Welcome to Cliqon

Cliqon is a website development system and application development system, created with PHP server-side scripts plus a combination of HTML, CSS and Javascript files to power a content management and administration system. It utilises the latest programming methodologies. It allows designers and developers to create web sites and applications for customers, extremely quickly and without many of the limitations that appear prevalent in competing systems.

It has been designed to resolve four key issues in modern European web site design and development:

- dynamic database driven
- fully multi-lingual
- presentational design independent
- flexible and feature rich

Any of these items, when taken on their own, is a justification for choosing Cliqon as your preferred web site development environment or framework but when combined, they represent a compelling case for choosing Cliqon for the infrastructure and framework for your next web site.

Dynamic

Every textual and image element of a Cliqon site is obtained from either the database or a text file. The database of choice is MySQL but other varieties are supported. This database can be maintained on-Cliqon using a comprehensive database management toolset. Therefore the textual content can be created, updated and deleted whenever required and these changes are immediately reflected on the website.

There are a significant number of benefits that a dynamic web-site can deliver to an owner. For example, it is an essential element of good website marketing that whenever Search Engines index the content, they find that the content is constantly being updated. This improves the ranking of the web site in the search engine listings. Another benefit is that a Cliqon derived web site can be used as an on-Cliqon catalogue for a variety of products and other items such as boats and properties and these listings can be constantly maintained and kept up to date.

Multi-lingual

Unlike other, so called, "multi-lingual" content management systems, every aspect of the Cliqon web program supports multi-lingual content, for example - catalogue items, form prompts, images with embedded text, flash movies, drop down lists, help texts and so on.

Cliqon can support any number of idioms and has been tested with most European languages. Cliqon utilises the Google translation facilities to provide a machine translation of the textual content from the master language to the other languages of the web site at the time of entry into the database not at the time of display. A machine translation is an excellent way of starting the process of content translation but will need refining and improvement in due course.

For the 25 “jurisdictions” (countries, regions or areas) of Europe where the use of more than one language is normal, the way that Cliqon has been designed to support multiple languages is a unique method.

Design independent

Cliqon is a web design toolset with which to build web sites rather than a content management system that utilises templates. Each type of system has its own advantages and disadvantages. The Cliqon toolset can be used to underpin any web site design, that can be conceived, based on a CSS stylesheet. It can also be used to turn an existing static or single language web site into a dynamic, multi-lingual version. There are virtually no limits to what can be supported with Cliqon. However it does require a more comprehensive understanding of Dynamic HTML to achieve these graphical results.

Flexible and feature rich

Cliqon was first conceived in 2005 and has undergone continuous development since that time. It now contains a wide range of useful modules and widgets. These additional functions and facilities are intended for use by both visitor and administrator alike. The module plugin system is used by the authors of Cliqon to enhance the toolset and is also available to developers so that they can add facilities for their own customers and make the facilities available to other developers on a open source or paid basis.

Introduction

The Documentation for Cliqon is divided into a number of sections. Each section is an item at the root of the documentation tree.

Installation

The section on Installation provides explanation and help about the automated installation process which guides you through the basic steps of installing a Cliqon system. If there are some reasons why the wizard cannot be used, the Section also contains a series of pages explaining how to setup the system manually.

Configuration

The section on Configuration describes in detail all the standard configuration values for a default Cliqon system. It explains how these values are mostly stored in TOML format in either text files on the server or as textual entries in the database.

The section describes the Cliqon TOML format. It describes how the configuration values are used to manage the inputs and outputs of the system including forms, displays and reports. It explains how to tune existing values for your own needs and how to create new files and database values for your own development needs.

Administration

The Administration system of a Cliqon internet system will serve one or both of two functions. If the toolset has been used to create a presentational website, then the owners and operators of the website will use the administration modules to create and maintain the content of the website. If the Cliqon system has been used to create an intranet type system such as a project management or CRM system, the enhanced administration system may be the complete application in itself. Finally Cliqon may have been used to build a combined system such as an eCommerce website consisting of on-line catalogue, cart and order processing module, in this case the administration module will be enhanced to offer all the necessary functions.

The Administration section of the documentation will explain all the functions of the standard Cliqon system in terms of what administrative processes are supported – the ability to enter data and content into the database, the ability to process that data for different purposes (usually associated with the use of Cliqon as an application – for example: eCommerce, CRM or Auction), and view the processed data in a format that is appropriate to the type of data – images in a gallery, data in a table or grid, locations and menus in a tree format and events in a calendar format.

The section will explain how to import and export data, how to configure a grid or list, how to design forms and reports, how to create and maintain languages, how to create menus, how to convert configuration files into values held in the database, the caching and logging system, how to maintain an online help system, how to use the portal system for useful links, news articles, blog and events, plus a series of pages explaining the administration utilities such as system update, site map, data dictionary and language strings used in the javascript.

How to develop your own administration modules is covered in System Build.

Installation

The section on installation in the Cliqon documentation has two subsections. The first describes the automated process of installing a Cliqon production system using the installation Wizard. The second subsection describes the process of installing Cliqon manually.

Common steps

Both subsections assume that the Zip file has been downloaded and uncompressed or unzipped to a suitable location on a web server that supports PHP and preferably Version 7.0 or above. Cliqon has also been tested on PHP 5.6 but is not designed to work on versions of PHP prior to 5.6.

A note on web server configuration files

Cliqon requires that all access to the system are channeled through "index.php" at the root of the system. This is normally achieved by creating a *".htaccess"* file on an Apache webserver, a "web.config" file on Internet Information Server (IIS) and on NGINX, you will create entries in the main Nginx configuration file.

These files are collectively known and defined as web server configuration files that contains commands known by the server that tell the server how to behave in certain instances.

Shown below are the standard files supplied with Cliqon for Apache and IIS. Then we offer a proposal for Nginx

Sample .htaccess

```
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond $1 !^(index\.php)
RewriteRule ^(.*)$ /index.php/$1 [L]

FallbackResource /index.php
```

Sample web.config

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <rewrite>
            <rules>
                <rule name="Imported Rule 1" stopProcessing="true">
                    <match url="^(.*)$" ignoreCase="false" />
                    <conditions logicalGrouping="MatchAll">
                        <add input="{REQUEST_FILENAME}" matchType="IsFile" ignoreCase="false"
                         negate="true" />
                        <add input="{REQUEST_FILENAME}" matchType="IsDirectory"
                         ignoreCase="false" negate="true" />
                        <add input="{R:1}" pattern="^(index\.php)" ignoreCase="false"
                         negate="true" />
                    </conditions>
                    <action type="Rewrite" url="/index.php/{R:1}" />
                </rule>
            </rules>
        </rewrite>
    </system.webServer>
</configuration>
```

Recommended nginx configuration

