Hygia Web-Application Testing framework

By Bas van Gaalen, April 2007.

Overview

Introduction

Hygia is a testing framework that can help monitoring the 'health' of web-applications. It is specifically designed for regression testing of web- or network accessible software. The basic idea behind Hygia is that it mimics client requests to the service to be tested, in an automated and potentially unattended way. The input and expected output of the application's end-points can be configured, using XML-files. Request and Response data can be manipulated with simple custom made plug-ins and the output can be verified with several types of validators. Test results can be seen in real-time with a web browser and (unattended) test session reports can be written to log files and send by email.
Hygia is written in PHP 5, but can test any application, written in any language that is accessible through a local or wide area network. It can test both front-end and back-end tiers of applications.

How is this test application different then other test suites like HTTPUnit, iValidator, et al? In one word: simplicity. Hygia is relatively simple to install and configure and provides intuitive configuration of test cases. Though its simple configuration, Hygia houses quite some horse-power under the hood.

(In case you're wondering: Hygia is the goddess of health in Greek mythology.)

Features

In short, Hygia's functionality can be summed up with these features:

- real-time detailed overview of test progress
- tests can be bundled in modules based on web-application logic and virtual host / domain name
- log reports of test sessions can be written to file and send to email in plain text and rich HTML, on a per-module basis
- full request and response data with connection and data transfer times is available in the reports
- end-point definition through XML configuration
- support for the HTTP and HTTP/SSL protocols
- support for HTTP authentication and authorization
- support for GET and POST requests
- basic compliance for rfc-1867 (Form-based File Upload in HTML)

- basic compliance for rfc-2616 (Hypertext Transfer Protocol – HTTP/1.1)
- request and response data streams can be manipulated through custom written plug-ins
- plug-ins also allow possibility to construct flows by saving session data between consecutive tests
- end-point output can be verified with several types of validators: regular expression, size, XSD Schema, Content Type, MD5 hash: there is not need for programatically validating responses
- plug-ins and validators are relatively easy to write due to 'semi-intelligent' Request and Response objects

Convention(s)

subject: For ease of writing the term 'subject' is being used for the application being tested. The application that Hygia is testing can be of any type, written in any language, as long at is understands web protocols, like HTTP and HTTPS, to communicate with the outside world. So this could be anything ranging from a REST-based API, a SOAP-service, a propriety proxy service to a simple website with HTML-forms, a back-end for a rich-client web application written in PHP, Java, Perl, Ruby, .NET, Coldfusion or whatever language, etc...

Requirements

PHP & PHP Extensions

Hygia is written in PHP 5.2.0 and relies heavily on PHP's object oriented nature and XML capabilities. Older versions of PHP or PHP's XML sub-system may not be adequate. Besides full-fledged XML, some other requirements need to be met.

Hygia uses PHP's mail functionality to send test reports by email. Also, Hygia will write full html reports to the filepath configured in the hygia/libs/core/TestConfig class and plain text reports to module logs as defined in the test configuration (see Configuration chapter). There directories need to be writable by the user PHP is running as.

LibXML version 2.2.26 with XPATH and Schema support enabled (full DOM/XML support). For its configuration, Hygia uses XML and when validating responses of subjects that use XML, Hygia will require Schema support. Earlier versions of LibXML have proven to be unstable or insufficient, but your millage may vary. If you experience weird XML-related behavior, try upgrading the version of LibXML you're using.

Hygia uses PHP's getcwd() function, so make sure the proper directory rights have been set to test sub-directories. Hygia 'chroots' to the test directories when running a certain test suite, so make sure that the directories have the proper rights set, for PHP to access these directories. Usually, when PHP runs as an Apache module (for instance), the user PHP runs as is 'nobody'. Check your configuration or ask your system administrator.

Hygia uses sockets to connect with the subject. Make sure they are enabled in PHP. Hygia acts as a client that connects with servers and transfers data back and forth and it relies on sockets to do this. The socket extension needs to be enabled in the PHP configuration.

OpenSSL, when required by a subject's configuration (when using the HTTPS protocol). Hygia itself does not require OpenSSL, but certain tests might. So depending on what your testing plans are, you might need to enable the OpenSSL extension in PHP.

