



USER GUIDE

Last revision date : 23/03/2008

Status/Version : 0.5.1

Contents

1	Introduction	3
1.1	About this document	3
1.2	Audience.....	3
2	GreasySpoon overview	4
3	Software description and requirements.....	6
3.1	Provided package	6
3.2	Service requirements.....	6
4	Installation.....	7
4.1	Preconditions	7
4.2	Installation	7
4.3	Starting service.....	8
4.4	ICAP Client configuration	8
5	Service and scripts administration.....	9
5.1	Service maintenance menu	9
5.2	Scripts Management menu.....	11
6	Scripts development	12
6.1	Scripts definition header.....	12
6.2	Scripts applying on users' requests	13
6.3	Scripts applying on servers' responses	15
7	Notes.....	17
7.1	Service customization	17
7.2	Useful resources for developers	17
7.3	Performances	18

1 Introduction

1.1 About this document

This document presents the installation and configuration of the Grease Spoon service. This service operates in cooperation with a HTTP proxy through ICAP (Netcache from Network Appliance, Bluecoat ProxySG appliance, Squid proxy with ICAP enabled support).

1.2 Audience

This document assumes that the reader has knowledge of Java, HTTP proxies and Linux/Windows server configuration:

- > Java Runtime Environment (JRE) installation/configuration
- > Network configuration under Windows and/or Linux
- > Web proxies architecture / ICAP

It is consequently intended for network engineers, web operation engineers, Java application developers, IT professionals and system administrators who have experience with the following:

- > Installing, configuring and administering Linux server
- > Managing Web traffic and connectivity through a Proxy or Reverse Proxy

2 GreasySpoon overview

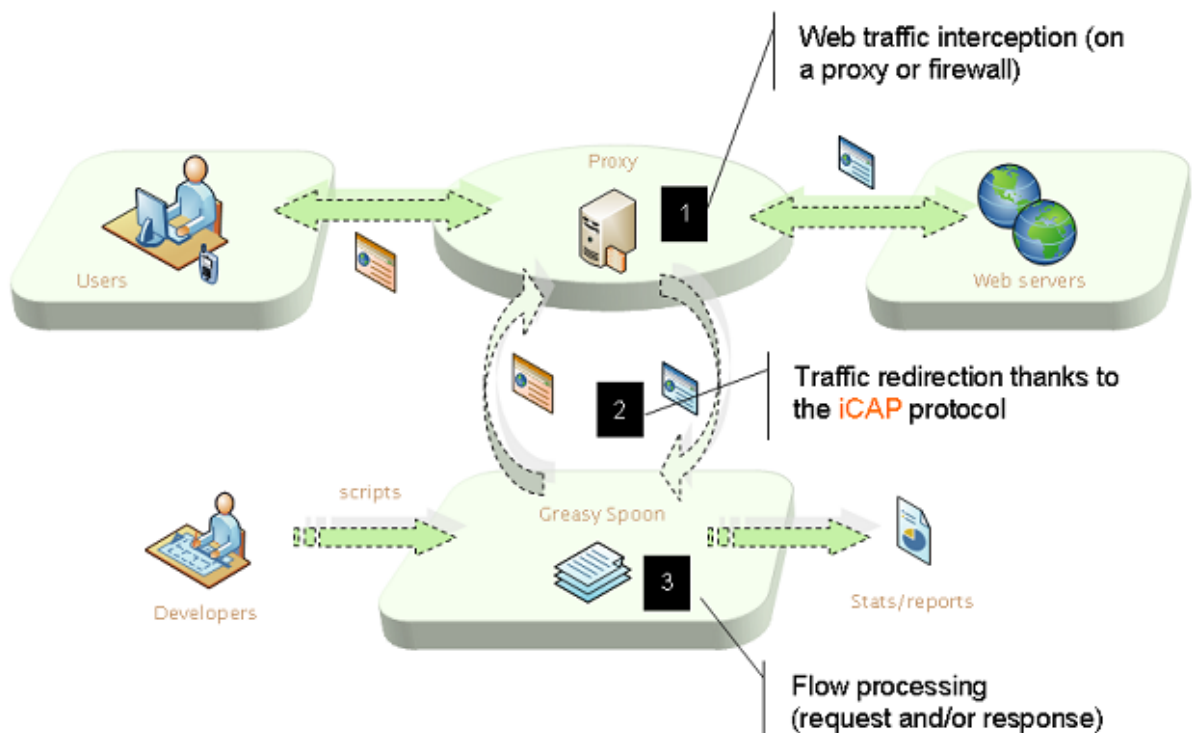
GreasySpoon service aims at improving users experience on web site by providing means to modify/enrichment web sites with content and scripts in an easy way. Using ability to manipulate web content using scripts on the network side, GreasySpoon can be defined as a transposition on the network side of Firefox GreaseMonkey plugin.

GreasySpoon service implementation relies on ICAP (Internet Content Adaptation Protocol, RFC 3507) protocol in order to intercept and modify content. ICAP is a lightweight network API dedicated to content adaptation, and is implemented by both commercial products (Bluecoat, Cisco, F5,...) and open source softwares (Squid, Shweby,...).

ICAP architecture is based on following concept:

- Users' traffic is intercepted by a web proxy (proxy-cache, firewall, security gateway). This web proxy acts as an ICAP client and forward web messages (either users' requests or servers' responses) to one or several ICAP servers.
- The ICAP server(s) has then the possibility to modify web messages to provide value added services (content enrichment, parental/employers' control, antivirus, etc).

