
# TeraXML Enterprise Search

# Version 6.0

# Programmer's Guide

Revision 6.1

Last Updated April 22, 2005

This is a Doclinx, Inc. Confidential document.

Unauthorized distribution of this document is prohibited.

Copyright © 2005 Doclinx, Inc. All rights reserved.

# Table of Contents



 Copyright © 2005, Doclinx, Inc.

 Copyright © 2005, Doclinx, Inc.

 Copyright © 2005, Doclinx, Inc.

 Copyright © 2005, Doclinx, Inc.

 Copyright © 2005, Doclinx, Inc.

# 1. Introduction

TeraXML Enterprise Search is a high-performance, scalable, XML and full-text Information Retrieval and Information Extraction (Entity Extraction) system for processing documents in various formats and languages.

TeraXML provides the following core capabilities:

- Capable of indexing and searching terabyte-sized data sets.
- Contextual search of XML data using the W3C XPath standard.
- Part-of-Speech Tagging, Sentence Boundary Detection and Noun-Phrase Detection.
- Named Entity Extraction for Names, Locations, Organizations and Dates.
- Support for most unstructured document formats including HTML, Adobe PDF and Microsoft Office and 200 other formats.
- Industry's fastest indexing time of 6 gigabytes per hour using a single indexing server.
- The industry's smallest index files (35% to 90% of document size).
- Virtually unlimited scalability using multiple load-balanced search servers.
- Powerful query language includes advanced features like Boolean operators, fuzzy search, wildcards, proximity search, and stemming.
- Unicode implementation provides native support for European, Middle-Eastern and Oriental languages.
- 64-bit architecture supports multi-terabyte data sets.
- 100% Java version for any platform with a supported Java 1.4 Virtual Machine.

This document described Information Retrieval and Linguistics APIs offered by TeraXML.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

# 2. TeraXML Architecture

This section describes the core technologies used in TeraXML Enterprise Search.

## 2.1 Overview

TeraXML is designed for very-large scale collections (measured in terabytes) where indexing speed of many gigabytes per hour are required. It supports multithreaded, multiple load-balanced search servers allowing for virtually unlimited user-base scalability.

TeraXML provides several query features such as Boolean, wildcard, proximity, fuzzy and stemming. XML contextual searching is done using the W3C XPath standard to specify XML elements and/or attributes.

The system uses 64-bit data structures allowing for very large collections. Character data is stored using Unicode, which allows TeraXML to support European, Middle-Eastern and Oriental language collections that are searchable in their native language. TeraXML automatically selects an appropriate word-break algorithm based on the current language being indexed.

TeraXML provides four types of services:

1. Capture a broad range of unstructured data.
   - Multiple languages.
   - Multiple repositories (files, websites, databases, lotus notes etc.).
   - Multiple document formats (HTML, XML, PDF, MS Office and over 200 others).
2. Tokenize the captured data and perform linguistic analysis to extract part-of-speech tags, sentence boundaries and Named Entities from the captured data.
3. Provide very high speed indexing of the tokenized data.
4. Provide fast, flexible search of document content and/or extracted entities to find relevant information.

TeraXML converts data from all inputs into a common format called "Standard Token Format" or STF which is stored in the Unicode character set. TeraXML's "indexing" components process the STF and build the index also referred to the "TeraXML Text Database". The "search" components process search queries provide results from the Text Database.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

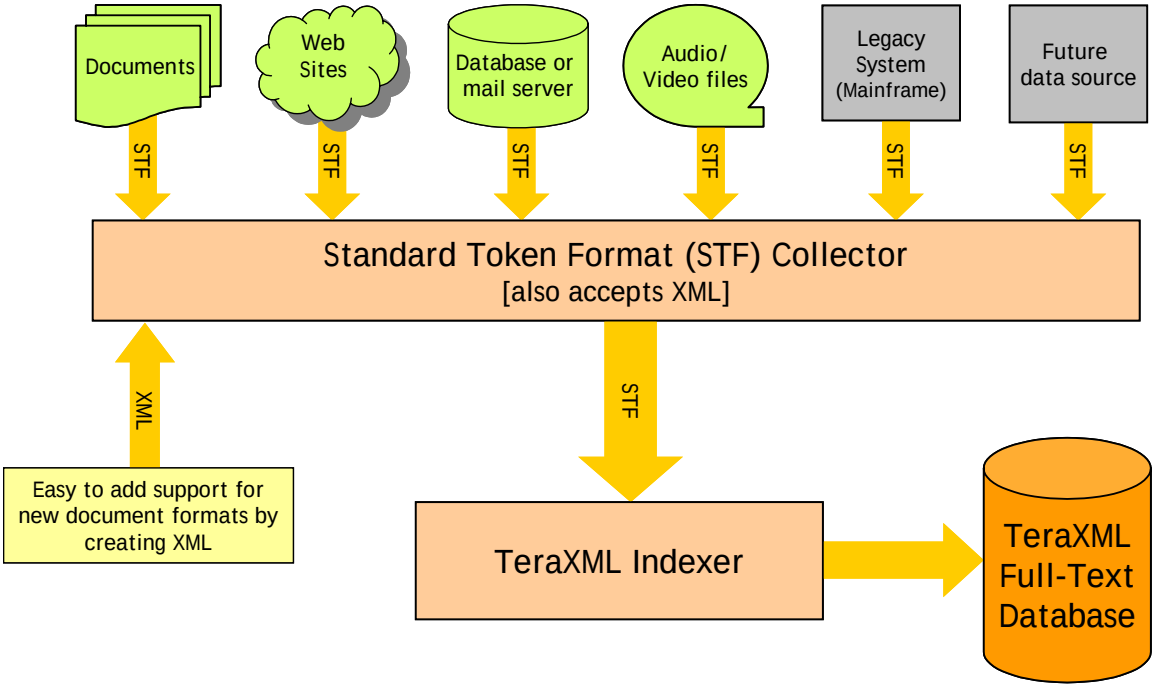


**Figure 1 TeraXML Process Flow**

TeraXML's Capture and Index services form the "Indexing Subsystem". Part-of-Speech tagging, morphological analysis and Named Entity detection services are provided by TeraXML's Linguistic Subsystem. Search services are provided by the "Search/Retrieval Subsystem".

The Indexing Subsystem takes documents as input and processes the context, words, and properties into an inverted index that provides fast lookup and searching of word combinations.

The Search/Retrieval Subsystem processes a search request and provides the search result locations into the original document set.

## 2.2 Indexing Subsystem

TeraXML is based upon the concept of a catalog metaphor for the organization of its data and functions. A catalog defines the set of files (or URLs) that comprise a collection. There can be multiple collections or catalogs, which can be individually built, maintained and searched by the software. Both a C and a Java API are available allowing external applications to use indexing and search functionality.


Copyright © 2005, Doclinx, Inc.

### 2.2.1 Catalogs

A catalog is defined as a single searchable set of documents. Catalogs support the following operations:

- Document Addition
- Document Deletion
- Document Replacement
- Catalog Synchronization (also called Catalog Update or Catalog Build)
- Catalog Merge

Documents can be added, deleted or replacement in a catalog. At discrete intervals, the catalog is synchronized (or updated) to reflect the changes (additions, deletions or replacements) since the last synchronize operation. This step causes the changes in the catalog to be searchable.

It is assumed that the initial build of the catalog comprises a large number of files that creates a "primary" database. Additions and deletions to a catalog are managed in an "update" database. The update and primary databases can be merged at any time. The merge point is determined by application requirements.

TeraXML catalogs support concurrent indexing and search. Search queries can be processed as documents are added, deleted or updated into the catalog. Whenever a catalog is "updated", i.e. all open search handles notified of the change immediately.

### 2.2.2 STF Generation

All input documents are first processed into an intermediate format called Standard Token Format (STF). This is accomplished by a module called an "STF filter". TeraXML provides three standard STF filters for processing HTML, XML, text and over other 200 document formats.

TeraXML allows document meta-data, such as document properties to be specified at this time. The STF Generation process and standard TeraXML filters are described in section 3.

Additional STF filters can be easily implemented to address special needs. TeraXML can index and subsequently search any document that has been converted to STF.

Files that are "added" to a catalog are immediately converted into STF using the appropriate filter. STF generated from an "added" file is concatenated to from a single STF file which can be indexed subsequently.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

### 2.2.3 Catalog Synchronization

After multiple files have been added or deleted, the catalog is synchronized. The synchronize operation builds the final, searchable full-text database from an STF input. This operation consists over several steps, which include *Dictionary Build*, *Inversion* and *Optimization*. These steps are described in detail in section 3.1.

## 2.3 Search Subsystem

Once a catalog is synchronized, TeraXML is able to provide search and retrieval services to an application or a server process. The search API provides mechanisms for specifying a search query, ordering results, and retrieving the result set. Since searching is a read-only operation, several instances of TeraXML can be running and searching the same database concurrently.

TeraXML can also optionally build alternative data structures that enhance search lookup. The alternative data structures allow provide stemming and fuzzy lookup functionality. Stemming is the concept of taking words and folding them into root or base words by removing prefixes, suffixes, plurals, tense, and other adaptations to the "core" word. Searches then can find slight word variations by using a single version of the word. Fuzzy lookup structures are a decomposition of words into their phonetic elements. This allows searches to compensate for misspellings in both the input text and/or the search queries.

## 2.4 Linguistics Subsystem

TeraXML analyzes textual data, performs linguistic analysis and extracts meaningful data from the supplied text.

TeraXML linguistics subsystem consists of several modules which can process English, French, Italian, German, Spanish, Chinese, Japanese, Korean and Arabic.

### 2.4.1 Tokenizer

The Tokenizer module (TOK) provides word tokenization functionality based on the algorithms provided in Chapter 5 of The Unicode Standard 3.0 and UAX #29 Text Boundaries in Unicode 4.0.0. Tailored segmentation is available for English to facilitate further linguistic processing. Specifically, the tokenizer separates plural endings ("'s", "s'", and "z'") from their base form.

The tokenizer module returns two results – the token itself and its offset within the input text stream.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

### 2.4.2 Part-of-speech Tagger

Part-of-Speech tagging is the process of identifying the grammatical parts of a sentence (called "POS tags", such as nouns, verbs, adjectives, adverbs, prepositions, etc. TeraXML's part-of-speech (POS) tagger does three things: it assigns possible parts of speech to each word, it guesses the part of speech for unknown words, and it decides which part of speech is the correct one for words with multiple possible POS tags. In general, this is done by looking at the context: the words and parts-of-speech adjacent to it.

English Part-of-Speech tags are shown below:

| Tag | Description | Description/Example |
|---|---|---|
| AUX | finite auxiliary verb | "Helper" verbs: do, be, have |
| AUXG | progressive auxiliary verb | "Helper" verbs in "-ing" form: doing, having, being |
| CC | conjunction | Joins words and phrases: and, yet, but |
| CD | cardinal number | Counting numbers 1,2,3 or "thousand dollars". |
| DT | determiner | The red ball. A pause to consider. |
| EX | expletive "there" | There is a monster under the bed. |
| FW | foreign words | Non-English words appearing in English text: adieu, mañana |
| IN | preposition | Links pronouns, nouns, or phrases to other parts of a sentence, and frequently indicates relationships in time and space:<br>The dog is on the couch.<br>He wrote the paper during study hall. |
| JJ | adjective | Describes a noun: The quirky professor kept odd office hours. |
| JJR | comparative adjective | "-er" adjectives: the shorter boy, the older woman |
| JJS | superlative adjective | "-est" adjectives: The tallest player on the team. |
| LS | itemization tokens | Like the roman numerals in the beginning of a book: i, ii, iii, iv. |
| MD | modal verb | Expresses ideas such as possibility or intention: can, could, would, need. |
| NN | common noun | A "thing": car, truck, building. |
| NNP | proper noun | Generally, things that are capitalized, such as personal or place names: Tom Cruise, Paris. |


Copyright © 2005, Doclinx, Inc.

| | | |
|---|---|---|
| NNPS | plural proper noun | Wal-Marts |
| NNS | plural common noun | Cars, trucks, buildings |
| PDT | predeterminer | All the trees |
| POS | possessive "'s" | That is Casey's bat. |
| PRP | Pronoun | Stands in for a person (or people) or thing: He was late for work. They are not going to win this game. It is on the nightstand. |
| PRP$ | possessive pronoun | Show ownership: yours, mine, ours, his, hers, its. |
| RB | Adverb | Usually modifies a verb and ends in "-ly": quickly sinking, sang loudly. |
| RBR | Comparative adverb | more, less |
| RBS | superlative adverb | most, least |
| RP | deictics (directional adverbs) | She stumbled backwards. |
| SYM | symbols | Non-punctuation marks: ® © > < |
| TO | the preposition "to" | To |
| UH | interjection | D'oh! |
| VB | base form verb | The infinitive: to spend, to save. |
| VBD | past tense verb | Action happening in the past: played, sang. |
| VBG | progressive aspect verb | Ending in "-ing": laughing, choking. |
| VBN | past participle verb | Generally ending in "-ed" or "-en": *written, passed* |
| VBP | verb base forms | The infinitive used in present tense |
| VBZ | third-person singular verb | Frequently ends with "-s": She *sends* flowers. |
| WDT | interrogative determiner | *Which?* |
| WP | interrogative pronoun | *Who? What?* |
| WP$ | interrogative possessive pronoun | *Whose?* |
| WRB | interrogative adverb | *When? Where? Why?* |


Copyright © 2005, Doclinx, Inc.

| | | |
|---|---|---|
| $ | dollar sign | They pay *$*25/hour. |
| `` | start quotation mark | He said "No, I won't… |
| " | end quotation mark | …do that" |
| , | Comma | Pears**,** oranges**,** apples. |
| . | Period | Used at the end of a sentence or for abbreviations: Mr**.** Smith went to Washington**.** |
| ( | left parenthesis | ( |
| ) | right parenthesis | ) |
| : | other punctuation | ; ! ? |

### 2.4.3 Sentence Boundary Detection

The Sentence Boundary Detection (SBD) module detects the start and end of each sentence in the input text stream. For English, this process is complicated by the fact that the period that is used to end a sentence is also used within a sentence for other purposes (such as abbreviations). Periods need not end sentences when they end an abbreviation; however, periods can end abbreviations and at the same time end a sentence. They also interact with other punctuation, especially quotation marks.

### 2.4.4 Base Noun Phrase detection

The Base Noun Phrase Detection module (BNP) detects Base Noun Phrases. One of the most important kinds of structure to assign to a document is the identification of noun phrases (NP). A phrase is a self-contained group of words with a discrete meaning; a noun phrase is a phrase that functions as a noun in a sentence. Noun phases can also be recursive. That is, a noun phrase may contain other noun phrases as component parts. For instance, the following are all noun phrases:

1. it
2. apples
3. the apple
4. the green apple
5. the round red juicy apple
6. the green apple on the table
7. the red apple on the table in the kitchen
8. the red apple that I ate at lunch yesterday

A base noun phrase is a noun phrase that is not recursive, that is, it doesn't contain other noun phrases inside it. So, the first four noun phrases in the list above are base noun phrases, and the

 Copyright © 2005, Doclinx, Inc.

remaining ones are complex noun phrases that contain base noun phrases inside them. Below, the list of noun phrases is repeated with the base noun phrases bracketed:

1. [it]
2. [apples]
3. [the apple]
4. [the green apple]
5. [the round red juicy apple]
6. [the green apple] on [the table]
7. [the red apple] on [the table] in [the kitchen]
8. [the red apple] that [I] ate at [lunch] yesterday

### 2.4.5 Named Entity Extraction

TeraXML's Named Entity (NE) module finds named entities in documents and classifies them into person, organization, location, or date. It looks at both the named entity and its context to determine whether it's a named entity, and if so what kind it is.

A Named Entity is a proper name. It can be the name of a person ("George Bush"), or an organization ("The White House"), or a location ("Washington"). It can also be a particular date ("July 14, 1789", but not "Tuesday").

### 2.4.6 Supported Languages

The following table describes the languages currently supported by the TeraXML Linguistics Subsystem.

| Language | Tokenization | POS | SBD | BNP | Named Entites | Stemming | Compounds | Readings |
|---|---|---|---|---|---|---|---|---|
| English | yes | yes | yes | yes | yes | yes | no | n/a |
| Chinese | yes | yes | yes | no | no | n/a | n/a | yes |
| Korean | yes | yes | no | no | no | yes | yes | no |
| Japanese | yes | yes | yes | yes | yes | yes | yes | yes |
| German | yes | yes | yes | yes | yes | yes | yes | n/a |
| French | yes | yes | yes | yes | no | yes | no | n/a |
| Italian | yes | yes | yes | yes | no | yes | no | n/a |
| Spanish | yes | yes* | yes | yes | no | yes | no | n/a |
| Arabic | yes | yes* | yes | no | yes | yes | no | no |

 Copyright © 2005, Doclinx, Inc.

## 2.5 Document Parsing Subsystem

TeraXML can process text contained in many different document formats. TeraXML offers built-in support for parsing XML, HTML, Text, PDF and MS Word documents. Additionally, over 200 other document formats are supported for which the parsing functionality is provided by a set of shared libraries which are available for a variety of platforms.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

# 3. TeraXML Internals

## 3.1 Indexing Subsystem – STF Generation

As files are added to a catalog, a parser (also called a "TeraXML Filter") is selected to parse the input file. TeraXML Filters output a data stream in a format called "STF" (Standard Token Format). All documents input to TeraXML are converted to STF for subsequent indexing.

TeraXML includes several filters to handle different document types and to perform lexical analysis on the input documents. The TeraXML API also provides mechanisms to create custom filters to handle new document types or data sources.

HTML filter: This filter handles HTML files. It extracts title and meta-tag information from the input HTML files. As web content frequently contains incorrect or malformed HTML, this filter is specially designed to handle those cases.

XML filter: This filter handles valid XML files and forms the context tree that allows contextual searching of XML documents. It checks XML files for validity only and does not perform any XML parsing. It is schema or DTD independent and can accept any valid XML file.

