

# **ZAP User's Guide and Reference**

**Per Mørkegård Hansen**

**Adam Dickmeiss**

## **ZAP User's Guide and Reference**

by Per Mørkegård Hansen

by Adam Dickmeiss

Copyright © 1996-2006 Index Data

This document is the programmer's guide and reference to the ZAP package version 1.4.10.

The documentation can be used on its own, or as a reference when looking at the example applications provided with the package.



# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
<b>2. Installation.....</b>	<b>4</b>
<b>3. General Notes .....</b>	<b>5</b>
<b>4. Template Invocation .....</b>	<b>6</b>
Apache Module .....	6
CGI.....	6
<b>5. Input Parameters .....</b>	<b>7</b>
Parameters in the ZAP Template File .....	7
Parameters in General .....	7
Individual Input Parameters .....	7
General .....	7
target .....	7
host(x) .....	7
name(x) .....	8
action.....	8
debug .....	8
cache .....	8
piggyback.....	8
timeout .....	8
content-type.....	9
charset .....	9
Search .....	9
start.....	9
number .....	9
show_offset .....	9
show_number .....	10
syntax .....	10
schema.....	10
element.....	10
syntax(x) .....	10
piggyback(x) .....	11
termN .....	11
fieldN.....	11
map(y).....	11
opN.....	12
op.....	12
servertotal.....	12
Scan .....	13
scannumber .....	13
scanposition.....	13
Authentication .....	13
authOpen(x) .....	13
authGroupId(x) .....	13
authUserId(x) .....	13

authPassword(x) .....	14
Referring to Input Parameters .....	14
Substitution Rules .....	14
<b>6. Result Variables .....</b>	<b>16</b>
Global Parameters .....	16
\$query .....	16
\$alltargets .....	16
Per Server Parameters .....	16
\$target .....	16
\$errorcode .....	16
\$errorstring .....	16
\$addinfo .....	16
\$hits .....	16
Per Record Parameters .....	17
\$record .....	17
\$no .....	17
<b>7. Template Sections .....</b>	<b>18</b>
Each section in detail .....	18
%%def .....	20
%%override .....	21
%%begin .....	21
%%end .....	21
%%query-empty .....	21
%%query-ok .....	21
%%record xxx .....	21
%%format xxx .....	21
%%records begin .....	22
%%records end .....	22
%%server-error connection .....	22
%%server-error timeout .....	22
%%server-error init .....	22
%%server-error protocol .....	22
%%server-error N .....	22
%%server-error .....	23
%%CCL-error N .....	23
%%CCL-error .....	23
%%server-hits N .....	23
%%server-hits .....	24
%%include .....	24
%%sort-record .....	24
%%sort-format .....	24
Embedded TCL .....	24
Handling targets in TCL .....	25
Environment Variables .....	25

<b>8. CCL</b>	<b>27</b>
CCL Attributes	27
U (use attributes)	27
S (structure attributes)	27
T (truncation attributes)	27
C (completeness attributes)	28
R (relation attributes)	28
Truncation	28
Examples	28
CCL Input Parameters	29
CCLtermN	29
CCLfieldN	29
CCLopand, CCLopor, CCLopnot	29
CCLcase	29
<b>9. Record Formatting</b>	<b>30</b>
GRS-1 Record Formatting	30
Conditional Statements	31
MARC record formatting	32
XML Record Formatting	32
SUTRS records	33
<b>10. Advanced Features</b>	<b>34</b>
Nested Templates	34
Record manipulation using TCL	35
Sorting	36
<b>A. About Index Data</b>	<b>39</b>
<b>B. License</b>	<b>40</b>

# List of Tables

7-1. CCL Error Codes.....23

# Chapter 1. Introduction

This is a short introduction to the setup and use of the Index Data/USGS Z39.50 gateway/Apache module ZAP. ZAP enables access from any WWW browser to one or more Z39.50-enabled targets. It is a simple and lightweight, but highly configurable Z39.50 gateway. It is stateless (ie. it does not maintain the notion of an individual "user session" with the target) to simplify administration and setup (at the cost of complicating interoperability with certain types of targets). However, when installed as an Apache module, the gateway can be setup to cache the association with the target, thus increasing responsiveness by obviating the need to re-establish the connection and Z39.50 association with every user request. (Note: In effect, a limited number of semi-permanent Z39.50 associations are shared *between* users!)

## Example 1-1. Simple Search Form

Here is a HTML form. The form sets defines form variables, such as `target` which specifies the Z39.50 host address we want to search, as well as `syntax` which specifies what format we would like to handle and display.

The form invokes `sort.zap` on the web server. The extension `.zap` specifies a ZAP template.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>ZAP! Simple Search</title>
  </head>
  <body>
    <h1>ZAP! Simple Search</h1>
    <form action="sort.zap" method=get>
      <center>
        <input type="hidden" name="target" value="indexdata.dk:210/marc">
        <input type="hidden" name="syntax" value="usmarc">
        <input type="text" name="term1" size="50" value="utah">
        <input type="hidden" name="querytype" value="rpn">
        <input type="submit" value="submit"><br><br>
      </center>
    </form>
  </body>
</html>
```

## Example 1-2. Simple ZAP template

We now show first part of the ZAP template, `simple.zap` which is invoked by the form from the previous example.

Observe that this file is in plain text. Just like HTML. The file consists of sections. The beginning of a section is marked by `%%` followed by a section name. The beginning of a section also marks the end of the previous section (sections cannot be nested).

The sections `%%begin` and `%%end` contains HTML contains ordinary HTML. The `%%begin` section is invoked first and "outputs" the leading HTML. The `%%end` "outputs" the last parts of the HTML.

The special sections, `%%override`, `%%def` contains variable assignments. These variables have same function as FORMS variables.

```
%%override

%%def
syntax=sutrs

debug=0
cache=1
%%begin
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<!-- $Id: introduction.xml,v 1.2 2003-11-07 12:08:53 adam Exp $ -->
<html>
  <head>
    <title>ZAP! Search Result</title>
  </head>
  <body>
%%end
    </body>
  </html>
```

We now turn to the remaining parts of the template. This, again, has several sections. Each section is executed in the various stages as the target returns information, such as `%%server-hits` (result count known), `%%server-error` (server returned a diagnostic), `%%format` (record to be formatted for display).

Sections are still HTML, but note that you can refer to ZAP variables from them using `$` (dollar sign) followed by variable name.

