Hyper-Text Query Language (HTQL)

Liangyou Chen, Ph.D.

1. What is HTQL

Hyper-Text Query Language (HTQL) is a simple language for querying and transformation of HTML Web documents. HTQL can be used to:

1) Extract HTML content from Web pages
2) Construct table structures from a Web page
3) Programmingly modify HTML pages

HTQL can be applied to both XML and plain text documents.

HTQL is related to XPATH (http://www.w3.org/TR/xpath/), but more suitable for HTML data extraction.

2. Simple Examples

To retrieve all <a> tags from an HTML page, the HTQL query is:

<a>

To retrieve the names and values of all <input> tags in a page, the HTQL query is:

<input>:name, value

To retrieve the value of an <input> tag whose name is 'q', the HTQL query is:

<input (name='q')>:value

To retrieve the first row of the first table in a page, the HTQL query is:

<table>1.<tr>1

3. HTQL Overview

HTQL includes five basic types of operations: tag selection, dot operation, schema construction, HTQL function, and plain-tag selection. A *query* is a sequence of HTQL operations. Each query may extract a single item, a list of items, or a table of items from a Web page. A *tuple* is the returned single item, or one item in the returned list, or one row in the returned table.

Tag selection defines certain filtering conditions for tags in a given Web page. Dot operation combines multiple tag selection operations and applies them recursively to the page. Schema construction defines a table scheme from the page. HTQL function transforms the extracted Web data. Finally, plain-tag selection enables querying plain-text documents as well as HTML documents.

The following sections explain these operations.

3.1. Tag Selection

Tag selection extracts tags and their associated attributes from a given page. A tag-selection query is composed of delimiters, a tag name, options, conditions, ranges, and attributes, as demonstrated in Figure 1.

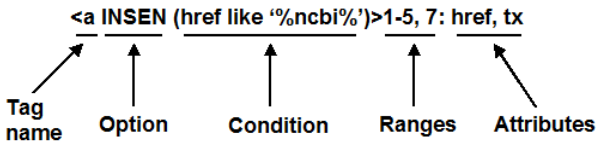

Figure 1. A typical tag-selection query. A tag-selection query is composed of delimiters, a tag name, options, conditions, ranges, and attributes. This query searches for <a> tags with an 'INSEN' option, which means the tag name is case insensitive. The query filters has a condition ('href' like '%ncbi%'), and will search for links that include a 'ncbi' pattern. The query will select only the first five and the seventh tags satisfying the condition. Finally, the query will extract the href and the tx (i.e., text) attributes from the returned <a> tags.

Table 1 shows more examples.

Table 1. Tag selection examples and meanings.

Tag Selection	Meaning
<a>1-4, 8	Get the first four and the eighth <a> tags.
<a>3-0#2	From the third to the last <a> tag, get every other tags.
<a (href = '/Entrez/') >	Get the <a> tag whose href is '/Entrez/'
<a EXCL (href is not null)>1	Get the first <a> tags that has an 'href' attribute but excluding the start- and end-tags.
<input>:name, value	Get the name and value attributes of all <input> tags.

A tag selection consists of a pair of tag delimiters '<' and '>' with some filtering conditions. Following the '<' delimiter, a *tag name* defines the tag to search for, followed by optional filtering *options* and *conditions*.

Options are reserved keywords, e.g., to specify whether the tag name is case sensitive. Table 2 shows more options.

Table 2. options and meanings.

Option	Meaning
SEN	Tag name and attribute name are case sensitive
INSEN	Tag name and attribute name are case insensitive
INCL	Start- and end-tags are included in the result
EXCL	Start- and end-tags are excluded from the result
RECUR	Tags recursively reside in another tag with a same tag name will not be counted
NORECUR	Tags recursively reside in another tag with a same tag name will still be counted
IRAN	Ranges are applied before conditions
ORAN	Ranges are applied after conditions
SEP	Treat the selected tags as separators to divide the text into multiple sections

Conditions are Boolean expressions enclosed by '(' and ')'. Expressions in HTQL follow a standard SQL syntax. Variables can be used in expression. Variables are attributes names of the current tag or global variables set by interface functions [e.g., by the setGlobalVariable() function in the COM interface].

Following the '>' delimiter, there can be optional *indices* in numerical ranges such as '1-4' and '5, 7', which mean that only tags in these ranges will be selected. Indices can be applied before or after the conditions are met depending on the options of IRAN and ORAN in the tag selection (see table 2). The default option is ORAN, which means that indices are applied after the conditions are met. Index 0 is used for the last index of the query result. For example, expression:

<a>2-0

retrieves from the second to the last <a> tags.

