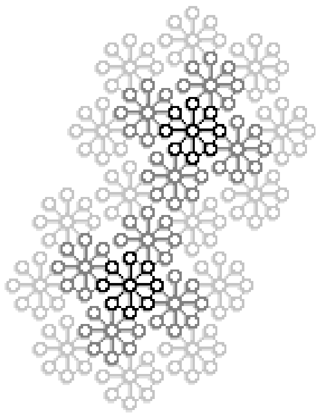
G-WAN

Global-WAN

Version 5 for Linux
(tested with Debian and CentOS 32-bit and 64-bit)



User's Manual

“Simplicity is the ultimate sophistication.”
(Leonardo da Vinci, 1452-1519)

“Complexity is the enemy of security.”
(Bruce Schneier, 1963-)

“The price of reliability is the pursuit of the utmost simplicity.”
(Sir Charles Antony Richard Hoare, 1934-)

“Today's scientists have substituted mathematics for experiments, and they wander off through equation after equation, and eventually build a structure which has no relation to reality.”
(Nikola Tesla, 1856-1943)

This manual has been published to help users understand and use the design and features of the G-WAN application server.

For comments and suggestions, please contact the authors at http://gwan.ch/

Table of Contents

Why G-WAN?

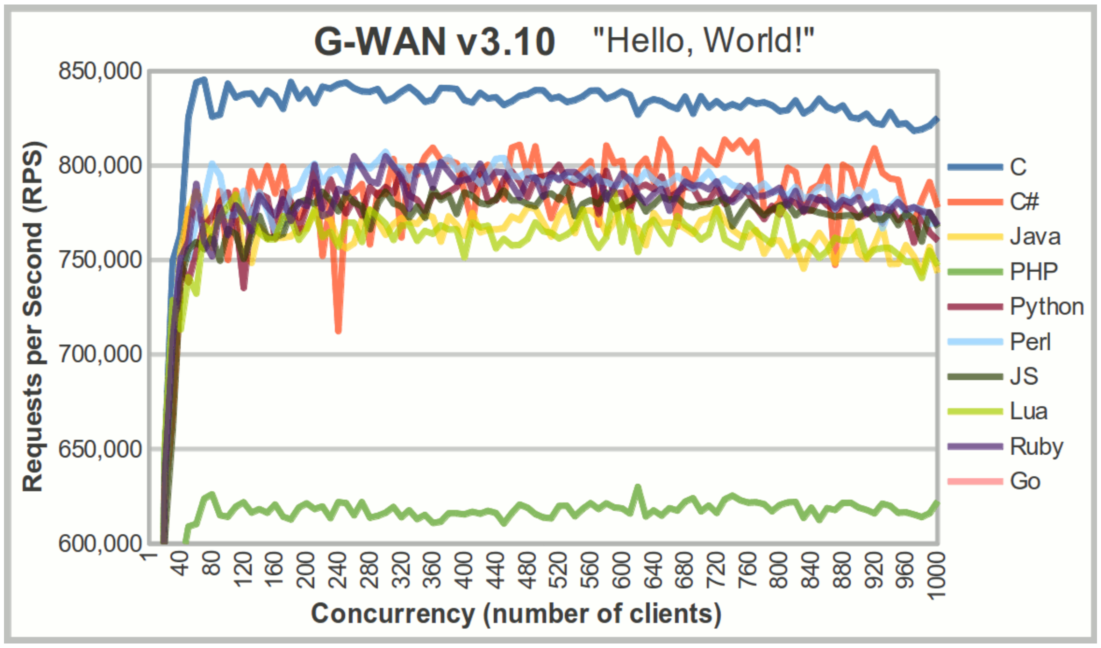
I wrote computer software for the past 30 years, starting in assembly language because only BASIC was available at the time and it quickly show its limits.

As TWD Industries AG's CEO, a company I founded in 1998, I decided to write G-WAN in late 2008 because other servers use cryptic configuration files and programming interfaces, on the top of requiring a datacenter and a team of sysadmins to merely do basic things.

We needed it to scale vertically to exploit parallelism. We wanted to deploy applications by running a single executable – and we wanted it to run optimally and safely out-of-the-box.

> G-WAN, is a *freeware*. It is an attempt to fill the gap for those who feel that *"productivity gains"* means that developers can waste *less* time and money – not more and more.
>
> You can contribute with feedback (use cases, bug reports, ideas, code, blogs) and with support plans. All contributions, as modest as they can be, help to enhance G-WAN.
>
> I believe that G-WAN adds fair value to the field because of its willingness to explore new directions and its nonpartisan support of competing virtual machines (very few application servers support competing virtual machines like Oracle's JVM and Microsoft's CLR).

This chart shows G-WAN running many "hello" scripts on a 6-Core Xeon Mac Pro:



ANSI C and C++ achieve this performance without caching. Others require caching to fly that high. G-WAN caching is disabled by default, but you can enable it globally or on a per request basis. Whatever your needs, G-WAN makes things easier, faster, and safer.

This manual will tell you how to achieve that, and much more.

Productivity vs. Performance

Most organizations don't have enough financial power to enjoy having the choice between hiring competent C engineers and/or building a datacenter.

Web developers need a prototype to convince others (customers, partners, investors) that their idea is worth paying for. And here PHP or Java will do the job: the first problem to solve is not to scale: it is to have something to sell in the first place.

Whether you use PHP, Java or C, G-WAN does it better, faster, and cheaper. But G-WAN is also easing the collaboration between PHP or Java with C – *doing all from one single tool.*

This allows you to use each programming language for what it does best – depending on the task (user-interface might be done better in PHP, but image processing will fly higher in C).

G-WAN has been designed to let developers and engineers focus on their projects rather than on system and hardware issues:

1. scripts in **16 languages** (C/C++, Java, C#, PHP...)
2. near-optimal **performance** and **multicore scalability**
3. a **low CPU/RAM usage** – even under heavy loads
4. a programmable **reverse-proxy** & **elastic load-balancer**
5. ease of use: **zero-configuration** (no configuration files)
6. **simple interfaces**: 7-line hello.c vs 120-line Nginx module
7. **no security breach** since the first release in year 2009.

Computers hate complexity almost as much as humans do. G-WAN makes them all a favor in this matter: to try G-WAN, just uncompress the downloaded archive and execute `./gwan`

It took us years to write, test and tune G-WAN. We did it to help yourselves. But we wrote this documentation and we maintain the http://gwan.ch/ web site to help you too.

Summary

This document consists in the following sections:

The G-WAN HTTP implementation is *probably safer* than others for at least two reasons:

– it contains thousands of times less lines of code (it also means less bugs);
– HTTP parsing works without libraries and buffer copies (no more exploits).

These unusual choices also make G-WAN unusually efficient.

With static contents, G-WAN is fast with large files, but, it will really shine with small files: web servers don't *receive* or *send* data: operating systems do it. So, if you serve a 1 MB file, the CPU time used by G-WAN is negligible as compared to the time used by the OS kernel to send data on the network. Result: you hit the limits of the kernel, not of the Web server.

> With small *static files* and HTTP keep-alives (to avoid the TCP handshake bottleneck), like with *dynamic contents* (to avoid the disk bottleneck), G-WAN can shine as it does *a larger part of the work* – especially on localhost (to remove the network bottleneck).

Optimization is not pointless: it allows you to run more tasks – on less and lower-specs machines. G-WAN inflates the margin of your business by reducing your fixed costs.

G-WAN fuels the http://gwan.ch/ Web site since its fist public release on June 30th, 2009, initially on Windows, and six months later on Linux. Since then, no vulnerability was found, despite constant (and sometimes clever) attacks.

If you find a problem then send us the details so we can take action without delay.

All contributions will receive full credits on G-WAN's Web site. And because the only goal pursued here is to make progress, you can count on a prompt reply.

I. The web server

Installation and configuration

To install G-WAN, download it from http://gwan.ch, copy the compressed archive in a folder of your disk (like /home/username/gwan), decompress it and run the ./*gwan* program.

> To configure Linux in order to *let* G-WAN run at full speed read http://gwan.ch/source/ab.c.txt (Linux and Windows TCP/IP tuning options to use all the potential of your hardware)

Decompressing the archive will create the following sub-directories (only the `www` sub-folder is mandatory, all other folders are optional):

```
/ gwan / 0.0.0.0_8080 / #0.0.0.0 / www        for HTML and image files
   |            |          |      / logs       for log files
   |  listening on this    |      / handlers   for Handlers  (Chapter 3)
   |  interface (any) and  |      / csp        for C scripts (Chapter 2)
   |  port number 8080     |      / cert       for SSL certificates
   |                       |
   |                    Host name (like domain.com)
where the gwan          or IP address (like 192.168.2.4)
executable is           (one “#”-prefixed root host and
located                 several “$”-prefixed virtual hosts)
```

To test the server, run G-WAN and enter the following URL in a web browser: `http://127.0.0.1:8080/`

By default, G-WAN listens on the 8080 non-privileged port, on *all* interfaces (0.0.0.0).

For each G-WAN *listener* tied to an interface and port, you will define one single *root host* and, optionally, one or several additional *virtual hosts*:

```
listener 1: / gwan / 192.168.2.4_80  / #trustleap.ch       (root host)
                                     / $gwan.ch            (virtual host)

listener 2: / gwan / 192.168.4.8_80  / #trustleap.com      (root host)
                                     / $gwan.com           (virtual host)
```

A *listener* receives incoming connections on a given network interface and port number.

A *root or virtual host* defines a Web site attached to a listener.

Many virtual hosts can be attached to a single listener. The listener finds which virtual host it must serve by looking at the ‘Host’ HTTP header (required by HTTP/1.1).

If such an HTTP header is not found (or not specified by a HTTP0.9/1.0 client) then the “#”-prefixed *root host* of the corresponding listener is used (as HTTP/1.1 clients *must* specify the Host header, G-WAN will return the HTTP error 400 if none is found).

> Why not use configuration files, like all other HTTP servers? A single source of information (vs. configuration files and directory names) prevents unnecessary inconsistencies and errors – and spares you the learning curve needed to become a “specialist” (at merely *using* a program).

Host Aliases

An alias lets you assign additional domain names to an existing (root or virtual) host. Like for other hosts, you just have to create a folder, but its contents (if any) are ignored:

```
/ gwan / 192.168.2.4_80/ #gwan.ch                (root host)
                       / #gwan.ch:gwan.com       (alias)
                       / #gwan.ch:trustleap.ch   (alias)
                       / #gwan.ch:trustleap.com  (alias)
```

And, to define an alias for a virtual host:

```
/ gwan / 192.168.2.4_80/ #gwan.ch                      (root host)
                       / $forum.gwan.ch                (virtual host)
                       / $forum.gwan.ch:forum.gwan.com  (alias)
```

An alias uses the following syntax: real_host **:** alias_host

> The above will not let you reach contents from the IP address that has been assigned to all those domain names: G-WAN will always reply “404: Not found” – even if you setup an alias called #gwan.ch:1.2.3.4. This is because IP addresses are not valid host HTTP headers.

