Public Variables:

1) Private PUB_WEB_DATA_ELEMENTS_HASH As clsTypeHash

`PUB_WEB_DATA_ELEMENTS_HASH` is a user-defined variable that is used to store data from the requests that were previously made by user. It consists of keys and corresponding data values. Each key consists of the ticker, for example MSFT, which selects a company, and the element number, which stands for a specific data value related to that company. For example, 617 (stands for Earnings Growth Rates - Past 5 Years sourced from Yahoo). The two strings are separated by "|".

```
TICKER0_STR|ELEMENT_VAL
```

In our example: MSFT|617

This key is structured so that it is unique for every user input. For every possible combination of 3 pieces of information: ticker, type of metric and source (latter two are embedded in the element number), there is only one unique value that exists. This ensures that there are no duplicate keys in the hash table.

The value that is related to each key includes the data requested by the user, which corresponds to a particular Ticker|Element. For example, for MSFT|617 (Earnings Growth Rates -- Past 5 Years – Company sourced from Yahoo) the corresponding value will be 0.1076.

@ RETRIEVE_WEB_DATA_ELEMENT_FUNC:

```
If PUB_WEB_DATA_ELEMENTS_HASH.Exists(KEY_STR) = True Then
```

This code checks whether an entry for a particular key already exists in the hash table. If it is true, meaning that the key already exists, the data corresponding to the key is simply returned from the hash table rather than being downloaded from the Internet.

2) Public PUB_WEB_DATA_PAGES_HASH As clsTypeHash

`PUB_WEB_DATA_PAGES_HASH` is used to store XML/HTML code of the web-page that was used for a particular user request. Source URLs are used as keys, and XML/HTML codes are stored as values.

@ RETRIEVE_WEB_DATA_ELEMENT_FUNC:

```
If PARAM_RNG(0) <> "Calculated" And
PUB_WEB_DATA_PAGES_HASH.Exists(SRC_URL_STR) = False Then
```

This code is executed if the data requested by user doesn't require additional calculations (e.g. element number is not from 0.txt list) and the web-page (XML/HTML code), that contains the data, has not been saved yet. The advantage of using a hash table versus a public array is that when checking whether a required XML/HTML code has already been saved is performed much faster by using a source URL as the key. Another advantage of this approach is that there are multiple elements that require data from the same web-page. Therefore, once a particular XML/HTML code was saved, it could be used for different user requests (element numbers) for a particular ticker (no need to download additional data).

```
RESULT_VAL = PARSE_WEB_DATA_FRAME_FUNC(PUB_WEB_DATA_PAGES_HASH(SRC_URL_STR),
"" & PARAM_RNG(1), TICKER1_STR, ERROR_STR)
```

The advantage of using a hash table in this case is that the required XML/HTML code could be found quickly using URL as the key, rather than looping through an array. XML/HTML is then passed for parsing.

3) Private PUB_WEB_DATA_RECORDS_HASH As clsTypeHash

The variable is used to store information about every element from https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i: 0 to 9). `CStr(ELEMENT_VAL)` – element number is used as the key. The corresponding value includes all the information (Source, url, search pattern strings, etc.) about the element.

@ RETRIEVE_WEB_DATA_ELEMENT_FUNC:

```
PARAM_RNG = PUB_WEB_DATA_RECORDS_HASH(CStr(ELEMENT_VAL))
```

The code is used to assign the string from https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i: 0 to 9) that relates to a particular element. The advantage of using the hash table in this case is that the required string could be quickly found using element number `(CStr(ELEMENT_VAL))` as the key.

@LOAD_WEB_DATA_RECORDS_FUNC:

```
If PUB_WEB_DATA_RECORDS_HASH.Exists(TEMP_STR) = True Then
PUB_WEB_DATA_RECORDS_HASH.Remove (TEMP_STR)
```

In this case the “Exists” method is used in order to check if a particular entry already exists in the hash table. If it does, then the entry is removed and a new one is added instead.

4) Public PUB_WEB_DATA_TABLES_FLAG As Boolean

`PUB_WEB_DATA_TABLES_FLAG` is used to check whether variables that are used to store the data (array and hash tables) have been set up.

@ RETRIEVE_WEB_DATA_ELEMENT_FUNC:

```
If PUB_WEB_DATA_TABLES_FLAG = False Then: Call START_WEB_DATA_SYSTEM_FUNC
```

This code is used to check whether the variables (array and hash tables) have been set up prior to extracting the data from the web. If they haven’t been set up, then the `START_WEB_DATA_SYSTEM_FUNC` is called and it sets up the variables.

5) Private Const PUB_WEB_DATA_FILES_VAL As Long = 9

`PUB_WEB_DATA_FILES_VAL = 9` stands for the number of .txt files that contain the information about the elements and their corresponding URLs (9 in total). Theses .txt files are stored online, and the URL to access these .txt files is stored in `PUB_WEB_DATA_FILES_PATH_STR`.

`PUB_WEB_DATA_FILES_PATH_STR` is used to store the 1st portion of a URL of a web-page (the portion that is constant for all the URLs used for data extraction) that contains the information about all the elements and the corresponding URLs.

The URL is being “completed” using the following code to retrieve data (see LOAD_WEB_DATA_RECORDS_FUNC):

```
For i = 0 To PUB_WEB_DATA_FILES_VAL

SRC_URL_STR = PUB_WEB_DATA_FILES_PATH_STR & CStr(i) & ".txt"
```

In this case `PUB_WEB_DATA_FILES_VAL = 9`. Then, I (0-9) is being converted to a string, added to the (`PUB_WEB_DATA_FILES_VAL`). Finally, “.txt” is being added to the URL to reference one of the files. These results in the following links:

https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-0.txt
https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-1.txt
https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-2.txt
https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-3.txt
https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-4.txt
https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-5.txt
https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-6.txt
https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-7.txt

https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-8.txt
https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-9.txt

Data is separated into different .txt files depending on the source (web-sites) of the elements. The logic is as follows:

0.txt contains all the elements that require additional calculations
1.txt contains all the elements that come from MSN
2.txt contains all the elements that come from Yahoo
3.txt contains all the elements that come from Google
4.txt contains all the elements that come from Morningstar
5.txt contains all the elements that come from Reuters
6.txt contains all the elements that come from Zacks
7.txt contains all the elements that come from AdvFN
8.txt contains all the elements that come from Earnings
9.txt contains all the elements that come from the other sources, such as BarChart, StockHouse, ETFOutlookHL and others.

Each of these links is assigned to `SRC_URL_STR`, which is used as an input for `SAVE_WEB_DATA_PAGE_FUNC`.

6) Private Const PUB_WEB_DATA_RECORDS_VAL As Long = 20000

`PUB_WEB_DATA_RECORDS_VAL` is a variable used to store maximum number of elements. Each element is a different type of metric extracted from different sources. For example Market Cap from Yahoo (941), P/E Ratio (TTM) from Reuters (13626), Basic Weighted Average Shares - FQ1 from Google (2887) and many others. Currently, the highest element number is 17002, which corresponds to FY10 Total Revenue extracted from Morningstar.

`PUB_WEB_DATA_RECORDS_VAL` is set above the highest element number that currently exists in order to allow user to add additional new elements.

@ RETRIEVE_WEB_DATA_ELEMENT_FUNC:

```
If ELEMENT_VAL > PUB_WEB_DATA_RECORDS_VAL Then
   RETRIEVE_WEB_DATA_ELEMENT_FUNC = "EOL"
```

The purpose of this code is to check whether the element number (`ELEMENT_VAL`), which was input by the user, corresponds to an existing element in the 0 – 9 .txt files. If user inputs `ELEMENT_VAL` higher than 20000, which implies that element user is requesting doesn't exist, the function will return "EOL" (End Of the Line).

However, it is important to notice that since `PUB_WEB_DATA_RECORDS_VAL = 20000` and the highest index of an element that is currently exists is 17006, if a user inputs element index that is higher than 17006, but lower 2000, he will get “Error” returned and not “EOL”.