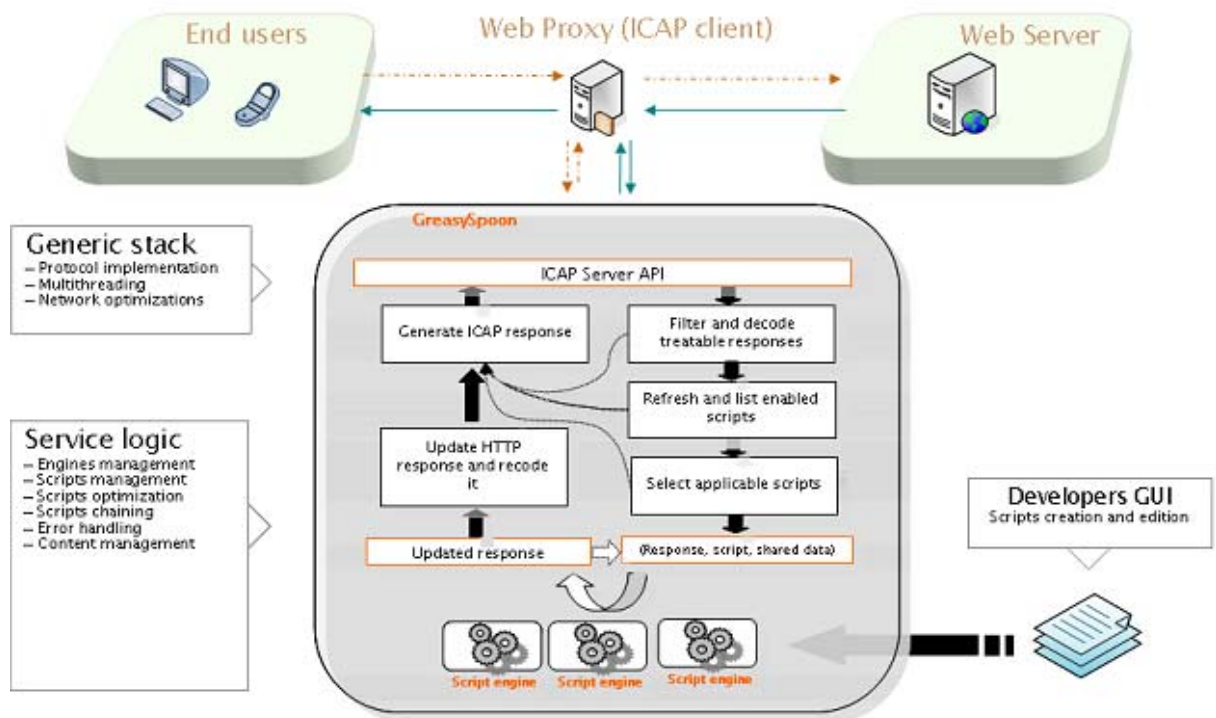
Fig. 1 ICAP architecture overview



Advantages of ICAP architecture are numerous:

- Performances, by using specialized components for each task
- Possibility to scale ICAP clients (proxies) and ICAP servers (services) independently
- Possibility to add/remove/update ICAP servers without service interruption
- Possibility to choice ICAP clients/servers based on the context requirements

Fig.2 GreasySpoon implementation overview



GreasySpoon scripting support is provided using Java JSR223 (<http://jcp.org/en/jsr/detail?id=223>), which defines an API allowing scripting languages to be used in Java Applications. JSR223 is included in Java runtime environment since Java 6 SE platform.

Several scripting engines and languages are available today through JSR223 API:

- ECMAScript (ECMA-262 Edition 3)
- Ruby
- Python
- Groovy
- ...

3 Software description and requirements

3.1 Provided package

GreasySpoon software is provided as a zipped archive called `greasyspoon-[version].zip`. This archive contains the following directories [d] and files [f]:

	Name	Content
d	greasyspoon	service directory
	icapserver.jar	Main application file
d	admin	Web pages for the Web administration interface
d	log	GreasySpoon logs
d	serverscripts	GreasySpoon scripts and libraries
d	conf	Directory containing all configuration files
f	icapserver.conf	ICAP server configuration file
f	greasyspoon.ini	greasyspoon service parameters
f	comments.properties	file allowing to specify comment character for languages used with greasyspoon service
f	services.properties	file containing low-level configuration parameters of GreasySpoon service and GreasySpoon scripts. This file follows Java properties syntax and should not be modified except for service customisation.
f	mime.types	file defining mime types associated to files extensions. This file is used by the web administration interface and follows Apache format.
d	libraries	script languages libraries that can be added to greasyspoon service

3.2 Service requirements

GreasySpoon service is developed in Java, and is packaged with all required libraries. Therefore, requirements should be the following:

- > Any Linux/Windows server with Java JRE 1.6.x runtime environment for GreasySpoon installation
- > A Web proxy/reverse-proxy with ICAP 1.0 support

GreasySpoon has been successfully tested with following configuration:

- > GreasySpoon running environment
 - > Windows XP Pro SP2 - JRE 1.6.0 / 1.6.0_01 / 16.0_02
 - > Windows 2000 Server SP4 - JRE 1.6.0
 - > Debian Sarge - Linux kernel 2.6.6-1-686-smp / JRE 1.6.0_02-b05
 - > Red Hat Linux server 3.2.3-34 - Linux version 2.4.21-15.Elsm - JRE 1.6.0_02-b05
- > ICAP Clients
 - > Network Appliance Netcache v6.0.x and 6.1.x
 - > Bluecoat Security Gateway SG
 - > Squid version 3.0

4 Installation

4.1 Preconditions

Following features must be installed before starting GreasySpoon installation:

- Java runtime environment version **1.6.x** or above (see <http://java.sun.com/javase/downloads/index.jsp>)
- Unzip utility

4.2 Installation

Following naming convention will be used in following chapters:

- **\$javapath**: JRE 1.6 installation directory
- **\$ipaddr**: server IP address on which GreasySpoon will be installed
- **\$gs**: greasyspoon service installation directory
- **\$proxyip**: LAN/Corporate web proxy IP address that should be used for outgoing flows (if any)
- **\$proxyport**: LAN/Corporate web proxy port that should be used for outgoing flows (if any)

Please ensure that all parameters are available before starting installation.

