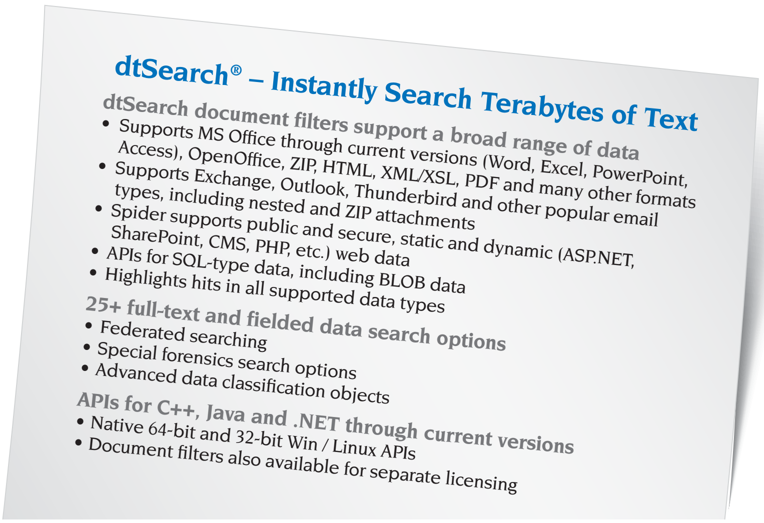
Document Filters, Search Engines & The Anatomy Of A Binary Format

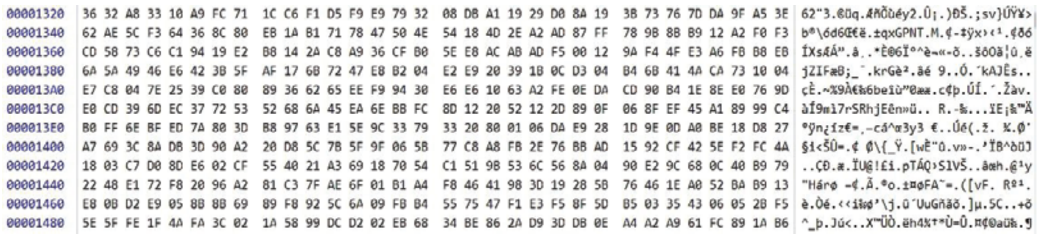


dtSearch® – Instantly Search Terabytes of Text

dtSearch document filters support a broad range of data
- Supports MS Office through current versions (Word, Excel, PowerPoint, Access), OpenOffice, ZIP, HTML, XML/XSL, PDF and many other formats
- Supports Exchange, Outlook, Thunderbird and other popular email types, including nested and ZIP attachments
- Spider supports public and secure, static and dynamic (ASP.NET, SharePoint, CMS, PHP, etc.) web data
- APIs for SQL-type data, including BLOB data
- Highlights hits in all supported data types

25+ full-text and fielded data search options
- Federated searching
- Special forensics search options
- Advanced data classification objects

APIs for C++, Java and .NET through current versions
- Native 64-bit and 32-bit Win / Linux APIs
- Document filters also available for separate licensing



```
00001320  36 32 A8 33 10 A9 FC 71  1C C6 F1 D5 F9 E9 79 32  08 DB A1 19 29 D0 8A 19  3B 73 76 7D DA 9F A5 3E  62"3.©□q.ÆÑÕùéy2.Û¡.)ÐŠ.;sv}ÚŸ¥>
00001340  62 AE 5C F3 64 36 8C 80  EB 1A B1 71 78 47 50 4E  54 18 4D 2E A2 AD 87 FF  78 9B 8B B9 12 A2 F0 F3  b®\ód6Œ€ë.±qxGPNT.M.¢-‡ÿx›‹¹.¢ðó
00001360  CD 58 73 C6 C1 94 19 E2  B8 14 2A C8 A9 36 CF B0  5E E8 AC AB AD F5 00 12  9A F4 4F E3 A6 FB B8 EB  ÍXsÆÁ".â¸.*È©6Ï°^è¬«-õ..šôOã¦û¸ë
00001380  6A 5A 49 46 E6 42 3B 5F  AF 17 6B 72 47 E8 B2 04  E2 E9 20 39 1B 0C D3 04  B4 6B 41 4A CA 73 10 04  jZIFæB;_¯.krGè².âé 9..Ó.´kAJÊs..
000013A0  E7 C8 04 7E 25 39 C0 80  89 36 62 65 EE F9 94 30  E6 E6 10 63 A2 FE 0E DA  CD 90 B4 1E 8E E0 76 9D  çÈ.~%9À€‰6beîù"0ææ.c¢þ.ÚÍ.´.Žàv.
000013C0  E0 CD 39 6D EC 37 72 53  52 68 6A 45 EA 6E BB FC  8D 12 20 52 12 2D 89 0F  06 8F EF 45 A1 89 99 C4  àÍ9mì7rSRhjEên»ü.. R.-‰...ïE¡‰™Ä
000013E0  B0 FF 6E BF ED 7A 80 3D  B8 97 63 E1 5E 9C 33 79  33 20 80 01 06 DA E9 28  1D 9E 0D A0 BE 18 D8 27  °ÿn¿íz€=¸—cá^œ3y3 €..Úé(.ž. ¾.Ø'
00001400  A7 69 3C 8A DB 3D 90 A2  20 D8 5C 7B 5F 9F 06 5B  77 C8 A8 FB 2E 76 BB AD  15 92 CF 42 5E F2 FC 4A  §i<ŠÛ=.¢ Ø\{_Ÿ.[wÈ¨û.v»-.'ÏB^òüJ
00001420  18 03 C7 D0 8D E6 02 CF  55 40 21 A3 69 18 70 54  C1 51 9B 53 6C 56 8A 04  90 E2 9C 68 0C 40 B9 79  ..ÇÐ.æ.ÏU@!£i.pTÁQ›SlVŠ..âœh.@¹y
00001440  22 48 E1 72 F8 20 96 A2  81 C3 7F AE 6F 01 B1 A4  F8 46 41 98 3D 19 28 5B  76 46 1E A0 52 BA B9 13  "Hárø –¢.Ã.®o.±¤øFA˜=.([vF. Rº¹.
00001460  E8 0B D2 E9 05 8B 8B 69  89 F8 92 5C 6A 09 FB B4  55 75 47 F1 E3 F5 8F 5D  B5 03 35 43 06 05 2B F5  è.Òé.‹‹i‰ø'\j.û´UuGñãõ.]µ.5C..+õ
00001480  5E 5F FE 1F 4A FA 3C 02  1A 58 99 DC D2 02 EB 68  34 BE 86 2A D9 3D DB 0E  A4 A2 A9 61 FC 89 1A B6  ^_þ.Jú<..X™ÜÒ.ëh4¾†*Ù=Û.¤¢©aü‰.¶
```

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

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.

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.

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.

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