



XML Handbook

Insuma GmbH

20th March 2003

Insuma reserves the right to make changes in the specifications and other information contained in this publication without prior notice. In case of doubt, the reader should consult Insuma to determine whether any such changes have been made. The software described in this document is furnished under a license and may be used or copied in accordance with the terms of such license.

Trademarks

Insuma and its logo are registered trademarks of Insuma GmbH. All other product and company names are either trademarks or registered trademarks of their respective companies.

Insuma GmbH
Sand 13
72076 Tübingen
Germany
Phone: +49 (7071) 3 65 96 10
Fax: +49 (7071) 3 65 96 14
E-mail: info@insuma.de
<http://www.insuma.de>

Contents

1 Overview	5
2 Search interface	6
2.1 Query DTD	6
2.2 Conditions	6
2.3 Boolean queries	8
2.4 Attributes	10
2.5 Result DTD	11
3 SOAP search interface	13
3.1 Operations	13
3.2 Input messages	13
3.3 Output messages	14
4 Indexing interface	15
4.1 Indexing DTD	15
4.2 Document attributes	15
5 Default attributes for HTML documents	18

Executive summary

This document describes the XML interfaces between Insuma*Focus* and outer world. It includes detailed description of XML tags and structure on Search and on Indexer sides. Line by line explanations as well as examples help in understanding the XML interfaces. This document is targeted to technical specialists in the companies - clients of Insuma GmbH.

1 Overview

Insuma*Focus* uses XML interfaces for interactions with the outside world.

There are two independent interfaces: the search interface and the document import interface.

Before going into details of each interface the underlying document model must be explained. The logical representation of a document is flat. A document is uniquely identified by its URL. A document normally has a number of attributes like 'author', 'body_text', 'creation_date', 'category', etc. Each attribute can have one of the following types:

- text (the only type where full-text search is possible)
- date (in fact, date-time)
- string (supports regular expression matching, no full-text)
- int
- float
- location (represents a geographic location)
- noindex (stored and returned as is, can't be used in any search expression).

Some attributes can be defined as multi-value. For example, a document may have several authors or belong to several categories. Exact attribute set is easily configurable via a configuration file. In case of ASP service this is done by Insuma, in case of in-house solution by the user.

2 Search interface

Search is performed in a SOAP-like manner¹: an XML-encoded query is sent to *InsumaFocus* over HTTP (or some other reliable communications channel) and the XML-encoded response is returned via the same channel.

2.1 Query DTD

Exact query DTD depends on the attribute set defined for the documents and varies between installation (see the discussion above). However, it has a common structure discussed here. An oversimplified DTD for queries is shown in Figure 2.1.

2.2 Conditions

Queries impose conditions on the document attributes with the help of `<condition>` elements. Each `<condition>` has 3 required attributes:

- `attr` (e.g., 'body') specifies on which attribute of the document the condition is imposed
- `predicate` (e.g., 'lt') specifies the sort of condition imposed with respect to the value.
- `value` is the second argument of the predicate.

For example, the condition

```
<condition predicate="ge" attr="modification_date" value="2002-07-01"/>
```

demands that the document must be modified not later than in July 2002 in order to be included in the result set. (The example implies that the database scheme defines document attribute 'modification_date' of type 'date').

The semantics of the predicates is as follows.

¹This is not exactly SOAP because of minor formalities like namespaces not used. A compliant SOAP interface can be produced if ordered.

```

<!ELEMENT query ((condition|or|and)+)>
<!ATTLIST query
    max_results CDATA          '10'
    start_from  CDATA          '1'
    show_attrs  NMTOKENS      #IMPLIED
>

<!ELEMENT or ((condition|or|and)+)>
<!ELEMENT and ((condition|or|and)+)>
<!ELEMENT condition EMPTY>
<!ATTLIST condition
    predicate (match|bmatch|in|lt|le|eq|ge|gt)      #REQUIRED
    attr      NMTOKEN                               #REQUIRED
    value     CDATA                                 #REQUIRED
>