```
location / {
  if (!-e $request_filename){
         rewrite ^(.*)$ /index.php break;
  }
}
```

Installation files

There are three specific files and one subdirectory of files that are used by the installation process and maybe safely removed or archived after the installation process is complete.

At the root of the Cliqon system is a blocking file entitled “notinstalled”. The existence of this file causes the installation template to be displayed from within the install subdirectory. When this file is removed, Index.Php will load and process “startup.php” instead of “install.php”.

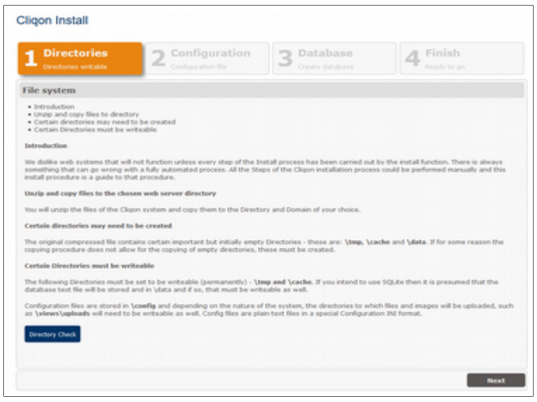
As explained previously, the installation process uses “Install.Php” in the \includes directory and this maybe removed. Similarly, the installation process has its own Controller called InstallController.Php in the \controllers subdirectory, which may be removed.

Finally, the majority of the Install files are located in the \install subdirectory and this may be removed in its entirety.

Wizard Installation

The automated install process comprises a four step process and provides its own detailed on-screen instructions and help. The main installation page explained that you will have downloaded one of the two standard versions of Cliqon – either with or without a demonstration front end with associated data.

Step 1



File system

- Introduction
- Unzip and copy files to directory
- Certain directories may need to be created
- Certain Directories must be writeable

Introduction

We dislike web systems that will not function unless every step of the Install process has been carried out by the install function. There is always something that can go wrong with a fully automated process. All the Steps of the Cliqon installation process could be performed manually and this install procedure is a guide to that procedure.

Unzip and copy files to the chosen web server directory

You will unzip the files of the Cliqon system and copy them to the Directory and Domain of your choice.

Certain directories may need to be created

The original compressed file contains certain important but initially empty Directories - these are: **\tmp, \cache** and **\data**. If for some reason the copying procedure does not allow for the copying of empty directories, these must be created.

Certain Directories must be writeable

The following Directories must be set to be writeable (permanently) - **\tmp and \cache**. If you intend to use SQLite then it is presumed that the database text file will be stored and in \data and if so, that must be writeable as well.

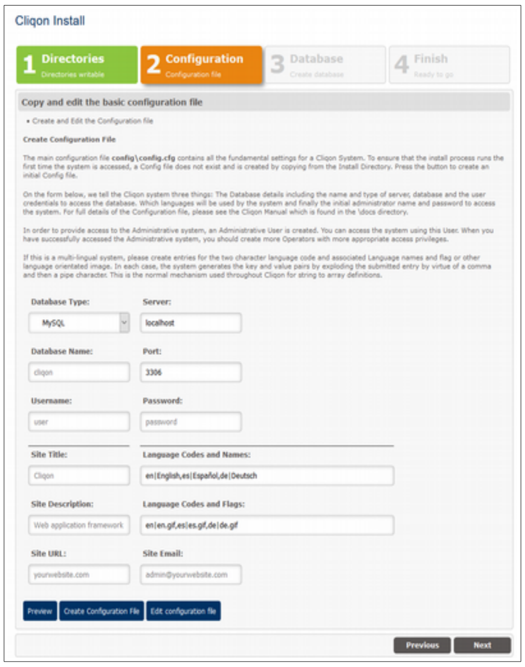
Configuration files are stored in **\config** and depending on the nature of the system, the directories to which files and images will be uploaded, such as **\views\uploads** will need to be writeable as well. Config files are plain text files in a special Configuration INI format.

In each case you should uncompress the Zip file to web server directory of your choice. We have tried to include the necessary empty directories in the compressed file but some mechanisms for decompressing a Zip file may not honour these sub-directories.

On the first screen of the Wizard we provide the means to check the existence of the necessary sub directories and ensure that they are writable.

Step 2

Cliqon requires the existence of a Configuration file that resides in /config and is known as config.cfg. The Configuration file is in TOML format. We provide a template for the Config file, named config.txt. It contains certain placeholders which will be updated from the Form on Step 2, with live values before the Config.Txt is converted to Config.Cfg.



Copy and edit the basic configuration file

- Create and Edit the Configuration file

Create Configuration File

The main configuration file **config\config.cfg** contains all the fundamental settings for a Cliqon System. To ensure that the install process runs the first time the system is accessed, a Config file does not exist and is created by copying from the Install Directory. Press the button to create an initial Config file.

On the form below, we tell the Cliqon system three things: The Database details including the name and type of server, database and the user credentials to access the database. Which languages will be used by the system and finally the initial administrator name and password to access the system. For full details of the Configuration file, please see the Cliqon Manual which is found in the \docs directory.

In order to provide access to the Administrative system, an Administrative User is created. You can access the system using this User. When you have successfully accessed the Administrative system, you should create more Operators with more appropriate access privileges.

If this is a multi-lingual system, please create entries for the two character language code and associated Language names and flag or other language orientated image. In each case, the system generates the key and value pairs by exploding the submitted entry by virtue of a comma and then a pipe character. This is the normal mechanism used throughout Cliqon for string to array definitions.

Database Type:	Server:
MySQL	localhost
Database Name:	Port:
cliqon	3306
Username:	Password:
user	password
Site Title:	Language Codes and Names:
Cliqon	en|English,es|Español,de|Deutsch
Site Description:	Language Codes and Flags:
Web application framework	en|en.gif,es|es.gif,de|de.gif
Site URL:	Site Email:
yourwebsite.com	admin@yourwebsite.com

You will need to provide the usual information to create a Database connection. In the case of SQLite, only the name of the database file is required and it is presumed to reside in /data, once created.

The form contains two form fields related to multiple languages. Every web framework that supports multiple languages has their own way of handling them. The history of Cliqon is that it was designed on the island of Mallorca where there are two official languages – Spanish and Catalan. In addition, the existence of significant numbers of residents from Great Britain and Germany, means that any web project for the island requires support for four languages. We did a great deal of research to establish the best way to handle multiple, concurrent languages and Cliqon embodies this research in its design.

We have tried as best as we can to ensure that Cliqon has no base language. If one exists in certain places – such as in error messages that we cannot acces, that language will be English. All languages are treated equally. If Cliqon is to be used in a single language environment, then one code, language name and flag should be created. The placeholders for the fields show how three languages might be handled. The key and value that make a language pair or set are separated by a vertical bar. Each pair is separated by a comma. For a single language website, you will provide one key|value pair.

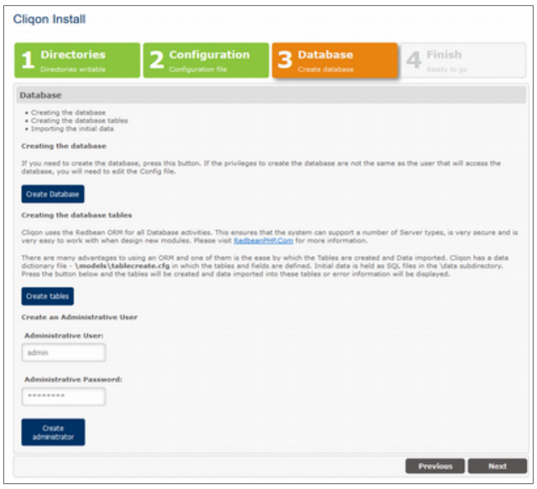
You will note the Configuration file conversion process generates a set of arrays for languages. We supply English and Spanish in our demonstration.

For more information, please the Language subsection within Configuration.

You can get a simple preview of your entries on the form before pressing the Create button. You can edit the configuration file after it is created in a popup editor. It is recommended that you read the Section on Configuration in this manual before attempting a manual edit.

Step 3

The third step covers three possible areas. In the case of a SQL database connection (as opposed to SQLite), if the database username and password can create a database, then you can invoke that process using the first button. If not, you will need to create the Database manually.



Database

- Creating the database
- Creating the database tables
- Importing the initial data

Creating the database

If you need to create the database, press this button. If the privileges to create the database are not the same as the user that will access the database, you will need to edit the Config file.

Creating the database tables

Cliqon uses the Redbean ORM for all Database activities. This ensures that the system can support a number of Server types, is very secure and is very easy to work with when design new modules. Please visit RedbeanPHP.Com for more information.

There are many advantages to using an ORM and one of them is the ease by which the Tables are created and Data imported. Cliqon has a data dictionary file - **\models\tablecreate.cfg** in which the tables and fields are defined. Initial data is held as SQL files in the \data subdirectory. Press the button below and the tables will be created and data imported into these tables or error information will be displayed.

Create an Administrative User

Administrative User: admin

Administrative Password: ********

Creating the tables now does two tasks – creating the tables and populating them with initial and essential data. As explained previously, the dataset may consist of a basic set which contains or does not contain entries for a demonstration front end.

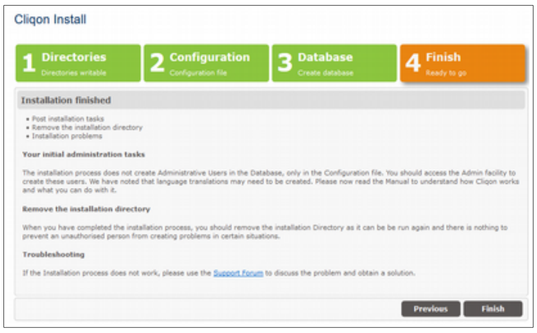
The reason we have to undertake the two tasks of table creation and initial data import, is because we use an ORM. Redbean does the process in reverse order, as most ORMs do, that is, it creates the columns of the table as it inserts the data for that column.

For more information, please the Database subsection within Configuration.

The third and final action on Steps 3, is the creation of the master administrative user. Who will be used to access the Administration system for the first time.

Step 4

The fourth and final step reminds you to remove the installation directory and other files. Pressing finish also deletes the Blocker file as explained previously and redirects you to the Login screen for the Administration system.



Installation finished

- Post installation tasks
- Remove the installation directory
- Installation problems

Your initial administration tasks

The installation process does not create Administrative Users in the Database, only in the Configuration file. You should access the Admin facility to create these users. We have noted that language translations may need to be created. Please now read the Manual to understand how Cliqon works and what you can do with it.

Remove the installation directory

When you have completed the installation process, you should remove the installation Directory as it can be be run again and there is nothing to prevent an unauthorised person from creating problems in certain situations.

Troubleshooting

If the Installation process does not work, please use the Support Forum to discuss the problem and obtain a solution.

Manual Installation

You have downloaded the Cliqon install ZIP from the Cliqon web site. You have uncompressed the Zip file into the root of the production or development web site. You will have attempted to run the site and been presented with the Wizard install setup. However for one reason or another you wish to setup the website manually. The steps in the process mostly cover the same areas as the automated process.

Check Directories

Please ensure that the following subdirectories exist and that they are writable. All references start at "/" (root):
/tmp, /cache, /log, /config, /admin/cache, /admin/config, /views/uploads and **/views/thumbs**

Copy and edit the Config file

In the **/install/config** subdirectory is a dummy configuration file entitled, **Config.Txt**. Copy this file to **/config** and rename to **Config.Cfg**.

Edit the Config file with a text editor. You will observe that it is in TOML format. For more information on TOML please read the appropriate section in this documentation. In principle, TOML is a method for presenting or storing a multi dimensional array in a straightforward and readable format.

There are certain sections and entries in the TOML that require attention and amendment before the new Cliqon system can be used. It should be noted that Config.Cfg is effectively single langauge and all the entries within the subarray "site" can be overwritten with multi-lingual entries within the Administration system.

The Config file holds the main database access configuration. We draw your attention to the original purpose of the Config.Txt file (that you copied over) is to act as template for a Configuration and during the process of installation, template variables wrapped in curly brackets {} will be overwritten with actual values. This is particularly relevant for the database subarray where all of the entries need to be amended to actual values. The ORM within Cliqon depends upon the existence of the PDO drivers so that it can provide an equal activity, irrespective of the SQL Server or Sqlite variety in use.

You will replace the values wrapped in curly brackets, change the the default values to values appropriate for your use of Cliqon.

Below the main site subarray are two sets of entries for the languages. As explained previously, Cliqon requires entries here, even if the system is single language.

For a single language you would enter:

```
[site:idioms]
en = 'English'
```

And for multiple languages, you might enter

```
[site:idioms]
en = 'English'
fr = 'Francais'
```

If you wish to use flag icons in your front end website design you might enter:

```
[site:idiomflags]
en = 'english.png'
fr = 'french.png'
```

As part of your website design process, if you want language flags, you will place them in an appropriate directory of the /views subdirectory such as /views/img.

When you choose to create the system manually, you have no suitable way to enter an administrative username and password for the first time you login to the Admin system. The solution we have adopted to is to record an override in the configuration file. You should complete the details manually. Once you have logged into the Admin system for the first time and have the opportunity to create Administrators and Operators in the database, it would be sensible to remove this subarray from the Configuration file as it will no longer be needed.

Some of the entries in the configuration file and within the site subarray are optional, such as a Google API key for use by Gmaps.Js as an example. You are welcome to add more entries to the Configuration file for your own purposes and we will explain in the section on Configuration how to access these values.

Setup the Database

Having completed the initial edit of the configuration file, you can proceed to setup the database. You will use a SQL editor of your choice, such as MySQL Workbench, Dbeaver or a browser based SQLite editor to create the base database. Obviously you will create this database in line with the entries that you have made in the Configuration file.

Import the tables, columns and database

If you are using a SQL Server to store the Cliqon data, use the same SQL editor to import the tables, columns and data into the database. Look in the **/data** subdirectory and you will find a series of SQL files that correspond to the tables in the database or schema. Open the file in a Query and execute it, restore it using data import or just simply name and run the file. Each SQL file will setup its columns and restore its data (if appropriate). As an example, "dbcollection" holds all the records to run the administrative system and more, whilst "dbitem" contains only a table columns create function. Please remember to "USE" the database you have just created. Once you have imported the data manually, you may want to delete these files from the filesystem.

If you are using Sqlite, visit the Cliqon website to download the latest Sqlite database template that you can use to start the Cliqon system. When you have downloaded the file and unzipped it, you will rename it to whatever you entered as the dbname variable and it is accessed from the /data subdirectory.

Delete the blocking file

The reason that Cliqon executes the install routine, rather than displaying the main web page, is the existence of the "notinstalled" blocking file. Once you have edited the Config file and setup the database, you can delete the "notinstalled" file and the main system will run.

Delete the install files and directories

See the main or parent Installation page to remind yourself which files and subdirectories you can safely delete once the installation is complete.

Login to Administration

Finally you can login into the Admiistration system. The landing page for a production Cliqon system or the demonstration system, both contain links to the Administration system which is accessed via http://sitename.com/admin/. You may wish to provide an link from your own design to the Admin system.

Create a proper administrator in the database

Having logged into the system you should create your own Administrator and system Operators before deleting the admin users sub array from the Configuration file.

Configuration

A Configuration array snippet is the basic building block of a Cliqon system. The array snippet is expressed in plain text TOML format and is stored within a Cliqon system as either database record or a physical config file (a file with a .cfg extension). A developer will tend to have a set of the original development configuration files to hand when developing a new system but will copy or transfer these to the database before publishing a production system.

When Cliqon configuration files or records are parsed by the Config or Toml Classes, multi-dimensional arrays are produced. These arrays are used by every element of a Cliqon system – database access strings, forms, reports and display screens.

Cliqon configuration records can operate at up to three levels. We call the three levels, Services, Collections and Models. Configuration records cascade from the general to the specific. Like CSS Style commands the child values augment or override the parent values.

The concept of a service configuration record describes a facility such as a Datagrid, Datalist, Calendar, an administrative or main web site Menu, a Report or Siteupdate.

The concept of a collection record describes a Cliqon table or database schema. It contains information about the fields of the table and array snippets for any services that the table might need to enter, manipulate, process and display data.

The concept of a model record describes a table type schema. It contains specific information about how a particular service is configured for a given tabletype.

Although the way that Cliqon is divided into Collections and Models is explained in greater detail in the next sections, it is important to summarise the distinction here so that the two paragraphs above make sense. Most older frameworks and traditional web applications utilise a database table schema for every possible activity in the system. Displayed is a schema for a modern Wordpress. Whilst it has been refined over the years, it still treats Posts and Comments as discrete tables. Cliqon does not have such a division. As a general statement, the seven tables of a basic or standard Cliqon system are grouped into two – front-facing: dbitem and dbarchive, administrative: dbcollection, dbsession, dblog, dbuser and dbindex. Alternativel, one can group them by purpose – single or multi purpose. Tables dbcollection, dbitem are multi-purpose and the rest are single purpose. When we use the concept of multi-purpose, we mean that the table contains different models or table types. The dbitem table contains models for news, sections, components, library, events, image references, and useful links. There are a series of columns which specified tasks such as placing the records of a given menu subtree into an order or providing access control parameters for the record. As a general rule the multi-lingual content for any record is stored in JSON format in one field / column.

We contend that using this mechanism to define table data makes a Cliqon system extremely flexible. An additional column of content can be configured in a configuration record and because Cliqon uses an ORM, the new data field will automatically be included after the cache has been deleted and the new process first used.

TOML

The letters TOML stands for Tom's Obvious, Minimal Language. It has been developed by Tom Preston-Werner (former CEO of Github). For more information please visit https://github.com/toml-lang/toml/.

TOML aims to be a minimal configuration file format that's easy to read due to obvious semantics. TOML is designed to map unambiguously to a hash table. TOML should be easy to parse into data structures in a wide variety of languages. In some ways TOML is very similar to JSON: simple, well-specified, and maps easily to ubiquitous data types. JSON is great for serializing data that will mostly be read and written by computer programs. Where TOML differs from JSON is its emphasis on being easy for humans to read and write. Comments are a good example: they serve no purpose when data is being sent from one program to another, but are very helpful in a configuration file that may be edited by hand.

Cliqon includes two Classes for handling TOML files. We provide a standard 3rd party Toml Class and we also have a modified version for the use of Cliqon, which is the Config Class.

The Basics

The basic form is key = value