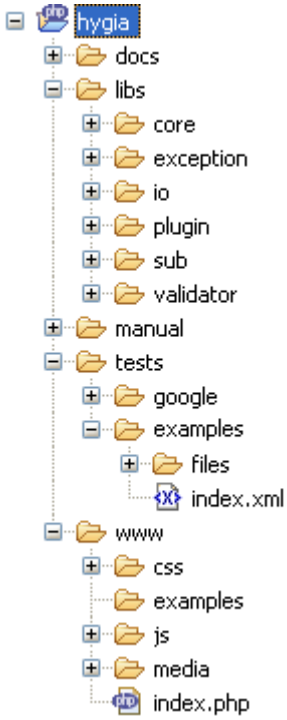
Installation

Download

Hygia can be downloaded from SourceForge:
See: http://sourceforge.net/projects/hygia

Also, Hygia can be downloaded from GoogleCode:
See: http://code.google.com/p/hygia/

Install



Hygia itself is a web application, written in PHP, and as such needs to be run on a web server to be accessible by a browser. It can also be run from a cronjob for continuous monitoring services on a subject.

Hygia does not need to be installed on the same server as the subject. It is even preferred not to do so, as long as the system it is running on can connect to the system(s) to be tested.

When unpacking the downloaded file to a directory of choice, a directory structure, similar to the one at the right, will appear.

The *docs* directory holds several informational files, like this manual in several formats (Open Office, Adobe Acrobat PDF), some images that can also be found in this manual, some Doxygen related files and the API documentation, of which at least parts might be interesting, if you're planning to extend the system with new validators and/or plug-ins. The API documentation is created from the source code by DoxyGen.

The *libs* directory contains the complete source code of the Hygia application itself, divided into some logical parts, of which *core* holds a few libraries that comprise the main application.

The *manual* directory holds this user manual in several formats.

Tests are stored in the *tests* directory – more on this later.

Ideally, Apache (or your web server of choice) should be set up with a virtual host pointing to the *www* sub directory. For instance, while developing Hygia, the following Apache virtual host configuration was used:

```
NameVirtualHost *

<VirtualHost *>
       ServerName hygia
       DocumentRoot /dev/projects/local/hygia/www/
</VirtualHost>
```

With Hygia installed in the `/dev/projects/local` directory (refer to Apache's manual for more information on setting up virtual hosts). Then 'hygia' domain was added to the Operating System's host files (for Linux this usually can be found in /etc/hosts and for Windows it is c:\windows\system32\drivers\etc – search your Operating System's manual for more information). Opening something like http://hygia/?mod=test in a browser, would start Hygia.

Configuration

This chapter will explain how to define the end-points of the subject.

Main Application Configuration

The first thing to do, after unzipping the packing in a directory of choice, is to open up `hygia/libs/core/TextConfig.class.php` in your favorite editor. This file defines some constants used in Hygia to either name things or locate things. The file looks like this:

```
abstract class TestConfig {

    // application title and version
    const ApplicationTitle          = 'Hygia Web Application Test Suite';
    const ApplicationVersion        = '0.9';

    // web root of hygia, used in html header
    const ApplicationWebRoot        = 'http://hygia/';

    // path and file name (excluding extension) of html reports
    const ReportFilePath            = '/var/log/hygia-report';

    // show debug content with a binary percentage below this threshold
    const ShowBinaryPercTresh       = 10;

    // show only content below this threshold (number of bytes)
    const ShowContentLengthTresh    = 1024;

}
```

Set the `ApplicationWebRoot` to the domain used for Hygia. Setting this correctly makes sure the HTML reports can access the style sheet and external javascript code.

The `ReportFilePath` is the path and base file name (without extension) of the HTML reports. A date & time stamp will be added, plus the .html extension. A report of every single run will be generated here. Make sure the directory is writable for PHP/Apache.

The request and response data is available in the full HTML reports and `ShowBinaryPercTresh` defines the percentage of binary data allowed in these reports. When testing file uploads or downloads or other binary data, it might be wise to alter this value as you see fit.

When the request or response body is exceptionally long, you might want to choose not to show it. `ShowContentLengthTresh` defines the threshold below which the content will be included in HTML reports.

Test Configuration

Hygia does not record use cases, instead it requires manual configuration of subject end-points. The advantage of this is method, is that this makes it possible to specifically define what to validate in the response data. Also both input and output data will not be an exact blue-print, but a more flexible definition of the data. This means that it is possible to make layout or design-technical changes to the subject, without the need of re-recording or re-defining the test cases.

