Total.HTM

NoCMS, Only HTML

@purpose: **Docu, 101, Intro, Reference, Help, ...**

@author: **Ernesto Sun**

@started: 2017-10-15
@version: 2017-11-18

Table of Contents

Legend

> Paragraphs that have a green line at the left side – like this one – are marked as easy and specially **helpful for novices**.

> The paragraphs with a red line are marked as more complicated. You might need some **developer experience** to understand them.

```
{
// Source Code looks like this.
}
```

1) Intro

I assume you like **HTML** - the language of the web - or you are excited about learning it. Here it is all about **writing clean regular HTML**.

Total.HTM grew from the need to have easy managed static content based on web standards. Todays great websites want to be **modern** – **JavaScript based** - and as well have a complete **fallback to server generated static HTML**. Crawlers and robots appreciate this as well as blind people or agents that rely on any strange Internet reading device.

Good old friendly semantic HTML, like the XHTML 1.1 standard from better ages, are the **automatic result** of using Total.HTM. Little effort to style, little effort to add content, and little trouble to consume the website as final end user.

Total.HTM provides a **No-Headache HTML solution**. Total.HTM is all about static websites but this does not at all deny dynamic web stuff within. Total.HTM gives the developer a simple and clear HTML template to put PHP code, JS code or any dynamic code into it.

Let's look at a typical **workflow** of a website realization **using Total.HTM**:

1) Client, Meeting, Talking about Design, Logical Structures, Content **Strategy**
2) **Multimedia**, Categories and Text. Picture-Selection, Translations, ...
3) Designer creates a vector file or any **sketch** for layout and design
4) Developer modifies **Total.HTM** in HTML and CSS to **fit layout and design**
5) **Static content** is worked into **Total.HTM** by developer or content person
6) Dynamic content is **connected to Total.HTM** by developer, API, AJAX, ...
7) **Testing Total.HTM** in the local browser directly just as a file on the computer
8) The developer uses the **Total.HTM MAKE** process and creates the **RELEASE**.
9) When needed **Total.HTM** is **changed** and the admin invokes MAKE again

The RELEASE generation is done by a short script called **MAKE**. The auto-generated **RELEASE is all the website really needs** to be great in the Internet. The result is both Javascript-based and machine friendly raw HTML served by URL parameters.

The Total.HTM RELEASE includes **SEO-friendly things** like Sitemap and Manifest as well as the usual HTML metadata. Total.HTM supports **multiple languages** and **nested sections**.

Total.HTM is **Open Source** and you may **feel welcome** to contribute to the community. Made for people that like the basic web standards.

I am sure you will understand Total.HTM by reading this document. For this and for your future I wish you **a great time**.

1.1) What to Focus On First

Open the file **total.htm** that is in the root of the web folder you have downloaded. You only need a good text editor and a browser.

Some text editors I like to use are Notepad++ or Atom. Just open the file `/total.htm` in the editor you like. The **HTML should look good** and be colorful (syntax highlighting).

Just look at this HTML code. It might appear quite simple and logical to you. At the top is the head with all the metadata and further below the **content organized in sections**. At the bottom, as usual, is the footer.

> If you do not feel well with this code, if there is no logic appearing to you by looking at it, just focus on the part inside the so called `section-body.` Search for this term. Inside this section-bodies you find just text and images. That's maybe all you need to know about for now.

To work with Total.HTM you need to **feel well with HTML**. There is a ton of resources out there to learn about it.

1.2) What to Focus On Second

See the CSS files in the sub folder `/css`. In there are **four basic CSS files**: `/system.css, /layout.css, /design.css` and `/custom.css` in this exact order. If you are not a tech expert don't mind `/system.css`.

The **Layout CSS file** `/layout.css` might be more interesting. It contains CSS for margins, positions, overlays and such. The separation between `/layout.css` and `/design.css` is part of a CSS management strategy. Font-styles, colors, borders and such are to be found in the **Design CSS file** `/design.css`. Project-related CSS can be put into `/custom.css`. Per default that file is empty.

Now, see the **image folders** `/img` **and** `/image`. The second one is for **website pictures**. The shorter `/img` folder is for **system images and symbols** and other more general image material like language flags. For other **multimedia files** just use the folders `/audio, /video, /pdf` and alike.

Now, you can complete your overview by examining `/js/system.js` if you know some JavaScript. Or you just trust the JavaScript part and focus on layout, design, content and multimedia as well as on attaching your dynamic stuff (API, AJAX, ...).