7) Private Const PUB_WEB_DATA_PAGES_VAL As Long = 30000

`PUB_WEB_DATA_PAGES_VAL` is used to determine size of `PUB_WEB_DATA_PAGES_MATRIX` or `PUB_WEB_DATA_PAGES_HASH` depending on where user wants to store the information. `PUB_WEB_DATA_PAGES_HASH` or `PUB_WEB_DATA_PAGES_MATRIX` is used to store XML/HTML code of the web pages that were used to extract data for user requests.

`PUB_WEB_DATA_PAGES_VAL` is set equal to 30000, where 30000 is artificially high number, which is assumed to be never reached, since it is very unlikely that user will be making requests from more than 30000 different source during one session.

After loading the text files with the `LOAD_WEB_DATA_RECORDS_FUNC` 10 entries are made. These 10 entries correspond to the .txt files loaded from https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i:0 to 9).

8) Private PUB_WEB_DATA_PAGES_MATRIX(1 To PUB_WEB_DATA_PAGES_VAL, 1 To 2) As String

`PUB_WEB_DATA_PAGES_MATRIX(1 To PUB_WEB_DATA_PAGES_VAL, 1 To 2)` is used to store XML/HTML code of the web-pages from which the data is extracted, if user prefers to store the data using array rather than hash table (case 0 in `SAVE_WEB_DATA_PAGE_FUNC`).

However, using array for these purposes significantly slows down the overall process. The routine SAVE_WEB_DATA_PAGE_FUNC checks if a particular XML/HTML code has already been saved. To do that the function loops through every single element of the array.

The array takes 6.6 milliseconds, while the collection takes 1.7 milliseconds.

9) Private Const PUB_WEB_DATA_ELEMENTS_VAL As Long = 100000

`PUB_WEB_DATA_ELEMENTS_VAL` is used as a variable that determines the size of `PUB_WEB_DATA_ELEMENTS_HASH`, when it is being set:

```
PUB_WEB_DATA_ELEMENTS_HASH.SetSize PUB_WEB_DATA_ELEMENTS_VAL
```

`PUB_WEB_DATA_ELEMENTS_HASH` is used to store data about user requests. `TICKER0_STR|ELEMENT_VAL` is used as a key and the value is equal to a value that was returned to the user.

`PUB_WEB_DATA_ELEMENTS_VAL` is set equal to 100000, where 100000 is artificially high number, which is assumed to be never reached, since it is very unlikely that user will make more than 100000 requests of different pieces of data during one session.

10) Private Const PUB_WEB_DATA_FILES_PATH_STR As String = https://raw.github.com/rnfermincota/IVEY/master/smf-elements/smfelements-.

Randy Harmelink's stock market functions add-in (finance.groups.yahoo.com/group/smf_addin//) is using local .txt files to store the information regarding the elements and corresponding urls. Therefore, for his code to be executed properly `PUB_WEB_DATA_FILES_PATH_STR` should be equal to "C:\Documents and Settings\ AddIns\smf-elements-" (or whatever the local address is). The problem with this approach is that, when a user downloads the add-in, he/she has to go into the code and change the address manually to the one that contains .txt files.

This problem is resolved through usage of the .txt files stored on-line. This makes the address the same no matter who and where is using the add-in, so no additional changes are required.

11) Private Const PUB_WEB_DATA_ELEMENT_LOOK_STR As String = "~~~~~"

`PUB_WEB_DATA_ELEMENT_LOOK_STR` is used as a "dummy" string variable, it is used in the web-page address instead of a ticker, which is being input by user. For example: http://finance.yahoo.com/q/ks?s=~~~~~. Once user runs the function, he inputs ticker, for example MSFT. Then the "~~~~~" portion of the url address is being replaced by the ticker (MSFT), which results in the following url string: http://finance.yahoo.com/q/ks?s=MSFT. This url is used to download data on Microsoft.

<u>@ RETRIEVE_WEB_DATA_ELEMENT_FUNC:</u>

```
SRC_URL_STR = Replace(PARAM_RNG(2), PUB_WEB_DATA_ELEMENT_LOOK_STR,
TICKER2_STR)
```

The url address is assigned to `PARAM_RNG(2) = "http://finance.yahoo.com/q/ks?s=~~~~~"`. It includes `PUB_WEB_DATA_ELEMENT_LOOK_STR` = "~~~~~". `PUB_WEB_DATA_ELEMENT_LOOK_STR` is being replaced by the ticker, which is part of the user input. After that `PARAM_RNG(2) = "http://finance.yahoo.com/q/ks?s=TICKER"`. This is url is used to download the data on a particular company.

12) Private Const PUB_WEB_DATA_ELEMENT_DELIM_STR As String = ";"

`PUB_WEB_DATA_ELEMENT_DELIM_STR = ";"` is used as a delimiter to split the strings that contain the information about each element (source, url, search pattern strings, etc.) taken from https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i: 0 to 9).

@ RETRIEVE_WEB_DATA_ELEMENT_FUNC:

```
PARAM_RNG = Split(PARAM_RNG & CASES_STR, PUB_WEB_DATA_ELEMENT_DELIM_STR)
```

This code is used to merge `PARAM_RNG & CASES_STR` and then split it into separate substrings each containing a separate piece of information about a particular element. For example, for element 3500:

```
PARAM_RNG = Google;Annual Income Statement -- Diluted Weighted Average Shares
-- FY1;http://www.google.com/finance?fstype=ii&q=~~~~~;1;INCANNUALDIV;Diluted
Weighted Average Shares; ; ;0;</TABLE;0;0
```

It is merged with `CASES_STR = ";N/A;N/A;N/A;N/A;N/A;N/A;N/A;N/A;N/A;N/A"`

And then split into:

```
PARAM_RNG(0) = "Google"
PARAM_RNG(1) = "Annual Income Statement -- Diluted Weighted Average Shares --
FY1"
PARAM_RNG(2) = "http://www.google.com/finance?fstype=ii&q=~~~~~"
PARAM_RNG(3) = "1"
PARAM_RNG(4) = "INCANNUALDIV"
PARAM_RNG(5) = "Diluted Weighted Average Shares"
PARAM_RNG(6) = " "
PARAM_RNG(7) = " "
PARAM_RNG(8) = "0"
PARAM_RNG(9) = "</TABLE"
PARAM_RNG(10) ="0"
PARAM_RNG(11) = "0"
PARAM_RNG(12) = "N/A"
PARAM_RNG(13) = "N/A"
PARAM_RNG(14) = "N/A"
PARAM_RNG(15) = "N/A"
PARAM_RNG(16) = "N/A"
PARAM_RNG(17) = "N/A"
PARAM_RNG(18) = "N/A"
PARAM_RNG(19) = "N/A"
PARAM_RNG(20) = "N/A"
PARAM_RNG(21) = "N/A"
```

`PARAM_RNG(i)` are later used in the parsing algorithm.

13) Public Const PUB_WEB_DATA_SYSTEM_ERROR_STR As String = "Error"

The advantage of using one public variable for error is that no matter in which function the error occurs it could be identified using the same code, for example:

```
If DATA_STR = PUB_WEB_DATA_SYSTEM_ERROR_STR Then: GoTo ERROR_LABEL
```

In our case `RETRIEVE_WEB_DATA_ELEMENT_FUNC` calls `SAVE_WEB_DATA_PAGE_FUNC`, which in turn calls `CALL_WEB_DATA_PAGE_FUNC`. No matter where error occurs, comparison against the same "Error" string could be used in any function.

Functions:

1) START_WEB_DATA_SYSTEM_FUNC

The purpose of `START_WEB_DATA_SYSTEM_FUNC` is to setup array and hash table variables prior to extracting data from the web.