Test Suite, Module, Test Item

The definition of the end-points of the subject is described with XML, so a basic knowledge of XML is necessary in order to configure Hygia. Test configurations need to be saved in a subdirectory under hygia/tests (see the directory structure in the Installation chapter). Every XML file describes a Test Suite, which is basically a set of tests or use cases to be run on a subject. A Test Suite is divided into one or

more Modules that contain the Test Items.

A Module can be a logical part of the subject, like for instance the set of libraries that controls user accounts, or the database / persistence layer. Every module is limited to one single server configuration, defined by protocol, domain and port to connect with. All tests defined in the module will use these server settings.

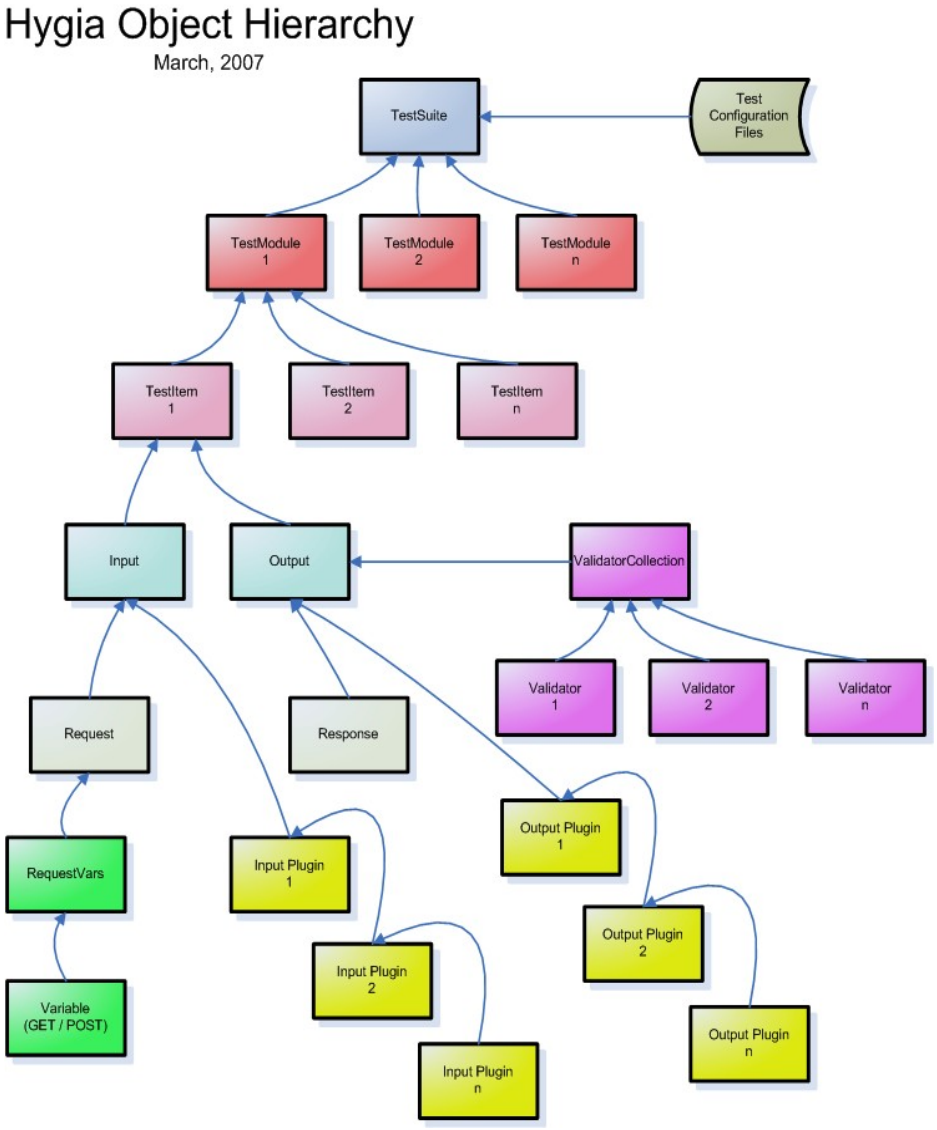
A Module holds one or more Test Items, which is the definition of an end-point of a subject, like for instance the URL for signing in, with input parameters user name and password and output validators to verify the reply of the server.