As you might expect by now: **You are free to choose any additional framework** or custom construction. Total.HTM uses jQuery but can be translated into Vanilla (Raw JavaScript without framework) if wanted. The only JavaScript file needed to make Total.HTM 'dynamic' is `/system.js`.

> Just add **your own JavaScript and CSS files and anything you want**. The web was meant to be free. Free are the good old web standards like HTML5, CSS3 and JavaScript.

2) Lang Lang

Each HTML element can have an **lang attribute** `lang=".."` set to some **language code like 'de'** for German, or 'en' for English. This is part of the HTML standard.

Not part of the HTML standard is the **lang element** `<lang>` that is **introduced with Total.HTM**. The element `<lang>` is always used together with the attribute `lang=""`. For example:

```
<lang lang="de">Eineindeutig</lang>
<lang lang="en">Bijective</lang>
```

> If you make a multi language website, **put all supported translations next to each other** using `<lang>` elements wherever needed. For single language websites forget about the `<lang>` element.

2.1) Single Language Websites

> Give the `<html>` element itself the appropriate `lang=".."` attribute. Remove the language switches and all the `<lang>` elements you dont need. Thats it.

```
<html lang="en">
```

2.2) Multi Language Websites

To support multiple languages, do not set any `lang=""` attribute to the `<html>` element. Instead use **multiple lang lang elements** `<lang lang="..">` inside the `<head>` to create distinct metadata for each language you want to support.

Only a few metadata elements are language dependent. Such as title, description and keywords.

The **MAKE process** will generate multiple **landing pages for each language**. Each one will have `<html lang="..">` set valid.

> Note: The HTML standard does not allow to **assign multiple languages to one HTML document**. Total.HTM provides a solution for that: **The MAKE** process simply generates one valid landing page for each language. **The RELEASE** reads the `Accept-Language` header-data on request at server-side and delivers the **most matching language** to each single end user **at request time**.

Whenever I hear 'lang lang' I am reminded to a wonderful plant with this unbelievably great smell. Info: https://en.wikipedia.org/wiki/Cananga_odorata

If you want to **switch the default language** for Total.HTM browser viewing, just change the lang attribute and the lang-class for the `<body>` element. Like this:

```
<body id="body" lang="de" class="no-js lang-de">
```

> Ignore things like the class `no-js` if you want.

3) The Web App Shell

Total.HTM wants to build a comfortable **bridge between PWA, SPA, API and LYNX**. :) In the context of static web content. But you know: Static data is often quite dynamic and dynamic data is often quite static.

> **Total.HTM is all about static HTML** content, text in various languages, metadata, images, videos and such.

Total.HTM focuses on the static part of the game to free the developers minds about the questions **where and how to include dynamic data**. (DB, API's, REST, AJAX, ...)

Total.HTM builds the bridge between:

PWA: **Progressive Web App**. Web App features for enabled by browsers.

SPA: **Single Page Application**. To stay at one smart website. JavaScript based.

LYNX: A **traditional browser** to load websites as simple text. This is the only way blind and deaf people, for example, can browse websites. By getting text expressed at special devices. Good old Lynx! From Wikipedia: ...the oldest web browser still in general use and active development...

> API: **Application Programmers Interface**. This topic has various aspects in Total.HTM even tough it is all only about static content here. We want to be able to get the exact HTML section and media file directly from the server. The MAKE process prepares the REALEASE together with the file `/index.php`. This allows **direct URL commands to get sections, languages and media files**. The Total.HTM RELEASE really is an API. Working as website and web app and API at the same time delivering just static content.
> Example: `http://exalot.com?lang=es&section=donate&only=image`

(...)

Ok, summing up: We want to have this **new age JavaScript App Feeling** together with appreciating all this HTML 5 and PWA features. But we also want to include handicapt people, people from foreign cultures, people with slow internet, 'freaks' that don't trust Javascript, ... We want to be modern and we want to have **nice fallbacks to the old style**. And we want to **reuse** the website and all it's **assets** in unknown ways.

Total.HTM allows us to have all the fancy stuff, and also to **respects strict web** accessibility standards like **ROCA**. See `http://roca-style.org` (ROCA guidelines are influential to Total.HTM.)

So, why do I write all this here, in the Topic Web App Shell... Because the bridge I am talking about opens some questions about layout and overall design **strategy**. The **concept** of the website itself must work with the **requirements** mentioned above. With and without JavaScript.

