re-Isearch

Handbook 0.9 (work in progress)

Author: Edward C. Zimmermann <edz@nonmonotonic.net>

This project was funded through the NGI0 Discovery Fund, a fund established by NLnet with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology under grant agreement No 825322.

The latest version should be available via http://www.nonmonotonic.net/re-isearch

I. Background / History

Isearch was an open-source text retrieval software first developed in 1994 as part of the Isite Z39.50 information framework. The project started at the Clearinghouse for Networked Information Discovery and Retrieval (CNIDR) of the North Carolina supercomputing center MCNC and funded by the National Science Foundation to follow in the track of WAIS (Wide Area Information Server) and develop prototype systems for distributed information networks encompassing Internet applications, library catalogs and other information resources. Edward Zimmermann having worked both on WAIS and his own Z39.50 server quickly joined the fray with his company BSn. From 1994 to 1998 most of the development was centered on the Clearinghouse for Networked Information Discovery and Retrieval (CNIDR) in Research Triangle, North Carolina and BSn in Munich. By 1998 much of the open-source core developers re-focused development into several spin-offs. In 1998 it became part of the Advanced Search Facility reference software platform funded by the U.S. Department of Commerce.

Isearch was widely adopted and used in hundreds of public search sites, including many high profile projects such as the U.S. Patent and Trademark Office (USPTO) patent search, the Federal Geographic Data Clearinghouse (FGDC), the NASA Global Change Master Directory, the NASA EOS Guide System, the NASA Catalog Interoperability Project, the astronomical pre-print service based at the Space Telescope Science Institute, The PCT Electronic Gazette at the World Intellectual Property Organization (WIPO), [[Linsearch (a search engine for Open Source Software designed by Miles Efron), the SAGE Project of the Special Collections Department at Emory University, Eco Companion Australasia (an environmental geospatial resources catalog), the Open Directory Project and numerous governmental portals in the context of the Government Information Locator Service (GILS) GPO mandate (ended in 2005). A number of sites worldwide continue (despite development long ago suspended) continue to use Isearch in their production systems.

One of the main split-offs was the (closed source and proprietary) IB engine developed by Bsn. With some new algorithms it was bespoke deployed in a large number of high profile projects ranging from news search for Wirtualna Polska (one of the largest and most known Web portals in Poland); genomic search for the Australian National Genomic Information Service's human genome project (and its eBiotechnology workbench split-off); the D-A-S-H

search portal against racism, antisemitism and exclusion (funded within the framework of the action program "Youth for tolerance and democracy - against right-wing extremism, xenophobia and anti-Semitism", the YOUTH program of the European Community and with additional support from the German Federal Agency for Civic Education); the e-government search (Yeehaw) of the U.S. State of Utah to agronomic cooperation across the Mediterranean region.

Its radical approach and re-think of search was even on display at the ISEA2008: 14th International Symposium on Electronic Art in a collaboration with the Dutch design cooperative Metahaven: https://isea-archives.siggraph.org/art-events/metahaven-exodus-cross-search/. In the words of the project "*Exodus is the compound name for a 'research engine' into algorithms and visual strategies for searching the internet, revealing the structural properties of web content and its inherent distribution of influence. Exodus promotes bridging behaviour across the web's new borders of power.*"

Development of IB halted in 2011 as its main developers moved on to other projects. While still being deployed by a number of sites—why break a working system—it was no longer updated or actively maintained (even before 2011 it seemed that most sites only bothered updating when they shifted hardware platforms and had to). The software sat idle in the attic for 10 years.

Now through the generous support of the Nlnet Foundation and the European Union's Next Generation Internet (NGI) initiative its being reborn,open-sourced and renamed as the re-Isearch engine (as a tribute to its roots).

II. Motivation. What does re-Isearch offer?

Mainstream search engines are about finding any information: "a list of all documents containing a specific word or phrase". Because of this, search engines paradoxically return both too much information (i.e. long lists of links) and too little information (i.e. links to content, not content itself). The re-Isearch engine is, by contrast, about exploiting document structure, both implicit (XML and other markup) and explicit (visual groupings such as paragraph), to zero in on relevant sections of documents, not just links to documents.

Organizations of all sizes and within all industries generally distribute their corporate knowledge amid a variety of applications: from customer relationship systems, staff directories, content management systems (CMS), electronic document and records management systems (EDRMS) to library catalogs. These Balkanized data stores tend to demand large efforts to extract, transform, load (or ingest).

Our goal is, by contrast, to provide enabling technology to develop and provide distributed federated information search and retrieval services to a heterogeneous mix of text, data (a large number of standard types such as numerical, ranges, dates etc), images/video/audio, geographic information, network objects and databases.

Key Features and Benefits:

- ➔ Cost effective access to a heterogeneous mix of XML and other data of any shape and size. Allows for the rapid creation of scalable (XML) warehouses.

Handbook 0.9 (work in progress)

- ➔ All the capabilities you can ever expect in an enterprise search solution and then some: including phrase, boolean, proximity, wildcard, parametric, range, phonetic, fuzzy, thesauri, polymorphism, datatypes (including numeric, dates, geospatial, ranges etc.) and object capabilities.
- ➔ Relevant ranking by a number of models including spatial score for geospatial queries, date, term frequency, match distribution etc.
- ➔ Object oriented document model: Supports W3C XML, ISO Standard 8879:1986 SGML and a wide range of common file (such as Word, Excel, RTF, PDF, PostScript, HTML, Mail, News), citation (such as BibTex, Endnote, Medline, Papyrus, Refer, Reference Manager/RIS, Dialog, etc), scientific, ISO and industry formats including standards such as USPTO Green Book (patents), DIF (Directory Interchange Format), CAP (Common Alerting Format) and many more.
- ➔ Automatic structure recognition and identification for "unstructured" textual formats (e.g., such as, alongside metadata, lines, sentences, paragraphs and pages in PDF documents).
- ➔ Sophisticated extendable type system allowing for numerical, date, geospatial and other search strategies, including external datastores and brokers parallel to textual methods: "Universal Indexing".
- ➔ Synchronized information: As soon as context its indexed (appended) its available. Functional Append/Delete/Modify and transaction-consistent revision information deliver consistence and up-to-date information without the time-lag typical of many search engines.
- ➔ True search term highlighting (exactly what the query found, structure etc.) including Adobe Acrobat PDF Highlighting.
- ➔ Extendable/Embeddable/Programmable: Java, Python, Tcl, C++ and other other language APIs.
- ➔ Support for a number of information retrieval protocols including ISO 23950 / ANSI NISO Z39.50, SRW/U and OpenSearch.
- ➔ Runs on a wide range of hardware and operating systems.
- ➔ Easy to maintain, tiny, scalable and fast. Energy efficient: One can even start off with inexpensive low-power hardware (even embedded boards like NVIDIA’s Jetsons). On small machines we’ve supported several concurrent user sessions searching GB of data and still delivered search performance measured in fractions of a second.
- ➔ Does not demand advance setup or pre-processing.
- ➔ Unlike most search engines it is not based on "Inverted file indexes". Because of the limitation of "inverted indexes" most search engines typically index text (excluding common words and long terms as "stop words") and only a fixed and limited number of (defined at indexing time), additional fields (since they are expensive). Our aim is to provide unlimited query flexibility without having to know in advance what questions users are going to ask.
- ➔ Unlike databases we don’t require conversion into a "common format" with a schema set in concrete.
- ➔ Unlike XML databases we support also non-hierarchical structures and overlap.
- ➔ Unlike search engines we can index all elements, their structure, and their contents. This means that one can quickly evaluate text queries, structural queries, and queries that combine both text, objects (numerical, geospatial etc.) and structural constraints (e.g.,find diagram captions that mention engine in articles whose title contains Airbus).
- ➔ Virtual "indexes" allow for the design of logically segmented information indexes and fast on-demand search of arbitrary combinations thereof. Via the field and path mapping architecture this can be implemented completely transparent to search.

- ➔ Index collection binding: multiple indexes can be imported into an index. This allows for the custom creation of indexes on the basis of a large catalog of indexes— highly relevant to publishers as their customers tend to subscribe to only a sub-set of products (e.g. journals).
- ➔ Full ability to search specific structure/context in information without even knowing their details (such as tag or field names).
- ➔ User defined "search time" unit of retrieval: the structure of documents can be exploited to identify which document elements (such as the appropriate chapter or page) to retrieve. No need for intermediate documents or re-indexing.
- ➔ No need for a “middle layer” of content manipulation code. Instead of getting URLs from a search engine, fetching documents, parsing them, and navigating the DOMs to find required elements, it lets you simply request the elements you need and they are returned directly.
- ➔ "Any-to-Any" architecture: On-the-fly XML and other formats.

The default modus is to index all the words and all the structure of documents. It provides powerful and fast search without prior knowledge about the content yet enables arbitrarily complex questions across all the content and from different perspectives. Not bound by the constraints of "records" as unit of information, one can immediately derive value from content with the flexibility to enhance content and the application incrementally over time without "breaking anything".

Semantic Revelation

The engine has been designed to allow for the implementation of a novel remarkably powerful yet simple paradigm of retrieval that we’ve called "semantic revelation". The basic idea is that clusters of associations define their own implicit semantics for terms. Folksonomies (the current fashion of social tagging) assume everyone is speaking the same language with the same shared background. They don't. Everything is NOT Miscellaneous (as David Weinberger suggests). Words derive their commonality in meaning from those that associate with one another. We don't know what the words mean but might assume that if people are talking to one another that they have shared semantics. That's the basics of normative communication.

John Searle introduced the notion of an 'indirect speech act' as a kind of indirect 'illocutionary' act: "In indirect speech acts the speaker communicates to the hearer more than he actually says by way of relying on their mutually shared background information, both linguistic and nonlinguistic, together with the general powers of rationality and inference on the part of the hearer." Our model goes backwards.. It first asks.. "Who is talking to each other". And "who do I want to talk with". Its guilt by association.

The point is: We don't care what a word or sentence means. We just "assume" that when people talk they understand, more or less, each other but neither is everyone talking with each another nor do they even want to..

When there is no dialogue between listeners, the same words and phrases as spoken by a speaker can mean different things. Search is also searching for dialogue. Its all, of course, not de-coupled from ranking..

This approach increases the accessibility and visibility of information in the Internet and at the same time allows for

people to individually define their own thresholds for what kinds of conversations that they don't want to see. Search after all is about helping people find what they are looking for AND NOT what they don't want to find!

See also: the collaborative works on Multipolar Search: EXODVS Cross Search Presentation @ ISEA 2008. National Museum of Singapore.

Unit of Retrieval

In "traditional" search engine models there is a standard unit of the record. Its the unit of index (the page, PDF, Word document) and retrieval. By contrast we have a user defined "search time" unit of retrieval: the structure of documents is exploited to identify which document elements (such as the appropriate chapter or page) to retrieve. Retrieval granularity may be on the level of sub-structures of a given document or page such as line, paragraph but may also be as part of a larger collection.

The domain of search is information and this may be a relevant part of a document or a collection of relevant documents (such as a Journal, Newspaper, Encyclopedia, Social Network etc.).

III. Hardware & OS Prerequisites

The re-Isearch engine is developed in a simplified version of C++. It is intended to be compile-able on the widest possible range of hardware, operating systems and compilers. It is also designed to run, if needed, in a comparatively small memory footprint (previous version have run on 32-bit machines with as little as 8 MB physical RAM[1]) making it suitable for appliances. It has also been designed to try to impose a minimal computing impact on the host. Rather than run multiple threads and a high CPU workload it's strategy is to be fast but not at the cost of other processes, heat or increased energy consumption.

Within this design, the limiting factor is I/O. Performance is related more to memory and storage throughput than CPU speed. Fast SSDs (SAS SSDs and for those with external disks NVMe units connected via USB versions USB 3.1 rev 2 or Thunderbolt are at an obvious advantage) are preferable over HDDs. Gigabit interconnect and faster bus sytems like NVLINK over PCIe deliver more than CPU cores. The faster the RAM, the better the I/O bandwidth and the faster the mass storage the better. More RAM memory too delivers a boost since the engine can use it to speed up indexing (and in searching large collections can cache more in memory and avoid swapping). While indexing is more or less sequential, search is random access but using memory mapping.

While a board such as the Raspberry Pi Zero or B might be ill-suited due to their poor I/O performance (typically max. 25 MB/s), the Raspberry 4 with its USB 3.0 interface is already fine for some use cases. Keeping to lower cost ARM based embedded boards[2] the $50 USD NVIDIA Nano is probably a better choice. With its 4 lanes and 5 GB/s interfaces it can get beyond 200 MB/s.

1 By comparison alone a Java runtime requires 128 MB physical RAM to run.
2 In NONMONOTONIC's Munich lab we have a large selection of embededed boards from the tiny NodeMCU to NVIDIA's Xavier.

Handbook 0.9 (work in progress)

Running on a reference Intel i7 notebook (2.90 GHz) using a low cost Samsung T5 USB 3.2 drive (500 MB/sec read/ write) and using for indexing 512MB Memory, we get around 5600k words/min (roughly ½ million emails in under 20 minutes). Indexing full text (without deep parsing) we see speeds as fast as 99000k words/min (17-20x as fast). This all without a noticeable workload. Thunderbolt 4 (already provided on a 2021 Apple hardware) provides up to 40 Gbit/s and some newer drives are already appearing that have read/write rates over 2.5 GB/s.

A typical data center server can easily handle many parallel indexing jobs while concurrently providing search without a hitch and substantially higher I/O throughputs.

While the primary development platform for the re-Isearch engine was Solaris SPARC it was used extensively on x86, MIPS, PowerPC, Alpha, PA-Risc and a number of Unix operating systems (Solaris, Scientific Linux, IRIX, AIX, Tru64, HP-UX) and Microsoft's Windows (pre-Windows 10). Re-Isearch, by contrast, has as primary development platform x86 Ubuntu with ARM based Ubuntu (currently NVIDIA) as 2nd platform. It should compile and run without issues on other x86 and ARM based Unix and Linux operating systems.

Compatibility support code for Win32/64 API is included but is, at this time, not supported. Official support for Windows (ARM and x86), Android (ARM) , Apple IOS (ARM) and iPadOS are on the road-map. An experimental fork for HPC clusters too are in planning.

✗ File Systems

Given the want for portability and to be able to transfer indexes between machines (or via distributed file systems like AFS) we selected to use many compact (dense) files rather than single files with "holes" (which when copied are large and sparse). This design decision pushes the organization of reading and writing to the file system. Under Unix, especially Linux, one might want to consider alternative file systems (or non-standard configurations) to improve performance.

The use of file systems such as XFS is strongly advised against. Good choices range for use on external SSDs range from F2FS (Flash-Friendly File System) to exFAT (its main disadvantage is that it is proprietary and even the reverse engineered versions have run afoul of Microsoft patents).

File System	**Max. number of files**
F2FS	Depends on volume size
exFAT	2,796,202 per directory
ext2, ext3	32K per directory
ext4	64K per directory (by default)
Reiser4	Unlimited
NTFS	4,294,967,295 (Total folder = Total disk)

Ext4 is not a bad file system (and the 64K limit can be lifted) but journals are counter-productive for indexes.

Handbook 0.9 (work in progress)

While journals offer data consistency for unexpected system crashes or power losses they also suffer from performance decrease due to the extra journal writes. Generally indexes are easily reconstructed so have little need for the level of data consistency afforded.

If one want to use ext4 on an external SSD for indexing it is advised that one disable journaling. Assuming that the SSD is /*dev*/sda2 mounted on /*media*/ssd:

i. unmount /*media*/ssd
   Unmount the disk. Note that in general it is not safe to run e2fsck on mounted filesystems.
ii. tune4fs -O ^has_journal /dev/sda2
   Turn off the journal feature
iii. e4fsck –f /dev/sda2
   Optionally check the filesystem to confirm its integrity
iv. reboot
v. dmesg | grep EXT4
   Check the messages to confirm that its without journaling

On Apple platforms (IpadOS, MacOS, IOS) we have alongside Ex-FAT, FAT-32, HSF+, and APFS for external drives—2021 models with Thunderbolt 4.

✗ Hardware and OS Checklist:

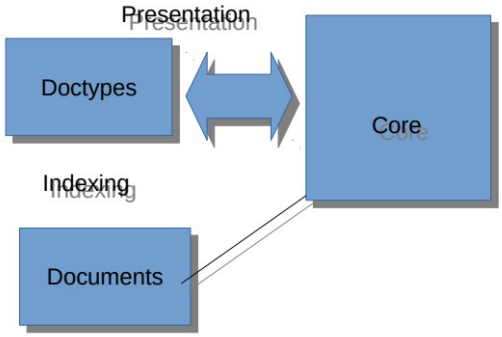
- ✔ 32 or 64-bit POSIX Unix/Linux.
- ✔ Min. 8 MB Physical RAM.
- ✔ Virtual Memory System activated.
- ✔ Choose a faster file system configured for the use case
- ✔ If possible choose faster memory, fast disks, preferably SSDs

IV. Design

The **Core** engine contains all the classes and methods (C++) to ingest documents, parse them, create indexes, store objects and provide search thereto. On a higher metalevel we have the engine kernel which provides the core indexing, search, retrieval and presentation services, manages objects and dispatches to handlers.

Datatypes (object data types handled polymorphic to text)

Data types are handled by the core and extendable within code. Many of these are well known from standard schemas such as string, boolean, numerical, computed, date and many more motivated by user needs such as phonetic types. Milestone 1 (CORE) shall contain a large assortment including the possibility to install a callback and some local types. A list and some



documentation of available types is available at runtime as the system contains a rudimentary self documentation of the installed handlers.

Data-type name	Description
string	String (full text)
numerical	Numerical IEEE floating
computed	Computed Numerical
range	Range of Numbers
date	Date/Time in any of a large number of well defined formats including common ISO, W3 and IETF formats.
date-range	Range of Date as Start/End but also +N Seconds (to Years)
box	Geospatial bounding box coordinates (N,W,S,E): RECT{c0 c1 c2 c3}
gpoly	Geospatial n-ary coordinates as list of vertices.
time	Numeric computed value for seconds since 1970, used as date.
ttl	Numeric computed value for time-to-live in seconds.
expires	Numeric computed ttl value as date of expiration
boolean	Boolean type. 1, 0, True, False, Yes, No, Ja, Nein, Ein, Aus, On, Off, Oui, No, Tack ...
currency	Monetary currency
dotnumber	Dot number (Internet v4/v6 Addresses, UIDs etc)
phonetic	Computed phonetic hash applied to each word (for names)
phone2	Phonetic hash applied to the whole field
metaphone	Metaphone hash applied to each word (for names)
metaphone2	Metaphone hash (whole field)
hash	Computed 64-bit hash of field contents
casehash	Computed case-independent hash of text field contents
lexi	Computed case-independent lexical hash (first 8 characters)
privhash	Undefined Private Hash (callback)
isbn	ISBN: International Standard Book Number
telnumber	ISO/CCITT/UIT Telephone Number
creditcard	Credit Card Number
iban	re-IsearchAN: International Bank Account Number
bic	BIC : International Identifier Code (SWIFT)
db_string	External DB String (callback)
callback	Local callback 0 (External)
Local1 – local7	Local callback 1 - 7 (External)
special	Special text (reserved)

Datatypes may be set in the .ini (see sections below) or in some document types such as XML explicitly in the document, e.g. <DATE type="date"> … </DATE> to define an element DATE as type "date". In this light we support a number of synonyms for the datatypes as known in XML schemas.

xs:string	Alias of string
xs:normalizedString	Alias of string
xs:boolean //	Alias of boolean
xs:decimal	Alias of numerical
xs:integer	Alias of numerical
xs:long	Alias of numerical
xs:int	Alias of numerical
xs:short	Alias of numerical
xs:unsignedLong	Alias of numerical
xs:unsignedInt	Alias of numerical
xs:unsignedShort	Alias of numerical
xs:positiveInteger	Alias of numerical
xs:nonNegativeInteger	Alias of numerical
xs:negativeInteger	Alias of numerical
xs:positiveInteger	Alias of numerical
xs:dateTime	Alias of date
xs:time	Alias of time

Once an element is associated with a datatype it is assumed all instances of that element contain the same datatype. Should the indexer encounter a mismatch it will generally issue a warning message.

✗ Notes on the DATE datatype:

The date datatype parser and search algorithms are probably the least straightfoward types in the engine. This is to a great extent due to the large range of date formats and semantics in widespread use.

- The date parser for long date names understand only the following languages: English, French, German (and Austrian variants), Spanish, Italian and Polish. Adding additional languages is straightfoward: see date.cxx
- Valid dates included are also the national numerical only (non-ISO) standards using the '-'. '.' and '/' styles of notation. Dates are either intrinsically resolved or should they be ambiguous using the locale. In a date specified as 03/28/2019, its clear that the 28 refers to day of the month, while an expression such as 03-03-23 is ambiguous. Sometimes the the use of a dash, slash or period has a semantic difference but sometimes they are used interchangeably. In Sweden for instance the '-' notation is generally used as Year Month Day (e.g. 99-12-31) while in the US it is written as Month Day Year and in much of the European continent its Day Month Year. In Britain both Month Day Year and from the end of the 19th Century Day Month Year are, aligning with the Continent, commonly encountered. In the UK, in fact, all of the following forms 31/12/99, 31.12.99, 31.xii.99 and 1999-12-31 are encountered. With years in YYYY notation or with NN > 31 its clear that it can't be day of the month just as NN > 12 can't be the number of the month.

- In two digit YY specified years the year is resolved as following:

Current Year Last Two Digits	Two Digit Year Specified	Year RR Format Returns
0-49	0-49	Current Century
50-99	0-49	One Century after current
0-49	50-99	One Century before current
50-99	50-99	Current Century

Unix Model: Year starts at 70 for 1970

00-68	2000-2068
69-99	1969-1999

- The parser understands precision of day, month, year
- The parser understands BC and BCE dates, e.g. “12th century b.c.”

The parser supports number of special reserved terms for dates such as "Today", "Now", "Yesterday", "Last Week", "Past Month", "14 days ago" etc. Inputs such as 2/10/2010 are ambiguous and should they be desired added as local formats (See next section).

Relative dates controlled vocabulary

“Today” := the LOCAL date (of the server) without time (day precision).
“Yester[day|week|month|year]” := the past X (X precision) from the point of view of the local date/time where X is one of day, week, month or year (e.g. Yesterday is the day before today in day precision)

“Tomorrow” := the day after today (day precision)
"This day" := Today
"This Month" := the current month (LOCAL) in month precision
"This year" :? the current year (LOCAL) in year precision.
Now or Present -- the date/time at the moment the word is parsed.
[Last|Past] [NNN] Sec[onds]|Min[utes]|H[ours]|Day[s]|Week[s]|Month[s]|Year[s]|Decade[s]|Millennium [Ago|Past]
Last|Past&nsp;Sun[day]|Mon[day]|Tue[sday]|Wed[nesday]|Thu[rsday]|Fri[day]|Sat[urday]

Examples: "*14 hours ago*", "*6 days past*", "*Last year*", "*Past month*", "*200 minutes ago*", "*Last Monday*"

Note: "Yesterday" is date precision while "Past Day" is date/time precision. The two also might refer to two different days due to the time difference between local time (from the perspective of the server) and UTC/GMT: "Past Day" is based upon the date/time as per UTC/GMT while Today, Yesterday and Tomorrow are based upon "local" time. Note also that "14 days ago" defines a date as well as method of comparison based upon day precision---thus something different from 335 hours ago or 2 weeks ago. Last year, resp. month etc., each define a precision of year, resp. month etc.

The prefix "*End of*" is interpreted as the end of the period.

The sequel in date ranges: NNNNs (as in "*1950s*") NNNN century (as in "*19th century*", "*19 Jh.*" etc) Past|Last [NNN] Seconds|Minutes|Hours|Days|Weeks|Months|Years|Decades|Millennium
Last|Past Sun[day]|Mon[day]|Tue[sday]|Wed[nesday]|Thu[rsday]|Fri[day]|Sat[urday]

Examples: "*Past 14 days*", "*Past 24 hours*", "*Past month*", "*Since last Friday*" NOTE: The prefix words "*Within*", "*during*" and "*the*" are skipped, e.g. "*Within the past day*" is reduced to "*Past day*".

Since means from then until now. Since last week means from the start of last week to now. "*Last week*, by contrast, means only the days of last week (Sunday through Sat.). Date ranges, in contrast to dates, are precise to time but start at the beginning and end at the end of their respective unit, e.g. last month starts at midnight of the first and ends just as midnight strikes on the last day of the month (in Oct. its the 31th day).

Local Extensions to handle "ambiguous" national formats

A number of local formats can be easily added via a "datemsk.ini" file. That file is quite simple and uses host-platform STRPTIME(3) function.

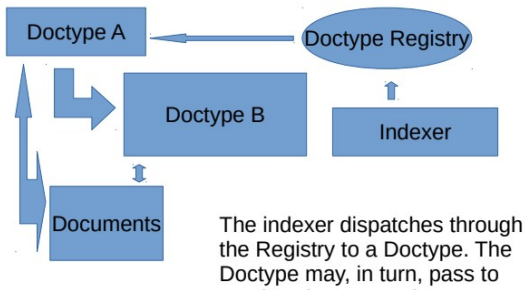
Object Indexes (data type handlers)

The indexer supports a number of datatypes. These are handled by a data type registry. All data is stored as string (the octets defining the words/terms, e.g. xs:string or xs:normalizedString in XML schema jargon) and, if specified or determined, the specific object type. These different data types have their own for-purpose indexing structures—and search algorithms and and semantics for matching as well as relations. Some have also very special functions. The datatype TTL, for example, is a numeric computed value for time-to-live in seconds. The same structures are used for another datatype called „expires“ but with the time-to-live associated with the record to define record expiration. There are also a number of datatypes that use hash structures such as sound or phonetic hashes which have proven useful in name searches. All datatypes carry their own most basic documentation into the registry.

Paths to index items of a specific datatype have the optional :type qualified, e.g. `DATE` is as datatype specifically as date `DATE:date`.

Doctypes (Document handlers)

Services to handle the various document formats (ingest, parse, recognize start and end of records with multi-record file formats, recognize start and end of fields, decode encodings, convert and present) are handled by a so-called “doctype” system. These doctypes are built upon a base DOCTYPE class. They are managed and dispatched by a registry. All doctypes carry their own basic documentation (and tree heirarchy) when they register into the registry. We have both a growing collection of “built-in” types (provided with the core engine and covered by the same license and conditions as the rest of the engine) and “plugin-in” types. The latter are loadable at run-time. These loadable plug-ins can handle all or just partial services and are not just



The indexer dispatches through the Registry to a Doctype. The Doctype may, in turn, pass to another doctype and so on.

descendents of a built-in document handler class but also can pass control to other types not immediately in its class path. Milestone 1 shall contain some 60 formats including several multi functional that transparently use confiigurable external conversion tools—for example OCR to process scanned documents or an image captioning tool to process photographs alongside the embedded metadata is readily implemented. All parsers have some self documentation available at runtime of their options and class tree.

Field Unification (indexing/search)

Since the engine is designed to support a wide range of heterogenous documents and record formats a facility was developed to allow for name, resp. query path alignment. Defined by a configuration file, field names (and paths) can to be mapped during indexing to alternative names. This is intended to handle the issue of semantically equivalent field (or path) contents having syntactically different names (or paths). It also allows one to skip fields (for example P=<Ignore>). These settings may be defined unique to doctype, to doctype instance or to database (e.g. the search indexing target). Field unification can also be used to map elements that the same name and similar semantics but different different datatypes (such as computed versus numerical).

Ranking (search results)

The set of elements (records) on a search response tends to be sorted by scores—but there are a number of other sorting methods available. Scores are normalized by a number of methods (that impact the sort). The standard methods in the core engine are NoNormalization, CosineNormalization (Salton Tf-idf), MaxNormalization (weighted Cosine), LogNormalization (log cosine score), BytesNormalization (normalize frequency with a bias towards terms that are closer to one another) and CosineMetricNormalization.

Scores and ranking can also be boosted by a number of progressions: Linear, Inversion, LogLinear, LogInversion, Exponential, ExpInversion, Power and PowInversion.

Both sorting and normalization algorithms are user extendable and designed for customizations.

Queries can also weight various matches or field results to give them more importance (or less). There are also methods to cluster or shift position in results ranking with hits (matches) that are closer to one another („magnitism“).