Generic filter: This filter handles over 200 file formats other than HTML, XML, and plain text. It is capable of extracting words from many different file types. The Generic filter can determine the file type by reading the first few bytes of the file and is not dependent on the extension of the input file.

This filter also detects a pre-determined set of document properties that can be converted into a context tree and used for searching.

Text filter: This filter handles text files and HTML files. It is extremely fast since it only extract words in the input stream and ignores all other information. It is used where speed is of utmost importance.

PDF filter: This filter handles PDF files. It internally converts PDF files to an XML representation which is contextually indexed. PDF metadata and properties are preserved in the generated XML. TeraXML's contextual XML indexing allows searches to be limited by or to the metadata (properties) stored in the PDF document.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

MS Word filter: This filter is identical to the PDF file except that it can handle MS Word documents.

Individual files, entire directories or lists of files can be added to a catalog. Contents of each file added to the catalog are parsed by one or more filters which generate Standard Token Format (STF). The STF stream contains all words, meta-data and context information required to create the full-text database.

It should also be noted that the word break algorithm is encapsulated in the parsing routines. All determination of what characters comprise a "word" is defined in the filters. Regular expressions may be specified to modify the built-in word-break algorithm. Each filter emits STF tokens indicating words, punctuation, markup, and context. The XML filter builds an XML "Context Tree" that allows the user to specify an XPath to perform contextual search queries.

STF creation can be controlled by setting several options through the TeraXML API.

### 3.2 Indexing Subsystem – Catalog Synchronization

The Catalog Synchronization processes any pending changes (additions, deletions or updates) and creates the searchable Text Database. The process is also referred to a "Catalog Build". This process consists of the following steps:

- Dictionary Build
- Locator Inversion
- Optimization
- Full-Text Database Creation

A single API call performs all the steps of building the full-text database. Each step supports several options that are specified in parameter blocks defined for that step.

An application first creates a new catalog entry in a catalog root directory. It is envisioned that this root will be the single location where sets of catalogs reside. Once a catalog is created, a collection of files is added to the catalog. After all of the original set of files is added a call is made to synchronize the catalog. This call starts the catalog build steps to yield a searchable full-text database.

After a catalog is initially built, files can then be added, deleted or replaced in the catalog. When these changes to the collection are ready for indexing, another synchronize operation is performed. All changes made after the initial catalog build are stored in an "update" database. However, the search software treats the two separate databases (primary and update) as one logical entity. All index structures created for the two separate databases can be merged together

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

for optimal search performance. This merged database is identical to a database that was built in a single operation.

### 3.2.1 Dictionary Build

The Dictionary Build step scans the STF file and finds all unique words in the input data set. TeraXML supports multiple dictionaries called Dictionary Regions, which are logically separate. Up to 16 Dictionary Regions can be defined. Each Dictionary Region can be further subdivided into sub-regions. Dictionary sub-regions are a performance optimization to enhance various classes of field specific search behavior and are optional. Dictionary sub-regions may be classified as being of a specific "type" as follows:

- Text
- Hypertext Link
- Integer (32-bit)
- Float (32-bit)
- Date
- Time
- Money

The Dictionary Build step optionally constructs stemming or fuzzy search data structures.

TeraXML supports user specified "stopwords". Stopwords are those words that occur very frequently and therefore their utility in searching is marginal. The removal from the full-text database provides a considerable performance enhancement and index size reduction.

### 3.2.2 Locator Inversion

The Locator Inversion step consists of building a list of locations (Locators) where all words in each dictionary occur. Locators are packed data structures that define a word's physical location in the input data set, and also store context or type information associated with that word. The number of Locators is directly proportional to the input data size and therefore it can be very large. Due to this, Locator Inversion requires the most processing time of all build steps.

### 3.2.3 Optimization

The Optimization step reorganizes the Dictionary and Locator data structures created in prior indexing steps. It also performs validity checks on the data. This step is essential in obtaining high performance searches.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

### 3.2.4 Full-Text Database Creation

Full-Text Database Creation is the final indexing step. It creates the full-text database file from intermediate files created in prior indexing steps. This is the only file required by TeraXML to perform full-text searches on the input documents.

The Full-Text Database can be optionally encrypted in this step to prevent unauthorized access to the database.

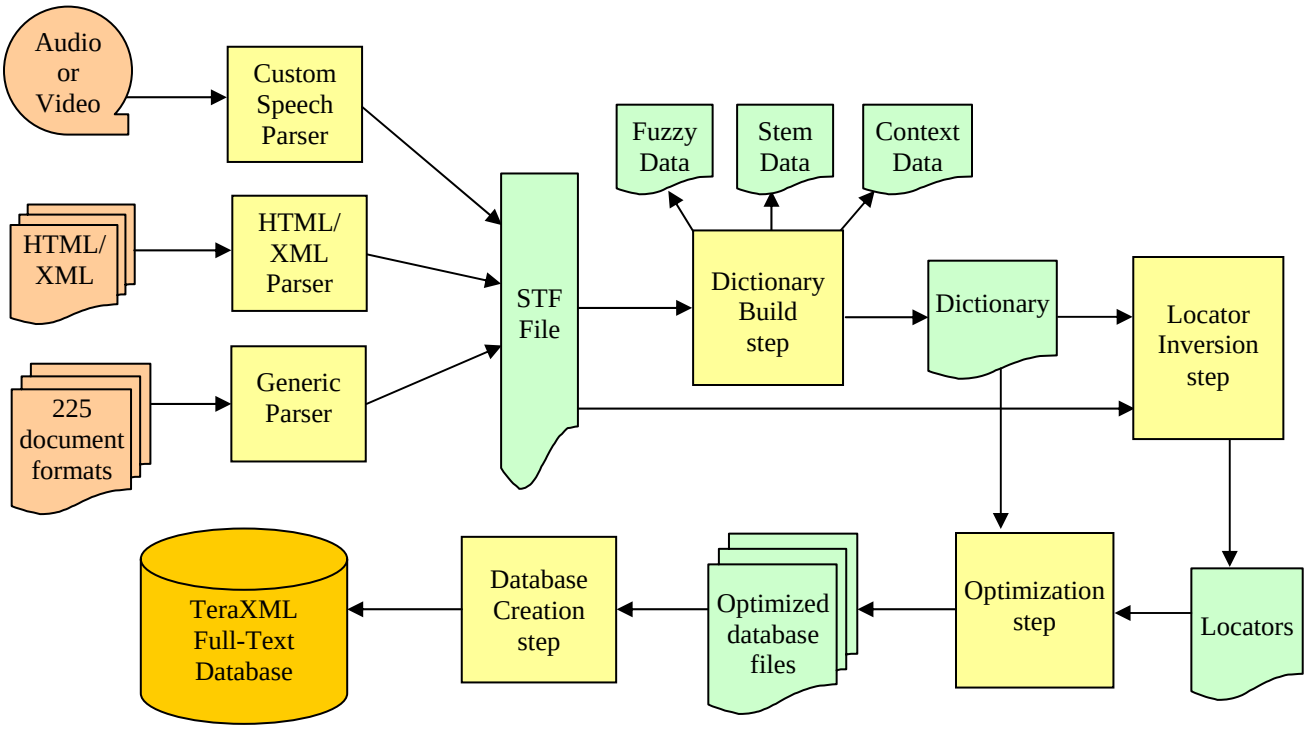


**Figure 2 Indexing Subsystem Process flow**

## 3.3 Indexing Subsystem – Catalog Merge

Any changes made after the initial creation of a catalog are saved in the "update" full-text database. Consequently, the "update" full-text database continues to grow as more and more changes are applied to the catalog. The Catalog Merge process combines the primary and update full-text databases and all associated data structures to produce a single, merged primary full-text database.


Copyright © 2005, Doclinx, Inc.

Catalog Merge involves merging the thesaurus structures, context trees, dictionaries and locators. There is also an interaction with the context and locator merging required because joining context trees can change the values of context attributes. A mapping array is created in the context process that is passed to the dictionary/locator merge operation in order to fix up these values.

Thesaurus merge is a simple process where the two associative mapping arrays are merged into one common map. New items from the update map are simply added to the result map while duplicate entries are removed.

The Dictionary Merge process merges the primary and update Dictionaries into a new combined primary Dictionary. Similarly, the Locator Merge process merges the primary and update Locators into new combined primary Locators. Any changes required to the output Locators are provided by the mapping array. This mapping change can also result in an update to the locator-packing schema.

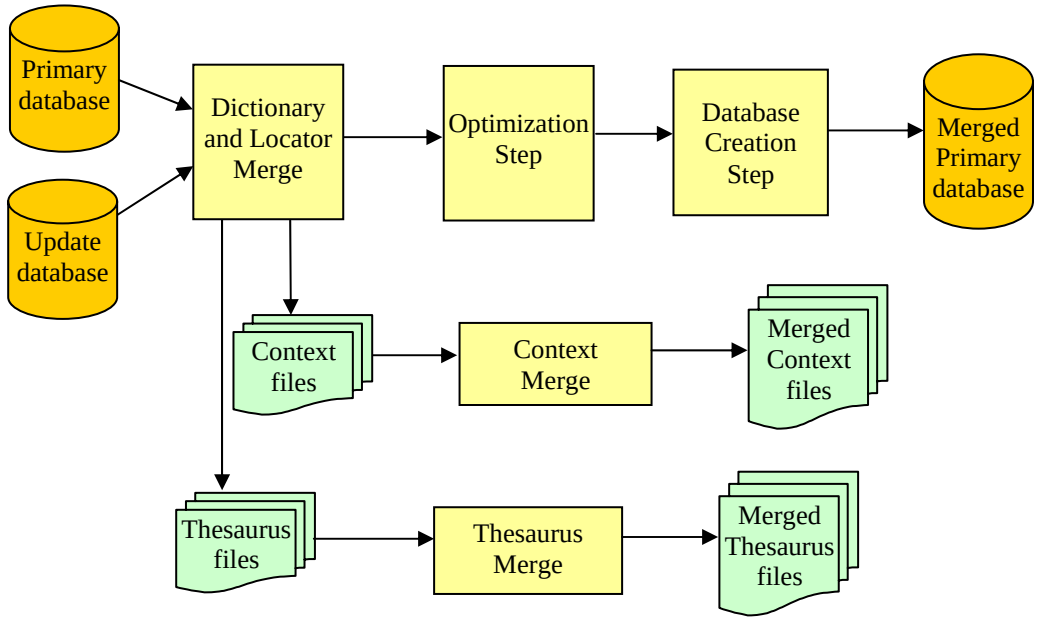


**Figure 3 Catalog Merge Process Flow**

### 3.4 Search Subsystem

Catalog Search provides the functionality to perform full-text search queries on the entire catalog. Catalog Search is designed for fast concurrent (multithreaded) searching.


Copyright © 2005, Doclinx, Inc.

Catalog search uses a list metaphor to describe the results of a search. For each search, a list is created, the search performed, and the results are retrieved from the list. A search query is specified in terms of the TeraXML query language. The XPath syntax is supported for performing XML contextual searches. Catalog search provides the following search features:

- Full contextual search of XML documents using XPath
- Standard Boolean searches (and, or, not)
- Wildcard searches
- Proximity searches
- Fuzzy searches
- Stemming
- Natural Language searches
- Range searches based on data type
- Parametric (fielded) searches

While the catalog mechanism supports both a primary and an update database, searches and the corresponding results are seamlessly merged by the software to give the appearance of a single, logical collection of documents.

When a catalog is updated, all search objects are notified that the catalog has been updated so that any data structures within applications using the system can be updated. For example, if an application initially determined that there were *x* items in the catalog when first opened, then it would be notified of a change so it could then determine via the API the new total number of documents in the Text Database.

### 3.5 Linguistics Subsystem

The Linguistics Subsystem functions as a server within TeraXML and provides the linguistics services within TeraXML. Data communication with this sub-system is through XML messages. The results of linguistic processing are returned as XML for further processing by other TeraXML modules.

### 3.6 Document Parsing Subsystem

Document Parsing is the process of extracting text from document in various formats. The Document Parsing module within TeraXML functions as a server. Documents are sent to the server and the extracted text is returned by the server in an XML format for processing by other TeraXML modules.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

TeraXML offers built-in support for parsing XML, HTML, Text, PDF and MS Word documents. Additionally, TeraXML supports over 200 other document formats from which it can extract text to be indexed. Parsing functionality for these other documents is provided by a set of native shared libraries that are available for a variety of operating systems.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

# 4. Key Data Structures and Concepts

## 4.1 Locators

Locators are bit-packed data objects that are byte aligned. A locator indicates where and in what context a word location occurs. All locators have the location information along with optional type or context information. Location information is the DOC, PAR, and WORD fields. DOC is the document ID corresponding to an individual file. The PAR field is a paragraph number. Paragraphs are determined by the parser logic. With the generic parser (document formats other than HTML, XML and text), the filter software automatically picks up this information. The HTML/XML filter uses a definition file that tells which tags are to be interpreted as paragraph markers.

A locator can be defined by a sequence of integer pairs {i,l} where **i** is the index of the attribute and **l** is the length in bits. A pack/unpack data structure defines a locator mapping for an individual database. This structure is an array on integer pairs defining the locator attribute fields present:

e.g. {{0, 7}, {3, 5}, {6, 10}, {15, 9}} word indicate a locator with attributes 0, 3, 6, and 15 present. The size of the locator would be 31 bits or 4 bytes in length.

## 4.2 Attributes

TeraXML provide 16 attributes, each 16 bits in length. Two attributes can be combined to store 32-bit information. Some of the attributes are reserved by the system while others are available for to the user for custom parametric search applications. For example, line number and page numbers could be stored to enable searching by line or page numbers.

Reserved attributes are shown in the table as follow:

| Attribute Number | Predefined usage |
|---|---|
| 0 and 1 | Document ID |
| 2 | Document ID prime (extension) |

 Copyright © 2005, Doclinx, Inc.

| | |
|---|---|
| 3 and 4 | Paragraph number |
| 5 | Paragraph number prime (extension) |
| 6 | Word offset |
| 7 | Title (or heading) level |
| 8 to 14 | User defined (available for customization) |
| 15 | XML context identifier |

A special system attribute (attribute 16) is reserved for bit settings and is not used in the catalog search schema.

### 4.3 Virtual Arrays

Both the catalog system and the basic search and retrieval functions use virtual arrays. Virtual arrays provide a seamless mechanism for handling large data sets backed by a file. The method provides fast access to the data without the overhead of dealing with file system details. The TeraXML kernel must be able to handle variable sized lists of objects that can be quite large. Therefore, a simple virtual buffering system that supports unlimited size arrays in an efficient manner simplifies the coding effort. Note that the virtual array system does NOT do the virtual memory management; that function is supplied by the LRU buffer manager. The system provides a way to "lock" current memory to avoid contention for critical sections.

The header file varray.h defines the virtual array buffering methods.

The following prototypes partially describe the functionality of the virtual array system:

```
INT vaOpen(VARRAY *va, CHAR *fileName, VOID *mem, UINT memSize, UINT32
vaSize, INT eltSize, UINT16 flags);
```

This method creates a virtual array with a memory buffer of size *memSize*. Each array component is of size *eltSize* and the upper limit to the virtual array is *vaSize*. A named file can be used for backing store. The flags control several operational characteristics of the array. For instance, the upper size of the array can be "Unlimited" which allows as much memory as the file system permits.

```
VOID *vaAddr(VARRAY *va, UINT32 elt);
```

 Copyright © 2005, Doclinx, Inc.

This method returns the address of element *elt* in virtual array *va*.

```
VOID *vaPtr(VARRAY *va, UINT32 elt, INT size);
```

This method returns a pointer element *elt* of the virtual array *va*. The returned pointer defines a piece of memory at least *size* bytes in length. The memory buffer is "locked" during this operation to ensure that the LRU system does not "re-use" the memory.

```
VOID  vaClose(VARRAY *va);
```

This method destroys the virtual array *va*.

### 4.4 LRU Hard disk & Database Buffering

A sophisticated virtual buffering system for CD-ROM and hard disk files is essential for TeraXML's performance. The buffer management system is independent of the virtual array and File I/O abstractions. This allows sharing of buffers across multiple data objects. The system works with a pool of 2 KB buffers. These buffers will be entered into a pool that provides memory for all I/O and virtual array buffering. Buffer requests are multiples of 2 KB. Memory can be moved around in the pool to allow larger blocks to be made by compacting memory. The buffering strategy is designed to allow flexibility in terms of buffer management along with providing better performance based upon frequency of use and LRU aging parameters. The header file lru.h defines the LRU buffering functions.

#### 4.4.1 Config.sys or template.sys

The template.sys file contains several indexing and tuning parameters. Following are example entries found in the template.sys file. This is the prototype *.sys* file for all indexing runs. The build software sets most of the values and the ones that can be changed are few and enumerated in the following discussion.

**dpControl.parAttrs** - This is the list of attributes that are NOT reset upon encountering a new paragraph. Note that attribute values 9 and 10 are ignored because they are not included in the locator output as dictated by the several regions defined subsequently. In this example, multiple paragraph titles are allowed. Excluding this value will cause title attributes to be set to 0 on every new paragraph.

All sections starting with region: These define the separate dictionary regions and are numbered 0 to 15. If three region block are defined, then DRIs 0, 1, 2 can be used. The region settings control individual locator and dictionary characteristics. The region.type controls the data type for the dictionary (e.g., text, link, date, INT32, etc.).