First the function sets up `PUB_WEB_DATA_PAGES` variables. These variables are used to store XML/HTML code of the web-pages from which the data is extracted. Depending on user's choice in `SAVE_WEB_DATA_PAGE_FUNC`, XML/HTML code could be saved either as an array or using has hash table (case 0 for array, case else for hash table). Therefore, the variables for array are created and set to null and variables for hash table are created and set to null:

```
For i = 1 To PUB_WEB_DATA_PAGES_VAL
PUB_WEB_DATA_PAGES_MATRIX(i, 1) = "": PUB_WEB_DATA_PAGES_MATRIX(i, 2) = ""
Next i

Set PUB_WEB_DATA_PAGES_OBJ = New Collection

Set PUB_WEB_DATA_PAGES_HASH = New clsTypeHash
PUB_WEB_DATA_PAGES_HASH.SetSize PUB_WEB_DATA_PAGES_VAL
PUB_WEB_DATA_PAGES_HASH.IgnoreCase = False
```

Next, the function creates and sets to null hash table variables that are used to store information about each element (source, url, search pattern strings) - `PUB_WEB_DATA_RECORDS_HASH`, and information about previous user requests, e.g. particular element for a particular ticker - `PUB_WEB_DATA_ELEMENTS_HASH`:

```
Set PUB_WEB_DATA_RECORDS_HASH = New clsTypeHash
PUB_WEB_DATA_RECORDS_HASH.SetSize PUB_WEB_DATA_RECORDS_VAL
PUB_WEB_DATA_RECORDS_HASH.IgnoreCase = False

Set PUB_WEB_DATA_ELEMENTS_HASH = New clsTypeHash
PUB_WEB_DATA_ELEMENTS_HASH.SetSize PUB_WEB_DATA_ELEMENTS_VAL
PUB_WEB_DATA_ELEMENTS_HASH.IgnoreCase = False
```

Finally, the function sets `PUB_WEB_DATA_TABLES_FLAG` equal to true, which indicates that all the variables, that are used to store data are set:

```
PUB_WEB_DATA_TABLES_FLAG = True
```

2) GET_WEB_DATA_PAGE_FUNC

The purpose of `GET_WEB_DATA_PAGE_FUNC` is to retrieve XML/HTML code of the web-page, which url is specified in the input.
Inputs:

`pURL` – url of the web-page
`pPos` – used in TRIM_LINE
`pLen` – used in TRIM_LINE
`pOffset` – used in TRIM_LINE
`pUseIE` – defines the method XML/HTML code will be downloaded in `CALL_WEB_DATA_PAGE_FUNC` (0,1 – XML Object, 2 – Internet Explorer Object, Else – HTML Document Object)

`GET_WEB_DATA_PAGE_FUNC` calls `SAVE_WEB_DATA_PAGE_FUNC`, which in turn calls `CALL_WEB_DATA_PAGE_FUNC`.

`CALL_WEB_DATA_PAGE_FUNC` allows user to choose one 3 options on how to download the web-page: XML Object (`HTTP_TYPE` Case 0,1), Internet Explorer Object (`HTTP_TYPE` Case 2) and HTML Document (`HTTP_TYPE` Case Else). Default value is `HTTP_TYPE = 0`, which corresponds to XML Object, and that is the one that most commonly used.

`CALL_WEB_DATA_PAGE_FUNC` uses XML object from Microsoft XML, v. 6.0 Library. XML object is used to send a request to the server that store the required web-page and then download the XML/HTML code of the web-page.

First, the function creates a new XML object named `HTTP_OBJ`:

```
Dim HTTP_OBJ As New MSXML2.XMLHTTP60
```

Next, it checks whether user wants to download ("GET" method) or upload ("POST" method) data. `HTTP_TYPE = 0` stands for downloading data, other values of `HTTP_TYPE` stand for uploading data. Default value is `HTTP_TYPE = 0`:

```
If HTTP_TYPE = 0 Then
```

Following that, HTTP_OBJ object is used to send a request to the web-server. It is done through usage of “GET” method:

```
HTTP_OBJ.Open "GET", SRC_URL_STR, False
HTTP_OBJ.send
```

Next, the function checks the status of the HTTP_OBJ object. Status is a property of an XML object, which is used in order to check the status of the request that was sent to the web-server. Status 0 means the response to HTTP request was empty. Status 200 is equal to “OK” meaning that HTTP request was successful. For “GET” method this means that the response will contain the requested entity. For “POST” method this means that the response will contain the result of the action (uploading data).

CALL_WEB_DATA_PAGE_FUNC checks whether HTTP_OBJ Status is equal 0 or 200 (e.g. whether it was successful), and if so saves the response as DATA_STR:

```
Select Case HTTP_OBJ.Status
   Case 0: DATA_STR = HTTP_OBJ.ResponseText
   Case 200: DATA_STR = HTTP_OBJ.ResponseText
   Case Else: GoTo ERROR_LABEL
End Select
```

DATA_STR containing XML/HTML is then passed back to SAVE_WEB_DATA_PAGE_FUNC

3) SAVE_WEB_DATA_PAGE_FUNC

The function is used to save the XML/HTML code of the web-page that contains the required data. There are 2 options how the XML/HTML code could be saved: using 2-dimensional array (Case 0) and using Collection Object (Case Else).

In Case 0, the function uses PUB_WEB_DATA_PAGES_MATRIX to store the data. PUB_WEB_DATA_PAGES_MATRIX is 2-dimensional array. The 1st column of PUB_WEB_DATA_PAGES_MATRIX consists of HTTP_TYPE:SRC_URL_STR, which is unique and used as a key. The 2nd column is used to store the data (XML/HTML code of the web-pages).

The function loops through the contents of the 1st column of PUB_WEB_DATA_PAGES_MATRIX and compares HTTP_TYPE:SRC_URL_STR (the key) of the web-page that was requested to the contents of the 1st column. If there is a match that means that the web-page has already been requested during this session and XML/HTTP has already been downloaded (since HTTP_TYPE:SRC_URL_STR is unique). Therefore, the data stored in the 2nd column of PUB_WEB_DATA_PAGES_MATRIX is returned:

```
Case PUB_WEB_DATA_PAGES_MATRIX(i, 1) = HTTP_TYPE & ":" & SRC_URL_STR: Exit
For
DATA_STR = PUB_WEB_DATA_PAGES_MATRIX(i, 2)
```

If the loop reaches an empty value in the 1st column of `PUB_WEB_DATA_PAGES_MATRIX`, this means that the function has gone through all the entries and the match hasn't been found. In this case the required web-page must be downloaded by using `CALL_WEB_DATA_PAGE_FUNC`:

```
Case PUB_WEB_DATA_PAGES_MATRIX(i, 1) = ""
         DATA_STR = CALL_WEB_DATA_PAGE_FUNC(SRC_URL_STR, HTTP_TYPE)
         If CLEAN_FLAG = True Then: GoSub CLEAN_LINE
         PUB_WEB_DATA_PAGES_MATRIX(i, 1) = HTTP_TYPE & ":" & SRC_URL_STR
         PUB_WEB_DATA_PAGES_MATRIX(i, 2) = DATA_STR
DATA_STR = PUB_WEB_DATA_PAGES_MATRIX(i, 2)
```

A new key for the 1st column, the XML/HTML code is saved in 2nd column and returned from the function.

Case Else uses a collection object instead of an array. The logic is similar to Case 0. The function checks if the key already exists, if it does – then it returns the value, if it doesn't – the function uses `CALL_WEB_DATA_PAGE_FUNC` to download the data and creates a new entry in the collection.

```
If TRIM_FLAG = True Then: GoSub TRIM_LINE
```

The purpose of `TRIM_LINE` is to extract a portion of XML/HTML code that starts from `POS_VAL` and is equal to `LEN_VAL` in length. `POS_VAL` and `LEN_VAL` are specified by the user. If `POS_VAL` is an integer then the function will cut the string starting from `POS_VAL`, if `POS_VAL` is not in integer, then the function finds `POS_VAL` in the `DATA_STR` (XML/HTML code) and starts from its position. If `LEN_VAL` is smaller than the number of characters in `DATA_STR` starting from `POS_VAL`, then the length of the substring to be cut is equal to `LEN_VAL`, if it is higher than the number of characters – the rest of `DATA_STR` is returned:

```
j = IIf(IsNumeric(POS_VAL), POS_VAL, InStr(DATA_STR, POS_VAL) + OFFSET_VAL)
k = IIf(j + LEN_VAL <= Len(DATA_STR), LEN_VAL, Len(DATA_STR) - j + 1)
DATA_STR = Mid(DATA_STR, j, k)
```

The purpose of CLEAN_LINE and the `PARSE_WEB_DATA_PAGE_SYNTAX_FUNC` is to replace HTML syntax characters with their corresponding representations in the web-page. For example, "&amp;", which is HTML syntax for ampersand is replaced by actual "&":

```
DATA_STR = Replace(DATA_STR, "&amp;", "&")
```

Another function of `PARSE_WEB_DATA_PAGE_SYNTAX_FUNC` is to replace <th>, </th> tags with <td>, </td> tags.

```
DATA_STR = Replace(DATA_STR, "<TH", "<td")
DATA_STR = Replace(DATA_STR, "</TH", "</td")
DATA_STR = Replace(DATA_STR, "<th", "<td")
DATA_STR = Replace(DATA_STR, "</th", "</td")
```

<th> is HTML tag that is used identify heading cells of a table. In its nature heading cell is the same cell as any other cell in the column, except for that it is identified differently in XML/HTML code of a web-page. In some instances the required piece of data is contained within the heading cell of a table. Therefore all the <th> tags should be replaced with <td> (regular cell) tags, so that `PARSE_WEB_DATA_CELL_FUNC` could treat heading cell as regular cell and extract data from it.