```
%%query-empty
<h1>ZAP! Search Result</h1>
You specified an empty query. Go back and try again!<br>
%%query-ok
<h1>ZAP! Search Result</h1>
Your query was $query.<br>

%%server-error connection
<hr><h2>Couldn't connect to $target.</h2>

%%server-error protocol
<hr><h2>$target: Protocol error</h2>

%%server-error init
<hr><h2>$target: Connection rejected</h2>

%%server-error 114
<hr><dt><h2>$target: </h2><dd>You specified a field that this system does not recognise.<br>

%%server-error
```



```

<hr><dt><h2>$target: Search error</h2><dd>$errorstring${addinfo?: $addinfo} ($errorcode)<d

%%server-hits 0
<hr><h2>$target: No hits</h2>

%%server-hits 1
<hr><h2>$target: one hit</h2>

%%server-hits
<hr><h2>$target: $hits hits</h2>

%%format usmarc
245      ""
245/*    ""
245*/a    "$data<br>"

%%record sutrs
<p>
#$no:
<pre>
$record
</pre>
</p>

%%records begin
${startprevious?<a href="?start=$startprevious&${term*}&${target*}&${field*}&${element=}

<dl>
%%records end
</dl>
${startnext?<a href="?start=$startnext&${term*}&${target*}&${field*}&${element=}&${synta

```

## Chapter 2. Installation

Please refer to the file `ZAP_README` that is included with the distribution.

ZAP can be configured and installed either as an Apache module or as a conventional CGI program. The former gives superior performance and some additional flexibility in the location of the configuration files (templates). However, the latter does not require recompilation of the Apache Server, or changes to the configuration files. We whole-heartedly recommend the Apache module option.

## Chapter 3. General Notes

The user input to ZAP comes from a conventional HTML form. The form action type may be either POST or GET, although we generally prefer the latter for this type of application (particularly if it is desirable for users to bookmark the results of a query or the reference to an individual record in a database). Of course, GET URLs may also be specified outside of forms, thus enabling static links into Z39.50 databases fronted by ZAP.

The output that ZAP sends back to the browser is always based on the content of a "template file". This file, basically, contains the bits of HTML which is to be output under different circumstances (ie. at the beginning and the end of the page, when the search fails or succeeds, when records are retrieved, etc.).

Either the form-parameters to ZAP, or the template file may supply parameters to the operation (ie. query terms, database names, formats, etc.). The template file may be set up to supply defaults, or to *override* the parameters supplied by the form (e.g. to keep users from specifying random targets to access).

Once the user input and the template have been digested, ZAP executes the requisite *operation* and returns the results to the user formatted according to the template. In the current version of ZAP, the default operation is implicit "Open connection, perform search, get records back, display records". ZAP can however also do a scan. ZAP is under constant development and other types of operation may be included in the future.

# Chapter 4. Template Invocation

This chapter describes how ZAP templates are called from web pages.

## Apache Module

If ZAP is installed as an Apache module according to the documentation included with the distribution, then ZAP is invoked simply by addressing the filename of the desired template file in the normal way: e.g. in a form:

```
<FORM METHOD=GET ACTION="http://myserver.com/demo/mysearch.zap">
```

Or in a conventional HTML link:

```
<A HREF="http:mysearch.zap?id=7432"> MY RECORD! </A>
```

In the Apache setup file, you decide what suffix you want to use for the files. We use ".zap" in all of our examples, but this is not a law: You may already have another application which uses the .zap suffix, and in that case you may wish to select a different one for ZAP.

## CGI

The CGI installation requires that all of the ZAP templates are located in a specific directory, determined at compile time. The name of the template file is specified as part of the path name in the invocation. For instance in a form:

```
<FORM METHOD=GET  
ACTION"http://myserver.com/cgi-bin/zap/mysearch.zap">
```

Or in a conventional HTML link:

```
<A HREF="http://cgi-bin/zap/mysearch.zap?id=7432"> MY RECORD! </A>
```

If ZAP had been configured to look for templates in the "demo" directory used in the test example above, then these two examples would have been exactly equivalent in terms of functionality.

# Chapter 5. Input Parameters

These are the parameters, which govern the actions of ZAP. With the exception of the template name, they may all be supplied either from the input form, in a separate URL, or in the template file.

## Parameters in the ZAP Template File

In the template file, parameters can be supplied in the "def" or "override" sections of the file (see Chapter 7). A typical application of ZAP may use a combination of parameters supplied from the form, and parameters given within the template file.

## Parameters in General

All other parameters than the template name are supplied in the normal form way: In a POST form, or following the filename (after a ?), and expressed as "attribute=value" statements separated by "&": Exactly what you get from a GET form, but also easy enough to generate by hand if you wish. For example:

```
<A HREF="http:mysearch.zap?id?7432&syntax=usmarc">
```

## Individual Input Parameters

### General

The input parameters that apply generally to the ZAP template are presented below.

### target

The target to connect to. If you make repeated declarations of this, ZAP will submit the query to all of the targets. The value is either hostname[:port][/database+database+...], or a private, logical name. If you use a logical name, you must use host(x) ( see below) to define the actual host address.

### host(x)

The hostname, port, and database corresponding to the private logical name x. Example:

```
host (test) =www.indexdata.dk:210/gils
```

The target is then set by:

```
target=test
```

### **name(x)**

A display name corresponding to the private logical name x (useful for display to the user when multiple targets are searched concurrently. Example:

```
name(test)=Index Datas test server
```

### **action**

Can be set to `search`, `scan` or `init` where `search` is default. When set to `init` only an `init` request is sent to the server. When set to `search` the normal "Open connection, perform search, get records back, display records" operation is performed. When set to `scan` a `Scan` request is sent.

### **debug**

If this is set to 1, the gateway will dump the parameters on the HTML output. This option is useful for debugging your forms. Default is 0 (no debug information is displayed).

### **cache**

If this is set to 1, the connections to the server will be cached by each process (works only in the Apache module version). We recommend that you not enable this until you are happy with the way your setup works in general.

### **piggyback**

Turns piggybacking off, since default is on. Piggybacking is the concept of having records send back along with the search response in order to save a network round trip. However some servers don't like this and may cause problems if it is turned on. Example:

```
piggyback=0
```

## **timeout**

The time in seconds with no network activity before the search is abandoned for a given server. Default is 20 sec.

## **content-type**

The content type of the document. If this is not set, `text/html` is sent. Example:

```
content-type=text/html
```

## **charset**

The character set of the page. Example:

```
charset=UTF-8
```

# **Search**

The input parameters that apply specifically to the search are presented below.

## **start**

The offset of first record to be retrieved. Offsets are numbered 1, 2, 3.. If start is unset, the value 1 is assumed.

The records retrieved from a target is thus

```
start, start+1, ... start+number-1
```

These are the records shown (formatted, etc) unless `show_offset` and `show_number` is set.

## **number**

The number of records to retrieve. If the number of hits is less than `start+number`, ZAP will retrieve fewer records. If number is omitted, 10 is assumed (10 records retrieved).

**show\_offset**

The offset of first record to show - within records retrieved from the target. Setting this allows you to view a subset of the records retrieved from the target. Useful if ZAP is configured to sort records. First record shown is 0. next is 1, etc.

And the records shown is:

```
start+show_offset, start+show_offset+1, ... start+show_number-1.
```

If start+show\_offset is greater than start+number-1, then nothing is shown. If start+show\_number is greater than start+number, then the last record shown is start+number-1.

**show\_number**

Number of records to show in ZAP. See the Section called *show\_offset*.

**syntax**

The preferred record syntax. If the target supports the syntax specified, records should be returned in this format. Currently supported values are: Ausmarc, Canmarc, Carmarc, CCF, Danmarc, Finmarc, GRS1, HTML, Ibermarc, Interarc, ISDSmarc, JPmarc, Librisarc, MAB, Malmarc, Normarc, Picamar, RUSmarc, SBN, SIGLEmarc, SOIF, SUTRS, SWEmarc, UKmarc, Unimarc, USmarc, XML

Apart from these OID can be used. See OID at the Z39.50 Maintenance Agency (<http://lcweb.loc.gov/z3950/agency/defns/oids.html>) for more information.

If syntax is not set, ZAP does not specify preferred record syntax, so the target may decide type of record to be returned (if any).

**schema**

The schema to ask for. Both strings and OID can be used. See OID list at the Z39.50 Maintenance Agency (<http://lcweb.loc.gov/z3950/agency/defns/oids.html>) for more information. Example:

```
schema=holdings
```

**element**

The element set name to ask for. Most targets supports element set B and F which stands for Brief and Full respectively.



**syntax(x)**

A record syntax corresponding to the private logical server name x. Example:

```
syntax(test)=grs1
```

If a syntax is not specified explicitly for a given target, the syntax given by the unqualified parameter is used.

**piggyback(x)**

This parameter is used to turn piggyback off for a given target. Default is on (1). Example:

```
piggyback(test)=0
```

**termN**

A search term. N should be a low integer, ie. term1. You may supply as many entries as you wish, but there must be no "gaps" - ie. you can't just supply a term1 and a term3. A term may have an empty string value, however, in which case it is ignored.

**fieldN**

The search qualifications which should be applied to termN. The syntax is "@attr [attribute-set] type=value", where attribute-set is optional (default is Bib1), and type is the numeric attribute type, and value is the attribute value. The @attr expression may be repeated if multiple attributes are to be used. The contents of the variable may also be a symbolic name, defined by the map(y) variable below.

Please see Bib-1 attribute set page at the Z39.50 Maintenance Agency (<http://lcweb.loc.gov/z3950/agency/defs/bib1.html>) for further details on the Bib1 attribute set.

**map(y)**

This introduces a mapping of a symbolic field name to a specific attribute combination. It can be used to simplify the parameter passing from the form, and also to aid in a more intuitive display of the query, if you use the query variable to give feedback to the user. Example:

```
map(Title)=@attr 1=4
```

If you put this in your def or override section of the ZAP template (see below), then you could use "Title" as a value for fieldN.

In order to map a specific attribute combination to a specific target the syntax `map(y,x)` is used. Example:

```
map(Title)=@attr 1=4
map(Title,test)=@attr 1=1
```

The above code will, when searching in the title field, search all targets with the attribute `1=4` except the "test" target that will be searched with `1=1`.

## opN

The operator which connects termN and the first subsequent, non-empty term. The value should be one of "and", "or", or "not". If there is a termN and a termN+1, but there is no opN, then "and" is assumed.

## op

Specifies which operators should be used if a term is left empty. Can be set to either left or right. Default is left. In the following example we have a query of the form "term1 op1 term2 op2 term3" where "term2" is an empty string. In that case the query is interpreted as:

```
op=left:      term1 op1 term3
op=right:     term1 op2 term3
```

## servertotal

This setting controls when/how ZAP collects results from multiple targets. These are the possible values and their meaning:

0 (or omitted)

In this configuration, ZAP invokes the sections `%%server-hits`, `%%records begin`, and all record display sections as soon as each target has returned all records. This is the fastest way to produce results to the user, but it does not allow to display a merged result or even a total hit count (for all targets).

This mode is the default

1

Here, ZAP does not invoke sections `%%server-hits`, `%%records begin`, .. etc until all targets has completed operation. In this mode, a ZAP template may display total hits before each record is displayed.

s

In this mode, ZAP, will collect all results, then invoke sections `%%server-hits`, `%%sort-record`, `%%sort-format` for all targets. This mode makes ZAP to sort all records retrieved from the target. When sorting is complete `%%records` begin and other formatting sections are invoked as usual - but in sorted order.

## Scan

The input parameters that apply only to scan are presented below.

### scannumber

The number of records to ask for when a scan is performed. (See the Section called *action*). Default is 10.

### scanposition

The position of the closest match in the scan result set. If `scanposition=1` the closest match will be in the top of the list and if `scannumber=10` and `scanposition=10` it will be in the bottom of the list. Default is 5.

## Authentication

Depending on the target you should provide either `authOpen` (v2/v3 targets) or one or more of the three others (v3 targets only).

### authOpen(x)

Open-authentication. This authentication scheme is a Z39.50 v2/v3 feature. Usually the string has the form `user/password`, e.g. `"peter/secret"`. Example:

```
authOpen(localhost:9999/Default)=myname/mypassword
```

### authGroupId(x)

Group ID for authentication. This is a Z39.50v3 feature.

**authUserId(x)**

User name authentication. This is a Z39.50v3 feature. Example:

```
authUserId(localhost:9999/Default)=myname
```

**authPassword(x)**

Password for authentication. This is a Z39.50v3 feature. Example:

```
authPassword(localhost:9999/Default)=mypassword
```

## Referring to Input Parameters

The ZAP input parameters can be referred to by the notation "\$parametername". The example below shows a link to a search with three search terms:

```
<A HREF="http:mysearch.zap?term1=$term1&term2=$term2&term3=$term3"> MY RECORD! </A>
```

There is, however, a shorthand notation for referring to all the "terms", "fields", "operators" and "targets" where you instead of `term1=$term1&term2=...` simply writes `${ parametername*=}`. The example above can then be written as

```
<A HREF="http:mysearch.zap?${term*=}"> MY RECORD! </A>
```

## Substitution Rules

When you declare parameters in the template file, you can use \$-substitution to construct values from the contents of other parameters. For instance, if you wish to create a ZAP template that supports a URL such as: "http:getrecord.zap?id=xxxxxx", you might use something like this:

```
term1=$id
field1=@attr 1=12
```

where the value of the parameter `id` is used as a search term. Similarly, if you wish to supply the name of the database independently of the hostname, like:

```
http:search.zap?id=xxxx&hostname=myhost&db=mydatabase
```

you could do something like:

```
target=$myhost/$mydatabase
```

in the `def` or `override` sections of your template.

# Chapter 6. Result Variables

These variables can be used in the HTML sections to provide dynamic information to the user - ie. information that depends of the results of the operations under execution. Some parameters are global - ie. specific to the operation currently under execution, but not specific to any given operation. Others are specific to the target currently under processing, while yet others are "local" to the current record.

## Global Parameters

### **\$query**

The query term(s). If operators have been used, they are also included.

### **\$alltargets**

This variable contains a preformatted "URL string" with all the targets that was searched. As an example, if you have searched two targets `z3950.loc.gov:7090/voyager` and `z3950.bibsys.no:2100/BIBSYS`, `alltargets` will contain the string `&target=z3950.loc.gov:7090/voyager&target= z3950.bibsys.no:2100/BIBSYS`.

## Per Server Parameters

### **\$target**

The address for the current target.

### **\$errorcode**

Bib-1 diagnostic code.

### **\$errorstring**

An English-language string representation of the errorcode.

### **\$addinfo**

The additional-info string of the diagnostic, if any.

## **\$hits**

The number of hits for the current server.

## **Per Record Parameters**

### **\$record**

Expands to the full record as a string. Only works for SUTRS.

### **\$no**

The current record number (offset) in the result set.

# Chapter 7. Template Sections

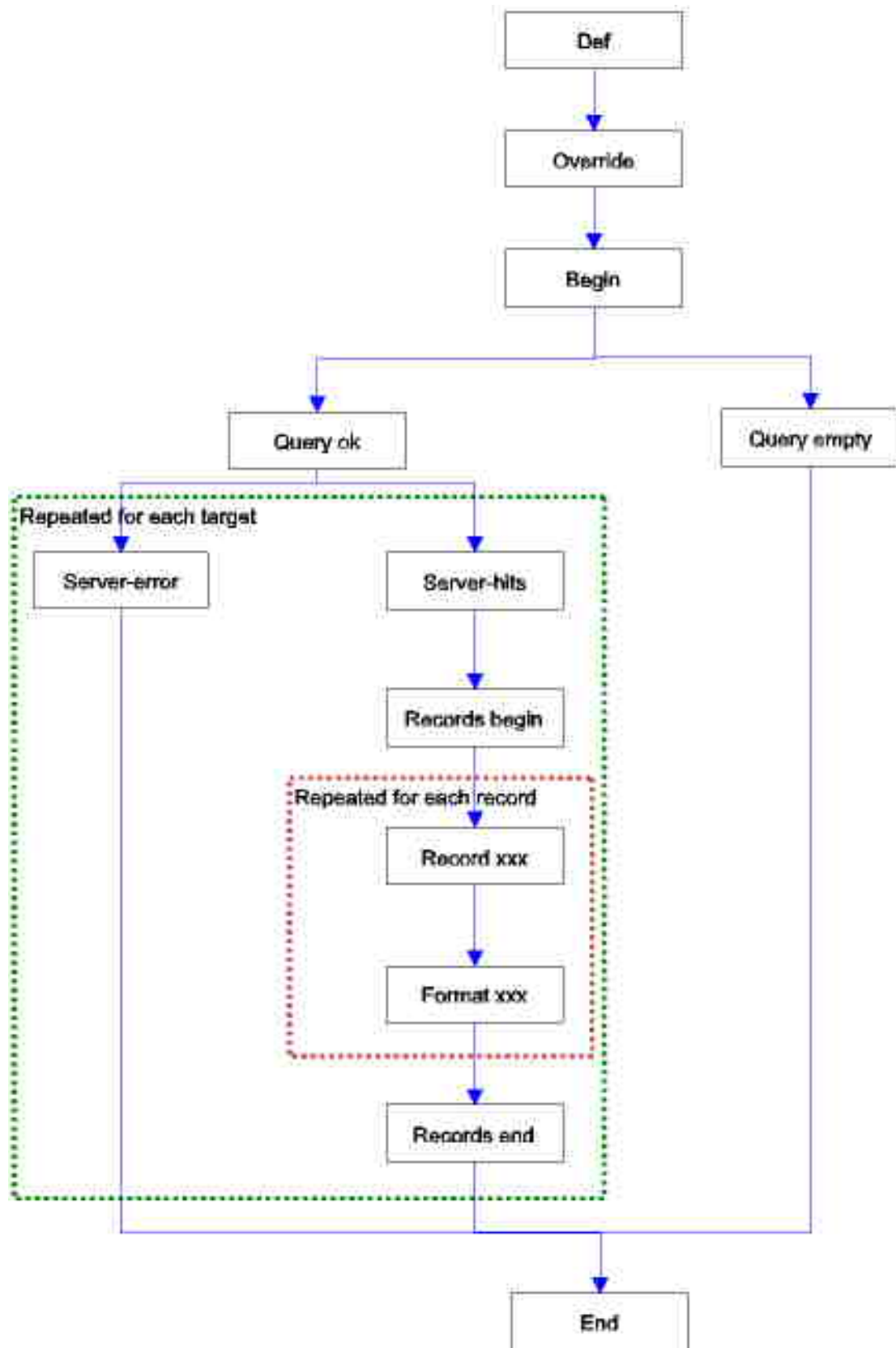
The ZAP template consists of a number of sections. Each section, with a couple of exceptions, should contain the HTML fragment, which is to be output when a given situation arises. However, the `def` and `override` sections contain input parameters which are not directly output, and the `record` section sometimes uses special syntax to govern the formatting of a given record structure (e.g. GRS-1). Each section is headed by a `%%` in the leftmost column, followed by the name of the section (e.g., `%%begin`).

## Each section in detail

The sections are executed in the order depicted in figure 1. First the `%%def`, `%%override` and `%%begin` sections are interpreted. If the query is valid the search is performed and this will result in either the `%%server-error` or the `%%server-hits` section is executed. This procedure is repeated for each target. If the search results in any hits the `%%records begin` is executed and then for each record the `%%record xxx` and `%%format xxx` (where `xxx` is the record format the records are returned from the server in) is executed. Finally the `%%end` section is executed.







## **%%def**

In this section all the initial declarations can be made such as what target to search, how many records to display and so forth. Parameters being transferred from a form will override the same parameters in this section.

## **%%override**

Same as the `%%def` section but parameters declared in this section will override the parameters being transferred from a form.

## **%%begin**

This section is sent to the user before anything else takes place. It typically contains the HTML header information (title, etc.). If the section is omitted, nothing is sent.

## **%%end**

This section contains the last HTML to be Figure 1 Flow diagram of the sections in ZAP sent to the user. If the section is omitted, nothing is sent.

## **%%query-empty**

This section is sent if the user supplied an empty query.

## **%%query-ok**

This section is sent if the query is syntactically correct, but before it is sent to the target(s). One thing it might be used for is to display the interpreted query to the user. The query is available in a string form in the variable `$query`.

## **%%record xxx**

This section is invoked for each record with a given record format where `xxx` is the record format. This is done before the actual record formatting takes place in the `%%format` section. Please refer to the Section called *syntax* in Chapter 5 for a list of valid record formats.

## **%%format xxx**

This section contains the formatting expressions for a given record format, where xxx is the record format. It governs the mapping of an individual record to HTML. Please refer to the Section called *syntax* in Chapter 5 for a list of valid record formats. See Chapter 9 on how to format records.

## **%%records begin**

This section is sent before the first record for a given target. It might be used, for instance, to begin an HTML table or bullet-list.

## **%%records end**

This section is sent after the last record for a given target.

## **%%server-error connection**

This section is sent if ZAP is unable to establish a physical (network) connection with the given target (\$host), or if the network connection drops suddenly during the processing of the query (the latter condition typically indicates that the target has crashed).

## **%%server-error timeout**

This section is sent if ZAP times out during the connection phase. The time out time is set with the timeout variable (see the Section called *timeout* in Chapter 5).

## **%%server-error init**

This section is sent if the attempt to initialize the Z39.50 association with the target fails.

## **%%server-error protocol**

This section is sent if the protocol decoder layer fails on an incoming protocol package. It indicates a breach of protocol by the target, or a bug in ZAP's protocol layer (knock on wood).

## **%%server-error N**

N is a number corresponding to a Bib-1 diagnostic code. Use this to generate a user-friendly message when a recognized error is returned.

Please refer to <http://lcweb.loc.gov/z3950/agency/defns/bib1diag.html> for further information on Bib-1 diagnostic codes.

## %%server-error

This is a catch-all - if no specific section is found to match the bib-1 diagnostic code, this section is sent to the user. Use \$errorcode, \$errorstring, \$addinfo, to get the diagnostic code, a human-readable string (often in English), and the diagnostic additional-info field (if it is supplied), respectively.

## %%CCL-error N

N is the number corresponding to the errorcode returned from YAZ' CCL parser. Use this section to generate a user-friendly message when a errorcode is returned.

**Table 7-1. CCL Error Codes**

Code	Description
0	OK (no error)
1	Search word expected
2	' )' expected
3	Set name expected
4	Operator expected
5	Unbalanced ' )'
6	Unknown qualifier
7	Qualifiers applied twice
8	' =' expected
9	Bad relation
10	Left truncation not supported
11	Both left - and right truncation not supported
12	Right truncation not supported

## %%CCL-error

This is a catch-all - if no specific section is found to match the YAZ CCL parser errorcode, this section is sent to the user. Use \$errorcode, \$errorstring, to get the diagnostic code and a human-readable string (the Section called *%%CCL-error N*), respectively.

## %%server-hits N

N is the number of hits in a resultset and this section is invoked when a resultset contains the specified number (N) of hits.

### Example 7-1. Checking for specific hit count

```
%%server-hits 0
Sorry, no hits.
```

```
%%server-hits 1
There was one hit.
```

## %%server-hits

This is a catch-all - if no specific section is found to match the number of hits, this section is invoked.

### Example 7-2. Displaying any hit count

```
%%server-hits
There was $hits hits in $target.
```

## %%include

%%includes is actually not a section but a call to include a file on that particular place.

### Example 7-3. include

Lets say that you have put all the HTML code for the beginning of the page in the file header.html then you can include it the e.g. the begin section as follows

```
%%begin
%%include header.html
```

## %%sort-record

This section should return the sort key for a record. It is only invoked if sorting is enabled. See the Section called *servertotal* in Chapter 5 This section is similar to %%record but should *not* make output. Upon completion of this section, ZAP inspects the TCL variable `sort` and assumes it is the sorting key.

## %%sort-format

This section should return the sort key for a record. It is only invoked if sorting is enabled. See the Section called *servertotal* in Chapter 5 This section similar to %%format but should *not* make output. Upon completion of this section, ZAP inspects the TCL variable `sort` and assumes it is the sorting key.

## Embedded TCL

If you enabled TCL support when you compiled ZAP, you can embed bits of TCL code anywhere in your HTML sections. The TCL sections should begin with "%{" and end with "%}". The ZAP variables are available in the TCL interpreter, and you can generate output to the user from within the TCL script using the special command "html", which simply sends each of its arguments on to the user.

### Example 7-4. Tcl embedding

TCL variables are however not directly accessible from ZAP. If you have created a variable in TCL and you want to print it out in one of your HTML sections you will have to tell ZAP explicitly that this is a TCL variable. Example:

```
%{set myvar 42%}
The value of myvar is: %{html $myvar%}
```

In order to work around the inconvenience of TCL variables not being accessible from ZAP a special set command called `setz` is used. Example:

```
%{setz myvar 42%}
The value of myvar is: $myvar
```

## Handling targets in TCL

The list of targets is a special case. When the target list is received from an HTML form, it is done by repeating instances of the property "target" - in effect the target variable behaves as if it had several different values (corresponding to each target being searched), and there is no direct way of handling this in TCL.

To remedy this situation, the command `addz` has been introduced in order to make it possible to add elements to the list of targets being searched.

As an example, lets say that you, along with one or more targets you have chosen on your search page, always want to search Library of Congress. Setting `target=z3950.loc.gov:7090/voyager` would only overwrite the selection you have made on the search page, but the code:

```
%{addz z3950.loc.gov:7090/voyager%}
```

in the `%%def` or `%%overwrite` section, would do the trick.

## Environment Variables

Only a few of the web servers environment variables are available in ZAP and only as a TCL array called `env`. The environment variables available are `QUERY_STRING`, `REMOTE_USER` and `REMOTE_IP`.

In order to get the users user name in a page that is password protected you would write something like this:

```
%{setz username $env(REMOTE_USER)%}  
The username is: $username
```



# Chapter 8. CCL

ZAP is also capable of performing CCL (Common Command Language) searching. CCL allows you to shorthand references to fieldnames such as e.g. "ti" for title and "au" for author. The shorthand names are called CCL fields. A CCL search could then look something like:

```
ti=computer and au=knuth
```

To set this up you need first to define the CCL fields in a file called `cclfields.zap`. The syntax is

Fieldname u=use attribute s=structure attribute t=truncation attribute c=completeness attribute

For more information on valid attributes, please see the section below.

Specific CCL fields for a specific target can be set up by creating a file with the required fields and a filename that follows the filename convention `cclfields.targetname.zap` where `targetname` refers to the target's logical name (please see the Section called *name(x)* in Chapter 5)

## CCL Attributes

### U (use attributes)

Bib-1 is the default Use attribute set. Please refer to <http://lcweb.loc.gov/z3950/agency/defns/bib1.html> for valid values of the Bib-1 use attributes. Other attribute sets can be specified.

### S (structure attributes)

The structure is one of the values from <http://lcweb.loc.gov/z3950/agency/defns/bib1.html> - 4 or it may be one of the special values: values below:

pw

This sets ZAP up to automatically set structure to either word or phrase. If the search term is only a single word like "Computer" structure is set to word and if it consists of more several words like "computer programming" structure is set to phrase.

al

This makes ZAP add the logical operator "and" between the words in a query. E.g. the query "computer programming" would be interpreted as "computer" and "programming".

ol

Same as al but uses the logic operator "or".

## T (truncation attributes)

Please refer to <http://lcweb.loc.gov/z3950/agency/defns/bib1.html> - 5  
(<http://lcweb.loc.gov/z3950/agency/defns/bib1.html>) for valid values of the truncation attributes.

In addition to the attributes listed in the above document l (left) and r (right) can be used. This enables Zap to set the truncation according to the users specification. If structure is set to e.g. t=r and the user enters "computer?" truncation is set to right truncation. If the user only enters "computer" no truncation is set.

## C (completeness attributes)

Please refer to <http://lcweb.loc.gov/z3950/agency/defns/bib1.html> - 6  
(<http://lcweb.loc.gov/z3950/agency/defns/bib1.html>) for valid values of the completeness attributes.

## R (relation attributes)

Please refer to <http://lcweb.loc.gov/z3950/agency/defns/bib1.html> - 2  
(<http://lcweb.loc.gov/z3950/agency/defns/bib1.html>) for valid values of the relation attributes.

## Truncation

What characters a user can use to indicate that a word should be truncated can be set with the @truncation declaration in the cclfields.zap file. If you e.g. would like to use both "\*" and "?" for truncation it would look like this:

```
@truncation ? *
```

## Examples

### Example 8-1. CCL field definition

An example of a CCL field definition can be seen below:

```
ti    u=4    s=pw    t=l,r    c=2
```

where the fieldname has been set to "ti", the Bib-1 use attribute has been set to 4 (title), structure has been set to pw so that ZAP automatically sets the attribute to either "phrase" or "word". Both left and right truncation has been enabled and completeness has been set to 2 (complete subfield).

To specify e.g. a GILS use attribute, the following syntax is used:

```
ti    gils,u=2021    s=104
```

## CCL Input Parameters

The input parameters that apply specifically to the CCL searching are presented below.

### CCLtermN

The CCLterm parameter is similar to the normal term parameter (the Section called *termN* in Chapter 5) and can be mixed in with the normal term parameters in the sense that you can have a term sequence term1, term2, cclterm3, term4, ...

Note that also here no missing terms are allowed. All terms must be set to at least an empty string.

### CCLfieldN

The CCLfield parameter is similar to the normal field parameter (see the Section called *fieldN* in Chapter 5) and can be mixed in with the normal field parameters in the sense that you can have a field sequence field1, field2, cclfield3, field4, ...

Note that also here no missing fields are allowed. All fields must be set to at least an empty string.

### CCLOpand, CCLOpor, CCLOpnot

These parameters enable you to redefine the boolean operators in your CCL search. The default values are "and", "or" and "not". If say you want to use a "+" instead of "and" you will set

```
CCLOpand=+
```

### CCLcase

Governs if the CCL fieldnames should be case sensitive or not. Default is 1 (case sensitive). To make the fieldnames case insensitive, CCLcase should be set to 0, as in:

```
CCLcase=0
```

# Chapter 9. Record Formatting

The record formatting section is consulted whenever a record is to be displayed. The supported record formats are listed in the Section called *syntax* in Chapter 5 and you can have many record formatting sections for different record formats in the same .zap file.

In the following subsection we will go through the formatting different main record formats but we will start off with GRS-1 record formatting and use them to explain a number of general concepts about formatting records in ZAP. Therefore, even if you are not interested in GRS-1 records, be sure to read this section carefully.

## GRS-1 Record Formatting

The record formatting section (`%%format grs1`) consists of a number of lines. In the beginning of each line is a GRS-1 tagpath, ie. a list of (x,y) vectors separated by "/". Each vector corresponds to a tag type and a tag value (string or numeric). The character "\*" may be used as a wildcard matching all tag values for a given type. The tagpath is followed by whitespace, and then by a string (surrounded by double quotes), which is to be sent to the user for each occurrence of the GRS-1 element corresponding to that tag-path. The order of the display of elements depends on the order of the lines in the GRS-1 section - on the order of the elements in the GRS-1 file.

A GRS-1 tag can also be just a string tag. In this case the tagpath will consist of the stringtags separated by "/" or a combination of stringtags and normal (x,y) GRS-1 tags.

The code after the tagpath may include references to the \$data variable, which expands to the CONTENTS of the data element, if it is unstructured. The code can't include any newlines.

The basic string value mentioned above is sent only for primitive data values (ie. unstructured GRS-1 elements). If you wish to send output for a structured element, you should prepend the string with a "b:" or "e:". The former is emitted before the contents of the structured element are consulted - the latter after.

In some circumstances it is desirable to send output only the first time an element is met, e.g. you want to write "Keywords" only the first time a keyword field is found. This can be done by appending a ":1" to the end of the tagpath.

### Example 9-1. Record Formatting with tag paths

The simplest case is an unstructured element, title, with the tag (2,1):

```
(2,1)      "The title is: $data"
```

A more complex element might be, say, a structured author element (2,2) with sub-elements name (2,7) and e-mail-address (2,13):

```
(2,2)      b:"Author: "  
(2,2)/(2,7) "$data"  
(2,2)/(2,13) " (email: $data) "
```

Given the values "Sebastian Hammer" and "quinn@indexdata.dk" for the name and e-mail-address sub-element, respectively, the following display would result:

Author: Sebastian Hammer (email: quinn@indexdata.dk)

If the field (2,2)/(2,13) exists more than one time in the record, ie. a person has more than one e-mail address, the last line in the example above could be changed to the following, in order to only write "(email:" once:

```
(2,2)/(2,13):1      " (email: "
(2,2)/(2,13)        " $data "
(2,2)/(2,13):1      " ) "
```

Finally, let us assume a Title element (2,1) which is sometimes unstructured, and sometimes structured into the well-known tag (1,19), and a locally string-tagged sub-elements (e.g. (3,"Subtitle")).

```
(2,1)               "Title: $data" b:"Title: "
(2,1)/(1,19)        "$data"<BR>
(2,1)/(3,*)         " ($tagvalue: $data) "
```

This example might lead to an output like this:

```
Title: Macbeth (Subtitle: An Electronic Version)
```

There is a more general method than \$data available to access individual elements of the record. The format

```
"${ " [ "/" ] <tag> { "/" <tag> } } "
```

Will map to the contents of the element matched by the tag-path supplied. The format of "tag" is the usual (x,y) vector. If there is an initial "/" in the tagpath, the search is "rooted", ie. the entire tag-path must match from the root of the record. If there is no "/" in the beginning of the path, the search is limited to the "current" sub-tree of the record.

This:

```
${ / (2,1) }
```

looks for the TagSetG element "Title" at the root of the record;

```
${ (2,1) }
```

looks for the same element in the *current* sub-tree!.

This:

```
${ / (4,70) / (2,1) }
```

looks for the given element, starting from the root of the record.

## Conditional Statements

It is possible to embed conditional statements in the display, such that a given bit of content is displayed only if a condition (usually the existence of a variable or field) is true. The syntax is

```
"${" <expression> "?" <text1> ":" <text2> "}"
```

The expression may be a variable (as described above) or an element reference. If the variable/element exists, then text1 is displayed, otherwise text2 is displayed. Either text may be empty, and both may contain other, embedded element/variable references or conditional expressions. No language would be complete without a mechanism for generating recursive and completely incomprehensible statements - to baffle the novice and boost the ego of the journeyman.

## MARC record formatting

A MARC record does not contain tag types and can therefore in this respect be compared with GRS-1 records with string tags. Therefore the tagpath only consists of a list of tag names separated by "/". In the MARC terminology this is usually referred to as field name and subfield name.

Apart from this the record formatting is exactly like GRS-1.

Please refer to the document <http://lcweb.loc.gov/z3950/agency/defns/oids.html#5> (<http://lcweb.loc.gov/z3950/agency/defns/oids.html>) for information on the different MARC formats.

### Example 9-2. MARC Formatting

Let's say we want to display the title and author field in an USMARC record. The title is stored in field 245, subfield a and the author name is stored in field 100 where the subfields for surname is a and for first name it is h. This would look something like this in ZAP:

```
%%format usmarc
245
245/*
245/*/a "<B>Title:</B> $data"
100
100/*
100/*/a "<br><B>Author:</B> $data"
100/*/h ", $data"
```

Given the values "The Ugly Ducklin", "Hans Christian " and "Andersen" for title, first name and surname, respectively, the output would be:

```
<B>Title:</B> The Ugly Ducklin
<B>Author:</B>Andersen, Hans Christian
```

## XML Record Formatting

XML records are also handled like GRS-1 records with string tags.

### Example 9-3. XML Formatting

Let's say we have a record with the tags "author" and "title". This would look something like this in ZAP:

```
%%format xml
author "Author: $data"
title "<br>Title: $data"
```

Given the same data values as in the MARC example above (see Example 9-2) the output would be:

```
Author: Hans Christian Andersen
Title: The Ugly Ducklin
```

## SUTRS records

Since SUTRS is simple, unstructured text there is no tagpath. The whole record is simply put in the variable "record" and can be printed out where appropriate.

### Example 9-4. SUTRS Formatting

The display of a SUTRS record could look something like this in ZAP:

```
%%record sutrs
<p>This is record number $no:<br> <pre> $record</pre>
```

# Chapter 10. Advanced Features

## Nested Templates

In ZAP it is possible to, from one ZAP template, call another ZAP template and display the result in the first template. This can be an advantage in some circumstances where a search is required that is impossible with a standard ZAP template.

This is done with the TCL function **callZap**. The function takes two arguments; first the path to the ZAP template to call and second the variables to send to the ZAP template being called. The variables should be formatted the way parameters are formatted in a URL ie. name=value and separated with "&". A "?" in the beginning of the variable string is however not necessary.

All the output from the ZAP template being called will then be displayed where the **callZap** function is being called.

### Example 10-1. Calling Templates

Let's say that we search in a database and for each record we get back we also want to check if a record with the same record id exists in another database. The search in the other database is done with the ZAP template `check-id.zap` and the only parameters that is sent to the template is the Id number to search for, the type of search to perform and the database to search in. If we say that the records type is GRS-1 and the tag for the record id is a string tag called `record-id` it would look something like this in ZAP:

```
(3,record-id) "Record Id: $data %{callZap check-id.zap field1=id&entry1=$data&target=
```

The ZAP template `check-id.zap` would look something like this:

```
%%def
map(id)=@attr 1=1032
syntax=grsl

%%server-error init
$name: Connection rejected

%%server-error
Search error in $name.
$errorstring${addinfo?: $addinfo} ($errorcode)<br>

%%server-hits 0
This record does not exist in $target.

%%server-hits 1
This record does exist in $target.

%%server-hits
This record does exist in $target $hits times.
```

If **callZap** is being called in a variable declaration, only one special variable is passed from the called ZAP template, namely the variable `result`;



**Example 10-2. Getting results from a Template**

We use the same example as above. Then it would look something like this in ZAP:

```
{setz myvar [callZap check-id.zap field1=id&entry1=$data&target=$target]%}
(3,record-id) "Record Id: $data $myvar"
```

The ZAP template `check-id.zap` would look something like this:

```
%%def
map(id)=@attr 1=1032
syntax=grsl

%%server-error init
set result "$name: Connection rejected"

%%server-error
set result "Search error in $name.
$errorstring${addinfo?: $addinfo} ($errorcode)<br>"

%%server-hits 0
set result "This record does not exist in $target."

%%server-hits 1
set result "This record does exist in $target."

%%server-hits
set result "This record does exist in $target $hits times."
```

## Record manipulation using TCL

The record sets are not directly accessible from TCL and in order to do record manipulation the functions **getMarc** (for MARC records) and **getGrs** (for GRS-1 and XML). The functions take a string of arguments as input where the elements are the tagpath to the record element you want to get but with spaces instead of "/".

For the **getMarc** function you also have to specify the type of the output you want as the first element in the list of arguments for the function. The valid values are "list" and "field". The field type will give you only the data in the field you requested. List will return a list with the tagpath and the data

It is only meaningful to use **getMarc** and **getGrs** in the record and format sections where there are records to manipulate.

**Example 10-3. Record manipulation using TCL**

The following code

```
Title: %{html [getMarc field 245 * a]%}
```

would with the data "The Art of Computer Programming" in MARC field 245, subfield a, produce the following output:

```
Title: The Art of Computer Programming
```

## Sorting

**Example 10-4. Sorting**

We'll present a ZAP template that retrieves at most 100 USMARC records from the target and show 10 records at a time. We use title (field 245, subfield a) as sorting criteria.

The template could be invoked the search form presented in Example 1-1.

```
%%override
servertotal=s
number=100
show_number=10

%%def
syntax=sutrs

%%begin
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>ZAP! Search Result</title>
  </head>
  <body>
%%end
    </body>
</html>

%%query-empty
<h1>ZAP! Search Result</h1>
You specified an empty query. Go back and try again!<br>
%%query-ok
<h1>ZAP! Search Result</h1>
Your query was $query.<br>

%%server-error connection
<hr><h2>Couldn't connect to $target.</h2>
```

```

%%server-error protocol
<hr><h2>$target: Protocol error</h2>

%%server-error init
<hr><h2>$target: Connection rejected</h2>

%%server-error 114
<hr><dt><h2>$target: </h2><dd>You specified a field that this system does not recognise.<br>

%%server-error
<hr><dt><h2>$target: Search error</h2><dd>$errorstring${addinfo?: $addinfo} ($errorcode)<d

%%server-hits 0
<hr><h2>$target: No hits</h2>

%%server-hits 1
<hr><h2>$target: one hit</h2>

%%server-hits
<hr><h2>$target: $hits hits</h2>

%%sort-format usmarc
245      ""
245/*     ""
245*/a    "%{ set sort $data %}"

%%format usmarc
245      ""
245/*     ""
245*/a    "$data<br>"

%%record sutrs
<p>
# $no:
<pre>
$record
</pre>
</p>

%%records begin
<dl>
%%records end
</dl>
%{
    if {[info exists show_offset]} {
        set show_offset 0
    }
    html "show_offset = $show_offset<br/>"
    if {$show_offset >= $show_number} {
        html "setttting prev offset<br/>"
        setz prev_show_offset [expr $show_offset - $show_number]
    } else {
        setz prev_show_offset {}
    }
}

```

```

    }
    if {$show_offset < [expr $hits - $show_number]} {
        setz next_show_offset [expr $show_offset + $show_number]
html "setting next_offset<br/>";
    } else {
        setz next_show_offset {}
    }
}
%}
${prev_show_offset?<a href="?show_offset=${prev_show_offset}&${term*}&${target*}&${field*}
${next_show_offset?<a href="?show_offset=${next_show_offset}&${term*}&${target*}&${field*}

```

The TCL code in the `%%records end` section calculates the offsets for previous page and next page. If `prev_show_offset` is a non-empty string, the previous page button is displayed. And the same mechanism is used for `next_show_offset`.

# Appendix A. About Index Data

Index Data is a consulting and software-development enterprise that specializes 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 hyper-media 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 ApS  
Købmagergade 43 2.  
1150 Copenhagen K  
Denmark  
Phone +45 3341 0100  
Fax +45 3341 0101  
Email <info@indexdata.dk>

# Appendix B. License

Copyright © 1995-2006 Index Data.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Index Data nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS AND CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.