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

- region.type: type of region (enclosed in "")
- region.sub_aaidx: indicates in DRI is organized into sub-regions based on the index.
- region.sub_length: # of bits to required for the sub-region aaval
- region.locaAttrs: Vector of integers designating aaidx values to include in locators.
- primaryDpData.maxAttValue This is an output value that shows the maximum value for each attribute index.
- primaryDpData.attLength This array indicates the number of bits required for each locator aaidx value.

```
dpControl.createOutline       = FALSE;
dpControl.createCitref        = FALSE;
dpControl.createIVA           = FALSE;
dpControl.parAttrs            = {7,9,10,15};
dpControl.formAttrs           = 0;
dpControl.stopThreshold       = 0;
dpControl.titlePrefixLength   = 128;
dpControl.docTitleThreshold   = 0;

region.type                   = "text";
region.excludeMask            = 0;
region.excludeValue           = 0xffff;
region.sub_aaidx              = 9;
region.sub_length             = 4;
region.locAttrs               = {"0","7","8","15"};

region++;

region.type                   = "hyperlink";
region.excludeMask            = 0;
region.excludeValue           = 0xffff;
region.sub_aaidx              = -1;
region.sub_length             = 0;
region.locAttrs               = {"0"};

region++;

region.type                   = "text";
region.excludeMask            = 0;
region.excludeValue           = 0xffff;
region.sub_aaidx              = -1;
region.sub_length             = 0;
region.locAttrs               = {"0","8"};

primaryDpData.maxHeadingLevel = 1;
primaryDpData.maxDocHeadingLevel = 1;
primaryDpData.docIdxRecordLength = 138;
primaryDpData.collocateTopTitles = FALSE;
primaryDpData.attFlagOrMask   = 0;
primaryDpData.maxAttValue     = {6,0,0,2,0,0,119,1,1,14,0,0,0,0,0,0};
primaryDpData.attLength       = {3,0,0,2,0,0,7,1,1,4,0,0,0,0,0,0};
primaryDpData.dbOffsets        = {0,0,0,0,0,0,0,0,0,16384,0,0,18432,22528,
                                  24576,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                  0,0};
primaryDpData.dbLengths        = {0,0,0,0,0,0,0,0,0,1330,0,0,4096,2048,7,0,0,
                                  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
primaryDpData.dbVolumeSeqs     = {-1,-1,-1,0,0,0,-1,-1,0,0,0,0,0,0,0,-1,-1,-1,-1,
                                  -1,0,0,-1,0,0,0,0,0,0,0,0,-1,0,0,0,0,0};
```

## 4.5 Context Trees

Context relationships are represented as an n-ary tree with an associate lookup that enables locating sub-trees quickly. Context trees use *VArrays* to hold the fixed and variable length data.

 Copyright © 2005, Doclinx, Inc.

A hash thread joins common nodes providing efficient lookup across common sub-trees and nodes. Each node or tag element is represented by a numerical range that encompasses the list of ordered node numbers describing the child nodes and itself. A leaf node has a single value.

Merging context trees involves walking each level-1 sub-tree in the update tree to find whether that tree is present in the main tree. If it is, then all locator attributes represented by the merged sub-tree must be updated to the values found in the matching primary tree. If the sub-tree does not match, then it is added as a level one sub-tree to the primary tree and again locator attribute values representing the context numbers are again mapped.

Searching for context entails looking up the context path in the context tree. This will yield a range or set of ranges based on the context attribute. The search then uses these attribute numbers to qualify matching words in the query.

The structure for a context node is as follows. Note that a flag field provides special information about a node (e.g. whether the node is a tag or an attribute).

```
struct CNODE
{
    INT32   cn_child;            // 1st child node #
    INT32   cn_parent;           // parent node #
    INT32   cn_next;             // sibling node #
    INT32   cn_hashThread; // pointer to name equivalent siblings in other sub-trees

    UINT16  cn_attrLo;           // Low value of child nodes
    UINT16  cn_attrHi;           // High value (the node # itself)
    INT32   cn_symbol;           // Index of node symbol in varray
    UINT16  cn_flags;            // Node flags
};
```

### 4.6 Thesaurus Structures

There are three thesaurus objects defined and used for enhancing word lookup alternatives: stem thesaurus, fuzzy thesaurus, and a classical thesaurus of related root meanings.

Each thesaurus is represented as an in-memory associate array with word keys and a comma-separated list of alternatives. The thesaurus object can automatically save and load thesauri information from a file. Merging of a thesaurus simply requires that all keys in the two lists are represented and that any duplicate alternatives are removed.

The classical thesaurus allows custom thesauri to be developed specific to an application or a database. It is well know that thesaurus should be context aware. It is very important to use a legal thesaurus for collections of legal documents as opposed to using a standard thesaurus or even a medical thesaurus. The wrong thesaurus can create bad or even dangerous matches in an inappropriate context. Having a general-purpose thesauri mechanism achieves the greatest database design flexibility.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

Stem thesauri are created during the dbuild operation. Unique words are passed to the catalog manager via a callback. The stemming algorithm is based upon "An algorithm for suffix stripping", M. F. Porter and originally published in "Program", 14 no. 3, pp. 130-137, July 1980. Each unique word is analyzed for removal of suffixes, plurals, prefixes, tense variations, etc. A "base" or root word is created for any of these situations. A thesaurus list is then created using this "stem" word as the key. During word search, the stem key is created from the query word and used to access all other words with the same common stem. The header file stem.h defines the stemming classes and methods.

The fuzzy thesaurus is a list of alternative words that are related by phonetic pronunciation based on the Double Metaphone (DM) algorithm. Lawrence Phillips developed the Double Metaphone algorithm. Like the Soundex algorithm, it compares words that sound alike but are spelled differently. DM was designed to overcome difficulties encountered with Soundex. This implementation was modified from a program written by Gary A. Parker and published in the June/July, 1991 (vol. 5 nr. 4) issue of C Gazette. As published, this code was explicitly placed in the public domain by the author. A thesaurus with the phonetic base as a key provides a list of words related to a given word. This enables correction of misspelled words in both the collection and in queries. The header file dmetaph.h defines the phonetic reduction class and methods.

### 4.7 Plurals

The pluralization method employs a public domain algorithm based on "An Algorithmic Approach to English Pluralization" by Damien Conway. Given a word, all plural and non-plural variants are returned as a comma separated list. The header file plural.h defines the pluralization class and methods.

### 4.8 Obsolete Search Handles

If read-only XS search handles are open during a catalog update operation, they become obsolete upon completion of the build or merge process. This happens as the operation completes. The update completion sequence will invalidate handles during the finalization of the build or merge. The handles then have to be closed and re-opened in order to access the new database information of the catalog. Any functions XS Search operations using an obsolete handle return an XS_OBSOLETE error.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

# 5. TeraXML Query Language

TeraXML provides a rich and powerful query language to perform context-based XML and full-text queries on searchable databases. The query language is an enhanced form of the FTQL (Full Text Query Language) standard developed in the late '80s.

XML contextual search queries are supported using the W3C XPath standard. XPath information is available at http://www.w3.org/TR/xpath.html. The TeraXML Query language provides a rich feature set to perform Boolean, fuzzy, proximity and range searches. Wildcards, stemming and different data types are also supported. XML context-based queries may be freely mixed with full-text queries to perform complex searches.

TeraXML provides very flexible data organization schemes with features such multiple dictionaries, data types (text, number, float, date and currency), partitioned indexes, fields and the ability to store state information with each and every word. The query language has several operators to leverage these advanced features, which enable the design of optimal data storage schemes allowing very-large knowledge sets to be indexed and searched efficiently.

TeraXML search query language syntax is described below. Examples are presented first to illustrate common queries followed by the precise grammar.

## 5.1 Query Examples

1. Given the XML markup:

```
<name MI = "J">Schmitt
     <first>Steve</first>
</name>"
```

   a. Find documents that contain "steve" in the first name:

```
steve IN XPATH "name/first"
```

   b. Find documents where the value of the attribute "MI" in <name> is equal to "J".

```
"J" in xpath "name/@mi"
```


Copyright © 2005, Doclinx, Inc.

2. Given the XML markup:

```
<employee-record>
      <identification>
            <name nickname = "buddy">
                  <first>joe</first>
                  <middle>bob</middle>
                  <last>thornton</last>
            </name>
            <ss number = 366546660>
            <age>47</age>
      </identification>
      <status>disabled</status>
      <salary>40000<salary>
</employee-record>
```

The following are some searches using XPath notation to describe the record:

a. Relative path examples:

   Find documents where <salary> is between 30000 and 50000:

   ```
   XPATH //salary BETWEEN 30000,50000
   ```

   Find documents where <ss> has an attribute "number" with a value of "366546660".

   ```
   XPATH //ss/@number = 366546660
   ```

   Find documents where last name starts with "thorn".

   ```
   thorn* in XPATH name/last
   ```

b. Absolute path examples

   Find documents where employee's age is less than 50.

   ```
   XPATH "/employee-record/identification/age" < 50
   ```

   Find documents where employee's status is "disabled".

   ```
   disabled in XPATH "employee-record/identification/status"
   ```

3. Search for BOB or RAY and COMEDY but do not include HOPE. (Parentheses may be used to explicitly specify operator precedence).

   ```
   (bob OR ray) AND comedy BUTNOT hope
   ```


Copyright © 2005, Doclinx, Inc.

4. Search for words with proximity operators:

```
(door AND garage) proximity 4 words
(door AND concrete AND heavy) proximity 2 paragraphs
```

5. Search for a number between 1.0 and 2.5 in field 4.

```
FIELD 4 BETWEEN 1.0, 1.5
```

6. Search for all dates after April 26, 1957 in fields 2 and 3.

```
FIELD 2,3 > 19570426
```

   Note: This query uses an integer (32-bit) collating sequence for representing dates (Y2K compliant).

7. Search for part number specification (e.g. catalog-model-partno) in documents 1 .. 10. Assume `CATALOG` is defined as "9", `MODEL` as "10" and `PARTNO` as "47".

```
FIELD CATALOG>47 AND FIELD MODEL BETWEEN 4,8 AND FIELD
PARTNO=1145 SET 1,10
```

8. Search for "Desert Fox" or "Erwin Rommel" in field Author.
   Assume `AUTHOR` is defined as `DRI 2 FIELD 8[10]`.

```
 "desert-fox" IN ALL OR "Erwin-Rommel" IN AUTHOR
```

9. Search for "Bob Hope" in any field except field 8.

```
 "bob hope" in field ~8
```

## 5.2 Grammar

1. Lower case words are meta-definitions (i.e. non-terminals defined in terms of terminals and other non-terminals).

2. UPPER case words are terminal symbol KEY or reserved words.

3. The "|" symbol implies "OR".

4. The "*" symbol means the previous object repeated 0 or more times.

 Copyright © 2005, Doclinx, Inc.

5. Literal punctuation is specified with double-quotes. (e.g. "&" implies the ampersand character).

6. Optional items are specified using []. (e.g. WORD[S] means WORD or WORDS).

7. The sequence """" denotes a single double-quote character.

8. A definition in comments /* ... */ is an English explanation or a regular expression definition.

```
query                 ::= set_query
                          | set_query SET range-list

set-query             ::= term
                          | set-query AND term
                          | set-query BUTNOT term

term                  ::= field-item
                          | "&" field-item
                          | term OR field-item
                          | "&" term OR field-item

field-item            ::= compare-condition
                          | between-condition
                          | proximate-condition

compare-condition     ::= field-spec comp-op word

comp-op               ::= "=" | "!=" | "<" | "<=" | ">" | ">="

between-condition     ::= field-spec BETWEEN word "," word
                          | field-spec OUTSIDE word "," word

proximate-condition   ::= phrase-term
                          | phrase-term PROXIMITY distance

distance              ::= constant group-unit

group-unit            ::= WORD[S]
                          | SENTENCE[S]
                          | PARAGRAPH[S]
                          | DOCUMENT[S]

phrase-term           ::= phrase-list
                          phrase-term phrase-term-op phrase-list

phrase-term-op        ::= "+" | "~"

phrase-list           ::= field-phrase
                          | phrase-list "," field-phrase

field-phrase          ::= phrase
                          | field-spec ":" phrase
```

 Copyright © 2005, Doclinx, Inc.

```
                        | phrase IN field-spec

phrase              ::= phrase-item
                        "(" set-query ")"

phrase-item         ::= approx-word
                        | phrase-item order-op approx-word

order-op            ::= " " | "-"

field-spec          ::= DICTIONARY constant field-list
                        | DRI constant field-list
                        | XPATH xpath-spec
                        | TAG xpath-spec
                        | field-list

xpath-spec          ::= A valid W3C XPath specification

field-list          ::= ALL
                        | FIELD[S] aalist-spec

aalist-spec         ::= aalist-item
                        aalist-spec "," aalist-item

aalist-item         ::= constant
                        | "~" constant
                        | constant[aaval-spec]
                        | "~" constant[aaval-spec]

aaval-spec          ::= aaval-spec-item
                        | aaval-spec "," aaval-spec-item

aaval-spec-item     ::= constant
                        | constant ".." constant

approx-word         ::= word
                        | "@" word

word                ::= real-word
                        | """exact-order-phrase"""

real-word           ::= numeric
                        | id
                        | id"*"
                        | id"?"*

range-list          ::= constant-list
                        | BETWEEN constant "," constant
                        | OUTSIDE constant "," constant

constant-list       ::= constant
                        | constant-list "," constant

exact-order-phrase  ::= real-word
                        | real-word " " exact-order-phrase

constant            ::= /* [0-9]+ */

id                  ::= /* [A-Z][A-Z0-9_]* , or if in quotes,
                           can be any non-blank character */
```


Copyright © 2005, Doclinx, Inc.

```
numeric              ::= /* constant or floating pt. # (e.g. 1.23) */
```

### 5.3 Notes

1. A <real-word> definition can vary from database to database. Right truncation style wildcards are supported. The representation may vary according to word definition, but will support the following level of functionality:

   Form 1: xxx* <- matches ALL words starting with "xxx"

   Form 2: xxx?? <- matches ALL words starting with "xxx" and are of length <= 5.

2. A <constant> is an integer (e.g. 47).

3. A quoted word is not interpreted, so anything inside will constitute a word except for a blank (' '). Blanks inside a quoted string will be construed as a separator denoting a string of words to find in exact order. Some implementations may have this feature turned off.

4. A NOT condition can not form a single term.

5. Sentences may or may not be implemented in a specific application.

6. The "&" symbol is an anchor. It forces a term to be evaluated first instead of the MIN-term order.

7. The "@" symbol means to find the word CLOSEST to the given word.

8. The "~" symbol means the NOT operator.

### 5.4 OPNODE Query Parse Tree Structure

A query is processed by the query module and converted into an OPNODE parse tree. The tree is then scanned in preorder by the boolean logic manager to evaluate the expression and compute the results of the search. The result of the search is a list of locators satisfying the query.

Each leaf node represents a search word while parent nodes denote the connectivity and relationships between the nodes. Each node has an operation type along with a relationship vector. The type specifies the Boolean operation to perform on the node (along with the evaluation stack, if not empty). The relationship vector is an order list of attributes and codes that specify what attributes in the word locator must be satisfied to qualify for a "match". One or two symbols are associated with each leaf node – a second symbol exists for opnode function types

 Copyright © 2005, Doclinx, Inc.

such as range searching (e.g. upper and lower bound). Results for each operation and word lookup counts are also stored in the node after evaluation.

A stack accumulator is used in conjunction with evaluation the Boolean tree. This allows for evaluation of nested expressions (i.e. parenthetical expressions). A 'load' operation puts a word or sub-expression on the stack. All other operations apply the node function the contents of the stack and the node operand.

Given a query expression: "(word1 + word2 + 'word3 word4') proximity 1 paragraph", the resulting parse tree would look like:

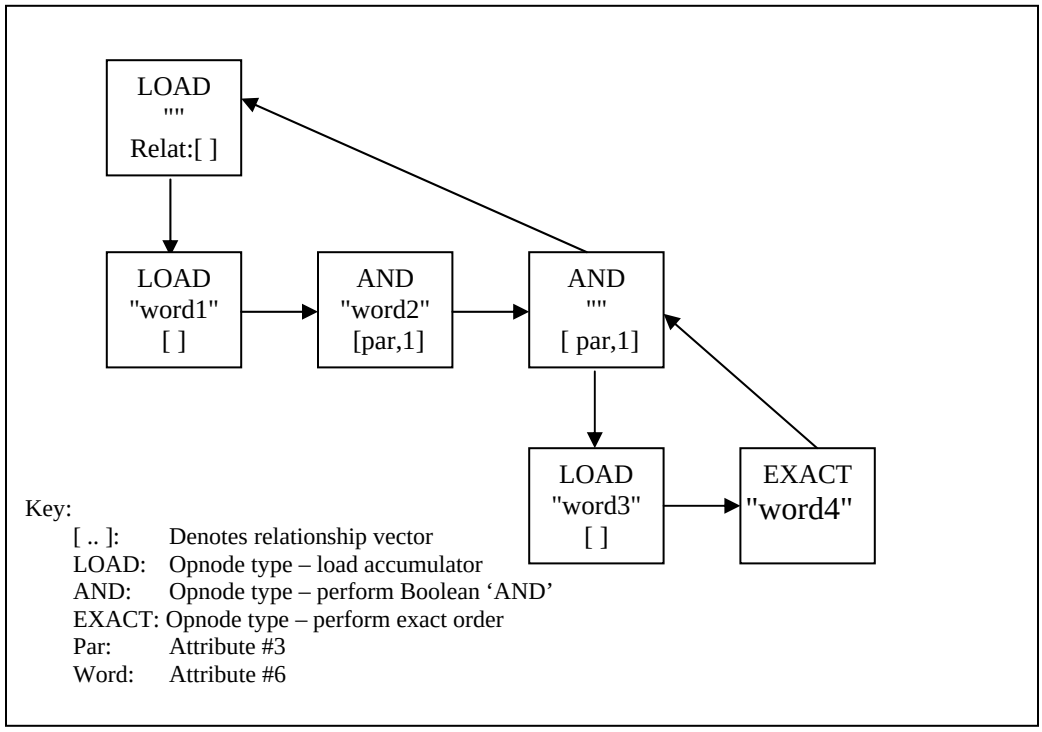


**Figure 4**: OpNode Tree

 Copyright © 2005, Doclinx, Inc.

# 6. TeraXML Linguistics API

TeraXML offers a Web (http) based API using XML messages to access the linguistics information extracted by TeraXML. The client sends a request in the form of a URL with parameters and the server responds with a XML data containing the results of the analysis requested by the client.

TeraXML Linguistics API can process a text file and return one of the following items:

1. Part-of-Speech Tags.
2. Base Noun Phrases.
3. Sentences.
4. Named Entities.

## 6.1 Processing a local file text file

The TextAnalyzer.jsp servlet allows the user to specify a local filename to be analyzed by TeraXML's Linguistic Subsystem.