```

Figure 2.1: Simplified query DTD

‘match’ means that the similarity between the text in the document field and the ‘value’ is above zero. This predicate is present in virtually every meaningful search. The similarity is fuzzily defined and calculated using information retrieval techniques. Higher similarity means increased chances that the document is relevant to the user. Exact types of information retrieval techniques (morphology, type error tolerance, phonetics) used for the ‘match’ predicate can be configured on the per-attribute basis. They will be described in a document specific to your installation. The following example will find all documents whose `body` attribute contains ‘monty’ or ‘python’. Documents containing these words many times, or where ‘monty’ occurs near ‘python’ in the text are moved to the top of the result list.

```
<condition predicate="match" attr="body" value="Monty Python"/>
```

‘bmatch’ (that means boolean match) has the same meaning as ‘match’ but it can additionally understand boolean statements (negation, and, or, substring) in the ‘value’ part. See more detailed description in the next chapter. Since ‘bmatch’ does not calculate ranks of the documents, one should use it along with match condition if qualified ranking required.

```

<condition predicate="match" attr="body" value="Monty Python"/>
<condition predicate="bmatch" attr="body"
value="Monty -Python"/>

```

‘like’ means the *whole* string matches the pattern provided. The pattern has standard SQL syntax, i.e., matching is case-insensitive and ‘%’ matches any (including empty) sequence of characters. An example below matches ‘VW456-789/2’.

text	match, bmatch, like (like can be slow!)
string	lt, le, eq, ge, gt, like
int	lt, le, eq, ge, gt
float, date	lt, le, ge, gt (exact equality is dangerous)

Table 2.1: Predicates for document attributes

```
<condition predicate="like" attr="part_number" value="vw4%/2" />
```

In certain cases use of `like` may lead to full table scans, which can be slow. This happens only when any of the following is true:

1. The predicate is applied to an attribute of type `text`. Use it against an attribute of type `string` if the search takes too long, replicate the `text` attribute in question as a `string` attribute with another name.
2. The pattern starts with a wildcard, i.e., with ‘%’ character. Consider using a `text` attribute and the `match` predicate instead.

Still, full scans may be acceptable when the values of the attribute in question tend to be short and the number of documents containing the attribute is relatively small (e.g., 50000 strings 80 characters long allow for full scan in about 0.2 seconds).

‘lt’, ‘le’, ‘eq’, ‘ge’, and ‘gt’ have their traditional FORTRAN sense, i.e., ‘<’, ‘<=’, ‘=’, ‘>=’, and ‘>’ correspondingly.

```
<condition predicate="le" attr="price" value="100" />
```

means ‘price <= 100’.

Predicates that can be applied to individual attributes depend on the attribute type (and the generalised simplified DTD does not reflect this fact). They are summarised in the table 2.1.

2.3 Boolean queries

Conditions imposed on individual attributes can be combined with boolean operators AND, OR (expressed as `<and>` and `<or>` elements correspondingly).

The elements also imply brackets, see the examples.

```
<query>
  <or>
    <condition A/>
```

```
        <condition B/>
        <condition C/>
    </or>
</query>
```

means 'A or B or C'.

```
<query>
  <and>
    <condition A/>
    <or>
      <condition B/>
      <condition C/>
    </or>
  </and>
</query>
```

means 'A and (B or C)'

The `<query>` element itself implies 'and', so the latest example is equivalent to the shorter form:

```
<query>
  <condition A/>
  <or>
    <condition B/>
    <condition C/>
  </or>
</query>
```

Negation is supported for individual text attributes only, e.g.,

```
<condition predicate="bmatch" attr="title" value="bush -president"/>
```

Combination is rank-aware, i.e., better matches come at the top of the list. Weights can be configured for individual text attributes that determine how strongly these attributes influence the end ranking. E.g., an exact match in the `title` can be configured to be 10 times more important than a phonetic match in the `body_phonetic`. Weights are currently assigned by Insuma in a configuration file. Dynamic weighting in the query can be supported on request.

2.4 Attributes

The 'query' element supports several optional attributes:

- `max_results` specifies the maximum number of results allowed in the result set. The actual number of the returned hits may naturally be smaller, but not greater. The default value is 10, the maximum is normally configured to 100.
- `start_from` allows for retrieving further screens of results. If `start_from=50` and `max_results=10`, the documents positioned from 50 to 59 in the ranking will be returned in the result set. The default value is 1, the maximum is normally configured to 400.
- `show_attrs` allows for specifying a set of attributes to be returned in the result. These must be space-separated valid attribute names. The default is an empty set; in this case only URLs are returned for the hits.