Having an **app shell makes it all clear**. There is always the **header**, the **content** and the **footer**. And the metadata. All of the website but the content itself makes the web app shell. This shell is just the surrounding structure where the content takes place.

This **Web App Shell does not have to be loaded again and again** over the wires. If the browser once got the shell, it only has to load a piece of content when requested, section by section.

3.1) Layout Strategy

The **App Shell Concept works well with mobile** devices and is promoted by some big web companies. It usually has some fancy top area, this **hamburger**-menu (those three lines at a top corner), some helpful footer and the content in between.

Total.HTM totally allows you to change the HTML and CSS to **any layout of choice**. In some or another way you will have an App Shell. Just keep the **content ordered in sections** as explained below and Total.HTM MAKE can do a job for you.

I love standards, clear simple symbols and an ever-repeating and improving look&feel. Think about not changing the layout too much but transforming **the CI - the very meaning and feeling** of whatever the website represents - into this Web App Shell Layout. Just as an idea. If the companies CI (Corporate Identity) is reflected so clearly and simply as possible, **the boss will be happy**.

Dynamic data and gadgets come on top. Underlying is **a layout of total simplicity**.

3.2) CSS Strategy

There is much to find about CSS in the Wild Wild Web. CSS has grown over the years. Thank god all browsers support the **display:"flex" strategy**. Flex is a total 'Good to Know' when working with CSS.

CSS Media Queries allow to distinguish CSS code between different devices. More or less based on typical screen pixel resolutions. I recommend not to use much of that. If you know how to **use display:"flex"** your need for media queries will be small.

Keep the ordinary text at `font-size:"1.0em"`, or a bit above. Do you know about **"em"**? Another 'Must Know' when working with CSS. Please look it up, otherwise your CSS will not be good. It will not make people happy, specially not those with special needs. **Using "em" means to respect** the font-size settings of the actual user and his actual device.

> Mostly **use %** and sometimes "em" for the **width**. And **always use "em" for height**!

For example old people who have bad eyes might have set their font-size to "3em" relating to the default. That means they want to read text three times bigger as usual. They want all content to arrive much bigger as usual, so that they can see and understand what the website is all about.

> Total.HTM tries to include totally everybody. Help it with **friendly CSS**.

Check out what new CSS-features are supported at the browsers. Some great and well supported **CSS keywords to look up** are: `flex, opacity, fixed, border-radius, box-shadow, text-shadow, absolute, z-index, transition, transform, rotate, translate, transition-timing-function, gradient, rgba, ...`

3.3) Coding Style

Total.HTM promotes a specific style to write HTML. This has impacts to CSS and to JavaScript. Total.HTM is all about saving you from headache later on.

> Anything unique and somehow important has a **nice ID**. The ID is a fast and clear way to identify HTML elements. It is in fact **THE way to identify HTML elements**. That is why it is called ID.

The outermost logical container of something **unique should have an ID** set. Inner unique elements may also have IDs. For example:

```
<div id="lightbox" class="hide-nojs">
<div id="lightbox-bg-free" class="bg-free"></div>
<div id="lightbox-bg" class="bg">
<div id="lightbox-sym-close"><img src='img/close.png' alt='close'/></div>
<div id="lightbox-content">
</div></div></div>
```

In this example nearly all elements have an ID. Why not. The CSS can address the **elements with ID direct and fast**, and the JavaScript as well. Everybody happy. Example of some CSS addressing a HTML element by using its ID:

```
#lightbox-sym-close
{
position:absolute;
top:-1em;
right:-1em;
cursor:pointer;
}
```

And some simple JavaScript (using jQuery) addressing the lightbox:

```
$("#lightbox-sym-close").click(function(e)
{
    $("#lightbox").removeClass("show");
});
```

(…)

IDs are unique clear identifications to be used wherever useful. Friendly IDs have no **special characters** other than dash – and lowercase letters. They consist of simple English words.

Do not identify unique elements using class. Use **classes to identify logical groups**, kinds and potential multiple occurrences of things. By using ID's for everything unique, you will not need classes too much probably.