### 6.1.1 Request Format

Assuming the TeraXML is running on hostname "localhost", port 8080, the following URL should be used:

http://localhost:8080/teraxml/TextAnalyzer.jsp?file=c:\input.txt&lang=enNE=y&SB=y&POS=y&NP=y

This URL specified the following input parameters:

1. file=filename Name of text file to be processed. This file must contain text only and must be an absolute filename. MS Windows note: Its default configuration, TeraXML runs as a service on MS Windows and services are not allowed to access network shares without setting special permissions allowing the server computer access to network resources.
2. lang=en Language code that specifies the language of the input file. English is specified by the code "en". For a complete list, see section 9.
3. NE=y Return Named Entities with their types.
4. SB=y Return sentences in the input text file.

 Copyright © 2005, Doclinx, Inc.

5. POS=y Return Part-of-Speech tags in the input text file.
6. NP=y Return Base Noun Phrases detected in the input text file.

For example, to process the file c:\file1.txt and return POS tags, Sentences, Noun Phrases and Named Entities submit the URL:

http://localhost:8080/teraxml/TextAnalyzer.jsp?file=c:\file1.txt&lang=en&POS=y&SB=y&NP=y&NE=y

The server will return XML data encoded in UTF-8 with the mime-type "text/xml". This XML data can be processed by the caller to iterate over the requested information (POS tag, Sentences, Base noun phrase or Entities).

### 6.1.2 Response Format – Output XML

The XML returned by the TeraXML server conforms to the following DTD:

```
<!DOCTYPE response [
        <!ELEMENT response (NE|NP|SB|POS|error)>
        <!ELEMENT NE   (entity+)>
        <!ELEMENT NP   (phrase+)>
        <!ELEMENT SB   (sentence+)>
        <!ELEMENT POS  (token+)>
        <!ELEMENT entity (#PCDATA)>
        <!ATTLIST entity type CDATA #REQUIRED>
        <!ELEMENT phrase (#PCDATA)>
        <!ELEMENT sentence (#PCDATA)>
        <!ELEMENT token (#PCDATA)>
        <!ATTLIST token type CDATA #REQUIRED>
        <!ELEMENT error (#PCDATA)>
]>
```

The following is an example of the XML file returned by the server, assuming that all items (Entities, Base noun phrases, Sentences and POS tags) were requested.

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
   <NE>
      <entity type="PERSON">George Bush</entity>
      <entity type="LOCATION">White House</entity>
      <entity type="ORGANIZATION">Republican Party</entity>
   </NE>
   <NP>
      <phrase>George Bush</phrase>
      <phrase>the White House</phrase>
      <phrase>the Republic Party</phrase>
   </NP>
   <SB>
      <sentence>He lives in the White House .</sentence>
      <sentence>He is a member of the Republican Party</sentence>
```

 Copyright © 2005, Doclinx, Inc.

```
   </SB>
   <POS>
      <token type="PRP">He</token>
      <token type="NNS">lives</token>
      <token type="IN">in</token>
      <token type="DT">the</token>
      <token type="NNP">White</token>
      <token type="NNP">House</token>   </POS>
</response>
```

### 6.2 Error Handling

In case an error occurs during processing of the specified input file, the following XML will be returned to the client:

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
   <error>
      Specified file c:\mydata\my-text-file.txt does not exist.
   </error>
</response>
```


Copyright © 2005, Doclinx, Inc.

# 7. TeraXML Search and Retrieval API

TeraXML offers a pure Java API to enable easy integration and allow TeraXML to be embedded into a java web or desktop application. Documentation for the Java API is available in the standard "javadoc" format and is provided separately from this document.


Copyright © 2005, Doclinx, Inc.

# 8. TeraXML XML over HTTP API

TeraXML includes two modules that expose an XML over HTTP API. Search functionality is exposed by the TeraXML SearchAgent. Indexing functionality is exposed by the TeraXML IndexAgent.

The XML over HTTP API uses URIs with specific name/value pairs to invoke search or indexing requests in the TeraXML server. The URI is the primary method used for message passing. Once the URI is processed, a well-formatted XML document is returned as a response.

XML over HTTP has several advantages over conventional APIs.

1. It is a widely accepted methodology enabling most developers to create applications quickly.

2. XML parsing tools are widely available in most development environments.

3. It offers cross platform compatibility and does not restrict the TeraXML server to specific platforms. For example, this API can be used in a Microsoft .NET environment to communicate with TeraXML server running on an IBM minicomputer.

4. It allows server side distributed processing and load balancing in manner completely transparent to the client process.

5. It enables process safety and independence. A client process that crashes will not crash the TeraXML server.

6. The http protocol works through most corporate firewalls.

7. Debugging and troubleshooting are simplified by logging all requests and responses to a text file.

All examples in this document assume that the TeraXML server is running on a machine havimg the hostname "myserver" on port 8080.

If the TeraXML server is running on the same machine as the client application making requests to the SearchAgent or the IndexAgent the hostname "localhost" or the IP address "127.0.0.1" may be used. Even in such situations, it is highly recommended that the machine's actual hostname be used. This is because "localhost" or "127.0.0.1" may fail if the Internet settings are configured to use an HTTP Proxy server and "localhost" hostname or "127.0.0.1" IP address is forwarded to the HTTP Proxy server.

Doclinx Confidential, Do Not Distribute without permission Copyright © 2005, Doclinx, Inc.

## 8.1 Using the SearchAgent API in XML Mode

In XML mode the SearchAgent accepts all input parameters in the form of a well-formed XML parameter block. This XML parameter block is passed to the server as follows:

```
http://myserver:8080/teraxml/SearchAgent.jsp?xml=XML-parameter-block
```

The actual format of the XML parameter block depends on the command being sent to the server. Available commands and their associated XML parameter blocks are as described below.

Please note that the data types referred to in the following tables are as follows:

| | | |
|---|---|---|
| [String] | = | A valid XML string value. |
| [Integer] | = | A valid integer value. |
| [Boolean] | = | A True or False value. True values may be specified by either "yes", "1", "true" or "t". False values may be specified as "no", "0", "false" or "f". |
| [Float] | = | A floating-point numeric value. |
| [Catalog] | = | A Catalog name as shown in the TeraXML Management Application, or a Catalog Identifier as shown in the response of the "Show" command. A Catalog Identifier is the name of that catalog's directory name in the TeraXML Server's "catalogs" directory. Typically this directory is located at "[TeraXML-install-root]/catalogs". |
| [Word] | = | A single word. |
| [Word List] | = | A comma separated list of words. |

### 8.1.1 Status Command

This command returns the current status of the TeraXML SearchAgent.

**Request**

```
<status showMemory="[Boolean]" doGC="[Boolean]"/>
```

Notes:
1. A *showMemory* value of true will show the memory free and total memory size of the JVM

 Copyright © 2005, Doclinx, Inc.

2. A doGC vale of true will perform Garbage Collection (GC) and will show memory before and after the GC.

**Response**

```
<response request="status"
                catalogHandles="[Integer]"
                searchHandles="[Integer]"
                b4GCmemoryFree="[Integer]"
                b4GCmemoryTotal="[Integer]"
                memoryFree="[Integer]"
                memoryTotal="[Integer]" (see section 8.1.10)>
        <catalogHandle name="[HandleName]">
                <catalog>[Catalog name]</catalog>
        </catalogHandle>
        <searchHandle name="[HandleName]">
                <catalog>[Catalog name]</catalog>
                <query>[Query]</query>
        </searchHandle>
</response>
```

### 8.1.2 Show Command

This command lists all available TeraXML catalogs and their respective status.

**Request**

```
<show catalogId="[Catalog]" verbose="[Boolean]" />
```

Notes:
1. A *CatalogId* value of "*" will show all available catalogs.
2. A *verbose* value of *true* will show all queries associated with this catalog.

**Response**

```
<response request="show" count="[Integer]" (see section 8.1.10)>
        <agentCatalog catalogId="[Catalog]"
                name="[String]"
                version="[Integer]"
                state="[String]"
                caching="[Boolean]"
                autoStart="[Boolean]"
                indexAltTitle="[Boolean]"
                documents="[Integer]" />
        <agentCatalog … />
        <activeQuery catalogId="[Catalog]"
                query="[String]"
                elapsedTime="hh:mm:ss" />
```

 Copyright © 2005, Doclinx, Inc.

```
        </agentcatalog>
        <agentCatalog …/>
</response>
```

Or,

<response request="show" msg="[String]" *(see section 8.1.10)* />

### 8.1.3 Start Command

This command marks a TeraXML catalog to be available for searching. Once a catalog is "started", it will process search commands. A TeraXML catalog must be "started" before any other catalog level SearchAgent operations on it can be performed.

**Request**

<start catalogId="[Catalog]" />

**Response**

```
<response request="start" count="[Integer]" (see section 8.1.10)>
        <agentCatalog catalogId="[Catalog]"
                name="[String]"
                version="[Integer]"
                state="started"
                caching="[Boolean]"
                autoStart="[Boolean]"
                indexAltTitle="[Boolean]"
                documents="[Integer]" />
</response>
```

Or,

<response request="start" msg="[String]" *(see section 8.1.10)*/>

Notes:
1. A *CatalogId* value of "*" will start all available catalogs.

### 8.1.4 Stop Command

This command sets the state of a TeraXML catalog such that it will not process any more search queries.

**Request**

<stop catalogId="[Catalog]" />

 Copyright © 2005, Doclinx, Inc.

**Response**

```
<response request="stop" count="[Integer]" (see section 8.1.10)>
        <agentCatalog catalogId="[Catalog]"
                name="[String]"
                version="[Integer]"
                state="stopped"
                caching="[Boolean]"
                autoStart="[Boolean]"
                indexAltTitle="[Boolean]"
                documents="[Integer]" />
 </response>
```

Or,

```
<response request="stop" msg="[String]" (see section 8.1.10)/>
```

Notes:
1. A *catalogId* value of "*" will stop all available catalogs.

### 8.1.5 Search Command

This command performs a search query and returns search results.

**Request**

```
<search catalogId="[Catalog]">
        <query page="[Integer]"
                pageSize="[Integer]"
                resultSort="[String]"
                relevancyWeighting="[String]"
                fuzzy="[Boolean]"
                plural="[Boolean]"
                stem="[Boolean]"
                test="[Boolean]">
        query text
        </query>
        <resultFields>[Comma-separated list of strings]</resultFields>
</search>
```

Notes:
1. A *pageSize* of "-1" will return all the results for the given query.
2. Predefined values for *resultFields* are:
   "file", "url", "title", "altTitle", "fileType", "accessTime", "creationTime", "writeTime", "docId", "hitCount", "abstract", "encoding", "relevancy", "composite", "compositeOffset", "compositeLength", "additionalText", "catalogDocumentId" and "excerpt:$n_1$:$n_2$".
3. For the "excerpt:$n_1$:$n_2$" field, "$n_1$" and "$n_2$" are numbers that specify the number of words before ($n_1$) and after ($n_2$) the search term that should be included in the excerpt.
4. User specified meta-data names are valid *resultFields* values.

 Copyright © 2005, Doclinx, Inc.

5. Predefined values for *resultSort* are "@fileName", "@fileDate", "@fileSize", "@fileType", "@hitCount", "@relevancy", and "@url". If no sort is specified then results are returned in natural order.
6. User specified meta-data names are valid *resultSort* values.
7. Predefined values for RelevancyWeighting are [TITLE]=#, [PROX]=#, and [EXACT]=#, where # is an integer in the range 0..16
8. XPATH values for RelevancyWeighting are also allowed.

**Response**

```
<response request="search"
        catalogId="[Catalog]"
        catalogName="[Catalog]"
        version="[Integer]"
        documentsInCatalog="[Integer]"
        documentsInResultSet="[Integer]"
        hitsInResultSet="[Integer]"
        searchTime="[Float]"
        resultsTime="[Float]"
        page="[Integer]"
        pageSize="[Integer]"
        resultSort="[String]"
        relevancyWeighting="[String]"
        maxRelevancy="[Integer]"
        resultFields="[String]"
        fuzzy="[Boolean]"
        plural="[Boolean]"
        stem="[Boolean]"
        count="[Integer]"
        (see section 8.1.10)>
        <documentInfo>
                ...list of values from fields specified by the resultFields tag...
        </documentInfo>
</response>
```

Or,

```
<response request="search" msg="[String]" (see section 8.1.10)/>
```

### 8.1.6 Linguistics Command

This command performs a linguistic analysis of the supplied search word and returns possible alternative search words.

**Request**

```
<linguistics catalogId="[Catalog]"
        word="[Word]"
        fuzzy="[Boolean]"
```

 Copyright © 2005, Doclinx, Inc.

```
        plural="[Boolean]"
        stem="[Boolean]" />
```

Notes:
1. If fuzzy is true then return fuzzy terms related to word.
2. If plural is true then return plural terms related to word.
3. If stem is true then return stem terms related to word.

**Response**

```
<response request="linguistics" (see section 8.1.10)>
        <fuzzyWords>[Word List]</fuzzyWords>
        <pluralWords>[Word List]</pluralWords>
        <stemWords>[Word List]</stemWords>
</response>
```

Or,

```
<response request="linguistics" msg="[String]" (see section 8.1.10)/>
```

### 8.1.7 Suggestions Command

This command returns alternative words to a specified input word from words found the index.

**Request**

```
<suggestions catalogId="[Catalog]" word="[Word]" fuzzy="[Boolean]" />
```

Notes:
1. If fuzzy is true then return fuzzy terms related to word.

**Response**

```
<response request="suggestions" (see section 8.1.10)>
        <word count="[Integer]">[Word]</words>
</response>
```

Or,

```
<response request="suggestions" msg="[String]" (see section 8.1.10)/>
```

### 8.1.8 Properties Command

This command returns alternative words to a specified input word from words found the index.

 Copyright © 2005, Doclinx, Inc.

**Request**

<properties catalogId="[Catalog]" autoStart="[Boolean]" caching="[Boolean]" />

**Response**

<response request="properties" count="[Integer]" *(see section 8.1.10)*>
    <agentCatalog catalogId="[Catalog]"
        name="[String]"
        version="[Integer]"
        state="[String]"
        caching="[Boolean]"
        autoStart="[Boolean]"
        indexAltTitle="[Boolean]"
        documents="[Integer]" />
</response>

Or,

<response request="properties" msg="[String]" *(see section 8.1.10)*/>

### 8.1.9 CatalogItem Command

This command performs a search query and returns search results.

**Request**

<catalogItem catalogId="[Catalog]"
    docId="[Integer]"
    pageSize="[Integer]"
    resultFields="[Comma-separated list of strings]"
    test="[Boolean]" />

Notes:
1. Predefined values for *resultFields* are:
   "file", "url", "title", "altTitle", "fileType", "accessTime", "creationTime", "writeTime", "docId", "hitCount", "abstract", "encoding", "relevancy", "composite", "compositeOffset", "compositeLength", "additionalText", "catalogDocumentId".

**Response**

<response request="catalogItem"
    catalogId="[Catalog]"
    catalogName="[Catalog]"
    version="[Integer]"
    docId="[Integer]"
    pageSize="[Integer]"
    resultFields="[String]"

 Copyright © 2005, Doclinx, Inc.

```
        resultsTime="[Float]"
        documentsInCatalog="[Integer]"
        test="[Boolean]"
        count="[Integer]"
        (see section 8.1.10)>
        <documentInfo>
                ...list of values from fields specified by resultFields...
        </documentInfo>
</response>
```

Or,

```
<response request="catalogItem" msg="[String]" (see section 8.1.10)/>
```

### 8.1.10 Response attributes common to all commands

The following attributes are present in the response XML for all commands.

**Response**

```
<response request="…"
        requests="[Integer]"
        concurrent="[Integer]"
        maxConcurrent="[Integer]"
        concurrentSearches="[Integer]"
        maxConcurrentSearches="[Integer]"
</response>
```

Notes:

1. *requests* is the count of total requests processed by the TeraXML server
2. *concurrent* is the count of concurrent requests being processed at this instance in time by the TeraXML server.
3. *maxConcurrent* is the largest concurrent value since the TeraXML server was started.
4. *concurentSearches* is the count of concurrent search requests being processed at this instance in time by the TeraXML server.
5. *maxConcurrentSearches* is the largest *concurrentSearchs* value since the TeraXML server was started.

 Copyright © 2005, Doclinx, Inc.

### 8.1.11 Using the SearchAgent API in Multiple Parameter mode

This mode allows you to specify all input parameters in the URL itself. The URL should be formatted as follows:

```
http://myhost:8080/teraxml/SearchAgent.jsp?cmd=c=&q=&rf=&rs=&p=&ps=&v=
```

The meaning of each parameter is described below. For precise details, see section 2 where XML the parameter block for each different command is described in detail.

| Parameter | Meaning |
|---|---|
| cmd | Command - search, show, start, status, stop, linguistics, suggestions, properties or catalogItem. |
| C | Catalog name or catalog id |
| Fz | Fuzzy (Boolean) |
| Pl | Plural (Boolean) |
| St | Stem (Boolean) |
| Di | Document Id (1 .. # of documents in Catalog) |
| Q | Query string |
| Rf | Result fields required for each search result item |
| Rs | Result sort key(s) |
| P | Page number requested |
| Ps | Page size requested |
| wd | Word |
| As | Auto Start (Boolean) |
| Ca | Caching (Boolean) |
| V | Verbose (Boolean) |

The data returned is the same XML format as described in section 2.


Copyright © 2005, Doclinx, Inc.

## 8.2 IndexAgent API

The IndexAgent exposes an XML over HTTP API. This API allows the client to submit Indexing Tasks that are executed in the TeraXML Server process. Certain IndexAgent API calls are asynchronous – which means that a call starts an Indexing Task in a server thread and returns immediately to the client an Event ID. The client then periodically checks the status of that particular task using the returned event ID.

The IndexAgent accepts all input parameters in the form of an XML parameter block. This XML parameter block is passed to the server as follows:

```
http://myserver:8080/teraxml/IndexingAgent.jsp?xml=XML-parameter-block
```

The actual format of the XML parameter block depends on the command being sent to the server. Available commands and their associated XML parameter blocks are as described below.

