Config-Definition-View Webserver

Michal Novotný

Draft v0.0.2

This document introduces the new CDV (Config-View-Definition) webserver that's basically the C application that allows web-pages to be written in a special language. The CDVWebServer has been partly inspired by Python programming language and partly by Bash language. The language itself is different in some respects however the structure is mostly designed for the webserver. There are several main requirements for the webserver, mainly to have the webserver lightweight but module-enabled with the config-definition-view directories separated. Of course, the webserver is not intended to be running under the superuser (root user) so the user impersonation should be encouraged. The system itself should be written with built-in .htaccess-like functionality as well as the minCrypt encryption system implemented directly into the core functionality of the system.

Document overview

1) Brief information

The system itself should be HTTP web-server that's using the functionality of standalone web-servers to implement the HTTP protocol itself. The system should be designed to be very lightweight as based on the patterns and definitions with displaying using the view information. There should be a possibility to load/enable the modules for the system however the modules could be loaded on-demand instead of loading all of them per each request. The virtual hosting and user-dirs could be enabled in the means of modules.

Another thing to implement should be the .htaccess' like functionality directly into the core functionality of the system.

For the standalone web-site there should be 3 directories containing all the data per each part, i.e. The directory for configurations (config directory), directory for definitions (definitions or defs directory, have to decide) and the views directory.

The config directory should support the include clause or it should automatically try to include the config.d directory where the definitions could reside. Mainly the sql-server definitions and others necessary.

The definitions directory should be containing all of the functions per each page to be used inside of user views. The views should be just a special tag-enabled HTML pages showing the real layout of the pages and should be the only part of the site accessible using the web-browser.

The view could look like as follows:

```
<html>
 <head><# print %page_title #></head>
  <body>
    <# print “Some data”, %page_name #>
  </body>
</html>
```

2) Technical details

The web-server should support modules to be enabled in the configuration file however the modules may be loaded on-demand, i.e. Not loading them when not necessary to load them. One of the core functionality should be the .htaccess/mod_rewrite-like translation of the URLs, i.e. The request like:

GET /index.html

Should access the index.view of the views directory and the index.def of the definitions directory. The page server to the client should be basically the index.view HTML page with the special tags translated using the index.def definitions.

The filesystem structure should be:

```
 <root> / config (should include config.d)
          / config.d / ... files ...
          / definitions / index.def
          / views / index.view
                    /  default.css
```

Where just the views part will be web-accessible. The definitions will be internally translated and used for substitution to the view file.

The minCrypt-encryption will be possible by default if the libminCrypt library is found and could be loaded directly into the system. This could be used for the database-encryption and/or to send/receive data encrypted using this algorithm. This needs the support on the second-side, i.e. On the client to send the data in such encrypted format.

The application won't be running under the user but it will be impersonated as some other user (at least when running under the root context).

3) Command-line argument support

Although this system is meant to be mainly HTTP server there should be the optional command-line arguments that will be used mainly for testing:

a) –test <filename>

This will test the integrity of data in the filename.

b) –test-server <port> [<project-name>]

This will run a test server on <port>, optionally only for the project <project-name> (or all projects).

c) –debug

This will enable the debugging output of all commands and queries run by the script/system.

d) –shell

This will run the interactive shell to provide all of the functionality as mentioned above and more, mainly:

1) load project-name
2) loadfile filename
3) run the command from file and return output

4) Roadmap

The system doesn't exist yet and should bring a new system for web hosting. The system is designed to be lightweight and should be implemented in the library form. This library is expected to have no more than several KiB (which is nothing in comparison to Apache web-server) and should have very few dependencies.

As the part of the system core there should be the XML-parser (libxml2) that will be also used for the definition file parsing. There should be a module for optional caching support of the requests and the data to be returned to the client's web-browser implemented by module.

Modules to be implemented should include:

1) Database modules – should support wide-spread databases
   a) MySQL
   b) PgSQL

2) Caching module – could be disabled by 'no-cache' header of configuration field
   a) Caching into a shared memory
   b) Caching into a database
   c) Caching onto a disk file

5) Implementation details

There should be 3 main low-level functions implemented directly in the webserver's library and also 2 high-level functions to use all of them as based on the minimal data accepted in a single parameter. All the functions mentioned should be exposed to the public API.

