ddl.set_attribute ('PATH', '/private/mydocs');
  ctx_ddl.create_preference ('DOC_PREF', 'Path my for my documents' 'OSFILE');
end;
```

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.

Filtering

Figure 8–4


INTERNAL FILTERS
HTML
AmiPro
Lotus 123
Microsoft Word
WordPerfect
Xwerox XIF
Data Store
Document(s)
Adobe PDF
Lotus Freelance
Microsoft Excel
Microsoft Powerpoint
EXTERNAL FILTERS (SUPPLIED)
Filter
Plain Text Filter (e,g, spell checking)
Plain Text Document(s)
Lexer
Format Filter (e.g. AutoCAD)
Format Filter (e.g. CorelDraw)
Format Filter (e.g. EBCDIC)
Format Filter (e.g. FrameMaker)
Format Filter (e.g. Interleaf)
Format Filter (e.g. MacWrite)
Format Filter (e.g. Quattro Pro)
Format Filter (e.g. WordStar)
Format Filter
Format Filter
EXTERNAL FILTERS

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 Mixed-Format Columns

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

Internal Filters

ConText provides internal filters for:

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

In addition, ConText provides the Autorecognize Filter, an internal filter for columns containing mixed formats.

Plain Text Filtering

Plain text requires 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 viewers:

- 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 viewers, see *Oracle8 ConText Cartridge Workbench User's Guide*.

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

Autorecognize Filter

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.

External Filters

ConText provides a framework for users to plug-in user-defined and/or third-party filters to extract plain text 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 viewers provided with the ConText Workbench.

For more information about the ConText viewers, 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.

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 system-supplied 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".

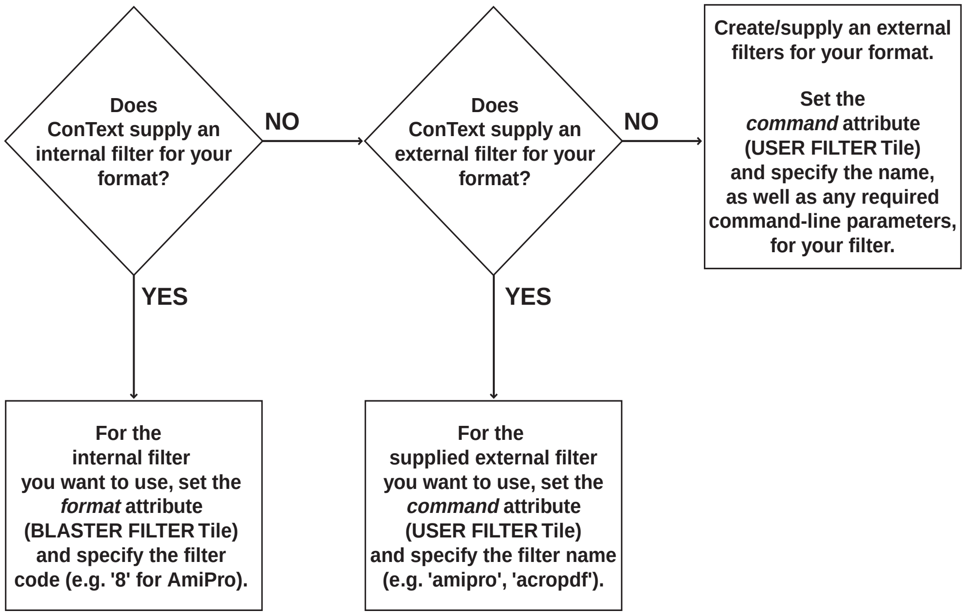
Supplied External Filters

ConText provides a number of external filters for filtering many of the most popular word processing and desktop publishing formats on a number of platforms.

See Also: For a complete list of the external filters supplied by ConText, as well as instructions for setting up and using the filters, see "Supplied External Filters" in Appendix D, "External Filter Specifications".

Filters for Single-Format Columns

Figure 8–5


Does ConText supply an internal filter for your format?
NO
Does ConText supply an external filter for your format?
NO
Create/supply an external filters for your format.
Set the *command* attribute (USER FILTER Tile) and specify the name, as well as any required command-line parameters, for your filter.
YES
For the internal filter you want to use, set the *format* attribute (BLASTER FILTER Tile) and specify the filter code (e.g. '8' for AmiPro).
YES
For the supplied external filter you want to use, set the *command* attribute (USER FILTER Tile) and specify the filter name (e.g. 'amipro', 'acropdf').

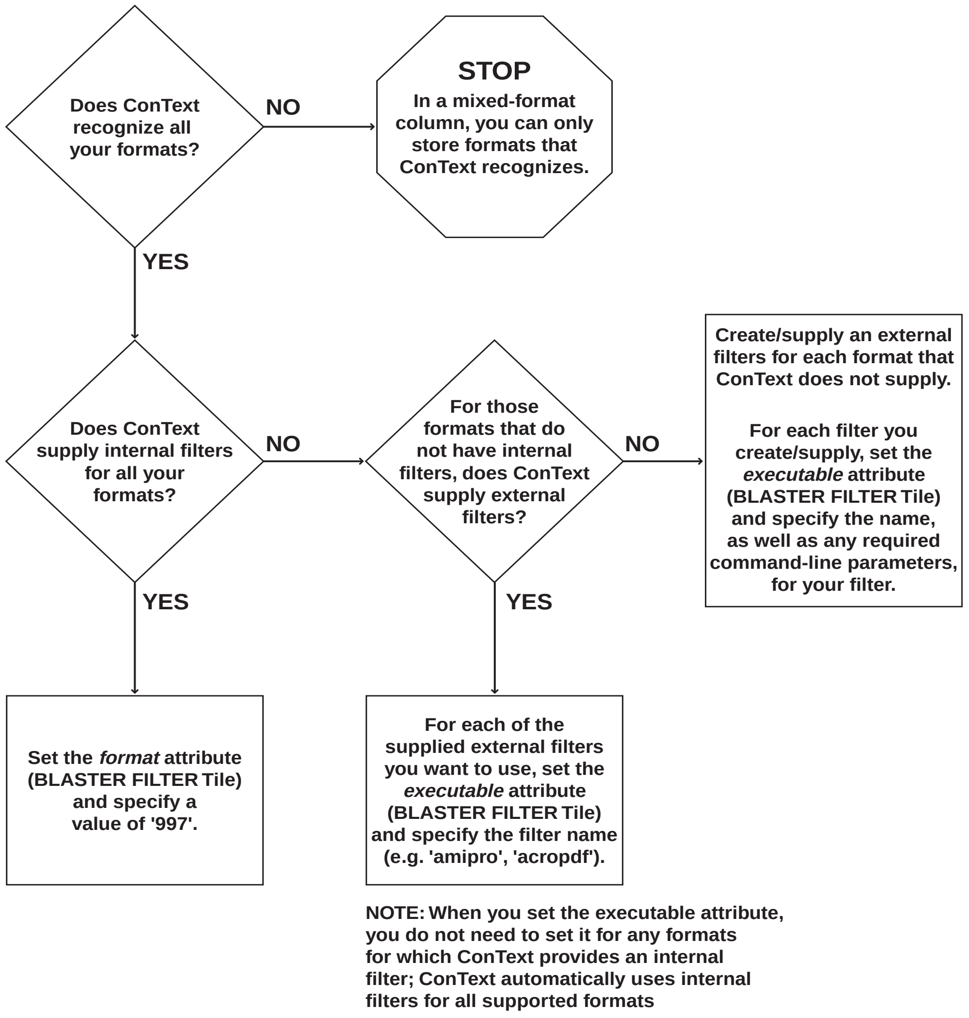
For columns that store documents in only one format, a single filter is specified in the Filter preference for the column policy. The filtering method for the column is determined by whether the format is supported by the internal or external filters.

Figure 8–5 illustrates the different filtering methods for single-format columns.

> **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

Figure 8–6


Does ConText recognize all your formats?
NO
STOP
In a mixed-format column, you can only store formats that ConText recognizes.
YES
Does ConText supply internal filters for all your formats?
NO
For those formats that do not have internal filters, does ConText supply external filters?
NO
Create/supply an external filters for each format that ConText does not supply.
For each filter you create/supply, set the *executable* attribute (BLASTER FILTER Tile) and specify the name, as well as any required command-line parameters, for your filter.
YES
YES
Set the *format* attribute (BLASTER FILTER Tile) and specify a value of '997'.
For each of the supplied external filters you want to use, set the *executable* attribute (BLASTER FILTER Tile) and specify the filter name (e.g. 'amipro', 'acropdf').
NOTE: When you set the executable attribute, you do not need to set it for any formats for which ConText provides an internal filter; ConText automatically uses internal filters for all supported formats

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.

Figure 8–6 illustrates the different filter specification methods for mixed-format columns.

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.

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

For a complete list of supported formats for mixed-format columns, see "Supported Formats for Mixed-Format Columns" in Appendix D, "External Filter Specifications".

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.

ConText provides the following Tile(s) for creating Filter preferences:

Tile	Description
BLASTER FILTER	Tile for filtering formatted text and/or plain text using internal filters, external filters or some combination of both.
FILTER NOP	Tile for plain text (does not require filtering)
HTML FILTER	Tile for filtering plain text containing HTML tags
USER FILTER	Tile for specifying external filter for a column.

BLASTER FILTER

The BLASTER FILTER Tile is used to specify either:

- internal filters are used to filter document
- multiple external filters are used to filter documents in a mixed-format column.

Attributes

BLASTER FILTER has the following attribute(s):

Attribute	Attribute Values
executable	*format id* (number), *filter executable, sequence* (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)
	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)

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* are mutually exclusive.

See Also: For a list of the format IDs supported by the *executable* attribute, see "Supported Formats for Mixed-Format Columns" in Appendix D, "External Filter Specifications".

format

The *format* attribute specifies the internal filter used for filtering text stored in a text column.

FILTER NOP

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

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):

Attribute	Attribute Values
code_conversion	0 (disabled)
	1(enabled)
keep_tag	*tag* (string), *sequence* (number)

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

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):

Attribute	Attribute Values
command	*filter executable*

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 (with the *executable* attribute) should be used instead of the USER FILTER Tile.

Filter Preference Examples

The following section provides two Filter preference examples.

> **See Also:** For more examples of creating Filter preferences, see "Creating Filter Preferences" in Chapter 9, "Setting Up and Managing Text".

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;
```

