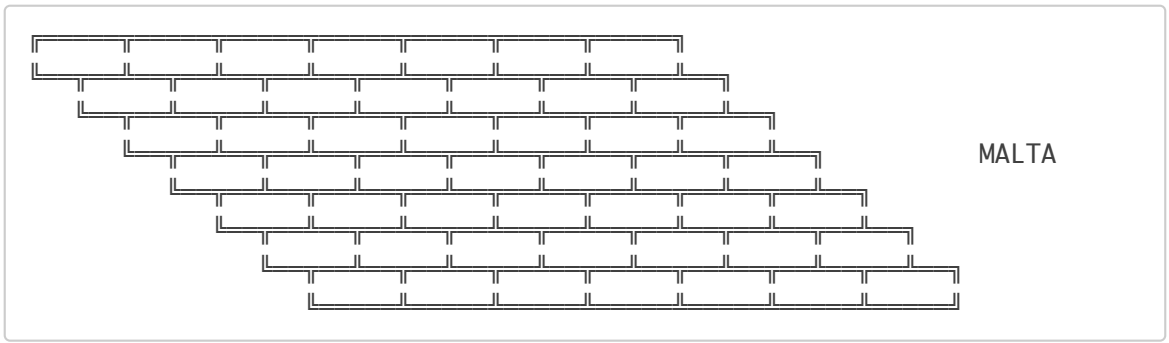

Malta is ...

a simple builder which allows to built on the fly big files editing its parts in separated files and assembling in one following a main template file. Imagine for example building a game where many objects acts, then there's the engine which will be composed by many different files, then third party libs, and the main script.
Most likely one aims to use a single minified file and maybe one stylesheet. *Malta* solves this problem and once started, the result will be updated on the fly as far as something relevant changes.

Get started

Plugins for postprocessing

Write Your plugin

Malta allows
As You start Malta you can start editing every file involved and the resulting file will be updated on the fly. In every involved file you can use variables coming from a json file, and even use a value obtained evaluating an expression that involves those variables.

Installation

If You do not have node installed yet, run:

```
$ curl http://npmjs.org/install.sh | sh
```

then install malta running:

```
$ [sudo] npm install malta [-g]
```

Usage

Malta can be started either in *single-file* mode either in a *multi-files* mode. The first one can be started running

```
$ malta templateFile outDirectory [-vars=non_default_vars_path] [-option
s=options] [-require|plugins=name1(options)[.name2(options)[...]]]
```

- **templateFile**
  is the base template used as base file.

- **outDirectory**
  is the folder where the output files will be written in.

- **-vars=*non/default/myvars.json***
  here is possible to tell Malta to consider a different file for variables (default would be *templateFolded*/vars.json); if used the path specified must be relative to the execution folder.

- **-options=*key1:value1,...***
  here the following key:values are considered:

  - *showPath*: *boolean* (default : true)
    Malta for certain files will prepend the file inclusion with a small comment which will be really helpful when looking in the resulting file one wants to know from which file a particular section comes from. Significant only in xml, svg, js, css, less, scss files.

  - *watchInterval* : *integer* (default 1000)
    This is the interval in milliseconds between every check for a modification in one of the involved files

  - *verbose* : *integer* (default : 1)
    0 no console messages 1 default messages 2 verbose messages

- **-plugins=**
  Malta is shipped with a number of plugins to do as postprocessing job every special task

that was done before like compiling less files, packing js, etc... To use one plugin a `-require` or `-plugins` argument must be specified when invoking *malta* and if the plugin allows it, some parameters can be passed to it.

3dot separated list of elements, one for each plugin with the following structure: `plugin-name(key:value,...)`

Multiple files

Seen that most of times it would be handy to engage many builds at once it's possible to start Malta in a *multi-files* as follows:

```
$ malta list.json
```

where *list.json* is a file containing one or more pairs, that commits Malta to build more than one file in one shot:

```
{
    "palette.less" : "../../public_html/css -vars=./vars/deploy.json",
    "common.less" : "../../public_html/css -plugins=malta-less(compress:
false) -options=skipPlain=true",
    "common.js" : "../../public_html/js -plugins=malta-js-packer",
    "lib.js" : "../../public_html/js -plugins=malta-js-packer",
    ...
}
```

where for each element the **key** is the *templateFile* and the **value** should contain `_outDirectory_ -vars=... -options=... -plugins=...`

Plugins

The version 3 of *malta* aims to simplify any postprocessing needed to be done on the composed file. In fact now *malta* just do the job of creating the big file from the template and all files and variables involved... nothing else, no packing for js, no less/sass compiling, only the clean plain big file. Whatever work is needed afterwards it needs to be done by a plugin, some plugins come shipped (at least all those needed to obtain same results as with versions < 3).

