

Hyper-Text Query Language (HTQL)

-- A Web Programmer's Guide

Liangyou Chen
liangyou@gmail.com

1. What is HTQL

Hyper-Text Query Language (HTQL) is a language for the querying and transformation of HTML, XML and plain text documents. HTQL is developed in C++ with fast and efficient data extraction algorithms. This guide explains the use of HTQL COM interface for potential use in JavaScript, Visual Basic, and ASP applications. HTQL can be used to:

- 1) Extract HTML elements from HTML pages
- 2) Retrieve HTML page through HTTP protocol
- 3) Update HTML pages from applications

2. Installation

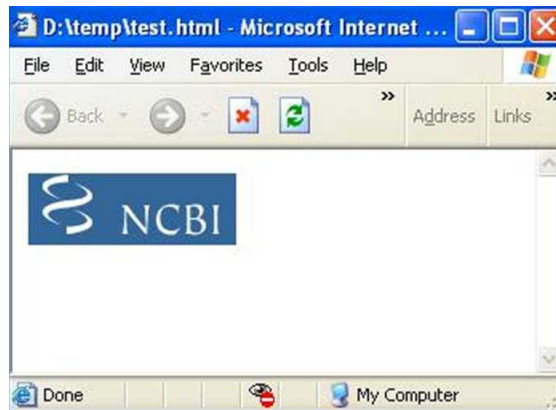
- 1) Download the HtqlCom.dll into a local directory, such as 'C:\htql\'.
2) Register the "HtqlCom.dll" by running:
C:\htql\> regsvr32 HtqlCom.dll

3. Demonstrating Examples

The following example shows the use of HTQL in an HTML page with JavaScript. The JavaScript code in this HTML page retrieves the first <a> tag from <http://www.ncbi.nlm.nih.gov/> and show it in the HTML body.

```
<!-- test.html -->
<html> <base href="http://www.ncbi.nlm.nih.gov/">
<body>
<script language=JavaScript>
    var a= new ActiveXObject("HtqlCom.HtqlControl");
    a.setUrl("http://www.ncbi.nlm.nih.gov/");
    a.setQuery("<a>");
    document.write(a.getValueByIndex(1));
</script>
</body>
</html>
```

The <base> tag in this page allows images with relative URLs to be shown correctly relative to the <http://www.ncbi.nlm.nih.gov/>. Visually, the above HTML page is shown as:



The following Visual Basic example does the same thing and shows the result in a message box:

```
' VB example
Dim a As Object
Set a = CreateObject("HtqlCom.HtqlControl")
i = a.setUrl("http://www.ncbi.nlm.nih.gov/")
i = a.setQuery("<a>")
MsgBox (a.getValueByIndex(1))
```

And it displays a message box:



A more complete example is shown in the following, which retrieved HTQL results in a loop using the moveFirst and moveNext functions until the isEOF function returns true.

```
<!-- test2.html -->
<html> <base href="http://www.ncbi.nlm.nih.gov/">
<body>
<script language=JavaScript>
  var a= new ActiveXObject("HtqlCom.HtqlControl");
  a.setUrl("http://www.ncbi.nlm.nih.gov/");
  a.setQuery("<input>:name, type, value");
  for (a.moveFirst(); !a.isEOF(); a.moveNext()){
    for (i=1; i<a.getFieldsCount(); i++)
      document.write(a.getValueByIndex(i) + ", ");
    document.write("<br>\n");
  }
</script>
</body>
</html>
```

This example queries a page at <http://www.ncbi.nlm.nih.gov/> with an HTQL expression:

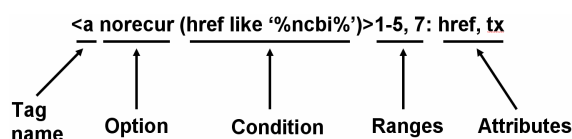
`<input>:name, type, value`

and displays the results in the HTML body as comma separated lines.

By changing the URL in the `setUrl` and the query in the `setQuery` sentences, the reader can use this example to display results for queries in the rest of this guide.

4. HTQL Basic

The basic of HTQL is the tag selection expression, which searches for HTML tags satisfying a certain condition, and returns a table of results. A tag selection consists of a pair of tag delimiters '<' and '>' and some filtering conditions. After the '<' delimiter, a tag name is used to represent the tag to be searched for. Following the tag name and preceding the '>' delimiter, there can be optional filtering options and conditions. Options are reserved keywords with predefined meanings such as whether the tag name is case sensitive. Conditions are Boolean expressions enclosed in '(' and ')'. After 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, and, following the ranges, there can be an optional attribute list led by a colon ':' and separated by commas. A complete example of a tag selection is shown in the following:



which searches the <a> tags with a 'norecur' option, filters the 'href' attribute with a pattern '%ncbi%', returns only the first five and the seventh tags satisfying the condition, and retrieves the href and the tx attributes from the <a> tags.

More tag selection examples with their meanings are given in the following table.

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

A list of options and their meanings are:

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

Expressions in HTQL follow a standard SQL syntax. Variables can be used in the tag condition expression. Variables can be global variables set by the setGlobalVariable function or names of tag attributes. The following example shows how to use global variable for a tag condition:

```
a.setGlobalVariable("v", "submit");
a.setQuery("<input (type=v)>");
```

which is equivalent to

```
a.setQuery("<input (type='submit')>");
```

Tag attributes include attributes shown specifically in the tag and general attributes defined in the following table:

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

Range indices can be applied before the conditions are met or after the condition are met depending on the options of IRAN and ORAN in the tag selection. 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 the second to the last <a> tag.