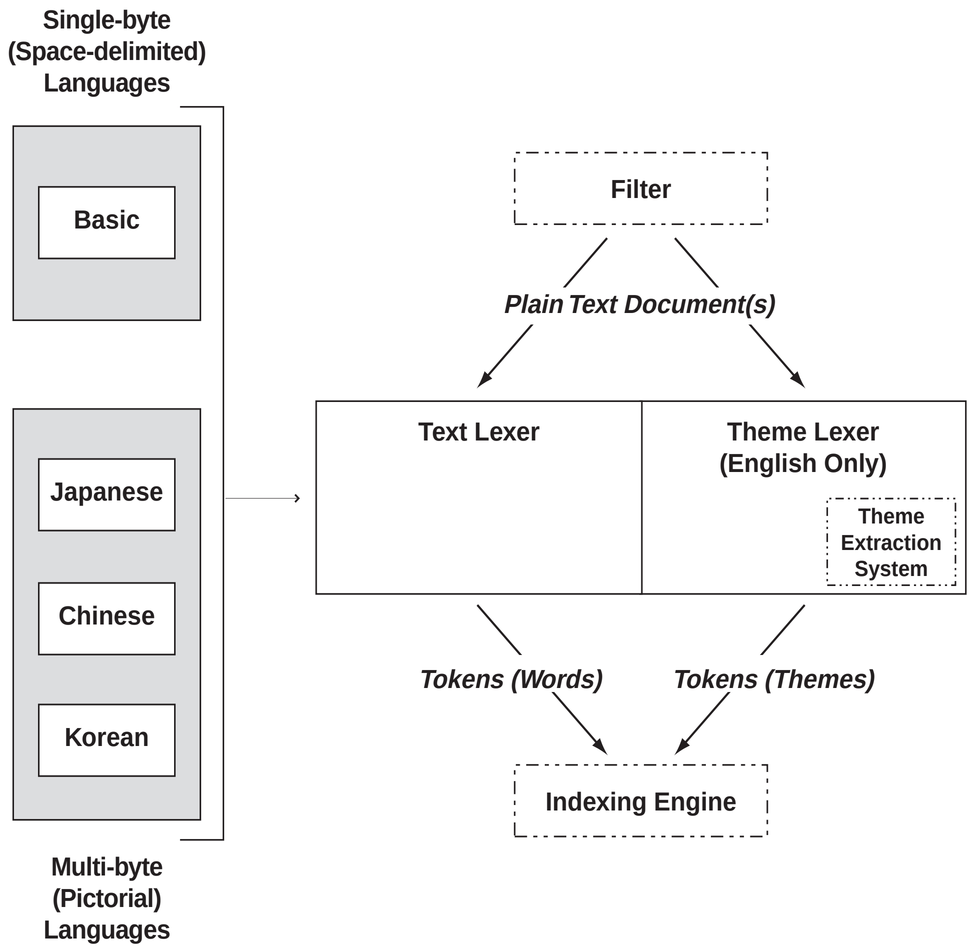
In this example, the <TITLE>, </TITLE>, <HEAD>, </HEAD>, <BODY>, </BODY>, <H1>, and </H1> HTML tags are retained by the HTML filter during filtering, provided the *startjoins* and *endjoins* attributes for the BASIC LEXER Tile are set appropriately.

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.

See Also: For more information about document sections, see "Document Sections" in Chapter 6, "Text Concepts".

Lexers

Figure 8–7



A lexer parses text and identifies tokens for indexing. ConText supports two types of lexers:

- Text Lexers
- Theme Lexer

The text lexer provided for English and other single-byte, space-delimited languages supports the following features:

- Base-letter Conversion
- NLS Compliance
- Composite Word Indexing

Text Lexers

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 (BASIC LEXER Tile) for all of the single-byte, space-delimited languages, such as English (7-bit character set) and other European languages (8-bit character sets). 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 provides three separate lexers for processing Japanese, Chinese, and Korean text.

The Chinese (CHINESE V-GRAM LEXER Tile) and Japanese (JAPANESE V-GRAM LEXER Tile) lexers do not rely on finding token boundaries within text; instead, they uses a dictionary 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.

Text Indexing Policies

By specifying one of the text lexers in the Lexer preference for a policy, you designate the policy as a text indexing policy.

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 more information about text indexing, see "Text Indexes" in Chapter 6, "Text Concepts".

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.

By specifying the THEME LEXER Tile in the Lexer preference for a policy, you designate the policy as a theme indexing policy.

Once a theme index is created for the policy, any text requests, including theme queries, on the policy will result in the theme index being accessed.

See Also: For more information about theme indexing, see "Theme Indexes" in Chapter 6, "Text Concepts".

Base-letter Conversion

For text indexes created on text columns containing languages that use an 8-bit (single-byte) character set, you can specify whether extended characters encountered in tokens are converted to their base-letter representation before their tokens are stored in the text index. Extended characters include special characters and characters with diacritical marks (e.g. accents, umlauts).

Text Indexing

Base-letter conversion is an attribute that you can set when creating a Lexer preference using the BASIC LEXER Tile.