4) LOAD_WEB_DATA_RECORDS_FUNC

The purpose of `LOAD_WEB_DATA_RECORDS_FUNC` is to download the information about each element (source, url, search pattern strings, etc.) from https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i: 0 to 9) and store it in hash table using `PUB_WEB_DATA_RECORDS_HASH` variable. Element number is used as a key and the rest of the string is stored as value. For example:

Annual Cash Flow -- Net Change in Cash -- FY1 sourced from Google (Element 3956)

Key = 3956

Value = Google;Annual Cash Flow -- Net Change in Cash -- FY1;http://www.google.com/finance?fstype=ii&q=~~~~~;1;CASANNUALDIV;Net Change in Cash; ; ;0;</TABLE;0;0

The general algorithm of `LOAD_WEB_DATA_RECORDS_FUNC` consists of two loops, one inside another. First loop from i = 0 to 9 is used to access the web pages at https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i: 0 to 9) at download their HTML/XML code. This is done through usage of `SAVE_WEB_DATA_PAGE_FUNC` function:

```
For i = 0 To PUB_WEB_DATA_FILES_VAL
SRC_URL_STR = PUB_WEB_DATA_FILES_PATH_STR & CStr(i) & ".txt"
DATA_STR = SAVE_WEB_DATA_PAGE_FUNC(SRC_URL_STR, 0, False, 0, False)
```

Once the web-page is downloaded it is saved in `DATA_STR`. Next, each web-page is split into separate lines and saved in `DATA_STR`. `Chr(10)` or new line, so the number of strings the web-page is split into is equal to the number of lines in this page. Each line is saved as a separate element in `DATA_ARR` array:

```
DATA_ARR = Split(DATA_STR, Chr(10), -1, vbTextCompare)
```

Next, for each web-page saved and split into separate lines, the function loops for each line and makes necessary adjustments to each line, where `SROW` – number of the first line and `NROWS` – number of the last line:

```
SROW = LBound(DATA_ARR)
NROWS = UBound(DATA_ARR)
For j = SROW To NROWS
        DATA_STR = DATA_ARR(j)
```

Next, the functions checks if the line is not empty, and makes necessary adjustments by replacing Randy Harmelink's functions with Nico's functions:

```
If DATA_STR <> Chr(13) And Trim(DATA_STR) <> "" And DATA_STR <> "0" Then
DATA_STR = Trim(DATA_ARR(j))
DATA_STR = Replace(DATA_STR, Chr(13), "")
DATA_STR = Replace(DATA_STR, "smfGetTagContent", "PARSE_WEB_DATA_TAG_FUNC")
DATA_STR = Replace(DATA_STR, "RCHGetTableCell",
"RETRIEVE_WEB_DATA_CELL_FUNC")
DATA_STR = Replace(DATA_STR, "smfStrExtr", "EXTRACT_WEB_DATA_STRING_FUNC")
```

Following that, the function checks if the line contains the information about the element and is not a comment by checking if the first character of the string is not a "`'`":

```
If TEMP_STR <> "'" Then
```

Next, the function takes first n number of characters of string up to `PUB_WEB_DATA_ELEMENT_DELIM_STR` = ";", which will be the number of the element:

```
k = InStr(1, DATA_STR, PUB_WEB_DATA_ELEMENT_DELIM_STR)
TEMP_STR = Left(DATA_STR, k - 1)
```

In the next step, the function uses element number (`TEMP_STR`) as the key and checks whether such an entry already exists in the hash table. If it does the function removes it:

```
If PUB_WEB_DATA_RECORDS_HASH.Exists(TEMP_STR) = True Then
PUB_WEB_DATA_RECORDS_HASH.Remove (TEMP_STR)
```

Finally, the function adds the line to the hash table using `TEMP_STR` (element number) as key and the rest of the line (`Mid(DATA_STR, k + 1)`) as value:

```
PUB_WEB_DATA_RECORDS_HASH.Add TEMP_STR, Mid(DATA_STR, k + 1)
```

The process is repeated for each page (i-loop) https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i: 0 to 9) and for each line on the page (j-loop).

5) RETRIEVE_WEB_DATA_ELEMENT_FUNC

If `PUB_WEB_DATA_TABLES_FLAG` is false, that implies that arrays/hash tables that are used to store data haven't been set yet, so START_WEB_DATA_SYSTEM_FUNC is executed.

The purpose of "If ELEMENT_VAL > PUB_WEB_DATA_RECORDS_VAL Then" is to check whether the element number (`ELEMENT_VAL`), which was input by the user, corresponds to an existing element in the 0 – 9 .txt files. If user inputs `ELEMENT_VAL` higher than 20000, which implies that element user is requesting doesn't exist, the function will return "EOL"

Then, the key for the hash table for a particular user input is created:

```
KEY_STR = TICKER0_STR & "|" & ELEMENT_VAL
```

The key consists of the ticker, for example MSFT and the element number, for example 617 (stands for Earnings Growth Rates - Past 5 Years sourced from Yahoo). The two strings are separated by "|". This key is structured so that it is unique for every user input. For every possible combination of 3 pieces of information: ticker, type of metric and source (later two are embedded in the element number), there is only correct value exists. This ensures that there are now duplicate keys in the hash table.

The following code checks whether an entry for a particular key already exists in the hash table:

```
If PUB_WEB_DATA_ELEMENTS_HASH.Exists(KEY_STR) = True Then
```

If it is true, meaning that the key already exists, the data corresponding to the key is simply returned from the hash table rather than being downloaded from the Internet. It is done through the following code:

```
RESULT_VAL = CONVERT_STRING_NUMBER_FUNC(PUB_WEB_DATA_ELEMENTS_HASH(KEY_STR))
RETRIEVE_WEB_DATA_ELEMENT_FUNC = RESULT_VAL
Exit Function
```

For:

```
PARAM_RNG = PUB_WEB_DATA_RECORDS_HASH(CStr(ELEMENT_VAL))
If PARAM_RNG = "" Then
RETRIEVE_WEB_DATA_ELEMENT_FUNC = "N/A" 'Undefined
```

```
Exit Function
```

First, `ELEMENT_VAL` is being converted from integer to a string. Next, a string that contains all the information about a particular element is assigned to `PARAM_RNG`. For example, if user runs the function as `RETRIEVE_WEB_DATA_ELEMENT_FUNC`("MSFT", 3500) or Diluted Weighted Average Shares - FY1 from Google, the following string is assigned to `PARAM_RNG`:

```
PARAM_RNG = 3500;Google;Annual Income Statement -- Diluted Weighted Average
Shares --
FY1;http://www.google.com/finance?fstype=ii&q=~~~~~;1;INCANNUALDIV;Diluted
Weighted Average Shares; ; ;0;</TABLE;0;0
```

If the element number, which was input by the user, corresponds to element that doesn't exist, for example 17007, "N/A" is returned.