If you want to let G-WAN reach your Web site from its IP address, you have do this:

```
/ gwan / 192.168.2.4_80/ #1.2.3.4                (root host)
                       / #1.2.3.4:gwan.ch        (alias)
                       / #1.2.3.4:gwan.com       (alias)
                       / #1.2.3.4:trustleap.ch   (alias)
                       / #1.2.3.4:trustleap.com  (alias)
```

HTTP Authentication

G-WAN supports the BASIC and DIGEST HTTP authorization schemes (RFC 2617).

The auth_basic.c example shows how to assign passwords to users and URIs. Credentials can be stored in a G-WAN handler or servlet, a local or remote database, or an LDAP server. Here is an example how this can be organized:

```
“acl”: {
“roles”: [
   {“name”:”admin”,    “description”:”full-access to all”},
   {“name”:”guest”,    “description”:”restricted-access to all”}
]
“rights”: [
   {“uri”:”/?auth_digest”,   “auth”:”DIGEST”,    “method”:”*”,      “role”:”admin”},
   {“uri”:”/?auth_basic”,    “auth”:”*”,         “method”:”GET”,    “role”:”guest”}
]
“users”: [
   {“name”:”paul”,  “role”:”admin”,  “HA1”:”a34b...78d1”},
   {“name”:”tom”,   “role”:”guest”,  “HA1”:”962f...eb51”}
]
}
```

“**roles**” are profiles associated with “**users**” to define “**rights**”: who can access an “**uri**”, using which HTTP authorization method (BASIC, DIGEST) and HTTP request method (GET/POST/PUT, etc.). A star character (wildcard) allows any method.

The “HA1” field is the RFC 2617-defined MD5(user:uri:password) hash.

You can define access rights for different applications by defining dedicated “roles” and by associating any related “uri” to the dedicated “roles”:

```
Application: shopping
“shop_admin”
“shop_guest”

Application: accounting
“accnt_admin”
“accnt_guest”
```

To keep things simple, it may help to reduce the number of Web applications used on a single root or virtual host (use sub-domain DNS entries like forum.gwan.com).

DIGEST (RFC 2617) was designed in 1999 by Verisign (SSL Certificate Authority) and Microsoft (the SSL CA repository, in charge of CA queries and CA validation). BASIC and DIGEST are said to be safe only if you delegate your security to SSL certificate providers.

Log files

G-WAN can use traditional (Apache-like) log files. To activate this feature, just create a sub-folder called /logs for the virtual hosts of your choice. Log files will not be generated/updated if the folder does not exist (or is renamed to, say, “/_logs”).

There are three different kinds of log files:

• `gwan.log`	global events: startup/shut-down, script loading errors;
• `error.log`	HTTP errors on a per virtual host basis;
• `access.log`	all HTTP requests (and errors) for a virtual host.

G-WAN’s performances are only slightly lower when log files are enabled. The difference is small (negligible for real-life use) but is noticeable in benchmarks.

Note: You have to stop and restart G-WAN to apply your log files changes.

G-WAN log files are automatically rotated daily at 0:00 (GMT) in order to make it easier to archive, trace and analyze them. Each file is renamed as follows:

- `gwan.log    => gwan_yyyy-mm-dd.log`
- `error.log   => error_yyyy-mm-dd.log`
- `access.log  => access_yyyy-mm-dd.log`

Where yyyy represents yesterday’s year, mm the month and dd the day.

You can create 'live' ASCII/HTML reports (and save them in the /logs folder) to watch a summary of G-WAN’s internal performance counters such as uptime, in/out traffic, RAM levels, number of connections, HTTP requests, script requests, HTTP errors, script errors, abnormal timeouts (attacks), etc. See the `server_report()` API call: http://gwan.com/api#report

This function makes G-WAN's internal performance counters available from C scripts (Chapter III), allowing your C scripts to log additional events under particular circumstances.

Command-line options

```
gwan [ -b | -d | -g | -k | -r | -t | -v | -w ]   [ argument ]
```

-b enables the TCP_DEFER_ACCEPT option (you are supposed to know what you are doing: this makes it impossible for G-WAN to reject timeout attacks).

-d daemon mode: gwan will still run after user logged off, but no longer output text in the terminal. Another 'angel' instance of gwan is run to restart gwan if it stopped. You can specify a group and/or user to dump root privileges:

```
gwan -d:group:user
gwan -d:user           (here the group used is the default user's group)
```

If you can't reach some files (HTML pages, image, CSS files, C scripts) then check the folder permissions (the account used to run G-WAN must have access to those files – we use the `0644` permission mask and the account '`www-data`' to run gwan with '`root`' as the owner of data files): `sudo ./gwan -d:www-data`

`-g` allows you to use `-w` with more workers than your machine has physical CPU Cores (you are supposed to know what you are doing: you are most likely wasting CPU and RAM).

`-k` gracefully stop all running gwan processes (useful to stop gwan when it is running as a daemon, see the -d option).

gwan uses the `Gwan_12345.pid` (parent) and `gwan_23456.pid` (child) files to find the processes to kill. If they don't exist or are not reachable then gwan will say: "no gwan instance found" and it will fail to stop the running daemon.

In that case, kill the gwan processes by using: `sudo killall -r gwan`
(type `'man killall'` to find how to filter by time, etc.)

`-r` run the specified C script and exit (no signal handler are installed: a crash will stop this new instance of gwan which is not acting as a server); This is useful to run arbitrary C source code (*not* G-WAN C servlets or handlers):
`./gwan -r ab.c` (to run the http://gwan.ch/source/ab.c.txt test);

`-t` stores all client requests in a "./trace" file before they reach the server. This impacts performance but lets you track attacks.

-v display the version number and build date (also listed in gwan/logs/gwan.log).

`-w` forces the number of server worker threads (-w 4 will bypass the number of physical CPU Cores on your machine).

Use -h to get the command line help.

Web Site Optimization (HTML, CSS, Javascript, and pictures)

Before HTTP compression, comments and blanks can be waved from HTML, CSS and JS files to reduce their size. But modifying files requires write access and makes them difficult to read, forcing people to use two copies: one for edition and one for production.

There is a better way: G-WAN does it on-the-fly when it loads files from disks. In CSS files, G-WAN also complements image links (for those many tiny icons that are < 4096 bytes in size) with “Data URI” base64-encoded images (RFC 2397):

```
// In the CSS file:
.extern { background:url("../imgs/extern.gif") no-repeat right; }becomes (the url link
is kept for the inept MS Internet Explorer):
.extern { background:url(data:image/gif;base64,R0lGOD...)
         *background:url("../imgs/extern.gif") no-repeat right; }

// In the HTML file, both are invoked as follows:
<a class="extern" href=”http://gwan.ch/”>gwan.ch</a>
```

Merging icons into CSS file(s) eliminates many connections because CSS files are *cached* by Internet browsers.

G-WAN minifies files in **daemon mode** only (to let developers recognize their code when using G-WAN and the Web browser to trace what's happening).

We can also save connections by grouping larger images (> 4096 bytes) in a single file. To get higher compression rates, group them **horizontally** and **by color set** (so they can share the same palette). The file “test_loans.png” contains 4 loan pictures:

```
// In the CSS file:
.clip  { position: absolute; top: 0; left: 0; }
.clipw { position: relative; }

.loan_1 { clip:rect(0 437px 185px     0); } // rect(y1, x2, y2, x1)
.loan_10{ clip:rect(0 874px 185px 437px); left: -437px; } // etc.


// In the HTML file, absolute position on a page:
<img src="imgs/loan_lan.png" alt="Loan" class="clip loan_1">

// In the HTML file, relative positioning (following text flow):
<div class="clipw" style="height: 185px; width: 437px;">
   <img src="imgs/loan_lan.png" alt="Loan" class="clip loan_1">
</div>
```

GIF has a low overhead and should be used for icons and other small images. PNG works better for larger images because it compresses better while using more verbose headers. The JPG format should be used with real-world photos.

Always reduce the number of colors to the smallest possible power of two (2, 4, 8, etc.) that respects your image palette: doing so will significantly reduce the file size.

Supported HTTP features

Protocols: HTTPS (SSLv2, SSLv3, TLS 1.1 + the TLS 1.2 “server_name” extension or Server Name Indication, see RFC 3546 and 4366), HTTP/0.9, HTTP/1.0, and HTTP/1.1

Methods: GET, HEAD, POST, PUT, DELETE, OPTIONS (a request can be 4-KB long); all the other 20 HTTP methods are parsed by G-WAN for C scripts, see gwan/include/gwan.h.

Encodings: entity, gzip, deflate (encodings are already parsed for handlers / servlets)

Conditions: If-[Un]Modified-Since, If-Match, If-None-Match (Etags), If-Range (bytes only)

Authorization: BASIC and DIGEST, with manual and automatic session support, see the session.c and auth.c examples

Others	HTTP servlets, HTTP handlers, directory listings, caches updated in real-time, HTTP compression (deflate, RFC 1950 and gzip, RFC 1952) DNT (Do Not Track) HTTP Header

Supported MIME types

```
atom  application / atom+xml
xls   application / excell
gwan  application / x-gwan
js    application / javascript
json  application / json
pdf   application / pdf
bin   application / octet-stream
exe   application / octet-stream
dll   application / octet-stream
ai    application / postscript
eps   application / postscript
ps    application / postscript
rdf   application / rdf+xml
xrdf  application / rdf+xml
rss   application / rss+xml
eot   application / vnd.ms-fontobject
amf   application / x-amf
arj   application / x-arj-compressed
rar   application / x-arj-compressed
bz2   application / x-bzip2
gwe   application / x-encrypted-gwan
fcs   application / x-fcs
ttf   application / x-font-ttf
woff  application / x-font-woff
~~~   application / x-msdownload
dat   application / x-ns-proxy-autoconfig
pac   application / x-ns-proxy-autoconfig
swf   application / x-shockwave-flash
tar   application / x-tar
tgz   application / x-tar-gz
gz    application / x-gunzip
crt   application / x-x509-ca-cert
der   application / x-x509-ca-cert
pem   application / x-x509-ca-cert
xhtm  application / xhtml+xml
xml   application / xml
zip   application / zip
mp3   audio / mpeg
```

wav	audio / wav
otf	font / opentype
gif	image / gif
png	image / png
jpg	image / jpeg
jpeg	image / jpeg
svg	image / svg+xml
ico	image / x-icon
bmp	image / x-ms-bmp
mf	text / cache-manifest
css	text / css
htm	text / html
html	text / html
shtm	text / html
asm	text / plain
aspx	text / plain
c	text / plain
cpp	text / plain
cs	text / plain
d	text / plain
for	text / plain
go	text / plain
h	text / plain
hpp	text / plain
java	text / plain
jsp	text / plain
m	text / plain
mm	text / plain
pas	text / plain
php	text / plain
py	text / plain
s	text / plain
txt	text / plain
rtf	text / richtext
ttl	text / turtle
mov	video / quicktime
mp4	video / mp4
mpg4	video / mp4
mpg	video / mpeg
mpeg	video / mpeg
ogv	video / ogg
webm	video / webm
flv	video / x-flv
mng	video / x-mng
asx	video / x-ms-asf
wmv	video / x-ms-wmv
avi	video / x-msvideo

As this list is hard-coded you cannot add MIME types in G-WAN but we will add any type that makes sense if users ask for it.

Updating static contents

When you need to add or update documents located in the `www` directory you can do so without stopping G-WAN (if the cache is enabled, all cached files are updated in real-time).

Updating servlets (C, C++, etc.)

When you need to add or update servlets located in the `csp` directory you can do so without stopping G-WAN (the latest version of a script is executed).

Default HTML CSS style sheet and HTTP Errors CSS style

To personalize the HTTP default style sheet (used by directory listings), you have to make your CSS style available under `/www/imgs/style.css`.

To personalize the HTTP error style, you have to create a CSS style sheet and make it available under `/www/imgs/errors.css`.

While G-WAN is supporting all the HTTP error codes (that's useful for servlets), only a subset is relevant for the server (like 404, Not found):