If base-letter conversion is enabled for the Lexer preference in a policy, during text indexing, 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 base-letter conversion is enabled. The following string of text is encountered:

```
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 National Language Support and the NLS_LANG environment variable, see *Oracle8 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*.

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 multilingual environments, such as multinational 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 or Dutch text, the BASIC LEXER Tile provides an attribute for enabling composite word indexing. With composite word indexing, tokens that are compound words (specifically nouns) 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 German-language 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 can be enabled for text columns containing only German or Dutch 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 both 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*.

Lexer Tiles

ConText provides the following Tile(s) for creating Lexer preferences:

Tile	Description
BASIC LEXER	Basic lexer used for extracting tokens from text in languages, such as English and most Western European languages, that use single-byte character sets.
CHINESE V-GRAM LEXER	Lexer used for extracting tokens from Chinese-language text.
JAPANESE V-GRAM LEXER	Lexer used for extracting tokens from Japanese-language text.
KOREAN LEXER	Lexer used for extracting tokens from Korean-language text.
THEME LEXER	Lexer which utilitizes the Linguistics Theme Extraction System to generate themes as tokens for theme indexing.

BASIC LEXER

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 and composite word indexing for German and Dutch text.

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.

BASIC LEXER has the following attribute(s):

Attribute	**Attribute Values**
continuation	*characters* (string)
numgroup	*characters* (string)
numjoin	*characters* (string)
printjoins	*characters* (string)
punctuations	*characters* (string)
skipjoins	*characters* (string)
startjoins	*non-alphanumeric characters that occur at the beginning of a token* (string)
endjoins	*non-alphanumeric characters that occur at the end of a token* (string)
whitespace	*characters* (string)
newline	*characters* (string)
sent_para	0 (disabled)
	1 (enabled)
base_letter	0 (disabled)
	1 (enabled)
mixed_case	0 (disabled)
	1 (enabled)
composite	0 (no composite word indexing)
	1 (German composite word indexing)
	2 (Dutch composite word indexing)

Note: The BASIC LEXER Tile attributes that use character strings can contain multiple characters. Each character in the string serves as a distinct character for that type of attribute.

For example, if the string '*_.-' is specified for the *printjoins* attribute, each individual character ('*', '_', '.', and '-') in the string is treated by ConText as a joining character that is included in the index entry for a token in which the character occurs.

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 hyphen ’-’ and 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. The defaults are period '.', question mark '?', and exclamation point '!'.

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..

In addition, BASIC LEXER uses *punctuations* characters in conjunction with *newline* and *whitespace* characters to determine sentence and paragraph deliminters for sentence/paragraph searching.

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 *skipjoins*, 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 User-defined 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.*

---

whitespace

whitespace specifies the characters that are treated as blank spaces between tokens. BASIC LEXER uses *whitespace* characters in conjunction with *punctuations* and *newline* characters to identify character strings that serve as sentence delimiters for sentence/paragraph searching.

The predefined, default values for *whitespace* are 'space' and 'tab'; these values cannot be changed. Specifying characters as *whitespace* characters adds to these defaults.

newline

newline specifies the characters that indicate the beginning of a new line of text. BASIC LEXER uses *newline* characters in conjunction with punctuations and whitespace characters to identify character strings that server as paragraph delimiters for sentence/paragraph searching.

The only valid values for *newline* are '\n' and '\r' (for carriage returns) and the default is '\n'.

sent_para

sent_para enables (1) or disables (0) sentence/paragraph searching. The default is '0'.

base_letter

base_letter specifies whether characters that have diacritical marks (umlauts, 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).

mixed_case

mixed_case 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).

Note: ConText ensures text queries match the case-sensitivity of the index being queried. As a result, if you enable case-sensitivity for your text index, queries against the index are always case-sensitive.

composite

The *composite* attribute specifies whether composite word indexing is disabled (0) or enabled for either German (1) or Dutch (2) text. The default is 0 (composite word indexing disabled).

Note: The *composite* and *mixed_case* attributes are mutually exclusive; Composite indexes do not support case-sensitivity.

See Also: For more information, see "Composite Word Indexing" in this chapter.

CHINESE V-GRAM LEXER

The CHINESE V-GRAM LEXER Tile is used for identifying tokens for creating text indexes for Chinese text.

CHINESE V-GRAM LEXER has the following attribute(s):

Attribute	Attribute Values
hanzi_indexing	1
	2

hanzi_indexing

The *hanzi_indexing* attribute specifies the number of characters used for pattern matching while indexing.

A value of 1 indicates that the Chinese lexer examines each character individually to determine token boundaries.

A value of 2 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

The JAPANESE V-GRAM LEXER Tile is used for identifying tokens for creating text indexes for Japanese text.

JAPANESE V-GRAM LEXER has the following attribute(s):

Attribute	Attribute Values
kanji_indexing	1
	2

kanji_indexing

The *kanji_indexing* attribute specifies the number of characters used for pattern matching while indexing.

A value of 1 indicates that the Japanese lexer examines each character individually to determine token boundaries.

A value of 2 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

The KOREAN LEXER Tile is used for identifying tokens for creating text indexes for Korean text. It has no attributes.

THEME LEXER

The THEME LEXER Tile is used in theme indexing policies to create theme indexes for English-language text. It has no attributes.

See Also: For an example of creating a theme indexing policy, see "Creating a Column Policy for Theme Indexing" in Chapter 9, "Setting Up and Managing Text".

Lexer Preference 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"

Indexing Engine

The indexing 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.

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 engine determines the amount of memory used to create ConText indexes and where in the database the indexes are stored.

See Also: For more information about creating an Engine preference, see "Creating an Engine Preference" in Chapter 9, "Setting Up and Managing Text".

Engine Tiles

ConText provides the following Tile(s) for creating Engine preferences:

Tile	Description
ENGINE NOP	No engine used for indexing (Not implemented - DO NOT USE)
GENERIC ENGINE	Indexing engine used to create index entries and store in database tables comprising the ConText index.

ENGINE NOP

The ENGINE NOP Tile specifies that no engine is used for indexing. This Tile is currently not implemented and should *not* be used to create Engine preferences for indexing.

GENERIC ENGINE

The GENERIC ENGINE Tile specifies that the indexing engine provided by ConText is used for indexing.

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".

GENERIC ENGINE has the following attribute(s):

Attribute	Attribute Values
** none **	N / A
index_memory	*memory in bytes* (integer)
optimize_default	*default ConText index optimization method*
i1t_tablespace, i1t_storage, i1t_other_parms	*tablespace* (string), *STORAGE clause* (string), and *other table creation parameters* (string) for token table
i1i_tablespace, i1i_storage, i1i_other_parms	*tablespace* (string), *STORAGE clause* (string), *and other index creation parameters* (string) *for index on token table*

Attribute	Attribute Values
ktb_tablespace, ktb_storage, ktb_other_parms	*tablespace* (string), *STORAGE clause* (string), *and other table creation parameters* (string) *for mapping table*
kid_tablespace, kid_storage, kid_other_parms kik_tablespace, kik_storage, kik_other_parms	tablespace (string), STORAGE clause (string), and other index creation parameters (string) for indexes on mapping table
lst_tablespace, lst_storage, lst_other_parms	tablespace (string), STORAGE clause (string), and other table creation parameters (string) for control table
lix_tablespace, lix_storage, lix_other_parms	tablespace (string), STORAGE clause (string), and other table creation parameters (string) for index on control table
sqr_tablespace, sqr_storage, sqr_other_parms	tablespace (string), STORAGE clause (string), and other table creation parameters (string) for SQE results table
sri_tablespace, sri_storage, sri_other_parms	tablespace (string), STORAGE clause (string), and other table creation parameters (string) for index on SQE results table

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_NEW_TABLE.

*xxx*_tablespace

i1t_tablespace, *ktb_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.

i1i_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

i1t_storage, *ktb_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.

i1i_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 SQL Reference*.

*xxx*_other_parms

i1t_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.

i1i_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 SQL Reference*.

For more information about the parallel query option in Oracle, see *Oracle8 Tuning*.

See Also: For more information about SQEs, see *Oracle8 ConText Cartridge Application Developer's Guide*.

Engine Preference 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_PARMS',' pctfree 20');
  ctx_ddl.set_attribute ('I1I_OTHER_PARMS',' parallel 2');
  ctx_ddl.create_preference ('DOC_ENGINE', 'Test case',
                             'GENERIC ENGINE' );
end;
```

Advanced Query (Wordlist) Options

ConText provides advanced query (Wordlist) options for expanding text queries using the following methods:

- Stemming
- Fuzzy Matching
- Soundex

ConText also provides an option for refining text queries using user-defined document sections.

Note: While the expansion options provided by ConText can be used in theme queries, ConText automatically provides expansion for theme queries through the Linguistics Theme Extraction System.

In addition, the concept of document sections does not apply to theme indexes.

As such, the Wordlist options are not generally used for theme indexes.

See Also: For more information about expanding and refining text queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

For more information about user-defined sections for refining queries, see "User-Defined Sections" in Chapter 6, "Text Concepts".

Stemming

Stemming expands a text 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.

Inflectional Stemming

For all the supported languages, the stemmers return standard inflected forms of a word, such as the plural form (e.g. *department* --> *departments*).

Derivational Stemming

For English, an additional stemmer is provided which returns standard inflected forms and derived forms (e.g. *department* --> *departments*, *departmentalize*).

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 SOUNDEX function in SQL.

For more information about the SOUNDEX function in SQL, see *Oracle8 SQL Reference*.

Wordlist Tiles

ConText provides a single Tile, GENERIC WORD LIST, for creating Wordlist preferences.

GENERIC WORD LIST

The GENERIC WORD LIST Tile is used to enable the advanced query options (stemming, fuzzy matching, Soundex, and user-defined section searching) for text indexes.

> **See Also:** For more information about expansion methods in queries, see *Oracle8 ConText Cartridge Application Developer's Guide*.

GENERIC WORD LIST has the following attribute(s):

Attribute	Attribute Values
stclause	*STORAGE clause* (string) for Soundex wordlist table
instclause	*STORAGE clause* (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)

Attribute	Attribute Values
	14 (French)
	15 (German)
	16 (Italian)
	17 (Spanish)
	18 (OCR text)
section_group	*name of section group*