Low-level functions should be:

1) loadConfigs(char *config_path)
   a) The config loader should support the inclusion of another directory since there should be configuration files split into several directories e.g. using *include config.d/**
   b) Also should support *load_module* directive to load some module necessary, like caching module or mysql module
2) loadDefinitions(char *defs_path)
   a) Definitions file contain the functions to be used on the page, the function should be using the XML pseudo-programming language with saving the data into the internal structures for the process
3) loadViews(char *view_path)
   a) The view should basically be the HTML formatted web-page used mainly as a template with processing done inside the <% and %> blocks that are basically meant for programming. The data should be substituted by the function results, etc.

High-level functions:

1) loadProject(char *project_file)
   a) This function should accept the project file that contains the definition of paths for the project, no need to use once the project resides in one directory and it's subdirectories (as it should) – for this purpose calling the loadProjectDirectory() would be better
2) loadProjectDirectory(char *project_path)
   a) Should be wrapper to get all the (i.e. config, definition, view) paths and call Load* functions with appropriate parameters

6) Database and query support

The CDV-Webserver should support querying the database using the modules available in the modules directory of server root directory. The database tables have to be created first which should be done in the definition file as well. The example of database tables creation follows:

```
<database>
    <table name="photogallery">
        <structure>
            <column type="int" primary="true|1">id</column>
            <column type="char" max_length="128">name</column>
            <column type="file" name_length="128">file</column>
            <column type="bool" default="False">public</column>
        </structure>
    </table>
</database>
```

In this example the new table named *photogallery* will be created if it doesn't exist or checked for the same structure if the database table already exists. Administration/user should be asked to alter DB as based on definition file.

The query should be defined using 2 approaches, the exact queries should be supported using CDATA tags as well as specially-formed definitions in the definition files like in the following example:

```
<query type="view|add|edit|delete" [=="SELECT|INS|UPD|DEL"]>
    <table>photogallery</table>
    <fields>
        <field>name</field>
        <field>file</field>
    </fields>
    <conditions>
        <condition type="eq">
            <field>public</field>
            <field>0</field>
        </condition>
    </conditions>
</query>
```

7) Internal database support

The CDV webserver should incorporate the internal database system similar to the standalone database systems. The system should be able to create databases and tables of various fields and field count, similar to the structures like theses:

```
typedef struct tTableDef {
	long id;
	char *name;
	int numFields;
} tTableDef;
typedef struct tTableFieldDef {
	long id;
	long idTable;
	char *name;
	int type;
} tTableFieldDef;

typedef struct tTableFieldDataInt {
	long id;
	 long idRow;
	int iValue;
} tTableFieldDataInt;

typedef struct tTableFieldDataLong {
	long id;
	 long idRow;
	long lValue;
} tTableFieldDataLong;

typedef struct tTableFieldDataStr {
	long id;
	 long idRow;
	char *cValue;
} tTableFieldDataStr;
```

There will be the communication between child processes and the parent process. The parent process will broadcast data change to all of the child processes with usage of the temporary files for the string data. The data types should be as follows:

int
long
shortstr - 1st byte for length, up to 256 bytes
mediumstr - 2 bytes for length, up to 65536 bytes
str - 3 bytes length, up to 16 MiB
longstr - 4 bytes length, theoretical limit up to 4 GiB
file - save to a separate file on disk

Database format:

CDVDB<totalHeaderSize><header>[<totalDataSize><data>]

header:

offset 00 - 02 = header length (3 bytes)
offset 03 - 04 = table count (2 bytes)
offset 05 - length = table definition data (sizeof(tdd) * numTables)

table definition data:

offset 00 - 01 = table definition length (2 bytes)
offset 02 - table name length (tnl)
offset 03 - tnl = table name
offset tnl+1 - table field count
offset tnl+2 - length = field definition data (sizeof(fdd) * numFields)

field definition data:

offset 00 = field defition length
offset 01 = type in this byte
offset 02 = field name length
offset 03 - length = field name

There should be the option to save just tables definition and also the option to save the data, but be aware of fragmentation.