	Action	Comment
Linux	Create a new account for GreasySpoon service <code>adduser greasyspoon</code>	
Linux	Logon in new account <code>su greasyspoon</code>	
All	Copy greasyspoon package locally	
All	Unzip greasyspoon package <code>unzip greasyspoon-[version].zip</code>	
All	Install desired script engines Example for ruby: <code>cd \$gs/libraries/ cd jruby cp *.jar \$javapath/lib/ext/</code>	Generally requires administration privileges on Linux
All	Edit config file <code>vi \$gs/config</code>	
	For javascripts requiring web access and if GreasySpoon service is installed behind web proxies, configure proxyhost and proxyport parameters <code>proxyhost \$proxyip proxyport \$proxyport</code>	
	Configure GreasySpoon ICAP service parameter <code>icap GreasySpoon \$ipaddr 1344 greasyspoon.ini</code>	
	Configure Administration interface IP address. Administration will only be reachable through given IP. <code>admin.ipbounded \$ipaddr</code>	
All (optional)	Edit greasyspoon.ini file Update path parameter with additional libraries that you want to be reachable in your scripts.	Access to specific Script Engines libraries must be provided here
Linux	Edit greasyspoon file <code>vi \$gs/greasyspoon</code>	
	Update javapath, path and pidpath variables accordingly to server locales <code>javapath=\$javapath path=\$gs pidpath=/tmp/</code>	
	Update execution rights of greasyspoon file to 770 <code>chmod 770 \$gs/greasyspoon</code>	

4.3 Starting service

	Starting service	Comment
Linux	As greasyspoon user or as root, run the greasyspoon script: \$gs/greasyspoon start	Greasyspoon script supports the standard switches under Linux: start stop reload restart status
Windows	To run GreasySpoon in background cd \$gs javaw.exe -jar icapserver.jar To run GreasySpoon in foreground \$gs/greasyspoon.bat	

4.4 ICAP Client configuration

ICAP setup on client side depends on the selected hardware/software. Thus, reader should refer to the hardware/software installation/configuration guides.

WARNING: Network Appliance Netcaches includes some specificities in their ICAP implementation. In order for GreasySpoon to work correctly with Netcaches, GreasySpoon service must be declared on Netcaches with following specific icap URI parameter **brand=netapp**. This parameter allows GreasySpoon to identify Netcaches and support their iCAP implementation.

Example of resulting configuration is presented hereafter.

Network Appliance NetCache (version 6.0.6)
<pre>config.icapv1.osl.reqmod_precache = greasyspoon_req config.icapv1.osl.respmod_precache = greasyspoon_resp config.icapv1.farm1.services = icap:// \$ipaddr:1344/greasyspoon_req?brand=netapp on config.icapv1.farm1.attr = greasyspoon_req REQMOD_PRECACHE on rr off weak config.icapv1.farm2.services = icap:// \$ipaddr:1344/ greasyspoon_resp?brand=netapp on config.icapv1.farm2.attr = greasyspoon_resp RESPMOD_PRECACHE on rr off weak config.icapv1.enable = on</pre>

Squid version 3.0 (build with -DICAP_CLIENT)
<pre>icap_enable on icap_preview_enable on icap_service greasyspoon_req reqmod_precache 0 icap://\$ipaddress:1344/greasyspoon_req icap_service greasyspoon_resp respmod_precache 0 icap://\$ipaddress:1344/greasyspoon_resp icap_class class_1 greasyspoon_req icap_class class_2 greasyspoon_resp icap_access class_1 allow all icap_access class_2 allow all</pre>

