Specifying and Using Application (Database) Profiles

Index Data, `info@indexdata.dk` $Revision$

YAZ includes a subsystem to manage complex database records, driven by a set of configuration tables that reflect a given profile. Multiple database profiles can coexeist in the same server, or even the same database. The record management system is responsible for associating a given record with a specific profile, and processing it accordingly. This document describes the various file formats for data and configuration files which are used by the module.

Contents

1 Warnings

- The subsystem descibed herein is under development. Not everything may work exactly as decribed, and details of the interface may change as the module matures.

- The exact workings of the subsystem may depend on the application in which it is used. This document focuses on the use of the module in the **Zebra** information server which is distributed by Index Data as an independent package.

2 Introduction

The retrieval facilities of Z39.50 are extremely flexible and powerful. They allow any level of structuring of database records. They allow controlled re-use of attribute sets (for searching) and tag sets (for retrieval) between application profiles; they allow precise selection of the desired sub-elements of a database record; they allow different variants of a given data element to be represented and selected in a structured way; and finally they allow the exchange of any type and amount of data to be represented in a single database record.

These powerful retrieval facilities are a recent addition to the protocol, and along with the flexible searching facilities, they make the protocol an extremely capable tool for precise, structured access to information systems. The retrieval facilities add new levels of flexibility and control to the protocol, which add to its value outside of its traditional domain of the library systems world.

The new facilities, however, also add new complexity to the protocol, which is already troubles by a too-steep learning curve. We have seen many good projects severely hindered or even thwarted by the sheer complexity of implementing the Z39.50 protocol.

At the same time, we feel that the most complex and powerful facilities of the protocol (Explain, structured retrieval, etc.), are also what the protocol needs to become more widespread, and to fulfill what we perceive to be its most noble potential: To provide everybody with standardised, well-structured access to the information resources of the world.

The purpose of **YAZ**, then, and of this module as well, is to *simplify* the use of the protocol for programmers and administrators, by providing simple APIs and configuration systems to access the functionality of the protocol. The **Retrieval** module deals specifically with the advanced retrieval functions which were added to the protocol with version 3, or Z39.50-1994.

3 Overview

3.1 External Data (record) Representation

The **Retrieval** module will eventually support a wide range of input formats, ranging from MARC data to USENET news archives. This section introduces what we think of as the *canonical* format - the one that gives the most general access to the various elements of the retrieval functionality.

The basic model presented by the Z39.50 retrieval system is that of a recursively defined tree structure, containing a list of tagged elements, which may in turn contain either data or more lists of tagged elements, and so forth.

We elect to represent this structuring externally by using an "SGML-like" syntax. The *internal* representation will be discussed later.

Consider a record describing an information resource (such a record is sometimes known as a *locator record*). It might contain a field describing the distributor of the information resource, which might in turn be partitioned into various fields providing details about the distributor, like this:

```
<Distributor>
    <Name> USGS/WRD </Name>
    <Organization> USGS/WRD </Organization>
    <Street-Address>
        U.S. GEOLOGICAL SURVEY, 505 MARQUETTE, NW
    </Street-Address>
    <City> ALBUQUERQUE </City>
    <State> NM </State>
    <Zip-Code> 87102 </Zip-Code>
    <Country> USA </Country>
    <Telephone> (505) 766-5560 </Telephone>
</Distributor>
```

This is how data that the retrieval module reads from an input file might look.

Depending on the database profile that is being used, it is likely that the data won't look like this when it's transmitted from the server to the client, however. Typically, the client will prefer to receive the data in a more rigid syntax, such as USMARC or GRS-1. To save transmission time and avoid ambiguities of language, the individual tags or field names, above, might be translated into numbers which are known by both the client and the server (by referring to a tag set).

The retrieval module supports various types of conversions that might be carried out by the server based on requests from the client. To do this, it needs a set of configuration files to describe the application profile that the given record adheres to.

CAUTION: Because the tables described below serve the dual purpose of representing an external application profile and an internal database profile, the terminology and structuring used will sometimes be somewhat different from the one suggested in the the Z39.50-1995.

3.2 The Abstract Syntax

The abstract syntax definition (ARS) is the focal point of the application profile description. For a given profile, it may state any or all of the following:

- The object identifier of the database schema associated with the profile, so that it can be referred to by the client.
- The attribute set (which can possibly be a compound of multiple sets) which applies in the profile. This is used when indexing and searching the records belonging to the given profile.
- The Tag set (again, this can consist of several different sets). This is used when reading the records from a file, to recognize the different tags, and when transmitting the record to the client - mapping the tags to their numerical representation, if they are known.
- The variant set which is used in the profile. This provides a vocabulary for specifying the *forms* of data that appear inside the records.
- Element set names, which are a shorthand way for the client to ask for a subset of the data elements contained in a record. Element set names, in the retrieval module, are mapped to *element specifications*, which contain information equivalent to the *Espec-1* syntax of Z39.50.
- Map tables, which may specify mappings to *other* database profiles, if desired.