For:

```
PARAM_RNG = Split(PARAM_RNG & CASES_STR, PUB_WEB_DATA_ELEMENT_DELIM_STR)
```

This code merges `PARAM_RNG` with `CASES_STR = ";N/A;N/A;N/A;N/A;N/A;N/A;N/A;N/A;N/A;N/A"`.

Next, the combined string is split in a number of sub strings using the `PUB_WEB_DATA_ELEMENT_DELIM_STR = ";"` as a delimiter. The string is parsed into a number of substring, using ";" as a starting point of each substring. Therefore element number is not included in the set of substrings, so `PARAM_RNG(0)` contains information about the source. Inclusion of element number is unnecessary and redundant in this case, because it is already being passed as `ELEMENT_VAL`.

In our example of `RETRIEVE_WEB_DATA_ELEMENT_FUNC("MSFT", 3500)`, this will result into the following strings assigned to `PARAM_RNG(i)`:

```
PARAM_RNG(0) = "Google"
PARAM_RNG(1) = "Annual Income Statement -- Diluted Weighted Average Shares --
FY1"
PARAM_RNG(2) = "http://www.google.com/finance?fstype=ii&q=~~~~~"
PARAM_RNG(3) = "1"
PARAM_RNG(4) = "INCANNUALDIV"
PARAM_RNG(5) = "Diluted Weighted Average Shares"
PARAM_RNG(6) = " "
PARAM_RNG(7) = " "
PARAM_RNG(8) = "0"
PARAM_RNG(9) = "</TABLE"
PARAM_RNG(10) ="0"
PARAM_RNG(11) = "0"
```

```
PARAM_RNG(12) = "N/A"
PARAM_RNG(13) = "N/A"
PARAM_RNG(14) = "N/A"
PARAM_RNG(15) = "N/A"
PARAM_RNG(16) = "N/A"
PARAM_RNG(17) = "N/A"
PARAM_RNG(18) = "N/A"
PARAM_RNG(19) = "N/A"
PARAM_RNG(20) = "N/A"
PARAM_RNG(21) = "N/A"
```

The `EVALUATE_LINE` sub serves a number of purposes. First, it is used to check whether user requests information about a property of a particular element. For example - `RETRIEVE_WEB_DATA_ELEMENT_FUNC`("P-URL", 3500), or user is requesting the link that is used to extract data for element 3500. In this case the following code will be executed:

```
Case TICKER1_STR = "P-URL"
RESULT_VAL = PARAM_RNG(2)
```

Since user is requesting information that is already stored in `PARAM_RNG` array, it is simply returned to user. Therefore `RETRIEVE_WEB_DATA_ELEMENT_FUNC`("P-URL", 3500), will return "http://www.google.com/finance?fstype=ii&q=~~~~~". Following the exact same process user can request any piece of information that is contained within `PARAM_RNG` array. This done through the execution of the following code depending on the type of information user is requesting (Source, P-Find1 –search string, P-Cells – number of cells to skip in a table, when parsing, etc.):

```
Select Case True
    Case TICKER1_STR = "SOURCE"
        RESULT_VAL = PARAM_RNG(0)
    Case TICKER1_STR = "ELEMENT"
        RESULT_VAL = PARAM_RNG(1)
    Case TICKER1_STR = "WEB PAGE"
        RESULT_VAL = PARAM_RNG(2)
    Case TICKER1_STR = "P-URL"
        RESULT_VAL = PARAM_RNG(2)
    Case TICKER1_STR = "P-CELLS"
        RESULT_VAL = PARAM_RNG(3)
    Case TICKER1_STR = "P-FIND1"
        RESULT_VAL = PARAM_RNG(4)
    Case TICKER1_STR = "P-FIND2"
        RESULT_VAL = PARAM_RNG(5)
    Case TICKER1_STR = "P-FIND3"
        RESULT_VAL = PARAM_RNG(6)
    Case TICKER1_STR = "P-FIND4"
        RESULT_VAL = PARAM_RNG(7)
    Case TICKER1_STR = "P-ROWS"
        RESULT_VAL = PARAM_RNG(8)
    Case TICKER1_STR = "P-END"
```

```
        RESULT_VAL = PARAM_RNG(9)
    Case TICKER1_STR = "P-LOOK"
        RESULT_VAL = PARAM_RNG(10)
    Case TICKER1_STR = "P-TYPE"
        RESULT_VAL = PARAM_RNG(11)
```

Next, `EVALUATE_LINE` is used to make the adjustments necessary to parse data from advfn.com:

```
Case UCase(PARAM_RNG(0)) = "ADVFN-A"
        RESULT_VAL = PARSE_ADVFN_WEB_DATA_ELEMENT_FUNC(TICKER1_STR, "A",
PARAM_RNG(3), PARAM_RNG(4), PARAM_RNG(5), ERROR_STR)
    Case UCase(PARAM_RNG(0)) = "ADVFN-Q"        RESULT_VAL =
PARSE_ADVFN_WEB_DATA_ELEMENT_FUNC(TICKER1_STR, "Q", PARAM_RNG(3),
PARAM_RNG(4), PARAM_RNG(5), ERROR_STR)
```

For:

```
If RESULT_VAL <> "" Then: GoTo 1983
```

If any condition in `EVALUATE_LINE` was satisfied (e.g. user was requesting a property of an element or requesting data from advfn.com), then a corresponding value was assigned to `RESULT_VAL`, so it is not null. Therefore, 1983 is executed.

```
1983:
If RESULT_VAL = ERROR_STR Then: GoTo ERROR_LABEL
Call PUB_WEB_DATA_ELEMENTS_HASH.Add(KEY_STR, RESULT_VAL)
RESULT_VAL = CONVERT_STRING_NUMBER_FUNC(RESULT_VAL)
RETRIEVE_WEB_DATA_ELEMENT_FUNC = RESULT_VAL
Exit Function
```

1983 does the following: first, it checks whether a value other than “Error” was assigned to `RESULT_VAL`. If it is not “Error”, then the key that was defined and assigned to `KEY_STR` and the corresponding value that is contained in `RESULT_VAL` are recorded into a hash table. Finally, `RESULT_VAL` is returned as output of the function.

For:

```
TICKER2_STR = CONVERT_YAHOO_TICKER_FUNC(TICKER1_STR, PARAM_RNG(0))
```

This code is used to make necessary adjustments to the string that contains the ticker, if user specifies the stock exchange, the data from which he wants to get. For example, `RETRIEVE_WEB_DATA_ELEMENT_FUNC("BB.TO ", 3500)` – in this case user requests information on RIM from Toronto Stock Exchange sourced from Google. The problem with this is that different web-site use different standards to indicate the stock exchange. So ticker for RIM traded on Toronto Stock Exchange will look differently on different web-sites. For example:

Google: TSE:BB
MorningStar: XTSE:BB
MSN: CA:BB

`CONVERT_YAHOO_TICKER_FUNC` is used to convert Yahoo standard ticker to the standard used by the other web-site. First, the function identifies information from which source is requested by the user. This is done through the following code:

```
Select Case True
   Case VERSION = 0 Or Left(UCase(VERSION), 5) = "ADVFN": GoTo ADVFN_LINE
   Case VERSION = 1 Or Left(UCase(VERSION), 8) = "BARCHART": GoTo
BARCHART_LINE
   Case VERSION = 2 Or Left(UCase(VERSION), 8) = "EARNINGS": GoTo
EARNINGS_LINE
   Case VERSION = 3 Or Left(UCase(VERSION), 3) = "MSN": GoTo MSN_LINE
   Case VERSION = 4 Or Left(UCase(VERSION), 6) = "GOOGLE": GoTo GOOGLE_LINE
   Case VERSION = 5 Or Left(UCase(VERSION), 11) = "MORNINGSTAR": GoTo
MORNINGSTAR_LINE
   Case VERSION = 6 Or Left(UCase(VERSION), 7) = "REUTERS": GoTo REUTERS_LINE
   Case VERSION = 7 Or Left(UCase(VERSION), 11) = "STOCKCHARTS": GoTo
STOCKCHARTS_LINE
   Case VERSION = 8 Or Left(UCase(VERSION), 10) = "STOCKHOUSE": GoTo
STOCKHOUSE_LINE
   Case VERSION = 9 Or Left(UCase(VERSION), 5) = "ZACKS": GoTo ZACKS_LINE
   End Select
GoTo 1983
```

Next the function goes to the line of the web-site, information from which is being requested, and makes the necessary conversion. Converted ticker is used as the output. So, for `RETRIEVE_WEB_DATA_ELEMENT_FUNC("BB.TO ", 3500)`, the output of `CONVERT_YAHOO_TICKER_FUNC` will be "TSE:BB".

```
SRC_URL_STR = Replace(PARAM_RNG(2), PUB_WEB_DATA_ELEMENT_LOOK_STR,
TICKER2_STR)
```

This code replaces the initial ticker – `TICKER1_STR` with converted ticker - `TICKER2_STR` in the `PARAM_RNG` array.

