ONLINE USER DOCUMENTATION

Contents

1. General Information

1.1 Procedure to create HTML Code

In our project, the procedure to create elements for the HTML code is the same for every object (except for the anchors and internal links, see further details in « Internal Links »). For this brief example, a text will be initialised and added to a list which will be the input for the html_code_generator_visitor. This all happens within the application class.

Step 1: Set variable the object.

Outside of the make function, the text needs to be set as follows:

```
text1 : TEXT
```

Step 2: Create the object

Now, inside the make function, the text will be created with the help oft he already created HTML_Factory called «html» as follows:

```
text1 := html.text(«example_text»)
```

The inputs for the creation of the files can be looked up in «2. Create Objects».

Step 3: Add object to the list

If there is only one HTML document that the user wants to create, the object needs to get added to the list that is already created.

```
liste.extend(text1)
```

Attention:
- If the user wants to create several HTML files in the application.e, section 1.3 will cover how to do that.
-The only acceptable inputs for the extension of the list are created HTML objects, not strings or other elements that were not created with our HTML_Factory.
-If the user added an element to a component(list, table or paragraph), it no longer needs to be added to the list as it will be added through the component.

Final step when all necessary objects are added to the list: Get the code from the elements in the list and store them in a string:

```
string := cgv.get_string (list)
```

With the function get_string(list), the html_code_generator visitor will visit all the objects in the list and return the whole string to «string». If the user wants to see his result, he can just print the string.

1.2 Application Class without any user input :

When the application.e file is opened for the first time, several objects will already be created for the user since he will always use those. A quick explanation should give the user a good overview :

cgv :	Object of Class HTML_CODE_GENERATOR VISITOR. Will only have to be used once for every html document when the user acesses to the get_string function to store the code string into a string object.
list :	Object of Class LINKED_LIST[COMPONENT]. Will store the all the created objects used for one HTML_DOCUMENT.
string :	Object of Class STRING. Stores the code it gets from the HTML_CODE_GENERATOR_VISITOR.
html :	Object of Class HTML_FACTORY. Is responsible for creating the HTML_COMPONENTS.
anchor_manager :	Object of Class HTML_ANCHOR_MANAGER created by the factory. Can create several anchors with a different id each.

1.3 How to make several .html-strings in one application.e

To create more than one file_string, all that needs tob e done is to initialize a new linked list and a new string just like the list and string that is already provided. In the list, you can add all the other objects that you want in a separate html_file and in the string the contents of the file will be stored in there. At the end, the code of the html_components in list2 are going to be stored in string2 like this:

```
string2 := cgv.get_string(list2)
```

2. Create Objects

2.1 Anchor

Class Description.

Will add an invisible anchor in the document which will be used to make jumps through an internal link to the part where the anchor is set.

Input Data :

id : STRING (user will not have to type the id, the anchor _manager will take care of that)

Features :

add_link(a_text : STRING) will return an internal link to the anchor with the same id and create an internallink just like the factory would create an object. Every link created by one anchor will have the same id as the anchor.

Set the Anchor:

anchor : ANCHOR

Create HTML Anchor:

anchor := anchor_manager.create_anchor

2.2 Anchor Manager

Class Description.

Is responsible to create anchors and distribute unique IDs to them.

Input Data :

None.

Features :

create_anchor : will return an Anchor and will create an Anchor just like the factory would create an object. With every new anchor, a new id will be assigned to it.

Set the Anchor Manager :

anchor_manager : ANCHOR_MANAGER

Create HTML Anchor Manager :

anchor_manager := html.anchor_manager

ATTENTION:

The user is not allowed to create another instance of the ANCHOR_MANAGER as it would give the same ids to different anchors.

2.3 External Link

Class Description.

Will add a link to an url or another html.file.

Input Data :

a_source : STRING; a_text : STRING

Features :

None

Set the External Link :

link : LINK

Create HTML Link:

link := html.link ("www.a_website.ch", "This will be the text for the link.")

2.4 Image

Class Description.

Will add an image for the document.

Input Data :

s : STRING; alt_name : STRING

Features :

None

Set the Image :

image: IMAGE

Create HTML Image :

image := html.image ("This_path", "alt")

2.5 Ordered List

Class Description.