A full example of a simple query is provided below.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE query SYSTEM "insuma_search_html.dtd">
<query max_results="5" show_attrs="summary">
  <condition predicate="match" attr="body_morpho"
    value="hochschule Heidelberg"/>
</query>
```

The search is performed in the whole document body, with morphology turned on and phonetics off (once again, the settings may vary between installations).

XML representation of results has the form like shown in Figure 2.2. The `<result>` element has the following attributes:

- `hits` is the number of `<document>` elements within the `<result>` element. It is provided for the convenience of user interfaces programmers only, so that they know the number of returned hits before parsing the whole XML response.
- `total_hits` is the total number of documents relevant to the query. It can be either an exact value, or an estimate. The estimate is returned when the number of matches is several times larger than the user-requested maximum.
- `total_hits_exact` can be 0 or 1. The value of 1 indicates that `total_hits` contains the exact number of matching documents found, and 0 indicates it is just an estimate.

The elements nested within `<document>...</document>` can come only from the list supplied as the `show_attrs` attribute of the `<query>` element. The list of allowed elements changes from installation to installation, please consult the annex supplied to your company.

```
<result hits="10" total_hits="34" total_hits_exact="1">
  <!-- Optionally the original query may be returned
  -->
  <query show_attrs="title author summary">
    <condition predicate="match" attr="body"
      value="search engine"/>
  </query>
  <document url="http://www.insuma.de/">
    <title>Search engine for your portal</title>
    <summary>...InsumaFocus is a portal search
engine...
    </summary>
    <author>Babanin, Alex</author>
    <author>Heuser, Udo</author>
  </document>
  <document>
  ...
  </document>
</result>
```

Figure 2.2: Result set representation

2.5 Result DTD

A sample DTD for the result set is provided in Figure 2.3.

```
<!-- This DTD describes a variant of InsumaFocus result
      set.
      It was generated for collection 'if9932'.
-->

<!ELEMENT result (document*|error)>
<!-- all the attributes are in fact integer numbers -->
<!ATTLIST result
      hits          NMTOKEN      #REQUIRED
      total_hits    NMTOKEN      #REQUIRED
      total_hits_exact (0|1)      '0'
>

<!-- The actual attributes present in the <document>
      are
      governed by the 'show_atts' attribute of the
      <query>.
-->
<!ELEMENT document (title|headings|keywords|description|body)*>
<!ATTLIST document
      url           CDATA        #REQUIRED
      url_i         CDATA        #IMPLIED
      seen_first    NMTOKENS     #IMPLIED
      seen_last     NMTOKENS     #IMPLIED
>

<!ELEMENT title (#PCDATA)>
<!ELEMENT headings (#PCDATA)>
<!ELEMENT keywords (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT body (#PCDATA)>

<!ELEMENT error (#PCDATA)>
<!ATTLIST error
      class (input|internal) 'internal'
>