When a plugin `myplugin` is requested the first place malta will search for it is `executionfolder/plugins/myplugin.js` if not found will be searched as `plugins/myplugin.js` into the local malta package path.

Usually a plugin has one dependency which allow the plugin code to be really simple.

Placeholders

Malta uses three kind of placeholders, to be used in the main template or in any file involved (but *vars.json*)

- **$$filePath$$**
  filepath is the path to the desired file relative to the templateFile directory; if starts with / then will be relative to the execution folder

- **$varname$**
  varname is the key for a variable that Malta will search in a *vars.json* file that should be found in the template folder

- **!{expression}!** *expression* can contain anything that must be evaluated (eval is used)

Hints

- **absolutely** use only spaces and tabs before file placeholders

- one line files (like already minified ones) **really** slow down the parsing, so the best thing is to avoid the inclusion of minified/packed files.

- to avoid loops Malta stops digging at the tenth nesting level.

Wired vars

There are some placeholders that can be used within any involved file:

- __TIME__ : the HH : MM : SS build time

- __DATE__ : the D / M / YYYY build date

- __YEAR__ : the YYYY format build year

- __FILESNUM__ : the number of files glued togheter

- __VERSION__ : Malta version

- __BUILDNUMBER__ : build number

Foo sample

Supposing in `~/myproject/` folder there is the following

```
myfile.js
vars.json
out/
src/
|- a.js
|- inner/
   |- b.js
```

The most important is the Malta template file being the first file used to build the glued file.

Here use the Malta placeholders and/or the wired vars to specify which files/variables must be included.

myfile.js :

```
/**
// use the `name` and `author` variables from the vars.json
// the wired __DATE__ variable
//
Name : $name$
Author: $author.ns$
Project : $more.repo$
Date: __DATE__
*/
+function(){
    var name = 'what',
        data = $data$,
        tenPrimes = $tenPrimes$;

    // write here the content of the src/a.js file
    // the path is relative to the template folder
    //
    $$src/a.js$$
    console.debug(!{$width$ * $height$ / 7 - 8}!) // new 'comp' placehol
der (>=2.3.8)
}();
```

and here is the **src/a.js** :

```
function hello(n) {
    alert('Hello ' + n);

    // as before, always relative to the template
    // even if this was at 10th inclusion level
    //
    $$src/inner/b.js$$
}
hello('Federico'), hello('Federico');
```

the last content file for that dummy sample is **src/inner/b.js** :

```
hello = function () {
    alert('Hello again ' + n);
};
```

and least but not last **vars.json** :

```
{
    "name":"myFabulousProject",
    "author":{
        "name" : "Federico",
        "surname" : "Ghedina",
        "ns" : "$author.name$ $author.surname$"
    },
    "more" : {
        "repo" : "https://github.com/fedeghe/malta"
    },
    "data" : {
        "namesurname" : "$author.name$ - $author.surname$"
    },
    "tenPrimes" : [2, 3, 5, 7, 11, 13, 17, 19, 23, 29],
    "width" : 112,
    "height" :124
}
```

Now from ~ execute:

```
 malta myproject/myfile.js myproject/out [-vars=myproject/local/variable
s.json]
```

in a while Malta will confirm the first creation of *myproject/out/myfile.js* and *myproject/out/myfile.min.js*.

The *myproject/out/myfile.js* will look like:

```
/**
Name : myFabulousProject
Author: Federico Ghedina
Project : https://github.com/fedeghe/malta
Date: 11/9/2013
*/
+function(){
    var name = 'what',
        data = {"namesurname":"Federico - Ghedina"},
        tenPrimes = [2,3,5,7,11,13,17,19,23,29];
    function hello(n) {
        alert('Hello ' + n);
        hello = function () {
            alert('Hello again ' + n);
        };
    }
    hello('Federico'), hello('Federico');
    console.debug(1976)
}();
```

Let Malta run and try editing the *myproject/myfile.js* or the *myproject/vars.json* (or the overridden one) or one of the involved files, and get a look at the output folder content. To stop it use Ctrl + c.

Plugins shipped ...

Malta comes shipped with some plugins to deal with js/css minification, less/sasss compiling,

markdown to html/pdf. All these plugins have one core dependency that does the jobs, the plugin code itself is just an adaptor function.