Will add an ordered list with HTML Components as elements. Will be initialized empty, created objects will be added through add_row(element : COMPONENT)

Input Data :

None

Features :

add_row(element : COMPONENT) : adds element to the ordered list.

Set the Ordered List :

ol : LISTS

Create HTML Ordered List :

ol := html.ordered_list

Add new row:

ol.add_row (link)

2.6 Paragraph

Class Description.

Will add an Paragraph with HTML Components as elements. Will be initialized empty, created objects will be added through new_element(element : COMPONENT)

Input Data :

None

Features :

new_element(element :COMPONENT): adds element to the paragraph

Set the Pararaph :

p : PARAGRAPH

Create HTML Paragraph:

p := html.paragraph

Add new element:

p.new_element (link)

ATTENTION:

The user is not allowed to place a paragraph into a paragraph!

2.7 Snippet

Class Description.

Allows the user to add HTML code to the already existing code. Proper indentation is up to the user.

Input Data :

a_string: STRING

Features :

None

Set the Snippet :

snippet : SNIPPET

Create HTML Snippet :

snippet := html.snippet (“This is HTML code”)

2.8 Table

Class Description.

Allows the user to add a table. The table will be empty when initialized but needs as input how many columns the user wants for the table. Through add_row(row_cells:LINKED_LIST[COMPONENT]) new rows can be added as long as the Linked List from add_row has as many elements as set columns.

Input Data :

a_column:INTEGER

Features :

add_row(row_cells:LINKED_LIST[COMPONENT])

Set the Table :

table : TABLES

Create HTML Table :

table := html.table (2)

Add new row:

```
table_row : LINKED_LIST[COMPONENT]
table_row.extend (link)
table_row.extend (link)
table.add_row (table_row)
```

ATTENTION:

HTML Objects that will be put into the row_cells linked list have to be treated first. For example: element.make_element and element.set_tab (0). If those functions are not performed, the indentation will be messed up.

2.9 Text

Class Description.

Will create HTML Component texts. Through the functions bold, underlined and italic, the text can get those attributes.

Input Data :

a_string: STRING

Features :

bold
underlined
italic

Set the Text :

t : TEXT

Create HTML Text:

t := html.text ("Paragraph text, is underlined, bold and italic")

Make text bold :

t.bold

Make text italic :

t.italic

Make text underlined :

t.underlined

2.10 Title

Class Description.

Will create a Title with size 1 to 6, other sizes are restricted.

Input Data :

a_text: TEXT; a_size:INTEGER

Features :

None

Set the Title :

title : TITLE

Create HTML Title:

title := html.title (t, 2)

2.11 Unordered List

Class Description.

Will add an unordered list with HTML Components as elements. Will be initialized empty, created objects will be added through add_row(element : COMPONENT)

Input Data :

None

Features :

add_row(element : COMPONENT) : adds element to the unordered list.

Set the Ordered List :

ul : LISTS

Create HTML Ordered List :

ul := html.unordered_list

Add new row:

ul.add_row (link)

3. Specific Examples.

To not only get to see Snippets, there are are also a few examples that can be inserted where « --set up the objects» and where « --add creation and list extension » is written.

3.1 Table with two rows that are the same

--set up the objects :

```
table_link : LINK
table_row : LINKED_LIST[COMPONENT]
table : TABLES
```

--add creation and list extension :

```
create table_row.make
table_link := html.link ("www.tables.ch", "This link is in a table")
table := html.table (2)

table_link.make_element
table_link.set_tab (0)

table_row.extend (table_link)
table_row.extend (table_link)

table.add_row (table_row)
table.add_row (table_row)

list.extend (table)
```

3.2 How Internal Links work

--set up the objects :

```
am : ANCHOR_MANAGER
an : ANCHOR
al, al2 : LINK
```

--add creation and list extension :

```
am := html.anchor_manager
an := am.create_anchor
al := an.add_link ("al is a Anchor LINK to an")
al2 := an.add_link ("al is another Anchor LINK to an")

list.extend (an)
list.extend (al)
list.extend (al2)
```