<!-- the text within the <error> element is a
      human-readable
      error message
-->
```

Figure 2.3: Sample result set DTD

3 SOAP search interface

An implementation of real SOAP interface is supported in some installations, if ordered. Please consult your documentation or Insuma GmbH to obtain Web Service Definition (WSDL) file for your system, if available. The interface is quite similar to the one defined in the previous chapter.

3.1 Operations

There are three very similar supported by the interface. They do essentially the same thing, and the choice of the operation to use depends on the programmer's preferences. The operations are `search`, `search_xml`, and `search_xml_str`. They all return a message of type `Result` described in section 3.3. The difference is in the representation of the query.

This representation is always scary-looking, though not necessarily very complex. The root cause of the difficulty is the need to express multiple conditions imposed on multiple document attributes and the logical relationships between them.

`search_xml` expects an XML entity `query`, conforming to the specification provided in the previous chapter. Producing XML is normally easier than parsing it, so just string manipulation functions may be sufficient for generating quite complex queries. This should be the easiest operation to use.

`search_xml_str` differs from `search_xml` only by encoding of the query. The former transfers the query encoded as an XML entity, i.e., the tags of `query` travel over the wire as is. The latter encodes the query as a normal SOAP string, i.e., `xsd:string`. The need for this may arise if the SOAP library used by the client does not support `xsd:ENTITY` type properly (or conveniently).

`search` allows for representing queries with programming language constructs like structures and arrays. Mapping into the XML constructs required by SOAP is done by the platform. However, deeply nested programming language constructs are required to express complex queries. The `Query` input message used by `search` is described below.

3.2 Input messages

The `Query` input message intends to follow closely the concepts described in section 2.

Elements of type `SimpleCondition` form the basic building blocks of queries. They correspond to the `<condition>` elements described in section 2. Each `SimpleCondition` contains exactly one of each of the following elements: `predicate`, `attr`, and `value`. Their semantics and allowed values are exactly the same as the semantics of the corresponding attributes of the `<condition>` elements. Simple conditions are used to express statements like “Author name matches ‘Hopkins’”.

Elements of type `ComplexCondition` can be used to combine logically any number of other conditions, both simple and complex. All conditions inside a complex condition are combined using the same logical operation, “and” or “or”. The operation used must be specified in the mandatory element `op`. `SimpleCondition` elements that are parts of a `ComplexCondition` are put into the `simple` array. Nested `ComplexCondition` elements are put into the `complex` array. Both arrays may be empty or non-empty independently of each other.

The `Query` type is just a sequence of one mandatory `ComplexCondition` element and several optional elements. The optional elements are `max_results`, `start_from`, and `show_attrs`. They are direct analogues of the corresponding optional attributes of the `<query>` element described in section 2.

3.3 Output messages

The purpose of using SOAP is often to relieve the programmer from the task of XML parsing. The results come in native constructs of the programming language used, like arrays and records. Since the structure of the result set is much simpler than the query, the corresponding SOAP representation is also more straightforward.

The `Result` output message and the corresponding `Result` type are very similar to the `<result>` element described in section 2. The elements `hits`, `total_hits`, and `total_hits_exact` have the same meaning as the corresponding attributes of `<result>`. The `documents` elements of `Result` have `Document` type and contain the actual hits.

The `Document` type is similar to the `<document>` element described in section 2. It consists of exactly one `url` element (corresponding to the `url` attribute of `<document>`) and a sequence of name-value pairs `properties`. Every name-value pair has type `NameValuePair` and consists of exactly two elements of type `string`: `name` and `value`. They represent the name and value of each document attribute, like `name = “title”, value = “Hamlet”`. Multi-value document attributes are represented by several name-value pairs with the same name element.

4 Indexing interface

Indexing interface is used for communication between Insuma search engine and the text source (back-end database, web site, etc.).

A set of documents being searched is called a collection. Each collection has one or several sources like the ones mentioned above, that update periodically. To provide for fast searches, the documents residing at the sources are indexed on the server running *InsumaFocus*. Search queries are always evaluated only against this local index.

The index must be regularly updated to reflect the changes at the document sources. It happens when a specially constructed XML file describing the changes is fed into *InsumaFocus*. Each `<document>` element of the file describes one document that is inserted, modified or deleted. We call such a file 'import file'. The command like

```
if_collection.py -n your_collection_name import_file
```

run at the server where *InsumaFocus* is installed applies the changes.

4.1 Indexing DTD

Supported document attributes are customer-specific. Import file DTD for your company must be defined in a separate document, e.g., an annex supplied by Insuma. If you have shell access to the server that runs your installation of *InsumaFocus*, you can obtain the right DTD by executing

```
if_collection.py -n my_collection_name --export-dtd > my_import.dtd
```

The import file must conform to this DTD. An example of such a DTD for third party HTML documents is given in table 4.1.

4.2 Document attributes

The generic structure of import files, however, does not change between installation. Basically, a collection consists of documents. You must specify the collection name in the name attribute

```
<!-- This DTD describes a variant of InsumaFocus
      collection update file.  It was generated for
      collection 'html'.  -->

<!ELEMENT collection (document*)>
<!ATTLIST collection
      name          NMTOKEN          #REQUIRED
      date          NMTOKENS         #REQUIRED
>

<!ELEMENT document (
      title|title_morpho|title_phonetic|
      headings|headings_morpho|headings_phonetic|
      keywords|keywords_morpho|
      description|description_morpho|
      body|body_morpho|body_phonetic|
      modified)*
>