```
“100 Continue”
“101 Switching Protocols”
“102 HTTP Processing”

“200 OK”
“201 Created”
“202 Accepted”
“203 Non-Authoritative Information”
“204 No Content”
“205 Reset Content”
“206 Partial Content”
“207 Webdav Multi-status”

“300 Multiple Choices”
“301 Moved Permanently”
“302 Found”
“303 See Other”
“304 Not Modified”
“305 Use Proxy”
“307 Temporary Redirect”

“400 Bad Request”
“401 Unauthorized”
“402 Payment Required”
“403 Forbidden”
“404 Not Found”
“405 Method Not Allowed”
“406 No Acceptable”
“407 Proxy Authentication Required”
“408 Request Time-out”
“409 Conflict”
“410 Gone”
“411 Length Required”
“412 Precondition Failed”
“413 Request Entity Too Large”
“414 Request-URI Too Large”
“415 Unsupported Media Type”
“416 Requested range not satisfiable”
“417 Expectation Failed”
“422 Unprocessable Entity”
“423 Locked”
“424 Failed Dependency”
“425 No Matching Vhost”
“426 Upgrade Required”
“449 Retry With Appropriate Action”

“500 Internal Server Error”
“501 Not Supported”
“502 Bad Gateway”
“503 Service Unavailable”
“504 Gateway Time-out”
```

```
“505 HTTP Version not supported”
“506 Variant also varies”
“507 Insufficient Storage”
“510 Not Extended”
```

If you use custom error codes after `600` you will have to supply their description.

Disabling Directory Listing

Just copy an `index.html` file in the specific directories that you want visitors not to browse. G-WAN only lists files in those directories that miss such an `index.html` file.

Note: this file must not be empty (it must at least contain a space character) but you can also use a more personalized message.

Enabling in-memory Caching

G-WAN can be used as a caching server, both for static and dynamic contents. This is mostly useful when the contents to serve fits in RAM, like for a small web site made of a few thousands of pages, or for the user interface of Web applications.

In those cases, using the in-memory cache is of great help because:

- the disk I/O bottleneck is waved, allowing greater performance and scalability
- disk I/O resources are left to other consuming tasks like database queries
- serving pages faster also lets you benefit from a natural protection from DoS attacks.

By avoiding disk I/O, in-memory caching is allowing static resources to achieve the same performance and scalability that dynamic contents enjoy with G-WAN.

> To verify this, just compare how much better `GET /nop.gif` will perform than `GET /100.bin` when in-memory caching is disabled. Caching will more than double G-WAN's performances (strangely, this won't be the case for Nginx – see the test below).
>
> G-WAN, Nginx and many other servers have an embedded resource called “bacon”. This transparent 43-byte GIF pixel can be served better than the equivalent file stored on disk because the server does not have to read it from the hard-disk.
>
> To test it with the same URI from G-WAN and Nginx use the following directive in your `nginx.conf` file: `location = /nop.gif { empty_gif; }`

But there are other server applications like CDNs (Content Delivery Networks), online Media, huge shopping centers, or even VPS hosting where the available amount of RAM can rarely satisfy the caching needs: there's not enough RAM to store the Web resources.

This is why G-WAN disables *automatic* in-memory caching by default. This also makes it easier to compare G-WAN to other servers (Nginx's memcached module is slower than G-WAN's KV store because memcached is itself a server so the bottleneck is the network here).

To enable *automatic* resources caching, use the `init.c` file located in the `/gwan` root directory.

Note that, even when the *automatic* use of the `/www` cache is disabled, you can still add entries into the cache from a script with `cache_add()`. This may be useful to rename a Web bacon on a per-user basis for example.

Inversely, when the `/csp` (servlets) cache is enabled, you can prevent a servlet from being cached by using the `RC_NOCACHE` return code.

Properly used, these caches will double the server performance and scalability.

Enabling or Disabling Timeouts, POST entity size, etc.

Unlike other Web servers like Apache2 or Nginx (and Application servers like GlassFish or Tomcat), G-WAN does not use configuration files: the values are adjusted on-the-fly depending on the current context (server load, fetched resource, etc.).

This may look difficult to believe for many, but relying on fixed-values is a very bad thing.

For example, Apache2 or Nginx use a fixed number of concurrent clients in configuration files to allocate the memory upfront before accepting connections.

In contrast, G-WAN starts with a much lower memory usage because it allocates this same memory on-demand – and it will allocate more resources on-the-fly as needed.

This lets G-WAN cope with unexpected traffic spikes. Apache2 and Nginx will not cope because users used a low fixed value to save RAM.

But configuration files are a wider problem for usability, performance and security:

- fixed timeouts assume that all clients take the same time to load files of all sizes!
- the maximum `POST entity` size accepted by a server should be tunable on-the-fly
- the `tcp_nopush` or `tcp_nodelay` socket options must change depending on the resource type rather than be fixed for a website.

Failure to use dynamic values means that Nginx or Apache2 need different configuration files (and different server sections) to serve each resource sizes optimally! As this can't be done at the byte level, you have to accept compromises (which translate into inefficiencies).

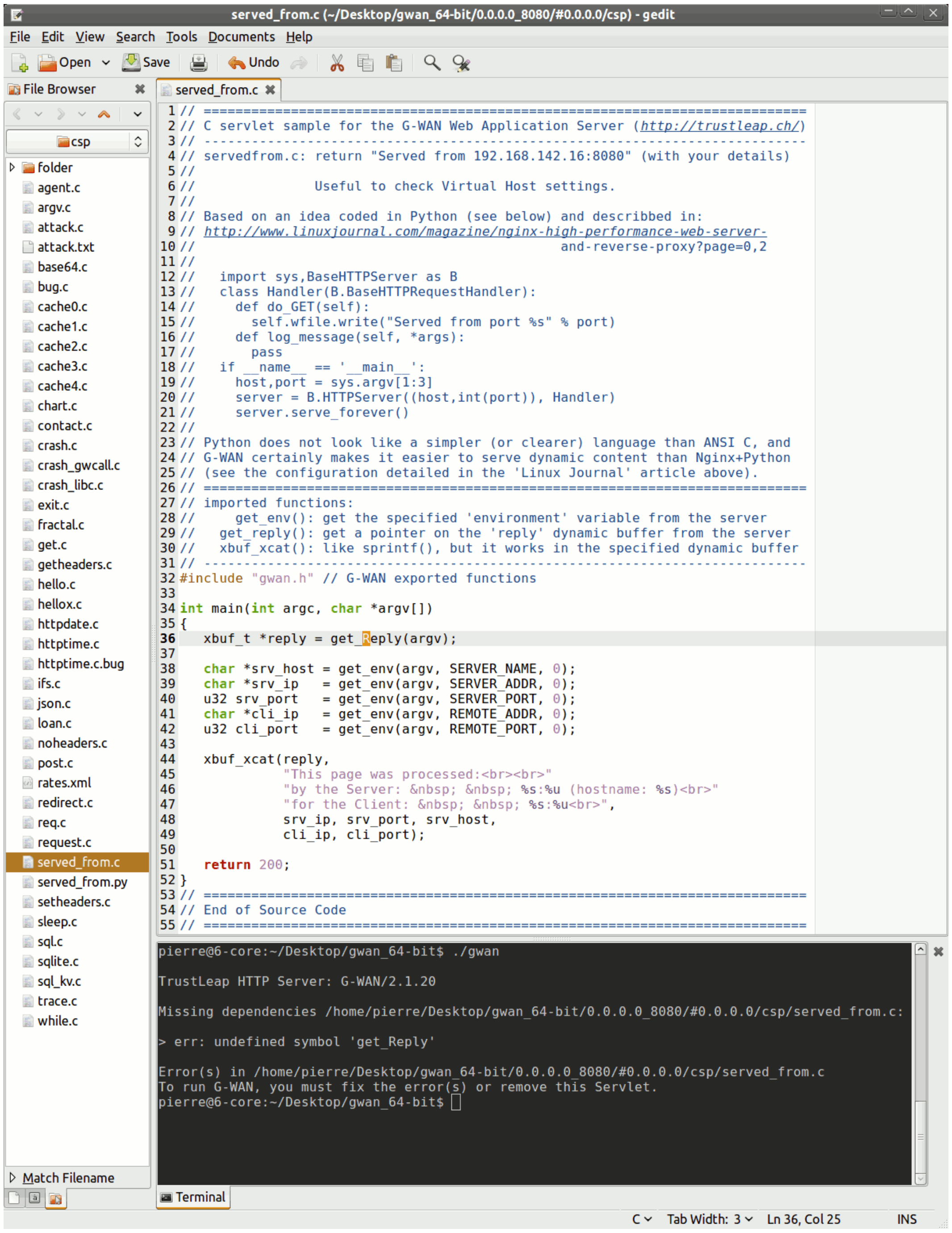
In the relevant cases, the G-WAN API allows you to decide on-the-fly and on a per-connection basis or globally what settings to apply. See the `get_env()` function and the related G-WAN environment variables (the `gwan/init.c_` file is an example) .

By using auto-tunable variables instead of fixed values, G-WAN not only makes it easier for users to deploy a server (there's less room for errors and the learning curve is much smaller) but it also makes G-WAN both more efficient and safer.

For example, dynamically adjusted timeouts defeat common DoS attacks. That's not a luxury. This must be a by-design feature.

II. Setting-up an IDE.

You can develop G-WAN servlets with gedit (it comes with Linux) and an Internet browser:



```c
// ============================================================================
// C servlet sample for the G-WAN Web Application Server (http://trustleap.ch/)
// ----------------------------------------------------------------------------
// servedfrom.c: return "Served from 192.168.142.16:8080" (with your details)
//
//              Useful to check Virtual Host settings.
//
// Based on an idea coded in Python (see below) and describbed in:
// http://www.linuxjournal.com/magazine/nginx-high-performance-web-server-
//                                               and-reverse-proxy?page=0,2
//
//   import sys,BaseHTTPServer as B
//   class Handler(B.BaseHTTPRequestHandler):
//     def do_GET(self):
//       self.wfile.write("Served from port %s" % port)
//     def log_message(self, *args):
//       pass
//   if __name__ == '__main__':
//     host,port = sys.argv[1:3]
//     server = B.HTTPServer((host,int(port)), Handler)
//     server.serve_forever()
//
// Python does not look like a simpler (or clearer) language than ANSI C, and
// G-WAN certainly makes it easier to serve dynamic content than Nginx+Python
// (see the configuration detailed in the 'Linux Journal' article above).
// ============================================================================
// imported functions:
//     get_env(): get the specified 'environment' variable from the server
//   get_reply(): get a pointer on the 'reply' dynamic buffer from the server
//   xbuf_xcat(): like sprintf(), but it works in the specified dynamic buffer
// ----------------------------------------------------------------------------
#include "gwan.h" // G-WAN exported functions

int main(int argc, char *argv[])
{
   xbuf_t *reply = get_Reply(argv);

   char *srv_host = get_env(argv, SERVER_NAME, 0);
   char *srv_ip   = get_env(argv, SERVER_ADDR, 0);
   u32 srv_port   = get_env(argv, SERVER_PORT, 0);
   char *cli_ip   = get_env(argv, REMOTE_ADDR, 0);
   u32 cli_port   = get_env(argv, REMOTE_PORT, 0);

   xbuf_xcat(reply,
             "This page was processed:<br><br>"
             "by the Server: &nbsp; &nbsp; %s:%u (hostname: %s)<br>"
             "for the Client: &nbsp; &nbsp; %s:%u<br>",
             srv_ip, srv_port, srv_host,
             cli_ip, cli_port);

   return 200;
}
// ============================================================================
// End of Source Code
// ============================================================================
```

```
pierre@6-core:~/Desktop/gwan_64-bit$ ./gwan

TrustLeap HTTP Server: G-WAN/2.1.20

Missing dependencies /home/pierre/Desktop/gwan_64-bit/0.0.0.0_8080/#0.0.0.0/csp/served_from.c:

> err: undefined symbol 'get_Reply'

Error(s) in /home/pierre/Desktop/gwan_64-bit/0.0.0.0_8080/#0.0.0.0/csp/served_from.c
To run G-WAN, you must fix the error(s) or remove this Servlet.
pierre@6-core:~/Desktop/gwan_64-bit$ 
```

gedit lets you write servlets and you just have to press F5 in the Internet browser to see the results of a G-WAN servlets like http://127.0.0.1:8080/?hello.c

We have found the following gedit plugins handy:

– file browser pane (the list of files on the left of the screenshot)
– embedded terminal (the black rectangle at the bottom of the picture)
– session saver (it re-opens the folder and all the files)
– automatic code-completion
– indent lines
– etc.

Install them with:

```
sudo apt-get install gedit-plugins
```

More plugins that may help:

http://live.gnome.org/Gedit/Plugins

And, as always with C, you can write your own:

http://live.gnome.org/Gedit/NewMDIPluginHowTo

III. Application Server: Dynamic contents

Servers need *scripts* for rapid-development and *compiled filters* for raw speed. G-WAN *scripts* do both – with compiled code performances. If you are in doubt about which language to use then C survived 40 years for a reason: it fits the task.

> *“By using C, applications that previously had required big machines could be executed on small ones, like the 8080.”* (Thomas Plum, 1976)

But you can use many programming languages with G-WAN, and most will fly almost as high as C if you enable dynamic contents caching with `get_env(argv, USE_CSP_CACHE);` (see the init.c_ example in the ./gwan directory).

If you consider using C/C++ or Objective-C, then keep reading to see how G-WAN's API can help you to extract each CPU cycle from your CPU Cores.

Else, you can skip the API calls list in this chapter because your language already has APIs for doing all that, and the G-WAN API can't be ported without loosing its speed. Just read this chapter to learn how G-WAN works (return values, RESTFUL URIs, crash reports).

Assuming that G-WAN is installed and running, if you look at the files located in the `/csp` directory, you will see C source code files (the “servlets”). Servlets are run when clients request the corresponding URL with a `“?”` query character: http://127.0.0.1/?bench.c

To display the `bench.c` source code you would rather use: http://127.0.0.1/bench.c

Your first C servlet: “301 moved permanently”

Redirecting users is useful after you moved or deleted the previous URL on your server. All the information necessary for a redirect is in the headers. The body of the response is typically empty, but one is created here to see how to proceed:

```
int main(int argc, char *argv[])
{
   xbuf_t *reply = get_reply(argv); // get a pointer on the server reply buffer

   xbuf_xcat(reply,
            “HTTP/1.1 301 Moved Permanently\r\n”
            “Content-type: text/html\r\n”
            “Location: new.html\r\n\r\n”
            “<html><head><title>Redirect</title></head>”
            “<body>Click <a href=\”new.html\”>here</a>”
            ”.</body></html>”);

   return 301; // return an HTTP code (301:’Moved’)
}
```

The function `xbuf_xcat()` works like `sprintf()` and lets you write the reply that the server will send to the client (without worrying about the length of the buffer because it is extended automatically).

Your “reply” buffer can contain HTTP headers only, or just HTML code and no headers, or both headers and HTML. When HTTP headers are *missing*, the server creates headers to match your `main()`’s return code (the HTTP status code).

All the standard HTTP status codes are supported but if you use your own custom codes (in the `600`+ range) then the server can’t imagine their purpose so you will have to explicitly define headers and an HTML message (if you target human clients).

The following example (without headers) is equivalent to the previous example (which explicitly defined response headers):

```
int main(int argc, char *argv[])
{
   xbuf_t *reply = get_reply(argv); // pointer on server reply buffer

   static char szURI[] = ”new.html”; // new location
   xbuf_xcat(reply,
             ”<html><head><title>Redirect</title></head>”
             “<body>Click <a href=\”%s\”>here</a>.</body></html>”,
             szURI);

   return 301; // return an HTTP code (301:’Moved’)
}
```

A servlet can use this auto-completion feature to reduce the code to its simplest expression (for example, to filter connections per IP address, CIDR, or country):

```
int main() // status code 401 means ‘Unauthorized’
{
   …           // do whatever you need to filter connections
   Return 401; // gwan uses '401' to build headers and an HTML reply
}
```

Today, either your servlets will define all the headers or you will expect the server to do it all for you. Environment variables (like an up-to-date HTTP date stamp) are available to make it easier to quickly build HTTP headers.

Other dynamic buffer routines will help you in the task of building a reply.

Note: to send something else than HTML (like a PNG or an XML document), you MUST:

- explicitly define HTTP headers (servlet examples are provided) *or,*
- return an invalid HTTP status code (see the JSON explanation below) *or,*
- use the get_env() function call to setup a MIME type for your reply (see `fractal.c`):

```
// specify a MIME type so we don't have to build custom HTTP headers
char *mime = (char*)get_env(argv, REPLY_MIME_TYPE);

// note that we setup the FILE EXTENTION, not the real MIME type:
mime[0] = '.'; mime[1] = 'g'; mime[2] = 'i'; mime[3] = 'f'; mime[4] = 0;
```

This latest method is the easiest (and the most efficient), but, hey, we are programmers: that means that G-WAN is not there to limit your options.

Sending non-HTTP Replies (JSON, etc.)

A servlet may need to talk to a client without HTTP headers. Here, G-WAN's HTTP headers automatic completion (based on the HTTP status code) is a nuisance.

To prevent HTTP headers automatic completion, just make your servlets return an *invalid* HTTP status code in the 1-99 range (inclusive).

By doing so, you can send whatever you wish, and G-WAN will not interfere.

Dynamic buffers

Dynamic buffers, like memory pools, are an efficient way to reduce the burden of memory management for high-performance programs. They are also convenient: servlets can just fill dynamic buffers without having to care about size, alignment, allocation lifetime, locks or heap fragmentation.

They are also immensely safer: you can't overflow dynamic buffers (unless you are using all the memory available on a machine) and 'bad' pointers are more likely to point to legal memory areas – the kind that will not cause a crash.

Each C servlet has a "reply" `xbuffer` aimed at sending information to clients.

But it may also be useful to create additional dynamic buffers in your servlets (to load an HTML template file, or to get the reply of a query sent to a web server).

You are expected to call `xbuf_free()` to release any dynamic buffer that you have created (but you should never free the server "reply" buffer).

`xbuf_reset()`	(re)initiatize a dynamic buffer (without freeing memory)
`xbuf_frfile()`	load a file, and store its contents in a dynamic buffer
`xbuf_tofile()`	save the dynamic buffer in a file
`xbuf_frurl()`	make an HTTP request, and store the result in a dynamic buffer
`xbuf_cat()`	like strcat(), but in a dynamic buffer rather than a string
`xbuf_ncat()`	like strncat(), but it also copies binary data in the specified buffer
`xbuf_xcat()`	formatted strcat() (a la sprintf) in the specified dynamic buffer
`xbuf_insert()`	insert bytes at a given position in the buffer
`xbuf_delete()`	delete bytes at a given position in the buffer
`xbuf_getln()`	get an LF-terminated text line from a buffer
`xbuf_findstr()`	find a given string into the buffer
`xbuf_repl()`	replace a string by another string in a buffer
`xbuf_replfrto()`	like the call above, but from/to given pointers in the buffer
`xbuf_free()`	release the memory previously allocated for a dynamic buffer

The servlet examples (`/csp` folder) demonstrate the syntax of all those functions.

Why not just enumerate their parameters here like in most manuals? Because nothing replaces working examples: we have seen countless documentations that miss critical *context* information – like in which order the API calls should be called to have any effect.

Getting GET/POST parameters

Sending information is only half of the job: often, you will also need to get information sent by the client (via `GET` or `POST` HTTP requests).

G-WAN transparently processes `GET`, `POST` and `PUT` in the very same way to let you access parameters with the same code (via the `get_arg()` call), but you can also walk `main() argv[]` the 'manual' way (see the `argv.c` example):

```
unsigned int i = 0;
while(i < argc)
{
    xbuf_xcat(reply, "argv[%u] '%s'<br>", i, argv[i]);
    i++;
}
```

Please refer to the `csp/contact.c, csp/loan.c and csp/argv.c` examples.
You can invoke those examples as follows:

http://127.0.0.1/?contact.c

Getting server “environment” variables

Traditional ‘environment’ variables are available to servlets. G-WAN variables, like the current HTTP date/time are also available: the work is already done by the server.

