# Composr Mobile SDK

The Composr Mobile SDK is a collection of standalone utilities used for building an app. Programmers can use Composr Mobile SDK to create apps to integrate functionalities available in a website based on Composr with much easier effort. All the core parts of Composr API are made available as iOS and Android libraries.

The PHP APIs are also included in Composr Mobile SDK and are called as CMS Utilities.

# CMS Utility Classes

The CMS Utilities classes are:

* CMS\_Arrays
* CMS\_Strings
* CMS\_Langs
* CMS\_Preferenece
* CMS\_Timestamps
* CMS\_HTTP
* CMS\_Users
* CMS\_Flow
* CMS\_Forms
* CMS\_Database
* CMS\_Notification\*
* CMSNetworkManager
* More Utilities

# CMS\_Arrays

The CMS\_Arrays class provides the array operation functionalities. The class contains following utilities:

## collapse\_1d\_complexity

*Prototype:*

(NSArray \*)collapse\_1d\_complexity:(NSString \*)key :(NSArray \*)arr;

*Description:*

Takes a list of maps and turns it into a simple list by extracting just one element from each map.

*Usage:*

[CMS\_Arrays collapse\_1d\_complexity :<key to be extracted> :<array>];

*Example:*

Let us consider that you have an array of maps. Each maps contains an id and a title as keys. Using the collapse\_1d\_complexity utility, you can get all the title key values from the array. The code will be:

NSArray \*inputArray = @[@{@"id" : @"1",@"title" : @"India"},

 @{@"id" : @"2",@"title" : @"Pakistan"}];

NSString \*inputKey = @“title";

NSArray \*output = [CMS\_Arrays collapse\_1d\_complexity:inputKey :inputArray];

NSString \*inputKey = @“title";

NSArray \*output = [CMS\_Arrays collapse\_1d\_complexity:inputKey :inputArray];

The output from above code will be: @[@“India",@"Pakistan"]

## collapse\_2d\_complexity

*Prototype:*

(NSDictionary \*)collapse\_2d\_complexity:(NSString \*)keyKey :(NSString \*)valKey :(NSArray \*)arr;

*Description:*

Takes a list of maps and simplify by extracting two elements from each map. First element is kept as the key and second as value in the resultant map

*Usage:*

[CMS\_Arrays collapse\_2d\_complexity:<key of key> :<key of val> :<array>];

*Example:*

Consider an array of maps with each map contains an id and a title. You need to convert the array into a map by fetching two values from each map by keeping the first as the key and second as the value.

NSArray \*inputArray = @[@{@"id" : @"1",@"title" : @"India"},

 @{@"id" : @"2",@"title" : @"Pakistan"}];

NSString \*inputKeyKey = @"id";

NSString \*inputKeyVal = @“title”;

By calling,

[CMS\_Arrays collapse\_2d\_complexity:inputKeyKey :inputKeyVal:inputArray];

would return a dictionary of the following form:

@{

 @"1" : @"India",

 @"2" : @"Pakistan"

};

The values from the id field is taken as key and the value of the title field is taken as value.

##

## explode

*Prototype:*

(NSArray \*)explode:(NSString \*)sep :(NSString \*)str;

*Description:*

Similar to PHP explode functionality, will split multiple strings from input string. The utility will return an array of strings, where each split is a substring of string formed by splitting it on boundaries formed by the string separator.

*Usage:*

[CMS\_Arrays explode:<separator string> :<source string>];

*Example:*

The example shown below is an instance with split strings separated by hyphens.

NSString \*source = @“This-is-an-example-string”;

NSString \*separator = @“-”;

Calling, [CMS\_Arrays explode:separator :source]; will return an array of strings:

@[@“This”,@"is",@"an",@"example",@"string"];

## implode

*Prototype:*

(NSString \*)implode:(NSString \*)sep :(NSArray \*)arr;

*Description:*

Implode will join multiple array elements to form a single string. The functionality is as same as in PHP implode.

*Usage:*

[CMS\_Arrays implode:<joiner string> :<source array>];

*Example:*

Code example on having an array of strings and joining strings by using hyphen through implode.

NSArray \*source = @[@“This”,@"is",@"an",@"example",@"string"];

NSArray \*joiner = @“-”;

Calling, [CMS\_Arrays implode:joiner :source]; will return a single string:

@“This-is-an-example-string"

## map\_to\_list

*Prototype:*

(NSArray \*)map\_to\_list:(BOOL)useKey :(NSDictionary \*)dict;

*Description:*

map\_to\_list will return either all keys or all values from a map based on the boolean value that is set.