If the element requested doesn't require any additional calculations and hasn't been requested and saved yet, then it is being downloaded and saved using `SAVE_WEB_DATA_PAGE_FUNC`

```
If PARAM_RNG(0) <> "Calculated" And
PUB_WEB_DATA_PAGES_HASH.Exists(SRC_URL_STR) = False Then
```

```
    DATA_STR = SAVE_WEB_DATA_PAGE_FUNC(SRC_URL_STR, PARAM_RNG(11), True, 0, 
False)
```

The XML/HTML is then saved into a hash table using url as the key:

```
Call PUB_WEB_DATA_PAGES_HASH.Add(SRC_URL_STR, DATA_STR)
```

For the `GoSub PARSE_LINE:`

The purpose of `PARSE_LINE` is to determine which parsing algorithm should be used for a particular element: `PARSE_WEB_DATA_FRAME_FUNC`: or `PARSE_WEB_DATA_CELL_FUNC`. If the 2nd substring (url) or the 3rd substring for a particular element starts with "?", then `PARSE_WEB_DATA_FRAME_FUNC` is used. If the element is obsolete (no longer available), then none of the algorithms is used. In all the other cases `PARSE_WEB_DATA_CELL_FUNC` is used.

```
If RESULT_VAL = ERROR_STR Then: GoTo ERROR_LABEL
Call PUB_WEB_DATA_ELEMENTS_HASH.Add(KEY_STR, RESULT_VAL)
```

If the result value returned from `EVALUATE_LINE` is not equal to "Error", then the result value is recorded into a hash table using `TICKER0_STR|ELEMENT_VAL` as a key.

```
RESULT_VAL = CONVERT_STRING_NUMBER_FUNC(RESULT_VAL)
```

The purpose of `CONVERT_STRING_NUMBER_FUNC` is to convert the `RESULT_VAL` (the piece of data requested by the user) to the conventional format. For example, if data is expressed as $10M, it will be converted to 10,000,000; if it is expressed as 10 Jan. 2013, it will converted to 10/01/2013; if it is expressed as 10%, it will converted to 0.1 etc.

The data from different sources might be expressed using different standards, so the purpose of this function is to allow user to use the data in his models right away without doing any additional conversions.

```
Case Left(PARAM_RNG(2), 1) = "="
RESULT_VAL = Evaluate(Replace(Mid(PARAM_RNG(2), 2), 
PUB_WEB_DATA_ELEMENT_LOOK_STR, TICKER1_STR))
```

This code applies to the elements, which require additional processing using other functions, for example element 542:

```
MSN;StockScouter Rating --
Summary;=SUBSTITUTE(SUBSTITUTE(smfGetTagContent("http://investing.money.msn.c
om/investments/stock-ratings?symbol=~~~~~","p",1,"Stock Rating 
Summary"),"<b>",""),"</b>","")
```

For such elements url substrings start with "=" and look the following way:
`"=FUNCTION(url)"`

Since all the .txt files (https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i:0 to 9)) were taken from Randy Harmelink's add-in, such urls include his functions. Therefore it is necessary to replace such functions names with different (Nico's) function names:

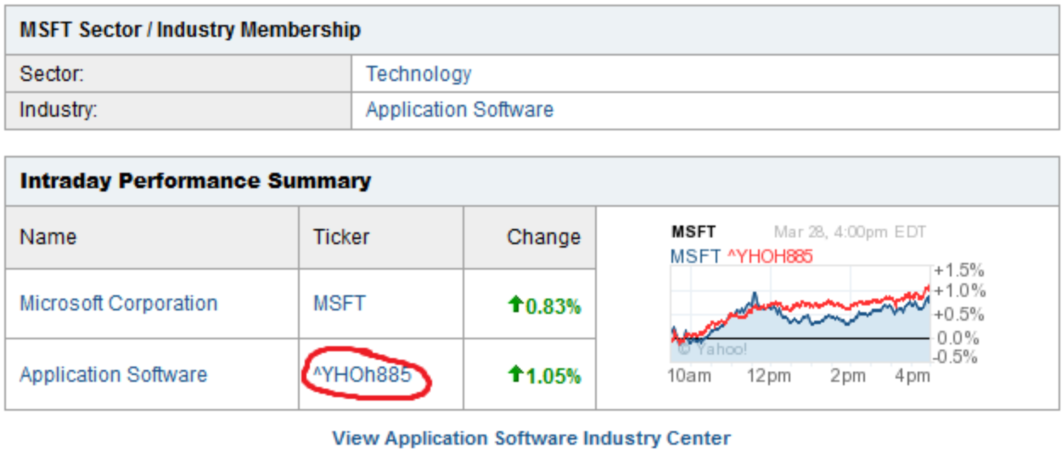
```
DATA_STR = Replace(DATA_STR, "smfGetTagContent", "PARSE_WEB_DATA_TAG_FUNC")
DATA_STR = Replace(DATA_STR, "RCHGetTableCell",
"RETRIEVE_WEB_DATA_CELL_FUNC")
DATA_STR = Replace(DATA_STR, "smfStrExtr", "EXTRACT_WEB_DATA_STRING_FUNC")
```

6) PARSE_WEB_DATA_FRAME_FUNC

`PARSE_WEB_DATA_FRAME_FUNC` is used, when extraction of the data could be done by using a much more efficient algorithm then the one that is used in `PARSE_WEB_DATA_CELL_FUNC`. For example, element 13868:

```
YahooIN;Industry Symbol;http://finance.yahoo.com/q/in?s=~~~~~;?;;;;;;;;0
```

This element is located in the table, so `PARSE_WEB_DATA_CELL_FUNC` could be used to extract the data:

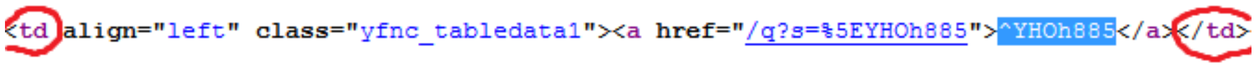


MSFT Sector / Industry Membership	
Sector:	Technology
Industry:	Application Software

Intraday Performance Summary

Name	Ticker	Change
Microsoft Corporation	MSFT	⬆0.83%
Application Software	^YHOh885	⬆1.05%

View Application Software Industry Center



```
<td align="left" class="yfnc_tabledata1"><a href="/q?s=%5EYHOh885">^YHOh885</a></td>
```

However, using `PARSE_WEB_DATA_CELL_FUNC` is very inefficient in this case. The required piece of data – “^YHOh885” starts with “>^” combination of characters. This combination of characters is unique for the whole XML/HTML code. Therefore, it is much faster to locate the position of “>^” and then take 8 characters starting from the position of “^”, rather than going through the whole `PARSE_WEB_DATA_CELL_FUNC` algorithm.

So element 13868 has “?” instead of its 3rd substring, therefore `PARSE_WEB_DATA_FRAME_FUNC` is being used. The extraction of the data is done through the following code:

```
Case "Industry Symbol" ' Yahoo
     i = InStr(DATA2_STR, ">^")
     If i = 0 Then GoTo ERROR_LABEL
     PARSE_WEB_DATA_FRAME_FUNC = Mid(DATA1_STR, i + 1, 8)
```

7) PARSE_WEB_DATA_CELL_FUNC

The purpose of `PARSE_WEB_DATA_CELL_FUNC` function is to find the requested piece of data in XML/HTTP code once it has been saved as a string.

