CFW Documentation

Contents

[1 File Structure 2](#_Toc21597402)

[2 Change DB Password 3](#_Toc21597403)

[3 Admin Account 3](#_Toc21597404)

[4 Logging 3](#_Toc21597405)

[4.1 Example Code 3](#_Toc21597406)

[4.2 Sample Configuration 3](#_Toc21597407)

[4.3 Known Issues 4](#_Toc21597408)

[5 Resources 5](#_Toc21597409)

[5.1 JAR Resources 5](#_Toc21597410)

[6 Application 5](#_Toc21597411)

[6.1 HTTP to HTTPs Redirect 5](#_Toc21597412)

[6.1.1 Known Issues 5](#_Toc21597413)

[7 CFW Framework 6](#_Toc21597414)

[7.1 Data Handling 6](#_Toc21597415)

[7.1.1 CFWObject Example 6](#_Toc21597416)

[7.1.2 CFWField Examples 8](#_Toc21597417)

[8 CFW.js 10](#_Toc21597418)

[8.1 CFWTable 10](#_Toc21597419)

[8.2 CFW.ui.addToast 10](#_Toc21597420)

[9 Maintenance 11](#_Toc21597421)

[9.1 Font Awesome 11](#_Toc21597422)

[9.2 JQuery UI 11](#_Toc21597423)

# File Structure

* **/config**
  + **logging.properties:** Configuration file for java.util.logging.
  + **cfw.properties:** Configuration file which has to pe passed to CFWSetup.*initialize*("./config/cfw.properties");
  + **credentials.csv:** file containing username/password credentials when using authentication\_method=CSV
  + **keystore.jks:** Default keystore for testing purposes(Password: 123456)
* **/datastore:** Default folder for the database files.
* **/log:** Folder containing the log files.
* **/resources**
  + **/css**
    - **Custom.css:** Your custom css file which will be loaded by TemplateHTMLDefault.
  + **/js**
  + **/html**

# Change DB Password

Connect to the DB and use the following SQL to change the password of the database user:

**ALTER** **USER** sa **SET** PASSWORD 'yourpassword';

# Admin Account

The default admin account is as follows:

* **Username:** admin
* **Password:** admin

If you have changed and forgot the admin password of this account you can do the following to reset the password back to the default:

* In the “./config/cfw.properties”, set the value of “cfw\_reset\_admin\_pw” to true and restart the application. A log statement of level INFO will be written confirming the reset of the password. Change the setting to false after the restart to prevent subsequent resets of the password.
* Connect to the database and delete the admin account from the database. Restart the application, the admin account will be created again. This might delete as well all other data from the database related to the account.

# Logging

By default the logging uses a asynchronous logger, which writes incoming logs all 50 milliseconds.

## Example Code

**public** **static** Logger *logger* = CFWLog.*getLogger*(YourClass.**class**.getName());

CFWLog log = **new** CFWLog(*logger*).method("initialize").start();

log.end();

Explanations:

* Initalize a logger with CFWLog.getLogger().
* Create a new CFWLog instance passing the logger
* Use CFWLog.start() and CFWLog.stop() method to create duration logs.

## Sample Configuration

handlers = com.pengtoolbox.cfw.logging.AsyncLogHandler, java.util.logging.FileHandler, java.util.logging.ConsoleHandler

config =

com.pengtoolbox.pageanalyzer.level = INFO

com.pengtoolbox.cfw.level = INFO

com.pengtoolbox.cfw.logging.AsyncLogHandler.level = FINE

com.pengtoolbox.cfw.logging.AsyncLogHandler.limit = 10000000

com.pengtoolbox.cfw.logging.AsyncLogHandler.append = false

com.pengtoolbox.cfw.logging.AsyncLogHandler.count = 1

com.pengtoolbox.cfw.logging.AsyncLogHandler.pattern = ./log/applog\_%u\_%g.log

#java.util.logging.FileHandler.level = FINE

#java.util.logging.FileHandler.filter =

#java.util.logging.FileHandler.formatter = com.pengtoolbox.cfw.logging.CFWLogFormatterJSON

