Handbook 0.3 (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. From 1994 to 1998 most of the development was centered on the Clearinghouse for Networked Information Discovery and Retrieval (CNIDR) in North Carolina and BSn in Germany. 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 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

Handbook 0.3 (work in progress)

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 it was no longer updated or actively maintained. 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.
- ➔ 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".

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.

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 IB 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-

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 0.3 (work in progress)

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.

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:

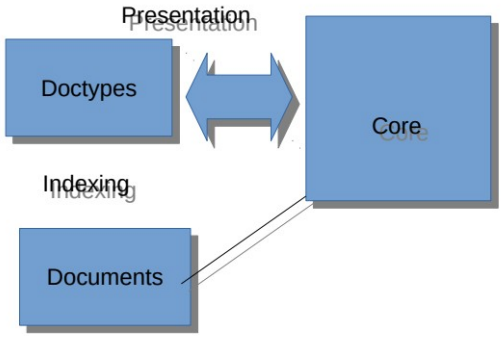
- ✔ 32 or 64-bit POSIX Unix/Linux.
- ✔ Min. 8 MB Physical RAM.
- ✔ Virtual Memory System activated.
- ✔ Choose a faster file system configured for the use case
- ✔ If possible choose faster memory, fast disks, preferably SSDs

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)

Handbook 0.3 (work in progress)

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	IBAN: 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."

Handbook 0.3 (work in progress)

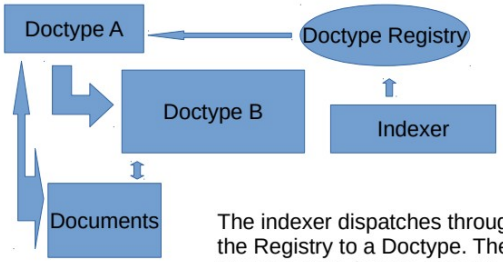
Object Indexes (data type handlers)

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 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.



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

Field Unitification (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.

Handbook 0.3 (work in progress)

V. Indexing

✗ 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 IBAN (includes Checksum)

Handbook 0.3 (work in progress)

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:

```
IB 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.
```

Handbook 0.3 (work in progress)

```
-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[ypes]      // 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 allows a large number of options. To get the available options for the database.ini file one uses the command -help o

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
```

Handbook 0.3 (work in progress)

```
     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>


                                    XMLBASE Class Tree:
                                          DOCTYPE
                                             |
```

Handbook 0.3 (work in progress)

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      // IBAN: 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]
```

Handbook 0.3 (work in progress)

```
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>
```

Handbook 0.3 (work in progress)

```
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.
 -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

Targets

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.

Handbook 0.3 (work in progress)

VII. IB Query Language

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 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”`.

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 (IB 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	
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater then or equal to

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

Handbook 0.3 (work in progress)

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
SIBLING		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

Handbook 0.3 (work in progress)

		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”).

Handbook 0.3 (work in progress)

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"'

Comparison to CQL

CQL query consist, like the IB 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	**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

IX. 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="IB 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) .

Handbook 0.3 (work in progress)

X. 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.

Handbook 0.3 (work in progress)