### 8.2.1 Copy Catalog Command

This command allows the client to create a new catalog by copying an existing catalog.

**Request**

```
<copyCatalog fromCatalogId="[Catalog]"
        name="[String]"
        caching="[Boolean]"
        autoStart="[Boolean]"
        <description>
                [Text]
        </description>
</copyCatalog>
```

Note:
1. If the new name is an existing catalog, then, a new version will be created.

**Response**

```
<response request="copyCatalog"
        fromCatalogId="[Catalog]"
        cataloged="[Catalog]"
        catalogName="[String]"
        caching="[Boolean]"
        autoStart="[Boolean]"
        eventId="[String]"
        (see section 8.1.10)>
```


Copyright © 2005, Doclinx, Inc.

```
        <description>[Text]</description>
</response>
```

Or

```
<response request="copyCatalog" (see section 8.1.10)>
        <errmsg>[String]</errmsg>
</response>
```

### 8.2.2 Create Catalog Command

This command allows the client to create a new catalog.

**Request**

```
<createCatalog name="[String]"
        XSLfileName="[String]"
        initialSize="[Integer]"
        fuzzy="[Boolean]"
        stem="[Boolean]"
        caching="[Boolean]"
        autoStart="[Boolean]"
        indexAltTitle="[Boolean]"
        indexModDate="[Boolean]"
        indexURL="[Boolean]"
        keepXMLfromPDF="[Boolean]"
        autoReplicate="[Boolean]"
        keepVersions="[Integer]">
        <description>
                [Text]
        </description>
</createCatalog>
```

Notes:
1. If initialSize is 0 then TeraXML will choose an optimal initial size based on the first set of documents added to the catalog.

**Response**

```
<response request="createCatalog"
        catalogId="[Catalog]"
        catalogName="[String]"
        version="[Integer]"
        initialSize="[Integer]"
        (see section 8.1.10)>
        <description>
                [Text]
        </description>
</response>
```

 Copyright © 2005, Doclinx, Inc.

or

```
<response request="createCatalog" (see section 8.1.10)>
        <errmsg>[String]</errmsg>
</response>
```

### 8.2.3 Delete Catalog Command

This command allows the client to delete a catalog.

**Request**

```
<deleteCatalog catalogId="[Catalog]" />
```

**Response**

```
<response request="deleteCatalog" count="[Integer]" catalogId="[Catalog]" (see section 8.1.10)>
```

or

```
<response request="deleteCatalog" (see section 8.1.10)>
        <errmsg>[String]</errmsg>
</response>
```

### 8.2.4 Database Command

This command allows the client to submit an indexing task which uses a JDBC connection to retrieve data from a database for indexing.

This is an asynchronous call. The "eventId" attribute in the response specifies a TeraXML Event that may be queried using the Status command.

For detailed explanation of each of the input parameters, please see the document "TeraXML Database Indexer Module".

**Request**

```
<database catalogId="[Catalog]">
        <inputset>
                <database>[String]</database>
                <userId>[String]</userId>
                <password>[String]</password>
                <indexStatement>[String]</indexStatement>
                <displayStatement>[String]</displayStatement>
                <driver>[String]</driver>
```

 Copyright © 2005, Doclinx, Inc.

```
            <primaryKeyColumns>[String]</primaryKeyColumns>
            <titleColumns>[String]</titleColumns>
            <metaDataColumns>[String]</metaDataColumns>
            <indexColumns>[String]</indexColumns>
        </inputset>
</database>
```

**Response**

```
<response request="database"
        catalogId="[Catalog]"
        catalogName="[String]"
        eventId="[String]"
        (see section 8.1.10)/>
```

Or,

```
<response request="database" (see section 8.1.10)>
        <errmsg>[String]</errmsg>
</response>
```

### 8.2.5 Spider Command

This command allows the client to submit an indexing task which spiders (crawls) a base URL, downloads found URLs locally and subsequently indexes the downloaded files.

This is an asynchronous call. The "eventId" attribute in the response specifies a TeraXML Event that may be queried using the Status command.

For detailed explanation of each of the input parameters, please see the document "*TeraXML Webcrawler Options*".

**Request**

```
<spider catalogId="[Catalog]">
        <inputset>
        <url>[String] </url>
        <mapFile> [String]</mapFile>
        <outputPath>[String] </outputPath>
        <defaultEncoding>[String] </defaultEncoding>
        <maxDepth>[Integer] </maxDepth>
        <indexAfterCrawl>[Boolean] </indexAfterCrawl>
        <recursive>[Boolean] </recursive>
        <timeStamping>[Boolean] </timeStamping>
        <mirror>[Boolean] </mirror>
        <pageRequisites>[Boolean] </pageRequisites>
        <honorRobots>[Boolean] </honorRobots>
        <spanHosts>[Boolean] </spanHosts>
```

 Copyright © 2005, Doclinx, Inc.

```
        <parentAscending>[Boolean] </parentAscending>
        <metaData name="[String]">[String] </metaData>
        <tries>[Integer] </tries>
        <waitRetrySeconds>[Integer] </waitRetrySeconds>
        <waitSeconds>[Integer] </waitSeconds>
        <timeoutSeconds>[Integer] </timeoutSeconds>
        <randomWait>[Boolean] </randomWait>
        <continue>[Boolean] </continue>
        <userId>[String] </userId>
        <password>[String] </password>
        <userAgent>[String] </userAgent>
        <setHTMLextension>[Boolean] </setHTMLextension>
        <acceptList>[String] </acceptList>
        <rejectList>[String] </rejectList>
        <domainList>[String] </domainList>
        <excludeDomainList>[String] </excludeDomainList>
        <followTags>[String] </followTags>
        <ignoreTags>[String] </ignoreTags>
        <includeDirectories>[String] </includeDirectories>
        <excludeDirectories>[String] </excludeDirectories>
        <XMLcompositXPaths>[String] </XMLcompositXPaths>
        <XMLtitleXPaths>[String] </XMLtitleXPaths>
        <XMLparagraphXPaths>[String] </XMLparagraphXPaths>
        <XMLintegerXPaths>[String] </XMLintegerXPaths>
        <XMLdateXPaths>[String] </XMLdateXPaths>
        <includeCDATA>[Boolean] </includeCDATA>
        <includeAttributes>[Boolean] </includeAttributes>
        </inputset>
</spider>
```

Notes:

1. When specifying metadata, you can specify a type by prefixing the metadata name with "i_" for integer and "d_" for date. This will ensure that sorting by that metadata works correctly.

**Response**

```
<response request="spider"
        catalogId="[Catalog]"
        catalogName="[String]"
        eventId="[String]"
        (see section 8.1.10)/>
```

Or

```
<response request="spider" (see section 8.1.10)>
        <errmsg>[String]</errmsg>
</response>
```

 Copyright © 2005, Doclinx, Inc.

### 8.2.6 Delete Command

This command allows the client to delete specific Catalog Entries using a search query.

This is an asynchronous call. The "eventId" attribute in the response specifies a TeraXML Event that may be queried using the Status command.

**Request**

```
<delete catalogId="[Catalog]">
        <query>[String]</query>
</delete>
```

**Response**

```
<response request="delete"
        catalogId="[Catalog]"
        catalogName="[String]"
        eventId="[String]" (see section 8.1.10)/>
```

Or

```
<response request="delete" (see section 8.1.10)>
        <errmsg></errmsg>
</response>
```

### 8.2.7 Merge Command

This command allows the client to start a Merge task. A Merge task should be performed on a catalog periodically after several updates have been performed on that catalog. The Merge task is responsible for merges the Primary and Update search Indexes into a single Primary Index and physically deleting any catalog entries deleted by the Delete task.

This is an asynchronous call. The "eventId" attribute in the response specifies a TeraXML Event that may be queried using the Status command.

For further information about Merging Catalogs please see the documents "TeraXML Enterprise Search" and "TeraXML Programmer's Guide".

**Request**

```
<merge catalogId="[Catalog]" />
```


Copyright © 2005, Doclinx, Inc.

**Response**

```
<response request="merge"
        catalogId="[Catalog]"
        catalogName="[String]"
        eventId="[String]"
        (see section 8.1.10)/>
```

Or

```
<response request="merge" (see section 8.1.10)>
        <errmsg>[String]</errmsg>
</response>
```

### 8.2.8 Show Command

This command allows the client to show existing catalog information.

**Request**

```
<show catalogId="[String]"/>
```

**Response**

```
<response request="show" count="[Integer]" (see section 8.1.10)>
        <catalog catalogId="[String]"
                catalogName="[String]"
                version="[Integer]"
                documents="[Integer]"/>
</response>
```

Or

```
<response request="show" (see section 8.1.10)>
        <errmsg>[String]</errmsg>
</response>
```

### 8.2.9 Status Command

This command allows the client to query the IndexAgent for a status of an Event. The eventId attribute value must be a value that was returned from a Spider, Delete or Merge command.

A TeraXML IndexAgent Event consists of an InputSetList, which consists of one or more InputSets. InputSets identify a Spider task.

 Copyright © 2005, Doclinx, Inc.

**Request**

```
<status eventId="[String]"/>
```

**Response**

```
<response request="status" eventId="[String]" (see section 8.1.10)>
        <runMode>[String]</runMode>
        <status>[String]</status>
        <extendedStatus>[String]</extendedStatus>
        <inputSetList id="[String]">
                <inputSet id="[String]">
                        <url>[String]</url>
                        <status>[String]</status>
                        <extendedStatus>[String]</extendedStatus>
                </inputSet>
        </inputSetList>
</response>
```

Or

```
<response request="status" (see section 8.1.10)>
        <errmsg>[String]</errmsg>
</response>
```

### 8.2.10 Version Command

This command returns the current version of the IndexAgent along with number of total and concurrent requests.

**Request**

```
<version />
```

**Response**

```
<response request="version" version="[String]" (see section 8.1.10)/>
```

### 8.2.11 MapFile Command

This command allows the client to submit a previously downloaded website for indexing using a Map File. Map Files are created by TeraXML's Webcrawler during the web crawling process. Websites may be downloaded but not indexed by setting the <indexAfterCrawl> option in the <spider> command to false.


Copyright © 2005, Doclinx, Inc.

**Request**

```
<mapFile catalogId="[Catalog]">
        <inputset>
                <mapFile>[String]</mapFile>
                <metaData name="[String]">[String]</metaData>
        </inputset>
</mapFile>
```

Notes:
1. When specifying metadata, you can specify a type by prefixing the metadata name with "i_" for integer and "d_" for date. This will ensure that sorting by that metadata works correctly.

**Response**

```
<response request="mapFile"
        catalogId="[Catalog]"
        catalogName="[String]"
        eventId="[String]"
        (see section 8.1.10)/>
```

Or

```
<response request="mapFile" (see section 8.1.10)>
        <errmsg>[String]</errmsg>
</response>
```

## 8.3 Testing the SearchAgent and IndexAgent

The SearchAgent can be tested by using the following URL:

```
http://myserver:8080/teraxml/SearchAgent.htm
```

The IndexAgent can be tested by using the following URL:

```
http://myserver:8080/teraxml/IndexAgent.htm
```


Copyright © 2005, Doclinx, Inc.

# 9. TeraXML C API

### 9.1.1 DLL_API dpBuild (CHAR *path, DPAPI_PARMS *parms);

```
Description:

The dpBuild() functions provides all the steps for taking STF files and building an
inverted index search structure. The dpBuild module also recognizes and processes
field description information so information can be searched by type. The output is a
".dat" file that contains all the information required for do full-text and fielded
searches. There are several steps in the build process. An error code and the step
indicate where the problem occurred.

Parameters:

    CHAR *path          - Full path expression pointing to input STF file.

    DPAPI_PARMS *parms  - Pointer to structure containing input, output and
                          error information for the build process.

    Component description (parms):

        All steps:

        BOOLEAN dpapi_intermediateFiles;

        If set, intermediate processing files are NOT erased. If FALSE,
        these files are automatically erased at the end of the last step.
        If an error is detected, the files are also preserved.

       CHAR *dpapi_templateFile;

        Pointer to the template for the .SYS DP control file. This file
        describes some additional parameters controlling the build process.
        Most of the parameter settings are determined by calculations
        during processing. These parameters should probably not require
        modification for XML search. This file is located in the
        catroot\ directory and is copied to the target catalog during
        the build process.

        Dbuild Step:

        UINT32  dbuild_memSize;

        Memory allocation size for dictionary build step. This should
        use as close to the physical machine size as possible. The DEFAULT
        is 1/2 of physical machine size. On current machines, this is now
        typically sufficient to limit the number of passes over the source to
        a single pass. (e.g 256MB+). A value of 0 indicates to use the
        default memory size calculation.
```

 Copyright © 2005, Doclinx, Inc.

```
UINT16  dbuild_contextAaidx;

The attribute array index to use for the context search information.
The range is 8..15. If 0, then no context search data structures
are built.

UINT16  *dbuild_contextMap;

An array of integers usually built by the catalog context routines.
This map established the correct hierarchical values for
performing context search. This parameter should be NULL
for non-context builds. Additionally, since the catalog
routines build this parameter, it should be NULL in that context
as well.

Invert Step:

UINT32 invert_memSize;

Memory allocation size for the inversion build step. This should
use as close to the physical machine size as possible. The DEFAULT
is 1/2 of physical machine size. A value of 0 indicates to use the
default memory size calculation.

INT32   invert_pct;

Number between 1 and 99 indicating the percentage of memory to use
for the dictionary during inversion. If 0, the default of 33%
is used. This can be increased in cases where an unusually large
number of unique words occur.

UINT16  invert_contextAaidx;
UINT16  *invert_contextMap;

See above information on dbuild analogs. Note that the dbuild and
invert values should be the same.

Reorg Step:

UINT32  reorg_memSize;

Memory allocation size for re-organization build step. If 0, the
DEFAULT value of 640K is used.

MkCDWeb (Make Web/CD Image) Step:

CHAR    *mk_outputPath;

Output PATH to place temporary and result .DAT file.
The .DAT file contains the complete set of data structures
built for the search routines (except for the separate, optional
context data structures).

UINT32  mk_memSize;

MkCDWeb memory allocation size. If 0, a DEFAULT value of 1/2
physical memory size is used.

CHAR    *mk_fileSelect;
```

 Copyright © 2005, Doclinx, Inc.

```
        A string denoting the sub-file components that comprise
        the .DAT file that is created. This value should always
        be set to NULL by applications.

        INT32   mk_sequence;

        DAT file sequence (multi-volume) -- Ignored (set to 0).

        CHAR    *mk_databaseId;

        Database Id. If set to NULL, uses time stamp. Default is suggested
        value.

        CHAR    *mk_key;

        Encryption key string to use for encoding database contents. If NULL,
        no encryption is used.

        CHAR    *mk_copyright;

        Name of the copyright file containing the vendor copyright
        information. This file defaults to the catroot/ file copyrght.dat
        if this parameter is NULL.

        CHAR    *mk_cdMapString;

        Size of 1st 3 .DAT sub-files. Do NOT set unless these files
        sizes are changed. Use value of NULL.

        INT32   mk_bWeb;

        Missing sub-files are NOT error. This MUST be set to a non-zero
        value.

        Return Data:

        UINT8   dpapi_phase;

        Step that failed if an error occurred. The values returned are
        defined in dpapi.h under the "Dataprep Build step indicators"
        heading. If no error occurs, then this is 0.

        int     dpapi_error;

        Returns one of the "DataPrep Return codes" in dpapi.h (if < 0).
        If positive, then the error is a build-step specific problem.


Input/Output Files:

    Input:

    STF input file  -- Parameter "path"
    copyrght.dat    -- Specified in component mk_copyright of parameter
                       "parms".
    template.sys    -- Default build control configuration file.

    Intermediate Files:

    "*.db2", "*.dbo", "*.ddp", "*.ddq", "*.ddr", "*.dlp", "*.ino"
```

 Copyright © 2005, Doclinx, Inc.

```
    Where "*" is the base name of the output file.

    Output files:

    "*.dat"         -- Output of build process. Path and name
                       is the base name defined by mk_outputPath
                       with the ".DAT" extension (unless the output
                       name uses and extension).
    template.sys    -- Updated .sys file copied from source defined
                       in parms component to output directory specified
                       by mk_outputPath.


Returns:
         INT     0  -- Successful merge (DPAPI_SUCCESS)
                -1  -- ERROR (DPAPI_ERROR)
```

### 9.1.2 DLL_API dpHtml2Stf (CHAR *path, SRC2STF_PARMS *parms);

```
Description:

The dpHtml2Stf() function provides access to the built-in HTML/XML
parser to convert HTML or XML files to STF. Parsing only looks for
words and limited structure items for HTML. For XML, tag/attribute
information is handled by a separate context process found with the
catalog functions.

All parsing parameters are defined. Note that the same structure is used
by the generic parser. The generic parser ignore all the HTML/XML
specific items (the ones starting with "ht_").


Parameters:

    CHAR *path          - Full path expression pointing to input HTML file.

    DPAPI_PARMS *parms  - Pointer to structure containing input, output and
                          error information for the conversion process.

    Component description (parms):

        Input parameters:

        CHAR   *sr_outputFile;              \

        Output path for the resulting .STF file. Must not be NULL.

        CHAR   *sr_logFile;

        Log path file. If NULL, log information to stdout. If "", then NO
        logging information is reported.

        CHAR   *sr_context;

        Context idx file for XML parsing. This is set/managed by internal
        catalog management routines. Users do not set this parameter.
```