A typical class is `text-menu`. It occurs **several times at the website**. All occurrences of `text-menu` belong to one logical group. Groups a typical use-case for classes.

```
<li><a href="#offer"><span class="text-menu">
    <lang lang="de">Leistungen</lang>
    <lang lang="es">Ofrecemos</lang>
    <lang lang="en">Offers</lang>
</span></a></li>

<li><a href="#about"><span class="text-menu">
    <lang lang="de">Über Uns</lang>
    <lang lang="es">Nosotros</lang>
    <lang lang="en">About</lang>
</span></a></li>
```

The class `text-menu` addresses menu text elements to make them all look similar.

```
.text-menu
{
  font-size:1.3em;
}
```

Classes are used for other purposes as well, such as switches and options. You find a number of **predefined classes** described further below. Total.HTM comes with a set of predefined classes that enable or disable certain functions,

If you have no special reason, consider examining the existing HTML and CSS coding style and just go along with it. It really helps in software projects to **stay with with one style** for the whole thing. No-Headache Policy. KISS: Keep it small and simple.

> Good programming is all about giving good names. Good names are simple and meaningful. **Good names, good code.**

3.4) DOM Tree

```
<html>
<head>
    <!-- ... -->
</head>
<body>
<header>
    <!-- ... -->
    <nav>
          <a href="#about">About</a>
          <a href="#team">Team</a>
    </nav>
    <!-- ... -->
</header>
<main>
    <section id="main">
          <section id="home">
                <div class='section-header'> <!-- ... --> </div>
                <div class='section-body'> <!-- ... --> </div>
          </section>
          <section id="about">
                <div class='section-header'> <!-- ... --> </div>
                <div class='section-body'> <!-- ... --> </div>

                <section id="team">
                      <div class='section-header'> <!-- ... --> </div>
                      <div class='section-body'> <!-- ... --> </div>
                </section>
          </section>
    </section>
</main>
<footer>
    <!-- ... -->
</footer>
</body>
</html>
```

As you see the code of Total.HTM is nothing unusual. Very **normal HTML**.

The **<main>** is the place for the sections. All content goes into sections. The **<header>** above and the **<footer>** below.

Each section has an ID that is very important. The `<nav>` `<a>` links point to the sections using this ID.

4) Sections

> All the static content is organized in sections. The sections can have sub-sections and so on. In that way a **tree of sections** is emerging. Each section has a unique id, that is also called the section name. We say "home section" or "about section" or "contact section". It feels natural to **organize the whole website into sections**.

The **main section** has the `id="main"` and has to be the only section within `<main>`. It is the **root of the section tree**.

To create a new section just copy the HTML-code of a section to another place, give it a new unique ID, thats it! You organize the sections just as HTML code. Create; Modify; Delete; Everything in this one file `/total.htm`.

You can **choose how sections behave**. Should all sub-sections be opened or just one at a time? Give the `<section>` some classes to control its behavior.

Total.HTM sections support **two section modes** at the moment:

• `sub-switch`	Only the current sub-section is visible.
• `sub-keep`	All sub-sections keep visible.

Further you can choose what **sub-sections** are visible at the start:

• `sub-show-none`	Initially no sub-section is visible.
• `sub-show-one`	Initially the first sub-section is displayed.
• `sub-show-two`	Initially the second sub-section are displayed.
• `sub-show-three`	Initially the three sub-section are displayed.
• `sub-show-all`	Initially all sub-section are visible.

An example how a section can be defined in HTML:

```
<section id="offer" class="sub-keep sub-show-none style-green">
```

> For reasons of **SEO (Search Engine Optimization)** and other functions of the RELEASE you can set attributes on `<section>` for **priority and change frequency**: `data-changefreq=".." data-priority=".."`

The **possible values for data-changefreq** are: `"always", "hourly", "daily", "weekly", "monthly", "yearly", "never".`

The **possible values for data-priority** are: Numbers between 0 and 1. A value like 0.7 means quite important. 0.0 means irrelevant.

Don't give all sections a maximum of data-priority. Create a **balanced picture** of the whole, identify the most relevant sections for SEO and give them higher values.

4.1) Content

All content is organized in sections. The HTML for the actual content is recommended to be put into the **section-body** `<div class='section-body'>`

inside `<section>`. This allows the RELEASE to load everything inside section-body with AJAX dynamically into the browser.

> The content flows. All the design flows, but the **content needs to flow** as well and that is often hard to maintain. Content is text, and also multi-media. To keep everything flowing and flexible this few guidelines exist.

Do not set a CSS `font-size` to a `<div>` or to any other block element. The reason is, that **font-size is used for positioning**. 2em margin has a certain meaning that changes when setting the font-size. All block elements keep with the side-wide `font-size:"1em"`, it's the reference for layout and position.

Only set the font-size of text elements. Text elements like `<span>` are leafs in the tree of HTML elements and thus do not interfere with positioning of outside elements like `<div>`.