8) Scripting language support

There should be support for the low-level scripting language to be used instead of standard definition. The file name should either end with the *cdvscript* extension or there should be a separate scripts directory in the file system structure. The language should be similar to standard languages like PHP and should be defined as follows:

```
<#
    define id auto;   // This should be automatic but good example how to
                      // override the data type of the variable, id will be
                      // initialized to null when having the define

    id = has_key(GET, “id”) ? GET[id] : null;
    // but this will be the same as:
    id = has_key(GET, “id”) { GET[id]; }
    // and the same as:
    id = has_get(GET, “id”) GET[id];
#>
```

But in the following example it will bail in case of GET[id] of string.

```
<#
    define id int;      // we preset the value data type to be int always

    // we have GET request with query string of ?id=some_string
    // the script will automatically end and bail with error message
    id = has_key(GET, “id”) GET[id];

    // Since the id value is defined as int, we have to have GET[id] as int,
    // but we may trap the exception using:
    try
        id = has_key(GET, “id”) GET[id];
    except
        print “Invalid id value. Exception is: %s”, exception.message;
#>
```

The scripting language will also support the POST requests and FILE requests. Uploading the files should be made easy as follows:

```
<#
    define photo file;      // file type is basically a string type with the
                            // information it has to do the upload first
    // The following definition would override it to string of original name
    photo_name = has_key(FILES, “photo”) FILES[photo];

    // This will print the file name of the uploaded file or null for no file
    // uploaded, type will be automatically changed to string
    print “Photo uploaded is saved as: %s”, photo

    // This will save it to the database table called photos using definition
    // from the definition file, the table may be null if there's just one table
    // for this view in the definition file, the query will be defined in the
    // definition file, save_db will add the new row, edit_db will edit row
    // by id and delete_db will delete the row by id (requires id arg. first)
    save_db(“photos”, photo, photo_name);
    edit_db(null, id, “photo”, “photo_name”);
#>
```

Definition file for example above:

```
<query type=”add”>
    <table>photos</table>
    <fields>
        <field>
            <name>filename</name>
            <value>%1</value>
        </field>
        <field>
            <name>original_filename</name>
            <value>%2</value>
        </field>
    </fields>
</query>
```

9) Examples

config file

---

sql_type = <sql_type> (e.g. 'mysql')
sql_user = <sql_username> (e.g. 'web')
sql_password = <sql_password>
sql_db = <sql_database_name>

load_module mysql
[include config.d/*]

---

definitions/photogallery.cdv

```
<page-definition>
  <include type=”config”>config-db</include>
  <functions>
    <function name=”get_photos” return=”array”>
      <params>
         <param name=”album_id_from_GET”>album</param>
      </params>
      <query-exact>
       <![CDATA[
        SELECT id, name FROM photogallery WHERE album_id = %album
       ]]>
      </query-exact>
    </function>
    <function name=”get_photo_albums” return=”array”>
      <query-exact>
       <![CDATA[
        SELECT id, name FROM photogallery_albums
       ]]>
      </query-exact>
    </function>
  </functions>
</page-definition>
```

views/photogallery.view

```
<html>
 <head><# print %page_title #></head>
  <body>
    <#
       if (exists GET, album_id) {
          include photogallery-photos
       }
       else {
          include photogallery-albums
       }
     #>
  </body>
</html>
```

views/photogallery-albums.view

```
<h1>ALBUM LIST</h1>

<#
tmp = get_photo_albums
for album in %tmp {
   print “<a href='?album_id=”, album['id'],”'>”, album['name'], “</a><br />”;
}
 #>
```

views/photogallery-photos.view

```
<h1>PHOTOS</h1>

<#
tmp = get_photos [album_id=1]
for photo in %tmp {
   [or: for photo in %(get_photos)]
   print “<a href='?photo_id=”, photo['id'],”'>”, photo['name'], “</a><br />”;
}
#>
```