```
# Comments start with hash or a semi-colon ;
foo = "strings are in quotes and are always UTF8 with escape codes: \n \u00E9"

bar = """multi-line strings
use three quotes"""

baz = 'literal\strings\use\single\quotes'

bat = '''multiline\literals\use
three\quotes'''

int = 5 # integers are just numbers
float = 5.0 # floats have a decimal point with numbers on both sides

date = 2006-05-27T07:32:00Z # dates are ISO 8601 full zulu form

bool = true # good old true and false
```

Lists

Lists (arrays) are signified with brackets and delimited with commas. Only primitives are allowed in this form, though you may have nested lists. The format is forgiving, ignoring whitespace and newlines, and yes, the last comma is optional (thank you!):

```
foo = [ "bar", "baz"
        "bat"
]

nums = [ 1, 2, ]

nested = [[ "a", "b"], [1, 2]]
```

Arrays and Tables

Foo and bar are keys in the table called table_name. Tables have to be at the end of the config file. Why? because there’s no end delimiter. All keys under a table declaration are associated with that table, until a new table is declared or the end of the file. So declaring two tables looks like this:

```
# some config above
[table1]
foo = 1
bar = 2

[table2]
        foo = 1
        baz = 2
```

The declaration of table2 defines where table1 ends. Note that you can indent the values if you want, or not. TOML doesn't care.

If you want nested tables, you can do that, too. It looks like this:

```
[table1]
        foo = "bar"

[table1.nested_table]
        baz = "bat"
```

nested_table is defined as a value in table1 because its name starts with *table1*.. Again, the table goes until the next table definition, so baz="bat" is a value in *table1.nested_table*. You can indent the nested table to make it more obvious, but again, all whitespace is optional:

```
[table1]
        foo = "bar"

        [table1.nested_table]
                baz = "bat"
```

This is equivalent to the JSON:

```
{
        "table1" : {
                "foo" : "bar",
                "nested_table" : {
                        "baz" : "bat"
                }
        }
}
```

Having to retype the parent table name for each sub-table is kind of annoying, but I do like that it is very explicit. It also means that ordering and indenting and delimiters don't matter. You don't have to declare parent tables if they're empty, so you can do something like this:

```
[foo.bar.baz]
bat = "hi"
```

Which is the equivalent to this JSON:

```
{
        "foo" : {
                "bar" : {
                        "baz" : {
                                "bat" : "hi"
                        }
                }
        }
}
```

Array of elements in a table

The last thing is arrays of tables, which are declared with double brackets thusly:

```
[[comments]]
author = "Nate"
text = "Great Article!"

[[comments]]
author = "Anonymous"
text = "Love it!"

This is equivalent to the JSON:

{
        "comments" : [
                {
                        "author" : "Nate",
                        "text" : Great Article!"
                },
                {
                        "author" : "Anonymous",
                        "text" : Love It!"
                }
        ]
}
```

Arrays of tables inside another table get combined in the way you'd expect, like [[table1.array]].

TOML is very permissive here. Because all tables have very explicitly defined parentage, the order they're defined in doesn't matter. You can have tables (and entries in an array of tables) in whatever order you want. This is totally acceptable:

```
[[comments]]
author = "Anonymous"
text = "Love it!"

[foo.bar.baz]
bat = "hi"

[foo.bar]
howdy = "neighbour"

[[comments]]
author = "Anonymous"
text = "Love it!"
```

Of course, it generally makes sense to actually order things in a more organized fashion, but it's nice that you can't shoot yourself in the foot if you reorder things "incorrectly".

Using Cliqon configuration records

In the previous sections we have described what Cliqon configuration files are, what they are used for and that they are plain text and written in TOML format.

The content of a generated array is usually consumed by a Cliqon facility. We divide Cliqon facilities into three groups:

- data input – the process of entering information into a Cliqon system using forms and data import routines
- data manipulation – a standard Cliqon system when used to manage the content for a presentational website, does not do a great deal of data manipulation, but when Cliqon is used as a

eCommerce system, then the processing of stock, orders and invoicing would constitute data-manipulation
- data-output – the processes of retrieving information from the system in the form of displays, reports and downloads

Most of these activities are made available by the processing of a PHP script in a Class and its Methods. Most standard Cliqon derived classes are stored in two subdirectories - /framework/app and /framework/core. As an example, most of the functions related to the Administration system exist as methods within the Admin Class, which is in /framework/app. However for completeness of explanation, it should be noted that obvious shared functions such as responding to a Posted form, is undertaken by a Core function, in this case Method::postForm() within Class Db, which is located in /framework/core.

The Core Class entitled Model and the method contained therein, called stdModel(), is the primary vehicle for consuming configuration files and this Method will be found throughout most Classes and Methods of the system. The stdModel method, takes three arguments – the name of the service, the name of the collection and finally, the name of the model. The Method attempts to read a Cached version of the configuration first, then the three sources of configuration information from the database and if database record is absent, then a configuration file from the file system. If no Cached version existed, one is created at this time.

Finally, we state that stdModel takes 3 arguments. This is true but within the Method, the array snippet entitled “common” is read from every file / record.

Cascading

If you understand how the three elements of the Model can be made to cascade, one over the other, then please ignore this explanation. However the process of cascade is so fundamental to the operation of a web site that it bears repetition.

Cliqon provides Class → Config and inside it a Method entitled cfgReadString() which takes a string of characters in TOML format derived by some mechanism – read from a file on the file system or read from a database record.

When we callMethod stdModel(), we include three arguments – Service, Collection, Model. The Model may be an empty string. So arguments might look like $args = datagrid, dbcollection, string or $args = datagrid, dbuser, ‘’.

Method stdModel() will return an array which will contain a multi-dimensional array consisting of the following sections:

```
common => [
        key1 => value3,
        key2 => value2,
],
service => [
        key1 => value1,
        key2 => value3,
]
```

The Services record provided:

```
service => [
        key1 => value1
        ]
```

The Collection record provided:

```
        common => [
                key1 => value1
        ],
        service => [
                key2 => value3,
        ]
```

The Model record provided:

```
        common => [
                key1 => value3,
                key2 => value2,
        ]
```

Inside Method stdModel(), Cliqon uses function() array_replace_recursive. This is a standard PHP function which W3Schools desxcribes as:

If a key from array1 exists in array2, values from array1 will be replaced by the values from array2. If the key only exists in array1, it will be left as it is. If a key exist in array2 and not in array1, it will be created in array1. If multiple arrays are used, values from later arrays will overwrite the previous ones.

Thanks to the Javascript library PHPJS, array_replace_recursive also exists as function in the Cliqon javascript library and is used extensively in the definition of popup windows and alerts.

Services

Consider the following example:

```
$config = $clq→resolve('Config');
$dgcfg = $config→stdModel('datagrid', 'dbcollection', 'string');
```

Datagrid is a Service. You would expect to find a Record in the list of Services which contained the following data:

```
; Datagrid.Cfg - Standard Config file for a Gijgo Data Grid

primaryKey = 'id'
autoLoad: 'true'
uiLibrary = 'bootstrap4'
iconsLibrary = 'fontawesome'
headerFilter = 'true'
fontSize = '14px'
notFoundText = '144:No records available'

[pager]
limit = 15
rightControls = false
sizes = [10,15,20]

[rowicons:editrecord]
        icon = 'pencil'
        formid = 'columnform'

[rowicons:viewrecord]
        icon = 'eye'
        formid = 'columnform'

[rowicons:deleterecord]
        icon = 'trash'
```

Class *Admin* contains a Method called *datagrid()*. The method will generate an array of data, starting with the standard options needed by a Gijgo datagrid. The generated array will be converted and sent to the Javascript routine in response to a REST demand from the Client loaded Javascript for HTML, Javascript options and data, all in JSON format.

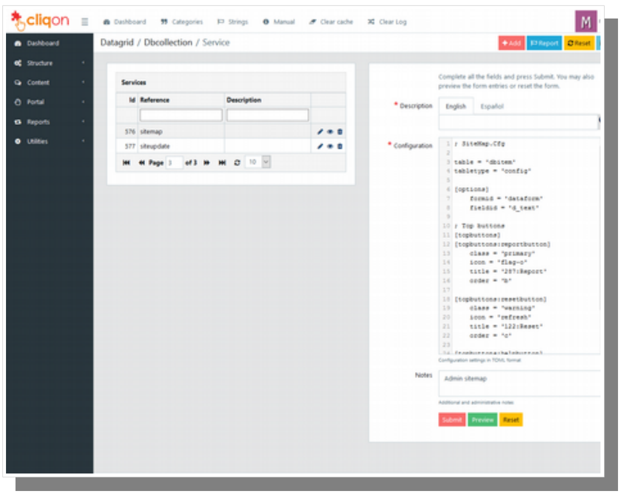
The Gijgo datagrid has default options for primaryKey and uiLibrary which need to be updated / overwritten from data sent by the Server.

As will be observed in the next section, the values for "primaryKey" and "uiLibrary" are not overwritten by entries in Collections or Models, thus these entries will be utilized. The same is also true for the rowicons.. However it is not unusual to see the entries for "pager" overwritten by the Model.

In the Administration system (henceforth referred to as "Admin") you will observe in the left hand menu system, an itm heading entitled Structure and within that submenu, an entry entitled Services. Clicking on the entry will display a list of the services currently recorded in the system. Currently there are about 20 services available for a Production version of Cliqon. As explained previously, these master service configuration files cover every area including Admin display facilities such as the Datagrid or the Datatree, administrative menus and other administrative facilities such as the Sitemap or Siteupdate.

The datagrid for Services includes all the facilities to create a new configuration record or edit an existing one. As was explained previously the Admin system can accept Service records as plain text files with a .cfg extension which, when used, are located in /admin/config subdirectory.

When you are augmenting a Cliqon system, it is entirely possible to work with a configuration file and when you have completely tested the result, copy the contents of the file to a Service record.



Datagrid / Dbcollection / Service

Services

Id	Reference	Description
576	sitemap	
577	siteupdate	

Page 1 of 3 · 10

Complete all the fields and press Submit. You may also preview the form entries or reset the form.

* Description: English | Español

* Configuration:

```
# SiteMap.Cfg

table = "dbitem"
tabletype = "config"

[options]
    formid = "dataform"
    fieldid = "d_text"

# Top buttons
[topbuttons]
[topbuttons:reportbutton]
    class = "primary"
    icon = "flag-o"
    title = "287:Report"
    order = "b"

[topbuttons:resetbutton]
    class = "warning"
    icon = "refresh"
    title = "122:Reset"
    order = "c"

[topbuttons:helpbutton]
```

Configuration settings in TOML format

Notes: Admin sitemap

Additional and administrative notes

When you create or edit a Service record, you will notice that the value of 'Reference' and the file name should be the same. Thus the database record with reference "Sitemap" would be found on the file system in /admin/config as sitemap.cfg.

When you open a configuration file or a configuration record, you will notice that the content is in plain text TOML format. We provide an administrative facility to maintain and edit files in the file system. The form for Service records (and others of a similar format) provides an instance of Codemirror to assist with the editing of the record. As a matter of note, the form preview facility converts the content of the Configuration field into an array before displaying the result. This is a useful tip to ensure that the generated array is as you would wish it to be.

As another matter of note, the User Access Control Level system can be amended to hide individual entries or an entire submenu from some administration operators. Therefore any fears that providing unauthorised or untrained operators access to these structurally sensitive files, should be allayed.

Recovery

As Services are fundamental to the operation of the system and the ability to access the database versions of the configuration records is central to being able to maintain any of the content of a system, you are reminded of the process of recovery.

Irrespective of the direction of record maintenance – you create database records from edited files or vice versa – the solution to any requirement for recovery is to use a raw database management facility such as a copy of Adminer, that we provide with the Cliqon system. All administrative records such as Service will be found in the Table "dbcollection" and in the case of "Service", have a type (c_type) value of "service". You can either amend the value portion of the record (c_common) or delete it, flush the cache and re-enter the data in Cliqon. Alternatively you might just delete the erroneous record and use the file system until the problem is resolved.

Collections

Collections describe the parent level in the record structure of tables and table types. The configuration entries in a Collection Configuration relate to Service records in one respect only. That is, you would expect to find entries in the Collection record for services such as Datagrid or Datatree that overwrite the values in the Service record.

```
; Config File for a Gijgo Grid
[datagrid]
        dataurl = ''

        [datagrid:columns:0]
                field = 'id'
                title = '9999:Id'
                headerCssClass = 'strong'
                width = '50'
                filterable = false
                align = 'right'
                order = 'a'
                sortable = false
                tooltip = '9999:Id'

        ; Top buttons
        [datagrid:topbuttons:addbutton]
                class = 'danger'
                icon = 'plus'
                title = '100:Add'
                formid = 'columnform'
                formtype = 'columnform'
                order = 'a'

        …….