```
REQUEST = 0,      // char  *REQUEST;        // "GET / HTTP/1.1\r\n..."
REQUEST_LEN,      // int    REQUEST_LEN     // strlen(REQUEST); with headers
REQUEST_METHOD,   // int    REQUEST_METHOD  // 1=GET, 2=HEAD, 3=PUT, 4=POST
QUERY_STRING,     // char  *QUERY_STRING    // request URL after first '?'
FRAGMENT_ID,      // char  *FRAGMENT_ID     // request URL after last '#'
REQ_ENTITY,       // char  *REQ_ENTITY      // "arg=x&arg=y..."
CONTENT_TYPE,     // int    CONTENT_TYPE    // 1="x-www-form-urlencoded"
CONTENT_LENGTH,   // int    CONTENT_LENGTH  // body length provided by client
CONTENT_ENCODING,// int    CONTENT_ENCODING// entity, gzip, deflate
SESSION_ID,       // int    SESSION_ID;     // 12345678 (range: 0-4294967295)
HTTP_CODE,        // int   *HTTP_CODE;      // 100-600 range (200:'OK')
HTTP_HEADERS,     // struct *http_t;        // see struct http_t above
AUTH_TYPE,        // int    AUTH_TYPE;      // see enum AUTH_Type {}
REMOTE_ADDR,      // char  *REMOTE_ADDR;    // "192.168.54.128"
REMOTE_BIN_ADDR, // u64    REMOTE_BIN_ADDR;// u64 ip = numeric_ip_address;
REMOTE_PORT,      // int    REMOTE_PORT;    // 1460 (range: 1024-65535)
REMOTE_PROTOCOL, // int    REMOTE_PROTOCOL // ((HTTP_major*1000)+HTTP_minor)
REMOTE_USER,      // char  *REMOTE_USER     // "Pierre"
REMOTE_PWD,       // char  *REMOTE_PWD      // "secret"
CLIENT_SOCKET,    // int    CLIENT_SOCKET   // 1032 (-1 if invalid/closed)
USER_AGENT,       // char  *USER_AGENT;     // "Mozilla ... Firefox"
SERVER_SOFTWARE, // char  *SERVER_SOFTWARE // "G-WAN/1.0.2"
SERVER_NAME,      // char  *SERVER_NAME;    // "domain.com"
SERVER_ADDR,      // char  *SERVER_ADDR;    // "192.168.10.14"
SERVER_PORT,      // int    SERVER_PORT;    // 80 (443, 8080, etc.)
SERVER_DATE,      // char  *SERVER_DATE;    // "Tue, 06 Jan 2009 06:12:20 GMT"
SERVER_PROTOCOL, // int    SERVER_PROTOCOL // ((HTTP_major*1000)+HTTP_minor)
VHOST_ROOT,       // char  *VHOST_ROOT;     // the (virtual) host root folder
WWW_ROOT,         // char  *WWW_ROOT;       // the HTML pages root folder
CSP_ROOT,         // char  *CSP_ROOT;       // the CSP .C files folder
LOG_ROOT,         // char  *LOG_ROOT;       // the log files folder
HLD_ROOT,         // char  *HLD_ROOT;       // the handlers folder
FNT_ROOT,         // char  *FNT_ROOT;       // the fonts folder
MIN_SEND_SPEED,  // int   *MIN_SEND_SPEED; // in bytes/sec (if < close)
```

```
   MIN_READ_SPEED,  // u32   *MIN_READ_SPEED; // in bytes/sec (if < close)
   READ_XBUF,       // xbuf_t*READ_XBUF;      // pointer to the read() xbuffer
   HEAD_XBUF,       // xbuf_t*HEAD_XBUF;      // response HTTP headers(), if any
   SCRIPT_TMO,      // u32   *SCRIPT_TMO;     // time-out in milliseconds
   KALIVE_TMO,      // u32   *KALIVE_TMO;     // time-out in milliseconds
   REQUEST_TMO,     // u32   *REQUEST_TMO;    // time-out in milliseconds
   NBR_CPUS,        // int    NBR_CPUS;       // total of available CPUs
   NBR_CORES,       // int    NBR_CORES;      // total of available CPU Cores
   NBR_WORKERS,     // int    NBR_WORKERS;    // total of server workers
   CUR_WORKER,      // int    CUR_WORKER;     // worker thread number: 1,2,3...
   REPLY_MIME_TYPE, // char  *REPLY_MIME_TYPE;// set script's reply MIME type
   DEFAULT_LANG,    // u8     DEFAULT_LANG;   // CC_D: /?hello.d => /?hello
   QUERY_CHAR,      // u8     QUERY_CHAR;     // replace '?' by [ -_.!~*'() ]
   REQUEST_TIME,    // u64    REQUEST_TIME;   // time (parse+build) in microsec
   MAX_ENTITY_SIZE, // u32   *MAX_ENTITY_SIZE;// maximum POST entity size
   USE_WWW_CACHE,   // u8    *USE_WWW_CACHE;  // 0:disabled (default) 1:enabled
   USE_CSP_CACHE,   // u8    *USE_CSP_CACHE;  // 0:disabled (default) 1:enabled
   CACHE_ALL_WWW,   // u8    *CACHE_ALL_WWW;  // 1:cache all /www at startup
   USE_MINIFYING,   // u8    *USE_MINIFYING;  // '1' by default (JS/CSS/HTML)
```

DOWNLOAD_SPEED lets you calm G-WAN's enthusiasm at slamming the door on the face of impolite visitors, cutting connections that do not send or receive data sufficiently quickly.

The default (fair?) value is 4,096 bytes per second. If you feel that it is acceptable for clients to be slower, set DOWNLOAD_SPEED to an integer value > 1 (like 2, 3, 10...):

```
   int *pDN_SPEED = (int*)get_env(argv, DOWNLOAD_SPEED);
   if(pDN_SPEED) // check that we got a pointer
      *pDN_SPEED = 2; // allow 2,048 bytes per second
```

If you don't trust all your visitors but would like a more permissive policy for a privileged group of users then you can use a G-WAN Handler to apply this option on a per case basis (by CIDR, IP address, authentication, etc.).

The SCRIPT_TMO value is addressed in the same way as DOWNLOAD_SPEED, get_env() giving you a pointer on the value that you can then read or modify.

```
READ_XBUF            // xbuf_t*                    // the G-WAN xbuffer used to store the HTTP request
```

Servlets can also access G-WAN’s internal performance counters:

```
CC_BYTES_IN         // unsigned long long
CC_BYTES_OUT        // unsigned long long
CC_BYTES_INDAY      // unsigned long long
CC_BYTES_OUTDAY     // unsigned long long
CC_ACCEPTED         // unsigned int        // total number of TCP connections
CC_CLOSED           // unsigned int        // total number of TCP connections
CC_REQUESTS         // unsigned int        // total number of requests
CC_HTTP_REQ         // unsigned int        // number of HTTP requests
CC_CACHE_MISS       // unsigned int        // requests not satisfied by the cache
CC_ACPT_TMO         // unsigned int        // attack: connection without request
CC_READ_TMO         // unsigned int        // attack: partial request received
CC_SLOW_TMO         // unsigned int        // attack: request sent too slowly
CC_SEND_TMO         // unsigned int        // attack: reply fetched too slowly
CC_CSP_REQ          // unsigned int        // number of Servlet requests
CC_STAT_REQ         // unsigned int        // number of Statistics requests
CC_HTTP_ERR         // unsigned int        // number of HTTP errors
CC_EXCEPTIONS       // unsigned int        // number of Servlet faults
CC_BYTES_INDAY      // u64                 // number of bytes received today
CC_BYTES_OUTDAY     // u64                 // number of bytes sent today
```

And this is G-WAN persistent pointers usable from scripts:

```
US_REQUEST_DATA = 200, // Request-wide pointer
US_HANDLER_DATA,       // Listener-wide pointer
US_VHOST_DATA,         // VirtualHost-wide pointer
US_SERVER_DATA,        // Server-wide pointer (global)
US_HANDLER_STATES,     // states registered to get server-state notifications
US_HANDLER_CTX_SIZE    // size of "Protocol Handler" per-connection context
```

Template Engines

Web development frameworks inevitably come with a template system. C#, Java and PHP mix scripting, variables and HTML (each using a different proprietary syntax) to achieve "*independence between the application user interface and the application logic*".

The G-WAN `contact.c` example is using an HTML template form where HTML comments are used to embed C script variables in the presentation layer:

```
<p><!--time--><br><!--ip--><br></p>
```

This portable and RFC-compliant choice has several advantages:

– the variable remains invisible until it is used (it's an HTML comment);
– the syntax is completely standard (that's not another patent mine-field);
– any other framework could use the same syntax overnight (openness);
– there is no limit about what you can put in such a variable (*you* decide).

Keeping it simple has its value: a lower learning curve, less bugs, etc.

The G-WAN Key-Value Store

A server is using lists. Some must be simple, others must be fast, and all must scale. Experts say that no data structure can do it all optimally.

Concurrency is a major feature. Most databases do not scale with concurrency because they rely on locks (like SQLite) or on delayed tasks (when they are "lock-free", like ORACLE).

– *Why make your own KV store when so many others already exist?*

– For the same reason that G-WAN was needed: it can be done much faster.

The G-WAN KV store uses only 7 functions, is faster than the best NoSQL DB engines, and scales seamlessly because it is "wait-free" (lock-free, without delayed tasks).

This store lets you create tables with records, as well as indexes – on-the-fly if needed.

Keys, like Values, are limited in size to 4 GB. Both can be ASCII strings or binary chunks and the `kv_add() / kv_get() / kv_del()` functions are the same whatever the case. A `kv_do()` call lets you apply a user-defined function to a subset of a KV store.

To see how to use those functions and `kv_init() / kv_free()`, look at `kv.c`.

To see how well it performs as compared to SQLite (b-tree) or Tokyo Cabinet TC (a hash-table) and TC-FIXED (a simple array), see the `kv_bench.c` example.

And this test is merely a single-thread test. Add concurrency and SQLite as well as Tokyo Cabinet die in pain because there a single write blocks other read/write threads.

Not in G-WAN's case. It is never ever blocking nor delaying any processing.

How solid is it? G-WAN relies on it and has been tested with very high concurrencies.

We could make it 2-4 times faster by pre-allocating memory instead of calling `malloc()` for each newly created record (but using `malloc()` lets G-WAN keep a low memory usage).

Using Persistence Pointers

Servlets and Handlers can use persistence to store data tuples, a socket connected to a database server or another application server, etc.:

```
// get the Virtual Host persistent pointer
void **ptr = (void**)get_env(argv, US_VHOST_DATA);