 Copyright © 2005, Doclinx, Inc.

```
(i.e. set to NULL).

CHAR   *sr_contextStr;

Context string file for XML parsing. This is set/managed by internal
catalog management routines. Users do not set this parameter.
(i.e. set to NULL).

UINT8   sr_flags;                  // SR_FLAGS (filter options)

Eight flags for controlling filter/parser behavior. Currently, the
following options (see dpapi.h) are defined:

SR_INCLUDEDOCTYPE   -- Include document type in document under DRI 2.
                       Document type values are defined in the
                       sccfi.h file. HTML and XML file types (using
                       the HTML/XML filter) are SR_HTML and SR_XML.

UINT8   sr_contextAaidx;           // Context aaidx

The aaidx value to use for context information. 0 implies no context
processing. This is for the generic filter only. Either this or
the following parameter can be non-zero (i.e. not both).

UINT8   sr_genericAaidx;           // Generic Filter tag aaidx

The aaidx value to use for generic filter tag information. If non-zero,
then the context aaidx should be zero. This parameter is ignored for
HTML/XML parsing.

Values for the tags can be found in the sccca.h file.

BOOLEAN sr_appendToOutput;

If set, then append the output to file specified in sr_outputPath,
create the file if it does not exist. If not set, then output file is
created/overwritten.

BOOLEAN sr_includePunctuation;

Place punctuation characters in output STF. This should usually
be FALSE.

BOOLEAN sr_debug;

Turn on debug info output

BOOLEAN sr_enableJapanese;

Enable recognition of Japanese words. This is off by default.

INT32   sr_maxWordChars;

Maximum length of a word. The largest value permissible is 128.
If set to 0, then the value defaults to 64.

CHAR   *sr_regExpression;

If not NULL, then a string in EGREP format is used as a regular
expression to determine word-break rules. If NULL, then the standard
```

 Copyright © 2005, Doclinx, Inc.

```
word break of letters followed by letters/digits is used.

 INT   sr_foldSettings;

Control bits for case folding.

FOLD_DIGIT      0x4000  - Fold all numeric characters to base set
FOLD_LATIN      0x8000  - Fold all Latin equivalents to A..Z


Components for HTML/XML Conversion ONLY

UINT8   ht_contextAaidx;

Context aaidx value to used for HTML/XML parsing.

BOOLEAN ht_continueOnFileError;

If set, continue parsing even if an error is detected.

BOOLEAN ht_includeAttributes;

If set, include attribute data in token output. This must be set for
XML filtering.

BOOLEAN ht_warnUnknown;

Warn if an unknown HTML tag is found. Set to FALSE for XML
parsing.

BOOLEAN ht_inputIsListOfFiles;

Allows input file path to specify a list of files. This value must
be FALSE when used in the catalog routines.

UINT32  ht_docIdStart;

Id value used to uniquely identify document -- NOT USED.

CHAR   *ht_defFile;

File that defines tag used for HTML/XML parsing and whether these
tags define a new paragraph.

CHAR   *ht_titleBuffer;
INT     ht_titleBufferSize;

Buffer for title. Set by calling application. Can be NULL.
If not NULL, then the size of the buffer must be set in
ht_titleBufferSize.

UINT32  ht_documentsProcessed;

Number of documents processed. For the catalog routines, this must
always be 1.

Return Data:

int     dpapi_error;
```

 Copyright © 2005, Doclinx, Inc.

```
        Returns one of the "DataPrep Return codes" defined in dpapi.h.
        This value is less than DPAPI_ERROR.

Input/Output Files:

    Input:

    HTML/XML input file  -- Parameter "path"

    Output file:

    STF token file       -- Output of conversion process. File path name
                            specified by sr_outputPath.

Returns:
         INT     0  -- Successful conversion (DPAPI_SUCCESS)
                -1  -- ERROR (DPAPI_ERROR)
```

### 9.1.3 DLL_API dpMerge (CHAR *path, MERGE_PARMS *parms);

```
Description:

Merges two .DAT files into a single .DAT file. It is more
efficient to search a single database as the files grow larger.


Parameters:

    CHAR *path          - Full path expression pointing to output HTML .DAT
                          file. This file will be overwritten. The file
                          can be the same name as one of the input files.

    DPAPI_PARMS *parms  - Pointer to structure containing input, output and
                          error information for the conversion process.

    Component description (parms):

        Merge Step:

        CHAR    *mg_prm;

        File path name of primary input .DAT file. This should be the larger
        or "base" database file. This parameter must not be NULL.

        CHAR    *mg_upd;

        File path name of secondary input .DAT file. This should be the
        smaller or "update" database file. This parameter can be NULL
        for the case where documents are being deleted from primary (i.e.
        no documents are being added).

        CHAR    *mg_logFile;

        Log path file. If NULL, log information to stdout. If "", then NO
        logging information is reported.
```

 Copyright © 2005, Doclinx, Inc.

```
CHAR    *mg_sysFile;

Pointer to the .SYS DP control file. This file describes
some additional parameters controlling the build/merge process.
Most of the parameter settings are determined by calculations
during processing. These parameters should probably not require
modification for XML search. This file is located in the catroot
directory and is copied to the target catalog during the build
process.

BOOLEAN mg_debug;

If set, turn on debug info output.

UINT32  mg_mem;

Memory allocation size for the merge step. This should use
as close to the physical machine size as possible. The DEFAULT
is 1/2 of physical machine size.

INT     mg_mode;

Merge control settings. These merge settings can control the
efficiency of merge and adjust/remove data for deletions.
These are bit options.

MINCALC_LOCS: If bit set, then locator size is recalculated
              to minimal size in case of deletions.

ADJUST_LOCS:  When documents are deleted, document #'s for
              subsequent documents are adjusted to account
              for the removed items.

BIGINT  *mg_delList;

List of deleted documents. These document words/locators are
removed from the .DAT file. Subsequent documents are adjusted
if the ADJUST_LOCS bit is set.

UINT16  *mg_locMap;

Pointer to mapping array that maps context indexes to new values
calculated from merging the context tree files.

UINT16  mg_contextAaidx;

Specifies the context aaidx for the above context mapping
functionality.

All steps:

BOOLEAN dpapi_intermediateFiles;

If set, remove intermediate files after merge completes.

Reorg Step:

UINT32  reorg_memSize;

Same as definition above for dpBuild.
```

 Copyright © 2005, Doclinx, Inc.

```
        MkCDWeb Step:

        CHAR   *mk_outputPath;
        UINT32 mk_memSize;
        CHAR   *mk_fileSelect;
        INT32  mk_sequence;
        CHAR   *mk_databaseId;
        CHAR   *mk_key;
        CHAR   *mk_copyright;
        CHAR   *mk_cdMapString;
        INT32  mk_bWeb;

        Same as definitions above for dpBuild.

        Return Data:

        BIGINT dpapi_docCount;

        Number of documents in collection. (should be 1 for catalog routines)

        UINT8  dpapi_phase;

        Step where merge failed if error occurs. The values returned are
        defined in dpapi.h under the "Dataprep Build step indicators"
        heading. If no error occurs, then this is 0.

        int     dpapi_error;

        Returns one of the "DataPrep Return codes" in dpapi.h (if < 0).
        If positive, then the error is a build-step specific problem.


Input/Output Files:

    Input:
        mg_prm (parm):    Input file path of primary   .DAT file.
        mg_upd (parm):    Input file path of secondary .DAT file.
                          May be NULL when only deletions are required.

    Output:

        Parameter "path": Output path of result merged .DAT file.


Returns:
         INT     0  -- Successful merge (DPAPI_SUCCESS)
                -1  -- ERROR (DPAPI_ERROR)
```

### 9.1.4 DLL_API catCreate(CHAR *root, CHAR *path, CHAR *name, FIELDS *extra, UINT32 hashSize, CAT_HANDLE *handle);

```
Description:

Catalogs are created in a catalog "area". The area is a directory that can
```

 Copyright © 2005, Doclinx, Inc.

```
be populated by one or catalogs. Each catalog is a self-contained sub-
directory containing all information to build, maintain, and search a
a database of text files. The catCreate() function creates a catalog
under the named catalog area. Once created, the name is reserved and
can only be opened unless the entire sub-directory is removed.

If the creation process is successful, then a handle is return to be used for
all subsequent catalog function calls. A value less than 0, indicates
an error. Some or all of the sub-directory structures may be created.
It is advisable to delete the entire sub-directory in an error occurs.

Parameters:

    CHAR *root:      This is the full path name of the directory containing
                     the build files required to build and search a
                     database. This parameter must be specified and can not
                     be NULL.

    CHAR *path:      The full path name of the catalog area. The catalog
                     will be created in a subdirectory under this "area"
                     with the name specified in the parameter list.
                     This parameter can NOT be NULL.

    CHAR *name:      Name of the catalog. Must be a valid directory
                     name. The subdirectory must not already exist.
                     This parameter can NOT be NULL.

    FIELDS *extra:   A pointer to an integer array specifying additional
                     fields in the catalog entry item. This allows users
                     an extension mechanism for adding additional data
                     in the future. Usually set to NULL for standard
                     configuration.

    UINT32 hashSize: Initial size of the hash table for fast lookup.
                     May be set to 0. If 0, then the hash table is not
                     created until after catAddFile has been used to
                     add files or other routines requiring access
                     to the hash table are called. The user can also
                     call the catMakeHashLookup() routine at anytime
                     in order to build or re-build the hash table.

    CAT_HANDLE *handle:
                     Pointer to location to save handle value. The
                     application does not have access to the contents of
                     the handle. The handle must be closed by the
                     application. When the dataprep DLL exists, any open
                     handles will automatically be closed. This
                     parameter must not be NULL.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog created successfully.
```

 Copyright © 2005, Doclinx, Inc.

### 9.1.5 DLL_API catOpen(CHAR *path, CHAR *name, INT mode, CAT_HANDLE *handle);

```
Description:

Opens an already existing catalog for further processing or searching.
The catalog must exist and been created and properly closed. The handle
allows access to existing data with subsequent catalog functions.
When done using the handle, the catClose() function should be called to
conclude usage.

Parameters:

    CHAR *path:     The full path name of the catalog area.
                    This parameter can NOT be NULL.

    CHAR *name:     Name of the catalog. The catalog must already
                    exist. This parameter can NOT be NULL.

    INT mode:       CAT_READONLY:   Read-only access to catalog
                                    Several handles can be open in shared
                                    mode.
                    CAT_OPENUPDATE:
                                    Catalog opened for update. Data
                                    structures can be changed.

    CAT_HANDLE *handle:
                    Pointer to location to save handle value. The
                    application does not have access to the contents of
                    the handle. The handle must be closed by the
                    application. When the dataprep DLL exists, any open
                    handles will automatically be closed. This
                    parameter must not be NULL.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog open successfully.
```

### 9.1.6 DLL_API catClose(CAT_HANDLE handle);

```
Description:

Closes a valid catalog handle returned by either catCreate() or catOpen().
No more operation may then be performed with the handle. All catalog
files are closed.

Parameters:

    CAT_HANDLE handle:
                    Handle object from catCreate() or catOpen(). Must not
                    be NULL or changed from open call. Handle is checked
                    for integrity.
```

 Copyright © 2005, Doclinx, Inc.

```
Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog closed successfully.
```

### 9.1.7 DLL_API catDelItem(CAT_HANDLE cat, BIGINT docId, BOOLEAN entireArchive);

```
Description:

Deletes the specified document entry from the catalog. The entry number is
from one to N (the number of entries -- see catEntryCount(). When catUpdate
is run, all entries deleted will be removed. No searches will then be
able to find information from removed documents.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    BIGINT docId:   Document ID number. Number from 1 to the # of entries.

    BOOLEAN entireArchive:
                    If set and the entry being removed is an archive file,
                    then all subsequent entries associated with the archive
                    are also removed.
Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Entry docId removed successfully.
```

### 9.1.8 DLL_API catDelFile(CAT_HANDLE cat, CHAR *file);

```
Description:

Deletes the specified document entry from the catalog. The file name with
path must match the entry in the catalog. If the file is not found,
then a CAT_NOTFOUND error is returned. For archive files, the name must
match the name of the archive file (without archive component name addition).
All components of an archive are removed.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    CHAR *file:     Filename to match against name stored in catalog
                    entry list. Must not be NULL.

    BOOLEAN entireArchive:
                    If set and the entry being removed is an archive file,
                    then all subsequent entries associated with the archive
                    are also removed.
Returns:
```

 Copyright © 2005, Doclinx, Inc.

```
INT     < 0:    Error. See dpapi.h for error code definitions under
                ERROR RETURNS (Catalog Interface).
        = 0:    Entry indicated by filename removed successfully.
```

### 9.1.9 DLL_API catAddFile(CAT_HANDLE cat, CHAR *source, CHAR **props, INT mode, INT filter);

```
Description:

Adds one or more files to catalog entry list. All files are filtered
and converted to STF at this time. If a file is not found or a
parser error occurs, the call fails and nothing is added.
Note that the data in the files is not searchable until a catUpdate()
call has been processed.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    CHAR *source:   Specifies a file or directory based upon mode.
                    Must not be NULL.

    CHAR **props;   Optional property list table. If NULL, no property lists are
                    applied to the files. Each property is a two-value object string
                    with the values separated by a tab ('\t') character. The catalog
                    must have context enabled and each property name is a single tag
                    name (like a GID for XML). The property value can be a multiple
                    word text fragment. E.g.,

                    "Proper Name\tSteve J. Schmitt"

                    The property value content is then searchable as part of the given
                    document(s) using the property name:

                    Query: schmitt in tag "*Properties/Proper Name"


    INT mode:       Specifies type of file reference given by source:

                    ADD_MODE_FILE:  Single file path name. Can be directory.
                    ADD_MODE_LOF:   Source points to a list of files. All
                                    entries in text file will be opened.
                                    One line corresponds to one file name.
                                    An entry can be a directory reference.
                    ADD_MODE_DIR:   Source is a directory or a directory
                                    with a wildcard descriptor (e.g.
                                    c:\myDocuments\*.doc).
                    ADD_MODE_MAPFILE:
                                   Source parameter is a map file. A map file is a
                                    CSV list used primarily as web-crawler input
                                    specification. Each line corresponds to 1 file
                                    entry.
                                    Line format: URL,path,title,mimetype,encoding
                     ADD_MODE_MODIFIED:
                                    If file(s) exist in catalog and have been
                                    changed, delete entry and add new version. If
                                    new, just ad as normal. Can be combined with 1st
```

 Copyright © 2005, Doclinx, Inc.

```
                                4 mode settings.
                    ADD_MODE_AUTODEL:
                                If file(s) exist in catalog, then delete entry and
                                add again (no DUPLICATE ERROR). Can be combined
                                with 1st 4 mode settings

    INT filter:     Specifies what filter to using for converting to
                    STF.

                    NO_FILTER:      Assume file is STF format
                    HTML_FILTER:    Use custom HTML filter
                    XML_FILTER:     Scan for XML tags/attributes and build
                                    context tree.
                    GENERIC_FILTER: Use generic filter (type if determined
                                    by filter)
                    DETECT_FILTER:  Detect file type and use either
                                    generic, HTML or XML filter as determined
                                    by file content).

       If the Generic filter (or detect filter selects the generic filter) is
       specified and the SRC2STF_PARMS parameter sr_contextAaidx is non-zero, then the
       following generic tags may be available from an arbitrary document type. These
       contextual tags are used in a fashion similar to XML generic tags.

          DOC COMMENT             KEYWORD                 LAST SAVED BY
          PRIMARY AUTHOR          SUBJECT                 TITLE
          ABSTRACT                ACCOUNT                 ADDRESS
          ATTACHMENTS             AUTHORIZATION           BACKUP DATE
          BILL TO                 BLIND COPY              CARBON COPY
          CATEGORY                CHECKED BY              CLIENT
          COMPLETED DATE          COUNT CHARS             COUNT PAGES
          COUNT WORDS             CREATION DATE           DEPARTMENT
          DESTINATION             DISPOSITION             DIVISION
          EDIT MINUTES            EDITOR                  FORWARD TO
          GROUP                   LANGUAGE                LAST PRINT DATE
          MAIL STOP               MATTER                  OFFICE
          OPERATOR                OWNER                   PROJECT
          PUBLISHER               PURPOSE                 RECEIVED FROM
          RECORDED BY             RECORDED DATE           REFERENCE
          REVISION DATE           REVISION NOTES          REVISIONNUMBER
          SECONDARY AUTHOR        SECTION                 SECURITY
          SOURCE                  STATUS                  DOC TYPE
          TYPIST                  VERSION DATE            VERSION NOTES
          VERSION NUMBER          BASE FILE LOCATION      MANAGER
          COMPANY                 USER DEFINED PROP


       For example, if the property "ABSTRACT" is defined for a generic
       document (e.g., MS Word or PDF file), then the following query can be
       used in a search:

       Query: "important data" in tag "ABSTRACT"
              "crime" in tag "TITLE"

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
```

 Copyright © 2005, Doclinx, Inc.

```
          = 0:    Possible error, no files added.
          > 0:    Number of files successfully added.
```

### 9.1.10 DLL_API catEntryCount(CAT_HANDLE cat);

```
Description:

Returns the number of entries in the specified catalog. There is one
entry for each docId.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
           >= 0:    Number of entries in catalog. 0 implies not entries
                    exist.
```

### 9.1.11 DLL_API catEntrySize(CAT_HANDLE cat, BIGINT docId);

```
Description:

Returns the size (in bytes) of a catalog entry record. This enables
application to allocate sufficient memory to hold a selected record.
The catGetEntry() routines requires a buffer of sufficient size to
hold an entry. The size will vary per entry due to the variable length of
file names, titles, and auxiliary information.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    BIGINT docId:   Document ID number. Number from 1 to the # of entries.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
           >= 0:    Bytes required for to hold entry record.
```

### 9.1.12 DLL_API catEntryStringSize(CAT_HANDLE cat, BIGINT docId, INT item);

```
Description:

Returns the number of bytes required for the string specified by item
from the docId entry record. The length includes space for the terminating
```

 Copyright © 2005, Doclinx, Inc.

```
'\0';

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    BIGINT docId:   Document ID number. Number from 1 to the # of entries.

    INT    item:    CAT_FILESTR: Selects the file name variable length item.
                    CAT_AUXSTR:  Selects the auxiliary variable length item.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
           >= 0:    Bytes required for selected string (including '\0')
```