        [datagrid:topbuttons:helpbutton]
                class = 'info'
                icon = 'info'
                title = '85:Help'
                order = 'x'
```

Thus *dbcollection* has service entries for virtually all the Services used for display and entry of data.

Common

All Collection and Model records will have a Common section. This, as the array key suggests, provides common information for this table via the stdModel() method to all activities in the system.
Most of the key/value pairs in a [common] section are intuitive. But it is worth explaining a few.

You will note how level is expressed as a string represented by 3 sets of double digits separated by colons. These are the default values for Read, Write and Delete used by the Access Control Level system (ACL) for this table. They can be overwritten at model and record level.

The value attached to “fieldsused” is a string of fieldnames that are used by this table. At Collection level, you would expect that all fields in the table schema are represented.

Values provide the default sort order and which field is used for version control.

Fields

All Collection and Model records will have a Fields section. This, as the array key suggests, provides information about the real or schema fields for this table via the stdModel() method to all activities in the system. Most of the key/value pairs in a [fields] section are intuitive. But it is worth explaining a few.

The most important information stored in the Fields subarray for an individual field, is the information about how the Field will be handled when record data for the field is written to the database. The majority of the fields are treated as strings and the final decision about format and size is left to the Redbean PHP ORM. In dbcollection, the two different fields are Revision (c_revision) and Document (c_document) which are 'integer' and 'json' respectively. So that this statement makes sense, we must consider the process of writing a record to the database.

As far as possible we have tried to ensure that Cliqon has a common interface for writing records to the database after a form is submitted. All forms are submitted using AJAX. All forms are submitted using the REST API and can accommodate a cross domain / remote json (jsonp) arrangement. Forms can be submitted utilising JSON Web Tokens (JWT). Some forms are submitted via the API Controller and some via the AJAX Controller. All common forms are responded to by $db→postForm($args).

To simplify and summarize the procedure in postForm(), the XHR Post containing FormData is validated and then split into real schema fields (starting with c_) and document or virtual fields (starting with d_). If a valid record ID has been included in the FormData (value greater than 0 / zero) then an update action is assumed (it should be noted that Redbean ORM does not make this distinction).

If a record insert action, the fields required for this Collection are retrieved via the Model. The value in FormData is used but if the key does not exist in the FormData array, the default value (defval) is used. The value of the 'required' key in the Collection > Field > Configuration should reflect the value in the Database Schema (if the Schema has been created manually or via the installation procedure).

If a record update action, the fields required for this Collection are retrieved via the Model. In most cases values in the real fields that are contained in the FormData Post will overwrite the existing record value. However this is handled differently when the Post contains virtual values (fields commencing d_). The postForm() Method must retrieve the JSON string contained in the Document field (c_document) and update it by a process of conversion from JSON to array, update it and then convert the array back to JSON.

As postForm() processes each field key / value pair it sends it for formatting to the required format. This is a two step process. Firstly, the fields with a specific function are processed. Currently there are four fields that require special processing – Who Modified (c_whomodified), Last Modified (c_lastmodified) and Revision (c_revision) or Version (c_version).

Who Modified – the current administrative operator is introduced
Last Modified – the current date and time is introduced
Version – The numeric value of the version number of this record is updated and a copy of the record before update is copied to the archive collection
Revision – The numeric value of the version number of this record is updated.

All the other fields of the Post are then sent to the dbEntry() Method to have the value formatted according to the value in 'dbtype' for that field. Current possible values include:

- password – the string value is converted to a Password Hash
- boolean – converts 'false' to 0 and 'true' to 1

- number – formats a number according to the rules defined in the main configuration file
- date – formats a date into year – month – day values
- datetime - formats a date into year – month – day and then hours and minutes values
- slugify – converts a string into a slugified value. That is it converts spaces to hyphens, converts accented characters to their ascii values and generally tidies up the string so that it can be used as a 'slug' or reference.
- json – the function will receive a JSON string that has been suitable 'escaped' to POST. It reverts the string to raw JSON and then tests the valid nature of the JSON before allowing it to be written to the database.
- toml – the function will receive a plain text string that has been suitable 'escaped' to POST. Like the JSON test above, it will ensure it is valid TOML before allowing it to be written to the database
- string – this is the default action and no formatting is carried out

As a general rule, the Client AJAX routine is informed of the success of the write (or otherwise) and a success alert is displayed. The display will then be updated to reflect the new or changed data.

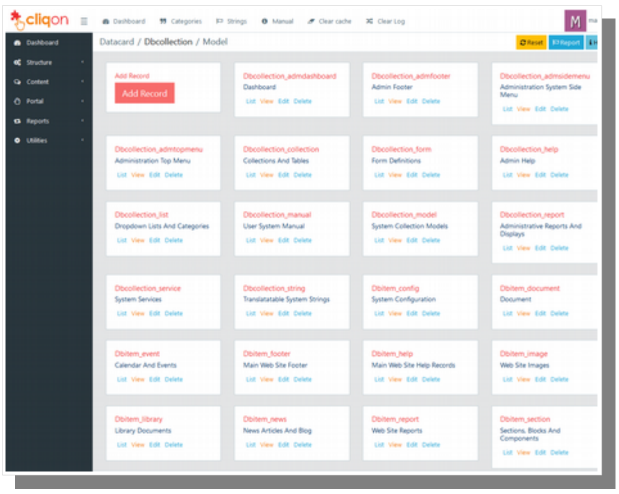
The datagrid for Collections includes all the facilities to create a new configuration record or edit an existing one. As was explained previously the Admin system can accept Collection records as plain text files with a .cfg extension which, when used, are located in /models subdirectory.

When you create or edit a Collection record, you will notice that the value of 'Reference' and the file name should be the same. Thus the database record with reference "dbcollection" would be found on the file system in /models as dbcollection.cfg.

Models

Models describe the second or child level in the record structure of tables and table types. The configuration entries in a Model Configuration relate to Service and Collection records in many ways. You will find sections entitled Common, Fields, the Service used by this Model (such as a datagrid or calendar), definitions for forms, views and reports.

All of this refers to the fact that the Class → Model takes three arguments for Model→stdModel(service, table, tabletype = ‘’) and that as the cascading takes place, Model overwrites Collection overwrites Service. As a matter of selection we have chosen Javascript utilities such as Gijgo (grid and tree), jqTree and Datatables where the options sent to the Javascript utility can ignore or throwaway, option keys and values that it does not need.



Datacard / Dbcollection / Model

Dbcollection_admdashboard
Dashboard

Dbcollection_admfooter
Admin Footer

Dbcollection_admsidemenu
Administration System Side Menu

Dbcollection_admtopmenu
Administration Top Menu

Dbcollection_collection
Collections And Tables

Dbcollection_form
Form Definitions

Dbcollection_help
Admin Help

Dbcollection_list
Dropdown Lists And Categories

Dbcollection_manual
User System Manual

Dbcollection_model
System Collection Models

Dbcollection_report
Administrative Reports And Displays

Dbcollection_service
System Services

Dbcollection_string
Translatatable System Strings

Dbitem_config
System Configuration

Dbitem_document
Document

Dbitem_event
Calendar And Events

Dbitem_footer
Main Web Site Footer

Dbitem_help
Main Web Site Help Records

Dbitem_image
Web Site Images

Dbitem_library
Library Documents

Dbitem_news
News Articles And Blog

Dbitem_report
Web Site Reports

Dbitem_section
Sections, Blocks And Components

Models equate to tabletypes for a given table. Thus they are displayed as Bootstrap cards as they tend to have longer references and descriptions.

All that has been explained in the pages for Services and Collections, apply to Models. Only the way that the database records are displayed on the screen is different in that it uses one of the two native Bootstrap 4 mechanisms for displaying lists (list or card) and in this case a Datacard.

Lists and Datacard show off the way that Cliqon utilises the Vue javascript framework. Cliqon does not use or push Vue to the limit of what can be achieved with Vue. Datacard and Datalist are examples of the hybrid template approach that is used throughout much of Cliqon administration and will probably be used for web site presentation.

Templates

For the administration system, Cliqon makes great use of templates. We anticipate that developers will wish to use the Cliqon templating system for their front end web sites as well.

Cliqon uses a slightly modified version of the Razr template engine developed by the Pagekit team, some years ago. This is in turn, a port of the Microsoft Razor parsing engine which can be found on Github under the Antaris badge.

Razr templates and the arsing engine are extremely straightforward and simple to understand and use. The parser using Regular Expressions to convert the contents of any file with included instructions into a HTML string. Templates are in effect PHP scripts but the regular PHP open and closer commands - *<php? Echo $action/$variable ?>* are replaced with a combination of the **@** symbol plus instructions and brackets **().**

Language

As a simple example, we might write:

```
<h1>@( $title )</h1>
@( 23 * 42 )
@( "<Data> is escaped by default." )
```

Or use the @raw() directive to output any PHP data without escaping.

```
@raw("This will <strong>not</strong> be escaped.")
```

You can access single variables and nested variables in arrays/objects using the following dot . notation.