// just an example of what can be done
typedef struct hive_s
{
   kv_t *my_kv_store; // yep, G-WAN has one!
   void *my_whatever;
   void *my_sql_persistent_connection;
} hive_t;

if(!*ptr) // if the pointer has never been used, attach our structure
   *ptr = (void*)malloc(sizeof(hive_t));

if(*ptr)
   (*ptr)->my_whatever = strdup("I want to remember this");
```

To store more than a single buffer, the persistence pointer can host linked-lists, trees, in-memory SQLite tables, memcached entries, or … G-WAN's Key-Value Store (see kv.c).

To let you chose the most efficient tool for your needs, G-WAN just provides a pointer. If it is not used then it will not consume memory.

See `handlers/main.c` to see how to use `US_HANDLER_DATA` with a G-WAN Handler .
See `csp/contact.c` to see how to query other variables with `get_env()`.

Making Blocking BSD Socket Calls Run Asynchronously

Web frameworks are either blocking (performing poorly by stacking hundreds of threads) or asynchronous (and difficult to use because everything must be a state-machine).

To perform and scale one must avoid blocking a server. Using many blocking threads is not as efficient as using true asynchronous calls (because of the threading overhead: more memory used by each thread, context switches, etc.).

Client connections are difficult to use with asynchronous servers because they have to re-use the HTTP server internal state-machine (and doing this requires clunky interfaces, just look at how difficult writing Nginx modules can be).

G-WAN lets you write ***procedural code*** using blocking BSD socket calls like `connect()`, `recv()` or `send()` – while behind the scene they run asynchronously.

With this feature, C scripts can process network events without waiting for them to complete. Without it, the **latency** of **database servers** or of other back-end **application servers** uselessly blocks an HTTP server (or reverse-proxy) like G-WAN.

And it works *transparently* with **existing TCP-based network libraries** like `libCURL, OpenSSL` or the `mySQL / PostgreSQL` client ANSI C libraries.

Of course, G-WAN´s `xbuf_frurl()` HTTP client (see `get_headers.c`, `request.c` or `attack.c`) is taking advantage of it to let you query remote servers without ever blocking G-WAN's threads.

Putting it all together

The `/csp/loan.c` example uses `AJAX` to process a form without reloading the whole HTML page. When users press the 'Calculate' button, the loan is displayed in the same HTML form used to gather data entered by the end-user.

This example can be used as the basis of more complex Web 2.0 applications (G-WAN already issues session ids, see `get_env()` and `SESSION_ID`, and SQL libraries provide persistence for session handling):

LOAN	DETAILS
Amount	10,000.00
Rate	3.50%
Term	1 year(s)
Cost	190.60 (1.91%)

YEAR 1

MONTH	PAYMENT	INTEREST	PRINCIPAL	BALANCE
January	849.22	29.17	820.05	9,179.95
February	849.22	26.77	822.44	8,357.51
March	849.22	24.38	824.84	7,532.67
April	849.22	21.97	827.25	6,705.42
May	849.22	19.56	829.66	5,875.76
June	849.22	17.14	832.08	5,043.69
July	849.22	14.71	834.51	4,209.18
August	849.22	12.28	836.94	3,372.24
September	849.22	9.84	839.38	2,532.86

October	849.22	7.39	841.83	1,691.03
November	849.22	4.93	844.28	846.75
December	846.75	2.47	844.28	0.00

```
 This page was generated in 0.01 ms.
 (on a 3GHz CPU 1 millisecond = 3,000,000 CPU clock cycles)
```

As PHP, Perl, Python, Java and C# are orders of magnitude slower than C, G-WAN uses advanced threading scheduling and sub-second caching to accelerate slow servlets.

To benchmark a servlet you have to measure the script execution time (printed above) but also the server processing and reply time which can be calculated by Weighttp (an evolution of AB [Apache Benchmark] that lets you use more than one worker thread with the -t switch) with different concurrency loads (ab -c 10, 100, 500, 1000):

```
weighttp -n 1000000 -c 100 -t 4 -k -H “Accept-Encoding: gzip” \
   “http://10.10.2.4:80/?loan&name=Eva&amount=10000&rate=3.5&term=10”
```

Modifying the term (number of years) lets you control the volume of calculations, the length of the resulting HTML page, and verify how it scales with high concurrencies.

As explained in great details on our Web site, AB *cannot* saturate a Web server designed to use *multicore* systems because AB is *single-threaded*. Many more details are available for the curious here: http://gwan.ch/en_apachebench_httperf.html

This source code will help you to make benchmarks and generate charts with Requests per second, as well as CPU and RAM usage: http://gwan.ch/source/ab.c.txt

Additional functions

The portable G-WAN calls below (documented in `gwan.h`) are available from C servlets:

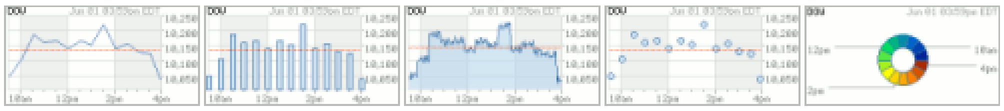
`cycles64()`	get the CPU clock cycle counter’s value (64-bit value)
`getms()`	get the current time in milliseconds (64-bit value)
`getus()`	get the current time in microseconds (64-bit value)
`s_time()`	equivalent to time(0) (but much faster under Windows)
`s_gmtime()`	equivalent to gmtime(); but faster (and thread-safe)
`s_asctime()`	equivalent to asctime(); but faster (and thread-safe)
`s_localtime()`	equivalent to localtime(); but faster (and thread-safe)
`time2rfc()`	format an HTTP date string from a given time_t value
`rfc2time()`	return a time_t value from an HTTP date string
`sw_init()`	a good pseudo-random numbers generator sw_rand()
`hw_init()`	a true hardware random numbers generator hw_rand()
`get_arg()`	get GET/POST application/x-www-form-urlencoded parameters
`get_env()`	get G-WAN’s “environment” variables
`url_encode()`	encode an URL so you can use it

`escape_html()`	encode a buffer so you can use it in HTML
`unescape_html()`	decode a buffer
`html2txt()`	remove all HTML tags from a buffer

`s_snprintf()` like the libc call, but with more tricks (all used by `xbuf_xcat`):

`%b`	binary conversion (use `%llb` for 64-bit integers)
	8 => “1000”
`%B`	encode a null-terminated string with base64
`%-B`	decode a base64 null-terminated string
`%12B`	encode a 12-byte binary buffer (null bytes do not stop encoding)
`%C`	generate a string of n times the specified character
`%3C`	’A’ => “AAA”
`%k`	1024 => “1 KB” (byte, KB, MB, GB... formatter; use `%llk` for 64-bit integers)
%H	calls escape_html()
%R	calls url_encode()
%T	calls html2txt()

`gif_build()`	build an in-memory GIF from a raw bitmap; see `fractal.c` and `chart.c`
`gif_parse()`	parse an in-memory GIF from a buffer; see the `chart.c` example
`dr_line()`	raw bitmap drawing primitives, see `gwan.h`
`dr_circle()`	
`dr_rect()`	
`dr_chart()`	draw area/bar/dot/line/pie/ring charts, see the `chart.c` example



It can also draw sparklines:

`md5()`	to calculate MD5 hash values
`Sha1()`	to calculate SHA1 hash values
`sha2()`	to calculate SHA2 hash values
`crc32()`	to calculate CRC32 checksums
`adler32()`	to calculate Adler32 checksums
`aes_init()`	to setup an encryption key (use `hw_rand()`)
`aes_enc()`	to encrypt data with the U.S. NIST FIPS PUB 197 standard (2001)
`gzip_cmp()`	to compress data under the GZIP and ZLIB (deflate) standard formats
`lzjb_cmp()`	to compress data very quickly
`lzjb_exp()`	to decompress data very quickly

`cacheadd()`	add (or update) a file or a buffer in G-WAN's memory cache
`cacheget()`	search a file or a buffer in G-WAN's memory cache
`cachedel()`	delete a file or a buffer from G-WAN's memory cache

`sendemail()`	send an email to an SMTP server (see the `contact.c` example)

`jsn_frtext()`	parse text to build a JSON tree
`jsn_totext()`	export a JSON tree into text
`jsn_byindex()`	search a value by its index in JSON tree
`jsn_byname()`	search a value by its name in JSON tree
`jsn_byvalue()`	search a value by its value in JSON tree
`jsn_add()`	add data to a JSON tree
`jsn_del()`	remove data from a JSON tree
`jsn_updt()`	update data in a JSON tree
`jsn_free()`	free the memory used by a JSON tree

`gc_malloc()`	`allocated temporary memory (freed when C scripts return)`
`gc_free()`	`free memory allocated by gc_malloc(), mostly useless`

`kv_init()`	`create a Key-Value`
`kv_add()`	`add a Key-Value tuple in the Store`
`kv_get()`	`search a Key-Value tuple`
`kv_del()`	`delete a Key-Value tuple`
`kv_free()`	`free` the Store and all its contents
`kv_do()`	`execute a user-defined function on a subset of the Store`

See the dedicated kv.c and kv_bench.c examples for how to use the Key-Value store.

"Pretty" URLs for Dynamic content generation

The default URI form is "`/?servlet`" (like "`/?hello.c`").

There's a simple way that avoids the "`/?`" prefix completely – *without URI rewriting* and which gives you total liberty, allowing requests like: "`/articles/coding/syntax`": use G-WAN's cache to store resources under a virtual path (see the cache.c example).

RESTFUL Web services

REST is a bunch of recommendations that aim to deliver stateless (and therefore scalable) Web services. The following suggestions are recurring:

– do not use query strings if possible (no `/?forum&topic=linking+issues`);
– use `GET` to fetch, `POST` to create, `PUT` to update, `DELETE` to erase data;
– keep all resources in a tree-like hierarchy (`/net/host/disk/dir/file`) ;
– keep URIs in lower-case (data can use upper-case);
– replace spaces by '_' (underscores);
– users must be able to bookmark all resources (so they can be cached too);
– resources must contain links to find more details about the resource: "`/products/412`" can contain the link "`/products/412/specifications`" ;

– use the "`Accept:`" HTTP header to let clients specify the format (xml, json, html...) they can use to read resources so your services can be more flexible;
– don't reply 404 for a partial path, reply with a parent or default resource;
– hide the server scripting technology (csp, jsp, php) so you can port applications to another language without changing the URLs.

Examples:

```
GET /?forum&listuser=Eva          (not RESTFUL: query, parameters)
GET /forum/users/Eva              (OK: GET tree-like)

GET /?adduser&name=Robert         (not RESTFUL, use POST to create)