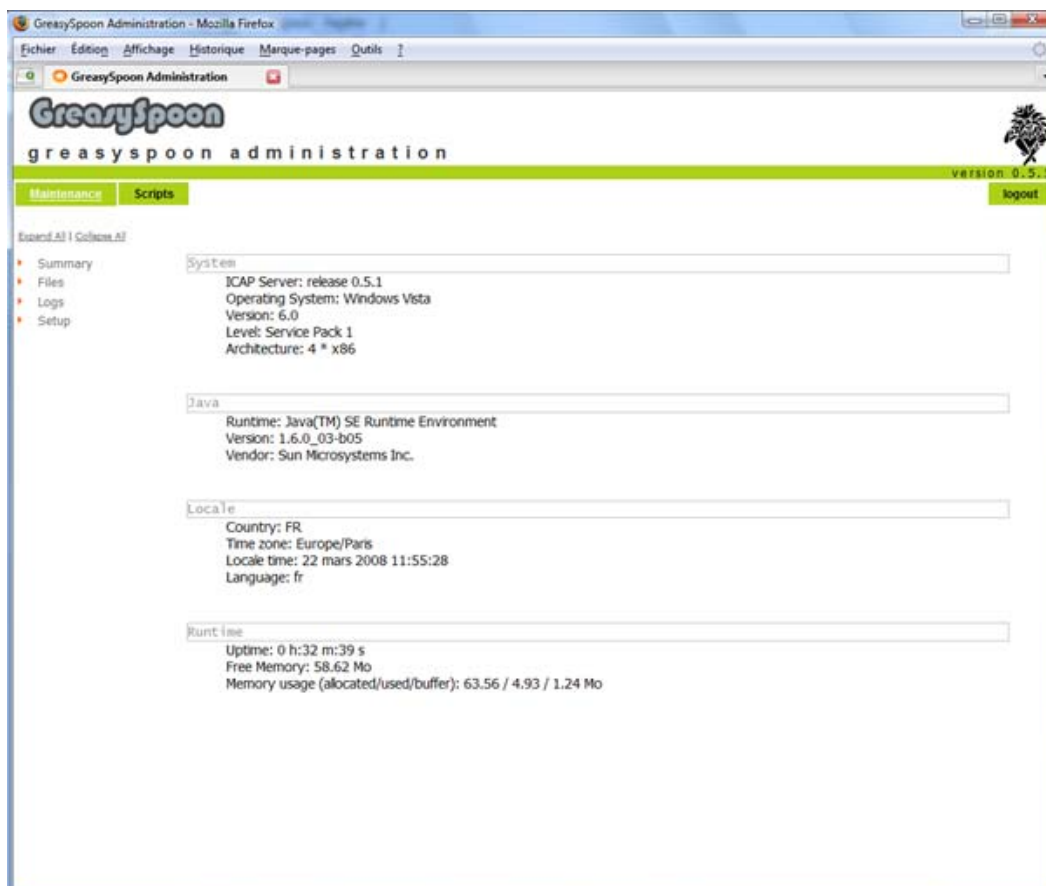
5 Service and scripts administration

Service and scripts administration is done through a web interface.

OS	Service Administration	Comment
All	Using Firefox/IE/..., connect on: http://\$ipaddress:8088 Default login and passwords are admin/admin	Script management can also be made directly using shell. Any change made to scripts (creation/modification/deletion)

Web interface provides two main functions: Greasyspoon service maintenance and scripts management.

Fig. 4 Administration interface



5.1 Service maintenance menu

Maintenance menu provides following features:

Name	Content
☒ Summary	GreasySpoon service information
- Versions	Provides several information about service runtime environment: <ul style="list-style-type: none"> • Operating system • Java environment • Language locale • Service Runtime
- Maintenance	Shows ICAP server configuration (uneditable) Allows to restart ICAP server to take modification on configuration files into account
☒ Files	Service files administration
Log files	Allows to rotate/download/delete log files
Server Files	Allows to edit/download GreasySpoon configuration files. Changes will be applied on next startup or when restarting server through Maintenance menu

☞	Logs	Log files consultation. Provide access up to the 400 log last lines.
-	Access	<p>Logs every ICAP requests processed by the service with following fields:</p> <ul style="list-style-type: none"> • Date (human readable) • Date (absolute time) • - (empty field) • Overall processing time in ms • ICAP call mode (REQMOD or RESPMOD) • ICAP service applied (greasyspoon) • ICAP Response code • Protocol (HTTP/) + request method in REQMOD or response code in RESPMOD • Requested URL (URL parameters are not logged) • [mime-type] in RESPMOD • [scriptname: script processing time in ms] List of applied scripts
-	Server	<p>Logs server events and accesses on web administration interface:</p> <ul style="list-style-type: none"> • Date (human readable) • Date (absolute time) • Log level • [Cause] • [Source IP address for web accesses] • Event
-	Service	<p>Contains events relative to scripts: modifications / scripts errors / ...</p> <ul style="list-style-type: none"> • Date (human readable) • Date (absolute time) • Log level • Script generating the event • [script processing time in ms] (only for valid processing) • Event or processed URL • [Exception raised by script]
-	Debug	<p>Used for GreasySpoon service debug. Logs detailed processing information with following fields:</p> <ul style="list-style-type: none"> • Date (human readable) • Date (absolute time) • Log level • Event/Messages • [Exception raised by service] <p>Debug log are activated if log Level is set from FINE to ALL. Warning: activating debug logs highly impacts server overall performances.</p>
☞	Setup	Configuration of administration web server and logs parameters. Changes made through setup are directly applied (no restart needed).
-	Administration	<p>Allows to modify web administration interface parameters:</p> <ul style="list-style-type: none"> • IP address: allows to restrict administration accesses on given address. This feature allows to run administration server on a dedicated network interface/VLAN. • Server port (default:8088) • Simultaneous connections: number of simultaneous requests that can be handled by the administration server • backlog: queue size for pending requests • Administration page: directory containing administration html pages • Over SSL: requires specific installation with a certificate keystore. Not described here. • Administration password: allows to change current password with a new one
-	Logs	Allows setting logs repository and logging level.