<!ATTLIST document
      url           CDATA            #REQUIRED
      status        (ok|same|deleted) 'ok'
      url_i         CDATA            #IMPLIED
      seen_first    NMTOKENS         #IMPLIED
      seen_last     NMTOKENS         #IMPLIED
>

<!ELEMENT modified (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT headings (#PCDATA)>
<!ELEMENT keywords (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT body (#PCDATA)>
<!ELEMENT title_morpho (#PCDATA)>
<!ELEMENT headings_morpho (#PCDATA)>
<!ELEMENT keywords_morpho (#PCDATA)>
<!ELEMENT description_morpho (#PCDATA)>
<!ELEMENT body_morpho (#PCDATA)>
<!ELEMENT title_phonetic (#PCDATA)>
<!ELEMENT headings_phonetic (#PCDATA)>
<!ELEMENT body_phonetic (#PCDATA)>
```

Figure 4.1: Example of import DTD

of the `<collection>` tag. It must match the name of the collection into which you import the documents, this is a safety feature. You must also specify the creation date of the XML file in the `date` attribute of `<collection>`. The format for the date is "YYYY-mm-dd [hh:mm:ss]" (square brackets mean that specifying time is optional, they should not be present in the attribute value).

Every `<document>` must have its `url` attribute set. This is the URL the users see in the result set. Sometimes you may want to use shadow pages, i.e., index something different from what the end user sees (e.g., the database records from which a dynamic page is generated instead of browser-specific Javascript). The alternative source of the document can be specified in the `url_i` attribute. It will never show up in a result set, but may be used for re-indexing. `seen_first` and `seen_last` attributes may be set to the date the document with the same URL and content was observed for the first and the last time. Their value defaults to the value of the `date` attribute of `<collection>`. The last modification date of the document is thus not later than `seen_first`, and the URL was checked to be up-to-date on `seen_last`.

The `status` attribute determines what happens with the document entry in the index:

- ‘ok’, the default value, means that the document is new or was modified. The document entry in the index is replaced with the one provided in the `<document>...</document>` element.
- ‘same’ means that the document did not change since the last update of the index. All the elements nested in `<document>...</document>` are ignored, the old values are kept. The only attribute that is changed in this case is `seen_last`. The feature may be needed when the document source does not report document modification dates reliably, and periodical polling is the only way to obtain this information.
- ‘deleted’ means the document is no longer available at the URL specified. The corresponding entry is removed from the index.

The document attributes are provided in elements of their own. Multi-value document attributes like `<author>` may occur several times within the same `<document>`. See your DTD for the list of elements allowed within a `<document>` element.

5 Default attributes for HTML documents

This chapter describes XML, which is derived from HTML formatting of source documents. Such XML is used by default when the source documents are pulled directly from a web site.

- `title`, `title_morpho`, `title_phonetic` represent the document title extracted from the `<title>` element of the indexed HTML document. `'_morpho'` used in the name of the attribute makes the match predicate use German morphology. In the same manner, `'_phonetic'` makes match predicate use phonetic search. Typing error correction is always on.
- `headings`, `headings_morpho`, `headings_phonetic` represent the headings encoded as `<h1>` – `<h6>` in the HTML document. The same `'_morpho'` and `'_phonetic'` convention is hold.
- `description`, `description_morpho` represent META DESCRIPTION extracted from the HEAD section of the HTML document if present.
- `keywords`, `keywords_morpho` represent META KEYWORDS extracted from the HEAD section of the HTML document if present.
- `body`, `body_morpho` represent all the text used in the BODY section of the HTML document plus title. All text from `<title>` and `<headings>` is repeated in `<body>`. In the same manner, all text from `<title_morpho>` and `<headings_morpho>` is repeated in `<body_morpho>`.
- `category` is a string specified for each document via a meta tag or in the form of a rule via the control centre. Has the type `string`. To specify the category in the body of the document, one must insert a meta tag like follows:

```
<head>
<!-- the usual things like <title> skipped -->
<!-- Here comes the category -->
<meta name="category" content="Diabetes">
<!-- more than one category can be specified for one document
      in separate meta tags -->
<meta name="category" content="Blutdruck">
<!-- Life goes on as normal -->
</head>
```

Categories defined in the manner described above can be later used in the search side: search limited to a category or a set of categories. The control centre allows for assigning categories to subdirectories, please consult the *InsumaFocus* Specification for details.