Inputs:

`DATA1_STR` = a string that contains the whole XML/HTTP code of the web-page. The search is performed inside this string. `DATA1_STR` is set to Upper Case and saved as `DATA2_STR`.

`FIND1_STR` = a string to locate inside `DATA2_STR`. Its position is saved under ii variable.

`FIND2_STR` = a string to locate inside `DATA2_STR` starting with the position of `FIND1_STR`. Its position is saved under ii variable.

`FIND3_STR` = a string to locate inside `DATA2_STR` starting with the position of `FIND2_STR`. Its position is saved under ii variable.

`FIND4_STR` = a string to locate inside `DATA2_STR` starting with the position of `FIND3_STR`. If `FIND4_STR` contains “|”, then it is split into substrings using “|” as a delimiter. The position of the first substrings that is found in `DATA2_STR` is saved under ii.

`FIND1_STR – FIND4_STR` are used as a “search pattern’. The purpose of it is to find a unique combination of characters in `DATA2_STR` that are located near the required piece of data. In most instances there is no need to specify all the four `FIND_STR`. For example, if there is a combination of characters or tag that is located near the required piece of data and is unique (appears only once in `DATA2_STR`), then `FIND1_STR` should be equal to this tag or a combination of characters, and the rest of the `FIND_STR` should be left blank. If there is no such unique tag or a combination of characters for the whole `DATA2_STR`, then such a combination of characters is found that will be unique for the substring of `DATA2_STR` that starts from `FIND1_STR`. This combination of characters is assigned to `FIND2_STR`. Similar logic applies to `FIND3_STR` and `FIND4_STR`.

`NROWS` = Indicates in which row in a table in a web-page the requested piece of data is located. In the parsing algorithm is number of rows to skip forward (if `NROWS` > 0) or backward (if `NROWS` < 0) from the position of `FIND_STR`.

`END_STR` is a syting that indicates the end of skipping forward/backward based on `NROWS` and `CELL_INT`. Usually set to "</TABLE" to make sure that the pointer doesn't leave the table that includes the required piece of data.

`CELLS_INT` = Number of cells in a table in a particular row in a web-page to skip before extracting the date. If `CELLS_INT > 0`, then it is the number of cells to skip starting from the beginning of the row, if `CELLS_INT < 0`, then it is the number of cells to skip starting from the end of the row and going backwards and if `CELLS_INT = 0`, the date is to be extracted from the first cell in the row.

`LOOK_INT` = Number of consecutive cells to search for data in.

In order to illustrate the parsing algorithm, element#1273 for Microsoft stock was used as an example. Element 1273 stands for Historic Volatility 20-day sourced from http://www.barchart.com. URL for element 1273 for MSFT stock is http://www.barchart.com/technicals/stocks/MSFT.

Inputs for this example will be:

```
DATA1_STR = XML/HTML code of http://www.barchart.com/technicals/stocks/MSFT
FIND1_STR = ">PERIOD"
FIND2_STR = ">PERIOD"
FIND3_STR = ">PERIOD"
FIND4_STR = ">"
NO_ROWS = 3
END_STR = "</TABLE"
CELLS_INT = 4
LOOK_INT = 0
```

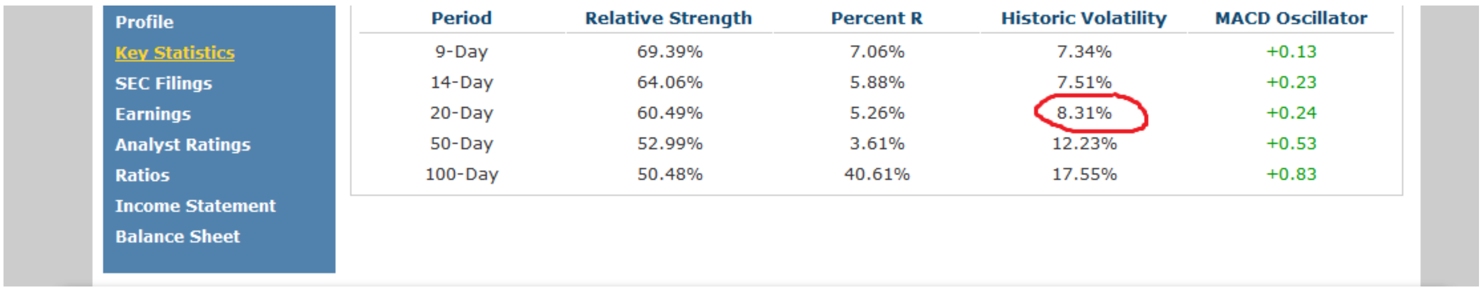
First, the function locates the position of `FIND1_STR` in `DATA2_STR` (`DATA2_STR = DATA_STR1` set to Upper Case) and assigns it to ii. In this case this will be ii = 72709. Next, the function locates the position of `FIND2_STR` in `DATA2_STR` starting from ii (position of `FIND1_STR`) and assigns it to ii. Next, the function locates the position of `FIND3_STR` in `DATA2_STR` starting from ii (position of `FIND2_STR`) and assigns it to ii. Next, the function locates the position of `FIND4_STR` in `DATA2_STR` starting from ii (position of `FIND3_STR`) and assigns it to ii.

```
DATA2_STR = UCase(DATA1_STR)
ii = 0
ii = InStr(ii + 1, DATA2_STR, UCase(FIND1_STR))
If ii = 0 Then GoTo ERROR_LABEL
If FIND2_STR > " " Then
    ii = InStr(ii + 1, DATA2_STR, UCase(FIND2_STR))
    If ii = 0 Then GoTo ERROR_LABEL
End If
If FIND3_STR > " " Then
    ii = InStr(ii + 1, DATA2_STR, UCase(FIND3_STR))
    If ii = 0 Then GoTo ERROR_LABEL
End If
If FIND4_STR > " " Then
    TEMP_ARR = Split(UCase(FIND4_STR), "|")
    h = UBound(TEMP_ARR, 1)
    For l = 0 To h
        jj = InStr(ii + 1, DATA2_STR, TEMP_ARR(l))
        If jj > 0 Then Exit For
        If l = UBound(TEMP_ARR, 1) Then GoTo ERROR_LABEL
    Next l
    ii = jj
End If
```

In our example, at this stage ii = 76547 (position of `FIND4_STR`).

Generally speaking, the purpose of using `FIND1_STR - FIND4_STR` is to locate the table that contains the required piece of data.

As shown on the picture (taken from http://www.barchart.com/technicals/stocks/MSFT), the requested piece of data is located in the 3rd row 4th column of the table. That is why `NROW = 3` and `CELLS_INT = 4`.



Period	Relative Strength	Percent R	Historic Volatility	MACD Oscillator
9-Day	69.39%	7.06%	7.34%	+0.13
14-Day	64.06%	5.88%	7.51%	+0.23
20-Day	60.49%	5.26%	8.31%	+0.24
50-Day	52.99%	3.61%	12.23%	+0.53
100-Day	50.48%	40.61%	17.55%	+0.83

In XML/HTML code the beginning and the end of a table that contains the date is defined using <TABLE> and </TABLE> tags; the beginning and the end of a row in the table is defined using <TR> and </TR> tags.

In our example `NROWS = 3` at it is a positive number. This means that the function will loop through `DATA2_STR 3` times locating the positions of <TR> and </TR> in `DATA2_STR` starting from ii (position of `FIND4_STR`). The function stops at the 3rd row that contains the required piece of data. The position of <TR> (beginning of the row) for the 3rd row is assigned to ii, and the position of </TR> (end of the row) for the 3rd row is assigned to kk.

```
Select Case True
Case NO_ROWS > 0
    jj = InStr(ii, DATA2_STR, UCase(END_STR))
    For l = 1 To NO_ROWS
        ii = InStr(ii + 1, DATA2_STR, "<TR")
        kk = InStr(ii, DATA2_STR, "</TR")
        If kk > jj Then: GoTo ERROR_LABEL
    Next l
```

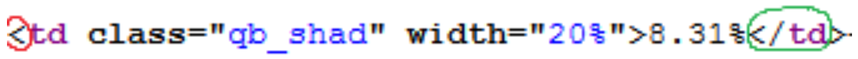
In our example ii = 77438, kk = 77688.