Presentation (retrieval)

For presentation the engine uses something called “Record Syntax” to define the response syntax either through reconstruction, reconstitution or transformation. Like datatypes and doctypes it too is handled by an extensible registry. These are defined by internally registered OIDs. A ITU-T / ISO/IEC object identifier (OID) is an extensively used identification mechanism. It is the job of the DOCTYPE subsystem (using the doctype associated with the record whose data the response contains either partially or in full) to build the appropriate reponse as requested. It is typically built as a reconstruction using the indexed structure and addresses of content. This allows one to control the final reconstruction to exclude sensitive information that might have been in the original record but to be excluded from some presentations.

V. Indexing

There are many options and hyperparameters available to indexing. One can create applications using either the native C++ interface, one of the language bindings (such as Python) or via the Iindex command line utility. The simplist way to index files is with the Iindex command line tool. It offers a host of options.

The most minimal run is something like:

```
Iindex -d MYDB file1 file2 file3
```

Here the indexer will create a MYDB index (should an index already exist it will be erased) and add file1 file2 file3 to it. Since the default document handler is AUTODETECT it tends to be able to guess a suitable handler.

The -d option sets the database path/name. In the above example it will create a number of files MYDB.xxx in the current working directory. If one wanted the index to do into /var/opt/index/MYDB one would use just that:

```
Index -d  /var/opt/index/MYDB  ….
```

Say you want to index all the HTML files in /var/opt/web, including sub-directories, using the HTMLHEAD document handler:

```
Iindex -d MYDB -t HTMLHEAD -name “*.htm*”  /var/opt/web
```

To control the files to be indexed there are a wealth of options:

```
-f list            // File containing list of filenames; - for stdin.
-recursive         // Recursively descend subdirectories.
-follow            // Follow links.
-include pattern   // Include files matching pattern.
-exclude pattern   // Exclude files matching pattern.
-incldir pattern   // Include dirs matching pattern.
-excldir pattern   // Exclude dirs matching pattern.
-name pattern      // Like -recursive -include.
                   // pattern is processed using Unix "glob" conventions:
                   // * Matches any sequence of zero or more characters.
                   // ? Matches any single character, [chars] matches any single
                   // character in chars, a-b means characters between a and b.
                   // {a,b...} matches any of the strings a, b etc.
                   // -include, -incldir, -excldir and -name may be specified multiple
                   // times (including is OR'd and excluding is AND'd).
```

✗ Index Combination

Indexes can be combined by two means:

- Import: the import of one db into another
- Virtualization: a map that defines what dbs to use for search

Import

Imagine we have two indexes: db1 and db2. One contains file1, file2, file3 and the other contains file4, file5 and file6.
By importing db2 into db1 we get a db that is as-if it had indexed file1, file2, file3, file4, file5 and file6.
The simplest means to import dbs into other is with the Iutil tool:

```
-import (X)    // Import database: (X) as the root name for imported db files.
```

In the above example:

```
Iutil -d db1 -import db2
```

Performance searching the combined database is no different than if a single database was initially created rather than two. If db1 has n unique words and db2 has m unique words the big O performance is typically O(ln (n+m)),

Virtualization

Here instead of importing the data from one database into another we just create a virtual map of databases to search. Search is linear across each database. Performance is $\sum Oi \; where \; Oi \; is \; the \; performance \; of \; the \; ith \; database$—*typically* $Oi \; is$ *O(ln n) where n is the number of unique words in the index.* As one can see import (physically importing and merging one database into another) provides faster search than virtualization. With two databases db1 and db2 with respectively n and m unique words the typical big O performance is O(ln(n) + ln(m)).

The overwhelming advantage of virtualization is the speed and ease of their creation as they are defined by a single plain text configuration file. They can be created on the fly and disposed of at will.

There are two ways: either setting the correct values in a db.ini file or by providing a db.vdb file

In an .ini

```
[DbInfo]
Collections=<List of virtual databases>
Databases=<List of physical databases>
vdb-file=<Path to file list> (default: <database>.vdb) # File has 1 entry per line
```

A db.vdb file is just a line (one per line) of paths to db.ini files.

✗ Index organization

An index is organized in a number of files

Extension	Contents
.ini	The database configuration. MS style profile.
.vdb	Definition of a virtual DB (out of multiple DBs)
.cwi	Common words (this is an information created during the indexing process)
.cat	Metadata Record file catalog (key, value)
.dfd	Database field data. This contains a table of fields and their associated datatypes
.inx	The index. This contains addresses
.mdt	The table associating record to key, address (in .mds) etc.
.mds	The mapping between address and files on the data system.
.sis	SIS cache. A file that contains a cache of prefixed terms (n octects)

.inf	Generated Metadata file (describes a resource)
.path	Meta file with linkage to source (used by externally processed documents/records)
.syn	External Synonyms
.spx	External Synonyms Parents
.scx	External Synonyms Children
.001 - .9zz	Field (path address data). NOTE: By default to keep to both 9.3 and case independent file names (to maintain compatibility with other operating systems) we have as default a max. of 12959 fields (paths). This limit could be easily increased.
As above but with a suffix to indicate the data type, e.g. .001d indicates an index of type “date”.	n Numeric r Num Range d Date e Date Range p Poly b Box h Phonetic hash c 64-bit/Numeric hash t Telephone num v Credit Card Number (Visa etc.) i re-IsearchAN (includes Checksum)

NOTE: Given the want for portability and to be able to transfer indexes between machines (or via distributed file systems like AFS) we selected to use many compact files rather than single files with “holes” (which when copied are large and sparse). This design decision pushes the organization of reading and writing to the file system.

Other files:

Extension	**Contents**
.iq1	Pass 1 queue (file queue)
.iq2	Pass 2 queue (record queue)
.mdk	MDT Key Index
.dbi	Database info
.sta	DB status
.gils	Centroid file (optionally generated)
.glz	Compressed Centroid file
.rca	Results Cache

When files are added to the indexer they get added to index queue#1 (.iq1). These files are read and the doctype parser segments it into records. This list of records is written to the index queue#2 (.iq2). The 2nd queue (.iq2) is read and the records are indexed. The reason for this is that a single file may contain many records and these records need not be of

the same document type. Typically the work to pass from queue#1 to queue#2 is relatively minor. Parsing of record structure, fields, data types etc. is part of the processing associated with queue#2.

✗ Indexing utility:

The standard indexing utility is called Iindex. It takes a number of options:

```
re-Isearch indexer 2.20210608.3.8a 64-bit edition (x86_64 GNU/Linux)
(C)opyright 1995-2011 BSn/Munich; 2011-2020 NONMONOTONIC Networks; 2020,2021 re.Isearch
Project.
This software has been made available through a grant 2020/21 from NLnet Foundation and EU
NGI0.

Usage is: Iindex [-d db] [options] [file...]
Options:
 -d db              // Use database db.
 -setuid X          // Run under user-id/name X (when allowed).
 -setgid X          // Run under group-id/name X (when allowed).
 -cd X              // Change working directory to X before indexing.
 -thes source       // Compile search thesaurus from file source.
 -T Title           // Set Title as database title.
 -R Rights          // Set Rights as Copyright statement.
 -C Comment         // Set Comment as comment statement.
 -mem NN            // Advise min. of NN RAM.
 -memory NN         // Force min. of NN RAM.
                    // Note: Specifying more memory than available process RAM can
                    // have a detrimental effect on performance.
 -relative_paths    // Use relative paths (relative to index path).
 -base path         // Specify a base path for relative paths.
 -rel               // Use relative paths and assume relation between index location
                    // and files remains constant.
 -absolute_paths    // Make file paths absolute (default).
 -ds NN             // Set the sis block to NN (max 64).
 -mdt NN            // Advise NN records for MDT.
 -common NN         // Set common words threshold at NN.
 -sep sep           // Use C-style sep as record separator.
 -s sep             // Same as -sep but don't escape (literal).
 -xsep sep          // Use C-style sep as record separator but ignore sep.
 -start NN          // Start from pos NN in file (0 is start).
 -end nn            // End at pos nn (negative to specify bytes from end-of-file).
 -override          // Override Keys: Mark older key duplicates as deleted (default).
 -no-override       // Don't override keys.
 -centroid          // Create centroid.
 -t [name:]class[:] // Use document type class to process documents.
 -charset X         // Use charset X (e.g. Latin-1, iso-8859-1, iso-8859-2,..).
 -lang ISO          // Set the language (ISO names) for the records.
                    // Specify help for a list of registered languages.
 -locale X          // Use locale X (e.g. de, de_CH, pl_PL, ...)
                    // These set both -charset and -lang above.
                    // Specify help for a list of registered locales.
```

```
-stop              // Use stoplist during index (default is none)
-l name            // Use stoplist file name; - for "builtin".
-f list            // File containing list of filenames; - for stdin.
-recursive         // Recursively descend subdirectories.
-follow            // Follow links.
-include pattern   // Include files matching pattern.
-exclude pattern   // Exclude files matching pattern.
-incldir pattern   // Include dirs matching pattern.
-excldir pattern   // Exclude dirs matching pattern.
-name pattern      // Like -recursive -include.
                   // pattern is processed using Unix "glob" conventions:
                   // * Matches any sequence of zero or more characters.
                   // ? Matches any single character, [chars] matches any single
                   // character in chars, a-b means characters between a and b.
                   // {a,b...} matches any of the strings a, b etc.
                   // -include, -incldir, -excldir and -name may be specified multiple
                   // times (including is OR'd and excluding is AND'd).
-r                 // -recursive shortcut: used with -t to auto-set -name.
-o opt=value       // Document type class specific option.
                   // Generic: HTTP_SERVER, WWW_ROOT, MIRROR_ROOT, PluginsPath
-log file          // Log messages to file; <stderr> (or -) for stderr, <stdout>
                   // for stdout or <syslog[N]> for syslog facility using LOG_LOCALN when
                   // N is 0-7, if N is 'D' then use LOG_DAEMON, and 'U' then LOG_USER.
-e file            // like -log above but only log errors.
-syslog facility   // Define an alternative facility (default is LOG_LOCAL2) for <syslog>
                   // where facility is LOG_AUTH, LOG_CRON, LOG_DAEMON, LOG_KERN,
                   //  LOG_LOCALN (N is 0-7), etc.
-level NN          // Set message mask to NN (0-255) where the mask is defined as (Ord):
                   // PANIC (1), FATAL (2), ERROR (4), ERRNO (8), WARN (16), NOTICE (32),
                   // INFO (64), DEBUG (128).
-quiet             // Only important messages and notices.
-silent            // Silence messages (same as -level 0).
-verbose           // Verbose messages.
-maxsize NNNN      // Set Maximum Record Size (ignore larger)[-1 for unlimited].
-nomax             // Allow for records limited only by system resources [-maxsize -1].
-a                 // (Fast) append to database.
-O                 // Optimize in max. RAM. (-optimize -mem -1)
-Z                 // Optimize in max. RAM but minimize disk (-merge -mem -1)
-fast              // Fast Index (No Merging).
-optimize          // Merge sub-indexes (Optimize)
-merge             // Merge sub-indexes during indexing
-collapse          // Collapse last two database indexes.
-append            // Add and merge (like -a -optimize)
-incr              // Incremental Append
-qsort B[entley]|S[edgewick] // Which variation of Qsort to use
-copyright         // Print the copyright statement.
-version           // Print Indexer version.
-api               // Print API Shared libs version.
-capacities        // Print capacities.
-kernel            // Print O/S kernel capacities.
-help              // Print options (this list).
```

```
 -help d[octypes]   // Print the doctype classes list (same as -thelp)
 -help l[ang]       // Print the language help (same as -lang help)
 -help l[ocale]     // Print the locale list (same as -locale help)
 -help o[ptions]    // Print the options (db.ini) help.
 -help t[types]     // Print the currently supported data types.
 -thelp             // Show available doctype base classes.
 -thelp XX          // Show Help for doctype class XX.
 -xhelp             // Show the information Web.

NOTE: Default "database.ini" configurations may be stored in _default.ini
in the configuration locations.
Options are set in section [DbInfo] as:
   Option[<i>]= # where <i> is an integer counting from 1 to 1024
Example:  Option[1]=WWW_ROOT=/var/httpd/htdocs/
```

An existing index can be appended to (and optimized) without interrupting search. While during the index phase many of the new records are not yet available the existing records (and whose addition has been completed) are available.

✗ Help Subsystem:

The system provides a large amount of tense information about the type system, available document type handlers and their options. While this and other documentation might attempt to be up-to-date the absolute up-to-date documentation is embedded into the software itself. This is especially important for the doctype and datatype registries.

The command line tool Iindex includes among it many functions a simple command line access to much of this information.

```
Iindex -help
```

provides the usage including a list of help options:

```
 -help              // Print options (this list).
 -help d[octypes]   // Print the doctype classes list (same as -thelp)
 -help l[ang]       // Print the language help (same as -lang help)
 -help l[ocale]     // Print the locale list (same as -locale help)
 -help o[ptions]    // Print the options (db.ini) help.
 -help t[types]     // Print the currently supported data types.
 -thelp             // Show available doctype base classes.
 -thelp XX          // Show Help for doctype class XX.
 -xhelp             // Show the information Web.
```

As well as a number of information options

```
 -copyright         // Print the copyright statement.
 -version           // Print Indexer version.
 -api               // Print API Shared libs version.
 -capacities        // Print capacities.
 -kernel            // Print O/S kernel capacities.

Iindex -copyright
Portions Copyright (c) 1995 MCNC/CNDIR; 1995-2011 BSn/Munich and its NONMONOTONIC Lab;
1995-2000 Archie Warnock; and a host of other contributors;
```

Handbook 0.9 (work in progress)

This software has been made available through a grant 2020/21 from NLnet Foundation and EU NGI0.

```
Iindex -api
API 4.0a/6  built with: g++-10 (Ubuntu 10.1.0-2ubuntu1~18.04) 10.1.0 (x86_64 GNU/Linux)

Iindex -capacities
Physical Index Capacities for this 64-bit edition:
  Max Input:   16384 Tbytes (Total of all files)
  Max Words:   1099511 trillion (optimized), >8796093 trillion (non-optimized)
  Max Unique:  120494 trillion words (optimized), >8796093 trillion (non-optimized)
  Word Freq:   Maximum same as "Max Words"
  Max Records: 12 million.
  Min. disk requirements: some fixed and variable amounts plus each
    record (160); word (8); unique word (~45); field (16).
Virtual Database Search Capacities:
  Max Input:   aprox. 4.1943e+06 Terabyte(s)
  Max Words:   unlimited (limit only imposed by physical index)
  Max Unique:  unlimited (limit only imposed by physical index)
  Max Indexes: 255
  Max Records: 31 million (Total all indexes)
Preset Per-Record Limits:
  Max Key:     36 characters
  Max Doctype Child Names: 15 characters
```

The above capacities are just theoretical and for the largest 64-bit configuration. Most of our 64-bit kernels can’t address the entire 64-bit space. Preset per-record limits, on the other hand, are arbitrary and set hardwired and should the need arise may be extended—or even shrunk. While in our many years we have not found a use case yet demanding keys beyond 36 characters in length we don’t exclude that they may be warranted.

The read should notice that the total number of records in a physical index has been limited to 12 million. This was a design choice to fit within an address space.

While it may be possible to increase the number of sub-indexes beyond 255 it is strongly advised against as Big O search performance is:

```
       O(m*log n)  =>  worse and average case
       O(m)        =>  Best case
```

where m is the number of sub-indexes and n is the number of unique words. Observe also that irrespective of the number of sub-indexes the total number of records is limited to 31 million. To put that number into perspective: the 14 million books in the Bibliothèque nationale de France, 15.5 million book in the German National Library and even the 25 million books in the British Library could fit each into a single search target— almost sufficient to fit the 39 million cataloged books in the US Library of Congress. Segmenting, however, by language or type the capacity is more than sufficient to handle the collections of any library including the 200 million items of the British Library or the 170 million in the Library of Congress.

```
Iindex -kernel
Max file handles: 1024
Max file streams: 1020
```

To get the list of available document handlers one uses the option -thelp:

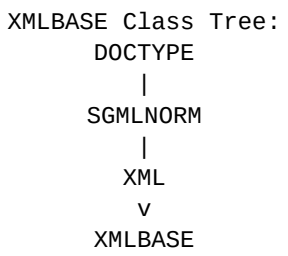
```
Iindex -thelp
```

```
Available Built-in Document Base Classes (v28.5):
        AOLLIST           ATOM     AUTODETECT       BIBCOLON
         BIBTEX         BINARY            CAP       COLONDOC
       COLONGRP       DIALOG-B            DIF        DVBLINE
        ENDNOTE      EUROMEDIA       FILMLINE    FILTER2HTML
    FILTER2MEMO    FILTER2TEXT     FILTER2XML      FIRSTLINE
            FTP           GILS        GILSXML        HARVEST
           HTML          HTML--      HTMLCACHE       HTMLHEAD
       HTMLMETA     HTMLREMOTE       HTMLZERO        IAFADOC
       IKNOWDOC          IRLIST        ISOTEIA     LISTDIGEST
     MAILDIGEST     MAILFOLDER        MEDLINE           MEMO
        METADOC       MISMEDIA     NEWSFOLDER         NEWSML
            OCR            ODT        ONELINE       OZSEARCH
        PAPYRUS           PARA            PDF      PLAINTEXT
             PS          PTEXT            RDF       REFERBIB
            RIS        ROADS++         RSS.9x           RSS1
           RSS2     RSSARCHIVE        RSSCORE           SGML
       SGMLNORM        SGMLTAG         SIMPLE           SOIF
         TSLDOC            TSV        XBINARY        XFILTER
            XML        XMLBASE        XPANDOC      YAHOOLIST
External Base Classes ("Plugin Doctypes"):
  RTF:               // "Rich Text Format" (RTF) Plugin
  ODT:               // "OASIS Open Document Format Text" (ODT) Plugin
  ESTAT:             // EUROSTAT CSL Plugin
  MSOFFICE:          // M$ Office OOXML Plugin
  USPAT:             // US Patents (Green Book)
  ADOBE_PDF:         // Adobe PDF Plugin
  MSOLE:             // M$ OLE type detector Plugin
  MSEXCEL:           // M$ Excel (XLS) Plugin
  MSRTF:             // M$ RTF (Rich Text Format) Plugin [XML]
  NULL:              // Empty plugin
  MSWORD:            // M$ Word Plugin
  PDFDOC:            // OLD Adobe PDF Plugin
  TEXT:              // Plain Text Plugin
  ISOTEIA:           // ISOTEIA project (GILS Metadata) XML format locator records



Format Documentation: http://www.nonmonotonic.net/re:search/[DOCTYPE].html
  Example: http://www.nonmonotonic.net/re:search/MAILFOLDER.html

Usage Examples:
  Iindex -d POETRY *.doc *.txt
  Iindex -d SITE -t MYHTML:HTMLHEAD -r /var/html-data
  find /public/htdocs -name "*.html" -print | Iindex-d WEB -t HTMLHEAD -f -
  Iindex -d DVB -include "*.dvb" -locale de_DE -recursive /var/spool/DVB
  Iindex -d WEB -name "*.[hH][tT][mM]*" -excldir SCCS /var/spool/mirror
```

Each doctype has its own documentation. It is available via -thelp DOCTYPE

```
Iindex -thelp XMLBASE
"XMLBASE": XML-like record format for with special handling for heirarchical
fields (example: for <a><b><c> defines a field a\b\c)

Index options:
  [XML]
  TagLevelSeperator=<character>  # default '\'
```

```
alternatively in the [<Doctype>] section of <db>.ini.
NOTE: Root level tags then get a trailed character. Example:
LOCATOR\
LOCATOR\AVAILABILITY
AVAILABILITY
would be produced from <LOCATOR>..<AVAILABILITY>...</AVAILABILITY></LOCATOR>
```



Every document type has its own information.

To get the available datatypes its -help t

```
Iindex -help t
The following fundamental data types are currently supported (v.37.17):
   string    // String (full text)
   numerical        // Numerical IEEE floating
   computed  // Computed Numerical
   range     // Range of Numerbers
   date      // Date/Time in any of a large number of well defined formats
   date-range       // Range of Date as Start/End but also +N Seconds (to Years)
   gpoly     // Geospatial n-ary bounding coordinates
   box       // Geospatial bounding box coordinates (N,W,S,E)
   time      // Numeric computed value for seconds since 1970, used as date.
   ttl       // Numeric computed value for time-to-live in seconds.
   expires   // Numeric computed ttl value as date of expiration.
   boolean   // Boolean type
   currency  // Monetary currency
   dotnumber        // Dot number (Internet v4/v6 Addresses, UIDs etc)
   phonetic  // Computed phonetic hash applied to each word (for names)
   phone2    // Phonetic hash applied to the whole field
   metaphone        // Metaphone hash applied to each word (for names)
   metaphone2       // Metaphone hash (whole field)
   hash      // Computed 64-bit hash of field contents
   casehash  // Computed case-independent hash of text field contents
   lexi      // Computed case-independent lexical hash (first 8 characters)
   privhash  // Undefined Private Hash (callback)
   isbn      // ISBN: International Standard Book Number
   telnumber        // ISO/CCITT/UIT Telephone Number
   creditcard       // Credit Card Number
   iban      // re-IsearchAN: International Bank Account Number
   bic       // BIC : International Identifier Code (SWIFT)
   db_string        // External DB String (callback)
```

```
   callback  // Local callback 0 (External)
   local1    // Local callback 1 (External)
   local2    // Local callback 2 (External)
   local3    // Local callback 3 (External)
   local4    // Local callback 4 (External)
   local5    // Local callback 5 (External)
   local6    // Local callback 6 (External)
   local7    // Local callback 7 (External)
   special   // Special text (reserved)
They are also available via the following alternative 'compatibility' names:
   text      // Alias of string
   num       // Alias of numerical
   number           // Alias of numerical
   num-range        // Alias of range
   numrange         // Alias of range
   numericalrange         // Alias of range
   numerical-range        // Alias of range
   daterange        // Alias of date-range
   duration         // Alias of date-range
   bounding-box           // Alias of box
   boundingbox      // Alias of box
   phonhash         // Alias of phonetic
   name      // Alias of metaphone
   lastname         // Alias of metaphone2
   hashcase         // Alias of casehash
   hash1     // Alias of privhash
   tel       // Alias of telnumber
   telnum           // Alias of telnumber
   phone     // Alias of telnumber
   telephone        // Alias of telnumber
   inet      // Alias of dotnumber
   ipv4      // Alias of dotnumber
   ipv6      // Alias of dotnumber
   xs:string        // Alias of string
   xs:normalizedString           // Alias of string
   xs:boolean       // Alias of boolean
   xs:decimal       // Alias of numerical
   xs:integer       // Alias of numerical
   xs:long          // Alias of numerical
   xs:int           // Alias of numerical
   xs:short         // Alias of numerical
   xs:unsignedLong        // Alias of numerical
   xs:unsignedInt         // Alias of numerical
   xs:unsignedShort       // Alias of numerical
   xs:positiveInteger     // Alias of numerical
   xs:nonNegativeInteger         // Alias of numerical
   xs:negativeInteger     // Alias of numerical
   xs:positiveInteger     // Alias of numerical
   xs:dateTime      // Alias of date
   xs:time          // Alias of time
Iindex [Fatal]: Usage: No files specified for indexing!
```

Options

The .ini files have a large number of options. The general ones are available via the -help o command

```
Iindex  -help o
Ini file (<database>.ini) options:
Virtual level <database>.ini Options:
[DbInfo]
Collections=<List of virtual databases>
Databases=<List of physical databases>
vdb-file=<Path to file list> (default: <database>.vdb) # File has 1 entry per line

Physical database level <database>.ini Options:
[DbInfo]
Databases=<Path to db stem (Physical Indexes)> # Default: same directory as .ini
BaseDir=<Base Directory> # WARNING: CRITICAL VALUE
useRelativePaths=<bool> # Use relative paths (0 or 1)
AutoDeleteExpired=<bool> # Automatically delete expired records (0 or 1)
MaxRecordSize=nnnn # Max. Record size (bytes) to index (default 51mb).
Headline[/RecordSyntax]=<format of headline>
Summary[/RecordSyntax]=<format of summary>
CacheSize=nnn  # Size of cache
Persist=<bool>  # Should the cache persist?
SearchCutoff=nnn    # Stop searching after nnn hits
MaxRecordsAdvice=nnnn   # Suggest limit for set complements.
CacheDir=<Directory to store cache>
VersionNumber=<Version>
Locale=<Global Locale Name>
Segment=<Short DB Title for use as virtual segment name>
Title=<Database Title> # Complete (exported) title
Comments=<Comments>
Copyright=<Copyright Statement>
StopList=<Language or Path/Format to stopwords list file>
DateCreated=<DateCreated>
DateLastModified=<Date of last modification>
Maintainer.Name=<Name of DB maintainer>
Maintainer.Email=<Email address for maintainer>
PluginsPath=<path to directory where plugins are installed>

[External Sort
<nn>=<path> # <nn> is number and path is to the external sort file
            # if not defined it looks for <DB>.__<nn>

[Ranking]
PriorityFactor=fff.ff   # Priority factor
IndexBoostFactor=fff.ff # Boost score by index position
FreshnessBoostFactor=fff.ff # Boost score by freshness
FreshnessBaseDateLine=date # Date/time, Records newer than this date
# get FreshnessBoostFactor added, older get substracted. The unit
```

```
# of resolution is defined by the precision of the specified date. Default is the date
# specified in DateLastModified [DbInfo] (Minutes resolution)
LongevityBoostFactor=fff.fff # Boost score by difference in days between
# the date created and date modified of the records.

[HTTP]
Pages=<Path to root of htdoc tree>
IP-Name=<ip address of server>
Server=<Server Name, e.g. www.nonmonotonic.com>

[Mirror]
Root=<Path to root of mirror tree>

[<Doctype>/Metadata-Maps]
<Tag>=<TagValue> # TagValue of <Ignore> means Ignore it

Low level index <database>.ini Options:
 [DbSearch]
 MaxTermSearchTime=<nnn>  # Max. time in seconds to search for a term (advice) [default 6].
 MaxSearchTime=<nnn>      # Max. time in seconds (nnn) to run a query (advice) [default 18].
 FindConcatWords=<bool>   # True/False: Search "flow-er"? Not found then "flower".
 PhraseWaterlimit=nnn     # At this point we go into heuristic modus for phrases.
 Freeform=<bool>    # Should we NOT store hits (no proximity etc.)?
 Phonetic=[soundex|metaphone] # Algorithm to use for phonetic term searches.

 [FindConcatWords]
 Force=<bool># Force search of XX-YY-ZZ for XXYYZZ if no match.

 [TermAliases]
 <Term>=<TermAlias> # To map one term to another

 [CommonWords]
 Threshold=nnnn     # Frequency to call common
 Words=<word1> <word2> .... # A list of common words with space seperators

Geospatial RECT{North West South East} [Note the canonical order]
queries need to have their data fields defined via Gpoly-types or:
 [BOUNDING-BOX]
 North=<Numeric Field for North Coordinates [NORTHBC]>
 East= <Numeric Field for East Coordinatesi [EASTBC]>
 West= <Numeric Field for West Coordinatesi [WESTBC]>
 South=<Numeric Field for South Coordinates [SOUTHBC]>

Stopwords are used during search on the basis of STOPLISTS.
STOPLIST can be passed a path or format (see below) or with a language
searches the formats:
       %F/%o.%l
       %B/lib/%o.%l
       %B/conf/%o.%l
       %B/conf/%l/%o
       %B/conf/%os/%l
```

```
       %B/%os/%l
       %B/%os/%o.%l
       /usr/local/lib/%l.%o
       /usr/local/lib/%o.%l
Where:
       %B      the pathname of the base of the package (e.g. )
       %F      the pathname of the library (e.g./home/edz/ib/ib2/lib/)
       %l      the locale (eg. de.iso-8859-1)
       %o      the object (eg.stoplist)
       %<x>    the <x> character (e.g. ``%%'' results in ``%''

NOTE: .INI files may contain other .ini files via an include directive:
#include <path>     # alternative include "path" (may be .ini or XML/SGML format)



Doctype.ini options may also be embeded into database.ini files as:
[<Doctype>]  # Doctype, e.g. [TEXT]
# Options are those from the <doctype>.ini [General] section <key>=<value>. Example:
FieldType=<file to load for field definitions>  # default <doctype>.fields
DateExpiresField=<Date of record expiration>
# Consult the individual Doctype documentation:  -thelp <doctype>

Note: If the software has NOT been installed in /opt/nonmonotonic please confirm that
you have created either a user "asfadmin" (if you are running ASF) or "ibadmin"
whose HOME directory points to where the software has been installed.
```

Some of the options like title and copyright notice set in the .ini configuration files can be conveniently set using the Iutil command line utility.

```
Iutil, Version 3.8a Jun  8 2021 (x86_64 GNU/Linux)
(C)opyright 1995-2011 BSn/Munich; 2011-2020 NONMONOTONIC Networks; 2020,2021 re.Isearch
Project.
This software has been made available through a grant 2020/21 from NLnet Foundation and EU
NGI0.

Usage is: Iutil [-d db] [options]
Options:
 -d (X)         // Use (X) as the root name for database files.
 -id (X)        // Select document with docid (X).
 -thes (X)      // File (X) contains Thesaurus.
 -import (X)    // Import database: (X) as the root name for imported db files.
 -centroid      // Create centroid.
 -vi            // View summary information about the database/record.
 -vf            // View list of fields defined in the database.
 -v             // View list of documents in the database.
 -mdt           // Dump MDT (debug option).
 -inx           // Dump INX (debug option).
 -check         // Check INX for consistency (WARNING: Slow and I/O expensive!).
 -skip [offset] // Skip offset in above test
 -level NN      // Set message level to NN (0-255).
 -newpaths      // Prompt for new pathnames for files.
```

```
 -relative (Dir)// Relativize all paths with respect to (Dir).
                // "." is current directory, "" is db path.
 -mvdir old=new // Change all paths old to new.
                // Example: /opt=/var will change /opt/main.html to /var/main.html but
                // not change /opt/html/main.html (see -dirmv below to change base of tree)
 -dirmv old=new // Move the base of tree from old to new.
                // Example: /opt=/var will change /opt/html/main.html to /var/html/main.html
 -del key       // Mark individual documents (by key) to be deleted from database.
                // Note: to remove records by file use the Idelete command instead.
 -del_expired   // Mark expired documents as deleted.
 -autodelete    // Set the autodelete expired flag to true.
 -deferdelete   // Set the autodelete expired flag to false.
 -undel key     // Unmark documents (by key) that were marked for deletion.
 -c             // Cleanup database by removing unused data (useful after -del).
 -erase         // Erase the entire database.
 -g (X)         // Use (X) as Stopwords list (language).
 -gl0]          // Clear external stopwords list and use default.
 -gt (X)        // Set (X) as the global document type for the database.
                // Specify X as * to get a list of available doctypes.
 -gt0           // Clear the global document type for the database.
 -server host   // Set the server hostname or IP address.
 -web URL=base  // map base/XXX -> URL/XXX (e.g. http://www.nonmonotonic.com/=/var/data/).
 -mirror ROOT   // Set Mirror root.
 -collapse      // Collapse last two database indexes.
 -optimize      // Optimize database indexes.
 -pcache (X)    // Set presentation cache base directory to (X).
                // Uses X/<DATABASE> for files. (X) must exist and be writable.
 -fill (X)[,..] // Fill the headline cache (not CacheDir must be set beforehand!) for the
                // different record syntaxes (,-list), where (X) is the Record Syntax OID
                // or any of the "shorthand names" HTML, SUTRS, USMARC, XML.
 -clip (NN)     // Set Db Search cut-off default.
 -priority (N.N)// Set priority factor.
 -title (X)     // Set database title to (X).
 -copyright (X) // Set database copyright statement.
 -comments (X)  // Set database comments statement.
 -o (X)         // Document type specific option.

Example:  Iutil -d POETRY -del key1 key2 key3
          Iutil -d LITERATURE -import POETRY
```

VI. Searching

The re-Isearch works with sets. The result of a query is a set called a “result set”. Result sets may be combined and operations performed upon them. The engine supports several query language variants with many binary and unary operators.