A test directory can holds several configuration files, a configuration file can hold several modules and a module can hold several test, so test definitions of subjects can be set up in about any possible way.

Object Hierarchy

A bit of what Hygia does behind the screens, when it reads the configuration files. You may skip this section, if too technical.

At runtime, Hygia will read the XML configuration files, as specified by the 'mod' request argument (see Starting Tests), parse them and dynamically build an object hierarchy in memory, as depicted in the image below. The API documentation describes the class functionality in more detail.



A Test Module

Every object has its function. A Test Suite or test-batch, the main element in the XML configuration, contains one or more Modules. Besides one or more Test Items, a Module has a few properties.

- Title of the Module, for display and reporting purposes
- A Server definition, consisting of:
  - protocol: either HTTP or HTTPS
  - domain: the domain where the subject is located, like www.example.com
  - port: the port, optional and defaults to 80 for HTTP and 443 for HTTPS
- Per-Module logging to email can be enabled, optionally only on errors in the test(s)
- Per-Module logging to plain text file can also be enabled and also with the on-error option

In the detailed description below, the number of children elements is described like this:
? = 0 or 1, optional element of which only one can exist
* = 0 or more, optional element, but can be unlimited number
+ = 1 or more, so at least one must exist
When none of these symbols is used, then the element is mandatory and there can be only one of it.

An example Module configuration, might look like this:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<test-batch>
    <module>
        <title>My Module</title>
        <server protocol="http" domain="localhost" port="80" />
        <log-email on-error="1">hygia@example.com</log-email>
        <log-file>/var/log/hygia/mymodule.log</log-file>
        <tests>
            ... test definitions go here ...
        </tests>
    </module>
    ... optionally more module definitions may follow ...
</test-batch>
```

<test-batch>
Tag: *test-batch*
Xpath: *test-batch*
Children: *module+*
Description: Container for module definitions.

<module>
Tag: *module*
Xpath: *test-batch/module*
Children: *title, server, log-email?, log-file?, tests*
Description: A module contains one or more test definitions (in the *tests* container) and represents an arbitrary part of the subject. This is where the server definition goes and optionally it is possible to enable per-module logging.

<title>
Tag: *title*
Xpath: *test-batch/module/title*
Description: The title of the module.

<server>
Tag: *server*
Xpath: *test-batch/module/server*
Attributes: - *protocol*: optional, string, Hygia currently supports the HTTP and HTTPS protocols
- *domain*: mandatory, string, the domain name or IP address of the location of subject
- *port*: optional, int, the port to connect with (default is port 80 for HTTP and 443 for HTTPS)
Description: A server tag holds the server definition in its attributes.

<log-email>
Tag: *log-email*
Xpath: *test-batch/module/log-email*
Attributes: - *on-error*: optional, int, 1 = log only on validation failures or 0 = (default) always log
Description: When defined, email logging is enabled. The tag holds one or more comma separated email addresses.

<log-file>
Tag: *log-file*
Xpath: *test-batch/module/log-file*
Attributes: - *on-error*: optional, int, 1 = log only on validation failures or 0 = (default) always log
Description: When defined, file logging is enabled. The tag holds the absolute path to the log file. The path should be writable by the user PHP is running as. Log lines will be appended and the log-file is not auto-rotated.

<tests>
Tag: *tests*
Xpath: *test-batch/module/tests*
Children: *test+*
Description: Container for the tests or end-points defined in this module.

Configuration of a Test Item

Each module contains end-point definitions: the Test Items. A Test Item defines the request and describes the response of a client to the subject and Hygia will basically mock a client (like a web browser) and perform the request.

The following properties can be defined in a Test Item:

- A test is by default always enabled, but it is possible to disable a specific test
- The title of the end-point definition
- The path to the end-point, with optional HTTP authentication credentials:
    - user: the user name
    - pass: the password
- Input arguments:
    - Request data properties consisting of:
        - Headers, optionally can be used to set subject specific headers or for instance Cookies
        - Request Variables of type GET or POST, defined as name=value
        - A request body, which – from the XML configuration perspective – is a relative path to a file and has the following properties:
            - Content-Type, necessary for setting the correct header
            - Name, optionally necessary for when mocking a file-upload
    - Input Plug-ins, custom plug-ins can be written to manipulate the input data (see also the chapter about Plugins)
- Output arguments:
    - A Response object, containing response data:
        - Response headers
        - Response body
    - Output Plug-ins, similar to Input Plug-ins, can be used to manipulate the output data (see chapter Plugins)
    - Validators, verify the output data (see the chapter about Validation)

A Test Item with all possible (and completely unrelated) properties, might look like this:

```
<tests>
    <test disabled="0">
        <title>Post Testing</title>
        <path user="bas" pass="bas">/test/access/</path>
        <input>
            <request>
                <headers>
                    <header>Set-Cookie: a=1; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/;</header>
                </headers>
                <vars>
                    <var type="post">name=bas</request-var>
                    <var type="post">pass=mybas</request-var>
                </vars>
                <body type="application/zip" name="userfile">files/example.zip</body>
            </request>
            <plugins>
                <plugin>request.XmlSetSession</plugin>
                <plugin>request.XmlSetCredentials</plugin>
            </plugins>
        </input>
        <output>
            <plugins>
                <plugin>response.XmlGetSession</plugin>
            </plugins>
            <validators>
                <validator type="re">/you have posted, name: bas and pass: mybas/i</validator>
                <validator type="md5">459c637ab65d823aa2064c482a0a5803</validator>
            </validators>
        </output>
    </test>
    ...
</tests>
```

<test>

Tag: *test*
Xpath: *test-batch/module/tests/test*
Children: *title, path, input, output*
Attributes: - *disabled*: optional, integer, 0 = test inactive, 1 = test active (default behavior)
Description: A Test Item defines a single end-point in a subject. The Input and Output elements describe the data stream going through the subject's end-point. The tests element can contain multiple test elements, but should have at least one. Tests are always run sequentially as defined in the XML configuration.

<title>

Tag: *title*
Xpath: *test-batch/module/tests/test/title*
Description: The title of this particular test, like for instance a short description of the use case being tested. Think about tests and titles like: "Signing in with correct credentials", "Signing in with non-existing user name", "Signing in with incorrect password", etc. Used in the reports and real-time information.

<path>

Tag: *path*
Xpath: *test-batch/module/tests/test/path*
Attributes: - *user*: optional, string, may be used in combination with *pass* attribute when HTTP Authorization and Authentication are required for this particular end-point, for example when .htaccess and .htpasswd files are set up.
- *pass*: optional, string, used in combination with *user* to define credentials.
Description: The absolute path on the server, as described in the Module definition, for this particular end-point. The path should of course not include the protocol and domain name and also no GET request variables should be defined here: these are described in the Input element. If this is a protected directory which requires HTTP Authentication, then the *user* and *pass* attributes can be used.

<input>
Tag: *input*
Xpath: *test-batch/module/tests/test/input*
Children: request*?, plugins?*
Description: The Input element describes the input properties of the end-point.

<request>
Tag: *request*
Xpath: *test-batch/module/tests/test/input/request*
Children: *headers?, vars?, body?*
Description: Request is the container for input data like HTTP request headers, request variables and a request body.

<headers>
Tag: *headers*
Xpath: *test-batch/module/tests/test/input/request/headers*
Children: *header+*
Description: The *headers* element contains one or more *header* definitions.

<header>
Tag: *header*
Xpath: *test-batch/module/tests/test/input/request/headers/header*
Description: A *header* tag contains a full request header. If the header already exists, it will be overwritten with the new value. Hygia itself will set a few headers when making a request, overwriting anything that might have been set in the test case: Keep-Alive, Connection, Content-Length and Content-Type (in case of a request body), Authorization (in case of HTTP Authentication and Authorization and the User-Agent, if it is not already set.
Certain subjects require specific headers to be set. Some web-applications use the propriety X-JSON header to send/receive JSON, where-as the message body can then be used for normal HTML.
The header tag can also be used to set cookies, for example:
`Set-Cookie: a=1; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/;`

<vars>
Tag: *vars*
Xpath: *test-batch/module/tests/test/input/request/vars*
Children: *var+*
Description: *vars* is the container tag for the *var* tags: the request variables.

<var>
Tag: *var*
Xpath: *test-batch/module/tests/test/input/request/vars/var*
Attributes: - *type*: optional, string, either GET (default) or POST
Description: One single request variable that is send to the server, using either the GET method (variable will be added to the URL) or using the POST method (variable will be added to the message body). A request-variable has the form: "`name=value`". The value will be URL-encoded.