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. 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 Preference 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 WORD LIST');
end;
```

Stop Words

To manage the size of text indexes, ConText supports defining stop words. Stop words are common terms that you do not want to include in a text index.

The collection of stop words for a text column is called a stoplist, as defined in a Stoplist preference. You can define up to 4095 stop words for a stoplist.

Note: Because theme indexes contain proportionately fewer entries than text indexes and size is not considered an issue, stoplists generally do not provide much value for theme indexes.

As such, ConText does not use stoplists for theme indexes. If a stoplist exists for a text column, the stoplist is ignored during theme indexing

Stop Words in Queries

ConText does not create index entries for words defined as stop words; however, it does record the stop words, up to eight, that proceed and follow an indexed term. This enables text queries for phrases which contain stop words.

To conserve space in the text index, ConText does not record the actual stop words in the index entries. Instead, ConText records code numbers, called sequences, that correspond to the stop words. Sequence numbers are assigned to stop words by the user when a stoplist is defined.

For example, the words *he, is, at, the,* and *of* are defined as stop words and each stop word is assigned a sequence by the user. During indexing, the string "he is at the top of the class" is encountered.

Index entries are created only for the words *top* and *class*; however, the words *he, is, at, the,* and *top* are stored as preceding and following stop words for the index entries.

As a result, users can query phrases such as ’he is at the top’ and ’top of the class’.

Case-sensitivity

Stoplists for case-sensitive text indexes are automatically case-sensitive, meaning that 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 stop word entry for *each* commonly occurring variation (i.e. lowercase, initial uppercase, all-uppercase) that may occur for a stop word. For example, some articles, such as *a* and *the* in English, often appear at the beginning of sentences. As a result, the initial uppercase form of the articles (*A* and *The*) should be included in the stoplist.

Stoplist Tiles

ConText provides a single Tile, GENERIC STOP LIST, for creating Stoplist preferences.

GENERIC STOP LIST

The GENERIC STOP LIST Tile specifies the terms that should not be included in the text index.

GENERIC STOP LIST has the following attribute(s):

Attribute	Attribute Values
stop_word	*word* (string), *sequence* (number)

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* (1 to 4095) for the word

Stoplist Preference 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.

9 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 Indexes
- Managing Preferences
- Managing Indexes
- Managing Thesauri
- Managing User-defined 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:

Task	Supported in Sys. Admin. Tool?	Supported in Config. Manager?
Using ctxload	No	No
Using ConText Servers for Automated Text Loading	No	No
Generating Document Textkeys	No	No
Updating/Exporting a Document	No	No

Note: The ConText Workbench includes a command-line 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 Workbench User'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 CTX_DDL.CREATE_SOURCE procedure 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 Chapter 7, "Automated 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 Executable and Utility".

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 "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 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 Policies

This section provides details for using the CTX_DDL PL/SQL package to perform the following policy administration tasks:

Task	Supported in Sys. Admin. Tool?	Supported in Config. Manager?
Modifying a Policy	Yes	Yes
Creating a Column Policy	Yes	Yes
Creating a Column Policy for an Object Table	Yes	Yes
Creating a Column Policy for Theme Indexing	Yes	Yes
Using Composite Textkeys in a Column Policy	Yes	Yes
Modifying a Policy	Yes	Yes
Deleting a Policy	Yes	Yes

Note: When creating policies (template or column), 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 policy names, such as *ascii_indexing_policy*.

Creating a Template Policy

To create a template policy, use the PL/SQL procedure CTX_DDLCREATE_TEMPLATE_POLICY and specify the names of the preferences that you want to use in the policy:

```
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. If *template_pol* is specified as the source policy when creating a column policy, these preferences are copied to the new column policy.

Creating a Column Policy

To create a column policy for text indexing, use the PL/SQL procedure CTX_DDL.CREATE_POLICY:

```
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 (i.e. template) policy. The text column (specified by *colspec*) is *articles* in a table named *docs* in the current user's schema. 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 SOUNDEX predefined preference owned by CTXSYS.

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 NULL COMPRESSOR predefined 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 Column Policy for an Object Table

The process for creating a column policy for a text column in an object table is identical to the process for a text column in a relational table; however, the object table must have a primary key defined.

Note: In addition to the required primary key for the object table, the datatype for the text column and textkey column must be one of the standard datatypes supported by ConText; user-defined object datatypes are not supported.

For more information about standard datatypes and user-defined object datatypes, see *Oracle8 Concepts*.

Consider the following example of a simple object table:

```
CREATE TYPE doc_t AS OBJECT(
id NUMBER,
text VARCHAR2(255));

CREATE TABLE doc_tab of doc_t (id PRIMARY KEY);
```

> **See Also:** For more information about the CREATE TYPE command, see *Oracle8 SQL Reference*.

When creating a policy for the *text* column, ConText uses the *id* column as the textkey, regardless of whether a *textkey* value is explicitly specified for CREATE_POLICY; however, you should always specify a textkey value to ensure the correct column is used. For example:

```
begin;
  ctx_ddl.create_policy(
          policy_name => 'doc_pol',
          colspec => 'doc_tab.text',
          textkey => 'id');
end;
```

Creating a Column Policy for Theme Indexing

To create a theme indexing policy, use the PL/SQL procedure CTX_DDL.CREATE_POLICY and specify the THEME_LEXER predefined preference:

```
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;
```

The following example illustrates how to create a policy identical to the previous policy example, except that THEME_LEXER is used in place of *doc_link*:

> **See Also:** For more information about theme indexes and queries, see "ConText Indexes" in Chapter 6, "Text Concepts".

Using Composite Textkeys in a Column 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".

Modifying a Policy

To modify a policy (column or template), use the PL/SQL procedure CTX_DDL.UPDATE_POLICY:

```
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.

Note: If a column 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.

Deleting a Policy

To delete a policy (column or template) from the ConText data dictionary, use the PL/SQL procedure CTX_DDL.DROP_POLICY:

```
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 column 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 Preferences

This section provides details for using the CTX_DDL PL/SQL package to perform the following administration tasks for preferences:

Task	Supported in Sys. Admin. Tool?	Supported in Config. Manager?
Creating a Preference:	Yes	Yes
Creating an Engine Preference	Yes	Yes
Creating a Data Store Preference for a Master Table	Yes	Yes
Creating Filter Preferences	Yes	Yes
Creating a Theme Lexer Preference	Yes	Yes
Creating a Stoplist Preference	Yes	Yes
Deleting a Preference	Yes	Yes

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.

Note: CREATE_PREFERENCE must be called immediately after SET_ATTRIBUTE to assign the specified attribute(s) to the preference that you are creating.

For example:

```
begin
  ctx_ddl.set_attribute ('PATH',
                         '/public/doc1:/public/doc2');
  ctx_ddl.create_preference ('PUB_DOCS',
                             'Docs stored in files',
                             'OSFILE');
end;
```

In this example, a Data Store 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.

The following attributes require multiple values:

- *stop_word* (GENERIC STOP LIST Tile)
- *executable* (BLASTER FILTER Tile)
- *keep_tag* (GENERIC WORD LIST Tile)

> **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 *docs_text.fk_docid*
	TITLE	VARCHAR2	Document title
	COMMENT	VARCHAR2	Text column
DOCS_TEXT	FK_DOCID	NUMBER	Foreign key to *docs.docid*
	CHAPTER	NUMBER	Detail information (combined with *docs.docid* uniquely identifies rows in *docs_text*)
	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 "Filter Tiles" in Chapter 8, "ConText 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 provided by ConText. The examples in this section use the external filters provided by ConText.

For a complete list of the external filters supplied by ConText, see "Supplied External Filters" in Appendix D, "External Filter Specifications".

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 *amipro*:

```
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 Formats for Mixed-Format Columns" in Appendix D, "External Filter Specifications".

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 create a column policy that uses the theme lexer (THEME LEXER Tile). For this purpose, ConText provides a predefined lexer preference, THEME_LEXER, that calls THEME LEXER.

Note: Because THEME LEXER does not have any attributes to be set, you do not generally need to create theme lexer preferences.

See Also: For examples of creating a policy that uses the THEME_LEXER predefined preference, see "Creating a Column Policy for Theme Indexing" in this chapter.

Creating a Stoplist Preference

To create a Stoplist preference:

1. Use CTX_DDL.SET_ATTRIBUTE to set the *stop_word* attribute (GENERIC STOP LIST Tile) for each word that you do not want ConText to index.

   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 Indexes

This section provides details for using the CTX_DDL PL/SQL package to perform the following indexing tasks:

Task	Supported in Sys. Admin. Tool?	Supported in Config. Manager?
Creating an Index	Yes	Yes
ConText Indexing in Parallel	Yes	Yes
Indexing Existing Columns (Hot Upgrade)	Yes	Yes
Updating an Index	Yes	Yes
Dropping an Index	Yes	Yes
Optimizing an Index	Yes	Yes
Resuming Index Creation/Optimization	Yes	Yes

Creating an Index

To create a ConText index (theme or text) on a table or view, 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 *i1i_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_PARMS', ' PARALLEL 4');
  ctx_ddl.set_attribute('KID_OTHER_PARMS', ' PARALLEL 4');
  ctx_ddl.set_attribute('KIK_OTHER_PARMS', ' 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 SQL Reference*.

For more information about the parallel query option, see *Oracle8 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.

Piecewise Optimization

To optimize the entries for an individual word in an index, set *opttyp* to DR_OPTIMIZE_PIECEWISE and specify a value for *term* in CTX_DDL.OPTIMIZE_INDEX.

For example:

```
begin
  ctx_ddl.optimize_index(policy_name => 'DOC_POL',
                         optyp       => ctx_ddl.dr_optimize_piecewise,
                         term        => 'company');
end;

begin
  ctx_ddl.optimize_index(policy_name => 'THEME_POL',
                         optyp       => ctx_ddl.dr_optimize_piecewise,
                         term        => 'ABC Corp');
end;
```

In the first example, OPTIMIZE_INDEX is called in piecewise mode for the word *company* in the index for a policy named *doc_pol.*

In the second example, OPTIMIZE_INDEX is called in piecewise mode for the theme *ABC Corp* in the index for a policy named *theme_pol.*

Resuming Index Creation/Optimization

If index creation / 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.