#java.util.logging.FileHandler.encoding =

#java.util.logging.FileHandler.limit = 10000000

#java.util.logging.FileHandler.count =

#java.util.logging.FileHandler.append = false

#java.util.logging.FileHandler.count = 2

#java.util.logging.FileHandler.pattern = ./log/trace\_%u.%g.log

java.util.logging.ConsoleHandler.level = FINE

java.util.logging.ConsoleHandler.filter =

java.util.logging.ConsoleHandler.formatter = com.pengtoolbox.cfw.logging.CFWLogFormatterJSON

java.util.logging.ConsoleHandler.encoding =

## Known Issues

The Logging will not be formatted correctly, if one of the following is the case:

* The config file ./config/logging.properties is not defined
* The package names have changed
* The class name of CFWLogFormatterJSON was modified
* Any java.util.logging class was used before the logging.properties was loaded.

# Resources

## JAR Resources

You can fetch resources from JAR files using the servlet “jarresource”.

Parameters

* **pkg:** The package where the resource is stored.
* **File:** The file to be fetched

Before being able to load from a certain package, you need to add the package to the allowed packages for security reasons:

CFW.Files.*addAllowedPackage*("com.pengtoolbox.cfw.resources");

Example Request:

*./jarresource?pkg=com.pengtoolbox.cfw.resources.fonts&file=fa-solid-900.eot*

# Application

## HTTP to HTTPs Redirect

Add a HTTPSRedirectHandler as the first handler in your handler collection.

handlerCollection.setHandlers(**new** Handler[] {**new** HTTPSRedirectHandler(), apiContext, rewriteHandler, pageanalyzerContext, CFWSetup.*createResourceHandler*(), CFWSetup.*createCFWHandler*(), **new** DefaultHandler() });

### Known Issues

* When both HTTP & HTTPS is enabled, and redirect\_http\_to\_https is disabled, and user switches from HTTPS to HTTP, he cannot login anymore without restarting the browser.

# CFW Framework

## Data Handling

The data handling simplifies the flow of data from browser forms, to objects to databases. The basic programming flow is like this:

* Create a class extending from CFWObject.class e.g MyClass
* Add CFWFields to MyClass and set the table name using
  + this.setTableName(TABLE\_NAME);
  + this.setPrimaryField();
  + this.addFields();
* Use myClassInstance.createTable() to create a database in the table

To create a Form do:

* Use myClassInstance.toForm() to create a form instance. All the values from the instance are automatically entered into the resulting form.
* Add a handler using formInstance.setFormHandler(new BTFormHandler() {…});

To map the post data sent with the form, use the following inside BTFormHandler:

* form.mapRequestParameters(request);
* or origin.mapRequestParameters(request);
* Note: form and origin reference the same instances of CFWField, therefore it always updates fields in form and origin. If you are using only some of all the fields of the origin object, it is recommended mapping the parameters to the form for security reasons. (Hackers could tamper with the field names and change other fields)

To persist the data, use the following inside BTFormHandler:

* origin.insert();
* origin.update();

### CFWObject Example

An example of how to create a CFWObject for simple group management.

**public** **class** Group **extends** CFWObject {

**public** **static** **final** String ***TABLE\_NAME*** = "CFW\_GROUP";

**public** **enum** GroupFields{

***PK\_ID***,

***NAME***,

***DESCRIPTION***

}

**private** **static** Logger *logger* = CFWLog.*getLogger*(Group.**class**.getName());

**private** CFWField<Integer> id =

CFWField.*newInteger*(FormFieldType.***HIDDEN***,

GroupFields.***PK\_ID***.toString())

.setPrimaryKeyAutoIncrement()

.setValue(-999);

**private** CFWField<String> name =

CFWField.*newString*(FormFieldType.***TEXT***,

GroupFields.***NAME***.toString())

.setColumnDefinition("VARCHAR(255) UNIQUE")

.addValidator(**new** LengthValidator(1, 255));