<body>
Tag: *body*
Xpath: *test-batch/module/tests/test/input/request/body*
Attributes: - *type*: mandatory, string, the content or mime-type of the content. This attribute is mandatory in the configuration because Hygia currently does not guess this.
- *name*: optional, string, when the body represents a file upload, then this is the name.
Description: The message body to be posted to the server. POST request variables should not go here, but in the request-vars as type POST. In the configuration, the value of this tag specifies a file path, relative to the test directory, to where the content is located. The type can be anything, depending on the subject or the functionality of this specific end-point definition. Currently Hygia understand only one file upload. Multipart/form-data conform rfc-1867 is supported, which means Hygia can built requests for file uploads combined with extra form-data.
The body can be of any time, ranging from XML, SOAP and plain-text to propriety binary formats or other complex types – this completely depends on what type of data the

subject needs. Since Hygia will be unable to guess all possible types, the type attribute is mandatory and thus needs to be explicitly supplied.

<plugins>
Tag: *plugins*
Xpath: *test-batch/module/tests/test/input/plugins*
test-batch/module/tests/test/output/plugins
Children: *plugin+*
Description: *plugins* is the container tag for the *plugin* tags. Both input and output can contain plugins.

<plugin>
Tag: *plugin*
Xpath: *test-batch/module/tests/test/input/plugins/plugin*
test-batch/module/tests/test/output/plugins/plugin
Description: The name and optionally directory of a plug-in, for example: `request.XmlSetSession`.
The plug-ins are located in the plug-ins directory below the test directory, so in the case of this example the path and name from the test directory would be: plugins/request/XmlSetSession.plugin.php. The directory (which is also the plug-in type) is added to distinguish more clearly between plug-ins that act on the request and plug-ins that act on the response. For more information on plug-ins, see the chapter Plugins.

<output>
Tag: *output*
Xpath: *test-batch/module/tests/test/output*
Children: *plugins?, validator**
Description: The *output* element describes the output properties of the end-point, or more correctly: it states what the output is expected to be, through the *validator*'s. Headers and body are not defined in the XML, but returned by the server and available to plug-ins through the Response object, as shown in the object hierarchy.

<validators>
Tag: *validators*
Xpath: *test-batch/module/tests/test/output/validators*
Children: *validator+*
Description: Validators is the container of validator definitions.

<validator>
Tag: *validator*
Xpath: *test-batch/module/tests/test/output/validators/validator*
Attributes: - *type*: mandatory, string (enumeration), specifies the type of content validator. Currently, these types are supported:
- `re`: Regular Expression
- `xsd`: XML Schema validation
- `dtd`: Document Type Definition validation (todo!)
- `size`: Validate the size of the response body
- `content-type`: Validate the content-type (as set in the response header) of the content
- `time`: Time validation, allows testing for response times
- `md5`: MD5 hash validation, for testing the validity of binary files

Description: With validators the response of a subject can be verified in several ways, as defined by the *type* attribute. Validators allow to configure how the subject is expected to respond to a pre-defined input. See chapter Validators for more information.
Multiple validators of different types can be defined for a single end-point.

TODO:

(later...)

Plugins

Custom written plug-ins allow manipulation of the input and output data and provides the ability to construct flows.

Difference between input and output plugins (work on Request or Response)

Plugins are chained and run in sequance of creation

Writing plug-ins

How do plug-ins work, methods from the super class (getVar, setVar)

data flows through the plugins, so output type is same as input type (request or response)

The Request and Response objects and their functionality

Flows & sessions

retrieving / inserting / replacing variables

Validators

Validators can verify the response of the server in several ways. Validators exists for schema's, dtd's, connection and transfer times, size validators, hashes, etc.

batch / collection

valid

failures

counts

Starting tests

'mod' url argument: - main test suite dir, - selecting a specific module

Writing tests, tips

do not only write 'positive' tests, but also test for 'negatives'

more do's and dont's

examples of tests

see tests/examples and www/examples

copyrights

hygia, apache, php, open office, adobe acrobat, doxygen, HTTPUnit, iValidator