In the following example, index creation is resumed for the text column in a policy named *doc_pol*.

```
execute ctx_ddl.resume_failed_index('DOC_POL')
```

You can also choose to start the index creation over from the beginning using CTX_ DDL.CREATE_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 creation failed.

The log also can be used to determine whether to resume index creation or simply start the operation over, based on the stage at which the creation failed and / or the percentage of the creation completed before failure.

Managing Thesauri

This section provides details for using the CTX_THES PL/SQL package and/or ctxload to perform the following indexing tasks:

Task	Supported in Sys. Admin. Tool?	Supported in Config. Manager?
Creating a Thesaurus	Yes	No
Creating a Case-sensitive Thesaurus	Yes	No
Creating the Supplied Thesaurus	No	No
Creating/Updating a Thesaurus Entry	Yes	No
Deleting a Thesaurus	Yes	No
Creating a Thesaurus Output File	No	No

Note: The Configuration Manager, included in the ConText Workbench, does not provide thesaurus maintenance or viewing functionality.

All thesauri maintenance tasks must be performed through the command-line or the System Administration tool, which is also included in the ConText Workbench.

In addition, thesauri viewing can be performed only through the System Administration tool. The System Administration tool provides a graphical interface for illustrating relationships between thesaurus entries.

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 or the System Administration tool.

Using CREATE_THESAURUS

The following SQL*Plus example creates an empty thesaurus named *tech_thes* using CREATE_THESAURUS:

```
variable thesid number
execute :thesid := ctx_thes.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 SQL*Plus 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 thesid number
execute :thesid := ctx_ddl.create_thesaurus('scinece_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 the Supplied Thesaurus

To create the thesaurus supplied by ConText, navigate to the directory containing the thesaurus load file.

For example, in a UNIX-based environment, type the following command at the operating system prompt:

```
cd $ORACLE_HOME/ctx/thes
```

Then run ctxload in thesaurus creation mode (-*thes* parameter) and specify the name of the load file (-*file* parameter). For the -*name* parameter, specify the name that you want to assign to the thesaurus created by ctxload.

Note: You can give the thesaurus any name; however, if you want to use the thesaurus as the default thesaurus, name it *DEFAULT*.

For example, in a UNIX-based environment, type the following command at the operating system prompt:

```
ctxload -thes -thescase y -file dr0thsus.txt -name generic_thes
```

In this example, a case-sensitive thesaurus named *generic_thes* is created using the 'dr0thsus.txt' load file. The supplied thesaurus is designed for use as a case-sensitive thesaurus; however, it can be used equally well as a case-insensitive thesaurus. To create the supplied thesaurus as case-insensitive, omit the -*thescase* parameter when running ctxload.

> **See Also:** For a complete description of ctxload, see Chapter 10, "Text Loading Utility".
>
> For more information about case-sensitivity in thesauri, see "Thesauri" in Chapter 6, "Text Concepts".

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, Oracle does not recommend updating entries using CREATE_PHRASE.
>
> When 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 thesaurus import file, named *tech_terms.out*, 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 User-defined Document Sections

This section provides details for creating user-defined sections and section groups and assigning a section group to a text column via the column policy:

Task	Supported in Sys. Admin. Tool?	Supported in Config. Manager?
Creating a Section Group	Yes	Yes
Creating a Section	Yes	Yes
Creating a Wordlist Preference with a Section Group	Yes	Yes
Creating a Policy for a Section Group	Yes	Yes
Viewing Sections and Section Groups	Yes	Yes
Removing a Section from a Section Group	Yes	Yes
Dropping a Section Group	Yes	Yes

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 user-defined 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 Preference Examples" and "Lexer Preference Examples" in Chapter 8, "ConText 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 user-defined sections and section groups that have been created in the ConText data dictionary, use the CTX_ALL_SECTIONS and CTX_ALL_SECTIONS 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.

10
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

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 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.

-commitafter
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.

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, ctxload 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).

Note: The thesaurus query operators (SYN, PT, BT, BTG, BTP, BTI, NT, NTG, NTP, NTI, and RT) are reserved words and, thus, cannot be used as phrases in thesaurus entries.

In addition, the string 'E$_' is reserved for internal use and cannot be used as a phrase in thesaurus entries.

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 / NTI 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 newline 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
  ```

```
RT 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
```

11

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 user-defined 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_INTTRIG	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 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 user-defined 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 or 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 or 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 "User-Defined 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
```

Notes

Clearing the attribute buffer is not required to create a preference. The buffer is cleared automatically after each call to CTX_DDL.CREATE_PREFERENCE.

CLEAR_ATTRIBUTE is used primarily for clearing attributes that have been set incorrectly for a preference prior to the actual creation of the preference.

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 BOLLEAN  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.

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_SOUNDEX',
                    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. Because a policy is owned by the user who creates it, the policy name must be unique for a user.

colspec

Specify the column and table in the current user's schema for which the policy is created. 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 preferences in the template policy are used to create the column policy, except when explicitly overwritten. The template policy can be owned by the current user or CTXSYS.

The default is DEFAULT_POLICY owned by CTXSYS.

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 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 (Compressor preferences 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',
                        desrcription => '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

A policy can only be created for a table in the current user's schema.

All of the arguments are optional, except for *policy_name*. If you do not specify a preference for one of the preference types, the preference (for that type) in DEFAULT_POLICY 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 from the other column names by a comma. 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',
                        desrcription => '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_SOUNDEX',
                    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 internal 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.

To reenble automatic DML after the trigger has been dropped, either the index must be dropped and recreated or a trigger must be created using CTX_DML.REINDEX.

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 ConText 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,
                       term        IN VARCHR2 DEFAULT NULL,
                       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 on the index:

- 1 (DR_OPTIMIZE_LAZY_DELETES) - use original token table to perform in-place deletion of references to deleted/modified documents, otherwise known as garbage collection
- 2 (DR_OPTIMIZE_COMPACT_INDEXES) - use original token table to perform in-place compaction of index fragments
- 3 (DR_OPTIMIZE_COMPACT_NEW) - use second, mirror token table to perform two-table compaction of index fragments
- 4 (DEFRAGMENT_TO_NEW_TABLE) - use second, mirror token table to perform combined two-table compaction/garbage collection
- 5 (DEFRAGMENT_IN_PLACE) - use original token table to perform combined in-place compaction/garbage collection
- 6 (DR_OPTIMIZE_PIECEWISE) - use original token table to perform combined in-place compaction/garbage collection for a single word, specified by *term*

The default for *opttyp* depends on the value set for the *default_optimize* attribute in the GENERIC ENGINE Tile (see "Notes" for more information).

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 compaction and/or garbage collection.

The default is 1.

term
Specify the word (index token or section name) for which piecewise optimization is performed. This argument is used only if *opttyp* is set to DR_OPTIMIZE_PIECEWISE.

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

opttyp must be fully qualified with the PL/SQL package name (CTX_DDL) as shown in the examples.

If *opttyp* is set to DR_OPTIMIZE_PIECEWISE, a value for *term* must be specified. If *opttyp* is set to any other value, any value specified for *term* is ignored.

The default for *opttyp* is the value specified for the *default_optimize* attribute (GENERIC ENGINE Tile) in the Engine preference of the policy for the text column to be optimized. If no value was specified for *default_optimize* in the Engine preference for the policy, the default is DEFRAGMENT_TO_NEW_TABLE.

threshold is used only when *opttyp* is set to DR_OPTIMIZE_COMPACT_INDEX (in-place compaction only). If *opttyp* is set to any value other than DR_OPTIMIZE_ COMPACT_INDEX, *threshold* is ignored.

threshold 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.

parallel is used only for two-table compaction and/or garbage collection. If a value that utilizes in-place compaction and/or garbage collection is specified for *opttyp, parallel* is ignored.

term is case-sensitive, regardless of whether the index (text or theme) is case-sensitive. As a result, ConText only performs piecewise optimization for those rows in an index that exactly match the value specified for *term*.

In addition, if a value is specified for *term* and no rows (index tokens or section names) exist for the specified value, OPTIMIZE_INDEX completes successfully; however, no optimization takes 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 index DML operation (creation or optimizationo).

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 indexing / optimization resumption.

operation
Specify the operation that was being performed on the index at the time of failure and needs to be resumed:

- 1 (OPERATION_CREATE)
- 2 (OPERATION_OPTIMIZE)

The default is 1.

parallel
Specify the degree of parallelism used for creating / optimizing the index. The default is 1.

opttyp
If *operation* is 2 (OPERATION_OPTIMIZE), use this argument to specify the type of two-table optimization to perform:

- 3 (DR_OPTIMIZE_COMPACT_NEW) - perform compaction of index fragments
- 4 (DEFRAGMENT_TO_NEW_TABLE) - perform combined compaction of index fragments and garbage collection of obsolete DOCIDs.

The default depends on the value set for the default_optimize attribute in the GENERIC ENGINE Tile (see "Notes" for more information).

switch_new
Internal use only.

drop_old
Internal use only.

Examples

```
begin
  ctx_ddl.resume_failed_index('MY_POLICY',
                               operation => 2,
                               parallel  => 2
                               opttyp    => DDL>DEFRAGMENT_TO_NEW_TABLE);
end;
```

In this example, index optimization is resumed with a parallelism level of 2 for the index for *my_policy*. The type of optimization performed is combined two-table compaction / garbage collection.

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 / optimization 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 example.

The default for *opttyp* is the value specified for the *default_optimize* attribute (GENERIC ENGINE Tile) in the Engine preference of the column policy. If no value was specified for *default_optimize* when the Engine preferece 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_SOUNDEX',
                    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 timestamp 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 when the internal trigger that is normally created during index creation does not exist.

Finally, it can be used to notify the system of updates to documents stored externally. For example, if a text column uses the OSFILE Tile for documents stored in the file system, REINDEX can be called when a 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

In the first two examples, REINDEX is called for a single document identified by textkey and either the policy name for the column or the column id.