**private** CFWField<String> description =

CFWField.*newString*(FormFieldType.***TEXTAREA***,

GroupFields.***DESCRIPTION***.toString())

.setColumnDefinition("CLOB")

.addValidator(**new** LengthValidator(-1, 2000000));

**public** Group() {

initializeFields();

}

**public** Group(String name) {

initializeFields();

**this**.name.setValue(name);

}

**private** **void** initializeFields() {

**this**.setTableName(***TABLE\_NAME***);

**this**.setPrimaryField(id);

**this**.addFields(id, name, description);

}

**public** **int** id() {

**return** id.getValue();

}

**public** Group id(**int** id) {

**this**.id.setValue(id);

**return** **this**;

}

**public** String name() {

**return** name.getValue();

}

**public** Group name(String name) {

**this**.name.setValue(name);

**return** **this**;

}

**public** String description() {

**return** description.getValue();

}

**public** Group description(String description) {

**this**.description.setValue(description);

**return** **this**;

}

}

### CFWField Examples

Create new CFWFields using CFWField.*new\*()* methods.  
You can set the default value for the field using setValue(). If this is not specified it always defaults to 0.

#### Boolean Field with Default true

Creates a simple Boolean field. A BooleanValidator is already added by CFWField.newBoolean().  
Sets the default to true.

**private** CFWField<Boolean> isDeletable =

CFWField.*newBoolean*(FormFieldType.***BOOLEAN***,

GroupFields.***IS\_DELETABLE***.toString())

.setValue(**true**);

#### Unique Text Field with Validator

Creates a text field with a unique constraint on the database column and a length validator.  
Default the default column definition set by newString() is "VARCHAR".

**private** CFWField<String> name =

CFWField.*newString*(FormFieldType.***TEXT***,   
GroupFields.***NAME***.toString())

.setColumnDefinition("VARCHAR(255) UNIQUE")

.addValidator(**new** LengthValidator(1, 255));

#### TextArea as CLOB with 2MB Limit

Creates a textarea and that will be saved to a CLOB field. Limits the data amount to 2MB with the length validator.

**private** CFWField<String> description =

CFWField.*newString*(FormFieldType.***TEXTAREA***,

GroupFields.***DESCRIPTION***.toString())

.setColumnDefinition("CLOB")

.addValidator(**new** LengthValidator(-1, 2000000));

#### Number with custom Label

Creates a number field with a custom label.  
By default the name of the CFWField will be used as label like this:

* Change underscores to blank
* Make all words start with an uppercase letter

**private** CFWField<Integer> number =

CFWField.*newInteger*(FormFieldType.***NUMBER***,

"Number\_Fieldxyz")

.setLabel("Enter a Number");

#### Select Field with Description

Creates a selection box with the two options "Active" and "Inactive".   
Adds a description that will be displayed as a tooltip on an info icon in the form in front of the label.

**private** CFWField<String> status =

CFWField.*newString*(FormFieldType.***SELECT***, UserDBFields.***STATUS***.toString())

.setColumnDefinition("VARCHAR(31)")

.setOptions(**new** String[] {"Active", "Inactive"})

.setDescription("Active users can login, inactive "+

"users are prohibited to login.")

.addValidator(**new** LengthValidator(-1, 31));

#### Select Field with Value/Label Pairs

Creates a selection box with values and labels. Values will be stored and labels will be for showing the options in the selection box.  
Below sets the default value to Banana.