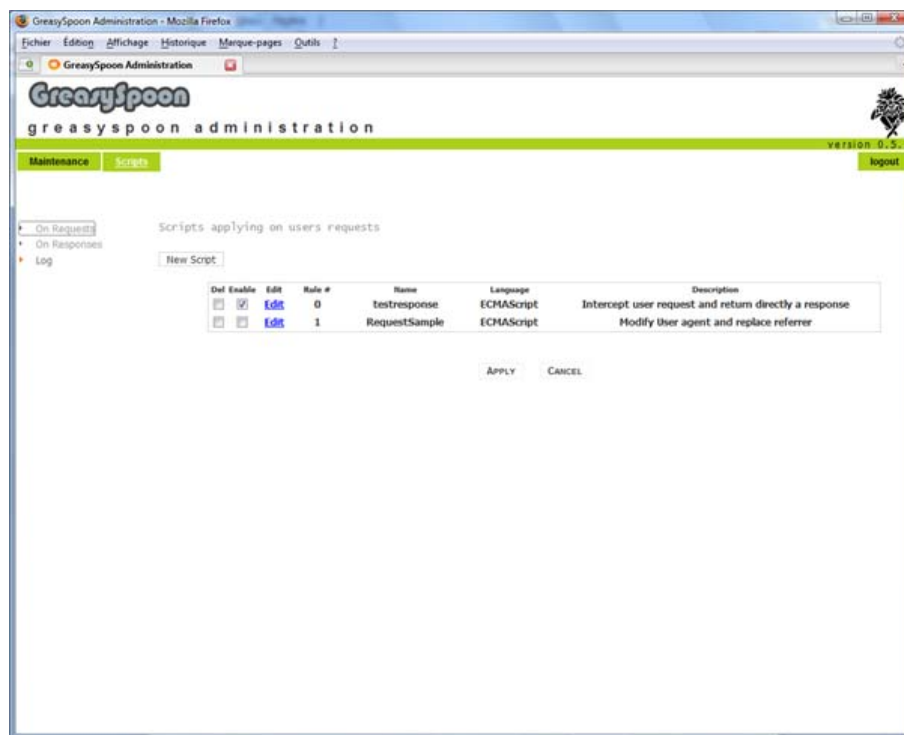
-	Users	<p>Allows creating accounts for web administration access other than admin account. New accounts can be set with different access rights:</p> <ul style="list-style-type: none"> - ADMIN: administration access, with any operation accepted - USER: GreasySpoon developer access. Account provides access only to Scripts menu, and scripts deletion is limited to scripts created by USER accounts. - NONE: equivalent to a supervision account; only provides access to Scripts menu but without modification rights.
---	-------	---

Note that master "admin" account cannot be deleted or overwritten.

5.2 Scripts Management menu

Scripts management menu provides facilities to create, edit, manage and delete GreasySpoon scripts.

Fig. 5 Management interface for scripts applying on server responses



	Name	Content
☞	On requests	<p>List existing scripts that modify end-users requests. Allows to:</p> <ul style="list-style-type: none"> • Create new scripts using one of the installed scripting languages • Enable/disable services on the fly • Edit existing/new scripts online • Delete scripts <p>Any validated change is applied immediately.</p>
☞	On responses	<p>List existing scripts that modify servers responses. Allows to:</p> <ul style="list-style-type: none"> • Create new scripts using one of the installed scripting languages • Enable/disable services on the fly • Edit existing/new scripts online • Delete scripts • Any validated change is applied immediately.
☞	Log	<p>Link to Service logs provided through Maintenance menu (see previous chapter). Log events relative to scripts (reloads, errors, exceptions ...).</p>

6 Scripts development

Scripts creation/edition is provided through "Scripts" menu of web administration interface. Two kind of scripts can be defined:

- Scripts applying on users requests: These scripts allow intercepting and modifying requests made by users to the content providers. Example of scripts that can be done are:
 - Requests header enrichment with custom parameters, like user id/name
 - Headers anonymisation like removal of Refer to improve user privacy
 - Request redirection from one site to another to avoid 302 messages
 - Response return, like a deny message for parental/employees' control
 - ...
- Scripts applying on server responses: These scripts allow intercepting and modifying responses provided by content providers. Example of scripts that can be done are:
 - Content removal (ads/unwanted content)
 - Content adaptation based on user's device information provided in initial request
 - Scripts insertions to improve users' experience
 - Mashups
 - ...

6.1 Scripts definition header

Creating a new script is made through the web interface, using the "New Script" button:

- Using "Scripts"=>"On requests" => "New Script" for scripts that will apply on users requests
- Using "Scripts"=>"On response " => "New Script" for scripts that will apply on servers responses

GreasySpoon scripts behavior is managed using a specific script header, characterized by:

- A starting tag **==ServerScript==**
- A closing tag **==/ServerScript==**

When creating a new script through the web interface, an default header is automatically generated.

This header is composed of following tags (default values are in bold):

- @name <String> : mandatory header, unique name used to characterize script
- @status <on|off> : optional header, allows to enable/disable script. Note that enabling/disabling the script can also be done using scripts list available through the web interface.
- @timeout <integer>: optional header, allows to configure processing threshold (in ms) after which script will be aborted. This value cannot exceed GreasySpoon overall processing time threshold, which is set globally by the administrator.
- @order <integer> : optional header, used to configure scripts execution order. Note that in case of values conflict, script with same order value will be chosen randomly.
- @description <String> : optional string used to provide a short description of what the script does
- @include <regular expression> : mandatory header, compared to server URLs to see on which sites (URLs) the script will applied. Regular expressions must follow PERL syntax (see [Mastering Regular Expressions, 2nd Edition, Jeffrey E. F. Friedl, O'Reilly and Associates, 2002](#) for more details). Several @include tags can be inserted in the header.
- @exclude <regular expression> : mandatory header, compared to server URLs to see on which sites (URLs) the script will **NOT** applied. Several @exclude tags can be inserted in the header. Note that exclude regexes are evaluated before @include regexes.

Example of such header is provided hereafter.

//----- ----	
//This is a GreasySpoon script.	
//----- ----	Custom comments
//WHAT IT DOES:	
// Simple script inserting "Hello world" on top of the page	
//----- ----	
//==ServerScript==	Header start tag
//@name scriptsample	Script unique name
//@status off	Script status (here the script is deactivated)
//@order 0	Requested order: here the script should be applied first
//@timeout 2000	Execution threshold: script will be aborted if processing exceeds 2 seconds (2000 ms)
//@description script example using ECMAScript	Script description
//@include .*	Applies per default on any URL
//@exclude http://.*\.\francetelecom\.{2,3}/.*	Except on sites in francetelecom (.fr, .com, ...) domains
//@exclude http://.*\.\orange\.{2,3}/.*	An except on sites in orange domains
//==/ServerScript==	Header end tag

6.2 Scripts applying on users' requests

GreasySpoon service provides following information to scripts applying on users requests:

- **requestheader** : string containing HTTP request header, as generated by user's browser
- **httprequest** : string containing HTTP request body, or null if none. Note that HTTP request body is only provided for POST requests.
- **user_id** : string containing either user id (login) or user ip address. User ID will be available if authentication is activated on HTTP proxy.
- **user_group** : string containing user group if available, or null if not. User group might be available if authentication is activated on HTTP proxy.
- **sharedcache** : a java hash table<String, Object> that is shared between all scripts

Information is provided as static variables so can be accessed/modified in any part of the script.