*Usage:*

[CMS\_Arrays map\_to\_list:<want keys ?> :<source map>];

*Example:*

Below example code will return only keys from the map if the value is set to YES, otherwise will return only values from a map that contains many keys and values.

NSDictionary \*source = @{

 @"key1":@"val1",

 @"key2":@"val2"

 };

Calling, [CMS\_Arrays map\_to\_list:YES :dict]; will return:

@[@“key1”,@“key2”]

Calling, [CMS\_Arrays map\_to\_list:NO :dict]; will return:

@[@“val1”,@“val2”]

# CMS\_Strings

CMS\_Strings utility provides the common string operations functionalities.

## strip\_tags

*Prototype:*

(NSString \*)strip\_tags:(NSString \*)str;

*Description:*

Removes all the HTML tags from the input string

*Usage:*

[CMS\_Strings strip\_tags:<input string>];

*Example:*

Consider the scenario where a string returned from a web service contains few HTML tags. These unwanted HTML tags can be removed by using strip\_tags.

NSString \*inputString = @“Hi. </br>How are you?”;

Calling, [CMS\_Strings strip\_tags:inputString]; will return:

@“Hi. How are you?”

## html\_entity\_decode

*Prototype:*

(NSString \*)html\_entity\_decode:(NSString \*)str;

*Description:*

Convert the common HTML entity into standard keyboard character. For example, the entity "&amp;" will be converted to "&".

*Usage:*

[CMS\_Strings html\_entity\_decode:<input string>]

*Example:*

Consider that HTML entity is present for double quotes in your string returned from a query. These entities can be converted to keyboard character by calling html\_entity\_decode method.

NSString \*inputString = @"&ldquo;I love cats &amp; dogs&ldquo;";

Calling, [CMS\_Strings html\_entity\_decode:inputString]; will return:

@"“I love cats & dogs“”.

## float\_format

*Prototype:*

(NSString \*)float\_format:(double)number :(int)decimalPoints :(BOOL)onlyIncludeNeededDecimalPoints;

*Description:*

Formats a number value nicely by including commas and decimal points wherever applicable. The float format can be performed by setting the boolean value to YES.

*Usage:*

[CMS\_Strings float\_format:<input double value> :<number of required decimal points> :<should consider the second param ?>];

*Example:*

double input = 12345678.012110;

[CMS\_Strings float\_format:input :0 :NO] will return @“12,345,678.01211"

[CMS\_Strings float\_format:input :0 :YES] will return @“12,345,678"

[CMS\_Strings float\_format:input :2 :YES] will return @“12,345,678.01"

## strpos

*Prototype:*

(int)strpos:(NSString \*)searchIn :(NSString \*)searchFor;

*Description:*

strops works similar to the functionality available in PHP strpos. The utility is used to find the position of the first occurrence of a substring in a string. If not found, then will return -1.

*Usage:*

[CMS\_Strings strpos:<input string> :<search string>]

*Example:*

For example, if you need to find the starting position of “s” in the string "I love cats and dogs” then strpos can be used.

Calling [CMS\_Strings strpos:"I love cats and dogs” :@“s"] will return 10.

## str\_replace

*Prototype:*

(NSString \*)str\_replace:(NSString \*)search :(NSString \*)replace :(NSString \*)searchIn;

*Description:*

Similar to the common search and replace string functionality in PHP. The str\_replace utility will replace all the instances of the search string with the provided replacement string.

*Usage:*

[CMS\_Strings str\_replace:<search\_string> :<replace\_string> :<input\_string>]

## substr

*Prototype:*

(NSString \*)substr:(NSString \*)searchIn :(int)offset :(int)length;

*Description:*

The utility returns the portion of string specified by the start and length parameters.

*Usage:*

[CMS\_Strings substr:<input\_string> :<start\_position> :<length to be copied>]

## trim

*Prototype:*

(NSString \*)trim:(NSString \*)str;

*Description:*

To strip the whitespace and newline characters from the beginning and end of a string, the trim utility can be used.

*Usage:*

[CMS\_Strings trim:<input\_string>]

## stringWithFormat

*Prototype:*

(NSString \*)stringWithFormat:(NSString \*)format array:(NSArray \*)arguments;

*Description:*

stringWithFormat utility is to replace the %@ strings from the string array values in the source string.

*Usage:*

[CMS\_Strings stringWithFormat:<input\_strings> array:<replace arguments>];

*Example:*

Consider that there is a generic string template from a web API saying @“Hi %@. Welcome to %@”. Use the stringWithFormat to substitute first instance of %@ the user’s name and the second instance with forum name.