Targets

When we search, we search a database (index) also known outside of the context of the engine itself as a “target”. Targets (search databases) are either individual indexes, defined by a <DB>.ini where <DB> is the name of the index or a virtually defined ensemble of indexes.

The engine first tries to reads ".ini" and, if it exists, gets a few fields and a file list. If the file list is empty or the ".ini" file did not exist it attempts to locate a file with extension ".vdb". If that file does not exist, it attempts to open the database normally. If it does exist, it opens that file and assumes there to be a list of database names separated by newline characters. It loads each database listed in the ".vdb" file and subsequent search and present operations are performed on the entire list of databases.

The important group of configuration settings are defined in the “DbInfo” section

```
[DbInfo]
Collections=
Databases=
vdb-fle=
```

Collections and Databases contain a list that is ‘,’ (comma) seperated (e.g. a,b,c). It understands quotations marks and escapes to allow names to include ‘,’. Since filenames with comma characters tend not to be terribly portable their use is advised against. See https://en.wikipedia.org/wiki/Filename#Comparison_of_filename_limitations

When it is not a virtual:

```
Databases=<Path to db stem (Physical Index)>
```

The subtle difference between Collections and Databases is that a “collection” can itself be a virtual database with a collection of databases while we normally assume a “database” to be an index. The heuristic for collections tries to detect if there is a circular definition (a collection that includes something that includes something that includes itself). We need to resolve everything into a unique list of physical indexes to search.

Virtual databases are quite powerful and since they are defined by a single file and can be created on-demand it allows for extremely flexible definitions of searching.

VII. re-Isearch Query Language

The result of a query is a result set. The most primitive set is the set of records that contain a single term (word or phrase). The engine supports also fielded data with extremely powerful and flexible methods. In its most basic form (when performing a search) you may specify a field, or a field path--- If no field is specified it means ANYWHERE in the record.

At the core is RPN and Infix notations:

Infix Notation

Infix notation is the common arithmetic and logical formula notation, in which operators are written infix-style between the operands they act on (e.g. 2 + 2).

Handbook 0.9 (work in progress)

RPN Notation

Reverse Polish notation (RPN), also known as postfix notation, was invented by Australian philosopher and computer scientist Charles Hamblin in the mid-1950s. It is derived from the Polish notation, which was introduced in 1920 by the Polish mathematician Jan Łukasiewicz.

In RPN the operands precede the operator, removing the need for parentheses. For example, the expression 3 * (4 + 7) would be written as `3 4 7 + *`.

Infix Expression	Prefix Expression	RPN (Postfix) Expression
`A + B`	`+ A B`	`A B +`
`A + B * C`	`+ A * B C`	`A B C * +`
`A + B * C + D`	`+ + A * B C D`	`A B C * + D +`
`(A + B) * (C + D)`	`* + A B + C D`	`A B + C D + *`
`A * B + C * D`	`+ * A B * C D`	`A B * C D * +`
`A + B + C + D`	`+ + + A B C D`	`A B + C + D +`

The overwhelming advantage of “Polish notations” is that one has no concern for the precedence of operators whence no need for parenthesis or other notational groupings. In RPN we process from left to right. Values are pushed onto the stack. With operators we pop the stack and perform the operation. Infix expressions are by their very nature much more difficult for computers to evaluate given the need to handle operator precedence. It is also more difficult, once people understand RPN, for humans as many are overwhelmed by operator precedence. The Internet is filled with loads of quizzes such as what the the result of “50 + 50 – 25 * 0 + 2 + 2” or “6 ÷ 2 * (1 +2)”.

Handbook 0.9 (work in progress)

All parts of the query language are case insensitive apart from terms. Fields (paths) are case insensitive. While XML, for example, is case-sensitive, we are ***case insensitive***. Should the same element name have different semantics (generally a very bad practice) by case in the source it needs to be converted (e.g. with a prefix).

Queries to the engine are done by a number of means: *1)* RPN expressions *2)* Infix (algebraic) *3)* Relevant feedback (a reference to a fragment) *4)* so called **smart** plain language queries *5)* C++ *6)* Via a bound language interface in Python or one of the other SWIG supported languages (Go, Guile, Java, Lua, MzScheme, Ocaml, Octave, Perl, PHP, R. Ruby. Scilab. Tcl/Tk).

Smart Queries

Smart queries try to interpret the query if its RPN or Infix or maybe just some terms. The logic for handling just terms is as-if it first searched for a literal expression, if not then tying to find the terms in a common leaf node in a records DOM, if not then AND‘d (Intersection of sets), if not then OR‘d (Union) but reduced to the number of words in the query.

Example: Searching in the collected works of Shakespeare:

(a) `rich water`

It find that there are no phases like “rich water” but in the `The Life of Timon of Athens' it finds that both the words “rich” and “water” are in the same line: “.. And rich: here is a water, look ye..”. The query is confirmed as `"rich" "water" PEER` (see binary operators below)

(b) `hate jews`

It finds that there are no phrases like “hate jews” but in the `The Merchant of Venice' we have in act lines that both talk about “hate” and “jews”. The smart query gets confirmed as `"hate" "jews" || REDUCE:2` (see unary operators below)

with the result “.. I hate him for he is a Christian ..” found in PLAY\ACT\SCENE\SPEECH\LINE

(c) `out out`

It finds a number of lines where “out out” is said such as “Out, out, Lucetta! that would be ill-favour'd” in `The Two Gentlemen of Verona'. The confirmed query is: `“out out”`.

Expressions

While „Smart“ searches is fantastic for many typical use cases (and why we developed it), power users tend to want to perform more precise queries. The implemented infix and RPN languages support fields and paths (like Xquery) as well as a very rich set of unary and binary operators and an assortment of modifiers (prefix and suffix). Since the engine supports a number of data types it includes a number of relations /.<,>,>=,<=,<> whose semantics of depends upon the field datatype. These operators are all overloaded in the query language.

Terms are constructed as **`[[path][relation]]searchterm[:n]`**

Example: `design`

This is the most basic search. Example `"design"` to find "design" anywhere. Or `title/design` to find "design" just in `title` elements. Since paths are case-insensitive there is no difference between `Title/design`, `tITle/design` and `title/design`.

```
Example: title/design
```

Paths can be from the root '\' (e.g. *record**metadata*\title) or from any location (e.g. title or metadata\title)). Paths can be specified with, depending upon indexing, '|' or '/' (or '\\') but one needs in the `field/term` format to be quite careful (with quotes) to delineate the field from the term.

Another way to approach the problem of searching an element (RPN):

Is to use special unary operator called WITHIN:path. `Searching Design WITHIN:title`

is equivalent to title/Design but often more convenient, especially where for the element we have a path. Instead of `\\A\\B\\C\\D\\E/"word"` (since we need to escape the \) we can use the expression: `"word" WITHIN:/A/B/C/D/E`

Note: the expressions:

```
term1 term2 && WITHIN:title
```

and

```
term1 WITHIN:title term2 WITHIN:title &&
```

yield the same results as they are both looking for term1 and term2 in the title element BUT they have different performance. The first expression can be slower since it builds the set for term1 and the set for term2 and then reduces the intersection to a new set that contains only those hits where term1 and term2 were within the title element. The second is taking an intersection of two potentially smaller sets. It is generally faster and better.

✗ Term and datatype search

Elements when searched are assumed to be of their intrinsic datatype when the query if of the same type. Searching a date field, for example, with a date implies a search using the date data type matching algorithms rather than as the individual terms. What constitutes a match depends upon the data type. Dates for example follow the rules of least precision comparison. A date query for 2001 matches, for example, 20010112. While >19 as a string is equivalent to 19* and matches 199 or even 19x as a numerical field all values > 19, e.g. 20, 21, 22, 23 etc.

Term and field paths support "Glob" matching

We support left (with some limitations in terms) and right (without limitations) truncated search as well as a combination of * and ? for glob matching. This can be universally applied to path (fieldname) and with some

limitations to searchterm. These can be connected by more than a dozen binary as well as a good dozen unary relational operators.

Wildcard	Description	Example	Matches
*	matches any number of any characters	Law*	Law, Laws, or Lawyer
?	matches any single character	?at	Cat, cat, Bat or bat
[abc]	matches one character given in the bracket	[CB]at	Cat or Bat
[a-z]	matches one character from the (locale-dependent) range given in the bracket	Letter[0-9]	Letter0, Letter1, Letter2 up to Letter9
{xx,yy,zz}	match any of "xx", "yy", or "zz"	{C,B}at	Cat or Bat

Example: `t*/design would if there was only one field starting with the letter t and it was title search for title/design.`

Glob in path names is quite powerful as it allows one to narrow down search into specific elements.

For every hit we can also examine where it occurred. Example: Searching for
Example: `PLAY\ACT\SCENE\SPEECH\LINE`

VIII. Query Operators (re-Isearch Language)

The re-Isearch engine has been designed to have an extremely rich and expressive logical collection of operators. Some operators can, however, be quite expensive. The complement, for example, of a set with a single result in a large dataset is large. Search time is directly related to the time to build the result set.

Binary Operators

Polymorphic Binary Operators	
Operator	
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater then or equal to

The above operators are overloaded and their semantics are specific to the object type of a field.

Operator	Semantics
< <=	Objects like date, numbers etc. have an order and <, resp. <=, applies as one might expect. For general "string" fields its interpreted as the equivalent to "*" (right truncation)
=	For general "string" fields its interpreted just as "/", viz. a member of the field.
> >=	As </<= above. For general "string" types the sematics are left truncation (as in *term)

Long Operator Name	Sym	Description
OR	||	Union, the set of all elements in either of two sets
AND	&&	Intersection, the set of all elements in both sets
ANDNOT	&!	Elements in the one set but NOT in the other
NOTAND	!&	As above but operand order reversed
NAND	&!	Complement of AND, elements in neither
XOR	^^	Exclusive Union, elements in either but not both
XNOR	^!	Exclusive not OR, complement set of XOR
PROX:num		PROX:0 := ADJ. PROX:n := NEAR:n
NEAR[:num]		matching terms in the sets are within num bytes in the source
BEFORE[:num]		As above but in order before
AFTER[:num]		As above but in order after
DIST[>,>=,<,<=]num		Distance between words in source measured in bytes. (and order)
num > 1: integer for bytes. As fraction of 1: % of doc length (in bytes).		
NEIGHBOR	.~.	
PEER	.=.	Elements in the same (unnamed) final tree leaf node
PEERa		like PEER but after
PEERb		like PEER but ordered after
XPEER		Not in the same container
AND:field		Elements in the same node instance of field
BEFORE:field		like AND:field but before
AFTER:field		like AND:field but after
ADJ	##	Matching terms are adjacent to one another
FOLLOWS	#>	Within some ordered elements of one another
PRECEDES	#<	Within some ordered elements of one another
PROX		Proximity
FAR		Elements a "good distance" away from each other
NEAR	.<.	Elements "near" one another.

Since the engine produces sets of result we can also combine two sets from different database searches into a common set as either augmentation or by performing some operations to create a new set.

Operators that can act on result sets from searching different databases (using common keys)	
JOIN	Join, a set containing elements shared (common record keys) between both sets.
JOINL	Join left, a set containing those on the right PLUS those on the left with common keys
JOINR	Join right, a set containing those of the left PLUS those on the right with common keys

Unary Operators

Operator	Sym	Description
NOT	!	Set compliment
WITHIN[:field]		Records with elements within the specified field. RPN queries "term WITHIN:field" and "field/term" are equivalent. (for performance the query "field/term" is preferred to "term WITHIN:field")
WITHIN[:daterange]		Only records with record dates within the range
WITHKEY:pattern		Only records whose key match pattern
Sre-IsearchLING		Only hits in the same container (see PEER)
INSIDE[:field]		Hits are limited to those in the specified field
XWITHIN[:field]		Absolutely NOT in the specified field
FILE:pattern		Records whose local file path match pattern
REDUCE[:nnn]		Reduce set to those records with nnn matching terms This is a special kind of unary operator that trims the result to metric cutoff regarding the number of different terms. Reduce MUST be specified with a positive metric and 0 (Zero) is a special case designating the max. number of different terms found in the set.
HITCOUNT:nnn		Trim set to contain only records with min. nnn hits.
HITCOUNT[>,>=,<,<=]num		As above. Example: HITCOUNT>10 means to include only those records with MORE than 10 hits.
TRIM:nnn		Truncate set to max. nnn elements
BOOST:nnn		Boost score by nnn (as weight)
SORTBY:<ByWhat>		Sort the set "ByWhat" (reserved case-insensitive names: "Key", "Hits", "Date", "Index", "Score", "AuxCount", "Newsrank", "Function", "Category", "ReverseHits", "ReverseDate", etc.)

SortBy:<ByWhat>

The SORTBY unary operator is used to specify a specific desired sort of the result set upon which it applies. It is used to sort (ByWhat keywords are case insensitive) by

- ➔ “Score” → Score (the default). The score, in turn, depends upon the selected normalization algorithm.
- ➔ “Key” → Alphanumeric sort of the record key.
- ➔ “Hits” → Number of hits (descending)
- ➔ “ReverseHits” → Number of hits (ascending)
- ➔ “Date” → Date (descending)
- ➔ “ReverseDate” → Date (ascending)
- ➔ “Index” → Position in the index
- ➔ “Newsrank” → Newsrank: a special mix of score and date
- ➔ “Category” → Category

- ➔ “Function” → Function
- ➔ “Private1” , “Private2” .. → A number of private sorting algorithms (installable) that use private data (defined by the algorithm) to perform its sort. This is quite site specific and “out-of-the-box” is not defined.
  - ➔ If the private sort handler is not installed (defined) it defaults to sorting by score (ascending).

Queries can also weight various matches or field results to give them more importance (or less). There are also methods to cluster or shift position in results ranking with hits (matches) that are closer to one another („magnitism“).

RPN vs Infix

Internally all expressions are converted into a RPN (Reverse Polish Notation) and placed on stacks.
In reverse Polish notation, the operators follow their operands; for instance, to add 3 and 4 together, one would write 3 4 + rather than 3 + 4. The notation we commonly use in formulas is called “infix”: the binary operators are between the two operands—and unary before—and groups are defined by parenthesis. RPN has the tremendous advantage that it does not need parenthesis or other groupings. It also does not need to worry about order and precedence of operations. In infix expressions parentheses surrounding groups of operands and operators are necessary to indicate the intended order in which operations are to be performed. In the absence of parentheses, certain precedence rules determine the order of operations (such as in the grade school mantra “Exponents before Multiplication, Multiplication before Addition”).

While Infix is generally more familiar (except perhaps users of HP Scientific calculators) with a bit of practice the RPN language is generally preferred for it clarity and performance.
Result sets from searches on terms on the stack are cached to try to prevent repeated searches. Caches may also be made persistent by using disk storage. This is useful for session oriented search and retrieval when used in stateless environments.

Since it is a true query language it is possible to write extremely ineffective and costly queries. There is a facility to give search expressions only a limited amount of „fuel“ to run. One can also explicitly select to use these partial results.

Example RPN	Example Infix
title/cat title/dog || title/mouse ||	title/cat || title/dog || title/mouse
	title/(“cat” || “dog” || “mouse”)
speaker/hamlet line/love AND:scene	(speaker/hamlet and:scene line/love)
out spot PEER	out PEER spot
from/edz 'subject/"EU NGI0"' &&	from/edz && 'subject/"EU NGI0"'

Handbook 0.9 (work in progress)

Personalized thesauri

re-Isearch provides a flexible model for the support of personalized thesauri. They are hooked into the internal query structure (a push stack of terms and their associated attribute structures) as OR'd expansion. The expansion is fully independent of the query language originally used to write the expression.

Thesaurus Format:

```
# Rows of the form:
# parent phrase = child1:weight+child2+multiword child:weight+ ... +childN
#
# White space is ignored at the start and end of child terms
# Comments start with #
spatial=geospatial+geographic+terrestrial # Here are more comments
land use=land cover + land characterization + land surface + ownership property
wetlands=wet land+NWI+hydric soil+inundated
hydrography=stream+river+spring+lake+pond+aqueduct+siphon+well
Al-Qaida=Al-Qa'ida+Al-Kaida+Al-Qaida+Al-Qaida+al-qaeda
kernkraft=automkraft
nuclear=atomic
```

Thesauri are associated with query structures so its easy to implement an interface where each use can manage their own collection of personalized thesauri. Since the thesauri effect only the query structure they have no negative effect on search caching.

Examples

```
# Sample synonyms
war=krieg:2+combat+battle
peace=pax
```

The Infix query: (`"War" and "Peace"`) is expanded as-if (`("war" or "krieg":2 or "combat" or "battle")` and (`("peace" or "pax")`) was entered into the system. The RPN query: `"War":2 "Peace" &&` (really the same as the above Infix query save the use of the weight "2" on the "war" term) is expanded as-if "war" `"krieg":4 || "combat" || "battle" || "peace" "pax" || &&` was entered into the system. Notice that the weights are multiplicative.

Geospatial Search

Two common types of spatial queries used in mapping are proximity and bounding box searches. While "*proximity searches*" return all results within a certain distance of a location, "*bounding box queries*" return all results in a bounding box of four points defined by its four corners. These types of searches are useful in a wide variety of mash-ups.

Proximity searches don't try to be precise since they ignore terrain and other factors. This makes them computationally simple as long as the distance is relatively small.

Haversine formula: R = earth's radius (mean radius = 6,371km) $\Delta lat = lat_2 - lat_1$ $\Delta long = long_2 - long_1$ $a = \sin^2(\Delta lat/2) + \cos(lat_1).\cos(lat_2).\sin^2(\Delta long/2)$ $c = 2.atan2(\sqrt{a}, \sqrt{(1-a)})$ $d = R.c$

Since we are only interested in relatively small distances and rough accuracy we can use the simpler law of cosines:

```
dist = acos ( sin(deg2rad(lat1)) * sin(deg2rad(lat2)) + cos(deg2rad(lat1)) *
cos(deg2rad(lat2)) * cos(deg2rad( lon1 - lon2 )) ) * factor
```

factor = 4552.15 for statue miles (* 0.8684 for nautical miles and * 1.609344 for km). See: http://www.movable-type.co.uk/scripts/latlong.html

Geospatial Proximity search

Fieldname{{Latitude, Longitude}[operator]value were operator: is <, <=, =, >, >= value=distance is km (default) with the modifiers N, K, M for, resp., nautical miles, km and statue miles.

Geospatial Bounding box search

Search for a box is defined by the RECT operator: RECT{N,W,S,E}.

Hierarchical Search

Since many formats (like XML, the underlying model for RSS, Atom and CAPS) have an explicit ancestry of nodes and paths we exploit them for contextual search. This allows for some interesting and powerful searches:

- Find stories with words in the same unnamed node (for instance in the same Title or same Description).
- Find stories with words in the same named node (for instance in the same explicit Title or same Description).
- Find stories with words in the same named explicit node path (for instance in the same explicit Title of an image of a ...).
- Select content of named nodes, named paths or object attributes.
- Select content of descendants of a named ancestor of record hits. Descendants or ancestors may be specified as the name of a node or a path or even a sub-path. This allows one to dynamically at query time define a view and model (unit of recall) to content.

In RSS, for example, this means that one can search for items in the same, for example, item description and request the title and URL of the items that match. Just what one needs given the multiple items per feed!

Lets look a bit closer by example of XML fragments (from SGML/XML markup of Shakespeare's works by Jon Bosak):

```
<SPEECH>
 <SPEAKER>LADY MACBETH</SPEAKER>
 <LINE>Out, damned spot! out, I say!--One: two: why,</LINE>
 <LINE>then, 'tis time to do't.--Hell is murky!--Fie, my</LINE>
 <LINE>lord, fie! a soldier, and afeard? What need we</LINE>
 <LINE>fear who knows it, when none can call our power to</LINE>
 <LINE>account?--Yet who would have thought the old man</LINE>
 <LINE>to have had so much blood in him.</LINE>
</SPEECH>
```

First off I we have an idea of "nearness": being in the same leaf. The words "*out*" and "*spot*" are in the same node (with path ...\SPEECH\LINE). Its named SPEECH ancestor is the above speech--- the only speech in all of Shakespeare's works to have the words "out" and "spot" in the same LINE. The SPEAKER descendant of that SPEECH is "LADY MACBETH".

The word "spot" is said within the works, by contrast, in many other speeches by speakers in addition to Lady Macbeth: SALISBURY in `The Life and Death of King John', BRUTUS as well as ANTONY in `The Tragedy of Julius Caesar', MISTRESS QUICKLY in `The Merry Wives of Windsor', VALERIA in `The Tragedy of Coriolanus', ROSALIND in `As You Like It' and MARK ANTONY in `The Tragedy of Antony and Cleopatra'.

```
<Videos>
  <Video ASIN="B0000DJ7G9">
 <Title Screenplay="William Shakespeare" Alt="Othello">The Tragedy of Othello: The Moor of
Venice</Title>
    <Director>Orson Welles
    <Length>90 minutes</Length>
    <Format>DVD</Format>
    <Certificationg>U</Certification>
  </Video>
<Videos>
```

In the above the value of the `Screenplay` attribute to `Title` is *William Shakespeare*. In re-Isearch one can search for the content of the tag (`TITLE`), the content of an attribute (named mapped with a default of `TITLE@SCREENPLAY` in this example) but also for information like films with Venice in the title whose screenplay was from Shakespeare. Here we appeal to an implicit direct parent (an abstract virtual container that contains the both attribute and field data).

In XML we not only have a parent/child ancestry of nodes but we also have within nodes a linear ordered relationship. One letter follows the next and one word follows the other in a container. In the above example "Yet" precedes "here's" and "a" follows after and finishing with "spot". We have order and at at least a qualitative (intuitive) notion of distance.

In XML we do not, however, have any well-defined order among the siblings (different LINEs). The XML 1.0 well-formedness definition specifically states that attributes are unordered and says also nothing about elements. Document order (how they are marked-up) and the order a conforming XML parser might decide to report the child elements of SPEECH might not be the same. Most systems handling XML from a disk and using popular parsers typically deliver it in the same order but the standard DOES NOT specify that it need be--- and for good reason.

See also: Presentation Methods and Elements.

Order

Lady Macheth says "*spot*" in another speech too..

```
<SPEECH>
  <SPEAKER>LADY MACBETH<<SPEAKER;>
  <LINE>Yet here's a spot.</LINE>
</SPEECH>
```

These "*spot*"s are in "PLAY\ACT\SCENE\SPEECH\LINE"
In XML we not only have a parent/child ancestry of nodes but we also have within nodes a linear ordered relationship. One letter follows the next and one word follows the other. In the above example "*Yet*" precedes "*here's*" and "*a*" follows after and finishing with "*spot*". We have order and at at least a qualitative (intuitive) notion of distance.
One could then specify an inclusion (within the same unnamed or named field or path), an order and even a character (octet) metric.

We have not attempted to implement a word metric as the concept of word is more complicated then commonly held. Is edz@bsn.com a single word? Two words? One word? Maybe even 3? What about a URL? Hyphenation as in "auto-mobile"? Two words? On the other hand what does such a distance mean?
Our metric of distance is defined as the file offsets (octets) as the record is stored on the file system. This too is less than clear cut as the render from HTML of Überzeugung and Überzeugung are equivalent but their lengths are different.

```
zum Thema Präsentieren und Überzeugen
```

The file system offsets between the word "Thema" and "und" depends upon how "Präsentieren" was encoded. As UTF-7, UTF-8, UTF-32, Latin-1 or with entities ¨ and &;Uuml; or &#nnn form. Does one, instead, treat the rendered level? This too is misleading since different output devices might have quite different layouts. What about columns? And the tags?

Tags are, of course, even worse with words since we have started to associate a semantics for distance. Look at an XML fragment like

```
<name>Edward C. Zimmermann</name><company>NONMONOTOMIC Lab</company>
```

What is the word distance between "Lab" and "Edward" keeping in mind that XML markup is equivalent if NAME is specified before or after COMPANY?

```
<company>NONMONOTOMIC Lab</company><name>Edward C. Zimmermann</name>
```

These are equivalent content and the word distance? That's right.. its not well defined (that is why the XML people came out withe xs:sequence for Schemas to denote in applications when order matters)! Leaving order aside, do we count “name” and “company” as words?
Order and inclusion do make sense, even some rough guide to distance but words? We are, of course, open to convincing examples!

We instead can say the distance between “Zimmermann” and “NONMONOTONIC” in the first example is 17 and in the second 30. The guiding force is not how the data is represented in an internal model but in how the data was provided in the input.

Counting containers

We can also count containers using C++ (or one of the language bindings).

```
GetNodeOffsetCount(GPTYPE HitGp, STRING FieldNameorPath= "", FC *Fc= NULL)
```

where HitGp is the address of a hit, FieldNameofPath the name, resp. path, of the container we are interested in and Fc (optional) is its calculated start/end coordinates.
This is useful to be able to determine which page (paragraph or line) a hit is on but may also be used in an abstract sense.
Example:

```
  <ingredients>
    <item>Chocolate</item>
    <item>Flour</item>
    <item>Butter</item>
    <item>eggs</item>
  </ingredients>
```

The ***order*** is the order in the mark-up. Chocolate might be the 25th item in the document but its the first item in the particular instance of INGREDIENTS. This is possible since we can check the count of the instance of INGREDIENTS and can look at the previous we can count also the offset of ITEM.