All information is provided as string to avoid casting on Script Engine side, except the shared cache which is transmitted as a Java hashtable. To use this hash table, standard Java methods must be used:

- **get(<string>key)** to retrieve content associated to given key
- **put(<string>key, <object>value)** to store a content in the table using given key

Any modification made on HTTP request header and/or body will be handled by GreasySpoon service and will be reflected to the user's request.

A request anonymizer script can be proposed as a first example of request modification that illustrates scripts possibilities:

//----- ----	Developer comment
//This is a GreasySpoon script.	
//----- ----	
//WHAT IT DOES:	

```
// - change user agent
// - remove browser referer
// - To test with http://www.ericgiguere.com/tools/http-header-viewer.html
//-----
//==ServerScript==
//@name      RequestSample
//@status on
//@description Request anonymizer
//@include   http://www.ericgiguere.com/.*
//==/ServerScript==
//
//Available elements provided through ICAP server:
//-----
// requestheader : (String)HTTP request header
// httprequest   : (String)HTTP request body
// user_id       : (String)user id (login or user ip address)
// user_group    : (String)user group or user fqdn
// sharedcache  : (hashtable<String, Object>) shared table
between all scripts
//-----
headerstring = "User-Agent: ";

i = requestheader.indexOf(headerstring ) + headerstring .length;
il = requestheader.indexOf("\r\n", i);

requestheader = requestheader.substring(0,i)
                + "GreasySpoon/1.0 (GreasySpoon; fr; rv:0.1.0.0)
Gecko/20070914"
                + requestheader.substring(il);
headerstring = "Referer";
i = requestheader.indexOf(headerstring);
if (i>0){
    il = requestheader.indexOf("\r\n", i) +2;
    requestheader = requestheader.substring(0,i) +
requestheader.substring(il);
}
}
```

Script control header

Information provided by GreasySpoon service

Locate User-agent header value

Search header end of line

Replace user agent value by a new one

Search if request includes a Referer

If yes, simply remove it

GreasySpoon service also allows to directly provide a response to end-user based on his request. A script denying access to `forbiddendomain.com` can be proposed as an example of direct response:

```
//-----
//This is a GreasySpoon script.
//-----
//-----
//WHAT IT DOES:
//
//-----
//==ServerScript==
//@name      directresponse
//@status off
//@description Intercept user request and return directly a
response
//@include   http://www.forbiddendomain\.{2,3}/.*
//@exclude
//==/ServerScript==
//
//Available elements provided through ICAP server:
//-----
//requestheader : (String)HTTP request header
//httprequest   : (String)HTTP request body
//user_id       : (String)user id (login or user ip address)
//user_group    : (String)user group or user fqdn
//sharedcache  : (hashtable<String, Object>) shared table
between all scripts
//-----
requestheader="HTTP/1.1 200 OK\r\n"
                +"Content-Type: text/html;charset=ISO-8859-1\r\n\r\n";

httprequest   = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
Transitional//EN\"><html><head><title>direct response
example</title></head>"
                + "<body><h1>!! You should go back to Work !!</h1>"
                + "</body>"
                + "</html>";
```

Apply script on forbiddendomain domain

Generates directly a HTTP response header

Generated a HTTP response body with a message to user

6.3 Scripts applying on servers' responses

GreasySpoon service provides following information to scripts applying on server responses:

- **requestheader** : string containing HTTP request header, as generated by user's browser
- **responseheader** : string containing HTTP response header
- **httpresponse** : string containing HTTP response body
- **user_id** : string containing either user id (login) or user ip address. User ID will be available if authentication is activated on HTTP proxy.
- **user_group** : string containing user group if available, or null if not. User group might be available if authentication is activated on HTTP proxy.
- **sharedcache** : a java hash table<String, Object> that is shared between all scripts

Like for requests scripts, information is provided as static variables so can be accessed/modified in any part of the script.

All information is provided as string to avoid casting on Script Engine side, except the shared cache which is transmitted as a Java hashtable. To use this hash table, standard Java methods must be used:

- `get(<string>key)` to retrieve content associated to given key
- `put(<string>key, <object>value)` to store a content in the table using given key

Any modification made on HTTP response header and/or body will be handled by GreasySpoon service and will be reflected to the final response provided to the user. Please note that HTTP request header is only provided for potential scripts usage and that changes made on it will be bounded to the script execution context.

