Document Filters, Search Engines & The Anatomy Of A Binary Format

When you view a document in Microsoft Word, you expect the text to be crystal clear. The same applies when you display a database in Access, a presentation file in PowerPoint, a spreadsheet in Excel, a PDF in Adobe Reader, an email in Outlook/Exchange or Thunderbird, etc. Further, these applications make it easy not only to view the text but also to locate specific words for basic navigation within the file.

But what if you need to search across millions or billions of files? Pulling up each file individually in its associated application would take far too much time. Opening an untrusted document in its native application also creates a risk of virus infection. Instead, you would want a separate search engine to automatically search through all the data at once.

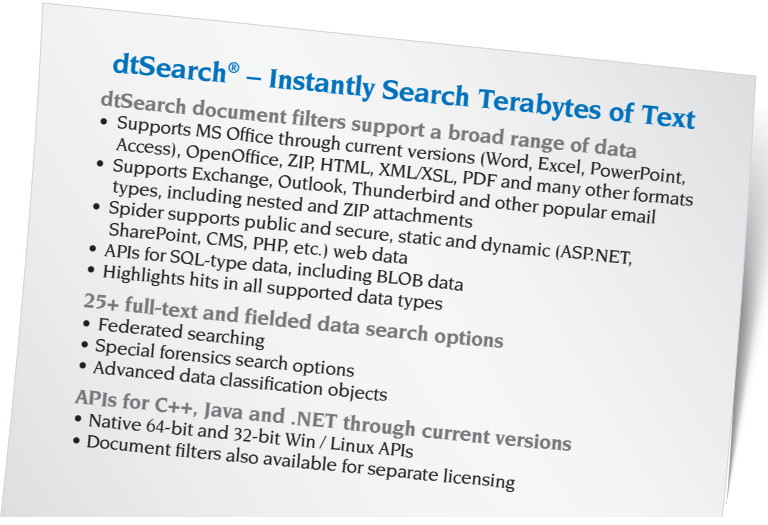
Binary Formats

Just as it is inefficient for you to sequentially retrieve a large number of files in their associated applications, so that process is inefficient for a search engine. Instead, a search engine needs to review data in binary format, bypassing the need to pull up each file in a separate program.

The problem is that file text that looks crystal clear inside its associated application typically appears as gibberish in binary format. Take a look at the image at the top right of this page for a look at a product description as it appears in Word. The bottom image shows a sample from this document as it appears in binary format.

Returning this binary format to the readable text that appeared in Word requires a lot of parsing. The industry name for the process that parses binary formats is document filters.

Document filters and search engines all parse binary formats to different levels of depth. The parsing process this article describes reflects the dtSearch® product line. While this article's anatomy of binary formats is a general one, the stages this article describes to unravel these formats may not precisely reflect other product lines than dtSearch.



File text that looks crystal clear inside its associated application typically appears as gibberish in binary format.

Binary Format Identification

Before parsing a binary format, the document filters need to identify what type of document or other object the binary format represents. In fact, identifying the right data specification is all-important, as the file specification for Word is nothing like the specification for Outlook or PDF.

Further, the document filters need to figure out the data type of a binary format preferably without reference to any document name or extension. For example, suppose a user saves a Word file with an

extension of .PDF instead of the Word extension .DOCX. Only by using the binary format itself to identify the data type instead of the extension can document filters effectively recognize and parse this file.

Evolving Specifications & Unicode

After figuring out the data type, the document filters can begin to apply the correct specification to decode the data. File specification data can be enormous. For example, Microsoft's documentation of the .DOC Word file format alone is more than 600 pages.

The document filters must also take into account the fact that all major data formats continue to evolve. If Microsoft makes a change to the .DOCX Word specification, the document filters have to apply this update for all new Word documents. And the document filters have to do so without interfering with the parsing of existing Word documents.

The next item for the document filters is to identify relevant text encoding. Some documents such as newer versions of Word store data in Unicode. Other document formats can store text in language-specific encodings, which the document filters must identify and translate into Unicode.

Metadata & Recursively Embedded Objects

In addition to parsing the main body of the text, the document filters have to identify and correctly handle other elements of a document, including headers and footers, fields such as subject and author, and even potentially hidden metadata. Then there is the issue of nested objects.

A Word document can embed an Access database, which can itself embed an Excel spreadsheet, which can further embed a PowerPoint. The document filters need to recognize and drill through all of the different levels of nested document objects to fully parse the text.

Document filters and search engines all parse binary formats to different levels of depth. The parsing process this article describes reflects the dtSearch® product line. While this article's anatomy of binary formats is a general one, the stages this article describes to unravel these formats may not precisely reflect other product lines than dtSearch.

Database & Online Data

It is not only documents that can embed other documents as nested objects. An SQL database can store documents inside BLOB data within the database. An email can attach documents directly or as part of a ZIP or RAR archive. Documents — including standard Office files such as Word documents or emails — can appear online in the context of Web-based static (HTML, XSL/XML, PDF, etc.) data. Or they can appear within Web-based dynamic data (MS SharePoint, ASP.NET, CMS, PHP, etc.).

The document filters need to handle all of these different data types just to ensure proper handling of documents. And that's not even to mention the surrounding SQL, email, compression, static, and online data itself, which the search engine needs to handle for comprehensive full-text searching.

Document Filters In Context

Parsing data is just the initial step for a search engine like dtSearch. After parsing the data, the search engine needs to create a search index. The search index itself is simply a programmatic device to enable very fast searching of a wide range of data.

A single search index can hold a large variety of data, including documents, emails and attachments, databases, and other Web-based static and dynamic data. In doing so, the index can enable concurrent or multithreaded federated searching across all of these different data types at once. After processing a search request from its index, the search engine will return a list of matching files or other data.

The search engine then returns to the document filters to display the complete text of retrieved data. dtSearch products display the complete text by converting data types that are not already Web-ready to HTML for browser-based display. The final step is to retrieve "hit offsets" from the index. The hit offsets tell the search engine and its document filters where to highlight hits in the browser-based data display. P

dtSearch

This article describes the dtSearch product line. Please visit www.dtsearch.com for fully-functional evaluation versions of all dtSearch products, as well as hundreds of reviews and developer case studies.

(800) IT-FINDS or (301) 263-0731 • www.dtsearch.com