Comparison of the re-Isearch language to CQL

CQL query consist, like the re-Isearch language, of either a single search clause or multiple search clauses connected by boolean operators. It may have a sort specification at the end, following the 'sortBy' keyword. In addition it may include prefix assignments which assign short names to context set identifiers.

Re-Isearch Expression (re-Isearch language)	**CQL Expression**
dc.title/fish	dc.title any fish
dc.title/fish dc.creator/sanderson OR	dc.title any fish or dc.creator any sanderson

Ranking and Normalization

Ranking

We support many models of sorting result sets:

- By Record Key

- By order as indexed
- By an external sort
- By Score
- By Adjuncted Score
- By number of different term matches
- By Date (either forward or reverse)
- By Category
- By Newsrank (a heuristic that combines "score" with a function of chonology). The idea behind "Newsrank" is that newer stories are more significant than older ones.

Score

Score is the product of normalization of hits and there are several models.

Spatial Ranking

Spatial searches are scored according to the work "*A spatial overlay ranking method for a geospatial search of text objects*"[3], Lanfear, Kenneth J. & U.S. Geological Survey, 2006, USGS Reston, Va.. The spatial overlay score tries to correlate how well an object's footprint matches the search's spatial extent (defined by a bounding box).

Normalization

The re-Isearch engine supports many models of normalization. These may be chosen at search time:

- Cosine normalization: Despite being many decades old, still seems to be among the best. Its main drawback (beyond the need to create full sets) is that it tends to be biased towards shorter records, finding them more frequently than alone the linear distribution of lengths might suggest.
- Log Normalization: Normalized according to the log of document length.
- Max Normalization: Normalized to favor those with more different hits.
- Bytes Normalization: Various document length normalization models have been proposed to address the bias of Cosine Normalization towards shorter records. They, however, nearly always tend to overly penalize long records. With Byte Normalization a middle ground approach is taken: the cosine model is slightly modified to also take document length distribution into consideration.
- Cosine Metric Normalization: Yet another variant of Cosine Normalization. The byte metric distance between "hits" (multiple term searches) is used to favor records where these are closer. Since hits are always closer in shorter records than in longer we limit ourselves to the minimum of all the distances rather than an average and adjust.

Score Bias

Scores can in turn be "biased" according to "priority" (a per record scalar), "category" (a per-record predicate) and temporal (date metadata). The skew can be used to "boost" or downgrade scores. In re-IsearchU News priority is caculated as a value to represent a number of features of the information source (such as data quality).

3 http://pubs.usgs.gov/of/2006/1279/2006-1279.pdf

Magnetism

The re-Isearch engine also features a unique feature called magnetism. It allows (set via per-index parameters) to allow items in a result list to "attract" to one another (or repel) allowing them to fall into segmented clusters.

IX. Presentation Elements and Methods

One of the design goals of re-Isearch was to facilitate implementation of the ISO 23950 (ANSI/NISO Z39.50) Information Retrieval Protocol services standard. Z39.50 makes it easier to use large information databases by standardizing the procedures and features for searching and retrieving information.

ISO 23950 has the concepts of record syntax and record elements.

Presentation Elements (Metadata)

Presentation is requested of a record by its element name and its preferred record syntax.

- The element names specifies whether the client wants full records ("f"), brief records ("b") one of of the available fields (or a combination thereof).
- The preferred record syntax specifies the format of retrieval. It can be USMARC, SUTRS, GRS-1 or some other format. The values which may be set for this option must be taken from an enumerated set which is specified by the binding.

Element Name	Element	Description
Brief	B	Brief description of records (typ. Title or Headline)
Full	F	A "Non-brief" description, typically the full viewable record (not always the complete stored record). Views may depend upon user privileges and rights which different users getting different "full record" presentations.
Metadata	M	A metadata record describing the record
Location	L	Location (often URI) to access the record.

The elements "B" and "F" are mandated by the ISO23950/Z39.50 standard. re-Isearch reserves the 1-letter element space for extensions and special derived record elements.

Ancestors and Descendants

In re-Isearch one can select the name ancestors (via name or path) for hits. One can also request a named descendant of such an ancestor.

Lets look a bit closer by example of XML fragments (from SGML/XML markup of Shakespeare's works by Jon Bosak):

```
<SPEECH>
 <SPEAKER>LADY MACBETH</SPEAKER>
```

```
 <LINE>Out, damned spot! out, I say!--One: two: why,</LINE>
 <LINE>then, 'tis time to do't.--Hell is murky!--Fie, my</LINE>
 <LINE>lord, fie! a soldier, and afeard? What need we</LINE>
 <LINE>fear who knows it, when none can call our power to</LINE>
 <LINE>account?--Yet who would have thought the old man</LINE>
 <LINE>to have had so much blood in him.</LINE>
</SPEECH>
```

The words "*out*" and "*spot*" are in the same node (with path ...\SPEECH\LINE). Its named SPEECH ancestor is the above speech--- the only speech in all of Shakespeare's works to have the words "out" and "spot" in the same LINE. The SPEAKER descendant of that SPEECH is "LADY MACBETH".

These elements are specified as

```
AncestorPath[/DescendantPath]
```

Examples:

- *SPEECH* (or *PLAY\ACT\SCENE\SPEECH*) would return the content (markup) of (all) the speeches that contain the hits
- *SPEECH/SPEAKER* (or *PLAY\ACT\SCENE\SPEECH/SPEAKER* or *SPEECH/PLAY\ACT\SCENE\SPEECH\ SPEAKER* or ...) would return their speakers.

If, on the other hand, we wanted to know the different speakers in the act where Lady Macbeth said "*Out, damned spot! out, I say*" we would request *SCENE/PLAY\ACT\SCENE\SPEECH/SPEAKER) (or SCENE/SPEAKER). This would yield of list including "Doctor" and "Gentlewoman" alongside the guilt ridden Lady.*

X. *Presentation Syntax*

In re-Isearch we may distinguish between the format that a record was created or stored and the format requested from a search.

Record Syntaxes

Record Syntax	Description
RAW	Raw. As stored on the file system. The whole record is just read into the system as stored.
SUTRS	Simple unstructured text plain text. This might be--- but must not be--- the same as RAW but may be rendered.
HTML	Provide a Web version for view using a contemporary browser. This might not be in HTML markup but may contain the HTTP MIME header followed by the binary stream for some forms of context (such as PDF).
SGML	Provide a SGML version of a metadata record for the resource.
XML	An XML version of metadata.

Record Syntax OIDs

Object Identifier	Name	Usage	Reference
1.2.840.10003.5.**100**	Explain	Explain Records	REC.1
1.2.840.10003.5.**100.1**	Revised Explain Syntax definition to support ZSQL query	Explain Records	ZSQL additions to Explain

1.2.840.10003.5.**101**	SUTRS	simple, unstructured text	REC.2
1.2.840.10003.5.**102**	OPAC	OPAC Records	REC.3
1.2.840.10003.5.**103**	Summary	Summary Records	REC.4
1.2.840.10003.5.**104**	GRS-0	(superceded by GRS-1)	
1.2.840.10003.5.**105**	GRS-1	structured information	REC.5
1.2.840.10003.5.**106**	ESTaskPackage	Extended Services	REC.6
1.2.840.10003.5.**107**	fragment	fragmentation	FragmentSyntax
1.2.840.10003.5.**108**	(not used)	(previously assigned to HTML; superceded by 1.2.840.10003.5.109.3)	
1.2.840.10003.5.**109**	✗ IANA mime types	(see note following this table)	
1.2.840.10003.5.**109.1**	pdf	mime type application/pdf	Portable Document Format Reference; Manual, Adobe Systems, Inc, Addison-Wesley Publishing Co. ISBN 0-201-62628-4, 1993.
1.2.840.10003.5.**109.2**	postscript	mime type application/postscript	rfc 1341
1.2.840.10003.5.**109.3**	html	mime type text/html	rfc 1866
1.2.840.10003.5.**109.4**	tiff	mime type image/tiff	Tag Image File Format (TIFF) class F, revision 6.0
1.2.840.10003.5.**109.5**	~~pdf~~ **error, should be gif**	mime type image/gif	rfc 1341
1.2.840.10003.5.**109.6**	jpeg	mime type image/jpeg	rfc 1341
1.2.840.10003.5.**109.7**	png	mime type image/png	Internet-Draft draft-boutell-png-spec-04.txt, "Png (Portable Network Graphics) Specification Version 1.0" Informational RFC; see item 11 of the IESG minutes from meeting of 7-11-96.
1.2.840.10003.5.**109.8**	mpeg	mime type video/mpeg	rfc 1341
1.2.840.10003.5.**109.9**	sgml	mime type text/sgml	rfc 1874
1.2.840.10003.5.**109.10**	xml (no specific version)	mime type text/xml	RFC 2376
1.2.840.10003.5.**109.10.1.0**	xml version 1.0		
1.2.840.10003.5.**109.10.1.1**	xml version 1.1		
1.2.840.10003.5.**109.11**	(*not used*)		
1.2.840.10003.5.**110**	✗ Z39.50 mime types	(see note following this table)	
1.2.840.10003.5.**110.1**	tiff-b		
1.2.840.10003.5.**110.2**	wav		
1.2.840.10003.5.**111**	SQL-RS	SQL	Z+SQL Profile
1.2.840.10003.5.**112**	xml-b	XML record according to the schema or definition identified by the element set name.	Implementor agreement: Requesting XML Records
1.2.840.10003.5.**1000**	✗ locally registered syntaxes		

1.2.840.10003.5.**1000.3.1**	Latin-1	locally registered by Lucent Technologies	
1.2.840.10003.5.**1000.3.2**	TIFF	locally registered by Lucent Technologies	
1.2.840.10003.5.**1000.3.3**	Postscript	locally registered by Lucent Technologies	
1.2.840.10003.5.**1000.6.1**	GRS-0	locally registered by CAS	also registered as 1.2.840.10003.5.104
1.2.840.10003.5.**1000.6.2**	CXF	locally registered by CAS	
1.2.840.10003.5.**1000.147.1**	ADF	locally registered by Astrophysics Data System (Smithsonian Astrophysical Observatory)	

A mime type may be registered as a Z39.50 mime type when there is no current appropriate iana type. When a mime type is so registered it is with the understanding that that registration will be depricated if and when an appropriate mime type is defined by iana, and it will be superceded by a subsequent registration in the iana category.

Highlighting and hit context

The engine provides a wide range of highlighting methods inclusive of support of Adobe's PDF Highlighting. Hit context exploits the document structure to text within its own container. The context for a hit for "Edward" in `<name>Edward Zimmermann</name><address>NONMONOTONIC...` *contains, for example, the name Edward Zimmermann but not NONMONOTONIC. Algorithms are provided to select both the context of individual "hits" with a record but also to calculate an "optimal" context based around a measure of hit distance metrics.*

XI. Centroids (Meshes and P2P)

The re-Isearch engine is not just about search but about what we have called “re-search”, a recursive model of search where one searches first for where to search rather than just demanding a list of records.
One of the approaches developedto enable this within a distributed network is a so-called “bag of words” centroid. Each target (index) can be viewed as a potential node withing a search mesh or peer-to-peer network. Here one searches not for specific records but for databases that might have the data one is looking for: targets with content (terms) in specific fields that meets a query rather than for document or records.
To make this possible one can generate a so-called “Centroid” file for a target. It is a file (using a kind-of XMLish format) that lists each field (path) and its included terms (and their frequency).

Here is a fragment of the file generated for an index of the complex works of Shakespeare:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
        <!-- generator="re-Isearch 4.0a x86_64 GNU/Linux" -->
        <!DOCTYPE Locator SYSTEM "centroid.dtd">
<centroid src="/var/opt/nonmonotonic/indexes/bart/BART" version="0.1">
 <words slots="22945" slot_length="28" source_charset="iso-8859-1">
  <word freq="37">0</word>
  <word freq="40">1</word>
```

```
  <word freq="1">10</word>
….
</words>
 <fields count="77">
  <ACT type="string" elements="22819">
    <word freq="1">10</word>
    <word freq="1">2d</word>
    <word freq="2">2s</word>
….
    <word freq="1">zone</word>
    <word freq="21">zounds</word>
    <word freq="1">zwaggered</word>
  </ACT>
….
<PLAY type="string" elements="22911">
    <word freq="1">10</word>
    <word freq="74">1992</word>
    <word freq="37">1994</word>
    <word freq="37">1996</word>
    <word freq="74">1999</word>
…
 </fields>
</centroid>
```

Searching for a terms like “zounds” in an ACT element would here turn up the target BART where the word occurs 21 times. Searching BART a searcher could then see that the terms occurs in 6 plays of Shakespeare and most prominently in the `The First Part of Henry the Fourth' (occurring 10 times) but also occurring only once in both the `The Tragedy of Titus Andronicus' ('Zounds, ye whore! is black so base a hue?) and `The Life and Death of King John' (Zounds! I was never so bethump'd with words) .

XII. Scan Service

Instead of searching for records or content that meets the demands of some query sometimes one might want to search for terms or queries themselves. This is called “scan”. With it one can browse indexes to view a list of the words or phrases included. Scan supports searching into structure and since it allows for all the magic of of term search once can use it to find specific words to search rather than wildcards to keep noise down.
Example instead of “cheap*” (in the collected works of Shakespeare) once could see that we had also the term cheapside alongisde cheapen, cheapest and cheapy. Cheapside is a street in the City of London, the historic and modern financial centre of London. It was for a long time one of the most important streets in London. Shakespeare used Cheapside as the setting for several bawdy scenes in Henry IV, Part I. In Henry VI, Part II, the rebel Jack Cade: "all the realm shall be in common; and in Cheapside shall my palfrey go to grass".

Searching (the collected works of Shakespeare again) for “jew*” one would see that “jewel” occurs in many more plays (29) than “jew” (7 plays) but is less frequent (an incidence of 69 times in `The Merchant of Venice') or “jewish” (which occurs twice in `The Merchant of Venice' but no where else). Jewel, by contrast, was most frequent in `The Life of Timon of Athens' but there it occurred only 8 times.

The scan facility can be used to develop other search interfaces such as facets.

XIII. Facets

Faceted search is a popular strategy for *information exploration* to assist the searcher

- Discover relationships or trends between contents.
- To enable dynamic classified browsing.
- Help users who don't know precisely what they can find or what to search for (exploration and discovery).

From the perspective of the information provider they

- Allow one to push searches (navigate them) into specific areas (define top-down narratives).
- Create an aura of relevance.

The primary design goal is to allow users to move through large information spaces in a flexible manner both refining and expanding the queries, while maintaining a consistent representation of the collection's structure and "not getting lost".

Effective faceted search depends on well organized taxonomies. Many organizations, unfortunately, have not done the upfront classification work needed to fully leverage facets for search and guided navigation. The “let’s just use what we have and see what happens” approach may sometimes provide some benefit but nearly always miss their potential and often mislead and navigate users into the wrong direction. Our experience is that the cost of "bad navigation" can be— especially in eCommerce applications— very high. Even with well defined taxonomies we've found that only in homogeneous well-defined settings with shared semantics (men, women, boys, girls, pants, skirts, tops etc.) can they be effective.

Typical implementations of faceted search is via (db intensive) relational database calls. With indexing applied to fields this still tends to limit them to sites with a relatively small number of items (nodes), low update frequency (inserts to indexed fields demand re-indexing) and with comparatively low traffic.
re-Isearch handles things differently. Information is de-coupled from the RDMS and off-loaded. Faceted search is implemented in re-Isearch via cacheable search calls including:

- Field/Path faceting: the counts for all terms, or just the top terms in any given field or path.
- Path Fragement Faceting: Like field/path faceting but is a relative path (in the document tree) via the field/path instance where a given query found a match.
- Query faceting: the number of documents in the current search results that also match the given query.
- Date faceting: the number of documents that fall within certain date ranges.

- Data faceting: the number of documents that fall with a numerical range in a *specific numerical field*, geospatial bounding box etc.

Like Search and Scan, **Faceted Search** its offloaded from the application front end (CMS etc.) and provided as services via the server.

XIV. Programming Languages

Since the internal representation of a query is a RPN stack and we have a number of programming interfaces (C++, Python, Java, PHP, Tcl, etc.) we have the tremendous power to express and store what we wish. At current we don't have a SQL or SPARQL interface but it should be relatively easy for a contributor to write in Python (or one of the other languages).

Python

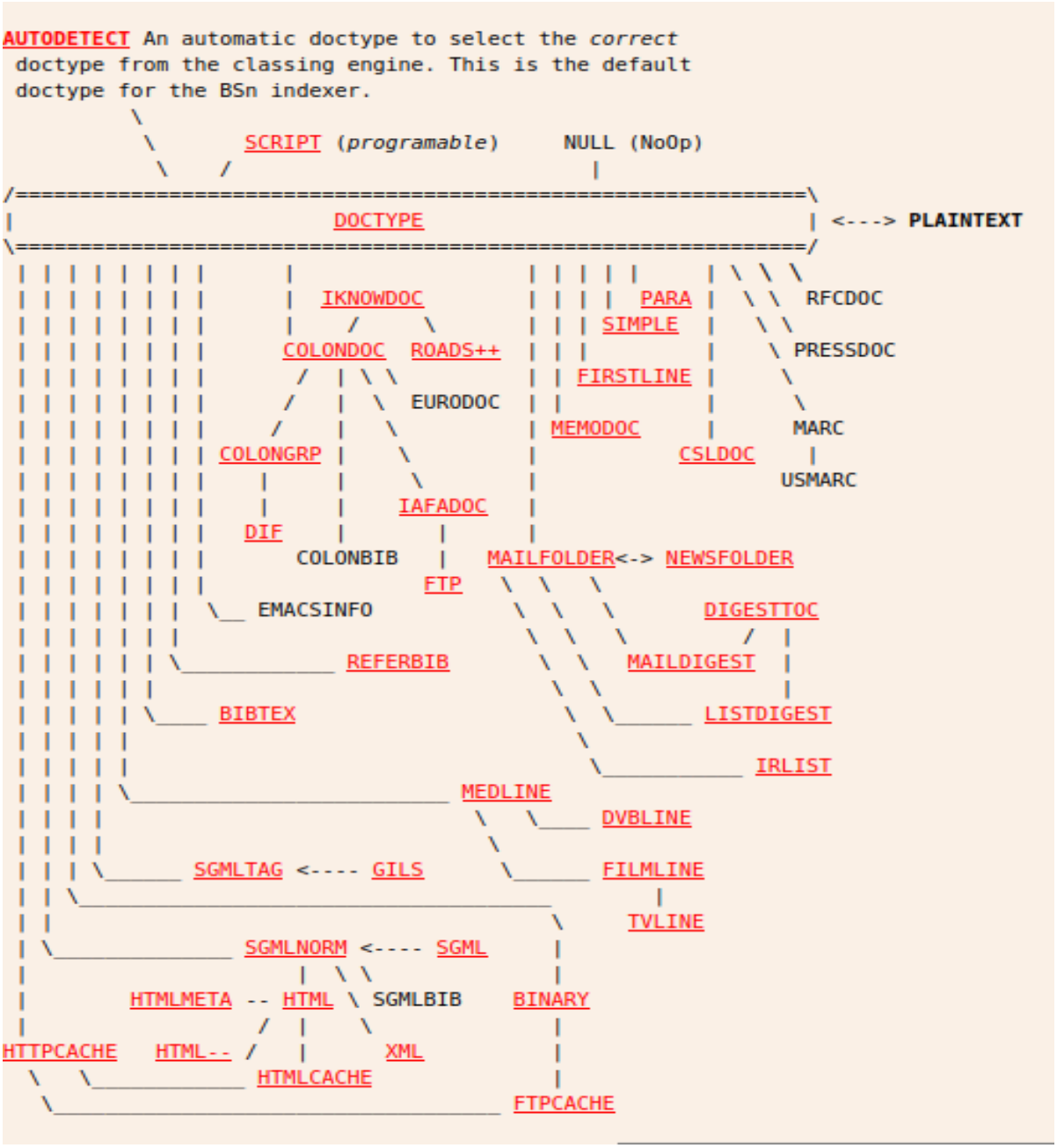
In Python one can build a query like:

```
query = "beschaffungsmanagement:3 OR beschaffungsmarketing:3 OR beschaffungsmarkt:3
OR beschaffungsplanung:3 OR beschaffungsprozesse:3 OR (deterministische:3 FOLLOWS
beschaffung:3) OR einkaufspolitik:3 OR (stochastische:3 FOLLOWS beschaffung:3) OR
strategien:2 OR strategie OR (c:3 FOLLOWS teilemanagement:3) OR
beschaffungsmarktforschung:3 OR (double:4 FOLLOWS sourcing:4) OR (global:4 FOLLOWS
sourcing:4) OR (modular:4 FOLLOWS sourcing:4) OR (multiple:4 FOLLOWS sourcing:4) OR
(single:4 FOLLOWS sourcing:4) OR sourcing:3 OR methoden:2 OR methode OR lieferant:3
OR lieferanten:2 OR logistikdienstleister:3 OR rahmenvertraege:3 OR tul:4 OR
spediteur:3 OR spediteure:2 OR spediteuren:2 OR spediteurs:2 OR stammlieferant:3 OR
vertraege:3 OR vertrag:2 OR vertraegen:2 OR vertrages:2 OR vertrags:2 OR
zulieferpyramide:3 OR partner:2 OR partnern OR partners OR beschaffungskosten:3 OR
einkaufscontrolling:3 OR einkaufsverhandlungen:3 OR incoterms:3 OR
wiederbeschaffungszeit:3 OR zahlungskonditionen:3 OR konditionen:2 OR kondition OR
einfuhr:3 OR einfahre:2 OR einfahren:2 OR einfahrend:2 OR einfahrest:2 OR
einfahret:2 OR einfahrt:2 OR einfuehrt:2 OR einfuehre:2 OR einfuhren:2 OR
einfueren:2 OR einfuehrest:2 OR einfuehret:2 OR einfuhrst:2 OR einfuhrt:2 OR
eingefahren:2 OR einzufahren:2 OR eust:4 OR einfuhrumsatzsteuer:3 OR inbound:3 OR
jis:4 OR (just:3 FOLLOWS in:3 FOLLOWS sequence:3) OR jit:4 OR (just:3 FOLLOWS in:3
FOLLOWS time:3) OR sendungverfolgung:3 OR stapler:3 OR staplern:2 OR staplers:2 OR
we:4 OR wareneingang:3 OR wa:4 OR warenausgang:3 OR wareneingangskontrolle:3 OR
zoll:3 OR zoelle:2 OR zoellen:2 OR zolles:2 OR zolln:2 OR zolls:2 OR gezollt:2 OR
zolle:2 OR zollen:2 OR zollend:2 OR zollest:2 OR zollet:2 OR zollst:2 OR zollt:2 OR
zollte:2 OR zollten:2 OR zolltest:2 OR zolltet:2 OR zollware:3 OR transport:2 OR
transporte OR transporten OR transportes OR transports"
db_path="/var/opt/nonmonotonic/NEWS";
pdb = IDB(db_path); ## Open the index
squery = SQUERY(query); # Build the query
irset = pdb.Search(squery, ByScore); # Run the query
## The resulting irset is a set which we can perform operations upon or combine with another irset
## from another search using the operators in the tables above—for  example Or, Nor, And, ….
irset = irset1.And(irset2); ## This is like irset = irset1 AND irset2
```

As one can see it is relatively straightforward to build alternative query languages to run.

XV. Doctypes

This is an old (historic) map of the doctypes



Some, like script, have been abandoned and instead supplemented by using the engine within a script language—instead of embedding Python into the engine, the engine is embedded into Python as a module. Many others are similar just in name only.

What has not changed is that the root class of all document handlers is the “DOCTYPE” class—vastly expanded since that map. Every other class is a descendant. Some like AUTODETECT are not true document handlers but managers for detecting which document format to use to handle some document. Others like SGMLNORM are master document types useful in themselves for handling SGML document but also to provide parser services for XML documents. The XML class (child of SGMLNORM itself child of DOCTYPE) has a child XMLBASE. It adds special handling for heirarchical fields. XMLBASE itself has a number of children like XMLREC and GILSXM. GILSXML too has children like NEWSML. Some handlers are not purely hierarchical but as a family pass work to their children (rather than the always the other way around). A good example is MAILFOLDER, a handler designed for folders of mail (RFC822) messages (a child of PTEXT). Since mails can sometimes have other formats like mailing lists it like AUTODETECT but on mail messages tries to detect the format. If it looks like a digest it gets passed to the MAILDIGEST class (a child of MAILFOLDER) which in turn might pass it to LISTDIGEST (a child of MAILDIGEST) or even AOILLIST (which is designed for AOL's RFC-noncompliant Listserver Mail Digests).

The current collection (July 2021):
Available Built-in Document Base Classes (v28.6):

AOLLIST	ATOM	AUTODETECT	BIBCOLON
BIBTEX	BINARY	CAP	COLONDOC
COLONGRP	DIALOG-B	DIF	DVBLINE
ENDNOTE	EUROMEDIA	FILMLINE	FILTER2HTML
FILTER2MEMO	FILTER2TEXT	FILTER2XML	FIRSTLINE
FTP	GILS	GILSXML	HARVEST
HTML	HTML--	HTMLCACHE	HTMLHEAD
HTMLMETA	HTMLREMOTE	HTMLZERO	IAFADOC
IKNOWDOC	IRLIST	ISOTEIA	LISTDIGEST
MAILDIGEST	MAILFOLDER	MEDLINE	MEMO
METADOC	MISMEDIA	NEWSFOLDER	NEWSML
OCR	ODT	ONELINE	OZSEARCH
PAPYRUS	PARA	PDF	PLAINTEXT
PS	PTEXT	RDF	REFERBIB
RIS	ROADS++	RSS.9x	RSS1
RSS2	RSSARCHIVE	RSSCORE	SGML
SGMLNORM	SGMLTAG	SIMPLE	SOIF
TSLDOC	TSV	XBINARY	XFILTER
XML	XMLBASE	XMLREC	XPANDOC
YAHOOLIST			

External Base Classes ("Plugin Doctypes"):

```
  RTF:               // "Rich Text Format" (RTF) Plugin
  ODT:               // "OASIS Open Document Format Text" (ODT) Plugin
  ESTAT:             // EUROSTAT CSL Plugin
  MSOFFICE:          // M$ Office OOXML Plugin
  USPAT:             // US Patents (Green Book)
  ADOBE_PDF:         // Adobe PDF Plugin
  MSOLE:             // M$ OLE type detector Plugin
  MSEXCEL:           // M$ Excel (XLS) Plugin
  MSRTF:             // M$ RTF (Rich Text Format) Plugin [XML]
  NULL:              // Empty plugin
  MSWORD:            // M$ Word Plugin
```