To use each plugin his dependency must be installed:

- *malta-js-uglify* (*uglify-js*) what if
- *malta-css-uglify* (*uglifycss*)
- *malta-js-packer* (*packer*)
- *malta-less* (*less*)
- *malta-sass* (*node-sass*)
- *malta-markdown* (*markdown*)
- *malta-markdown-pdf* (*markdown-pdf*)
- *malta-svg2png* (*svg-to-png*)

... and not

Writing a new plugin is really easy, the plugin code is just an adaptor function li:

add-header-comment.js

```
/**
 * require section
 */
var fs = require('fs');

/**
 * main plugin function adaptor
 */
function myfunc(content, options) {
    /**
    * here basically all plugins create another file,
    * but even the primary output can be overwritten;
    *
    * the context here is the Malta instance thus will
    * be easy to retieve all informations needed
    */
```

```
    var start = +new Date,
        self = this,
        fpath = options.headerCommentFile+ '.txt',
        ext = self.utils.getFileExtension(self.outName),
        newContent = ext in self.comments ?
            self.comments[ext].replace(/\%content\%/, fs.readFileSync(fp
ath).toString()) + content
            :
            content,
        msg = 'plugin ' + path.basename(__filename) + ' wrote ' + self.o
utName + ' (' + self.getSize(self.outName) + ')';

    fs.writeFile(self.outName, newContent, function(err) {
        // do whetever is needed to handle the error
    });

    self.notifyAndUnlock(start, msg);

    self.listen(fpath);

    return [
        new Promise(function (solve, reject){solve(newContent)}),
        newContent
    ];

}

// here must be declared on which files the plugin will act on
//
myfunc.ext = ['js'];

// export
//
module.exports = myfunc;
```

the function will be called passing :

- content : the content of the resulting 'resolved' Malta template
- options : a json containing what is written between squared brackets after the plugin name
  `add-header-comment[headerCommentFile:'test/src/maincomment']`

context is the Malta instance

Changelog

- **3.0.0** Malta main file completely rewritten:
  - new plugin based architecture, now doing something more with some files is really easy
  - every previously additional file now is produced only if the user needs it
  - write Your own plugin is extremely easy
- **2.4.1** removed some unuseful messages from console
- **2.4.0** .svg files will automatically output even a .png
- **2.3.8** added *comp* placeholder for evaluate simple arithmentic expressions
- **2.3.7** better handling for `less` compiler exceptions
- **2.3.5** fixed some typos in the README file
- **2.2.8** var placeholder replace with JSON.stringify output in case of object
- **2.2.7** new options available for files that can be minified/packed
- **2.2.6** fixed a small bug in the console ouput messages for the packed versions of js files
- **2.2.5** in case of js files will be written even a packed version using the amazing Dean Edwards npm port
- **2.2.4** added detection for placeholders loops into the variable json
- **2.2.3** variables json can contain inner placeholders at any level
- **2.2.2** fixed a bug related to vars substitution
- **2.2.1** in file placeholders is possible to use absolute paths, will be based on to the execution path; vars.json file can contain deeper literals that can be references with . or / separator in the placeholder (see examples above)
- **2.2.0** some refactors

- **2.1.3** in vars.json nested vars can be used
- **2.0.6** markdown to pdf support added, just use .pdf.md for the templates file
- **2.0.5** markdown support added, every .md tpl will produce the glued .md and resulting .html file
- **2.0.4** lack of --force drives to a new version just to remove a console.log !!!
- **2.0.3** is possible to specify the complete path (relative to the execution folder) of the variable json.
- **2.0.2** if using a json file for multi build, a ! as first key character will tell Malta to ignore this line
- **2.0.1** fixed README links
- **2.0.0** no more stop if the same file is included more times, still check for loops over 5000 files
- **1.1.1** removed some ugly and unuseful messages from console
- **1.1.0** updated console usage message
- **1.0.21** fixed a bug naming minified css
- **1.0.20** added support for .scss files, fixed a bug using less package
- **1.0.19** fixed a bug that hanged the process when, being not caught, a parsing exception was thrown by uglify-js and/or uglifycss
- **1.0.18** some refactors and corrections to console output
- **1.0.17** automatically write even minified version for css files (even less originated)
- **1.0.16** accepts a json to execute multiple builds with one call
- **1.0.15** removed beginning os specific slash in inclusion comments
- **1.0.14** some fixes and refactor
- **1.0.13** __BUILDNUMBER__ predefined build number var (file based)
- **1.0.12** fixed path sep for win####

- **1.0.11** fixed deadly circular inclusion check; update only modified files
- **1.0.10** xml files indentation for inner files removed
- **1.0.9** some minor fixes on messages
- **1.0.8** hint paths changed
- **1.0.7** added support for .less files
- **1.0.5** real path is included only for .xml .js .css files
- **1.0.3** real path included just before every inclusion
- **1.0.1** not found $vname$ placeholders are untouched
- **1.0** added __FILESNUM__, __VERSION__ to the placeholders builtin set
- **0.0.11** fixed inclusion indentation
- **0.0.10** involved files count fixed
- **0.0.9** fixed build on vars.json change
- **0.0.8** parse error thrown by uglify is catched and stops no more Malta