### 9.1.13 DLL_API catGetEntry(CAT_HANDLE cat, BIGINT docId, BYTE *buffer, INT bSize);

```
Description:

Returns the entire entry record, including the variable length string
data as a record that can be accessed using the structure definition CAT_ITEM
found in dpapi.h. The variable length data is appended to the end of the
structure.

e.g. cPtr = (CAT_ITEM *) buffer; // Where buffer is the parameter above

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    BIGINT docId:   Document ID number. Number from 1 to the # of entries.

    BYTE *buffer:   Point to buffer of sufficient size to hold entry.
                    (see catGetEntrySize()).

    INT  bSize:     Size of buffer (in bytes).

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
           >= 0:    Actual size of record returned.
```

### 9.1.14 DLL_API catGetEntryString(CAT_HANDLE cat, BIGINT docId, INT item, CHAR *buffer, INT bSize);

```
Description:

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.
```

 Copyright © 2005, Doclinx, Inc.

```
    BIGINT docId:   Document ID number. Number from 1 to the # of entries.

    INT    item:    CAT_FILESTR: Selects the file name variable length item.
                    CAT_AUXSTR:  Selects the auxiliary variable length item.

    BYTE *buffer:   Point to buffer of sufficient size to hold string item.
                    (see catGetEntryStringSize()).

    INT  bSize:     Size of buffer (in bytes).

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
           >= 0:    Actual length of string (as per strlen()).
```

### 9.1.15 DLL_API catUpdate(CAT_HANDLE cat, INT mode);

```
Description:

Performs a "build" or update of the current database set. After files
have been added or deleted, this function is called to allow searching
with this updated information. The mode controls some operations
performed during the update and whether to add to the primary or update
database (see mode parameter below).

An application can start with MODE_UPDATE 1st time. Since a primary does
not exist, it will be created. Then subsequent calls will build and add to
the secondary database. It is envisioned that a large primary will first be
created and then much smaller additions will be added to the update.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    INT mode:       MODE_PRIMARY: Update the primary database
                    MODE_UPDATE:  Update to the secondary database

                    One of the values above must be selected. You can
                    optionally 'or' (|) any of the following BIT
                    options to the mode parameter to control space
                    usage and access:

                    MODE_REMOVE_DELS:  Remove deleted file docIds
                                       from .DAT file.
                    MODE_COMPRESS_CAT: Remove deleted entries from
                                       catalog..
                    MODE_REHASH:       Rebuild hash lookup table at end of
                                       update.
                    MODE_MIN_LOCSIZE:  Minimize size of locators (resulting
                                       from deleted documents).

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
```

 Copyright © 2005, Doclinx, Inc.

```
            = 0:    Catalog database successfully updated.
```

### 9.1.16 DLL_API catPrimaryMerge(CAT_HANDLE cat, INT mode);

```
Description:

Forces a merge of the primary and secondary databases resulting in a single
primary database. This should be run at the point where there are too many
entries in the secondary database (e.g. > 1000). After merge, not update
exists until a catUpdate() with MODE_UPDATE is executed.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    INT mode:       MODE_PRIMARY: IMPLIED

                    You can optionally set any of the following BIT
                    options control space usage and access:

                    MODE_REMOVE_DELS:  Remove deleted file docIds
                                       from .DAT file.
                    MODE_COMPRESS_CAT: Remove deleted entries from
                                       catalog..
                    MODE_REHASH:       Rebuild hash lookup table at end of
                                       update.
                    MODE_MIN_LOCSIZE:  Minimize size of locators (resulting
                                       from deleted documents).

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Primary catalog database successfully updated.
```

### 9.1.17 DLL_API catMakeHashLookup(CAT_HANDLE cat, UINT8 pct);

```
Description:

Build or rebuilds the hash lookup data structure. The size of the
hash table is either re-used or recalculated depending on the pct
parameter. This is useful to call after an update with deletes
(and the MODE_REHASH option is not used) or after an number of files
have been added with the catAddFile() function.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    UINT8 pct:      If 0, then the current hashsize is used.
                    If between 1 and 100, then the hash table
                    if that percentage of the number of entries in
```

 Copyright © 2005, Doclinx, Inc.

```
                    the catalog.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog hash table successfully built.
```

### 9.1.18 DLL_API catFindFile(CAT_HANDLE cat, CHAR *fileName, FIND_MODE mode, BIGINT *result);

```
Description:

Finds the catalog entry corresponding to the specified file name.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    CHAR *fileName: String file name to search. Must not be NULL.

    FIND_MODE bDel: FIND_ALL: find any name in catalog
                    FIND_DEL: find only files that have been deleted
                    FIND_NOTDEL: find only files that are NOT deleted

    BIGINT *result: Pointer to integer. Returns the docId of requested
                    entry matching fileName. Must not be NULL.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Filename found.
```

### 9.1.19 DLL_API catSetLogging(CAT_HANDLE cat, CHAR *name, INT level);

```
Description:

Turns on logging of status/debug information to the specified file.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    CHAR *name:     Name of file to send log information. If NULL,
                    then output is sent to stdout. A "" string
                    disables output.

    INT  level:     Severity level threshold. The application
                    can restrict or expand the class of error and/or
                    debugging messages produced by the library.

                    DEBUG_LEVEL 3 -- Print all information
```

 Copyright © 2005, Doclinx, Inc.

```
                    WARN_LEVEL  2 -- Print only warnings and fatal messages.
                    FATAL_LEVEL 1 -- Print only fatal messages
                    NODEBUG     0 -- Do not print message.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog logging enabled.
```

### 9.1.20 DLL_API catEndLogging(CAT_HANDLE cat, BOOLEAN delFile);

```
Description:

Ends logging of build information.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    BOOLEAN delFile:
                    Deletes log file after it is closed.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog logging disabled.
```

### 9.1.21 DLL_API catSetParms(CAT_HANDLE cat, SRC2STF_PARMS *sPtr, DPAPI_PARMS *dPtr, MERGE_PARMS *mPtr);

```
Description:

Set the parameter block definitions for the catalog functions. The
structures are defined above. These override the following defaults
where allowed (some options can not be set by the application):

    SRC2STF_PARMS:

    Components that are invariant:

        sr_outputFile         = // Set by catalog routines.
        sr_appendToOutput     = TRUE;
        ht_inputIsListOfFiles = FALSE;
        sr_context            = // Set by catalog routines.
        sr_contextStr         = // Set by catalog routines.

    Default component settings:

        sr_flags              = SR_INCLUDEDOCTYPE;

        if (filter & (IS_XML | IS_AUTOTYPE))
```

 Copyright © 2005, Doclinx, Inc.

```
        ht_contextAaidx = DEFAULT_CONTEXT_AAIDX;

    if (filter & (IS_GENERIC | IS_AUTOTYPE))
        sr_genericAaidx = DEFAULT_GENERIC_AAIDX;

    if (catSetLogging() called)
        sr_logFile = log name;

    All other values are set to 0, FALSE, or NULL

Definitions:

sr_flag         – bit field. Permitted values for the sr_flag field.

  SR_INCLUDEDOCTYPE - Include document type integer in DRI 2 of the
                      parsed output (STF).
  SR_NOSPANSCRIPT   - Include words found in javascript block.
  SR_XMLSTRICT      - return parser error if XML file does not begin
                      with <?xml ....?>.

sr_stfFile      - Location for STF output file (defaults to catalog DIR)
sr_logFile      - Conversion information log file  (defaults to cat DIR)

// Only ONE of the following two modes can be used (i.e. one must == 0)
sr_contextAaidx - Context aaidx for context tags with generic filter
sr_genericAaidx – aaidx for generic filter (no context tree)

sr_appendToOutput      - Append to output (else overwrite)
sr_includePunctuation  - Place punctuation tokens in STF
sr_debug;              - Turn on parser debug info output
sr_enableJapanese;     - ** deprecated.
sr_maxWordChars        - Maximum length of a word (255 max)
sr_regExpression       - Regular expression word break rule (C++ only)

sr_foldSettings        - Control bits for case folding

   FOLD_DIGIT    - Fold all numerics to base ASCII numbers in lower 128
   FOLD_LATIN    - Fold all accented characters to basic Latin representation.

sr_includeWords;       - Control bits for word inclusion

   SR_INCLUDE_CDDATA   - If bit set, include CDDATA words in parse.
   SR_INCLUDE_COLL_HDR - If bit set, and processing XML composite
                         file, include words from encapsulating
                         header tag(s) for each composite file.


sr_encoding            - Text encoding to use (no auto detect)

sr_altTitle            - Alternate title
sr_indexAltTitle       - true if indexing alt title
sr_addedText           - Other added (non-indexed) text

// Include/Exclude fmt: *.xml;foo.*;file.ext

sr_includeList         - File include wildcards
sr_excludeList         - File exclude list

ht_contextAaidx        - Context aaidx for XML parsing
ht_stopOnFileError     - If set, stop ADD if error encountered.
```

 Copyright © 2005, Doclinx, Inc.

```
    ht_includeAttributes   - Output attribute name/attribute value data
    ht_warnUnknown         - Warn about unknown GIDs (HTML only)
    ht_inputIsListOfFiles  - ** deprecated
    ht_docIdStart          - Starting doc ID (ATTR 8), none = 0
    ht_defFile             - **deprecated
    ht_titleBuffer         - Optional title buffer
    ht_titleBufferSize;    - Size of above
    ht_documentsProcessed  - ** deprecated (always 1)



    DPAPI_PARMS:

    Components that are invariant:

        mk_bWeb           = 1;
        mk_outputPath     = // Set by catalog routines.
        dbuild_contextMap = // Set by catalog routines.
        invert_contextMap = // Set by catalog routines.

    Default component settings:

             invert_pct = 33;  // good value for default

        dbuild_contextAaidx:
        invert_contextAaidx:
            If above 0, then
            if the SRC2STF_PARMS are set, then uses ht_contextAaidx
            else uses DEFAULT_GENERIC_AAIDX

        All other values are set to 0, FALSE, and NULL


    MERGE_PARMS:

    Components that are invariant:

        mg_prm      = // Set by catalog routines.
        mg_upd      = // Set by catalog routines.
        mg_delList  = // Set by catalog routines.
        mg_locMap   = // Set by catalog routines.
        mk_bWeb     = 1;

    Default component settings:

        mg_contextAaidx = DEFAULT_CONTEXT_AAIDX;

        All other values are set to 0, FALSE, and NULL

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    SRC2STF_PARMS *sPtr:
                    Pointer to STF filter options. Can be NULL (use defaults).

    DPAPI_PARMS *dPtr:
                    Pointer to database build options. Can be NULL (use
                    defaults).
```

 Copyright © 2005, Doclinx, Inc.

```
    MERGE_PARMS *mPtr:
                    Pointer to database merge options. Can be NULL (use
                    defaults).

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog created successfully.
```

### 9.1.22 DLL_API catEntryStringUpdate(CAT_HANDLE cat, BIGINT docId, INT item, CHAR *su);

```
Description:

Function to change a variable string item in a catalog entry. The entry is
identified by the docId and which string is denoted with the item parameter.
It is most efficient to change the last item entered because this avoids any
potential compaction. There is a macro defined to simplify changing the
last filename string item:

    catLastEntryStringUpdateFile(cat, newStr)

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    BIGINT docId:   Document ID number. Number from 1 to the # of entries.

    INT    item:    CAT_FILESTR: Selects the file name variable length item.
                    CAT_AUXSTR:  Selects the auxiliary variable length item.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog created successfully.
```

### 9.1.23 DLL_API catSetFuzzyBuild(CAT_HANDLE cat, BOOLEAN set, INT32 size, INT maxCh);

```
Description:

Enable fuzzy operations for catalog. This function must be called to
enable building of fuzzy search structures. After the primary build,
fuzzy build can no longer be enabled. While fuzzy searching can later be
disabled, the value of performing that option is limited.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    BOOLEAN set:    Enable/disable building of catalog fuzzy structures.

    INT size:       % of hash entry count to allocate for hash table.
```

 Copyright © 2005, Doclinx, Inc.

```
                    0 indicates to use default. If creating the hash table,
                    and # is > 100, then that is the absolute # of entries
                    to use.

    INT maxCh:      Maximum length of phonetic match. 0 implies the default
                    which is to use all characters.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog created successfully.
```

### 9.1.24 DLL_API catSetStemBuild(CAT_HANDLE cat, BOOLEAN set, INT32 size);

```
Description:

Enable stemming operations for catalog. This function must be called to
enable building of stemming structures. After the primary build, stemming
can no longer be enabled. While stemming can later be disabled,
he value of performing that option is limited.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.

    BOOLEAN set:    Enable/disable building of catalog stemming structures.

    INT size:       % of hash entry count to allocate for hash table.
                    0 indicates to use default. If creating the hash table,
                    and # is > 100, then that is the absolute # of entries
                    to use.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog created successfully.
```

### 9.1.25 DLL_API catSetXMLSemantics(CAT_HANDLE cat, INT action, CHAR *pathList);

```
Description:

Sets XML semantic actions for XML (and HTML with context). Actions include
the following semantics defined by "action" to apply to element tags
(or attributes): These settings apply as long as the catalog is open or
until cleared. All subsequent "added" files use these semantics.

Parameters:

    CAT_HANDLE cat: Valid handle to opened catalog.
```

 Copyright © 2005, Doclinx, Inc.

```
    INT action:

        CLEAR_SEMANTICS:    Remove all tag element semantics (reset).

        XML_EMPTY_TAG:      Element has no end tag (HTML)
        XML_AUTO_END_TAG:   Element is automatically ended when same tag
                            is processed without intervening end tag (HTML).
        XML_TITLE_TAG:      Element contains title information.
        XML_ABSTRACT_TAG:   Element contains abstract information
        XML_PARA_TAG:       Element is a new paragraph
        XML_INT_TAG:        Element data are one or more 32-bit integers
        XML_REAL_TAG:       Element data are 32-bit floating point #(s).

       These "bit" settings (except for CLEAR_SEMANTICS) can be or'd together.

    CHAR *pathList:

       A string of one or more XML XPATHS separated by the "," character. Tag contents
       represented by each XPATH have the type action applied.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog created successfully.
```

### 9.1.26 DLL_API addMap8(CAT_HANDLE cat, CHAR *mapName, UINT16 *table);

```
Description:

This function adds a 256-character UNICODE mapping table to the catalog in order to
translate byte character sets to their corresponding UNICODE equivalents. Note that
the mapping tables must be set each time a catalog is opened for update (i.e. they are
not retained in the catalog information file). The tables are ONLY required when
adding files that contain character codes outside the standard ASCII-7 character set.
For example, if an HTML file has the following meta tag:

      <meta http-equiv="Content-Type" content="text/html; charset=windows-1256">

then this function should be called with "windows-1256" as the mapName and the proper
mapping table in a 256 item array of 16-bit codes. The codes should translate the byte
values in the added files to the proper UNICODE equivalents. Note that an error
condition is raised (or reported to the log file) if a mapping table specified in an
XML or HTML tag is encountered, but no mapping table has been set up using this
function. Several standard mapping tables are provided in the TeraXML package as .h
files. UTF8 and SHIFT_JIS encoded files are handled by default and do require any
mapping tables.

Parameters:

CAT_HANDLE cat:  Valid handle to opened catalog.

CHAR *mapName:   String representing the map table name. Should correspond to the
                 HTML/XML file encoding name string.

UINT16 *table;   Table of 256 entries containing the UNICODE value that corresponds to
                 the 8-bit value
```

 Copyright © 2005, Doclinx, Inc.

```
Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    ERROR RETURNS (Catalog Interface).
            = 0:    Catalog created successfully.
```

#### 9.1.27 DLL_API catWaitToExit(DWORD millesecTO);

```
Description:

This will wait until everything in the API has exited (entire DLL)
Can give an expiration (Time Out) value.

Parameters:

    DWORD millesecTO:
                Time out value to wait. If 0, wait until all complete.

Returns:
    INT     < 0:    Function timed out. See dpapi.h for error code definitions
                    ERROR RETURNS (Catalog Interface).
            = 0:    All API functions exited.
```

## 9.2 Search Subsystem API

#### 9.2.1 DLL_API catXSOpen(CHAR *path, CHAR *name, XS_HANDLE *handlePtr);

```
Description:

Opens a catalog for searching. The path and name correspond to the
parameters of the catOpen() call. The catalog must exist and should have
a least one catUpdate() execution applied. A handle is returned for use with
subsequent catalog search functions. If the catalog does not exist or
is corrupt, then function will fail.

Parameters:

    CHAR *path:     The full path name of the catalog area.
                    This parameter can NOT be NULL.

    CHAR *name:     Name of the catalog. The catalog must already
                    exist. This parameter can NOT be NULL.

    XS_HANDLE *handle:
                    Pointer to location to save search handle value. The
                    application does not have access to the contents of
                    the handle. The handle must be closed by the
```

 Copyright © 2005, Doclinx, Inc.

```
                    application using the catXSClose() function. The handle
                    is used in all subsequent cat search calls. This
                    parameter can NOT be NULL.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search Catalog opened successfully.
```

### 9.2.2 DLL_API catXSClose(XS_HANDLE xsh);

```
Description:

Closes a search catalog handle. Handle must be valid value returned from
catXSOpen() call.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search Catalog closed.
```

### 9.2.3 DLL_API catXSSearch(XS_HANDLE xsh, CHAR *query, INT mode);

```
Description:

Performs search of catalog. The query language is the standard TeraXML
query syntax with a couple of additions for context searching.

    (1). context search 'caps in tag "name/first"'

        finds <name>
                <first>caps</first>
                <last> bozo</last>
              </name>

    (2) "a little cap" will search for phrase in exact order by default.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    CHAR *query:    Search query -- must not be NULL.

    INT mode:       MODE_PRM:  Searches only primary database
                    MODE_UPD:  Searches only update database
                    MODE_BOTH: Searches entire catalog
```

 Copyright © 2005, Doclinx, Inc.

```
                    // Search Options

                    MODE_PLURAL - enable plural/deplural search
                    MODE_FUZZY  - enable fuzzy search
                    MODE_STEM   - enable stemming
                    MODE_THES   - enable word/tag thesaurus

                    // Search Operation

                    MODE_ANDORMODE - enable mode where all hits in a
                                     document (A & B() get ALL matching
                                     locators.

                    // Search Order

                    #define MODE_RELEVANCY_ORDER

                    // character folding modes (Note: Must be > 0x2000)

                    MODE_FOLD_DIGIT         FOLD_DIGIT
                    MODE_FOLD_LATIN         FOLD_LATIN

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search performed successfully.
```

### 9.2.4 DLL_API catXSGetDoc(XS_HANDLE xsh, UINT index, BIGINT *docId, BIGINT *hits);

```
Description:

After a successful search, a specific hit item can be retrieved.
The number of documents found is returned by catXSGetDocCount().
The item number can be from 1 to this document count.
The document id and the number of hits in that document are returned.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    UINT index:     Which document hit to retrieve.

    BIGINT *docId:  Pointer to location to put document ID of the
                    selected index item. Must not be NULL.

    BIGINT *hits:   Pointer to location to put number of hits
                    in document. Must not be NULL.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

 Copyright © 2005, Doclinx, Inc.

### 9.2.5 DLL_API catXSGetDocList(XS_HANDLE xsh, DOCHIT *list, UINT start, UINT nItems);

```
Description:

Retrieves up to nItems of document hit items. A document hit item
is a docId and the number of hits in that document. The total number
of document hits is obtained from catXSGetDocCount(). The structure
DOCHIT is defined in dpapi.h.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    DOCHIT *list:   Pointer to buffer of size nItems * sizeof(DOCHIT);

    UINT start:     Starting index of nth item in DOCHIT list. First list
                    item is index 1.