In order to illustrate scripts possibilities in response mode, a first basic hello world script in JavaScript can be proposed:

```
//-----
//This is a GreasySpoon script.
//-----
//WHAT IT DOES:
//    - find body tag
//    - insert message after body tag
//    - insert custom header in response
//-----
//==ServerScript==
//@name          sample
//@status off
//@description   Javascript example
//@include       .*
//@exclude       http://.*\s.excludedomain\.{2,3}/.*
//==/ServerScript==
//
//Available elements provided through ICAP server:
//-----
//requestheader : (String)HTTP request header
//httprequest   : (String)HTTP request body
//responseheader : (String)HTTP response header
//httpresponse  : (String)HTTP response body
//user_id       : (String)user id (login or user ip address)
//user_group    : (String)user group or user fqdn
//sharedcache   : (hashtable<String, Object>) shared table between all
scripts
//-----
//Use cache to store a request counter
i = sharedcache.get("counter");
i++;
//Insert Hello after body tag
bodypos =
httpresponse.indexOf(">",httpresponse.toLowerCase().indexOf("<body")) +1;
httpresponse = httpresponse.substring(0,bodypos )
                + "<h1>Hello World n°"+i+" !!!</h1>"
                +httpresponse.substring(bodypos);
sharedcache.put("counter", i);
```

Apply script
on any URL
except for
exclude_domain
ending with
any TLD

Locate body
tag
Insert Hello
World after
body

A more complex example that takes advantage of ruby libraries can be proposed hereafter. This script demonstrates ruby Hpricot library capacity so manipulate HTML content in a simple way.

```

#-----
#This is a GreasySpoon script.
#To use it, you need
#   - jruby
#   - hpricot ruby library
#-----
#==ServerScript==
#@name           CustomizePage
#@description    Customize test page by replacing/changing
content
#@status off
#@include        http://www\.targetdomain\.{2,3}/.*           Apply script on target
                                                                domain only

#==/ServerScript==
#-----

require 'hpricot'                                           Libraries used by script
require 'open-uri'

def process(httpresponse)                                    Content modification
                                                                function
    doc = Hpricot(httpresponse)                             Parse doc using Hpricot
                                                                library

    #change background color
    (doc/"body").set("bgcolor", "#ffffff")                 Change page background
                                                                color

    doc1 = Hpricot(open('http://www.anotherwebsite.fr', :proxy =>
'http://myproxy:3128'))                                    Load page from another web
                                                                page

    (doc/"#res").prepend((doc1/"#banner").to_html)         Extract content and
                                                                insert it into page

    (doc/"#search").set("href", "http://www.voila.fr")    Change search engine
                                                                link

    return "#{doc}"                                         #return modified content
End

$httpresponse = process($httpresponse)                     Main method: just call
                                                                modification function

```

7 Notes

7.1 Service customization

Service is provided with a standard configuration file that should match most deployment environments.

However, following parameters in /conf directory can be easily modified to adapt service to specific environment.

File	Parameter:default value	Comment
greasyspoon.ini	maxtimeout = 5000	GreasySpoon service threshold. Whole scripts processing time must remain lower than this threshold or processing will be aborted (modification already done by scripts is kept)
	errorthreshold = 10	Scripts that failed successively a greater number than this threshold are automatically turned off.
services.properties	GreasySpoon.mimesupported=html xml txt	Responses mime types that are transferred to GreasySpoon service. Others mime types responses are discarded. Mime types can be modified or enriched by adding new values separated by [space] character.
	GreasySpoon.compress=on	Set if GreasySpoon service automatically compresses all responses using gzip to browsers that support it. If set to false, only content that was originally compressed will be compressed.
	SpoonScript.reqheadparam=requestheader	Name of the variable sent to scripts and containing request header
	SpoonScript.reqbodyparam=httprequest	Name of the variable sent to scripts and containing request body
	SpoonScript.repheadparam=responseheader	Name of the variable sent to scripts and containing response header
	SpoonScript.resbodyparam=httpresponse	Name of the variable sent to scripts and containing response body
	SpoonScript.useridparam=user_id	Name of the variable sent to scripts and containing user identifier
	SpoonScript.usergroupparam=user_group	Name of the variable sent to scripts and containing user group
	SpoonScript.cacheparam=sharedcache	Name of the variable sent to scripts and containing shared cache
	SpoonScript.icapuserheader=x-authenticated-user	Name of ICAP header from which user identifier will be extracted
	SpoonScript.userfallbackheader=x-client-ip	When above ICAP header is not provided, name of header from which user identifier will be extracted. Header lookup is made in following order: ICAP header, HTTP request header then HTTP response header.
	SpoonScript.icapgroupheader=x-authenticated-groups	Name of ICAP header from which user group will be extracted
	SpoonScript.groupfallbackheader=user-agent	When above ICAP header is not provided, name of header from which user group will be extracted. Header lookup is made in following order: ICAP header, HTTP request header then HTTP response header.

7.2 Useful resources for developers

For Ruby:

- <http://ruby-doc.org/docs/ProgrammingRuby/>
- <http://www.ruby-doc.org/stdlib/>
- <http://code.whytheluckystiff.net/hpricot>

For ECMAScript:

- http://developer.mozilla.org/en/docs/Rhino_documentation

7.3 Performances

The following graphs summarize GreasySpoon performances measured with basic scripts.

Tests were made using:

- An IBM xs336, 2*2.8Ghz Xeon, 1.5 Go RAM running GreasySpoon service
- GreasySpoon v0.4.2RC3 release
- Java JRE 1.6 (build 1.6.0_03-b05) with options `-Xms512m -Xmx1024m -XX:+AggressiveOpts -XX:+UseParallelGC -XX:+UseBiasedLocking`

All following tests were running:

- A simple request enrichment script developed in **ECMAScript**

```
i = requestheader.indexOf("\r\n\r\n");
requestheader = requestheader.substring(0,i) + "\r\nX-Customheader: greasyspoon" + requestheader.substring(i);
```