```
execute ctx_dml.reindex('MY_POLICY', '1')

execute ctx_dml.reindex(3451, '1')
```

In the last example, REINDEX is used to create a trigger named *resume_update* 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 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;
```

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.

REINDEX can be used to enable automatic DML queue notification when the internal DML trigger does not exist. The internal trigger may not exist for the following reasons:

- trigger could not be created (i.e. indexing a view)
- trigger was not initially created (i.e. create_trig in CTX_DDL.CREATE_INDEX set to 'FALSE')
- trigger was dropped using CTX_DDL.DROP_INTTRIG

To enable automatic DML, a trigger must be created on the object containing the text for which the index needs to be updated. In the case of a ConText index on a view, the object is the base table for the view. For a table without a trigger, the object is the table itself.

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: These are the only procedures and functions that are required for creating and maintaining thesauri and thesaurus entries. The remaining procedures and functions in CTX_THES are used *internally* by ConText to enable the thesaurus operators in query expressions.

In addition, CTX_THES calls an internal package, CTX_THS. The procedures and functions in CTX_THS should *not* be called directly.

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.

Note: The thesaurus query operators (SYN, PT, BT, BTG, BTP, BTI, NT, NTG, NTP, NTI, and RT) are reserved words and, thus, cannot be used as phrases in thesaurus entries.

n addition, the string 'E$_' is reserved for internal use and cannot be used as a phrase in thesaurus entries.

rel
Specify the new relationship between *phrase* and *relname*:

- SYN (i.e. *phrase* is synonymous term for *relname*)
- PT | USE | SEE (i.e. *phrase* is preferred synonymous term for *relname*)
- BT (i.e. *phrase* is broader term for *relname*)
- NT (i.e. *phrase* is narrower term for *relname*)
- BTG (broader generic term)
- NTG (narrower generic term)

- BTP (broader partitive term)
- NTP (narrower partitive term)
- BTI (broader instance term)
- NTI (narrower instance term)
- RT (related term)

See Also: For more information about the relationships you can define for thesaurus entries, see "Thesaurus Entries and Relationships" 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

This part contains the following appendices:

A
Supplied Stoplists

ConText supplies an English stoplist, which serves as the default stoplist. In addition, ConText supplied stoplists for many European languages.

Note: Each of the supplied stoplists, except for the English stoplist, is provided as a SQL script that can be run to create a Stoplist preference for the indicated language.

The English stoplist is provided as a predefined Stoplist preference named DEFAULT_STOPLIST.

This appendix describes how to create Stoplist preferences for all the supplied stoplists, except the English stoplist, and lists the stop words in each stoplist:

- Creating a Supplied Stoplist
- English
- Danish (DA)
- Dutch (NL)
- Finnish (FI)
- French (FR)
- German (DE)
- Italian (IT)
- Portuguese (PR)
- Spanish (ES)
- Swedish (SE)

Creating a Supplied Stoplist

To create a Stoplist preference for any of the following languages, run the SQL script for the language:

Language	SQL Script
Danish (DA)	drstopda
Dutch (NL)	drstopnl
Finnish (FI)	drstopfi
French (FR)	drstopfr
German (DE)	drstopde
Italian (IT)	drstopit
Portuguese (PR)	drstoppr
Spanish (ES)	drstopes
Swedish (SE)	drstopse

The exact location of the SQL scripts is operating system dependent; however, the scripts are generally located in the following directory structure:

```
<Oracle_home_directory>
    <ConText_directory>
        demo
```

See Also: For more information about the directory structure for ConText, see the Oracle8 installation documentation specific to your operating system.

Editing the Scripts

If you need to add or remove words from a stoplist, you can directly edit the SQL script for the stoplist.

Note: If you want to edit a stoplist, you should edit the respective script *before* running it.

You can edit the scripts using any line/text editor, provided the editor supports the character set for the language of the stoplist(s) you are editing.

Running the Scripts

The SQL scripts can be run in SQL*Plus or any other utility that supports SQL scripts. Each script takes a single argument, *preference_name*, as input.

The following SQL*Plus example creates a Stoplist preference named *danish_stopwords*:

```
SQL> @drstopda danish_stopwords
```

This example assumes SQL*Plus was started from the directory in which the stoplist scripts are located, which is the recommended method for running the scripts.

English

The following English words are defined as stop words in the DEFAULT_STOPLIST preference provided by ConText:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	be	had	it	only	she	was
about	because	has	its	of	some	we
after	been	have	last	on	such	were
all	but	he	more	one	than	when
also	by	her	most	or	that	which
an	can	his	mr	other	the	who
any	co	if	mrs	out	their	will
and	corp	in	ms	over	there	with
are	could	inc	mz	s	they	would
as	for	into	no	so	this	up
at	from	is	not	says	to	

Danish (DA)

The drstopda script creates a Stoplist preference containing the following Danish words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
af	en	god	hvordan	med	og	udenfor
aldrig	et	han	I	meget	oppe	under
alle	endnu	her	De	mellem	pe	ved
altid	fe	hos	i	mere	rask	vi
bagved	lidt	hovfor	imod	mindre	hurtig	
de	fjernt	hun	ja	ner	sammen	
der	for	hvad	jeg	hvoner	temmelig	
du	foran	hvem	langsom	nede	nok	
efter	fra	hvor	mange	nej	til	
eller	gennem	hvorhen	meske	nu	uden	

Dutch (NL)

The drstopnl script creates a Stoplist preference containing the following words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
aan	betreffende	eer	had	juist	na	overeind	van	weer
aangaande	bij	eerdat	hadden	jullie	naar	overigens	vandaan	weg
aangezien	binnen	eerder	hare	kan	nadat	pas	vanuit	wegens
achter	binnenin	eerlang	heb	klaar	net	precies	vanwege	wel
achterna	boven	eerst	hebben	kon	niet	reeds	veeleer	weldra
afgelopen	bovenal	elk	hebt	konden	noch	rond	verder	welk
al	bovendien	elke	heeft	krachtens	nog	rondom	vervolgens	welke
aldaar	bovengenoemd	en	hem	kunnen	nogal	sedert	vol	wie
aldus	bovenstaand	enig	hen	kunt	nu	sinds	volgens	wiens
alhoewel	bovenvermeld	enigszins	het	later	of	sindsdien	voor	wier
alias	buiten	enkel	hierbeneden	liever	ofschoon	slechts	vooraf	wij
alle	daar	er	hierboven	maar	om	sommige	vooral	wijzelf
allebei	daarheen	erdoor	hij	mag	omdat	spoedig	vooralsnog	zal
alleen	daarin	even	hoe	meer	omhoog	steeds	voorbij	ze
alsnog	daarna	eveneens	hoewel	met	omlaag	tamelijk	voordat	zelfs
altijd	daarnet	evenwel	hun	mezelf	omstreeks	tenzij	voordezen	zichzelf
altoos	daarom	gauw	hunne	mij	omtrent	terwijl	voordien	zij
ander	daarop	gedurende	ik	mijn	omver	thans	voorheen	zijn
andere	daarvanlangs	geen	ikzelf	mijnent	onder	tijdens	voorop	zijne
anders	dan	gehad	in	mijner	ondertussen	toch	vooruit	zo
anderszins	dat	gekund	inmiddels	mijzelf	ongeveer	toen	vrij	zodra
behalve	de	geleden	inzake	misschien	ons	toenmaals	vroeg	zonder
behoudens	die	gelijk	is	mocht	onszelf	toenmalig	waar	zou
beide	dikwijls	gemoeten	jezelf	mochten	onze	tot	waarom	zouden
beiden	dit	gemogen	jij	moest	ook	totdat	wanneer	zowat
ben	door	geweest	jijzelf	moesten	op	tussen	want	zulke
beneden	doorgaand	gewoon	jou	moet	opnieuw	uit	waren	zullen
bent	dus	gewoonweg	jouw	moeten	opzij	uitgezonderd	was	zult
bepaald	echter	haar	jouwe	mogen	over	vaak	wat	