**private** CFWField<Integer> keyValSelect =

CFWField.*newInteger*(FormFieldType.***SELECT***, "KEY\_VAL\_SELECT")

.setValue(2);  
  
**public** **void** initialize() {

LinkedHashMap<Integer, String> options = **new** LinkedHashMap<Integer, String>();

options.put(1, "Apple");

options.put(2, "Banana");

options.put(3, "Plumb");

options.put(4, "Strawwberry");

keyValSelect.setValueLabelOptions(options);

**this**.addFields(...,keyValSelect, ...);

}

#### Date field With Default

Creates a new date picker field with a default epoch time value for "26.01.2020".  
Times will always be converted to epoch time for request parameters and saving to database.  
Dates before 1582 are not supported well because of the cutoff between the Julian and Gregorian calendars.( ISO-8601 standard supported)

**private** CFWField<Date> date =

CFWField.*newDate*(FormFieldType.***DATEPICKER***, "DATE")

.setValue(**new** Date(1580053600000L));

#### Timestamp field

Creates a new date and time picker field.  
The default will be the current date and time of the users browser when no value is specified.  
Times will always be converted to epoch time for request parameters and saving to database.  
Dates before 1582 are not supported well because of the cutoff between the Julian and Gregorian calendars.( ISO-8601 standard supported)

**private** CFWField<Timestamp> timestamp =

CFWField.*newTimestamp*(FormFieldType.***DATETIMEPICKER***, "TIMESTAMP");

# CFW.js

This section describes how to use various methods of cfw.js.

## CFWTable

Create a bootstrap table.

**var** cfwTable = CFW.ui.createTable();

cfwTable.isNarrow = **true**;

cfwTable.isHover = **true**;

cfwTable.isResponsive = **true**;

cfwTable.isStriped = **true**;  
cfwTable.tableFilter = **true**;

cfwTable.addHeader('&nbsp;');  
cfwTable.addHeaders(['&nbsp;','Timings','Status','Duration','URL']);

cfwTable.addRows('<tr> <td>A</td> <td>B</td> </tr>');  
cfwTable.addRows($('<tr>'));

cfwTable.appendTo(parent);

// add all keys from the first object in the array as headers

**for**(**var** key **in** data[0]){

cfwTable.addHeader(key);

}

## CFW.ui.addToast

Add a Toast message to the application.

//Regular Toast with primary style

CFW.ui.addToast("My Title", "Some Message");

//Small toast without body, styled as danger

CFW.ui.addToast("An error occured!!!", **null**, "danger");

//Success message and autohide in 3 seconds

CFW.ui.addToast("Saved Successfully!!!", **null**, "success", 3000);

# Maintenance

## Font Awesome

When upgrading font awesome:

* Update the CSS and font files in the package ”com.pengtoolbox.cfw.resources”
* Update font-awesome.css, @font-face.src definitions have to load files using the jarresource servlet.

@font-face {

font-family: *'Font Awesome 5 Free'*;

font-style: *normal*;

font-weight: *900*;

font-display: *auto*;

src: *url("./jarresource?pkg=com.pengtoolbox.cfw.resources.fonts&file=fa-solid-900.eot")*;

src: *url("./jarresource?pkg=com.pengtoolbox.cfw.resources.fonts&file=fa-solid-900.eot?#iefix")* *format("embedded-opentype"),* *url("./jarresource?pkg=com.pengtoolbox.cfw.resources.fonts&file=fa-solid-900.woff2")* *format("woff2"),* *url("./jarresource?pkg=com.pengtoolbox.cfw.resources.fonts&file=fa-solid-900.woff")* *format("woff"),* *url("./jarresource?pkg=com.pengtoolbox.cfw.resources.fonts&file=fa-solid-900.ttf")* *format("truetype"),* *url("./jarresource?pkg=com.pengtoolbox.cfw.resources.fonts&file=fa-solid-900.svg#fontawesome")* *format("svg")*;

}

## JQuery UI

When upgrading JQueryUI:

* Update the CSS and image files in the package ”com.pengtoolbox.cfw.resources”
* Update jquery-ui.min.css
  + Replace: url("images/
  + With: url("/cfw/jarresource?pkg=com.pengtoolbox.cfw.resources.images&file=

## Summernote

When upgrading Summernote:

* Update the CSS and image files in the package ”com.pengtoolbox.cfw.resources”
* Update jquery-ui.min.css
  + Replace: ./font/
  + With: ./jarresource?pkg=com.pengtoolbox.cfw.resources.fonts&file=
  + Also replace all the cryptic stuff after the font filenames