Following the indices, there is an optional attribute list led by a colon ':' and separated by commas. Attributes can be specific tag attributes or general attributes defined in table 3. By default, tag names and attribute names are case insensitive. However, if option SEN is given, they become case sensitive.

Table 3. General tag attributes.

General Attribute Name	Meaning
TN	Tag name
TX	Content excluding the start- and end-tags
HT	Content including the start- and end-tags
XX	Text after the end-tag
HX	Content after the start-tag and before the first tag following the end-tag
FX	Text after the start-tag and before the first tag following the start-tag
PX	Text before the start-tag and after the first tag preceding the start-tag
ST	The start-tag
ET	The end-tag
TH	Header of the table column

3.2. Dot Operation

Multiple tag selection operations can be concatenated by dot '.' operations, which means that a later tag selection is applied to results of the earlier tag selection recursively and the final results are the union of results from the final tag selection. For example, expression:

<table>1-2.<a>1-3

searches the first two <table> tags and, in each result table, retrieves the first three <a> tags. Expression:

<form>.<input>

searches for all <form> tags and retrieves all <input> tags that are enclosed in the <form> tags.

3.3. Schema Construction

Schema construction decomposes data into a list of fields, each constructed by a HTQL sub-expression. Schema construction starts by a delimiters '{', and ends by a delimiter '}'. Following the starting '{', there is a list of field construction terms in the form of "field_name=field_htql;", where field_htql represents an HTQL expression which extracts data from the previous results and field_name represents a name for the extracted data. For example, expression:

<table>.<tr> {column1=<td>1; column2=<td>2;}

constructs two fields from each <tr> of a table, where the first field is taken from the first <td> tag in the <tr> and is named 'column1', and the second field is taken from the second <td> tag in the <tr> and is named 'column2'. The field_htql can start with an attribute list, which extracts tag attributes from the previous results. For example, expression:

<form>.<input> {Name=:name; Value=:value;}

constructs fields 'Name' and 'Value' from the name and value attributes of the <input> tags that are enclosed in a <form> tag.

Schema construction results can be filtered by conditions after a '|' following the field list. For example, expression:

<form>.<input> {Name=:name; Value=:value; Type=:type | Type <> 'hidden'}

constructs a table with fields 'Name', 'Value' and 'Type' and removes tuples with a hidden type.

A second '|' delimiter in a schema construction marks the start of a number of field transformation terms, in the form of name-expression pairs separated by ';'. Names in the field transformation are optional, and when the name is not present, the result field name is copied from the first schema field where the transformation is applied. Field expressions can use variables from the schema fields or from global variables. Schema fields can be referred by name or by indices in the form of '%*n*', where *n* stands for the *n*'th field. For example, expression:

<table>.<tr> {C1=<td>1; C2=<td>2 | C1 <> C2 | CC=C1+%2 }

constructs two fields from every <tr> of a table, filters the results by a condition 'C1<>C2', and reconstructs a new field named 'CC' as the concatenation of C1 and C2.

In a schema construction expression, field construction, field condition and field transformation can all be optional, however, the correct number of '|' in frond of the corresponding terms must be preserved. Schema constructions can appear multiple times in an HTQL expression and can be used wherever a tag selection can be used. When a schema construction is followed by a tag selection, only the first field of the schema is used for the tag selection.

3.4. HTQL Functions

HTQL provides a number of *functions* to transform Web data. An HTQL function is marked by a leading '&', followed by the function name and an optional list of parameters parenthesized by '(' and ')'. A function can appear in anywhere a tag selection can be used. The function will be applied to the first field of each tuple of the previous results, and field names can be used as variables in the function parameters. Table 4 shows a list of frequently used HTQL functions. For example, expression:

&about .<About>.<Version>

returns the version of the HTQL release.