```
array(
    'title' => 'I am the walrus',
    'artist' => array(
        'name' => 'The Beatles',
        'homepage' => 'http://www.thebeatles.com',
    )
)
```

Example

```
<h1>@( $title )</h1>
<p>by @( $artist.name ), @( $artist.homepage )</p>
```

Conditional control structures

Use @if, @elseif, @else for conditional control structures. Use any boolean PHP expression.

Example

```
@set($expression = false)
@if( $expression )
    One.
@elseif ( !$expression )
    Two.
@else
    Three.
@endif
```

Loops

You can use loop statements like foreach and while.

```
@foreach($values as $key => $value)
    <p>@( $key ) - @( $value )</p>
@endforeach

@foreach([1,2,3] as $number)
    <p>@( $number )</p>
@endforeach

@while(true)
    <p>Infinite loop.</p>
@endwhile
```

Include

Extract reusable pieces of markup to an external file using partials and the @include directive. You can pass an array of arguments as a second parameter.

```
<section>@include('partial.tpl', ['param' => 'parameter'])</section>
```

and in partial.tpl:

```
<p>Partial with @( $param )<p>
```

Output

```
<section><p>Partial with parameter<p><section>
```

Extending templates with blocks

Use the @block directive to define blocks inside a template. Other template files can extend those files and define their own content for the defined blocks without changing the rest of the markup.

Example

```
@include('child.tpl', ['param' => 'parameter'])
```

In parent.tpl:

```
<h1>Parent template</h1>

@block('contentblock')
    <p>Parent content.</p>
@endblock

<p>Parent content outside of the block.</p>
```

And in child.tpl:

```
@extend('parent.tpl')

@block('contentblock')
    <p>You can extend themes and overwrite content inside blocks. Paremeters are available as
well: @( $param ).</p>
@endblock
```

Note: To use an @ symbol in your text, such as *info@domain.com*, enter the symbol twice – *info@@domain.com*.

How Cliqon uses Razr templating

The sections above were taken from the Pagekit/Razr Readme. In practice Razr templating is significantly more powerful than this Readme suggests. The most important thing to understand is that fundamentally, Razr generates HTML with embedded PHP scripts. Razr provides only a few keywords for use attached to the **@** symbol but virtually any PHP activity that can be written or configured on one line can exist within the brackets. Also snippets of ordinary PHP can be included in the template and these will be just passed through the engine "as is" and without any conversion.

As a general rule, this version of Cliqon only executes the template engine from a Controller. Let us look at a summarised version of the PageController which can be found in /controllers.

```
class PageController
{
        private $cfg;
        private $idiom;
        private $page;
        private $rq = [];

        function get($idiom, $page)
        {

                global $clq;
                $this->cfg = $clq->get('cfg');
                $clq->set('idiom', $idiom);
                $this->page = $page;
                $cms = $clq->resolve('Cms');
                method_exists($cms, $page) ? $method = $page : $method = "page";

                // Load Template Engine
                $tpl = new Engine(new FilesystemLoader($clq->get('basedir')."views"),
                  $clq->get('basedir')."cache");
                $template = $this->page.'.tpl';
                $vars = [
                        'rootpath' => $clq->get('rootpath'),
                        'viewpath' => $clq->get('rootpath').'views/',
                        'includepath' => $clq->get('rootpath').'includes/',
                        'page' => $this->page,
                        'cmscontent' => $cms->$method($idiom, $_REQUEST),
                        'cfg' => $this->cfg,
                        'idiom' => $this->idiom,
                        'scripts' => $clq->get('js'),
                        'rq' => $_REQUEST
                ];

                echo $tpl->render($template, $vars);
        }
}
```

In the next section we will describe Cliqon Routing. At this point please accept that a URL has been generated along the lines of http://domain.com/page/en/news/?a=24. Routing loads the PageController and the two important arguments are the language and page name, plus the $_REQUEST.

Cliqon makes the following arbitrary decisions about the configuration of the Template system and most of these decisions are represented by values in the site Configuration. Also we have chosen to demonstrate the front-end page controller. Thus it is relatively straightfoward. In comparison, the AdmninController is more complicated as it contains more programming related to Access Control.

We recommend that all PHP scripting and programming related to the main website is contained in the Class entitled Cms. You will probably want to create a Method for each Page. However we must stress that Cliqon is designed to be so flexible that it can accommodate virtually any way of working. Our recommendations for the front-end are based on the way we write the Administration system and the hundreds of web sites that we have written for customers.

The Controller identifies that there is a Method in CMS that has the same name as the page.

We now create an instance of the Template engine, giving as arguments to the Constructor the ability to load a template file from filesystem and the subdirectory where the generated HTML will be cached. We have chosen that our instance of the Template Engine shall be known as $tpl. We now define the name of our page template, which we call the same name as the page.

Our template called $page.tpl contains variables. We need to provide values for these variables. We create an array called $vars. In the $vars array we write key/value pairs. The value of a pair can be anything – a string of characters, the result of a Method call or a sub array.

One of the variables is special. It contains the dynamic javascript that will be consumed by a special @raw($scripts) block within the Document.Ready block on the Template.

Thus, as a Developer, you could elect to have no PHP template variable substitution within the template itself and do everything at Javascript level using Vue templates as an example.

So in $vars, we have defined a few useful variables, for example, the paths to the views, images, includes and public subdirectories. We have populated a main Content block with dynamic HTML content retrieved from the CMS Page Method. In the Page Method, we also generated the appropriate Javascripting which the Page Method "sets" and the Controller "gets". You will understand that this is a Convenience mechanism. You can write your Controllers, Classes and Methods anyway you want.

Finally we can Echo the "Render". Where the Template Render method receives two arguments, the name of the template and the arguments for the template. You can also replace Render with Publish. We just prefer Echo Render because it reminds us which aspect of our programming is producing the result on the screen.

Configuration and arbitrary decisions

Cliqon has a very flexible and expandable administration system. We choose to configure the "views" directory (the place where templates are stored) as a sub-directory called admin. We cannot visualise why you would want to change this.

We choose to give our templates the extension .tpl. We choose to make the subdirectory from which the website templates are retrieved, a subdirectory called /views.

In the administration system, we keep child templates for the main "admindesktop" template in /admin/partials. We choose to keep component templates that are used by the administration Class and Methods in the /admin/components subdirectory.

If we wanted to create several different access routes to the data – for example, pages for desktop, pages for mobiles and a completely different system for user interaction with the data, all of these can be accommodated by a Cliqon system.

Front-end templates

A few words about configuring templates for a Cliqon front end. We provide two versions of the Cliqon Framework that you can download - a demonstration version and a production version. The demonstration version is a modified instance of the Cliqon website. The production version has a single page as a template. It is this version we shall refer to as an example.

Displayed below is a summarized version of index.tpl.

```
@include('partials/header.tpl')
@include('partials/cookieconsent.tpl')
</head>
<body class="landing-page">

    @include('partials/nav.tpl')

    <div class="wrapper">
        <div class="page-header page-header-small">
            <div class="page-header-image" data-parallax="true" style="background-image:
url('@($viewpath)img/bg6.jpg');">
            </div>
            <div class="container">
                <div class="content-center">
                    <h2 class="title"><img class="img-fluid col-sm-5"
src="@($viewpath)/img/logo.png" title="This is Cliqon" alt="@($viewpath)/img/logo.png"
style="margin-top: 40px;" /></h2>
                    <div class="text-center">
                        <a href="#pablo" class="btn btn-primary btn-icon  btn-icon-mini">
                            <i class="fa fa-facebook-square"></i>
                        </a>
                        …………….
                    </div>
                </div>
            </div>
        </div>
        <div class="section section-about-us">
            <div class="container">
                <div class="row">
                    <div class="col-md-8 offset-md-2 text-center">
                        <h2 class="title">@(Q::uStr('2:What is Cliqon?'))</h2>
                        <h5 class="description">@(Q::uStr('3:Cliqon is a web app ....'))</h5>
                    </div>
                </div>
                ……………………..
                 </div>
            </div>
        </div>

        @include('partials/footer.tpl')

    </div>

@include('partials/script.tpl')