4.2) Text

Always use **text elements wrapped** by `<div>` block elements. The basic text element is `<span>`. It should always be inside a `<div>`. Do not use the deprecated element `<font>`. Use `<span>` instead.

Do **not use the paragraph** element `<p>`. It conflicts with text-like-elements as the list `<ul>`. Instead of `<p>` put text that relates to each other into its own `<div>`. For explicit **line breaks** use the `<br/>`. Let the browser break text, so that it flows well at any font- and screen-size .

Treat the `<h1>, <h2>, <h3>, <h4>, <h5>` **headings like text elements**. Just like `<span>`. Put heading elements into `<div>` as well.

Write all labels, captions and titles with **first letter upper-case**. Like: "The Little Prince". Because you can set lower-case or upper-case easily in CSS `transform-text`. This way you can choose between all 3 variations. The style with first letter upper is called **camel-case**.

For usual text use `<span>` and within that **use semantic text elements**. Semantic text elements are recommended. Such as `<abbr>, <code>, <label>, <var>, <address>, <time>, <cite>, <code>`.

To make text **bold** surround it with `<b>`. Do not use `<strong>` or other ways.

A well done example somewhere in a `<div class='section-body'>`:

```
<div>
    <span>
          <lang lang="de">Die Natur weiß es am Besten!<br/>Lasst sie uns
schützen!</lang>
          <lang lang="en">Nature knows best!<br/>Lets protect her!</lang>
    </span>
</div>
```

The **link element** `<a>` is not meant to be used as text element. Also don't use it as block element or logical container. Use `<span>` inside `<a>` to write the text for the link. Like this:

```
<div>
    <a href="#contact"><span>Contact Us</span></a>
</div>
```

Predefined **classes** for text elements, are: `text-big` and `text-menu`.

4.3) Images and Media

Give `<img>` images **a nice alternative text** in the `alt="..."` attribute to respect people with special needs. A good alt often looks like a good filename.

> Use the `<picture>` element instead of `<img>` if you have great resolutions of images and you want the user to receive a high resolution on big screens. Or if an image can be delivered significantly smaller for mobile devices. The picture element just allows you to define several versions of **images for different screens**.

Use the HTML5 **multimedia** elements like `<audio>` or `<video>`.

Provide a nice **logo** and a nice **main icon** based on the logo. The icon should look good at 512*512 and as well at 64*64. It should consider different background-colors to be used on. It should consider the use of transparency and how much of it not to disappear on some backgrounds.

In the folder `/image/icon/` are **various sizes of the icon** image. They can be auto-generated with tools. E.g.: `https://app-manifest.firebaseapp.com`

The main **logo file** might be called `/logo.png` and is located at the root folder, together with the main **web icon** called `/favicon.ico`. Use this old default name meaning 'favorite icon'.

Images are organized within the sub-folders of `/image` and `/img`. Do not use special characters of any sort in filenames or directory names. Recommendation: Always use lower case names in file systems. Only use _ in between words. Like: `/nice_folder/this_is_a_nice_filename.txt`

Within content consider using **images with small resolutions** such as 400*300 pixel only. That is ok, because on big screens the website appears only at the center and the images appear usually only half-width or third or 25%. At mobiles the images are displayed full width but here the screen is small anyway. Ergo: In fact, images are **not needed in high resolution** inside the content flow.

If you use small images and you have a **bigger version of the image** in the same folder, you can use a cool **automatic feature of Total.HTM**. Just give the smaller file a "`_s`" like small into the filename and Total.HTM MAKE creates links to the bigger version for you. For an image called `/team.jpg` just make a smaller file called `/team_s.jpg` and keep it in the same folder within `/image` somewhere. The rest is done automatically by MAKE.

> If you manually link an image using `<a href="..">`, the feature described above will not be applied to this image. An automatic link to a bigger version of an image is only generated by MAKE if the bigger version exists in the same directory and if no manual link was set for this image in the HTML code.

For images used as content (that's most images probably) use the class `img-content`. Use this class also to define the standard design of your content images with CSS, using things like `border-radius`, `box-shadow` and such. All regular **content images *should* have this class `img-content` set.**

Further useful classes for defining the **image flow** within text – or more generally speaking: box flow within text – can be: `float-left`, `float-right`, `width-half`, `width-third`, `width-quarter`

> Images that have the class `img-content` set, automatically get **full width on tiny screens**.

4.4) Boxes

Here we are: Living in boxes. Well, if so, lets do it well. For many purposes where you might think about lists or tables, **boxes are a nice solution**. Whenever you have more than one of something, consider using boxes. Boxes are the first choice of designing plurality. Boxes allow the layout to adapt well between **all kind of screens**.