- Possibly, a set of rules describing the mapping of elements to a MARC representation.
- A list of element description (this is the actual ARS of the profile), which lists the ways in which the various tags can be used and organized hierarchically.

Several of the entries above simply refer to other files, which describe the given objects.

4 The Configuration Files

This section describes the syntax and use of the various tables which are used by the retrieval module.

The number of different file types may appear daunting at first, but each type corresponds fairly clearly to a single aspect of the Z39.50 retrieval facilities. Further, the average database administrator who's simply reusing an existing profile for which tables already exist, shouldn't have to worry too much about these tables.

4.1 The Abstract Syntax (.abs) Files

The name of this file type is slightly misleading, since, apart from the actual abstract syntax of the profile, it also includes most of the other definitions that go into a database profile.

When a record in the canonical, SGML-like format is read from a file or from the database, the first tag of the file should reference the profile that governs the layout of the record. If the first tag of the record is `<gils>`, the system will look for the profile definition in the file `gils.abs`. Profile definitions are cached, so they only have to be read once during the lifespan of the current process.

The file may contain the following directives:

name *symbolic-name*

This provides a shorthand name or description for the profile. Mostly useful for diagnostic purposes.

reference *OID-name*

The reference name of the OID for the profile. The reference names can be found in the **util** module of **YAZ**.

attset *filename*

The attribute set that is used for indexing and searching records belonging to this profile.

tagset *filename*

The tag set (if any) that describe that fields of the records.

varset *filename*

The variant set used in the profile.

maptab *filename*

(repeatable) This points to a conversion table that might be used if the client asks for the record in a different schema from the native one.

marc ***filename***

Points to a file containing parameters for representing the record contents in the ISO2709 syntax. Read the description of the MARC representation facility below.

esetname ***name filename***

(repeatable) Associates the given element set name with an element selection file. If an (@) is given in place of the filename, this corresponds to a null mapping for the given element set name.

elm ***path name attribute***

(repeatable) Adds an element to the abstract record syntax of the schema. The *path* follows the syntax which is suggested by the Z39.50 document - that is, a sequence of tags separated by slashes (/). Each tag is given as a comma-separated pair of tag type and -value surrounded by parenthesis. The *name* is the name of the element, and the *attribute* specifies what attribute to use when indexing the element. A ! in place of the attribute name is equivalent to specifying an attribute name identical to the element name. A - in place of the attribute name specifies that no indexing is to take place for the given element.

NOTE: The mechanism for controlling indexing is inadequate for complex databases, and will probably be moved into a separate configuration table eventually.

The following is an excerpt from the abstract syntax file for the GILS profile.

```
name gils
reference GILS-schema
attset gils.att
tagset gils.tag
varset var1.var

maptab gils-usmarc.map

# Element set names

esetname VARIANT gils-variant.est  # for WAIS-compliance
esetname B gils-b.est
esetname G gils-g.est
esetname W gils-b.est
esetname F @

elm (1,10)              rank                        -
elm (1,12)              url                         -
elm (1,14)              localControlNumber     Local-number
elm (1,16)              dateOfLastModification Date/time-last-modified
elm (2,1)               Title                       !
elm (4,1)               controlIdentifier      Identifier-standard
elm (2,6)               abstract               Abstract
elm (4,51)              purpose                     !
elm (4,52)              originator                  -
elm (4,53)              accessConstraints           !
elm (4,54)              useConstraints              !
elm (4,70)              availability                -
```

```
elm (4,70)/(4,90)       distributor                -
elm (4,70)/(4,90)/(2,7) distributorName            !
elm (4,70)/(4,90)/(2,10 distributorOrganization    !
elm (4,70)/(4,90)/(4,2) distributorStreetAddress   !
elm (4,70)/(4,90)/(4,3) distributorCity            !
```

4.2 The Attribute Set (.att) Files

This file type describes the **Use** elements of an attribute set. It contains the following directives.

name ***symbolic-name***

This provides a shorthand name or description for the attribute set. Mostly useful for diagnostic purposes.

reference ***OID-name***

The reference name of the OID for the attribute set. The reference names can be found in the **util** module of **YAZ**.