<!-- End of Page -->
@include('partials/end.tpl')
```

We can draw your attention to a few important things in the code. The first is how we include “partials” to help us normalise the template codebase. As a second point, you can see how we include important variables such path information. As explained previously the Pagekit readme does not do justice to the power of the template engine. Not only can we include variables, but we can include static and instantiated methods to retrieve language content.

Administration

System Build

The section entitled System Build will cover three areas in detail. It will explain all the aspects of the Cliqon framework, starting with our definition of a MVC system. It will explain those aspects of Cliqon that we have written ourselves and the modules that we have imported and utilised from elsewhere and other authors, such as the Razr templating system and Redbean database handler.

The framework subsection will cover the following key server side functions – routing, authentication, REST api, database, ORM, html generation, template rendering, logging and debugging, configuration, file system, image manipulation, forms, menus, reports, lists, users, session and cookie handler, integration with services such mail, ftp and Node, text and array searching, JSON handler and finally plugins.

The framework subsection will also cover the following key client side facilities – style sheets, templates and javascript. The presentational framework for the Administration system is based on CoreUI. Rendering of every aspect of the Administration is accomplished using rendered templates, server generated HTML and AJAX.

The structure subsection will explain how we have used the framework functions to build an administration system which you, as a developer, can augment and enhance. It will explain how extend the Cliqon system by writing your own classes and methods as modules and plugins.

Finally the subsection on website presentation will explain how to create and maintain content in Cliqon so that Cliqon can act as a Content Management System (CMS) utilizing multi-lingual templates, pages, components, blocks and strings.

In principle, we can describe Cliqon as a Module – View – Controller system (MVC). There is no settled and agreed definition for the meaning of an MVC system.

Cookbook

The Cliqon documentation cookbook is equivalent to the flexible ring binder that many of us, who enjoy cooking, have in the kitchen to keep recipes that we have found or downloaded from the internet. The initial published Cliqon Cookbook will contain several sections. The first section will contain definition details of the expected page types that tend to populate the majority of presentational websites and beyond. The list includes single page websites (SPAs), home pages, rich content pages, listing pages for useful links and documents, galleries, calendars, forms and so on.

At a different level, we are very fond of CSS frameworks such as Bootstrap and Pure, or ideas that inspire frameworks such as Material Design. The Cookbook will provide detailed examples of how to integrate Cliqon multi-lingual strings and components within templates with these CSS Frameworks.

At another level, we are also fond of Javascript frameworks such as Vue and Angular, supporting Javascript general libraries such as jQuery or application specific libraries such as FullCalendar or Morris. The Cookbook will provide detailed examples of how to integrate the Javascript libraries into Cliqon components, templates and files.

We will show you a complete example of how to integrate Cliqon with an HTML template that we have downloaded from the internet.

As the months and years progress the list of recipes in the Cookbook will grow. It will become a miscellany of ideas on how to use Cliqon for different purposes.

Cliqon contains the following modules for immediate use. Additional pages and even modules can be created very quickly so that the system can grow and change to meet the needs of the web site owner.

Rich content pages

Cliqon contains the tools to manage a wide variety of different designs. Complete web pages can be stored in the database and edited with an on-Cliqon web page editor. Web pages can take the form of templates with a series of snippets of textual content displayed in appropriate places.

Useful web Cliqon

The system contains a module that provides multi-lingual web Cliqon to other web sites. This can be used as a simple internet directory or search engine.

Document library

This module manages a library of downloadable files and documents. These files could be PDF documents for use by the sales and technical departments or lists of other types of downloadable files and documents.

Digital News

A comprehensive facility is provided to manage digital news items. These items are stored in such a way that they comply with the requirements of a RSS2 news feed.

Events Diary

A modified version of the news module is provided to present a calendar of future events. The diary can be presented as a listing of events on a week by week basis and also as a graphical calendar.

Catalogue

Cliqon includes a simple Catalogue module with which to display items for sale and to act as a shop window for products and items such as properties and boats. The Catalogue supports a shopping cart which records the items for sale, which can then be mailed as an order to the web site owner or transferred to PayPal for payment by credit card.

Gallery

A module is included to display a gallery of images as a slideshow.

Forms and Reports

Comprehensive facilities exist to design and develop a wide range of forms and reports. Forms can be used to input and manage information on the system and the reports designed to present information to both web site owner and visitor in ways that are unique to the needs of the particular web site.

Site Search and Map

Facilities to search the Site and its contents are provided plus a module to generate a Site Map.

Cliqon Page Types

The majority of websites in general and Cliqon websites in particular, provide four different type of web page for this type of informational or content driven website where the content conforms to an overall common template design.

Rich Text Content Page

A page which can contain any form of content – including tables, images and listings etc. The content of the page is created and managed in an online facility which looks Cliqon a word processor. The depth of the page is dependent on the quantity of text and content. A facility to print the content of the page is provided.

Listings Page

The content of the page is a series of rows of data, where each row is managed individually by the administration system. We use a Listings page for News, Weblinks and Library. The depth of the page is controlled by defining the number of rows on a page. Rows are categorised and searchable. In many cases Cliqon on an item will display more content in a popup window or take the visitor to the desired location.

In the case of a Catalogue Page, a simple shopping cart can be implemented with facilities to send an order by email or take money by PayPal or a local banking credit card acceptance facility. The Catalogue can display products as a textual listing or as images.

Form page

The content of the form consists of a data entry form with appropriate rules and instructions. The purpose of the form is to enter data into the system or in Cliqon, to create and submit an email. Confirmation of submission is provided.

Special Design Page

Sometimes, the preferred design for the website demands that a special page is created. The Home Page or Start page is invariably a special page. Facilities to manage the content are provided but normally the website owner cannot adjust the overall design for themselves.

Widgets

On every page, usually in the form of a Sidebar, a series of widgets are provided which can be switched on and off for a page as required. Currently we have widgets for:

- search the whole site for content
- display an RSS newsfeed
- display a simple calendar
- display categories for listings page as a clickable Cliqon or tagcloud
- display a series of important weblinks
- help or information for the page

Index

Appendices

List here

Acknowledgements
Definitions

Acknowledgements

List of all the Third Party authors who have contributed software and code to Cliqon.

Definitions

Description	Acronym	Definition
Access Control List	ACL	The
Administration system	Admin	
TOML	TOML	Tom’s Obvious Minimal Language: The ability to display complex multi-dimensional arrays of information in plain text with human readable comments, keys and values.
JSON Web Tokens	JWT	
	XHR	
	FormData	
Redbean PHP ORM	ORM	

JSON Web Tokens

REST API's are meant to be stateless. What that means is that each request from a client should include all the information needed to process the request. In other words, if you are writing a REST API in PHP then you should not be using $_SESSION to store data about the client's session. But then how do we remember if a client is logged in or anything else about their state? The only possibility is that the client must be tasked with keeping track of the state. How could this ever be done securely? The client can't be trusted!

Enter JSON web tokens. A JSON web token is a bit of JSON, perhaps something that looks like this:

```
{
  "user": "alice",
  "email": "test@nospam.com"
}
```

Of course, we can't just give this to a client and have them give it back to us without some sort of assurance that it hasn't been tampered with. After all, what if they edit the token as follows:

```
{
  "user": "administrator",
  "email": "test@nospam.com"
}
```

The solution to this is that JSON web tokens are signed by the server. If the client tampers with the data then the token's signature will no longer match and an error can be raised.

The JWT PHP class makes this easy to do. For example, to create a token after the client successfully logs in, the following code could be used:

```
$token = array();
$token['id'] = $id;
echo F::encode($token, 'secret_server_key');
```

And then on later API calls the token can be retrieved and verified by this code:

```
$token = F::decode($_POST['token'], 'secret_server_key');
echo $token->id;
```

If the token has been tampered with then $token will be empty there will not be an id available. The JWT class makes sure that invalid data is never made available. If the token is tampered with, it will be unusable. Pretty simple stuff!

The contents of the JWT Class have been incorporated into Framework.Php and the Framework Class.