Finnish (FI)

The drstopfi script creates a Stoplist preference containing the following Finnish words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
aina	hyvin	kesken	me	nyt	takia	yhdessd
alla	hoikein	kukka	mikd	oikea	tdssd	ylvs
ansiosta	ilman	kylld	miksi	oikealla	te	
ei	ja	kylliksi	milloin	paljon	ulkopuolella	
enemmdn	jdlkeen	tarpeeksi	milloinkan	sielld	vdhdn	
ennen	jos	ldhelld	koskaan	sind	vahemmdn	
etessa	kanssa	ldpi	mind	ssa	vasen	
haikki	kaukana	liian	missd	sta	vasenmalla	
hdn	kenties	lla	miten	suoraan	vastan	
he	ehkd	luona	kuinkan	tai	vield	
hitaasti	keskelld	lla	nopeasti	takana	vieressd	

French (FR)

The drstopfr script creates a Stoplist preference containing the following French words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	beaucoup	comment	encore	lequel	moyennant	près	ses	toujours
afin	ça	concernant	entre	les	ne	puis	sien	tous
ailleurs	ce	dans	et	lesquelles	ni	puisque	sienne	toute
ainsi	ceci	de	étaient	lesquels	non	quand	siennes	toutes
alors	cela	dedans	était	leur	nos	quant	siens	très
après	celle	dehors	étant	leurs	notamment	que	soi	trop
attendant	celles	déjà	etc	lors	notre	quel	soi-même	tu
au	celui	delà	eux	lorsque	notres	quelle	soit	un
aucun	cependant	depuis	furent	lui	nôtre	quelqu"un	sont	une
aucune	certain	des	grâce	ma	nôtres	quelqu"une	suis	vos
au-dessous	certaine	desquelles	hormis	mais	nous	quelque	sur	votre
au-dessus	certaines	desquels	hors	malgré	nulle	quelques-unes	ta	vôtre
auprès	certains	dessus	ici	me	nulles	quelques-uns	tandis	vôtres
auquel	ces	dès	il	même	on	quels	tant	vous
aussi	cet	donc	ils	mêmes	ou	qui	te	vu
aussitôt	cette	donné	jadis	mes	où	quiconque	telle	y
autant	ceux	dont	je	mien	par	quoi	telles	
autour	chacun	du	jusqu	mienne	parce	quoique	tes	
aux	chacune	duquel	jusque	miennes	parmi	sa	tienne	
auxquelles	chaque	durant	la	miens	plus	sans	tiennes	
auxquels	chez	elle	laquelle	moins	plusieurs	sauf	tiens	
avec	combien	elles	là	moment	pour	se	toi	
à	comme	en	le	mon	pourquoi	selon	ton	

German (DE)

The drstopde script creates a Stoplist preference containing the following German words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
ab	dann	des	es	ihnen	keinem	obgleich	sondern	welchem
aber	daran	desselben	etwa	ihr	keinen	oder	sonst	welchen
allein	darauf	dessen	etwas	ihre	keiner	ohne	soviel	welcher
als	daraus	dich	euch	Ihre	keines	paar	soweit	welches
also	darin	die	euer	ihrem	man	sehr	über	wem
am	darüber	dies	eure	Ihrem	mehr	sei	um	wen
an	darum	diese	eurem	ihren	mein	sein	und	wenn
auch	darunter	dieselbe	euren	Ihren	meine	seine	uns	wer
auf	das	dieselben	eurer	Ihrer	meinem	seinem	unser	weshalb
aus	dasselbe	diesem	eures	ihrer	meinen	seinen	unsre	wessen
außer	daß	diesen	für	ihres	meiner	seiner	unsrem	wie
bald	davon	dieser	fürs	Ihres	meines	seines	unsren	wir
bei	davor	dieses	ganz	im	mich	seit	unsrer	wo
beim	dazu	dir	gar	in	mir	seitdem	unsres	womit
bin	dazwischen	doch	gegen	ist	mit	selbst	vom	zu
bis	dein	dort	genau	ja	nach	sich	von	zum
bißchen	deine	du	gewesen	je	nachdem	Sie	vor	zur
bist	deinem	ebenso	her	jedesmal	nämlich	sie	während	zwar
da	deinen	ehe	herein	jedoch	neben	sind	war	zwischen
dabei	deiner	ein	herum	jene	nein	so	wäre	zwischens
dadurch	deines	eine	hin	jenem	nicht	sogar	wären	
dafür	dem	einem	hinter	jenen	nichts	solch	warum	
dagegen	demselben	einen	hintern	jener	noch	solche	was	
dahinter	den	einer	ich	jenes	nun	solchem	wegen	
damit	denn	eines	ihm	kaum	nur	solchen	weil	
danach	der	entlang	ihn	kein	ob	solcher	weit	
daneben	derselben	er	Ihnen	keine	ober	solches	welche	

Italian (IT)

The drstopfit script creates a Stoplist preference containing the following Italian 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"	
bensì	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	

Portuguese (PR)

The drstopfpr script creates a Stoplist preference containing the following Portuguese words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	bem	e	longe	para	se	vocj
abaixo	com	ela	mais	por	sem	vocjs
adiante	como	elas	menos	porque	sempre	
agora	contra	jle	muito	pouco	sim	
ali	debaixo	eles	nco	prsximo	sob	
antes	demais	em	ninguem	qual	sobre	
aqui	depois	entre	nss	quando	talvez	
ati	depressa	eu	nunca	quanto	todas	
atras	devagar	fora	onde	que	todos	
bastante	direito	junto	ou	quem	vagarosamente	

Spanish (ES)

The drstopfes script creates a Stoplist preference containing the following Spanish words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
a	aquí	cuantos	esta	misma	nosotras	querer	tales	usted
acá	cada	cuán	estar	mismas	nosotros	qué	tan	ustedes
ahí	cierta	cuánto	estas	mismo	nuestra	quien	tanta	varias
ajena	ciertas	cuántos	este	mismos	nuestras	quienes	tantas	varios
ajenas	cierto	de	estos	mucha	nuestro	quienesquiera	tanto	vosotras
ajeno	ciertos	dejar	hacer	muchas	nuestros	quienquiera	tantos	vosotros
ajenos	como	del	hasta	muchísima	nunca	quién	te	vuestra
al	cómo	demasiada	jamás	muchísimas	os	ser	tener	vuestras
algo	con	demasiadas	junto	muchísimo	otra	si	ti	vuestro
alguna	conmigo	demasiado	juntos	muchísimos	otras	siempre	toda	vuestros
algunas	consigo	demasiados	la	mucho	otro	sí	todas	y
alguno	contigo	demás	las	muchos	otros	sín	todo	yo
algunos	cualquier	el	lo	muy	para	Sr	todos	
algún	cualquiera	ella	los	nada	parecer	Sra	tomar	
allá	cualquieras	ellas	mas	ni	poca	Sres	tuya	
allí	cuan	ellos	más	ninguna	pocas	Sta	tuyo	
aquel	cuanta	él	me	ningunas	poco	suya	tú	
aquella	cuantas	esa	menos	ninguno	pocos	suyas	un	
aquellas	cuánta	esas	mía	ningunos	por	suyo	una	
aquello	cuántas	ese	mientras	no	porque	suyos	unas	
aquellos	cuanto	esos	mío	nos	que	tal	unos	

Swedish (SE)

The drstopfse script creates a Stoplist preference containing the following Swedish words:

Stop word	Stop word	Stop word	Stop word	Stop word	Stop word	Stop word
ab	de	emot	gott	lengt	ni	under
aldrig	ddr	en	hamske	lite	nu	uppe
all	de	ett	han	man	och	ut
alla	dem	fastdn	hdr	med	ocksa	utan
alltid	den	fvr	hellre	med	om	utom
dn	denna	fort	hon	medan	vver	vad
dnnu	deras	framfvr	hos	mellan	pe	vdl
enyo	dess	fren	hur	mer	se	var
dr	det	genom	i	mera	sedan	varfvr
att	detta	gott	in	mindre	sin	vart
av	du	fastdn	ingen	mot	skall	varthdn
avser	efter	fvr	innan	myckett	som	vem
avses	efteret	fort	inte	ndr	till	vems
bakom	eftersom	framfvr	ja	ndra	tillrdckligt	vi
bra	ej	fren	jag	nej	tillsammans	vid
bredvid	eller	genom	lengsamt	nere	trots att	vilken

B
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_OOBMBX	VARCHAR2(60)	Out-of-bound mailbox name for ConText server
SER_SESSION	NUMBER	No Longer Used
SER_AUDSID	NUMBER	ConText server audit session ID
SER_DBID	NUMBER	ConText server database ID
SER_PROCID	VARCHAR2(10)	No Longer Used
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	No Longer Used
SER_AUDSID	NUMBER	ConText server audit session ID
SER_DBID	NUMBER	ConText server database ID
SER_PROCID	VARCHAR2(10)	No Longer Used
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 predefined 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 predefined 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

Name	Null?	Type
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

C
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 administration tools.