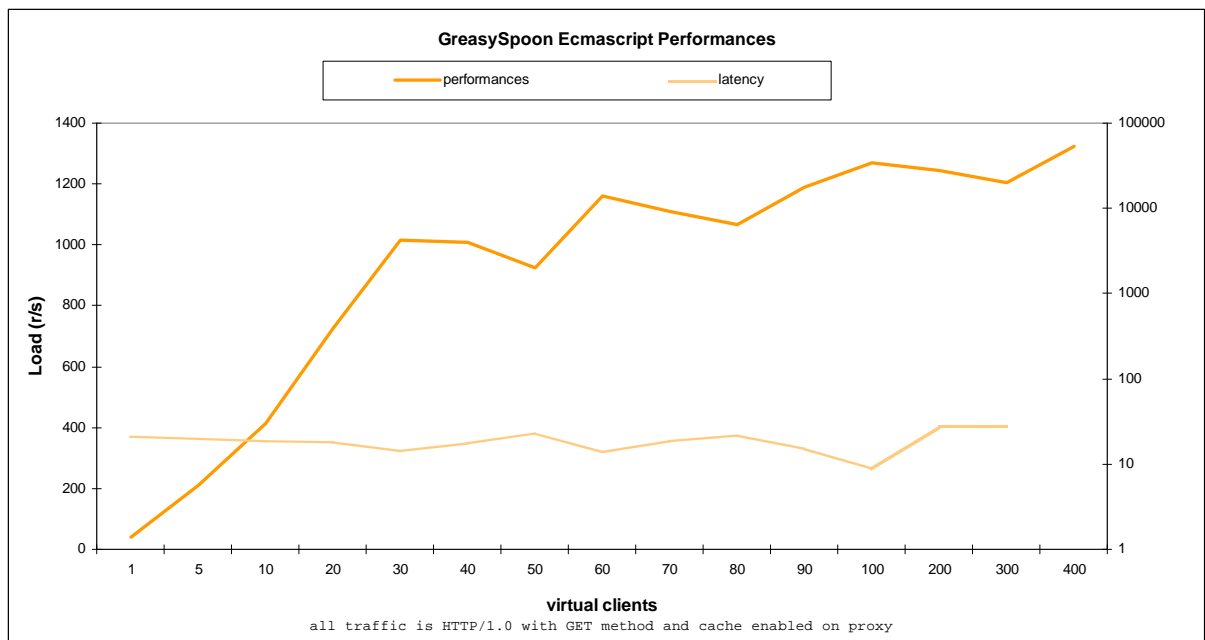
- A simple "Hello world" response enrichment script using one of the following languages:
 - **ECMAScript**

```
a1 = httpresponse.toLowerCase().indexOf("<body");
a2 = httpresponse.indexOf(">",a1)+1;
httpresponse = httpresponse.substring(0,a2)+"<h1>Hello JS World !!!</h1>"
+httpresponse.substring(a2);
```

- **Ruby**

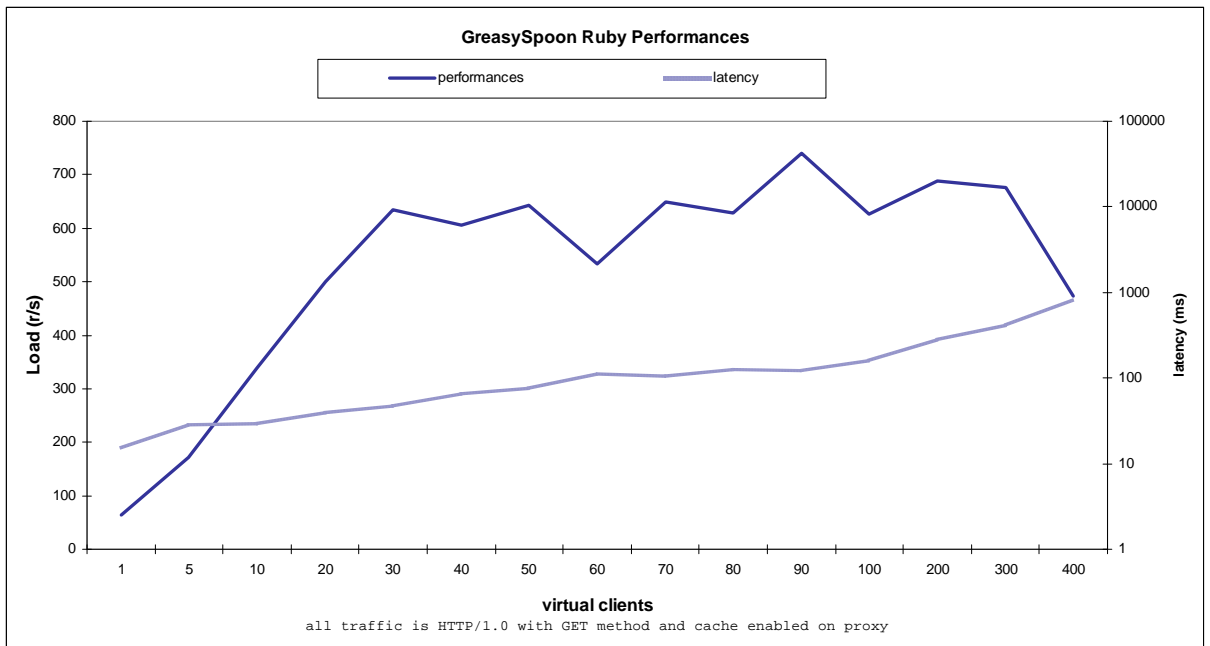
```
bodypos = $httpresponse.downcase.index("<body")
header = $httpresponse.slice!(0, $httpresponse.index(">",bodypos) + 1 )
header<<"<h1>Hello Ruby World 2.0 !!!</h1>"
header<< $httpresponse
$httpresponse = header
```

Performance of ECMAScript scripts is presented hereafter.



These results show an optimal performance of 1000 r/s with latency less than 20 ms.

The next graph presents performance of Ruby "hello world" script associated to the simple request ECMAScript script.



These results show an optimal performance of 500 r/s with latency less than 40 ms.