```
PDFDOC:            // OLD Adobe PDF Plugin
TEXT:              // Plain Text Plugin
ISOTEIA:           // ISOTEIA project (GILS Metadata)
```

NOTE: *The version built and/or distributed may have a list that differs from the one above as the "real" documentation is always the built-in list and NOT this handbook.*

NOTE: Every built-in doctype can be extended to have its own options by using the CHILD:PARENT convention, e.g. IDISS:XMLREC to define a document format IDISS that will use the XMLREC handler. The specific behavior is set by the options.

✗ General Options:

The various doctypes has a number of options allowing them to be used generically in a wider range of applications and for a larger class of document. **All the options set-able in a parent doctype can be set by its children.**

In the **`[doctype]`** section of the DB.ini (where doctype is the name of the doctype handler) one can specify a number of options specific to the database—versus in the doctype.ini configuration file where the options are set specific to the doctype in its **`[general]`** section-- as **`Key=Value`**

Key	Value
Headline	<Headline format to over-ride default Brief record> The format is %(FIELD1)...text...%(FIELD2) ... where the %(X) declarations get replaced by the content of field (X). The declaration %(X?:Y) may be used to use the content of field (Y) should field (X) be empty or undefined for the record.
Headline/<RecordSyntax OID>	<Headline format for Record Syntax>
Summary	<Format to define a Summary> # See Headline
Summary/<RecordSyntax OID>	<Format to define a Summary> # See Headline
Content-Type	<MIME Content type for the doctype>
FieldType	<file to load for field definitions> # default <doctype>.fields
DateField	<Field name to use for date of record>
DateModifiedField	<Date Modified field for record>
DateCreatedField	<Date of record creation field>
DateExpiresField	<Date of record expiration>
TTLField	<Time to live in minutes: Now+TTL = DateExpires>
LanguageField	<Field name that contains the language code>
KeyField	<Field name that contains the record key>
MaxWordLength	<Number, words longer than this won't get indexed>

Help	<Help text>

Each doctype in turn has its own doctype.ini configuration file where a number of its parameters are set.

[Fields]	Field=Field1 # Map fields into others
[FieldTypes]	Field=<FieldType>
[External/<RecordSyntax OID>]	Field=<program> # e.g. F=cat would pipe the record for full element presentation to cat
[Present <Language-Code>]	Field=<Display format name for Field when the record has language>
[Defaults]	Field=<Default Value> Default value for Field if not in record.

In the <db.ini> some of these can be embedded as

[Defaults <Doctype name>]	Field=<Default Value> Default value for Field if not in record.
[<Uppercase Doctype Name>]	The general options in the table abve

AUTODETECT:

Lets start off the with AUTODETECT type as it's often the go-to default. Class Tree:

DOCTYPE (Generic Document Type)

v

AUTODETECT

AUTODETECT is a special kind of doctype that really isn't a doctype at all. Although it is installed from the viewpoint of the engine as a doctype in the doctype registrary, it does not handle parsing or presentation and only serves to map and pass responsibility to other doctypes. It uses a complex combination of file contents and extension analysis to determine the suitable doctype for processing the file.

The identification algorithims and inference logic have been designed to be smart enough to provide a relatively fine grain identification. The analysis is based in large part upon content analysis in contrast to purely magic or file extension methods. The later are used as hints and not for the purpose of identification. These techniques allow the autodetector to distinguish between several different very similar doctypes (for example MEDLINE, FILMLINE and DVBLINE are all based upon the same basic colon syntax but with slightly different features). It allows one to index whole directory trees without having to worry about the selection of doctype. It can also detect many doctypes where there are, at current, no suitable doctype class available or binary files not probably intended for indexing (these include misc files from FrameMaker, SoftQuad Author/Editor, TeX/METAFONT, core files, binaries etc). At current ALL doctypes available are identified. For doctypes that handle the same document formats but for different functions (eg. HTML, HTML-- and HTMLMETA) given that being logical does not mean it can read minds. For these one must specify the document parser or the most general default parser would be chosen (eg. HTML for the entire class of HTML files).

Should the document format not be recognized by the internal logic it then appeals to, should it have been built with it (its optional) libmagic. That library has a user editable magic file for identification. If the type is identified as some form of "text", viz. not as some binary or other format, then it is associated with the PLAINTEXT doctype.

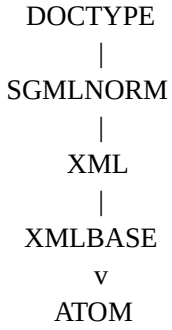
Since it has proved accurate, robust and comfortable it is the default doctype.

✗ Options in .ini:

```
[General]
Magic=<path>     # Path of optional magic file
ParseInfo=[Y|N]  # Parse Info for binary files (like images)

[Use]
<DoctypeClass>=<DoctypeClassToUse> # example HTML=HTMLHEAD
<DoctypeClass>=NULL  # means don't index <DoctypeClass> files
```

"ATOM": IETF AtomPub

Supports various flavors of the Atom 1.0 Syndication Format. The format is an XML language used for web feeds envisioned as a replacement for RSS. Its last release as a standard was in 2007. The Atom Syndication Format was issued as a Proposed Standard in IETF RFC 4287 in December 2005. The co-editors were Mark Nottingham and Robert Sayre. This document is known as atompub-format in IETF's terminology. The Atom Publishing Protocol was issued as a Proposed Standard in IETF RFC 5023 in October 2007. Two other drafts have not been standardized



Example:

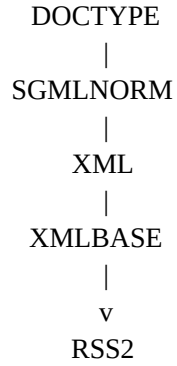
```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
       <title>Example Feed</title>
       <subtitle>A subtitle.</subtitle>
```

```
	<link href="http://example.org/feed/" rel="self" />
	<link href="http://example.org/" />
	<id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
	<updated>2003-12-13T18:30:02Z</updated>
	<entry>
		<title>Atom-Powered Robots Run Amok</title>
		<link href="http://example.org/2003/12/13/atom03" />
		<link rel="alternate" type="text/html"
href="http://example.org/2003/12/13/atom03.html"/>
		<link rel="edit" href="http://example.org/2003/12/13/atom03/edit"/>
		<id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
		<updated>2003-12-13T18:30:02Z</updated>
		<summary>Some text.</summary>
		<content type="xhtml">
			<div xmlns="http://www.w3.org/1999/xhtml">
				<p>This is the entry content.</p>
			</div>
		</content>
		<author>
			<name>John Doe</name>
			<email>johndoe@example.com</email>
		</author>
	</entry>
</feed>
```

The Atom format is, like RSS, still deployed by many sites but its use in the mainstream storefront has been widely upstaged by JSON. In the backend, AtomPub continues to have a strong presence via the OASIS OData protocol.

> *"A REST-based protocol, OData builds on HTTP, AtomPub, and JSON using URIs to address and access data feed resources. It enables information to be accessed from a variety of sources including (but not limited to) relational databases, file systems, content management systems, and traditional Web sites. OData provides a way to break down data silos and increase the shared value of data by creating an ecosystem in which data consumers can interoperate with data producers in a way that is far more powerful than currently possible, enabling more applications to make sense of a broader set of data. Every producer and consumer of data that participates in this ecosystem increases its overall value." –*
> *"OASIS Open Data Protocol (OData) TC | OASIS".*

RSS2



Handbook 0.9 (work in progress)

RSS2 is the Really Simple Syndication 2.0.(XML) format handler. Websites usually use RSS feeds to publish frequently updated information, such as blog entries, news headlines, episodes of audio and video series, or for distributing podcasts. An RSS document includes full or summarized text, and metadata, like publishing date and author's name. RSS formats are specified using a generic XML file.

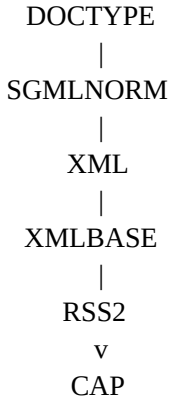
RSS verus ATOM:

RSS 2.0	Atom 1.0
author	author*
category	category
channel	feed
copyright	rights
—	subtitle
description*	summary and/or content
generator	generator
guid	id*
image	logo
item	entry
lastBuildDate (in channel)	updated*
link*	link*
managingEditor	author or contributor
pubDate	published (subelement of entry)
title*	title*
TTL	—

Note: default special extension is "etag" field for key.

CAP

Supports a variant of the OASIS Standard CAP-V1.1, October 2005 and CAP-V1.2, July 2010 in an RSS2 news feed.
Class Tree:



Handbook 0.9 (work in progress)

The Common Alerting Protocol (CAP) is an XML-based data format for exchanging public warnings and emergencies between alerting technologies. CAP allows a warning message to be consistently disseminated simultaneously over many warning systems to many applications, such as Google Public Alerts and Cell Broadcast. CAP increases warning effectiveness and simplifies the task of activating a warning for responsible officials.

Standardized alerts can be received from many sources and configure their applications to process and respond to the alerts as desired. Alerts from the Department of Homeland Security, the Department of the Interior's United States Geological Survey, and the United States Department of Commerce's National Oceanic and Atmospheric Administration (NOAA), and state and local government agencies can all be received in the same format by the same application. That application can, for example, sound different alarms, based on the information received.

By normalizing alert data across threats, jurisdictions, and warning systems, CAP also can be used to detect trends and patterns in warning activity, such as trends that might indicate an undetected hazard or hostile act. From a procedural perspective, CAP reinforces a research-based template for effective warning message content and structure.

Example:

```
<?xml version="1.0"  encoding="UTF-8"?>
<rss version="2.0">
<channel>
<title>Alerts Posted by ACMAD</title>
<link>http://www.acmad.org/alerts/rss.xml</link>
<description>Alerts posted by ACMAD (African Centre of Meteorological Applications for
Development)</description>
<language>en-us</language>
<copyright>public domain</copyright>
<pubDate>Fri, 14 Oct 2011 15:13:22 +0000</pubDate>
<docs>http://blogs.law.harvard.edu/tech/rss</docs>
<item>
<title>Geomagnetic Storm Alert</title>
<link>http://www.acmad.org/alerts/20111014150503.xml</link>
<description>There is likely to be a major geomagnetic storm and possible auroral activity
over the next few days. Space W eather sources at NOAA/NASA indicate that major solar flares
and a coronal mass ejection (CME) were observed at 9:30 a.m. Eastern Time on June
6.</description>
<author>echristian@usgs.gov</author>
<category>Met</category>
<guid>http://www.acmad.org/alerts/20111014150503.xml</guid>
<pubDate>2011-10-14T15:05:03-00:00</pubDate>
</item>
</channel>
</rss>
```

CAP feeds are typically available via RSS, Atom and ASN.1

See: http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2.html

Handbook 0.9 (work in progress)

BIBCOLON

COLONDOC for bibliographies

```
DOCTYPE
   |
METADOC
   |
COLONDOC
   v
BIBCOLON
```

Special fields are:

```
  Template: Class   // First field (mandatory)
  Handle: UniqueID  // 2nd field (mandatory)
  Name:             // Name associated with record
  Organization:     // Associated organization for name above
  Email:            // Internet email for name above
```

The UniqueID passed in Handle is used for the record key.

BIBTEX

DOCTYPE
^
BIBTEX

The native BIBTEX doctype class is for BibTeX bibliographic databases as used by the LaTeX package for the TeX-typesetting program.

The BIBTEX doctype supports TeX macro encodings of European characters, string substitution and hypertext links to "crossref" entries.

BibTeX is reference management software for formatting lists of references. The BibTeX tool is typically used together with the LaTeX document preparation system. Within the typesetting system, its name is styled as B I B T E X {\displaystyle {\mathrm {B{\scriptstyle {re-Isearch}}\!T\!_{\displaystyle E}\!X} }} {\mathrm {B{\scriptstyle {re-Isearch}}\!T\!_{\displaystyle E}\!X} }. The name is a portmanteau of the word bibliography and the name of the TeX typesetting software.
BibTeX has become one of the standard formats to store and share bibliographic data. Each BibTeX reference consist of three parts: the entry type, citekey, key-value pairs storing the bibliographic data.

NOTE: The engine does not support the @STRING construction.

```
@String{inst-LASL                = "Los Alamos Scientific Laboratory"}
```

during fielded search but gets resolved in presentation, e.g. an element using inst-LASL as value needs to be searched as such rather than “Los Alamos”.. during the presentation, however, the abbreviation does get resolved. This was a

design decision. Should one wish to have searches for "Los Alamos Scientific Laboratory" find also inst-LASL one can use the thesaurus facility. A script to process @STRING abbreviations into thesaurus entries should be quite trivial to implement.

✗ Example of a book:

```
@book{Robinson:1966,
      author    = "Robinson, Abraham, 1918-1974",
      title     = "Non-standard analysis",
      series    = “Studies in logic and the foundations of mathematics”,
      publisher = "North-Holland Pub. Co.",
      address   = "Amsterdam, NL",
      year      = 1966
    }
```

✗ Example of an article:

```
@Article{Finerman:1979:F,
  author =       "Aaron Finerman",
  title =        "Foreword",
  journal =      j-ANN-HIST-COMPUT,
  volume =       "1",
  number =       "1",
  pages =        "3--3",
  month =        jul # "\slash " # sep,
  year =         "1979",
  CODEN =        "AHCOE5",
  ISSN =         "0164-1239",
  ISSN-L =       "0164-1239",
  bibdate =      "Fri Nov 1 15:29:16 MST 2002",
  bibsource =
"http://www.math.utah.edu/pub/tex/bib/annhistcomput.bib",
  URL =          "http://dlib.computer.org/an/books/an1979/pdf/a1003.pdf;
                 http://www.computer.org/annals/an1979/a1003abs.htm",
  acknowledgement = ack-nhfb,
  fjournal =     "Annals of the History of Computing",
  journal-URL =  "http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?
punumber=5488650",
}
```

Handbook 0.9 (work in progress)

✗ BibTeX features 14 entry types:

Entry Type	Description
article:	Any article published in a periodical like a journal article or magazine article
book:	Book with designated published
booklet:	like a book but without a designated publisher
conference:	a conference paper
inbook:	Section or chapter in a book
incollection:	Article in a collection
inproceedings:	Conference paper (same as the conference entry type)
manual:	Technical manual
masterthesis:	Masters thesis
misc:	used if nothing else fits
phdthesis:	PhD thesis
proceedings:	whole conference proceedings
techreport:	technical report, government report or white paper
unpublished:	work that has not yet been officially published

Standard field types

- **address**: address of the publisher or the institution
- **annote**: an annotation
- **author**: list of authors of the work
- **booktitle**: title of the book
- **chapter**: number of a chapter in a book
- **edition**: edition number of a book
- **editor**: list of editors of a book
- **howpublished**: a publication notice for unusual publications
- **institution**: name of the institution that published and/or sponsored the report
- **journal**: name of the journal or magazine the article was published in
- **month**: the month during the work was published
- **note**: notes about the reference
- **number**: number of the report or the issue number for a journal article
- **organization**: name of the institution that organized or sponsored the conference or that published the manual
- **pages**: page numbers or a page range
- **publisher**: name of the publisher
- **school**: name of the university or degree awarding institution
- **series**: name of the series or set of books
- **title**: title of the work
- **type**: type of the technical report or thesis

- **volume**: volume number
- **year**: year the work was published

The BibTeX parser supports a number of additional keywords.

Currently recognizes the following keywords: `"ISBN", "ISSN", "LCCN", "URL", "abstract", "acknowledgement", "address", "affiliation", "annote", "availability", "author", "bibdate", "booktitle", "classification", "chapter", "coden", "confdate", "conflocation", "confsponsor", "copyright", "crossref", "edition", "editor", "howpublished", "institution", "journal", "key", "keywords", "language", "month", "note", "number", "organization", "pages", "pageswhole", "price", "publisher", "review", "school", "series", "subject", "thesaurus", "title", "type", "uniform", "volume" and "year"`

NOTE: The keyword "`key`" is quite special and unique to the engine in that it sets the record key.

This doctype can be used like the IKNOWDOC and ROADS++ doctypes for Whois++ services.
The MIME type for "Raw" records is "Application/X-BibTeX".

BINARY

The BINARY doctype has been designed for the management of multimedia files (audio/graphics), CAD drawings and non-textual databases whose content we don't want to try to directly index.

```
DOCTYPE
   v
BINARY
```

A "binary" file consists of 2 (two) components, a plain text description and the binary file itself. Looks for a file ,info that contains the text information (eg. for "x.tar" looks for "x.tar,info").

The first sentence in the ,info file is used to construct the "Title" field—as in FIRSTLINE. The Rest of the document is stored under the "Description" field.

The fields Title, Description and the whole document (Any Field) are searchable.

In the Presentation:

Title returns the first line.
The FULLTEXT ("F") attribute returns the binary file.
The field FULLPATH returns the complete path to the file.
The field BASENAME returns the basename of the binary file.
The Key is a unique name based on the basename of the binary file.

The default headline consists of the title and the type of the file in ().
The MIME type for depends upon the type of the binary file.

Built-in MIME bindings

Extension	MIME type	Extension	MIME type
NoExtension	*/*	.mpe	video/mpeg
Default	Application/Octet-Stream	.mpeg	video/mpeg
.ai	application/postscript	.mpg	video/mpeg
.aif	audio/aiff	.nvd	application/x-navidoc
.aifc	audio/aiff	.nvm	application/x-navimap
.aiff	audio/aiff	.pbm	image/x-portable-bitmap
.ani	application/x-navi-animation	.pdf	application/pdf
.arc	application/x-arc	.pgm	image/x-portable-graymap
.art	image/x-art	.pic	image/pict
.au	audio/basic	.pict	image/pict
.avi	video/x-msvideo	.pnm	image/x-portable-anymap
.bibtex	application/x-bibtex	.ps	application/postscript
.bin	application/x-macbinary	.qt	video/quicktime
.bmp	image/bmp	.ra	audio/x-pn-realaudio
.btx	application/x-bibtex	.ram	audio/x-pn-realaudio
.cpio	application/x-cpio	.ras	image/x-cmu-raster
.csv	application/csv	.refer	application/x-refer
.dcr	application/x-director	.rgb	image/x-rgb
.doc	application/x-msword	.rtf	application/rtf
.dir	application/x-director	.sgm	text/sgml
.dll	application/octet-stream	.sgml	text/sgml
.dp	application/commonground	.sit	application/x-stuffit
.dtd	text/sgml	.snd	audio/basic
.dvb	application/x-dvbline	.so	application/octet-stream
.dvi	application/x-dvi	.sql	application/x-sql
.dxr	application/x-director	.stl	application/x-navistyle
.eml	text/plain	.tar	application/x-tar
.eps	application/postscript	.tcl	text/plain
.exe	application/octet-stream	.tex	application/x-tex
.gif	image/gif	.text	text/plain
.gz	application/x-compressed	.tgz	application/x-compressed
.hqx	application/mac-binhex40	.tif	image/tiff
.htm	text/html	.tiff	image/tiff
.html	text/html	.txt	text/plain
.latex	application/x-latex	.voc	audio/basic
.lha	application/x-lharc	.xbm	image/x-xbitmap
.ltx	application/x-latex	.xpm	image/x-xpixmap
.java	application/x-java	.vrml	x-world/x-vrml

.jfif	image/jpeg	.wav	audio/x-wav
.jpe	image/jpeg	.wp4	application/x-wp4
.jpg	image/jpeg	.wp5	application/x-wp5
.jpeg	image/jpeg	.wks	application/x-lotus123
.js	application/x-javascript	.wrl	x-world/x-vrml
.ls	application/x-javascript	.xls	application/x-excel
.map	application/x-navimap	.Z	application/x-compressed
.mif	application/x-mif	.zip	application/x-zip-compressed
.mocha	application/x-javascript	.zoo	application/x-zoo
.mov	video/quicktime		

If compiled with the optional libmagic library it will, should it not have resolved the MIME type use it to try to determine an appropriate MIME type.

NOTE: The reason we look at the extension first is to allow for type spoofing as is commonly practiced by formats that use archivers such as ZIP (e.g. .jar files).

XBINARY

The XBINARY doctype is intended for indexing binary files.

DOCTYPE
|
BINARY
v
XBINARY

While the BINARY doctype uses plain text “,info” files to describe a resource (considered a “binary” file to indicate that its content is not being parsed for data), it instead, expects an “,info.xml” file. These files, encoded in XML, contain a description (metadata) of the file resource. The format of the XML file to describe the metadata is relatively free and up to the application. The only assumption is that there should be a “TITLE” element to describe the resource —an alternative element that would proxy as TITLE can, of course, be remapped to TITLE during indexing. The class allows a free choice of what XMLish doctype to use for parsing the ,info.xml” file—one can even use the FILTER2XML doctype to have a script (“filter”) create or modify the data passed to the parser.

- Uses the field TITLE for the default headline
- Uses the extension (as does BINARY) to determine the MIME type

Options:

```
XMLPATHS specifies if these should be stored (Default ON)
USECLASS specifies an alternative BASE class (default XML)
```

Note: XMLPATHS off sets the BASE class to XML. On set it to GILSXML which also handles TYPE= defines (see the documentation).

COLONDOC

Colon tagged documents are among the most commonly used form of ASCII markup. Field names are defined by, you guessed it, ':'.

COLONDOC Class Tree:

DOCTYPE
|
METADOC
v
COLONDOC

"COLONDOC": Colon document format files. Like "METADOC" but the default sep is ':'.

COLONDOC is not really (intended to be) a document type but a parent for this "major" class of document formats. The COLONDOC class has been designed to provide a convenient base class for the development of user document types.

Examples of children are: BIBCOLON, IADADOC, IKNOWDOC and through its child COLONGRP a number of other types such as DIF.

Colon Records:

```
TAG1: ...
.....
TAG2: ...
TAG3: ...
...
....
```

1. Fields are continued when the line has no tag
2. Field names **may NOT contain white space**. Although not explicitly required it is *recomended* that all field names use characters restricted to the the set of 7-bit ASCII characters excluding %(*per-cent*), $(*dollar*) and + (*plus*).
3. *Field names* are currently *case independent*, viz. *From*, *FROM* and *from* are all considered to name the same field.
4. The space BEFORE field names MAY contain white space
5. Between the field name and the ':' NO white space is allowed.

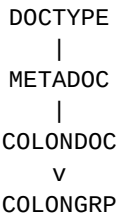
6. There is a compile time options to NOT allow white space before the start of the field name. While this makes life easier for continuation (one need not worry about formatting to prevent bogus fields) it forces a more rigid format.
7. There is no specific limitation on the length of a line. The maximal line and field length is given by the maximal size of a memory block defined by the O/S on the host platform. On most 32-bit Unix platforms this is around 2 GB.
8. Should the document contain several records the user must specify the record separator via a option to the Indexer (eg. -s " *****" for *Ziff CD* records)

Although the **COLONDOC** format is ill-suited to hierarchical data its COLONGRP child is.

COLONGRP

COLONGRP is a master colondoc doctype that supports some structure (sub-tags). If it inspired by, as well as parent of, DIF.

```
COLONGRP Class Tree:
```



The contents of the *first field* should be *unique*. It is used as a *unique* identification string for the record

The Field Name `Group` is a *reserved name*. It supports some structure via *Groupings*.

```
Group: GroupName
        Field:  Value
        Field:  Value
        Field:  Value
End_Group
```

Groups may contains groups. For each `Group` one must specify `End_Group` to inform the parser where the end of the group is.

```
Field: Value
Field: Value
Group: GroupName
       Field:  Value
       Field:  Value
       Group: GroupName
              Field:  Value
              Field:  Value
```

```
               Field:  Value
       End_Group
End_Group
```

Its probably best understood by looking at the example fragment below.

```
Entry_ID: 1013DS-A.cdf
Group:  Technical_Contact
        First_name:  Frances
        Middle_name:  S.
        Last_name:  Hotchkiss
        Phone:  508-457-2242
        Phone:  FAX 508-457-2310
        Group:  Address
                384 Woods Hole Road
                Quissett Campus
                Woods Hole, MA  02543-1598
                USA
        End_Group
End_Group
```

`Entry_ID`
: 1013DS-A.cdf would specify `1013DS-A.cdf` as the *unique key* for the record.

`Technical_Contact`
: contains everything from `First_name` to the `End_Group` in Address.

`First_name`
: contains `Frances`

`Middle_name`
: contains `S.`

`Last_name`
: contains `Hotchkiss`

`Phone`
: contains `508-457-2242` and `FAX 508-457-2310`

`Address`
: contains everything from `384 Woods Hole Road` to `USA`

Compared to SGML/XML:

Group: Level1
  Group: Level2
  tag1: tag1_value
  End_Group
End_Group
	<Level1>
	 <Level2>
	  <tag1>tag1_value</tag1>
	 </Level2>
	</Level1>

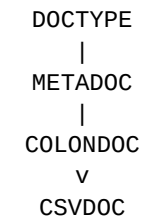
The MIME type for *Raw Records* is `Application/X-COLONGRP`.

Handbook 0.9 (work in progress)

CSVDOC

The native CSVDOC doctype class is for "Comma Separated Values" such as those exported from Claris FileMaker Pro (Merge), MS Access, Excel and other popular PC programs. Like its ancestors COLONDOC and METADOC it is a key-value type but, in contrast to them, defined along the colon rather than row.

CSVDOC Class Tree:



The doctype is especially well suited to the export of metadata records from PCs to provide GILS and other network services. Although many US Federal agencies, especially the smaller less budgeted institutions, tend to deploy personal computers (Windows, Win'95, WinNT and Mac) these platforms are ill-suited to network services. It is also desirable for security to de-couple the data collection from the Internet. The CSVDOC doctype allows for the continued use of familiar PC tools but with more robust, secure and powerfull Z39.50 and WWW services.

A CSLDOC looks like

```
Field0;Field1;Field2
"Field0";"Field1";"Field2"
```

i. The first line contains the field names.
ii. The following lines contain the contents of each field.
iii. The content lines may or may not have ""
iv. The field seperator may be some other character or even sequences of characters. Whatever they are they are consistent.

Example (From Claris FileMaker Pro):

```
ID;Artistpage;Sortiername;Artist;Artistphoto;Kuenstlername;LebenD;LebenE;AtelierD;AtelierE;E
inzelaustellungD;EinzelaustellungE;OefentlicheArbeitenD;OefentlicheArbeitenE
"A139";"K";"Aenderungsatelier Schweitzer & Stemmer";"Žnderungsatelier Schweitzer &
Stemmer";"";"Žnderungsatelier Schweitzer & Stemmer";"";"";"";"";"";"";""
"A104";"K";"Amey, Paul";"Amey, Paul";"";"Paul Amey";"";"";"";"";"";"";""
"A115";"K";"Doubrawa, Reinhard";"Doubrawa, Reinhard";"";"Reinhard
Doubrawa";"";"";"";"";"";"";""
"A124";"K";"Droese, Felix";"Droese, Felix";"";"Felix  Droese";"";"";"";"";"";"";""
"A183";"K";"Dumas, Marlene";"Dumas, Marlene";"";"Marlene Dumas";"";"";"";"";"";"";""
"A184";"K";"Emin, Tracey";"Emin, Tracey";"";"Tracey Emin";"";"";"";"";"";"";""
"A185";"K";"Esser, Uwe";"Esser, Uwe";"";"Uwe Esser";"";"";"";"";"";"";""
"A186";"K";"Eubel, Edgar A.";"Eubel, Edgar A.";"";"Edgar A. Eubel";"";"";"";"";"";"";""
"A187";"K";"Fabro, Luciano ";"Fabro, Luciano ";"";"Luciano Fabro";"";"";"";"";"";"";""
"A188";"K";"Fairhurst, Angus";"Fairhurst, Angus";"";"Angus
Fairhurst";"";"";"";"";"";"";""
```

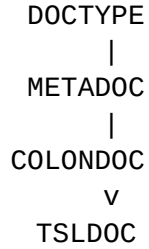
```
"A108";"K";"Fischer, Urs";"Fischer, Urs";"";"Urs Fischer";"";"";"";"";"";"";"";""
"A189";"K";"Fischli, Peter/ Weiss David";"Fischli, Peter/ Weiss David";"";"Peter Fischli/
David Weiss";"";"";"";"";"";"";"";""
"A129";"K";"Flinzer, Jochen";"Flinzer, Jochen";"";"Jochen Flinzer";"";"";"";"";"";"";"";""
"A191";"K";"Frank, Dinah";"Frank, Dinah";"";"Dinah Frank";"";"";"";"";"";"";"";""
"A192";"K";"Friedmann Gloria";"Friedmann Gloria";"";"Gloria
Friedmann";"";"";"";"";"";"";"";""
"A193";"K";"Fritsch, Katharina";"Fritsch, Katharina";"";"Katharina
Fritsch";"";"";"";"";"";"";"";""
"A194";"K";"Gabriel, Twin";"Gabriel, Twin";"";"Twin Gabriel";"";"";"";"";"";"";"";""
"A195";"K";"Geldmacher, Klaus";"Geldmacher, Klaus";"";"Klaus
Geldmacher";"";"";"";"";"";"";"";""
"A196";"K";"Gilbert & George";"Gilbert & George";"";"Gilbert &
George";"";"";"";"";"";"";"";""
"A197";"K";"Gillick, Liam";"Gillick, Liam";"";"Liam Gilick";"";"";"";"";"";"";"";""
"A198";"K";"Gober, Robert";"Gober, Robert";"";"Robert Gober";"";"";"";"";"";"";"";""
"A199";"K";"Gonschior, Kuno";"Gonschior, Kuno";"";"Kuno Gonschior";"";"";"";"";"";"";"";""
"A201";"K";"Gormly, Antony";"Gormly, Antony";"";"Antony Gormly";"";"";"";"";"";"";"";""
"A200";"K";"Gostner, Martin";"Gostner, Martin";"";"Martin Gostner";"";"";"";"";"";"";"";""
"A202";"K";"Granda, Alonso /M. Elena";"Granda, Alonso /M. Elena";"";"Alonso Granda /Elena
M.";"";"";"";"";"";"";"";""
"A203";"K";"Groá, Sabine";"Groá, Sabine";"";"Sabine Groá";"";"";"";"";"";"";"";""
"A204";"K";"Grubinger, Eva";"Grubinger, Eva";"";"Eva Grubinger";"";"";"";"";"";"";"";""
"A215";"K";"Hurst, Annette";"Hurst, Annette";"";"Annette Hurst";"";"";"";"";"";"";"";""
"A131";"K";"Ingold Airlines ";"Ingold Airlines";"";"Ingold Airlines";"";"";"";"";"";"";"";""
"A126";"K";"Innocente";"Innocente";"";"Innocente";"";"";"";"";"";"";"";""
"A216";"K";"Irwin";"Irwin";"";"Irwin";"";"";"";"";"";"";"";""
"A217";"K";"Jaxi, Constantin";"Jaxi, Constantin";"";"Constantin
Jaxi";"";"";"";"";"";"";"";""
"A218";"K";"Kabakov, Ilya";"Kabakov, Ilya";"";"Ilya Kabakov";"";"";"";"";"";"";"";""
"A219";"K";"Kane, Alan";"Kane, Alan";"";"Alan Kane";"";"";"";"";"";"";"";""
"A220";"K";"Katase, Kazuo";"Katase, Kazuo";"";"Kazuo Katase";"";"";"";"";"";"";"";""
"A222";"K";"Keikeik";"Keikeik";"";"Keikeik";"";"";"";"";"";"";"";""
"A272";"K";"Schmidt, Enno";"Schmidt, Enno";"";"Ennno Schmidt";"";"";"";"";"";"";"";""
"A273";"K";"Schmidt, Pavel";"Schmidt, Pavel";"";"Pavel Schmidt";"";"";"";"";"";"";"";""
"A274";"K";"Schneider, Rolf";"Schneider, Rolf";"";"Rolf Schneider";"";"";"";"";"";"";"";""
"A275";"K";"Schnyder, Achim";"Schnyder, Achim";"";"Achim Schnyder";"";"";"";"";"";"";"";""
"A276";"K";"Scholz, Norbert";"Scholz, Norbert";"";"Norbert Scholz";"";"";"";"";"";"";"";""
"A305";"K";"West, Franz";"West, Franz";"";"Franz West";"";"";"";"";"";"";"";""
"A306";"K";"Whiteread, Rachel";"Whiteread, Rachel";"";"Rachel
Whiteread";"";"";"";"";"";"";"";""
"A307";"K";"Wiegand, Suse";"Wiegand, Suse";"";"Suse Wiegand";"";"";"";"";"";"";"";""
"A313";"K";"Zangs, Herbert";"Zangs, Herbert";"";"Herbert Zangs";"";"";"";"";"";"";"";""
```

Since the CSV file format is not fully standardized the CSVDOC tries to be clever and determine the CSV format. The format is closely related to a number of other delimiter-separated formats that use other field delimiters such as tab characters and semicolons. A delimiter such as tab that is not present in the field data allows simpler format parsing. See the TSLDOC format.

Handbook 0.9 (work in progress)

TSLDOC

Class Tree:



TSLDOC is a base class for Tab Separated List (Tab delimited) files:

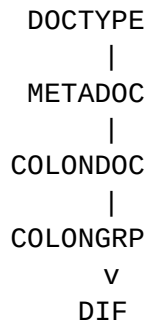
- Each line is a record
- Each line contain fields separated from each other by TAB characters (horizontal tab, HT, Ascii cc 9).
- "Field" means here just any string of characters, excluding TABs.
- Each line must contain the same number of fields.
- The first line may contain the names for the fields (on all lines), i.e. column headers.

It supports the following options:

```
[General]
 TabFields=Comma separated list of field names (e.g. field1,field2,field3)
   alternatively in the doctype.ini to define the "column" field names.
   If the option "UseFirstRecord" is specified (True) it is used
   as the fieldnames.
   If no TabFields are specified and UseFirstRecord was not specified
   True of False then UseFirstRecord will be assumed as True.
 CategoryField=<Field contains an integer for record category>
 PriorityField=<Field contains an integer to skew scores for record>
   For priority to have an effect one must define in "Db.ini"'s
   [DbInfo] PriorityFactor=NN.NN (a floating point number)
Alternatively in the "Db.ini" file [Doctype] section
```

DIF

Class Tree:



Handbook 0.9 (work in progress)

The Directory Interchange Format (DIF) is a de-facto standard used to create directory entries which describe a group of data. Its use is wide spread among US government agencies. A DIF record consists of a collection of fields which detail specific information about the data.
From the point of view of syntactical structure it is a heirarchical COLONDOC-based document format. The DIF class has been implemented as a child of COLONGRP.

A DIF record starts with a `Entry_ID:` followed by a **unique** identification string for the record (Handle), eg:
Entry_ID: 1013DS-A.cdf
The Field Name `Group` is a reserved name. It supports some structure via Groupings.

```
Group: GroupName
        Field:  Value
        Field:  Value
        Field:  Value
End_Group
```

Groups may contains groups. For each `Group` one must specify `End_Group` to inform the parser where the end of the group is.

Its probably best understood by looking at the example record below.

```
Group:  Technical_Contact
        First_name:  Frances
        Middle_name:  S.
        Last_name:  Hotchkiss
        Phone:  508-457-2242
        Phone:  FAX 508-457-2310
        Group:  Address
                384 Woods Hole Road
                Quissett Campus
                Woods Hole, MA  02543-1598
                USA
        End_Group
End_Group
```

✗ The fields

`Technical_Contact`
: contains: everything from First_name to the End_Group in Address.

`First_name`
: contains: Frances

`Middle_name`
: contains: S.

`Last_name`
: contains: Hotchkiss

`Phone`
: contains: 508-457-2242 and FAX 508-457-2310

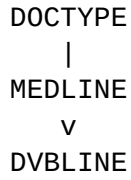
`Address`
: contains: everything from 384 Woods Hole Road .. to .. USA

The Brief_Record (Headline) is the contents of the Entry_Title field.

The MIME type is: `Application/X-DIF`

DVBLINE

`DVBLINE Class Tree:`



DVBLINE is a Medline family document format based upon a molestation of `Filmline` 1.x.

It provides the basic interchange format for the DVB pilot, a project to develop a cooperative model for database record-ship amongst the independent state German language media clearing houses.

The DVB software was developed for the Landesmedienzentrale (State Media Central Authority) Koblenz under contract of the State of Rheinland-Pfalz back in the mid 1990s.

In addition to the features of the `FILMLINE` Document handler it also supports sub-tags and "automatic" generation of a hypertext Web around the embeded background documents.

Sub-tags are specified with colon notation such as that provided by `COLONDOC`.

The Key for the record is derived from the DVB-Id number of the record.

A multiple-platform GUI workplace for the workflow has been developed in wxPython. This script driven program handles communication (flow) with a WWW workflow server, assists and shields the user from the details of the format (metarecord authorship), provides validation services and allows for import/export of other formats used in German education by decentral authorities.

DVBLine "tag" semantics

Field Tag	Attribute (Descriptive Name)	German Attribute Name
AB	abstract	Abstract

AD	audience	Adressat
AU	author	Buch
AW	awards	Awards
CG	camera	Kamera
CL	clue	Verl.Schluessel
CO	country	Produktionsland
CR	critics	Critics
CS	Character Set	CharacterSet
DA	date collected	+Datum-Erfassung
DE	description	Beschreibung
DI	director	Regie
DK	documentation	Dokumentation
DT	distribution	Vertrieb
ED	editor	Editor
FP	fee purchase	Kaufpreis
FR	fee rent	Mietpreis
GE	genre	Genre
GM	fee GEMA	GEMA
I0	DVB Record Status (Internal)	Status
I1	record author (Internal)	+Erschliesser
I2	Expert (Internal)	+Potentieller.Erschliesser
I3	external_links_exist (Internal Boolean)	+B.Hintergrunddok
I4	record_technical_author (Internal)	+Zustaendiger.Facherschliesser
I5	parallel_title	Paralleltitel
I6	agency	Auftraggeber
I7	local-collection	+Kaeufer
I8	FWU-call-id	FWU-Signatur
I9	FRG-Unified-call-id	BundesSig
re-Isearch	remarks	Bemerkung
IC	size	Anzahl
IE	external_links_public (Supplementary Material)	Hintergrunddok
IF	external_links (Secret)	+Hintergrunddok.geheim
IG	writers	Weitere-Urheber
KW	keywords	Schlagwoerter
LA	code-language	Sprache
LN	length	Laufzeit
LO	location	Standort
LT	lit_author	Vorlage
MD	color	Farbe
MM	media	Media
MU	music	Musik
NA	thesaurus-motbis	MOTBIS
NB	thesaurus-tee	TEE
PO	parent	Sammelmedien

PR	producer	Realisator
PS	commentary	Beurteilung
PU	publisher	Herausgeber
RA	ratings (FSK)	FSK
SA	collections	Sammlung
SD	sound	Ton
SE	series titel	Serientitel
SH	sort_title	Sortierhaupttitel
SI	local-call-id	ID-Nummer
SL	see also languages	Sprachfassungen
SM	see also media	Einzelmedien
SO	See Also	Kontextmedien
SS	series_sort_title	Sortierserientitel
ST	other-credits	Mitwirkende
SU	sort_subtitel	Sortieruntertitel
SY	systematic	Sachgebiete
TD	technical_data	Technische-Angaben
TE	title-uniform	Haupttitel
TI	title-cover	Wahrer-Titel
TO	original language title	Originaltitel
TU	title-caption	Untertitel
UA	UDC	Klassifikation-UDC
UB	Dewey	Klassifik-Dewey
VL	volume	Volume
VT	Synchronisation (Dub)	Synchronisation
YR	date-of-publication	Produktionsjahr
ZC	ILL-Code	ILL-Code
ZL	Language.Code	Language.Code
ZR	date-last-modified	CDAT
ZU	URL	URL
ZZ	misc	Verschiedenes

The MIME type for Raw is "`Application/X-DVBLine`"

EUROMEDIA

EUROMEDIA is multinational mediagraphic format based upon COLONGRP originally used to create and exchange national language multimedia metarecords within the EU-sponsored EUROMEDIA project.

```
DOCTYPE
   |
METADOC
   |
```

```
COLONDOC
    |
COLONGRP
    v
EUROMEDIA
```

Since the project had participating partners in Germany, France, Italy and Spain with record languages of, resp., German, French, Italian and Spanish it was designed to be both multi-lingual (supporting a national language) and fully human readable. The conversions from/to national language is handled via the “Unified Field” architecture which allows field names (and paths) can to be mapped during indexing to alternative names. During presentation using the requested language code these records (created on-the-fly) can have their keys remapped to names in a local vocabulary.

From the point of view of syntactical structure it is a heirarchical colondoc-based document format. The `EUROMEDIA` class has been implemented as a child of COLONGRP.

A `EUROMEDIA` record starts with a Local_number: followed by a unique 8-digit identification number for the record (Handle), eg:

```
Local_number: 00000024
```

Although the doctype supports any other unique key, the workflow solution and charter specify an 8-digit number. The Field Name Group is a reserved name. It supports some structure via Groupings.

```
Group: GroupName
        Field:  Value
        Field:  Value
        Field:  Value
End_Group
```

Groups may contains groups. For each Group one must specify End_Group to inform the parser where the end of the group is.

Default Language Field is `'Code_language'`
Default Date Field is `'Date_last_modified'`

The Brief Record (Headline) is defined, by default, by ‘Title’ (if not then ‘Title_Series’ then ‘Title_other_variant’).

Since it is multinational it supports alongside the `Dewey_classification number`, `Subject_TEE`, `Subject_MOTBIS` (as is widely used in French Education currently enriched and updated by Réseau Canopé) and `Local_subject_index`.

MISMEDIA

DOCTYPE
|
METADOC
|
COLONDOC
|
COLONGRP
v
MISMEDIA

MIS (Rheinland-Pfalz) colondoc mediagraphic records. This is an enhanced version of DVBLINE.

FILMLINE

Filmline Description:

```
Doctype Classes
      |
      |
   Doctype (Generic Document Type)
      |
      |
   Medline
      |
      |
   Filmline
```

The FILMLINE doctype is for filmline v 1.x records (version 2.x is based upon a XML/SGML DTD). It was developed by BSn and JFF (JFF - Institut für Medienpädagogik in Forschung und Praxis / JFF – Institute for Media Research and Media Education) for interchange of mediagraphic records and as a basis for database publishing for their printed film catalogue. It was based heavily upon the Medline format.

FILMLINE 1.x is currently being used in several European mediagraphic projects. In a modified form it also provides the interchange format for the German National DVB project as DVBLINE.

Filmline 1.x "tag" semantics

Field Tag	Attribute (Descriptive Name)
AB	abstract
AD	address
AU	author
AW	awards
CG	camera

CL	clue
CO	country
CR	critics
CS	charset
DA	date_collected
DE	description
DI	director
DK	documentation
DT	distribution
ED	editor
FD	distribution
FP	fee-purchase
FR	fee-rent
GE	genre
GM	GEMA
LA	language
KW	keywords
LN	length
LT	lit_author
MD	media-type
MM	Content-type
MU	music
PO	parent
PR	producer
PS	ps
PU	publisher
RA	ratings
SA	collections
SE	series
SH	sort-title
SS	series-title
SI	signature
SL	alt_lang
SM	children
SO	other-source
ST	other-credits
SU	subtitle
SY	systematic
TD	technical-data
TE	title-uniform
TI	title-cover
TU	title-caption
VL	volume
VT	sync

YR	date
ZU	URL

The MIME type: "Application/X-Filmline"

FIRSTLINE

FIRSTLINE Class Tree:

```
                                        DOCTYPE
                                           v
                                       FIRSTLINE
```

An extremely simple but also extremely useful and flexible document format. We often have fully unstructured text files that we want to index and search and for their indentification use the first (or some other specified) line or sentence to identify them in results. For the Brief Record (Headline) it uses the Nth line or sentence as specified.

Index-time options:

```
  Startline N    Specifies which line or sentence to use (default is 1)
```

XFILTER

The re-Isearch engine can be extended to support the widest range of data inputs through the use of external programs, so called filters. There programs should take a single argument as the path to the input file and should write their output in the specific format to standard output (stdout).

```
filter_path filename_path
```

Out-of-the box the engine provides doctypes to handle filtering to HTML, MEMO, TEXT and XML formats. The most generic is the XFILTER doctype.

XFILTER uses an external filter to process input to any DOCTYPE.

Class Tree:

```
                                        DOCTYPE
                                           v
                                        XFILTER
```

Options:

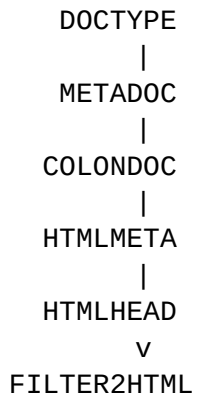
```
  FILTER   Specifies the filter program to use
  CLASS    Specifies which doctype class to use to handle the filtered records
```

Note:
Filters must take single arguments of the form: filter filename
and write to stdout. For output one should also define External/ filters (see DOCTYPE class)

FILTER2HTML

FILTER2HTML uses an external filter that converts the input file into HTML type files. It has similar goals to XFILTER where the class was set to pass processing to HTMLMETA.

Class Tree:



Options:

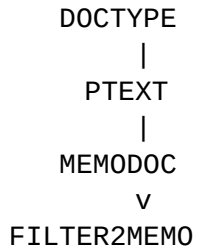
Filter	Specifies the program to use.
Content-type	Specifies the MIME content-type of the original.

These are defined in the "filter2html.ini" [General] or <Db>.ini [FILTER2HTML] sections.

Filters should take a single argument as the path to the (binary) input file and should write their output (HTML) to standard output (stdout)

FILTER2MEMO

FILTER2MEMO uses an external filter that converts the input file into MEMO type files. It has similar goals to XFILTER where the class was set to pass processing to MEMODOC.



Options:

Filter	Specifies the program to use.
Content-type	Specifies the MIME content-type of the original.

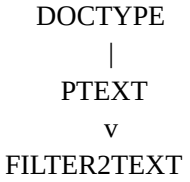
These are defined in the "filter2memo.ini" [General] or <Db>.ini [FILTER2MEMO] sections.

Filters should take a single argument as the path to the (binary) input file and should write their output (MEMO) to standard output (stdout)

FILTER2TEXT

FILTER2TEXT uses an external filter that converts the input file into TEXT (PTEXT) type files. It has similar goals to XFILTER where the class was set to pass processing to PTEXT.

Class Tree:



Options:

```
   Filter        Specifies the program to use.
   Content-type  Specifies the MIME content-type of the original.
```

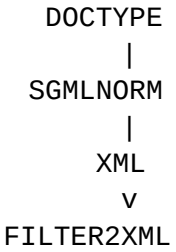
These are defined in the "filter2text.ini" [General] or <Db>.ini [FILTER2TEXT] sections.

Filters should take a single argument as the path to the (binary) input file and should write their output (PTEXT) to standard output (stdout)

FILTER2XML

FILTER2XML uses an external filter that converts the input file into XML type files. It has similar goals to XFILTER where the class was set to pass processing to XML.

Class Tree:



Options:

```
   Filter        Specifies the program to use.
   Content-type  Specifies the MIME content-type of the original.
```

These are defined in the "filter2xml.ini" [General] or <Db>.ini [FILTER2XML] sections.

Filters should take a single argument as the path to the (binary) input file and should write their output (XML) to standard output (stdout)

FTP

DOCTYPE
|
METADOC
|
COLONDOC
|
IAFADOC
v
FTP

While FTP archives are no only that widely used their management is still important. That is where the FTP doctype comes in.

The FTP doctype has been designed to support the distribution of binary data (via Z39.50 or HTTP) from FTP archives that adhere to the guidelines for FTP publication from the IAFA-WG. Like the BINARY doctype it has been designed to manage software archives, CAD drawings and non-textual information. The FTP document model offers much more control than the Binary model, but is a bit more complicated.

A "FTP" file consists of 2 (two) components, a IAFA (Internet Anonymous FTP Archives) metadata and the binary file itself. Looks for a file .iafa that contains the text information (eg. for "x.tar" looks for "x.tar.iafa"). All the fields in the IAFA description are searchable.

The FULLTEXT ("F") attribute returns the binary file.
The MIME type for Raw is specified in the IAFA file with the field Content-type. If not defined the default is "`Application/Octet-Stream`".

`See also:` P. Deutsch, A. Emtage, M. Koster and M. Stumpf, *Publishing Information on the Internet with Anonymous FTP (IAFA Templates)*, IETF IAFA WG Internet Draft, January 1995, <URL:ftp://nic.merit.edu/documents/internet-drafts/draft-ietf-iiir-publishing-03.txt>.

HARVEST / SOIF

SOIF format files and points to resource.

HARVEST Class Tree:

DOCTYPE
|

Handbook 0.9 (work in progress)

SOIF
v
HARVEST

HARVEST was an integrated set of tools to gather, extract, organize, and search information across the Internet.

SOIF

```
The SOIF (Summary Object Interchange Format) Grammar is as follows:
    SOIF            ::=  OBJECT SOIF | OBJECT
    OBJECT          ::=  @ TEMPLATE-TYPE { URL ATTRre-IsearchUTE-LIST }
    ATTRre-IsearchUTE-LIST  ::=  ATTRre-IsearchUTE ATTRre-IsearchUTE-LIST |
ATTRre-IsearchUTE
    ATTRre-IsearchUTE       ::=  IDENTIFIER {VALUE-SIZE} DELIMITER VALUE
    TEMPLATE-TYPE   ::=  Alpha-Numeric-String
    IDENTIFIER      ::=  Alpha-Numeric-String
    VALUE           ::=  Arbitrary-Data
    VALUE-SIZE      ::=  Number
    DELIMITER       ::=  ":<tab>"
```

Some common fields

Abstract
: Brief abstract about the object.

Author
: Author(s) of the object.

Description
: Brief description about the object.

File-Size
: Number of bytes in the object.

Full-Text
: Entire contents of the object.

Gatherer-Host
: Host on which the Gatherer ran to extract information from the object.

Gatherer-Name
: Name of the Gatherer that extracted information from the object. (eg. Full-Text, Selected-Text, or Terse).

Gatherer-Port
: Port number on the Gatherer-Host that serves the Gatherer's information.

Gatherer-Version
: Version number of the Gatherer.

Update-Time
: The time that Gatherer updated the content summary for the object.

Keywords
: Searchable keywords extracted from the object.

Last-Modification-Time
: The time that the object was last modified.

MD5
: MD5 16-byte checksum of the object.

Refresh-Rate
: The number of seconds after Update-Time when the summary object is to

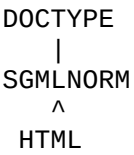
be re-generated. Defaults to 1 month.
Time-to-Live
The number of seconds after Update-Time when the summary object is no longer valid. Defaults to 6 months.
Title
Title of the object.
Type
The object's type.

SOIF is an alternative to IAFA. It has been widely replaced by RDF but a number of projects still use it.

HTML

The re-Isearch engine contains a number of doctypes suitable for indexing HTML. For general use most projects will use the doctypes focusing on the metadata rather than the tags and elements in the body of the HTML document as these tend to convey little information.

The HTML Doctype is for special documents marked-up using the Hypertext Markup Language of the World Wide Web (WWW). For general indexing of HTML document the HTMLHEAD doctype is strongly advised.



Although the HTML doctype is a subclass of SGMLNORM it does not require normalized HTML and handles most all HTML Markup (tag normalized or not) and entities, e.g. Ü (Ü), &amp (&), &#177 (±) etc.
The doctype was designed to support HTML 0.9, 2.0, 3.0, HTML 3.2, W3O's Cougar (4.0) as well as *many* vendor extensions, kludges and abuses. Most common incorrect uses of HTML markup are also correctly handled. It has been tested against a very large random sample of several gigabytes of HTML data from thousands of sites and has been used to in the field to index many terabytes of HTML pages at hundreds of sites. It does not handle CSS, Javascript and some HTML 5 constructs that build around it (like Canvas).

By default it indexes *all* fields that *represent content* (*container tags*) in an HTML document. Descriptive markup, such as <I> ,<B>,<TT> etc. are ignored.

Parser Levels

The parser knows several levels:

Basic (5)
Accept only a few tags and combine into some simple groups with plain english names. This level is for robots and web services.
Valid (4)
Accept only valid HTML, make less assumptions and complain.

Strict (3)
: Accept only known HTML tags, complain about bogus use.

Normal (2)
: Accept only known HTML tags

Maximal (1)
: Accept all tags except some descriptive markup.

Full (0)
: Accept anything.

Antihtml (-1)
: Use HTML-- for processing.

The level is specified as a *doctype option* (eg. -o LEVEL=*Level*) or via *LEVEL* in the .ini *HTML* section. Either the long name or number can be specified.

META

The *META* tag is used to embed metadata in a HTML document. It belong in the *HEAD* of the document (since much HTML is non-comformant the HTML parser will accept it anywhere).
In HTML 2.0 the *META*s may occur anywhere in the *HEAD*

```
<!ENTITY % head.content "TITLE & ISINDEX? & BASE? %head.extra">

<!ELEMENT HEAD O O  (%head.content) +(META|LINK)>
```

But in HTML 3.2 all the *META*s must be grouped together.

```
<!ENTITY % head.content "TITLE & ISINDEX? & BASE? & STYLE? &
                         SCRIPT* & META* & LINK*">

<!ELEMENT HEAD O O  (%head.content)>
```

HTML 4.0 specified META as a complex empty tag via a name/value pair: *NAME* and *CONTENT*.

```
<!ELEMENT META - O EMPTY>
<!ATTLIST META
        HTTP-EQUIV  NAME    #IMPLIED
        NAME        NAME    #IMPLIED
        CONTENT     CDATA   #REQUIRED    >
```

The HTML document type definition offers very little content structure making the development of structured databases difficult. The *META*-tag mechanism offers some relief.
Meta tags, <META NAME="AUTHOR" CONTENT="Elmer Fudd"> are *searchable* (in versions >1.9) as META.AUTHOR

```
  <META NAME="AUTHOR" CONTENT="Elmer Fudd"> -->
     META@         := NAME="AUTHOR" CONTENT="Elmer Fudd"
     META@NAME     := "CONTENT"
     META@CONTENT := "Elmer Fudd"
```

```
  Version > 1.9
     META.AUTHOR  := "Elmer Fudd"
```

In Version >1.10 <META HTTP-EQUIV="AUTHOR" CONTENT="Elmer Fudd">

```
  <META HTTP-EQUIV="AUTHOR" CONTENT="Elmer Fudd"> -->
     META@        := HTTP-EQUIV="AUTHOR" CONTENT="Elmer Fudd"
     META@NAME    := "CONTENT"
     META@CONTENT := "Elmer Fudd"
     AUTHOR       := "Elmer Fudd"
```

Versions based upon SGMLNORM >1.8 support *Schema* and *Type* specified in content, viz. `<META NAME="LANGUAGE" CONTENT="(Schema=ISO6639)en">` searchable under *META.LANGUAGE(ISO6639)*. The keywords *Schema*, *Scheme* and *Type* are currently supported.

- `<META NAME="AAA" CONTENT="(Schema=BBB)DDD>` maps to *META.AAA(BBB)*.
- `<META NAME="AAA" CONTENT="(Type=CCC)DDD>` maps to *META.AAA.CCC*.
- `<META NAME="AAA" CONTENT="(Schema=BBB)(Type=CCC)DDD>` maps to *META.AAA.CCC(BBB)*.
- `<META NAME="AAA" CONTENT="(Schema=BBB)(Type=CCC)(Schema=DDD)EEE>` maps to *META.AAA.CCC(BBB.DDD)*.
- `<META NAME="AAA" CONTENT="(Type=BBB)(Type=CCC)DDD>` maps to *META.AAA.BBB.CCC*.

HTML 4.x extends *META* to:

```
<!ATTLIST META
  %i18n;                          -- lang, dir for use with content string --
  http-equiv  NAME       #IMPLIED  -- HTTP response header name  --
  name        NAME       #IMPLIED  -- metainformation name --
  content     CDATA      #REQUIRED -- associated information --
  scheme      CDATA      #IMPLIED  -- select form of content -- >
```

The definition of `Scheme=` overrides the specification of Scheme in Content, eg. `CONTENT="(Scheme=..)..."`.
This allows the development of interfaces to search for HTML pages based on meta-data, viz. content that HTML has no container for.
See also: META HTTP-EQUIV and NAME equivalences.

BASE

The doctype option or environment variable `WWW_ROOT` is used to synthesize— should it *not* have been defined— the value for <BASE HREF="..." ... >. This enables CGI interfaces (or clients) to "derive" a fully-qualified URL to the original document and resolve relative hypertext references.
In Version >1.10:

1. The .dbi/.ini file is examined for a *Pages* entry in the *HTTP* section.
2. else if the doctype option *WWW_ROOT* is defined it is used
3. else if the doctype option *HTTP_PATH* is defined it is used

4. else if the doctype option *HTTP_PATH* it is used
5. else if the doctype option *HTDOCS* is defined it is used
6. else the environment is checked for the above variables, processed in that order.

Should the path specified in `<BASE HREF=path>` have a trailing /, then the filename is concatinated. This way one may specify the same `BASE` for all files in the same directory.

The `URL` is searchable via the `BASE` field. Since binary data such as images, audio and video files *often* have descriptive names, eg. `ElmerFudd.au`, this can also provide a pragmatic basis for search of multimedia information.

In Version >1.10 one can also specify the name/port of the WWW server. This is designed to support multiple virtual hosts or mapping to PURL (Persistant URL) resolver.

1. The .dbi/.ini file is examined for a *Server* entry in the *HTTP* section. A URL of type `http://www.bsn.com:8080` is expected.
2. If the .dbi/.ini file does not contain an entry then a *HTTP_Server* doctype option is expected.
3. If this is not defined then the standard HTTP Server environment variables *SERVER_NAME* and *SERVER_PORT* are used to build the URL. If these are undefined then one is probably not running under a HTTPD and a `file:///` method based URL is contructed (only HTML).

An interesting option is to set the *Server* to point to a PURL resolver instead of the Web server. This way the resource URL can persist beyond, or be de-coupled from, the URL structure within a web.

LINK

The complex values of LINK are, like META stored. The REV and REL values can be used to model external flows to and from the HTML document instance.

I18N:

Support for i18n

```
<!ENTITY % i18n
 "lang        NAME       #IMPLIED  -- RFC 1766 language value --
  dir         (ltr|rtl)  #IMPLIED  -- default directionality --" >
```

is not yet available. It is currently in development (it may already be available but was not yet ready at the time this handbook section was being authored).

Notes:

The HTML Doctype is NOT a validator and makes many assumptions. It tries to guess the intent of some of the HTML kludges in common use. It is, none-the-less, advisable to validate HTML pages and to adhere to a content model.

HTML-- / ANTIHTML:

DOCTYPE