POST /forum/users HTTP/1.1        (OK, use PUT to update data)
Host: gwan.ch
Content-Type: application/json
{ “user”: { “name”:”Eva” } }
```

As some vendors (like eBay) present APIs that they call “RESTFUL” and which are based on queries (?) and parameters (&), the concept is far from being strictly defined.

With (strict) RESTFUL URLs, it seems that you have first to test if the URL exists as a static content (like a directory). If none is found then you have to check if a servlet can match any part of the URL prefix before you report “404: Not found”.

Unless your server only serves dynamic contents, such a procedure is very inefficient.
This can be resolved by using:

– G-WAN Virtual Servers for each RESTFUL service like secure.host.com while host.com only serves non-RESTFUL services;

– You can also use a G-WAN Handler to rewrite URLs (and each Virtual Server can use a different Handler) in order to be truly RESTFUL;

If you absolutely need to get rid of the '?' query character then use a G-WAN handler to overwrite one single character:
(in the example below, a * is used instead of the ?)

client : `http://localhost/*hello.c` (make sure that a slash precedes the star: "/*")
handler: `http://localhost/?hello.c` (G-WAN returns 404 if the script does not exist)

The RESTFUL substitute character can safely be chosen from the unescaped URI character set: " - _ . ! ~ * ' ( ) "(see rfc_2396 section *"2.3. Unreserved Characters")*

The rest is discipline: instead of using query parameters you will have to use a hierarchy (but no query) and attributes (instead of parameters) in your URLs:

```
GET /?loan.c&name=Eva&amount=10000&rate=3.5&term=10
```

would become something like:

```
   GET /'loan.c/name/Eva/amount/10000/rate/3.5/term/10
```

Here, G-WAN cannot transparently parse ‘parameters’ like in a query because there is no explicit link between “`name`” and “`Eva`” or “`amount`” and “`10000`” in the syntax. For this to happen, you must rather use:

```
   GET /'loan.c/name=Eva/amount=10000/rate=3.5/term=10
```

Caching, Expires Header

Let’s say that you generate dynamic contents with a C servlet:

```
http://127.0.0.1/?servlet&arg1=123&arg2=456
```

But (a) you don’t want the same contents to be generated for each request, and (b) you want to make these contents available at a “pretty” URL (no “`/?servlet`”).

After you generate the page, just before calling return(200) in your code, insert the following code in your C servlet:

```
   // note: no starting ‘/’ in the virtual path
   static char path[] = ”tools/counter.html”;  // a ‘virtual’ path
   int expire = 0; // 0:never

   // 200 is the HTTP status code returned by the server for this cached entry
   // (play with redirections: ret = 301, or with cached JSON entries: ret = 1)
   if(cacheadd(argv, path, reply->ptr, reply->len, 200, expire) < 0)
      error(); // out of memory

   return 200; // return an HTTP code (200:’OK’)
```

`'path'` is the “pretty” path (not the URL) that you want to use. `cacheadd()` will just update any existing cache entry.

Use a relevant file extension to let G-WAN pick a specific MIME type so that HTTP compression can be applied when needed (without extension, `“html/text”` is used).

The `expire` value can be 0 (never expire, staying cached until you delete it), or it can be the number of seconds before it will expire (60*60=3600 for a one-hour lifespan).

Expire lets you to put entries in the cache – and forget about them, but expire also lets G-WAN to generate relevant `“Expires:“` and `“Cache-Control:“` HTTP headers on your behalf (telling proxy servers and browsers to query G-WAN only when needed).

The expiration feature lets you define expiring links to a given resource for clients. See the cache.c example.

HTTP Compression (gzip and deflate)

For each host, the `/gwan/.../gzip` directory is used to cache on disk the *static* documents that have already been gzipped. This cache is refreshed automatically if the *static* document is updated on disk in its `/www` directory.

For *dynamic* contents, if a client supports gzip or deflate then servlet outputs that are > 499 bytes are compressed on-the-fly (compressing smaller buffers wastes resources pointlessly).

G-WAN will not try to gzip dynamic contents it if your servlets use a MIME type that cannot be compressed (like JEPG, GIF or PNG images).

Scripts execution errors, crashes and debugging

At run time G-WAN signals syntax errors, undefined symbols, etc. in the terminal used to run gwan, before C servlets execute. This allows you to start G-WAN with servlets that (at least) compile and link.

G-WAN also “gracefully” handles C servlet crashes and reports where exactly in the C source code the fault happened (instead of stopping the server).

For example, if you let G-WAN run this code:

```
1. void crash() { *((int*)(0))=0xBADC0DE; } // write access violation
2. int  main () { crash(); return 200; }
```

G-WAN will tell you which line in your C source code file did it wrong:

```
Exception:       c0000005 Write Access Violation
Address:         06d3b413
Access Address:  00000000

Registers:       EAX=0badc0de CS=001b EIP=06d3b413 EFLGS=00010246
                 EBX=00000000 SS=0023 ESP=0166df34 EBP=0166df3c
                 ECX=00000000 DS=0023 ESI=00000104 FS=003b
                 EDX=0166fc58 ES=0023 EDI=0166f47c CS=001b

Call chain:(line) PgrmCntr(EIP) RetAddress FramePtr(EBP) StackPtr(ESP)
crash():    1      06d3b413      06d3b4a6   0166df3c      0166df34
main():     2      06d3b4a6      0042d1ea   0166df64      0166df34

Servlet: csp/crash.c
Query  : /?crash.c    (may be useful to reproduce the error)
Client : 127.0.0.1    (may be useful to identify recurring offenders)
```

Until you fix the code, G-WAN reports an “internal server error” (status 500).

G-WAN execution errors, crashes and debugging

When used in daemon mode, if a child dies the `gwan` parent process forks again to continue servicing clients. If there is a failure, it may be G-WAN (or something else), you need to know, and you must not have to search forever to find out.

The `/gwan/trace` file only lists the child start/stop status:

```
   Fri, 28 Oct 2010 09:11:46 GMT: start
   Fri, 28 Oct 2010 09:11:51 GMT: clean stop
```

If a child crashed, you will find a stack frames dump instead of a “`clean stop`”.

The `/gwan/gwan.log` file will also tell what happened before a new child was forked:

```
[Thu, 28 Oct 2010 10:25:16 GMT] * child normal exit(3)     exit code
[Sat, 28 Oct 2010 14:34:07 GMT] * child clean stop   Ctrl+C, gwan -k
[Sat, 28 Oct 2010 16:52:43 GMT] * child abort(11)       11 : SIGSEGV
```

The daily server HTML report also indicates how many forks took place, and lists the system, parent, and child uptimes:

```
System Uptime: 01 day(s) 00 month(s) 00 year(s) 12:32:44
Parent Uptime: 01 day(s) 00 month(s) 00 year(s) 10:40:40
Child  Uptime: 01 day(s) 00 month(s) 00 year(s) 10:40:40
fork: 1 (times parent started a child)
```

In daemon mode, if there is more than one fork then check `/gwan/trace`. If you have such a crash then contact us, we will do our best to help you find what caused the crash.

Web Applications Security

Cross-site scripting, injection attacks or request forgery are made easy and having success for simple reasons which can easily be listed:

– the surface of vulnerability is expending with new Web browser features;
– web developers already have a job and just can't cope with these issues;
– fixing the whole stuff would severely harm the so-called 'advertising' business.

By identifying data flows, simple cryptographic tags would greatly limit the room for abuses in the “cool-features” area because servers could distinguish between clients (the good, the bad and the ugly) -even in a single aggregated flow.

More sophisticated users would find it priceless to be in a position to actually trust what transits on public networks (with today’s tools, this goal remains out of reach).

It is typically claimed that cryptography is weakened or avoided to preserve performances and scalability when securing content. This is mainly due to the fact that people reuse generic libraries instead of writing on-purpose code (like G-WAN). A thing or two can also be done in this matter.

IV. Extending the Joy

A development platform must let developers and third-parties extend its features. And it must be as easy to use as possible – both to save time and to avoid errors.

G-WAN works with:

– `Servlets` (to handle HTTP forms, query databases, etc.);
– `Handlers` (to filter, encode, authenticate, log, implement protocols, etc.);
– `Libraries` (to add new functions to Servlets and Handlers);
– `Applets` (on the client side) and with the optional G-WAN 'maintenance' script.

A word about interfaces

Usually, plug-ins connect with the server through interfaces. They rely on formats that you have to learn, they are uselessly error-prone and complex – and they can even become obsolete and are replaced (it was the case for IBM Apache and Microsoft IIS).

For all those reasons, the best interface is “none”:

– `Servlets` copied into the gwan/.../`csp` sub-folder will be used;
– `Handlers` copied into the gwan/...`/handlers` sub-folder will be used;
– `Includes` copied into the gwan`/include` folder will be used;
– `Libraries` copied into the gwan`/libraries` folder will be used (#pragma link);
– `Fonts` copied into the `gwan/fonts` sub-folder will be used;
– `main.c` copied into the `/gwan` folder will be used as a maintenance script.

To disable a servlet/handler/library just delete or rename it extension (to *.c_, *.so_ -or anything else than the expected *.c and *.so).

To disable any of these capabilities, remove (or rename) the `/csp` folder to completely disable servlets, the `/handlers` folder to completely disable Handlers, etc.

Servlets

Servlets let programmers build Web applications on the top of G-WAN.

Servlets let you build replies to client HTTP requests. A reply can be an HTML page, just HTTP headers, both, or a PNG image, an XML document, etc.

You do not have to stop and restart `gwan` to update modified servlets (or to load new ones). G-WAN does it on-the-fly. Servlets are covered in Chapter II.

Connection Handlers

Connection Handlers are C scripts like servlets. But instead of just letting you build the reply of an HTTP request, they allow you to act at all the different stages of a connection:

(a) **after** the connection is **accepted**;

Use the client IP address to filter access to your server:

```
switch(ip_range) {
case x: return 0;    // close connection
case y: return 1;    // build reply based on created URL
case z: return 2;    // send reply provided in reply buffer
default: return 255; // continue normally with current data
}
```

(b) **after** data is **read** from the connections;

Here you can use G-WAN as a TCP server, controlling its behavior, decoding or altering a request, directly replying, etc.:

```
switch(choice) {
case x: return 0;    // close connection
case y: return 1;    // read more data from client
case z: return 2;    // send reply provided in reply buffer
default: return 255; // continue normally with current data
}
```

(c) **before** and **after** an HTTP request has been **parsed and validated**;

Here you can use G-WAN as a front-end server, and redirect or alter requests:

```
switch(choice) {
case x: return 0;    // close connection
case z: return 2;    // send reply provided in reply buffer
default: return 255; // continue normally with current data
}
```

(d) **if** the requested **resource** was **not found**;

You can change the server reply (or the HTTP error), or close the connection:

```
switch(choice) {
case x: return 0;    // close connection
case z: return 2;    // send reply provided in reply buffer
default: return 255; // continue normally with current data
}
```

(e) **before** and **after** the server reply is **sent** to the client;

You can encode the server reply before it is sent, or log the request with an alternate method (in selected cases for example), or stop HTTP keep-alives:

```
switch(choice) {
case x: return 0;    // close connection (do not send)
default: return 255; // continue normally with current data
}
```

Handlers have three entry points (see the /gwan/include/gwan.h file for details):