ordinal ***integer***

This value will be used to represent the attribute set in the index. Care should be taken that each attribute set has a unique ordinal value.

include ***filename***

This directive, which can be repeated, is used to include another attribute set as a part of the current one. This is used when a new attribute set is defined as an extension to another set. For instance, many new attribute sets are defined as extensions to the **bib-1** set. This is an important feature of the retrieval system of Z39.50, as it ensures the highest possible level of interoperability, as those access points of your database which are derived from the external set (say, bib-1) can be used even by clients who are unaware of the new set.

att ***att-value att-name [local-value]***

This repeatable directive introduces a new attribute to the set. The attribute value is stored in the index (unless a *local-value* is given, in which case this is stored). The name is used to refer to the attribute from the *abstract syntax*.

This is an excerpt from the GILS attribute set definition. Notice how the file describing the *bib-1* attribute set is referenced.

```
name gils
reference GILS-attset
include bib1.att
ordinal 2

att 2001                distributorName
att 2002                indexTermsControlled
att 2003                purpose
att 2004                accessConstraints
att 2005                useConstraints
```

4.3 The Tag Set (.tag) Files

This file type defines the tagset of the profile, possibly by referencing other tag sets (most tag sets, for instance, will include tagsetG and tagsetM from the Z39.50 specification. The file may contain the following directives.

name ***symbolic-name***

This provides a shorthand name or description for the tag set. Mostly useful for diagnostic purposes.

reference ***OID-name***

The reference name of the OID for the tag set. The reference names can be found in the **util** module of **YAZ**.

type ***integer***

The type number of the tag within the schema profile.

include ***filename***

(repeatable) This directive is used to include the definitions of other tag sets into the current one.

tag ***number names type***

(repeatable) Introduces a new tag to the set. The *number* is the tag number as used in the protocol (there is currently no mechanism for specifying string tags at this point, but this would be quick work to add). The *names* parameter is a list of names by which the tag should be recognized in the input file format. The names should be separated by slashes (/). The *type* is th recommended datatype of the tag. It should be one of the following:

- structured
- string
- numeric
- bool
- oid
- generalizedtime
- intunit
- int
- octetstring
- null

The following is an excerpt from the TagsetG definition file.

```
name tagsetg
reference TagsetG
type 2

tag     1       title           string
tag     2       author          string
tag     3       publicationPlace string
```

```
tag     4       publicationDate string
tag     5       documentId      string
tag     6       abstract        string
tag     7       name            string
tag     8       date            generalizedtime
tag     9       bodyOfDisplay   string
tag     10      organization    string
```

4.4 The Variant Set (.var) Files

The variant set file is a straightforward representation of the variant set definitions associated with the protocol. At present, only the *Variant-1* set is known.

These are the directives allowed in the file.

name ***symbolic-name***

This provides a shorthand name or description for the variant set. Mostly useful for diagnostic purposes.

reference ***OID-name***

The reference name of the OID for the variant set, if one is required. The reference names can be found in the **util** module of **YAZ**.

class ***integer class-name***

(repeatable) Introduces a new class to the variant set.

type ***integer type-name datatype***

(repeatable) Addes a new type to the current class (the one introduced by the most recent **class** directive). The type names belong to the same name space as the one used in the tag set definition file.

The following is an excerpt from the file describing the variant set *Variant-1*.

```
name variant-1
reference Variant-1

class 1 variantId

  type  1       variantId               octetstring

class 2 body

  type  1       iana                    string
  type  2       z39.50                  string
  type  3       other                   string
```

4.5 The Element Set (.est) Files

The element set specification files describe a selection of a subset of the elements of a database record. The element selection mechanism is equivalent to the one supplied by the *Espec-1* syntax of the Z39.50

specification. In fact, the internal representation of an element set specification is identical to the *Espec-1* structure, and we'll refer you to the description of that structure for most of the detailed semantics of the directives below.

NOTE: Not all of the Espec-1 functionality has been implemented yet. The fields that are mentioned below all work as expected, unless otherwise is noted.