```
    |
SGMLNORM
    |
  HTML
    v
ANTIHTML
```

Like HTML but not indexing the standard tags but only meta and title elements. In general the HTMLHEAD or HTMLZERO are the preferred document handlers but there are some use cases where HTML-- is called for.

HTMLMETA

DOCTYPE
  |
METADOC
  |
COLONDOC
  v
HTMLMETA

Metadata is data *about* an information resource. It is generally structured into a number of elements. There are many different attribute sets and standards for metadata. The HTMLMETA Doctype has been designed to help create and manage meta-databases derived from documents marked-up with Meta-data using the Hypertext Markup Language (HTML) of the World Wide Web (WWW).

The point of embeding the meta data record into the HTML source is that it is scalable from simplistic low cost solutions up to full blown sophisticated editorial systems. The issue is pragmatics and encoding the meta record in HTML holds the chance of being widely accepted.
From the viewpoint of gathering meta-records there is little difference (other than cost and data consistency) between human coded and machine produced copy.

The **HTMLMETA** doctype has been designed to support a flexible model for the encapsulation of generic metadata and is equally well suited to Dublin Core, GILS, EAD and other standards.
The documents processed by the doctype are HTML with meta-information embeded in the <HEAD>...</HEAD> of a document using attribute tags rather than container element content. This form is compatible with the HTML standards. The main difference between HTMLMETA and HTML doctypes are:

i. Only the fields in <HEAD> are processed.
ii. The field names are processed differently.
iii. Different Presentation.

- The content in <META NAME="Foo" Content="Bar"> are searchable in HTMLMETA (default behaviour) under FOO. The HTML and -- doctypes, by contrast, map the content to META.FOO.
- The presentation is the extracted record of meta-data and not the document.
- Similiar to the HTML-- doctype nearly all HTML-tags are ignored. The exception are the BASE, META and LINK EMPTY tags.
  HTMLMETA does not process *user tags*. While HTML-- would store the content of `<AUTHOR>Elmer Fudd</AUTHOR>` under *AUTHOR*, the HTMLMETA doctype would *ignore the field*.

The parser does not enforce the use of any particular attribute set. Using the GILS core attribute set, for instance, the HTMLMETA doctype can be used to *automatically* create GILS-compliant Z39.50 servers to HTML data resources. The default (RAW) data format produced is an **experimental** XMLformat suitable for export, remote meta-indexing, whois++ (distributed), X.500/LDAP or import into another re-Isearch database using the XML or GILS document handlers.

Mark-up Conventions

An HTML file containing the fragement:

```
<META NAME="AUTHOR" CONTENT="Elmer Fudd">
<META NAME="COMPANY" CONTENT="Acme Inc.">
<META NAME="ADDRESS" CONTENT="Looney Place 4, Acme Acres">
<META NAME="VERSION" CONTENT="%Z%%Y%%M% %I% %G% %U% BSN;">
```

Contains the fields *AUTHOR*, *COMPANY*, *ADDRESS* and *VERSION*. Each are fully searchable and mappable to attribute OIDs.

The above example would produce SUTRS Presentation fragment:

AUTHOR:	Elmer Fudd
COMPANY:	Acme Inc.
ADDRESS:	Looney Place 4, Acme Acres
VERSION:	@(#)HTMLMETA.html 1.11 11/27/2008 23:19:35 BSN (1)
LOCATION:	*URL to the original HTML (2)*

1) If document was processed by sccs-get to resolve the %?% symbols.
2) "The URL to the original HTML" is derived from the <BASE ... > or synthetically derived. For a GILS or other meta-search facility it can point to PURL (Persistant URL) resolver elsewhere, such as gils.org, that, in turn, maps to the resource. The later is of particular relevance for maintaining links to resources for robots and other page gatherers. See HTML BASE processing.

The *XML* representation would be:

```
<!ENTITY LOCATION CDATA "URL" -- Indexed Document -->
<AUTHOR>Elmer Fudd</AUTHOR>
<COMPANY>Acme Inc.</COMPANY>
<ADDRESS>Acme Inc.</ADDRESS>
<VERSION>@(#)HTMLMETA.i.html 1.11 11/27/98 23:19:35 BSN</VERSION>
```

An entity notation has been chosen instead of:

<LOCATION>*http://www.your.org/blah*</LOCATION>

or

<LOCATION HREF="*http://www.your.org/blah*"/>

so as not to break emerging DTDs.

Note: The name for the *LOCATION* as well as the XML notation is *subject to change* in future versions.
In Content, if following the '"' is a (*Keyword=YYY*)... it is specially processed. At current there are handlers for two *keywords*: *Schema* and *Type*. If the *keyword* is not in this set of *registered* words (with handlers) it is considered part of content, eg.
<META NAME="AUTHOR" CONTENT="(Fudd)"> is read as <AUTHOR>(Fudd)</AUTHOR>.

Specifying Complex Attributes

To specify a *schema* (for container data typing) one includes it in *CONTENT* as
<META NAME=*Attribute* CONTENT="(Schema=*Type*)*Content*">
The (more intuitive) alternative: <meta name="*Attribute*(*Type*)" content="*Content*"> is not valid HTML given the '('. It is, none-the-less, in the *HTML tradition*, accepted.
In the presentation *both* are mapped to: <*Attribute* Schema="*Type*">
Example:

- <meta name="date" content="(SCHEMA=ISO)1996-04-19">
- <meta name="date(ISO)" content="1996-04-19">

Which would produce an XML of: <DATE SCHEMA="ISO">1996-04-19</DATE>
In the above example the content data would be stored under the field *DATE(ISO)*

```
<!ATTLIST META
  %i18n;                      -- lang, dir for use with content string --
  http-equiv  NAME  #IMPLIED  -- HTTP response header name  --
  name        NAME  #IMPLIED  -- metainformation name --
  content     CDATA #REQUIRED -- associated information --
  scheme      CDATA #IMPLIED  -- select form of content --
  <
```

HTMLMETA interprets as equivalent:

i. <META NAME="DATE(ISO)" CONTENT="1996-04-19>
ii. <META NAME="DATE" CONTENT="(SCHEMA=ISO)1996-04-19>
iii. <META NAME="DATE" SCHEME="ISO" CONTENT="1996-04-19>

Marking Up heirarchical Data

To represent heirarchical data the *HTMLMETA framework* uses the '.' (*dot-notation*), as
`<META NAME="Tag0.Tag1..." CONTENT="Content">` eg.

```
<META NAME="AUTHOR.NAME" CONTENT="Elmer Fudd">
<META NAME="AUTHOR.AGE" CONTENT="60">
```

Produces:

```
<AUTHOR><NAME>Elmer Fudd</NAME><AGE>60</AGE></AUTHOR>
```

By extending to several levels of '.' one can define a simple heirarchical model of meta-data suitable for the development of sophisticated models and services.
The order of the METAs is ***significant***. To markup:

```
<TITLE>A Title
<NAME>A Name</NAME>
</TITLE>
```

One enters:

```
<META NAME="TITLE" CONTENT="A Title">
<META NAME="TITLE.NAME" CONTENT="A Name">
```

By contrast, the *META* markup:

```
  <META NAME="TITLE" CONTENT="A Title">
  <META NAME="NAME" CONTENT="A Name">
  <META NAME="TITLE.NAME" CONTENT="Another Name">
```

Would produce:

```
  <TITLE>A Title</TITLE>
  <NAME>A Name</NAME>
  <TITLE><NAME>Another Name</NAME></TITLE>
```

The convention is somewhat different from the one originally proposed at *W3C Distributed Indexing and Searching Workshop, May 28-29, 1996*:

`<META NAME= "schema_identifier.element_name" CONTENT="string data">`
In general HTML documents marked-up with this convention will be correctly interpreted. The major difference between it and the semantics of the HTMLMETA framework occurs when the element_name includes `'.'`: `<META NAME="GNU.FOO.BAR" CONTENT="Aardvarks">`.

HTMLMETA framework
: field *BAR* as a sub-element of *FOO* under *GNU*

W3C Workshop framework
: field *FOO.BAR* within the *named schema* GNU. This can be viewed as *FOO.BAR* under *GNU*. The order of the *META* tags is invariant.

Another common model and typical among *Dublin Core* implementations is:

```
  <META NAME="schema_identifier.element_name"
```

CONTENT="(Type=*element_type*)*Content*">. In the HTMLMETA framework it would be *interpreted* as:

```
 <META NAME="schema_identifier.element_name.element_type" CONTENT="Content">
```

The advantages of the HTMLMETA framework are:

- Explicit heirarchical meta model
- Better support of existing *META* markup.

Although many document/data models can be *"flattened"*, especially for S/R services a heirarchical model can provide advantages. Examine, for example, the following GILS attributes:

```
<!ELEMENT Contact-Name               - O (#PCDATA)>
<!ELEMENT Contact-Organization       - O (#PCDATA)>
<!ELEMENT Contact-Street-Address     - O (#PCDATA)>
<!ELEMENT Contact-City               - O (#PCDATA)>
<!ELEMENT Distributor-Name           - O (#PCDATA)>
<!ELEMENT Distributor-Organization   - O (#PCDATA)>
<!ELEMENT Distributor-Street-Address - O (#PCDATA)>
<!ELEMENT Distributor-City           - O (#PCDATA)>
```

Using something like this, instead:

```
<!ELEMENT Name           - O (#PCDATA)>
<!ELEMENT Organization   - O (#PCDATA)>
<!ELEMENT Street-Address - O (#PCDATA)>
<!ELEMENT City           - O (#PCDATA)>
<!ENTITY % address "NAME & ORGANIZATION & STREET-ADDRESS & CITY">
<!ELEMENT CONTACT O O  (%address)>
<!ELEMENT DISTRIBUTOR O O  (%address)>
```

has the advantage that the kind of content is more clearly defined as an object.
On would then markup as:

```
<META NAME="Distributor.City" CONTENT="Munich, Germany">
```

Mapping to a heirarchical DTD with *ORGANIZATION*, *STREET-ADDRESS* and *City* as subelements of *CONTACT* and *DISTRIBUTOR* allows for an *implicit and interoperable* search for *ORGANIZATION* which would **include** a search of both *CONTACT* and *DISTRIBUTOR*.

Real world example

The following example illustrates the use of Dublic Core Metadata in HTML.

```
<-- Ripped out from http://info.ox.ac.uk/~lou/wip/metadata.syntax.html -->
<meta name="title" content="A syntax for Dublin core Metadata:
 Recommendations from the second Metadata Workshop">
<meta name="author" content="Lou Burnard">
<meta name="author" content="Eric Miller">
<meta name="author" content="Liam Quin">
<meta name="author" content="C. M. Sperberg-McQueen">
<meta name="subject" content="metadata">
<meta name="subject" content="Second Metadata Workshop (Warwick, U.K.)">
<meta name="date" content="1996">
<meta name="date" content="(Schema=ISO)1996-04-19">
<meta name="object-type" content="article">
```

```
<meta name="form" content="HTML 2.0">
<meta name="form" content="(Schema=IMT)text/html">
<meta name="identifier"
      content="(Schema=URL)http://info.ox.ac.uk/~lou/wip/metadata.syntax.html">
<meta name="identifier"
      content="(Schema=URL)http://www.uic.edu/~cmsmcq/tech/metadata.syntax.html">
<meta name="source" content="(none)">
<meta name="language" content="(Schema=ISO6639)en">
```

The Meta-record fragment would be:

```
<!ENTITY LOCATION  CDATA "http://info.ox.ac.uk/~lou/wip/metadata.syntax.html">
<author>Lou Burnard</author>
<author>Eric Miller</author>
<author>Liam Quin</author>
<author>C. M. Sperberg-McQueen</author>
<subject>metadata</subject>
<subject>Second Metadata Workshop (Warwick, U.K.)</subject>
<date>1996</date>
<date SCHEMA="ISO">1996-04-19</date>
<object-type>article</object-type>
<form>HTML 2.0</form>
<form SCHEMA="IMT">text/html</form>
<identifier SCHEMA="URL">
        http://info.ox.ac.uk/~lou/wip/metadata.syntax.html</identifier>
<identifier SCHEMA="URL">
        http://www.uic.edu/~cmsmcq/tech/metadata.syntax.html</identifier>
<source>(none)</source>
<language SCHEMA="ISO639>en</language>
```

As with all the doctypes it includes an autodetection of implied hyperlinks, URLs and email addresses.
Note: The *LOCATION* refers to the derived URL of the document being indexed and this might be different from the encoded URL identifier. The later is the *more definitive* source and can well have a different encoded meta-record (different version).

Meta Level Grain

The HTML presentation, in turn, also contains Metadata and link information about the resource record derived from the <LINK...> content in the original resource.
The `<LINK REL="BASE" HREF=URL>` encodes the link from the meta record to the original HTML that contained the meta.
HTTP-EQUIV metadata for DATE-MODIFIED and EXPIRATION are transfered from the original resource to the meta-record. Rating system information such as PICS is also reflected in the META record.

```
<META http-equiv="PICS-Label" content='(PICS-1.0
"http://www.classify.org/safesurf/" l on "1997.05.02T13:23-0100 r (SS~~000 1)'>
```

Is a simple PICS classification for a HTML resource. While this PICS lable refers to the resource and might not apply to the metarecord it is still transfered.
The <HEAD> would contain:

```
<META http-equiv="PICS-Label" content='(PICS-1.0
"http://www.classify.org/safesurf/" l on "1997.05.02T13:23-0100 r (SS~~0 00
```

1)'>
And the content would contain:

PICS-Label: (PICS-1.0 "http://www.classify.org/safesurf/" l on "1997.05.02T13:23-0100 r (SS~~000 1)

or as XML:

<PICS-Label>(PICS-1.0 "http://www.classify.org/safesurf/" l on "1997.05.02T13:23-0100 r (SS~~000 1)</PICS-Label>

The *LINK* attributes are transfered to the HTML and mapped into the XML as:

`<LINK Type=Name TITLE=TITLE HREF=URL>`

To `<Type NAME=Name TITLE=Title HREF=URL/>` , eg. tags *REV* and *REL*.

The `<LINK REL="META" SRC="...">` and `<LINK REV="META" SRC="...">` are *special*! They are used to provide *external* meta-record definitions to a HTML resource. There are no definitive standards yet.

The `<LINK REL="SCHEMA.XXX" HREF="..." and <LINK REL="DTD.XXX" HREF="..."`
are used to define the descriptive schema and the DTD for the structure.

Lets look at a more complicated, "real-world", example:

```
<!-- Dublin Core Metadata Package -->
<META NAME="DC.title" CONTENT="Proposed Encodings for Dublin Core Metadata">
<META NAME="DC.author.name" CONTENT="Dave Beckett">
<META NAME="DC.author.email" CONTENT="D.J.Beckett@ukc.ac.uk">
<META NAME="DC.subject.keyword" CONTENT="metadata, dublin core">
<META NAME="DC.identifier.URL" CONTENT="http://www.hensa.ac.uk/pub/metadata/dc-
encoding.html">
<META NAME="DC.form.imt" CONTENT="text/html">
<META NAME="DC.language" CONTENT="en">
<LINK REL="SCHEMA.dc" TITLE="Dublin Core"
HREF="http://purl.org/metadata/dublin_core_elements">
<LINK REL="DTD.dc" TITLE="-//OCLC//DTD Dublin Core 1.0//EN"
HREF="http://purl.org/metadata/dublin.dtd">
```

The LINK specifies the schema used for the METAs whence is a kind of meta-DTD model about the structure of the meta-record. The DTD.dc refers to the DTD for the model, so one has a

```
<?XML ENCODING="Charset (IANA Name)" VERSION="1.0"?>
<!DOCTYPE DC PUBLIC "-//OCLC//DTD Dublin Core 1.0//EN"
"http://purl.org/metadata/dublin.dtd">
<!ENTITY LOCATION  CDATA "http://www.bsn.com/blah/blah/blah">
<DC NAME="Dublin Core" SCHEMA="http://purl.org/metadata/dublin_core_elements">
<TITLE>Proposed Encodings for Dublin Core Metadata"</TITLE>
<AUTHOR>
  <NAME>Dave Beckett</NAME>
  <EMAIL>D.J.Beckett@ukc.ac.uk</EMAIL>
</AUTHOR>
<SUBJECT>
  <KEYWORD>metadata, dublin core</KEYWORD>
</SUBJECT>
<INDENTIFIER>
  <URL>http://www.hensa.ac.uk/pub/metadata/dc-encoding.html&;lt;/URL>
```

```
</INDENTIFIER>
<FORM>
 <IMT>text/html</IMT>
</FORM>
<LANGUAGE>en</LANGUAGE>
</DC>
```

Since meta-data markup conventions are in flux and the HTML should survive past the next change, via the doctype option *MAPPING* one can specify a map file to map the container specified in *<META NAME=..>* to an alternative name. This feature has been provided to allow for the unification of field names between collections— eg. to interpret `<META NAME="DC.form.imt" CONTENT="text/html">` and `<META NAME="DC.form" CONTENT="(Schema=imt)text/html">`— and doctypes.
The XML produced is *NOT VALIDATED* but is assumed to have been correct. It is the duty of the HTML authors to confirm the correctness of the mark-up.
If `<LINK REL="DTD.dc" ...>` is not defined then the XML record produced would be without the doctype declaration.

XSpace Meta Content Framework (MCF)

In addition the HTML presentation contains a:
`<EMBED SRC="URL.mcf">`
as a link to a MCFified view of the hyper-content of the original document. This dynamic reference is a catalog of the links from the document *external* to the document and meta-record. The MCF file format is originally from Apple Research's ProjectX.

http://www.guha.com/mcf/mcf_spec.html

The MIME type for the MCF is: `Content-type: image/vasa`
Together with a MCFed Web one has the possibility to create a VRML-like geometric navigation model. It is easily reproduced from a HyTime model.

HTMLHEAD

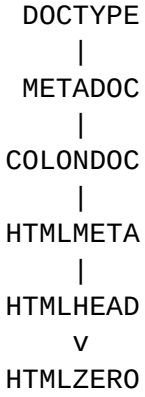
This is a variant of the HTMLMETA doctype intended for use by spinders and crawlers.

```
HTMLHEAD Class Tree:
                                DOCTYPE
                                   |
                                METADOC
                                   |
                               COLONDOC
                                   |
                               HTMLMETA
                                   v
                               HTMLHEAD
```

Processes META, TITLE and LINK elements in <HEAD>...</HEAD> or <METAHEAD>...</METAHEAD>

HTMLZERO

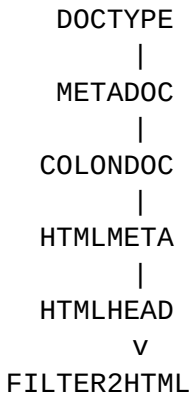
A HTMLHEAD variant that does not treat 'most' tagnames as words.



In contrast to rendered HTML it still supports fielded search all the metadata and allows one to use HTML comments to supplement documents with searchable but not 'visible' information.

FILTER2HTML

Class Tree:



The FILTER2HTML is a special variant of the HTMLHEAD doctype. Instead of processing files directly it us designed to use an external filter to convert the input file into HTML type files for processing by the HTMLHEAD document handler.

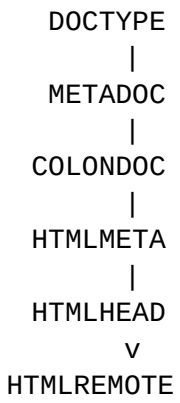
Options:

```
 Filter        Specifies the program to use.
Content-type  Specifies the MIME content-type of the original.
```

These are defined in the "filter2html.ini" [General] or <Db>.ini [FILTER2HTML] sections. Filters should take a single argument as the path to the (binary) input file and should write their output (HTML) to standard output (stdout).

HTMLREMOTE

Class Tree:

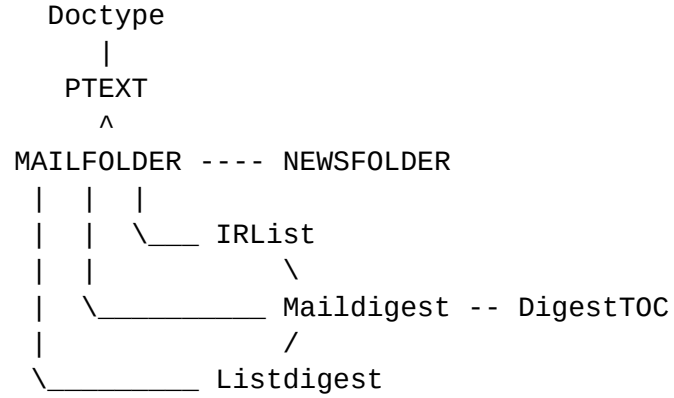


The HTMLREMOTE doctype is intended for processing Web pages retrieved from remote sites. Its main additional feature over HTMLHEAD is that it can resolve local paths to URLs. This way a retrieval can return a URL to the remote document rather than retrieve the local cached copy.

Search time Options:

```
   BASE    Specifies the base of the HTML tree (root) This can also be
specified in the doctype.ini as BASE=dir in the [General] section.
```

MAIL DOCTYPES



The mail doctypes have been designed to accept and correctly handle nearly anything that ends up in a mailfolder.

The `MAILFOLDER, NEWSFOLDER, IRLIST, MAILDIGEST, LISTDIGEST` and `DIGESTTOC` doctypes are all intimately intertwined into one another.

The `AUTODETECT` class knows how to identify these files, basically those with mail headers and those with news headers. All these files are passed to the `MAILFOLDER` doctype. It in turn understands the different types of mail and passes the records to the suitable doctype. These doctypes are also equiped with identification logic and can pass the doctype to yet another class.

In practice this means that: a folder can contain a mixture of all the kinds of mail that arrive in ones mailbox, even mixed with newsfolders. Due to this design it is well suited for use as part of an automatic list or behind a incoming mail filter.

MAILFOLDER Message Parsing

The MAILFOLDER mail parser understand a number of header tag keywords in mails including some non-standard X- elements such as "X-No-Archive", "X-OriginalArrivalTime", "X-Original-Date", "X-Supersedes", "X-To", "X-UID" and "X-Url". It importantly uses the “Message-Id” keyword value to define a key (the value is encoded into an ASCII alphanumeric string safe for transmission) for the record since they are by-definition unique identifiers.

> *“The "Message-ID:" field provides a unique message identifier that refers to a particular version of a particular message. The uniqueness of the message identifier is guaranteed by the host that generates it (see below). This message identifier is intended to be machine readable and not necessarily meaningful to humans. A message identifier pertains to exactly one version of a particular message; subsequent revisions to the message each receive new message identifiers.”* – IETF RFC5322 Internet Message Format[4]

Should multiple messages be encountered with the same Message-Id the handler will detect this and create an alternative key and typically mark the duplicate as “deleted”. This can happen when the same message is cross posted to numerous mailing lists and these lists are indexed together.

Note that according to the RFC it “SHOULD be present” but may not be present (should versus must).

The parser uses the Document-Id to not only assign unique record keys but also to resolve mail threads—messages referring to other messages especially using the "In-Reply-To:" and "References:" fields.

The other special keyword in email is “Expires”. The value of this field is a date defined as the “time at which a message loses its validity”. We use it to set the expiration date for the record. Depending upon configuration the record, once past its expiration date, will be either automatically deleted when the AutoDeleteExpired key is set to true (easily set using the Iutil tool with -autodelete) or manually triggered for deletion (available also via the Iutil command line tool via the -del_expired).

4 https://datatracker.ietf.org/doc/html/rfc5322

Limitations
Base 64 encoded messages and multipart messages are not deep processed. To do so would require that the attachments and components be decoded. Mails are processed as-in in the transport. Multipart boundaries are by-design not understood as multipart boundaries. This means that HTML and other encodings are not understood as such. Beyond the header elements there is the body and it is processed as text.

Should deep processing be demanded by an application, indexing its full content, one would need a pre-processing external filter to process the electronic messages into one or more records encoded in a manner more suitable. A mail, for example, containing a binary attachment can be processed into two records: one the mail body and the other the attachment. The attachment could then be indexed with an appropriate doctype and the mail could have a hyper-reference to said record.

`DIGESTTOC`

Internet Mail Digest (RFC1173, Listserver, Lyris and Mailman) Table of Contents

`NEWSFOLDER`

Newsfolders parsed by the `MAILFOLDER` document handler are "automaticaly" set with the `NEWSFOLDER DOCTYPE`. This is mainly to get the right MIME type for binding external applications such as a newsreader.
The MIME type for "Raw" records is "message/news".

`MAILDIGEST`

Internet mail digest file format per RFC 1153. This is the most common mail digest format. The `MAILDIGEST` is for single digests only. To process a folder of digests one should use the MAILFOLDER doctype. That doctype has been designed to identify RFC1153 digests and will pass the individual messages to the `MAILDIGEST` doctype.
The MIME type for "Raw" records is "Application/X-MailDigest". See also `MAILFOLDER`.

`LISTDIGEST`

Listserver mail digest file format. As with the `MAILDIGEST` doctype it is designed for single digests only. The `MAILFOLDER` doctype will identify `LISTDIGEST` messages and segment a folder into individual digests as records.
The MIME type for "Raw" records is "Application/X-ListDigest". See also MAILFOLDER.

`IRLIST`

Yet another mail digest format. The MIME type for "Raw" records is "Application/X-IRList".

This is a "helper" class for MAILDIGEST and LISTDIGEST. It handles the creation of a Table of Contents (with HyperLinks in HTML) for the contents of a digest. The Hyperlinks and Table of Contents is created "on-the-fly" during the Presentation. See
Digest TOC Example (HTML)
Digest TOC Example (SUTRS)
Raw Source.
The MIME type for "Raw" records is "message/rfc822". See also MAILFOLDER.

MEMODOC

```
MEMODOC Class Tree:
                                        DOCTYPE
                                           |
                                         PTEXT
                                           v
                                        MEMODOC
```

The MEMODOC doctype is designed to support a class of ASCII memos. The fields/tags are, like COLONDOC, defined by a : (Colon), eg. <Field Name>: <Field Value>

These pairs are parsed untill a boundary-seperator is encountered. This seperator is a combination of several _-+= characters (at least 4) or an empty line. The contents from the separator until the end-of-record is defined as the Memo-Body. In generic memos one would just format COLONDOC type lines with a blank line to separate to the body. In the text-body since it is a child of PTEXT it also tries to detect pages, paragraphs, lines and sentences as well as the first instances thereof as a unique field.

The Default Headline is contructed from, should they exist, a combination of the FROM and SUBJECT fields as: Memo from <From value>: <Subject value>

```
Example:
Field_name: Content of the field
Field_name2: Content of the 2nd field

Here is the content of the "MEMODOC-Body" field.