Table 4. HTQL functions

HTQL function	Meaning
&help	Returns a list of available functions in <function> tags
&about	Returns the HTQL version information

&tx	Remove all tags from the data
&txstr(max_len)	Remove all tags, and keep the length of result data within max_len
&html_encode	Encode text into HTML printable text
&html_decode	Decode HTML printable text back to plain text
&toupper	Translate into upper-case data
&tolower	Translate into lower-case data
&trim(chars,flags)	Remove any characters appearing in *chars*. *Flags* control whether to remove from the head, from the middle, from the tail, and from multiple line feeds. For example, if *flags*='1010', it removes from the head and the tail all characters in *chars*. For each flag, 0=no trim; 1=trim; 2= trim CRLF; 3=trim all
&get_url	Get the URL of a <a>, <base>, <frame> or <img> tag
& url	Translate to full URL address
&url_all	Translate all URLs to full URL addresses
&after(pattern, n)	Text after the n'th pattern
&before(pattern, n)	Text before the n'th pattern
&insert(str)	Insert str before the data in the source buffer
&insert_after(str)	Insert str after the data in the source buffer
&delete	Delete data from the source buffer
&update(str)	Replace the data with str in the source buffer
&replace(str)	Same as update(str)
&set_attribute(name, value)	Set an attribute value for the attribute name in the source buffer
&delete_attribute(name)	Delete the name attribute from the source page in the source buffer
&delete_attribute(name)	Delete the name attribute from the source page in the source buffer
&save(filename)	Save the (updated) source buffer into a file with the filename. Note that if there are multiple tuples in the results, the function may be called multiple times and the efficiency is decreased. Instead, the saveSourceData COM interface should be used.
&html_main_text	Get the main text of the HTML page
&html_main_date	Get the main date of the page
&html_title	Get the main title of the page

3.5. Plain-Tag Selection

Plain text can be queried in a similar way as hypertext by constructing plain-tags. A plain-tag is defined as text enclosed by a pair of start-tag and end-tag patterns. A plain-tag selection defines the start-tag and the end-tag within the tag-selection sentence in the form of "/'start-tag' ~ 'end-tag'/". Plain-tag selections use two '/' as the delimiters instead of the '<' and '>' in regular tag selections. For example, expression:

/'START'~'\n'/

selects all segments starting with a 'START' and ending with a new line character '\n'. If no end-tag is defined, the start-tag is used as a separator that divides the source text. For example, expression:

/'\n'/

separates the text by the new line character.

Conditions and options can be applied to plain tag selections as well. However, regular tag

selection has a default INCL option, while plain-tag selection has a default EXCL option, which means the start-tag and the end-tag are not included in the results of the plain tag selection by default. Furthermore, general attributes of a plain tag are supported in a limited basis.

4. HTQL Syntax

HTQL_expression ::= {HTQL_item}*

HTQL_item ::= tag_expression | schema_expression | function_expression

tag_expression ::= tag_selection {'.' tag_selection}*

tag_selection ::= hyper_tag_selection | plain_tag_selection

hyper_tag_selection ::= '<' tag_name selection_filters '>' {selection_ranges} { ':' attribute_list}

plain_tag_selection ::= '/' plain_tag selection_filters '/' {selection_ranges} { ':' attribute_list}

tag_name ::= ALPHA_NUMERIC

selection_filters ::= {selection_filter}*

selection_filter ::= ' ' selection_option | ' ' selection_condition

selection_option ::= 'SEN' | 'INSEN' | 'INCL' | 'EXCL' | 'RECUR' | 'NORECUR' | 'IRAN' | 'ORAN'

selection_condition ::= '(' BOOLEAN_EXPRESSION ')'

selection_ranges ::= selection_range {',' selection_ranges}

selection_range ::= NUMBER {'-' NUMBER} {'#' NUMBER}

attribute_list ::= attribute_name {',' attribute_list}

attribute_name ::= ALPHA_NUMERIC

plain_tag ::= start_pattern {'~' end_pattern}

start_pattern ::= ALPHA_NUMERIC

end_pattern ::= ALPHA_NUMERIC

schema_expression ::= '{' schema_definition {'|' schema_condition {'|' schema_transform}} '}'

schema_definition ::= schema_item {';' schema_definition}

schema_item ::= schema_name '=' { ':' attribute_list '.'} HTQL_expression

schema_name ::= ALPHA_NUMERIC

function_expression ::= '&' function_name { '(' function_parameters ')' }

function_name :: ALPHA_NUMERIC

function_parameters ::= | expression_list

expression_list ::= EXPRESSION {',' expression_list}

5. About HTQL

HTQL is designed and created by Dr. Liangyou Chen as part of his Ph.D. dissertation work:

> *Ad Hoc Integration and Querying of Heterogeneous Online Distributed Databases*. 2004. Mississippi State University.

Please cite the dissertation for references.