If you think boxes are boring behold! You can do the most crazy things with boxes, like rotating or shaking them. Boxes give us the liberty to make crazy things with them. The box concept is so basic that the brain can grasp the meaning in no time.

And the most important of all – the reason why I promote boxes so much here – is, that boxes together with the `display:flex` directive and some other tricks flow into the modern design so well. **Box oriented content layout** is just perfectly fitting the No-Headache policy of Total.HTM.

An example of a typical **box use in Total.HTM**:

```
<div class="box-container box-container-third">

<div class="box-sub">
    <img src="image/love_s.jpg" alt="love" class="img-content"/>
</div>

<div class="box-sub">
    <img src="image/nature_s.jpg" alt="nature" class="img-content"/>
</div>

<div class="box-sub">
    <img src="image/truth_s.jpg" alt="truth" class="img-content"/>
</div>

</div>
```

To **define boxes** just use an outer `<div class="box-container">` with as many `<div class='box-sub'>` as you want inside. Further you can control the width of the box container with classes: `box-container-third` and `box-container-half`.

Small screens show boxes in full width automatically.

4.5) Lists

Use **lists of the type ul only** `<ul> <li>`. Their CSS is overwritten not to make troubles. Lists are one of those HTML things that kept surprising us web developers. We like to create surprises, but we do not like to handle them.

Use lists for **listed text elements only**, not for other design and layout purposes. Consider boxes in case.

In Total.HTM lists have **custom list icons** stored at the folder `/img`. Create your own symbols and assign them over CSS if you like.

There are a few predefined classes to **set different list icons**. Those are `list-icon-arrow`, `list-icon-round` and `list-icon-none`. It is possible to create your own list icons and use classes to switch between them.

4.6) Tables

Do not use tables for design or layout purposes. Only use `<table>` **if you just want a table** filled with values inside your content.

A valid and complete **HTML syntax for table**:

```
<table>
<caption><div><span>Table Title</span></div></caption>
<thead>
  <tr>
    <th><div><span>Column Title A</span></div></th>
    <th><div><span>Column Title B</span></div></th>
  </tr>
</thead>
<tbody>
  <tr>
    <td><div><span>Content A1</span></div></td>
    <td><div><span>Content B1</span></div></td>
  </tr>
  <tr>
    <td><div><span>Content A2</span></div></td>
    <td><div><span>Content B2</span></div></td>
  </tr>
</tbody>
<tfoot>
```

```
    <tr>
      <td><div><span>Summary A</span></div></td>
      <td><div><span>Summary B</span></div></td>
    </tr>
  </tfoot>
  </table>
```

Pretty cool is the `<tfoot>` element. Use it for the **summary line at the bottom** of the table. Also very nice, and good for people with special needs, is the active use of `<caption>`.

5) Testing

The first level of testing is just opening `/total.htm` in any browser on the local computer. For example just opening the file in Firefox.

Just **have a look in the browser** if everything is as meant to be.

> A **very powerful** and 'not so complicated as it looks' set of tool is called '**Developer Tools**' or 'Inspector' within the browser. Definitely have a try. It gives you full control over any website. It might **open right in the browser tab** by pressing the key combination Ctrl + Shift + I

6) MAKE

The MAKE process creates the **so called RELEASE**. That auto-created set of files is **ready to be copied on the web server** and to be shown to the client. If you want to sell a website, do not show the file `total.htm` to the client before he paid. :)

Total.HTM is so direct and honest that it includes everything you (might think you) need. Keep the file on your personal development computer only if you want to sell websites. **Only give the RELEASE to the public** because it fully works (better than total.htm itself) and is 'not so easy' to copy.

As well feel welcome to **share** your `/total.htm` and everything **with everybody Open Source** and **free** for commercial use. **It is up to you!** Total.HTM wants to be helpful to private developers who want/need to make money as well as to Open Source initiatives.

MAKE takes the `/total.htm` and creates a set of files that build an optimized 'web presence' for users, special people, robots, apps, API, and SEO. This set of files – together **making up a well done website** – is **the so called RELASE**.

There are many reasons why to **use the RELEASE for real online websites** always.
An optimized (static part of a) website serves 1001 purposes, many still unkown.
Total.HTM makes it trivial for you. Just care for a well done HTML `/total.htm`,
the RELEASE is made by MAKE from it.

7) More...

Coming soon!