Options:
[General]
ParseMessageStructure=Yes|No // specifies if the message line/sentence/paragraph
                             // structure should be parsed (Default: YES)
AutodetectFieldtypes=Yes|No  // specified if we should guess fieldtypes (default:YES).
```

Handbook 0.9 (work in progress)

METADOC

Class Tree:

```
DOCTYPE
   v
METADOC
```

The METADOC doctype is a major base for a whole class of text style meta-document format files, especially COLONDOC and its family.

The format defines a set of a key-value pairs by default using the = and left (key) to right (value): e.g. key=value.

Example:

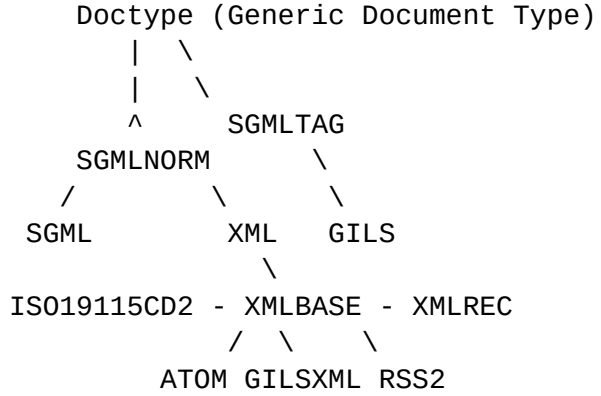
```
First_name=Frances
Middle_name=S.
Last_name=Hotchkiss
Phone=508-457-2242
Phone=FAX 508-457-2310
```

Indexing options (defined in .ini):

```
[General]

Style=List|Table // To use Lists or Tables for HTML presentations.
AllowWhiteBeforeField=True|False // Allow leading white space.
AutodetectFieldtypes=True|False // Guess fieldtypes
IgnoreTagWords=True|False        // Store tag names as searchable words?
SepChar=Character specified as Character or
     <space> (for ' '), <equals> (for '='),    <tab> (for the horizontal tab
     character), \nnn (octal number for character)  or xNN (hexidecimal). If not
     specified the default is '='.
```

These options are definable in all children of METADOC.

SGML DOCTYPES



Handbook 0.9 (work in progress)

The base for most of the SGMLish formats is SGMLNORM (there is another DOCTYPE called SGMLTAG that is designed for simple SGML-like taged documented) but not for SGML or XML documents).

SGML:

A **full** validating SGML parser. It works in 2 (two) passes. In the first pass the document is parsed, validated and normalized into a cannonical form and placed into a persistant cache. In the second pass the cannonical document is parsed and processed by the SGMLNORM handler.

The MIME type (for HTTP applications) is: `text/sgml`

XML:

"Beta Release (subject to change)"

The Extensible Markup Language (XML) is a language derived from SGML. XML has been designed to retain most of the power of SGML but with a substantially more simple grammar that is suited to tools such as yacc and lex.

The motivation for XML is to provide:

- A structured open document type for distributed applications working on large-scale networks such as the Internet.
- A simple language *compatible* with SGML
- An easy to author (even by hand) document format.
- An inter-operable basis for push delivery models.

The most significant features of XML are:

- A document can be processed without need of the DTD.
- All container tags are normalized
- Unicode character (UCS-2) set support via `&#nnnn;` style entities.
- Empty tags are explicit using a `<tag/>` markup convention.

The XML handler does not provide any significant language validation and since it is a child of SGMLNORM it also supports several SGML constructs that are not, and won't be, part of the standard.

Note: For the autodetect mechanism to work correctly one *must* either use the `.xml` (or `.XML`) file name extension or specify `<?XML VERSION="Version"?>` (only the `<?XML` is significant) in the document declaration. Since the XML document will be correctly parsed by the SGMLNORM doctype the major effect of *incorrect identification* would be in the MIME type for the *Raw record*.

SGMLNORM:

Handles SGML-type documents with "normalized" tags and entities, viz. end-tags and entity replacement.

- It is not a validating SGML parser and is not in complete adherence to *15.3* of the ISO

standard.

- It has been designed to provide the basic parsing and presentation services for heirarchical data for the Isearch engine, within the limitations and constraints of the Isearch architecture,
- The DTD is not processed and need not even be available on the indexing platform. Although the input is assumed to have Each non-empty tag (container) defines a field.
- In addition to storing tags values as a field it also stores the values of (complex) attributes to allow for a relatively complete search and retrieval of document content.

  In: <ONE TWO="Three">Four</ONE> Four is stored as the value of ONE Three is stored as the values for ONE@ and ONE@TWO

- SGML comments and declarations are correctly ignored. *Note:* Version 1.8 corrects a bug handling nested declaration.
- Empty tags per SGML Handbook Annex C.1.1.1, p.68 are also supported. The "null end tag" (*NET*), p.69, with the *short-tag feature* used in markup such as <tag/hello world/ is supported.
- Te parser also correctly processes XML conventions, viz. <EMPTY/> for empty containers.

The MIME type (for CGI applications) is: `text/sgml`

Index time Options:

```
Complex arg    // Takes 0 (ignore), 1 (accept) [default is 1]
IgnoreTagWords arg   // as above, store tag names or ignore them?
These can also be specified in the doctype.ini [General] section as
Complex=0 or 1.
IgnoreTagWords=0 or 1
```

Search time Options:

In the section [<DTD Name>] (example `[-//Free Text Project//DTD Play//EN])`

`Headline=<Headline format>.`

In the section `[Catalog]` the DTDs can be maped as `DTD=useDTD`

References:

- C. F. Goldfarb. ``The SGML Handbook." Y. Rubinsky, Ed., Oxford University Press, 1990.
- ISO 8879. Information Processing -- Text and Office Systems - Standard Generalized Markup Language (SGML), 1986.
- See Also
  - ISO/IEC JTC1/SC18/WG8
  - A Gentle Introduction to SGML

SGMLTAG:

Handles documents with SGML-like markup. It is NOT intended for SGML documents.

In v2.0+ of the doctype suite the SGMLTAG doctype is tuned to the needs of GILS documents.

The headline is constructed from the 2nd level tag.

The MIME type (for CGI applications) is

Application/X-SGMLTAG-<level-1>

Example: `<Rec> <Title> Personnel Action System </Title> <Originator>U.S. Geological Survey </Originator>`

For the above example: `Application/X-SGMLTAG-Rec`.
And the *Headline*: `Personnel Action System`.

GILS:

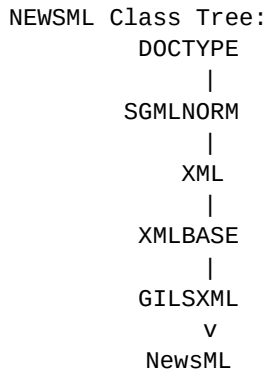
Handles documents with SGML-type markup for use in GILS systems. It is NOT intended for SGML documents. It uses SGMLTAG.

The MIME type (for CGI applications) is

Application/X-GILS-<level-1>

NEWSML

The doctype is designed to handle the newsml XML format from the International Press Telecommunications Council (IPTC), an organization established by a group including the Alliance Européenne des Agences de Presse, ANPA (now NAA), FIEJ (now WAN) and the North American News Agencies (a joint committee of Associated Press, Canadian Press and United Press International) to safeguard the telecommunications interests of the world's press.
http://www.newsml.org / https://iptc.org/



The format is basically XML but it also understands a few details about NewsML such as that elements "DateAndTime", "DateId", "DateLabel", "DateLineDate", "FirstCreated" and "ThisRevisionCreated" are date datatypes.

As default for the record key it uses the value of Duid from <NewsML Duid=”key..”>. For the date of the record itself it uses NewsML\\NewsEnvelope\\DateAndTime. For date created it uses "FirstCreated" and for date modified it uses "ThisRevisionCreated". Since news has a expiration date the engine uses for date expires “EndDate”;

PTEXT

Plaintext with "Page", "Paragraph", "Sentence" and "Line" fields as well as "firstParagraph", "firstSentence" and "firstLine" fields for the resp. first incidences of these fields.

PTEXT Class Tree:

DOCTYPE
v
PTEXT

A "Headline" field is, by default, set to the longer of "firstLine" or "firstSentence". If "Headline" is set to empty (<ignore>) then no "Headline" field will be created during indexing. The <ignore> value can be used to disable individually the line, sentence, paragraph and page detection.

These options are defined in the `Doctype.ini` [Fields] section.

Additional `DB.ini` Options:

[General]

```
ParseBody=Y|N       # To parse the body for the above "fields"
ZeroLengthPages=Y|N # Allow for empty pages to be stored (for page count)
Headline=<Fieldname>
```

The algorithm for line, sentence, paragraph and page detection is relatively straightforward and reflects European tradition.

Line
: The contents between two lines (defined by line feeds or carriage returns).

Sentence
: The contents between the demarcation of the end-of-sentence punctuation.

Paragraph
: The contents between block of text with some extra whitespace between their end and the start of the next block

Page
: Text between emissions of the page character.

```
NOTE: Elements Line and sentence may overlap. A line can contain parts of a
sentence or wholly one or more sentences. A sentence may be within a single
```

line or spread across multiple lines. A paragraph can consist of a single line or single sentence but never can a sentence contain more than one paragraph.

RECXML

The RECXML doctype is a special child of the XML document types used to “slice and dice” an XML document with multiple items into, from the perspective of re-Isearch, multiple records each containing a single item.

DOCTYPE
|
SGMLNORM
|
XML
|
XMLBASE
v
XMLREC

In order to slice-and-dice it needs to know what element is used for the “cut”. It is defined as the RecordSeperator option. When it encounters the <Seperator ...> it triggers a new record event. The entire content from the < in <Seperator .. until the > in </Seperator > is viewed as a single record. The declaration and whatever was before the first instance of the element is ignored just as the content in-between and after the closing last instance of the element.

In order to be XML compliant on presentation it needs two additional items configured:

(a) Preface: the preface is all the verbage including the document declaration.
(b) Tail: whatever XML should be inserted at the end.

TIP: In order to have multiple sets of prefaces and tails for different document types under the RECXML banner one uses the doctype child specification convention: NAME:RECXML where name is a unique identifier for the format. In its NAME.ini configuration file (in the usual places) or in the DB.ini under the NAME section one would have the preface, tail and record separator defined..

XVI. DOCTYPE Plugins

The re-Isearch engine is designed to be extended with both internal “built-in” doctypes but also “plugins”. The latter are loadable at run-time. These loadable plug-ins can handle all or just partial services and are not just descendents of a built-in document handler class but also can pass control to other types not immediately in its class path.
While the re-Isearch “built-in” doctypes are fullly open source and licensed according to the same license and conditions as the rest of the engine, the “plugin-in” types allow for independent distribution and may be covered by other licenses such as GPL or even closed source and proprietary.

Plug-ins are developed in C++ and are quite simple in structure.

The following example PLUGIN “NULL” does not do anything but shows the format.

```
/*@@@
File:           null.cxx
Version:        1.00
Description:    Class NULL
@@@*/
#include "doctype.hxx"

#ifndef _WIN32
# define __declspec(dllexport)
#endif

class IBDOC_NULL : public DOCTYPE {
public:
   IBDOC_NULL(PIDBOBJ DbParent, const STRING& Name);
   const char *Description(PSTRLIST List) const;

   ~IBDOC_NULL();
};

static const char myDescription[] = "Empty plugin";

// Stubs for dynamic loading
extern "C" __declspec(dllexport) {
  IBDOC_NULL *  __plugin_null_create (IDBOBJ * parent, const STRING& Name)
  {
    return new IBDOC_NULL (parent, Name);
  }
  int          __plugin_null_id  (void) { return DoctypeDefVersion; }
  const char * __plugin_null_query (void) { return myDescription; }
}

IBDOC_NULL::IBDOC_NULL(PIDBOBJ DbParent, const STRING& Name) : DOCTYPE(DbParent, Name) { }

const char *IBDOC_NULL::Description(PSTRLIST List) const
{
  List->AddEntry (Doctype);
  DOCTYPE::Description(List);
  return myDescription;
}

IBDOC_NULL::~IBDOC_NULL() {
}
#ifdef _WIN32
int WINAPI DllMain (HINSTANCE hInstance, DWORD fdwReason, PVOID pvReserved)
{
  return TRUE;
}
#endif
```

The important elements to be registered are the symbols (notice case):

```
int          __plugin_null_id
const char * __plugin_null_query

IBDOC_NULL *  __plugin_null_create
```

when the engine is fired-up and the doctype registery is initiated it sets as default search path for plugins (under Unix):
"~/.ib/plugins:/var/opt/nonmonotonic/ib/lib/plugins:~asfadmin/lib/plugins:~ibadmin/lib/plugins:/opt/nonmonotonic/ib/lib/plugins:."

The algorithm to see if a plugin exists and load it:

1. Look for a shared library. The file name of a plugin is upper-case letters for the class followed by the extension .sob.
2. Look for the symbol __plugin_xxx_id. The id function returns the version of the plugin.
3. Confirm that the value returned by a call to the _id function matches the version expected.
4. Construct the class using the create function call.

The routine to look at the _create symbol and get a pointer to the class:

```
static inline dt_constr_t Creator(HINSTANCE handle, const STRING &module)
{
  dt_constr_t create = NULL;
  if (handle)
    {
      STRING symbol (PluginSymbolPrefix);
      symbol += module;
      symbol += "_create";
      logf (LOG_DEBUG, "Symbol '%s' load", symbol.c_str());
      create = (dt_constr_t) dlsym (handle, symbol.c_str());
    }
  return create;
}
```

The call to dlsym obtains the address of a symbol in a shared object. Notice that we expect it to be void so we can call it without arguments. In the case of the _create we expect a pointer—with _id we expect and int and with _query we expect a C-style 0 terminated array of characters.

Note: __plugin_xxx_query where xxx is the name of the doctype class returns the description of the plugin. Recall every doctype is designed to carry with it some of its own documentation. This is also available from plugins.

Plugins are internally represented with their name followed by the ‘:’ character. The above NULL plugin would be seen as “NULL:”.

XVII. Tuning

XVIII. C++ API

Although the re-Isearch engine is written in a subset of C++ one generally does not— except for callbacks and ranking algorithms where ultimate execution speed are demanded— develop applications in c or C++ but typically one of the script languages

- Python
- Tcl
- Perl
- Ruby
- Java
- PHP

These languages are convenient for application development beyond even rapid prototyping and provide excellent performance and robustness. For most developers they are the first choice.

These higher level directly accessed classes execute quickly (as fast as native development since they are native) and are easy to use. For example in Python:

```
import sys
import string
from Isearch import *

query='chain test OR'

Db="WEB";
pdb = VIDB(Db);
print "This is PyISearch version %s/%s" % (string.split(sys.version)[0],
pdb.GetVersionID());
print "Copyright (c) 1999 Basis Systeme netzwerk/Munich";
if not pdb.IsDbCompatible():
  raise ValueError, "The specified database '%s' is not compatible with this version. Re-
index!" % `Db`
elements = pdb.GetTotalRecords();
print "Database ", Db, " has ", elements, " elements";
total = 10;
if elements > 0:
        rset = pdb.VSearchRpn(query, ByScore, 300, total); # RPN Query
```

At the core is a C++ API to which the interfaces are generated using SWIG and the following interface classes:

```
// Search Statistics
class IDB_STATS {
public:
  IDB_STATS();
  ~IDB_STATS();
  void SetHits(size_t nHits);
  void SetTotal(size_t nTotal);
  size_t GetTotal() const;
```

```
  size_t GetHits() const;
  void   Clear();
  void   SetName(const STRING newName);
  STRING GetName() const;
};

class VIDB_STATS {
public:
  VIDB_STATS();
  ~VIDB_STATS();
  void Clear();
  IDB_STATS operator[](size_t n) const;
  void SetTotal(size_t i, size_t total);
  void SetHits(size_t i, size_t total);
  void SetName(size_t i, const STRING Name);
};

class IDBOBJ {
public:
  IDBOBJ();
  ~IDBOBJ();

  bool getUseRelativePaths() const;
  bool setUseRelativePaths(bool val=1);

  STRING RelativizePathname(const STRING Path) const;
  STRING ResolvePathname(const STRING Path) const;
};
```

TO BE CONTINUED

Compatibility with the XML:DB API paradigm

General Requirements	
Language Independence - The API MUST NOT preclude the usage with more than one language binding.	Re-Isearch provides a large number of languages bindings including C++, Tcl, Python, Perl, PHP, Ruby, Java, C#)
Textual Interface - The API MUST provide a textual representation of XML result sets.	Yes.
XML-API Interface - The API SHOULD provide a SAX or DOM based representation of XML result sets.	A SAX or DOM based representation of the XML result sets is available via loading the XML representation of the result sets into DOM

XIX.Comparison to Other Engines

Lucene

Lucene is a very popular engine. It is the motor behind ElasticSearch, Solr, Neo4j and many other products. To compare Lucene with re-Isearch is really to compare potatoes with fish. They have quite different histories, design

considerations and goals. The following short sketch attempts, however, to outline a few points since we've been asked.

1. Design target
    - Lucene was designed to be a reasonably flexible fulltext search engine with support for fields. Its a more or less traditional unstructured text search system using an optimized inverted index.
    - re-Isearch was designed to be an object (neither text nor data centric) oriented search engine for abstract objects and structural paths (including overlays). Its been designed to try to manage wildly heterogeneous formats and extract also implicit structure. re-Isearch supports XML, SGML and other formats as-if native.
2. Java
    - Lucene is typically pure Java. Its 100% written in Java. Its more or less Java thread safe but not completely.
    - re-Isearch is written in C++. Java is provided via a SWIG created JNI (Java Native Interface) module. Since re-Isearch is written in C++ and interfaces via SWIG, application development is not limited to Java but Python (perhaps the most popular choice and by far the best developed re-Isearch interface), Tcl, Ruby and a number of other languages.
    - Lucene needs Java to run (although there are a few forks rewriting the algorithms and structures into other languages). Java is, in our educated opinion, a fine language for developing many kinds of applications (especially given the availability of Java developing talent) but is less than ideal for database, search and retrieval.
    - re-Isearch allows for applications to use its algorithms to be written in Java but does not require the use of Java. Our favorite language for writing applications to use re-Isearch is, in fact, Python and not Java.
3. Portability
    - Since Lucene is pure Java its portable to platforms with suitable JVMs. Packages should just run from platform to platform. No need, in theory, for specific binaries beyond the JVM.
    - re-Isearch is written in extremely portable C++. It can be targeted to most platforms (Win32, Linux, Solaris, BSD etc. are available) but demands a package for each platform. Applications, of course, written in re-Isearch's Java (or other language API) are portable to any platform.
4. Threads
    - Lucene is pure Java and with the exception of the query parser and a few other bits its thread and more or less process safe. Some of it is at the cost of S/R (search and retrieval) concurrency: searches are in memory (either in whole or segments) to make them thread robust (and avoid file system and I/O deadlocks) and so don't include changes to the index during the lifespan of the class (or segment read).
    - re-Isearch is designed to handle S/R concurrency. The indexer has been designed to be able to run nearly continuously with search always in-step with the index. Its built using the POSIX threads model. re-Isearch, however, isn't 100% 'thread-safe' in the sense that the programmer can use code indiscriminately from threads. Its been designed with reasonable process safety in mind to allow for robust development of search and retrieval applications. re-Isearch is typically used in Web server environments as a service (via protocols). re-Isearch can be via JNI embedded into application servers such as Tomcat but its not advised— and especially not on Linux 32-bit systems (due to the

design of the 32-bit kernel and how it uses low memory to manage memory mapped pages which together with the size of Tomcat+Apache and the max. address space leaves very little room despite available memory in RAM and swap) highly discouraged. De-coupling search from the application server increases the resilience of the application server to traffic. Java, after all, is about modularization and client-server objects.

Do threads make sense for search? See: Should one run search in threads? In a nutshell: not really and especially not with lower cost servers built around the x86 architecture and PCIe bus. They have single data ports which limit their input and output to their serial flow through the pipeline. With fast SSDs this is particularly apparent but even with slower harddisks it is noticeable as they are limited to reads within a sector. Wincherster harddrives' relatively small disk buffers favor sector-to-sector serial reads for highest throughput. Disk access will tend to use the cache less and demand more head movements (slower, more heat, more power needed etc.). In comparing queues to concurrent threads the later take more CPU and produce response times not better than the last in a sequential queue— Search performance, after all, is driven more by memory access speed and I/O latency than by CPU speed.

5. Searching multiple indexes, JOINs etc.
    - With lucene you can only search 1 index with a query. There is no means to create virtual indexes.
    - re-Isearch supports virtual indexes and index import. One can create on demand virtual collections of indexes and transparently search them with a single query.
    - With Lucene there is no means to import and merge multiple indexes into a single index.
    - With re-Isearch can also import multiple indexes into a single index.
    - re-Isearch supports also JOINs and via the object system these joins can be to RDBMSs.

6. Permitted document size and speed
    - Lucene normally indexes only the first 10,000 words of a document. When increasing this default out-of-memory errors can occur. Despite its memory demands it still seems to index quite slow. Its slow since it indexes on a per document basis. Eating a document, spitting out its index and then merging or optimizing into the main index.
    - re-Isearch indexes each and every word of a document. It does not matter how many words a document has. The amount of memory an indexing process uses is defined by the configuration and is not related to the size of the documents being indexed, the number or frequency of there terms. The more memory (as long as the system does not start constantly swaping) one gives a process, up to the total size of all the documents indexed, the faster the indexer will run but it can also run in a tiny memory footprint. The minimum memory an index process needs is the size of the largest record (it self-adjusts for this). re-Isearch can typically index significantly faster than one could copy the documents into a tar archive. The more and faster the I/O (memory, disk etc.) the faster the indexing process.
7. Field length
    - Lucene sets (by default) the max. field length by default to 10000 terms. This is to set an upper bound for the amount of memory used for indexing a single document. Since this still can lead to OOM (Out Of Memory)— especially on 32-bit Linux platforms— one is often better off reducing it to half that value.

- re-Isearch places no limitation of max. field length. Just as a document can contain an number of terms, a field can contain any number of terms— and one can have any number of fields as well.

8. Memory Demands
    - Lucene (including Java) needs a lot of memory to run. RAM memory consumption is more or less constant at a high level during both indexing and searching activities.
    - re-Isearch can be configured to use a specific amount of memory during index. It can also self-configure itself to run in a portion of the free RAM available on the system (determined at indexing start).
    - Lucene has a high fixed memory demand for search since segments of indexes are copied into memory.
    - re-Isearch has a low fixed memory demand. The amount of memory needed by a search is related to the size of the resulting result. The larger the number of records found and the more hits they have, the larger the memory used. Once the result set is no longer used its disposed of.
    - Since Lucene is Java it is dependent upon the Java memory management (garbage collector) to manage system memory. Java tends to "hog" memory: normally takes but returns little memory to the operating system.
    - re-Isearch uses memory mapping and allocates and deallocates memory to the operating system. The model has been designed to try to run in limited resources and create a minimal impact on total system performance (other programs running).
9. Exclused terms / Stop words
    - Lucene normally excludes "common" words, so-called "stop words", from the index. The general use is to have these stop words also automatically chosen on the basis of their frequency in the index. The default value of 0.4 means that words that are more common than 4 per 1000 words are automatically excluded from the index.
    - re-Isearch does not demand but can allow for the use of stop words. re-Isearch supports stop words on a per-language basis (language of document) and allows for distinct lists for use during indexing (exclude from the index) and search (exclude as ordinary term for search). Common practice is to index each and every term and only use stop words, if at all, during search. The term "war", for example, might not be too significant in German ("was" in English) but means quite something else in English (conflict, name of a 1960s funk band etc.).
10. Term length
    - Lucene places limits on the lengths of terms
    - re-Isearch is designed to handle terms/words of any length.
11. Search Terms/Wild cards/Truncated search terms
    - Lucene expands wildcards to terms before even searching. Queries are re-written into a more basic form consisting of a set of logically combined TermQuery's. The standard limit on the total number of terms in a query is 1024. For example, if the indexed documents contain the terms "car" and "cars" the query "ca*" will be expanded to "car OR cars" before the search takes place.Lucene "pre-processes" steps:

i. A sorted term enumeration is started from the term alphabetically closest to/after the given prefix (the term characters on the left). This enumerates all terms from all existing fields in the index.
ii. For each term, Lucene checks if the term actually starts with the prefix and belongs to the given field. If so the term is added to a BooleanQuery as a TermQuery with OR logic.
iii. The process produced a constructed BooleanQuery which contains exactly as many clauses as there were terms matching the prefix.

For a WildcardQuery, the process is similar in that the term value string containing wildcard(s) is also expanded to all matching terms for the given field and OR-combined using a BooleanQuery.

- re-Isearch places no similar limits but allows for wildcards in paths, terms etc. Its really a different model. The query ""ca*" will search for all the terms that start with "ca". If there is a limit defined in the search for time (which may be set to a limit or allowed to be unlimited) or number of records (which can be set to an absolute number, a number as a proportion of the total number of records in the index or unlimited) the search will stop when that mark is passed (default is unlimited number of records).
    i. re-Isearch does not expand the query into a boolean OR'd expression but searches directly for the query. This is more direct and allows for query structures to be re-used. With Lucene for each change in an index the Query must be re-parsed (Note: the Lucene query parser is NOT thread safe).
    ii. re-Isearch supports not just wildcards but full glob (POSIX 1003.2, 3.13) including some extensions. These can be applied to the entire search expression including path and term components.

- Lucene normally supports only wildcards to the right (prefix queries).
- re-Isearch supports both right and left truncation as well as generic glob expressions.

- Lucene does not normally allow for wildcards in field names
- re-Isearch allows for wildcards (glob expressions) in field names and paths.

- Lucene does not support combinations of phrase and wildcard as in the right truncated phrase search "search optim"*
- re-Isearch allows for wildcards.

12. Proximity
    - Lucene does not really support proximity but a concept of "phrase query slop": the maximum number of full word "moves" required to get all the words next to each other in the proper order. For simple paired expressions like "dog cat"~10 (the ~10 specifies the moves) it comes out as within 10 (or whatever positive integer specified) words.
    - re-Isearch has a concept of proximity. Distance, however, is not measured in words but bytes as in the original indexed document. This makes sense since in marked-up documents what's a word? re-Isearch also has heuristic concepts of near and can also restrict proximity to within a common field instance.

13. Boolean operations

- Lucene does not use the pure boolean information retrieval model or support boolean operators but simulates some of the basic user-side functionality for inclusion and exclusion via a modal prefix model. Lucene has two term prefix ops: "+" (for "must contain"), "-" (for "must not contain"). Terms without a prefix are "can contain". It supports via query parsers an emulation of "AND" "OR" "ANDNOT". The emulation, however, is somewhat quirky and often interprets queries differently than most (other than those familiar with the quirks) would ever expect.
- re-Isearch is overloaded with operators (probably more than most people have ever heard of).

14. Unary operators
    - Lucene has effectively no unary operators. The closest to unary operations are term boost (weight) and "fuzzy" but they are limited to use as term modifiers.
    - re-Isearch has a full-blown set of operators (again from different design considerations) and includes not just a rich set of binary but also unary operators including complement, operators to sort and manipulate sets, boost weight of the expression (according to a number of models) restrict to given fields/paths etc.

15. Query Languages/Interfaces
    - Lucene does not per say have a query language since it contains only terms and modifiers (+,-, weight). These may be processed in any order. There are a number of classes to try to convert other languages into Lucene's model.
    - re-Isearch uses a vector/boolean information retrieval model. Queries are driven by an object oriented automata. These may be created as program objects or parses from any of a number of query languages. At the heart of re-Isearch's model is an RPN stack based language and there are a number a number of classes to convert from other query languages into RPN.

    - Lucene's boolean query class limits the number of clauses (typ. 1024). This makes sense since Lucene too limits the number of terms in a query (typ. 1024).
    - re-Isearch's boolean query class places not limits on the number of clauses, terms, operators etc.

    - Lucene is best when not finding records. Since there are no operators and just terms and predicates that apply to them (either weight or modality) they can without worry be easily rearranged. The query: A -B C D +E for instance can be quickly optimized by the constraints "must have" E and "can't have B.
    - re-Isearch is run by a query automata. It can perform many optimizations but it can't just build upon short circuits and programs will need to run their course except in some simple cases such as all terms "ANDed".

16. Does the position of the matches in the text affect the scoring?
    - In Lucene: No, the position of matches within a field does not affect ranking.
    - In re-Isearch: Depends upon the score normalization model selected at search time. The CosineMetric normalization model, for instance, does use the position of the matches in text to affect the scoring. This is all search time and user selectable.

17. Field differences
    - Lucene lacks diagnostics. Searching even in a field that does not exist just returns no results but without reason. Since fields are *Case Sensitive* in Lucene this is a frequent source of error.

- re-Isearch contains diagnostics.

- Lucene's fields are *Case Sensitive*. There is, to our knowledge, no way to switch it.
- re-Isearch *by default* makes field names and paths *case insensitive* (as the case for SGML, SQL etc). Even through XML is case sensitive (and we were among those that opposed it) we are familiar with no productive XML document types and valid instances with two (or more) siblings differing only in case of their names. While possible in XML

  ```
  <organization><name>BSn</name><NAME>Edward C.
  Zimmermann</NAME></organization>
  ```

  its poor design just as there are reasons why domain names and email addresses too are not case dependent.

18. Structure search
    - Lucene is a traditional inverted index fulltext engine. Its quite good at handling a limited number of fields but is inappropriate for use with arbitary trees.
    - re-Isearch is not based on "Inverted file indexes" and uses other algorithms. It has no limits on the length of terms, their frequency and and can support arbitary structures and paths, including overlap.

    - The granularity of Lucene (unit of retrieval) is the record as defined at the time of indexing.
    - re-Isearch allows for search-time dynamic granularity. The scale of grain (sentence, paragraph, document, chapter, book, collection, source, community, hub, inter-hub bridge, sphere, inter-sphere bridge) is defined by the result of the search and by the query.

    - The product of a search in Lucene is a identification for the record. To extract elements one must load the document (parse etc.) into an object model that supports addressing elements.
    - re-Isearch has no no need for a “middle layer” of content manipulation code. Instead of getting IDs, fetching documents, parsing them, and navigating the DOMs to find required elements, re-Isearch lets you simply request the elements you need and they are returned directly.