If a column ':' is present, a list of attribute after the column are attributes to be selected, where attribute names can be actual attribute names or general attribute names. By default, tag names and attribute names are case insensitive. However, if option SEN is given, they becomes case sensitive.

5. Advanced HTQL Syntax

5.1 Dot Operation

Multiple tag selection operations can be concatenated by dot '.' operations, meaning 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 latest 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.

5.2 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 the 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 front 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.

5.3 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 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. The following is a list of frequently used HTQL functions.

HTQL function	Meaning
<i>&help</i>	Returns a list of available functions in <function> tags
<i>&about</i>	Returns the HTQL version information
<i>&tx</i>	Remove all tags from the data
<i>&txstr(max_len)</i>	Remove all tags, and keep the length of result data within max_len
<i>&html_encode</i>	Encode text into HTML printable text
<i>&html_decode</i>	Decode HTML printable text back to plain text
<i>&toupper</i>	Translate into upper-case data
<i>&tolower</i>	Translate into lower-case data
<i>&get_url</i>	Get the URL of a <a>, <base>, <frame> or tag
<i>&url</i>	Translate to full URL address
<i>&url_all</i>	Translate all URLs to full URL addresses
<i>&after(pattern, n)</i>	Text after the n'th pattern
<i>&before(pattern, n)</i>	Text before the n'th pattern
<i>&insert(str)</i>	Insert str before the data in the source buffer
<i>&insert_after(str)</i>	Insert str after the data in the source buffer
<i>&delete</i>	Delete data from the source buffer
<i>&update(str)</i>	Replace the data with str in the source buffer
<i>&replace(str)</i>	Same as update(str)
<i>&set_attribute(name, value)</i>	Set an attribute value for the attribute name in the source buffer
<i>&delete_attribute(name)</i>	Delete the name attribute from the source page in the source buffer
<i>&delete_attribute(name)</i>	Delete the name attribute from the source page in the source buffer
<i>&save(filename)</i>	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.

For example, expression:

&about .<About>.<Version>

returns the version of the HTQL release.

5.4 Plain Tag Selection

Plain text can be queried in a similar way as hyper text 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'/"
```

separate 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.

6. The HTQL COM Interface

The HTQL COM provides interface functions to set the source data and query expressions for query and retrieve query results. Once set, the source data are kept in an internal buffer of the HTQL object, and can be queried repeatedly. Query results are kept in an internal data structured and can be navigated in a sequential mode. The following subsections describe interfaces that support these operations.

6.1 Set the Source Data

The source data for HTQL to query can be set in three methods. The first method is to call the `setUrl` function with an URL parameter. By this method, the Web page will be fetched into the internal buffer either by an HTTP connection, or by a local file reading. The prototype of the `setUrl` function is:

```
long setUrl(BSTR url);
```

The second method is to set the source buffer from variables. The `setSourceData` function copies the data from a variable into the internal buffer, and the `setSourceUrl` sets the URL information about the data. The prototypes of these two functions are:

```
long setSourceData(BSTR data, long length)
```

```
long setSourceUrl(BSTR url)
```

The third method is to post a Web form action to a website and retrieve the form-generated data from the remote server. It is completed by calling the `setUrlToPost` function to set the URL of the form action, using the `setUrlParameter` and `setUrlCookie` to set form parameters and cookies, and calling the `postUrl` function to activate the form action. The `postUrl` function submits the form action by an HTTP connection to the remote server and retrieves the form-generated data into the internal buffer. The prototypes of these functions are:

```
long setUrlToPost(BSTR url)
```

```
long setUrlParameter(BSTR name, BSTR value)
```

```
long setUrlCookie(BSTR name, BSTR value)
long postUrl()
```

6.2 Set HTQL queries

Once the source data is set in the internal buffer, the data can be queried repeatedly for multiple HTQL queries. The queries are set by the `setQuery` and `dotQuery` functions. The `setQuery` function clears the previous query results and poses a new query to the source data. Query results are stored in an internal structure, which can be navigated by the `moveFirst` and `moveNext` functions and tested by the `isEOF` function. The prototypes of these functions are:

```
long setQuery(BSTR query)
long moveFirst()
long moveNext()
long isEOF()
```

The `dotQuery` function can be used to query data from the previous results in the context of a dot operation. The prototype of the `dotQuery` function is:

```
long dotQuery(BSTR query)
```

6.3 Use Global Variables

Global variables can be set by the `setGlobalVariable` function and can be used in expressions in a tag selection operation, a schema construction sentence, or a parameter of an HTQL function. Global variable should be set after the source data is set, and can be used continuously for different queries until the source data is reset. The prototype is:

```
long setGlobalVariable(name, value)
```

6.4 Get HTQL Query Results

HTQL query results can be retrieved by `getValueByIndex`, `getValueByName`, `getTuplesCount`, `getFieldsCount` and `getFieldName` functions. The prototypes are:

```
BSTR getValueByIndex(long index)
BSTR getValueByName(BSTR name)
long getTuplesCount()
long getFieldsCount()
BSTR getFieldName(long index)
```

6.5 Save the Updated Source

Data in the source buffer can be saved into a local file using the `saveSourceData` function. It takes a file name as the parameter. Note that source buffer may have been updated by HTQL functions such as the `&insert`, `&replace`, and `&delete` functions. The prototype of the function is:

```
long saveSourceData(BSTR filename)
```

6.6 Reset the COM Object

The COM object can be reset by the `reset` function, which destroys internal variables and data structures of the COM object. The prototype is:

```
void reset()
```

7. 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 }

8. 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 this dissertation for references.