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 created/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)	Index entry type (token or section)
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 for which DML has occurred (i.e. deletion or modification).

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 for use in the index.

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. It stores the DOCIDs of documents that have been modified or deleted from the text table.

Name	Type	Description
FIRST_DOC	NUMBER(38)	Internal use only
IDLIST	LONG RAW	Internal use only

DR_*nnnnn*_I1W

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_*nnnnn*_I1T*n*	DR_*nnnnn*_I1I*n*
DR_*nnnnn*_KTB	DR_*nnnnn*_KID
	DR_*nnnnn*_KIK
	DR_*nnnnn*_KSQ
DR_*nnnnn*_LST	DR_*nnnnn*_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.

D

External Filter Specifications

This appendix contains a list of the external filter formats supported for use in mixed-format columns. It also provides details for installation, setup, and usage of the external filters provided by ConText.

The following topics are covered in this appendix:

- Supported Formats for Mixed-Format Columns
- Supplied External Filters

See Also: For more information about external filters, see "External Filters" in Chapter 8, "ConText Indexing".

Supported 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
AmiPro 1.x - 3.1	19
AmiPro Graphics SDW Samna Draw	62
ASCII	90
AT&T Crystal Writer	46
AutoCAD (DXF, DXB)	53
CEOwrite 3.0	78
Computer Graphics Metafile (CGM)	79
CorelDraw 2.x and 3.x	59
CTOS DEF	75
DBase IV 1.0; DBase III, III +	37
DCA / FFT - Final Form Text	27

Document Format	ID
DCA / RFT - Revisable Form Text	0
Digital DX	15
Digital WPS-PLUS	47
EBCDIC	89
Enable 1.1, 2.0, 2.15	11
Encapsulated PostScript Preview; Encapsulated PostScript Bitmap	66
First Choice 3.0 Data Base	13
FrameMaker (MIF) 3.0; FrameMaker (MIF) 3.0 Win	42
Framework III, 1.0, 1.1	22
FullWrite Professional 1.0x	31
GIF (Graphical Interchange Format)	51
Harvard Graphics	87
HP Graphics Language (HPGL)	83
HTML Level 1, 2, 3	91
IBM Writing Assistant 1.0	16
IGES	52
Interleaf 5.2; Interleaf 5.2 - 6.0	32
JPEG (Joint Photographic Experts Group)	58
Legacy 1.x, 2.0	41
Lotus 123 4.x; Lotus 123 3.0; Lotus 123 1A, 2.0, 2.1	20
Lotus Freelance	85
Lotus Manuscript 2.0, 2.1	26
Lotus PIC	67
Macintosh Paint	88
Microsoft Windows Paint 2.x	70
Macintosh QuickDraw (PICT)	64
MacWrite 4.5 - 5.0	29
MacWrite II 1.0 - 1.1	30
Mass 11, Version 8.0 -8.33	36

Document Format	ID
MastSoft Graphics (MSG)	49
Micrografx Designer (DRW)	60
MS Access 2.0	39
MS Excel 5.0 - 6.0; MS Excel 4.0; MS Excel 3.0; MS Excel 2.1	21
MS Powerpoint for Windows 2, 3, 4	84
MS RTF; MS RTF (ANSI Char Set)	17
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
MS Word for Mac 5.0, 5.1; MS Word for Mac 4.0; MS Word for Mac 3.0	28
MS Word for Windows 2.0; MS Word for Windows 1.x	18
MS Word for Windows 6.0; MS Word for Mac 6.0	68
MS Works for Windows 3.0	69
MS Write for Windows 3.x	7
MultiMate 4; MultiMate Advantage II; MultiMate Advantage; MultiMate 3.3	6
Navy DIF (GSA)	35
OfficePower 7; OfficePower 6	44
OfficeWriter 6.0 - 6.2; OfficeWriter 5.0; OfficeWriter 4.0	9
OS/2 Bitmap; Windows Bitmap (BMP); Windows RLE	63
Paradox 3.5, 4.0	38
PC Paintbrush (PCX)	71
PDF (Adobe Acrobat)	57
PeachText 5000 2.1.2	82
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
Quattro Pro DOS; Quattro Pro Windows	45
Q&A 4.0; Q&A Write 1.x, Q&A 3.0	10
Rapid File 1.0	23
RGIP	61

Document Format	ID
Samna Word IV & IV + 1.0, 2.0	25
Sun Raster Graphics	65
TIFF (Tagged Image File Format)	50
Uniplex V7 - V8	77
Vokswriter 3, 4	74
Wang PC, Version 3	24
Wang WITA	55
Windows Clipboard	72
Windows ICON	73
Windows Metafile (WMF)	48
WiziDraw	86
WiziWord	56
Word For Word Intermediate Communications format (COM)	34
WordPerfect for Windows 6.1; WordPerfect for Windows 6.0; WordPerfect 6.0	1
WordPerfect 5.1 (Mail Merge)	2
WordPerfect for Windows 5.x; WordPerfect 5.1; WordPerfect 5.0	3
WordPerfect Graphics 1 (WPG)	4
WordPerfect Graphics 2 (WPG)	5
WordPerfect 4.2; WordPerfect 4.1	80
WordPerfect Mac 1.0	81
WordPerfect Mac 3.0; WordPerfect Mac 2.1; WordPerfect Mac 2.0	33
WordStar 5.0, 5.5, 6.0, 7.0	40
WordStar 2000, Rel 3.0	14
WriteNow 3.0	54
Xerox - XIF 5.0, 6.0	43
XYWrite IV; XyWrite III Plus	76

Supplied External Filters

ConText supplies a set of ready-to-use external filters, licensed from MasterSoft (Inso Corporation), which can be used for filtering documents in many of the most popular desktop publishing and word processor formats.

Availability of Filters

Because the external filters are supplied as executables, the availability of the filters is platform-dependent; however, the filters are generally supplied for the following platforms/operating systems:

- DEC Alpha
- Data General
- Hewlett-Packard HP 9000/9008
- IBM AIX
- Intel SVR4
- MS-DOS (Windows NT)
- SCO
- SGI SGI52/SGI52M
- SUN Solaris/Solaris X86/SunOS

See Also: To verify the availability of the filters on your platform/operating system, see the Oracle8 documentation specific to your operating system.

List of Filters

The following filters are supplied by ConText. 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	Executable Name
AmiPro for Windows	1, 2, 3	19	amipro	The executable names are operating system specific. It is generally not necessary to know the names of the filter executables because ConText supplies wrappers for each executable and the wrapper name is the value specified in the Filter preference(s) that use the filter. However, if you find it necessary to modify the wrappers, you may need to know the names of the filter executables. For more information about the names of the executables, see the Oracle8 installation documentation specific to your operating system.
Lotus Freelance for Windows	2	85	lotusfre	
Lotus 123	2, 3, 4	20	lotus123	
Lotus 123	5	N/A	lotus123	
MS Excel	5	21	msexcel	
MS Excel	7	N/A	msexcel	
MS PowerPoint for Windows	2, 3, 4	84	power234	
MS PowerPoint for Windows	7	N/A	power7	
MS Word for DOS	5.0, 5.5	8	worddos	
MS Word for Macintosh	3, 4, 5	28	wordmac	
MS Word for Windows	2	18	wordwn2	
MS Word for Windows	6	68	wordwn67	
MS Word for Windows	7	N/A	wordwn67	
PDF (Adobe Acrobat)	N/A	57	acropdf	
WordPerfect for DOS; WordPerfect for Windows	5.0, 5.1; 5.x	3	wp5	
WordPerfect for DOS; WordPerfect for Windows	6.0; 6.x	1	wp67	
WordPerfect for Windows	7.0	N/A	wp67	
Xerox XIF	5, 6	43	xeroxxif	

Note: The PDF filter has a status of production for the following platforms: DEC, HP, IBM AIX, MS-DOS (Windows NT), SGI, and SUN (Solaris, Solaris X86, and SunOS)

For all other platforms for which ConText supplies external filters, the PDF filter has a status of BETA.

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 2.4 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 ConText release 2.0 and already have wrappers for the external filters provided in the previous release, as well as preferences that call the wrappers, choose one of the following actions:

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

Otherwise, you must drop your indexes, policies, and preferences, then create new preferences, policies, and indexes that use the new wrappers/filters.

Note: If you are using the BETA PDF filter provided in previous releases and wish to use the new production PDF filter, 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 usage issues:

1. The wrapper name (e.g. ’amipro’), *not* the executable name, must be specified 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".

Index