```
Int init(char *argv[], int argc);
int main(char *argv[], int argc);
int clean(char *argv[], int argc);
```

`init()` lets you define persistent data structures to hold white-lists or black-lists, etc. It also lets you define which notifications main() will receive:

```
int init(int argc, char *argv[])
{
   u32 *states = (u32*)get_env(argv, US_HANDLER_STATES);
   *states = (1 << HDL_AFTER_ACCEPT)
           | (1 << HDL_BEFORE_PARSE)
           | (1 << HDL_AFTER_WRITE)
           | (1 << HDL_HTTP_ERRORS); // this one is new
   return 0; // >= 0:success
}
```

`main()` is called for each server request and will do the job at each a/b/c/d step.

`clean()` is called by G-WAN when a virtual host (or the server) is closing. You can use it to free your persistent structures, save your custom counters on disk, etc.

There is an Handler example called main.cxx located in your gwan/handlers folder. Just rename it to main.c to have it being used by G-WAN. It shows how to use the `US_HANDLER_DATA` persistence pointer.

Handlers can be used to to customize and extend G-WAN at will. Here are some ideas:

– `ip_acl.c`	(filter incoming connections by IP addresses or CIDRs);
– `ip2geo.c`	(filter or redirect incoming connections by country);
– `throttle.c`	(limit the number of concurrent requests sent by a client);
– `url_wr.c`	(rewrite URLs to hide servlets or to redirect to moved pages);
– `crypto.c`	(decrypt HTTP requests and encrypt HTTP replies);
– `syslog.c`	(log connections on a remote syslog server).

Handlers can even be used to implement other protocols like POP3, SMTP or IMAP, using G-WAN as a general-purpose socket server in addition to using it as a Web server (the less you have server processes installed and running, the less you will have maintenance and security troubles).

Handlers are defined on a per listener basis. They can't be defined on a per virtual host basis because when the before accept Handler is triggered the Host header has not yet been received (so we can't tell which virtual host should be served).

If you need to define specific handlers for a given Web site, just dedicate a listener for this Web site. You can define any number of IP addresses for a single machine, even with a single network interface, making it easy to create new listeners.

At the moment, only one handler can be defined per listener (a more elaborated mechanism will later wave this limitation).

Content-Type Handlers

Content-Type Handlers are C scripts which let you define a custom behavior when a STATIC FILE using a specific MIME type is requested by a client.

This feature was initially created for *"FLV Pseudo-Streaming"*, the ability for a Web server to satisfy Flash Video Player requests aimed at letting users play any part of a movie without having to download it completely:

GET "/movie.flv`?start=123"` `// 123 is the movie.flv file's bytes offset`

The Adobe Flash player requires the use of this HTTP request parameter and of an opaque header (`"FLV\x1\x1\0\0\0\x9\0\0\0\x9"`) that has to be sent before data.

The Apache / Lighttpd / Nginx FLV streaming modules require 130-350 lines of C code.

G-WAN's Content-Type Handler is much easier to create and to use:

```
#define FLV_HEAD "FLV\x1\x1\0\0\0\x9\0\0\0\x9"

int main(int argc, char *argv[])
{
   char *query = (char*)get_env(argv, QUERY_STRING); // query: "start=200000"

   if(memcmp(query, “start=”, sizeof(“start=”) - 1))
      return 200; // HTTP status (200:'OK')

   http_t *head = (http_t*)get_env(argv, HTTP_HEADERS); // set HTTP bytes range
   head->h_range_from = atol(query + sizeof("start=") - 1); // checked by G-WAN

   // insert the FLV Header
   http_header(HEAD_ADD | HEAD_AFTER, FLV_HEAD, sizeof(FLV_HEAD)-1, argv);
   return 206; // HTTP status (206:'Partial Content')
}
```

To make it work, you just have to copy this code in a file named "flv.c" stored in the /gwan/listener/host/handlers folder and (re)start G-WAN.

Libraries

Third-party (shared *.so or *.dll) Libraries are pre-compiled code used to extend the features made available to C servlets.

Sometimes, you may need to use closed-source resources (either for security: compiled code is harder to alter than source code, or for licensing reasons: the source code of a feature you really need is not available, or for convenience: features already available from the operating system).

G-WAN lets you use the Boutell GD library to create dynamic pictures, the GNU GSL library for scientific calculations, the Crypto library of your choice, and so on.

Two directives let you specify which libraries you want to use:

```
   #pragma include “[path]”
```

lets you specify an additional path where #include files that you want to link with your servlet should be searched.

```
#pragma link "[path]modulename[.ext]"
```

lets you specify the (static or dynamic) libraries that you want to link with your servlet. It lets you link .c , .obj, .a/.lib and .so/.dll files (the `pragma link` order counts). By default, the extension is ".obj" if none is supplied.

Any open-source, commercial, system or custom-made shared library written in your favorite language can be used by G-WAN without modification nor dedicated interfaces.
Start with `/lib` and `/usr/lib` (SQLite is already there).

Applets

Applets will be illustrated at a later date when real-life examples will be available.

The purpose of G-WAN applets is to give the client-side as much power as you can already find at the G-WAN server-side.

As the C language has full-access to the low-level resources of a machine, sand-boxing will be used to isolate the scope of Applets.

Maintenance Scripts

The maintenance script has no defined purpose: you decide what it will do.

If present in gwan's directory, the script called `main.c` will be executed until it elects to terminate (or until G-WAN is closed).

It can access G-WAN's internal structures and performance counters, like C servlets, but unlike servlets (or Handlers) it is not aimed at working on HTTP requests. The maintenance script is intended to sleep when it is idle, that is, most of the time.

You can use it, for example, to run external tasks, run other scripts, backup files, send alerts, etc. You can even use the maintenance script to run completely unrelated C programs by using a new G-WAN instance (`./gwan -r script.c`).

A fille called `main.cxx` is provided in the `gwan` folder. Just rename this maintenance script example to `main.c` and it will show you how a maintenance script works.

Extending G-WAN further

You can use a C Servlet or an Handler (use the after_read state) to:

- redirect requests to another application server and then have G-WAN cache them,
- invoke compiled libraries (your compiled code invoked by your C script or Handler),
- call yet another script engine.

Just keep in mind that using external code will inevitably add overhead (slow-downs) and bugs to G-WAN.

The less you have layers of code, the safest (and fastest) your system will be.

G-WAN was created for this sole reason.

V. Build Your Own Server

G-WAN was initially designed for the HTTP protocol, but the G-WAN `protocol handlers` let you use G-WAN to quickly release an email or a database server.

Why would you want to do that?

G-WAN is faster and more scalable than others because of:

• a more optimized implementation	(using less CPU and memory resources)
• a better designed architecture	(scaling better on multicore systems)
• a taste for finding new solutions	(rather than doing copy & paste).

So, by building your `LDAP` or `XMPP` server on the top of G-WAN, not only you will safe time but your project will inherit from the qualities that makes G-WAN a better server.

Protocol Handlers

Protocol Handlers replace G-WAN's default protocol handler (HTTP). You can only have one Protocol Handler per listener.

For more information, see the `PONG.c` Protocol Handler example in the distribution archive.

Feedback

Suggestions are welcome, but as our time is limited try to follow the guidelines below:

- General questions are more appropriate on http://stackoverflow.com where people can answer your questions, reply, and also learn from the replies.
- Suggestions and feedback can be sent directly to pierre(a t)trustleap(.)com

If you contact us directly, then please:

- use a relevant subject in your email so we know what you want,
- please go straight to the point and give a reproductible example,
- be kind: there is always room for enhancements in a program.

If many software vendors do not let you contact them (or do it in a way that defeats its purpose), there is a reason: this is a very time-consuming process.

The only way to keep this service available is to respect its constraints: like you, I have another job.

Usage Terms and Conditions

These terms and conditions govern the distribution, licensing and delivery of the G-WAN software product to you by TrustLeap.

BY ACCEPTING DELIVERY OF THE PRODUCTS AND SERVICES DESCRIBED ON TRUSTLEAP'S WEB SITE OR OTHER TRUSTLEAP'S DOCUMENTATION, USER AGREES TO BE BOUND BY AND ACCEPTS THESE TERMS AND CONDITIONS OF USE UNLESS CUSTOMER AND TRUSTLEAP HAVE SIGNED A SEPARATE AGREEMENT, IN WHICH CASE THE SEPARATE AGREEMENT WILL GOVERN.

These Terms constitute a binding contract between user and TrustLeap. User acknowledges agreement and acceptance of these Terms by making installing the software. These Terms may change without prior notice.

We reserve the right to make adjustments to distribution terms, products and service offerings for reasons including, but not limited to, changing market conditions. We make every effort to ensure the accuracy of the information available on our web site. However, the documents and graphics published on this site may contain technical inaccuracies or typographical errors. We make no representations about the suitability of the information and graphics presented on this site. All such documents and graphics are provided "as is" without warranty of any kind.

Restrictions on Use

"Modifications" means any addition, alteration or deletion from the original files distributed by TrustLeap.

TrustLeap hereby grants you a world-wide, royalty-free, non-exclusive license to copy and distribute the binary code versions of G-WAN at the condition no Modifications are made either to the files made available to you by TrustLeap.

"Response header" is the part of the response message output by the G-WAN Web server, containing but not limited to, header fields for date, content-type, server identification and cache control.

"Server identification field" means the field in the response header which contains the text "Server: G-WAN/x.x.x" where "x.x.x" is the program version number.

You agree not to remove or modify the server identification field contained in the response header.

In consideration for the license granted by TrustLeap to you, you may promote G-WAN by displaying the G-WAN 'powered' logo http://gwan.com/imgs/trustleap_pw.gif in marketing and promotional materials such as the home page of your Web site, other Web pages or any other document. You also agree that TrustLeap may identify your organization as a G-WAN user in conjunction with its own marketing efforts.

This agreement will terminate immediately and without further notice if you fail to comply with any provision of this agreement. Upon termination, You agree to uninstall or destroy all copies of the G-WAN files.

Distribution

Provided that you do not modify the files of the distribution archive you are hereby licensed to make as many copies of the software and documentation as you wish; give exact copies of the original version to anyone; and distribute the software and documentation in its unmodified form via electronic means.

There is no charge for any of the above.

You are specifically prohibited from charging, or requesting donations, for any such copies, however made.

Obtaining the Latest Version

Before distributing G-WAN please verify that you have the latest version. The latest version is always available on our website http://trustleap.ch/

Suggested One Line Program Description

G-WAN is a Web Application Server with 'edit & play' C servlets

Suggested Description

G-WAN is a Web server and an application server with C servlets and the whole takes 150 KB of code in addition to be far faster than other available Web servers.

C servlets are 'edit & play' scripts that let you use the power of C with the convenience of scripts. G-WAN is free for all. Feel free to distribute it around you!

Requirements

G-WAN Requires Linux (32-bit or 64-bit)

Copyright notice