In XML/HTML the beginning and the end of a cell in a table is defined using <TD> and </TD> tags.

Next, the function loops the through the row `k + LOOK_INT` times (whether `k = CELLS_INT`, if `CELLS_INT > 0` or `k = - CELLS_INT`, if `CELLS_INT < 0`), each time assigning the position of <TD> (beginning of a cell) to jj. Then jj is set to the position of “>”, which is where the tag ends and the value that is contained within the cell starts. Position of </TD> (end of a cell) is assigned to kk.

```
For l = 1 To k + LOOK_INT
    If CELLS_INT > 0 Then
        jj = InStr(jj, DATA2_STR, "<TD")
    Else
        jj = InStrRev(DATA2_STR, "<TD", jj)
    End If
    If jj = 0 Or jj < i Or jj > j Then GoTo ERROR_LABEL
    jj = InStr(jj, DATA2_STR, ">")
    If l >= k Then
        kk = InStr(jj, DATA2_STR, "</TD")
```

In our example the function loops the row 4 times (`CELLS_INT = 4`, `LOOK_INT = 0`), going through the cells, each time reassigning the positions of beginning (<TD>) and ending of the cell (</TD>) to jj and ii. The function stops at the 4th cell, which contains the required piece of data. jj = 77629, which is the position of beginning of the cell = “>” (red circle) and kk = 77635, which is the position of the end of the cell = “</TD” (green circle).



```
<td class="qb_shad" width="20%">8.31%</td>
```

Finally, the data is extracted from the cell:

```
TEMP_VAL = Trim(Mid(DATA1_STR, jj + 1, kk - jj - 1))
```

If the required piece of data consists of multiple lines (<br> = go to new line), the tag is replaced with new line character (`Chr(10)`):

```
TEMP_VAL = Replace(Trim(TEMP_VAL), "<br>", Chr(10))
```

End result:
`RETRIEVE_WEB_DATA_ELEMENT_FUNC("Msft", 1273) = 0.0831`

8) RETRIEVE_WEB_DATA_RECORDS_FUNC

The purpose of this function is to retrieve the information about each element (source, url, search pattern strings, etc.) from https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i: 0 to 9).

Inputs:

`INDEX_RNG` = one or multiple element numbers (1 to 17006)

`ERROR_STR` = string to return in case of error

First, the function takes `INDEX_RNG` and converts it to one column – multiple rows matrix and assigns its values to `INDEX_VECTOR`:

```
If IsArray(INDEX_RNG) = True Then
    INDEX_VECTOR = INDEX_RNG
    If UBound(INDEX_VECTOR, 1) = 1 Then: _
    INDEX_VECTOR = MATRIX_TRANSPOSE_FUNC(INDEX_VECTOR)
Else
    ReDim INDEX_VECTOR(1 To 1, 1 To 1)
    INDEX_VECTOR(1, 1) = INDEX_RNG
End If
NROWS = UBound(INDEX_VECTOR, 1)
```

Next, the heading string, which includes titles for each property of an element are written to the 0 row of `TEMP_MATRIX`:

```
NCOLUMNS = 14
```

```
HEADINGS_STR = "ID,VERSION,SOURCE,ELEMENT,P-URL,P-CELLS,P-FIND1,P-FIND2,P-
FIND3,P-FIND4,P-ROWS,P-END,P-LOOK,P-TYPE,"
ReDim TEMP_MATRIX(0 To NROWS, 1 To NCOLUMNS)
i = 1
For k = 1 To NCOLUMNS
    j = InStr(i, HEADINGS_STR, ",")
    TEMP_MATRIX(0, k) = Mid(HEADINGS_STR, i, j - i)
    i = j + 1
Next k
```

Following that, the function loops through each element (number of loops is equal to the number of element in `INDEX_RNG`). Element number is assigned to the first cell in each row of `TEMP_MATRIX`. Next, for each element the function loops 13 times calling `RETRIEVE_WEB_DATA_ELEMENT_FUNC` and using title names from the heading strings as tickers in this case and k = element number.

```
For i = 1 To NROWS
    k = INDEX_VECTOR(i, 1)
    TEMP_MATRIX(i, 1) = k
    For j = 2 To NCOLUMNS: TEMP_MATRIX(i, j) =
RETRIEVE_WEB_DATA_ELEMENT_FUNC(TEMP_MATRIX(0, j), k, ERROR_STR): Next j
Next i
```

Once `RETRIEVE_WEB_DATA_ELEMENT_FUNC` is called, one of the following lines will be executed and corresponding value will be returned:

```
Select Case True
Case TICKER1_STR = "SOURCE"
    RESULT_VAL = PARAM_RNG(0)
Case TICKER1_STR = "ELEMENT"
    RESULT_VAL = PARAM_RNG(1)
Case TICKER1_STR = "WEB PAGE"
    RESULT_VAL = PARAM_RNG(2)
Case TICKER1_STR = "P-URL"
    RESULT_VAL = PARAM_RNG(2)
Case TICKER1_STR = "P-CELLS"
    RESULT_VAL = PARAM_RNG(3)
Case TICKER1_STR = "P-FIND1"
    RESULT_VAL = PARAM_RNG(4)
Case TICKER1_STR = "P-FIND2"
    RESULT_VAL = PARAM_RNG(5)
Case TICKER1_STR = "P-FIND3"
    RESULT_VAL = PARAM_RNG(6)
Case TICKER1_STR = "P-FIND4"
    RESULT_VAL = PARAM_RNG(7)
Case TICKER1_STR = "P-ROWS"
    RESULT_VAL = PARAM_RNG(8)
Case TICKER1_STR = "P-END"
    RESULT_VAL = PARAM_RNG(9)
Case TICKER1_STR = "P-LOOK"
    RESULT_VAL = PARAM_RNG(10)
Case TICKER1_STR = "P-TYPE"
    RESULT_VAL = PARAM_RNG(11)
```

RETRIEVE_WEB_DATA_RECORDS_FUNC returns a matrix, which consist of 14 columns and number of rows that is equal to number of elements in INDEX_RNG + 1 row for the headings. Each row contains information about each element from https://raw.github.com/rnfermincota/BGCVI/master/WDS/smf-elements/smf-elements-i.txt (i: 0 to 9), that is split into substrings and each substring is recorded in a column with its corresponding title.

9) RESET_WEB_DATA_SYSTEM_FUNC

RESET_WEB_DATA_SYSTEM_FUNC is used if user wishes to remove all the data that is stored in the hash tables or arrays. The user may want to do that if he is reaching the maximum number of values that could be stored in a hash table/array or if he wants to update the data, e.g. download more recent XML/HTTP code of the web-pages.

First the function checks whether or not there is an active internet connection by calling XML_CHECK_HTTP_CONNECTION_FUNC:

```
If XML_CHECK_HTTP_CONNECTION_FUNC() = True Then
```

XML_CHECK_HTTP_CONNECTION_FUNC sends a request to google and if there is no error, then it means that the internet connection is active (exists):

```
Dim CONNECT_OBJ As MSXML2.XMLHTTP60
With CONNECT_OBJ
    .Open "GET", "http://www.google.com"
    .setRequestHeader "Content-Type", "application/x-www-form-urlencoded"
    .send MSG_STR
End With
```

If the connection is active, then the function calls REMOVE_CACHE_HISTORY_FUNC and START_WEB_DATA_SYSTEM_FUNC. REMOVE_CACHE_HISTORY_FUNC calls REMOVE_WEB_PAGE_CACHE_FUNC, which in turn nukes all the hash table and array variables. Then START_WEB_DATA_SYSTEM_FUNC creates them up again sets to null.

Finally, the function compares the version of Microsoft Excel with the version of Excel in which the workbook was last calculated. If the version is different, then a full recalculation of the data is performed and dependencies are rebuilt if necessary:

```
If Val(Excel.Application.VERSION) < 10 Then
    Excel.Application.CalculateFull
```

```
Else
    Excel.Application.CalculateFullRebuild
End If
```