The directives available in the element set file are as follows:

defaultVariantSetId ***OID-name***

If variants are used in the following, this should provide the name of the variantset used (it's not currently possible to specify a different set in the individual variant request). In almost all cases (certainly all profiles known to us), the name `Variant-1` should be given here.

defaultVariantRequest ***variant-request***

This directive provides a default variant request for use when the individual element requests (see below) do not contain a variant request. Variant requests consist of a blank-separated list of variant components. A variant compont is a comma-separated, parenthesized triple of variant class, type, and value (the two former values being represented as integers). The value can currently only be entered as a string (this will change to depend on the definition of the variant in question). The special value (@) is interpreted as a null value, however.

simpleElement ***path ['variant' variant-request]***

This corresponds to a simple element request in *Espec-1*. The path consists of a sequence of tag-selectors, where each of these can consist of either:

- A simple tag, consisting of a comma-separated type-value pair in parenthesis, possibly followed by a colon (:) followed by an occurrences-specification (see below). The tag-value can be a number or a string. If the first character is an apostrophe ('), this forces the value to be interpreted as a string, even if it appears to be numerical.
- A WildThing, represented as a question mark (?), possibly followed by a colon (:) followed by an occurrences specification (see below).
- A WildPath, represented as an asterisk (*). Note that the last element of the path should not be a wildPath (wildpaths don't work in this version).

The occurrences-specification can be either the string `all`, the string `last`, or an explicit value-range. The value-range is represented as an integer (the starting point), possibly followed by a plus (+) and a second integer (the number of elements, default being one).

The variant-request has the same syntax as the defaultVariantRequest above. Note that it may sometimes be useful to give an empty variant request, simlply to disable the default for a specific set of fields (we aren't certain if this is proper *Espec-1*, but it works in this implementation).

The following is an example of an element specification belonging to the GILS profile.

```
simpleelement (1,10)
simpleelement (1,12)
simpleelement (2,1)
simpleelement (1,14)
simpleelement (4,1)
simpleelement (4,52)
```

4.6 The Schema Mapping (.map) Files

Sometimes, the client might want to receive a database record in a schema that differs from the native schema of the record. For instance, a client might only know how to process WAIS records, while the database record is represented in a more specific schema, such as GILS. In this module, a mapping of data to one of the MARC formats is also thought of as a schema mapping (mapping the elements of the record into fields consistent with the given MARC specification, prior to actually converting the data to the ISO2709). This use of the object identifier for USMARC as a schema identifier represents an overloading of the OID which might not be entirely proper. However, it represents the dual role of schema and record syntax which is assumed by the MARC family in Z39.50.

NOTE: The schema-mapping functions are so far limited to a straightforward mapping of elements. This should be extended with mechanisms for conversions of the element contents, and conditional mappings of elements based on the record contents.

These are the directives of the schema mapping file format:

targetName ***name***

A symbolic name for the target schema of the table. Useful mostly for diagnostic purposes.

targetRef ***OID-name***

An OID name for the target schema. This is used, for instance, by a server receiving a request to present a record in a different schema from the native one. The name, again, is found in the **oid** module of **YAZ**.

map ***element-name target-path***

(repeatable) Adds an element mapping rule to the table.

4.7 The MARC (ISO2709) Representation (.mar) Files

This file provides rules for representing a record in the ISO2709 format. The rules pertain mostly to the values of the constant-length header of the record.

NOTE: This will be described better.

5 The Input (Data) File Format

The retrieval module is designed to manage data derived from a variety of different input sources. When used on the client side, the source format may be GRS-1 ISO2709. On the server side, the source may be a structured ASCII file, augmented by a set of patterns that describe the structure of the document.

What we think of as the native source format - the one that is guaranteed to provide complete access to the facilities of the module, is an "SGML-like" syntax, based on an inferred DTD, which is in turn based on the profile information from the various files mentioned in this document.

Like SGML, an input record consists of tags and data. The tags are enclosed by brackets (<...>). As a general rule, each tag should be matched by a corresponding close tag, identified by the same tag name preceded by a slash (/).

6 License

Copyright © 1995-2000, Index Data.

This is the Index Data "P" license - it applies exclusively to the record management module of the YAZ system, and to this document.

Permission to use, copy, modify, distribute, and sell this software and its documentation, in whole or in part, for any purpose, is hereby granted, provided that:

1. This copyright and permission notice appear in all copies of the software and its documentation. Notices of copyright or attribution which appear at the beginning of any file must remain unchanged.

2. The names of Index Data or the individual authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED, OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL INDEX DATA BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

7 About Index Data

Index Data is a consulting and software-development enterprise that specialises in library and information management systems. Our interests and expertise span a broad range of related fields, and one of our primary, long-term objectives is the development of a powerful information management system with open network interfaces and hypermedia capabilities.

We make this software available free of charge, on a fairly unrestrictive license; as a service to the networking community, and to further the development of quality software for open network communication.

We'll be happy to answer questions about the software, and about ourselves in general.


```
Index Data Ryesgade 3 DK-2200 Copenhagen N

Phone: +45 3536 3672
Fax  : +45 3536 0449
Email: info@indexdata.dk
```