XI. 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:

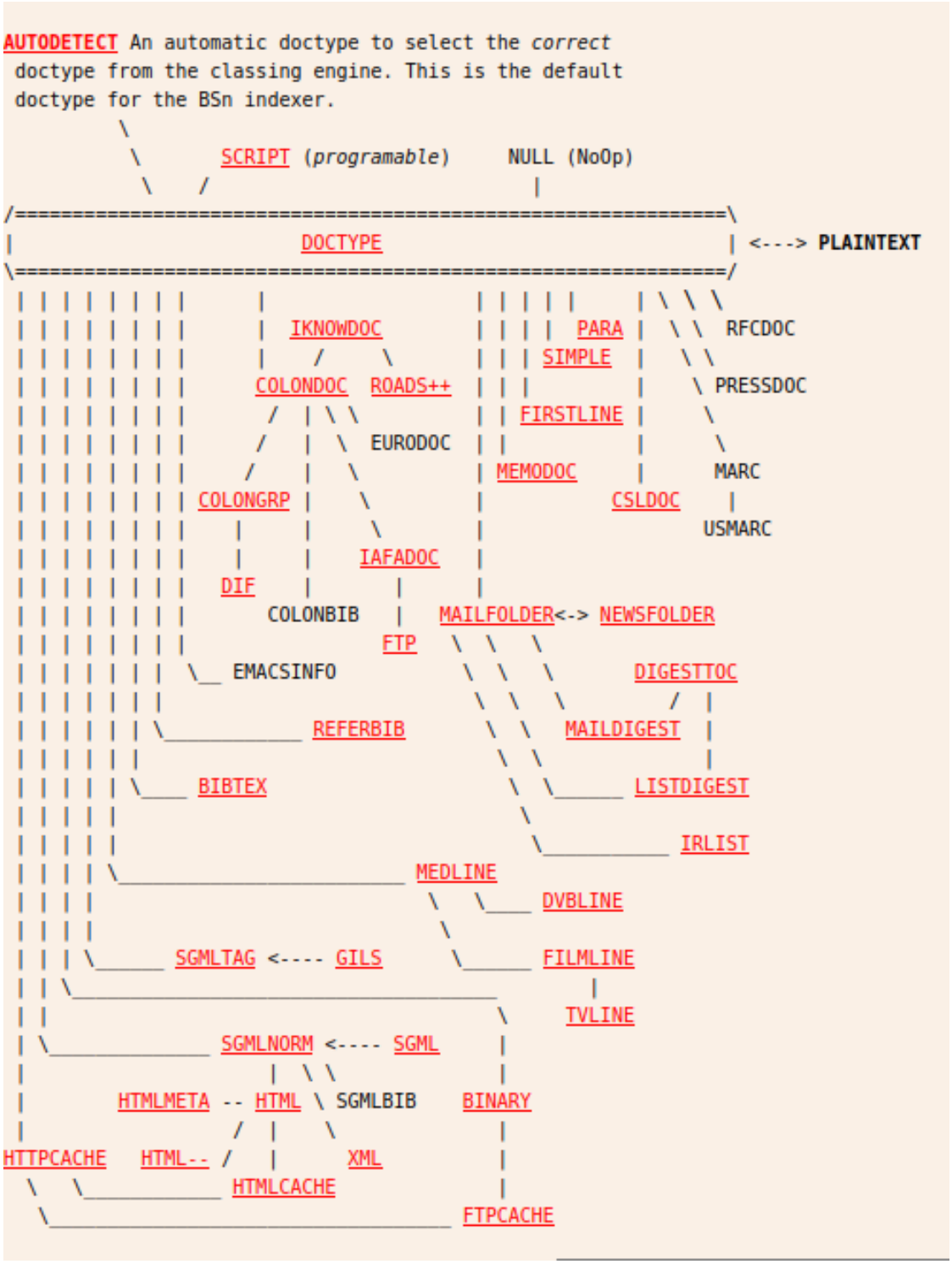
```
query = "beschaffungsmanagement:3 OR beschaffungsmarketing:3 OR beschaffungsmarkt:3
OR beschaffungsplanung:3 OR beschaffungsprozesse:3 OR (deterministische:3 FOLLOWS
beschaffung:3) OR einkaufspolitik:3 OR (stochastische:3 FOLLOWS beschaffung:3) OR
strategien:2 OR strategie OR (c:3 FOLLOWS teilemanagement:3) 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.

Handbook 0.3 (work in progress)

XII. Doctypes

This is an old map of the doctypes



Handbook 0.3 (work in progress)

The root class of all document handlers is the “DOCTYPE” class. 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)
```

Handbook 0.3 (work in progress)

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.

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.

Handbook 0.3 (work in progress)

[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.

Handbook 0.3 (work in progress)

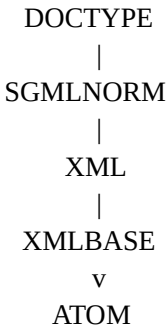
✗ Options in .ini:

```
[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".*

BIBCOLON

```
COLONDOC for bibliographies
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
```

```
BIBCOLON Class Tree:
      DOCTYPE
         |
      METADOC
         |
      COLONDOC
         v
      BIBCOLON
```

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

Handbook 0.3 (work in progress)

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 {IB}}\!T\!_{\displaystyle E}\!X} }} {\mathrm {B{\scriptstyle {IB}}\!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
    }
```

Handbook 0.3 (work in progress)

✗ 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",
}
```

✗ 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

Handbook 0.3 (work in progress)

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".

Handbook 0.3 (work in progress)

BINARY

The BINARY doctype has been designed for the management of multimedia files (audio/graphics), CAD drawings and non-textual databases.

```
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).

COLONDOC

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

Handbook 0.3 (work in progress)

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.

Colon Records:

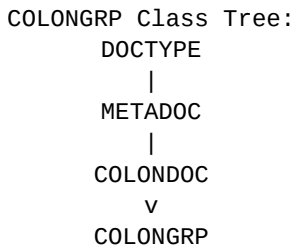
```
TAG1: ...
.....
TAG2: ...
TAG3: ...
...
....
```

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.



Handbook 0.3 (work in progress)

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.`

Handbook 0.3 (work in progress)

`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`.

NEWSML

The doctype is designed to handle the newml 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/

```
NEWSML Class Tree:
         DOCTYPE
            |
        SGMLNORM
            |
           XML
            |
         XMLBASE
            |
         GILSXML
            v
         NewsML
```

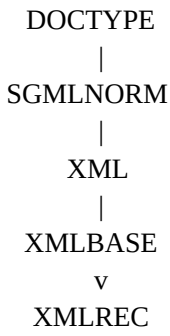
The format is basically XML but it also understands a few details about NewML such as that elements "DateAndTime", "DateId", "DateLabel", "DateLineDate", "FirstCreated" and "ThisRevisionCreated" are date datatypes.

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";

Handbook 0.3 (work in progress)

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.



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..

XIII. Tuning

XIV. C++ API