    UINT nItems:    Number of DOCHIT items to retrieve. Note that start +
                    nItems - 1 can not be greater than the total number
                    of document hits.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

### 9.2.6 DLL_API catXSHitList(XS_HANDLE xsh, UINT16 *aaList, UINT16 *buffer, UINT start, UINT nItems);

```
Description:

Retrieves the list of actual document locators. A document locator
describes the position in the document of the match and any of its
associated attributes. The aaList parameter dictates which values to
return. The buffer must be large enough to hold the number of attributes
per item requested times the number of items.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    UINT16 *aaList: A MAXUINT16 terminated variable list of attribute indexes.
                    Some attributes indicate document #, paragraph #, and
                    word count. Others are field attributes (e.g. title
                    or context id). Any or all of the 17 attribute values
                    can be returned. rsapi.h provides the STF mappings
                    for these indexes (see STF_ATTRIBUTE_INDICES)

    UINT16 *buffer: A buffer of length(aaList) * nItems * sizeof(UINT16)
                    bytes where length(aaList) in the number of indexes
                    in the list before MAXUINT16 terminator
                    (e.g {0, 3, 6, 15, 0xffff} would be length of 4)
```

 Copyright © 2005, Doclinx, Inc.

```
    UINT start:     Starting index of nth item in locator list. First list
                    item is index 1.

    UINT nItems:    Number of locator items to retrieve. Note that start +
                    nItems - 1 can not be greater than the total number
                    of document hits.

    NOTE:
        Macro for getting the number of "hits" returned from catXSGetHitList()

        #define catXSGetHitsReturned(xs, cnt) catXSGetExtendedError(xs, cnt)

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

### 9.2.7 DLL_API catXSGetRelevancyList(XS_HANDLE xsh, DOCHIT *list, UINT start, UINT nItems);

```
Description:

Retrieves up to nItems of document relevancy list. Similar to catXSGetDocList, except
that the documents ID's are returned in relevancy order and the weighting metirc is
returned in the hit count field. This retrieves the document # (catalog entry) and
relevancy from a set of search results. The relevancy number is a ranking value used
to order the terms. The set is organized from largest
most relevant) to smallest. Note that the catXSSearch method must be called before
performing this method.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    UINT16 *aaList: A MAXUINT16 terminated variable list of attribute indexes.
                    Some attributes indicate document #, paragraph #, and
                    word count. Others are field attributes (e.g. title
                    or context id). Any or all of the 17 attribute values
                    can be returned. rsapi.h provides the STF mappings
                    for these indexes (see STF_ATTRIBUTE_INDICES)

    UINT16 *buffer: A buffer of length(aaList) * nItems * sizeof(UINT16)
                    bytes where length(aaList) in the number of indexes
                    in the list before MAXUINT16 terminator
                    (e.g {0, 3, 6, 15, 0xffff} would be length of 4)

    UINT start:     Starting index of nth item in locator list. First list
                    item is index 1.

    UINT nItems:    Number of locator items to retrieve. Note that start +
                    nItems - 1 can not be greater than the total number
                    of document hits.

    NOTE:
```

 Copyright © 2005, Doclinx, Inc.

```
        Macro for getting the number of "hits" returned from catXSGetHitList()

        #define catXSGetHitsReturned(xs, cnt) catXSGetExtendedError(xs, cnt)

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

### 9.2.8 DLL_API catXSGetDocCount(XS_HANDLE xsh, BIGINT *nDocs);

```
Description:

Returns the number of document "hits" resulting from a search.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    BIGINT *nDocs:  Pointer to location for return value. The
                    number of documents matching the search query is
                    retrieved. The parameter must not be NULL.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

### 9.2.9 DLL_API catXSGetHitCount(XS_HANDLE xsh, BIGINT *nHits);

```
Description:

Returns the total number of locator hits resulting from a search.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    BIGINT *nHits   Pointer to location for return value. The
                    total number of locator hits is retrieved.
                    The parameter must not be NULL.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

 Copyright © 2005, Doclinx, Inc.

### 9.2.10 DLL_API catXSSetRelevancyTags(XS_HANDLE xsh, INT value, CHAR *tag);

```
Description:

Set multiplier for highly relevant tags (e.g. <title>). Allows application to set tag
multiplier to influence relevancy feedback. By default, multiplier is 1. Applications
can set this value higher (max value suggested is 16). Any words that match a query
and occur within specified tags will use the specified multiplier. A value of zero can
be used to omit selected tags from relevancy calculations.

Parameters:

       XS_HANDLE xsh:  Valid handle to opened search catalog object.

       INT value;      Relevancy multiplier (0..16)

       CHAR *tag;      XPath of tag expression or special value:
                           "<TITLE>": If hit has attrib. array[7] set, use multipler.
                           "<PROX>" : If proxmity used, then weight terms in proximity
                                      by multiplier.
                           "<EXACT>": If words in exact order as query, weight with
                                      multiplier
Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

### 9.2.11 DLL_API catXSGetDocMax(XS_HANDLE xsh, UINT32 *maxDoc, INT mode);

```
Description:

Returns the maximum document value in the primary or update database
or the combination of the two.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    UINT32 *maxDoc: Pointer to location to return maximum document
                    information. This parameter must not be NULL.

    INT mode:       MODE_PRM:  Retrieve the number of documents in
                               primary database.
                    MODE_UPD:  Retrieve the number of documents in
                               update database.
                    MODE_BOTH: Retrieve the sum of the documents
                               for the primary and update databases.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```


Copyright © 2005, Doclinx, Inc.

### 9.2.12 DLL_API catXSGetSearchTime(XS_HANDLE xsh, UINT32 *millSecs);

```
Description:

Returns time required to perform search.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    UINT32 *millSecs:
                    Pointer to location to return time. Time is
                    in milliseconds units. Parameter must not be NULL.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

### 9.2.13 DLL_API catXSGetExtendedError(XS_HANDLE xsh, INT *error);

```
Description:

Returns extended error information (RSAPI error code or query syntax
error).

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    INT *error:     Extended error code.

                    If function return is XS_RSERROR, then see
                    RS_STATUS error codes in rsapi.h.

                    If function return is XS_RSQUERY, then see
                    QERROR errors in rsapi.h.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

 Copyright © 2005, Doclinx, Inc.

### 9.2.14 DLL_API catXSSetLogFile(XS_HANDLE xsh, CHAR *fileName);

```
Description:

Sets logging for search operations.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    CHAR *filename: File path of log file.

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

### 9.2.15 DLL_API catXSGetCatalog(XS_HANDLE xsh, CAT_HANDLE *handlePtr);

```
Description:

Returns the catalog handle in use for the search operations.

Parameters:

    XS_HANDLE xsh:  Valid handle to opened search catalog object.

    CAT_HANDLE *handlePtr:
                    Pointer to location to return handle. This is useful
                    to get catalog handle in order to retrieve
                    information from result doc IDs (e.g. to get
                    a title or filename associated with a doc hit).

Returns:

    INT     < 0:    Error. See dpapi.h for error code definitions under
                    "Catalog Search Error Returns".
            = 0:    Search operation successful.
```

### 9.2.16 DLL_API catXSFinish(VOID);

```
Description:

Closes all outstanding open search handles and releases resources.

Parameters:         NONE

Returns:
```

 Copyright © 2005, Doclinx, Inc.

```
INT     < 0:    Error. See dpapi.h for error code definitions under
                "Catalog Search Error Returns".
        = 0:    Close of all search handles and resources successful.
```

## 9.3 Building with the C API

### 9.3.1 Include Files

The following two include files are required:

```
dpapi.h         // Include in this order
rsapi.h
```

Note: dpdefs.h and stddefx.h are included by dpapi.h and rsapi.h and contain basic definitions.

### 9.3.2 Library files

```
dataprep.lib    // Include these to lib files in link
libxml.lib
```

### 9.3.3 DLL files

```
dataprep.dll    // Load or place DLLs in standard locations
libxml.dll
```

### 9.3.4 Compiler Options

Make sure the following defines are set in the compiles of modules using these libraries:

```
_WIN32,SEEK_64,WIN32,_WINDOWS
```

Structure alignment is the standard 8-byte alignment factor

 Copyright © 2005, Doclinx, Inc.

# 10. Error Conditions and Recovery

## 10.1 Indexing Subsystem Error Codes

The Indexing Subsystem functions return a 32-bit integer code indication the success of an operation as shown the table below. All errors are indicated by values less than zero. A positive value indicates the function performed correctly. Most functions return zero; however, some procedures return counts that are greater than zero. Error codes are located in the include\dpapi.h file.

| | |
|---|---|
| CAT_SUCCESS | OK: Catalog Manager API function completed correctly |
| CAT_ERROR | General internal error -- not returned via API |
| CAT_NOTFOUND | When requested document (#) or file does not exist |
| CAT_FILEERROR | When file ops such as copy, rename, or delete fail |
| CAT_BUFFERSIZE | Unable to calculate entry size in catGetEntryBytes |
| CAT_LOGICERROR | Internal assertion or logic check failed |
| CAT_PARMERROR | API parameter out of range or <code>null</code> |
| CAT_ADDERROR | Error closing catalog in fail-safe mode (recoverable) |
| CAT_FIXEDERROR | I/O error reading/writing catalog .idx file |
| CAT_VARERROR | I/O error reading/writing catalog .str file |
| CAT_LIBERROR | Error in library function -- Buffer system failure, or error compressing/deleting items from catalog |
| CAT_LSTERROR | ** deprecated |
| CAT_HASHERROR | Error building/accessing hash lookup table for file entries |
| CAT_FILTERERROR | The parser (filter) returned an error when processing text in a file. See cat_ExtendedError for more information. |
| CAT_DPERROR | Error in building inverted index. See cat_ExtendedError. |
| CAT_MERGEERROR | Error merging databases. See cat_ExtendedError |
| CAT_DUPERROR | Attempt to add a file that already exists in catalog. |

 Copyright © 2005, Doclinx, Inc.

| | |
|---|---|
| CAT_STFERROR | Empty STF file, unable to build index |
| CAT_CREATEPARSER | Unable to create parser object |
| CAT_BADHANDLE | CatalogManager handle not initialized, already opened, closed. |
| CAT_DIRERROR | Directory does not exist or can not be created. |
| CAT_EXISTSERROR | Requested catalog already exists in catCreate() |
| CAT_TO | ** deprecated |
| CAT_CREATEXML | ** deprecated |
| CAT_CONTEXTMERGE | Error merging context trees for XML or tagged files |
| CAT_CONTEXTFAIL | Error building context tree |
| CAT_FUZZYERROR | Setting fuzzy match option after catalog has indexed data. |
| CAT_FUZZYOPEN | Error opening fuzzy logic file. |
| CAT_FUZZYMERGE | Error merging fuzzy logic files. (corrupt) |
| CAT_STEMERROR | Setting stemming mode after catalog has indexed data |
| CAT_STEMOPEN | Error opening stem thesaurus file. |
| CAT_STEMMERGE | Error merging stem thesaurus files. (corrupt). |
| CAT_READONLYERROR | Calling catalog changing function while opened in READ ONLY mode. |
| CAT_SECURITYERROR | ** Java version only. |
| CAT_UNKNOWNERROR | ** Java version only. |
| CAT_SHAREERROR | ** Java version only. |
| CAT_FILEOPENERROR | Unable to open file header (i.e. determine file type) |
| CAT_SPARMERROR | Call to catAddMap8() function without SRC2STF_PARMS set. |
| CAT_PROCESSERROR | Error in process file (for recoverable ADD mode) |
| CAT_ADDEXECERROR | Error executing add process in fail-safe (recoverable) mode. |
| CAT_NOTIMPLEMENTED | Filter not implemented. |
| CAT_NOPRIMARY | No primary database found for merge. |
| CAT_NOUPDATE | No update database found for merge. |

 Copyright © 2005, Doclinx, Inc.

## 10.2 Search Subsystem Error Codes

The Search/Retrieval subsystem (catXS* set of functions) returns "XS_" errors as shown in the table below. Most functions require a valid XS_HANDLE that is checked by every function; if not valid, the XS_BADHANDLE error is returned. Extra error information can be obtained (if available) after receiving an error code return. The catXSGetExtendedError() can retrieve this information unless the error is a BADHANDLE or XS_OBSOLETE error.

| | |
|---|---|
| XS_SUCCESS | No error (0) - ERRORS < 0 |
| XS_ERROR | ** deprecated |
| XS_INITERROR | Unable to initialize search library (missing DB files). |
| XS_PARMERROR | Bad parameter values (range or null) to search function. |
| XS_CATOPEN | Unable to open CatalogManager handle, bad path or missing catalog. |
| XS_DBOPEN | Unable to open database handle in catalog (.prm or .upd files). |
| XS_BADHANDLE | CatalogSearch object not initialized or opened. |
| XS_RSERROR | Search library error. See catXSGetExtendedError() for more details. |
| XS_RSQUERY | Bad query for catXSSearch. See #catXSGetExtendedError() for more details. |
| XS_NOUPDATE | Attempt to search just update database and update does not exist. |
| XS_NOPRIMARY | Attempt to search primary with no index data. |
| XS_NORESULTS | Request for results when no search performed or empty search (e.g. catXSGetDoc() method). |
| XS_NODB | Attempt to open database that has not been indexed. |
| XS_CLOSERROR | Error closing CatalogManager object. |
| XS_CATERROR | **deprecated. |
| XS_LOGICERROR | Unexpected error closing handle. |
| XS_ALLOCERROR | Unable to allocate document list in search open. |
| XS_CONTEXTERROR | Unable to open context handle. |
| XS_THESOPEN | Unable to open Thesaurus object. |
| XS_STEMOPEN | Unable to open Stemming map object. |
| XS_FUZZYOPEN | Unable to open Fuzzy lookup object. |
| XS_SELECTERROR | Search library error. See catXSGetExtendedError()for more details. |

 Copyright © 2005, Doclinx, Inc.

| | |
|---|---|
| XS_OBSOLETE | Catalog has been updated - refresh handle for latest state. |
| XS_NODOCS | No documents in database. (all deleted). |
| XS_RSSEARCH | Search error (but not query format error). See catXSGetExtendedError(). |
| XS_RELTAG | Error in tag lookup for relevancy weighting. |
| XS_UNKNOWN | Unexpected error in CatalogSearch method. |

### 10.3 Linguistics Subsystem Error codes

The Linguistics Subsystem does not implement error codes. Error conditions are flagged by a descriptive error message in the XML message returned by the server.


Copyright © 2005, Doclinx, Inc.

# 11. Language Codes

TeraXML uses ISO 639 language codes. The following table describes the language codes used by the Linguistics Subsystem.

| Language | Code |
|---|---|
| English | en |
| Chinese | zh |
| Korean | ko |
| Japanese | ja |
| German | de |
| French | fr |
| Italian | it |
| Spanish | es |
| Arabic | ar |

A complete list of ISO 639 language codes can be found at the URL http://www.w3.org/WAI/ER/IG/ert/iso639.htm.


Copyright © 2005, Doclinx, Inc.

# 12. API Glossary

| Term | Definition |
|---|---|
| Attribute | A 16-bit field available to define word context. An attribute array is the collection of all attributes that apply to a particular word, object, or operation. TeraXML provides 16 attributes, some of which are used by the system and other may be user defined. |
| CAT_HANDLE | Handle to catalog operations. |
| Context Tree | Tree containing the tag/attribute information. This is an n-ary tree with a control root and organizes with level 1 sub-trees for each document type. |
| .DAT file | File type that contains the full text searchable database. The .upd and .prm extensions are used to denote primary and update versions of the .dat file. |
| DOC_HIT | A document ID and locator hit count pair. |
| Double Metaphone | Algorithm for performing a phonetic reduction. Superior to more common Soundex procedures for name matching. |
| DRI | Dictionary Region Index. A value between 0 and 15. There are 16 different possible dictionaries. Each dictionary can have its own type. |
| Fuzzy Search | Method of taking phonetic reductions of words in order to match correctly and/or incorrectly spelled variations. |
| Locator | A document position and attribute value object (array of UINT16) |
| Stemming | The process of decomposing a word into its root word. |
| SRC2STF_PARMS | Control block parameter for the Catalog API. Controls and specifies optional information for the parser modules. |
| STF | Standard Token Format. Input file format to data preparation process. Provides a common interchange text format for the indexing software. |
| XS_HANDLE | Handle to XS search operation. |


Copyright © 2005, Doclinx, Inc.

# 13. Technical Support

Doclinx provides technical support for TeraXML via email. Please direct support questions to techsupport@doclinx.com.

Doclinx offers customized technical support such as 24x7 support, on-site support etc. upon request.


Copyright © 2005, Doclinx, Inc.