Calling, [CMS\_Strings stringWithFormat:@“Hi %@. Welcome to %@“ array:@[@“Alen”,@“CMS”]];will return:

@“Hi Alen. Welcome to CMS”

# CMS\_Langs

CMS\_Langs class provides all the necessary localisation functions. The common localisation functions are available in do\_lang utility which can be used with or without providing any parameters.

## do\_lang

*Prototype:*

(NSString \*)do\_lang:(NSString \*)key

*Description:*

The do\_lang method returns the localised value of the supplied input key string from localizable.strings based on the device language.

*Usage:*

[CMS\_Langs do\_lang:inputString]

## do\_lang: parameters:

*Prototype:*

(NSString \*)do\_lang:(NSString \*)strName :(NSArray \*)parameters;

*Description:*

Returns the localised value of the string after replacing the string with arguments.

*Usage:*

[CMS\_Langs do\_lang:<input\_string> :<arguments>]

*Example:*

[CMS\_Langs do\_lang:@"%@\_%@" :@[@“EXAMPLE",@"STRING"]]; will find the localised version of string found by replacing @“%@\_%@” with the arguments given. So, it will return the localised version of @“EXAMPLE\_STRING”.

# CMS\_Preferences

The CMS\_Preferences class utility provides the common methods used to define the user default preferences. This class contains get\_value and set\_value as utilities.

## get\_value

*Prototype:*

(NSString \*)get\_value:(NSString \*)string;

*Description:*

Get the default value from NSUserDefaults.

*Usage:*

[CMS\_Preferences get\_value:<key>]

## set\_value

*Prototype:*

(void)set\_value:(NSString \*)name :(NSString \*)value;

*Description:*

Assign a default value for key.

*Usage:*

[CMS\_Preferences set\_value:<key> :<value>]

# CMS\_Timestamps

Timestamp related functionalities such as get\_timezoned\_date and time are available as utilities under CMS\_Timestamps class. This class utility is called in the methods that you may need on timestamp values.

## get\_timezoned\_date

*Prototype:*

(NSString \*)get\_timezoned\_date:(int)timestamp :(BOOL)includeTime;

*Description:*

Converts the given timestamp to @"MMM dd, yyyy - HH:mm:ss” date format and returns the value as a string. If includeTime parameter is set to false, then the time is excluded in the output.

*Usage:*

[CMS\_Timestamps get\_timezoned\_date:<timestamp> :<includeTime ?>]

## time

*Prototype:*

(int)time;

*Description:*

Returns the UNIX timestamp.

*Usage:*

[CMS\_Timestamps time]

# CMS\_HTTP

The CMS\_HTTP class provides methods needed while connecting and parsing web services. This class contains the following utilities:

## rawurlencode

*Prototype:*

(NSString \*)rawurlencode:(NSString \*)str;

*Description:*

Returns a string in which all non-alphanumeric characters except -\_.~ will be replaced with a percentage (%) symbol, followed by two hexadecimal digits. This is the encoding described in [» RFC 3986](http://www.faqs.org/rfcs/rfc3986) for protecting the literal characters from being interpreted as special URL delimiters and for protecting URLs from being mangled by transmission media with character conversions. For example, in some email systems.

*Usage:*

[CMS\_HTTP rawurlencode:<input\_string>]

*Example:*

NSString \*inputString = @"test encoding the url + something”;

Calling, [CMS\_HTTP rawurlencode:inputString]will return:

@"test%20encoding%20the%20url%20%2B%20something"

## json\_decode

*Prototype:*

(id) json\_decode:(NSString \*)value;

*Description:*

Takes a JSON encoded string and converts it into a corresponding Objective-C variable.

*Usage:*

[CMS\_HTTP json\_decode:<input\_json\_string>]

*Example:*

NSString \*jsonString = @“{\"test\":\"val\",\"test1\":\"val1\"}";

Calling, [CMS\_HTTP json\_decode:jsonString]will return a dictionary of the form:

@{

@"test" : @"val",

@"test1" : @"val1"

};

## json\_encode

*Prototype:*

(NSString \*)json\_encode:(id)value;

*Description:*

Returns a string containing the JSON representation of an Objective-C variable.

*Usage:*

[CMS\_HTTP json\_encode:<input\_objc\_variable>]

*Example:*

NSDictionary \*inputDict = @{

 @"test" : @"val",

 @"test1" : @"val1"

 };

Calling, [CMS\_HTTP json\_encode:inputDict] will return the string of the form:

@"{\"test\":\"val\",\"test1\":\"val1\"}"

## get\_base\_url

*Prototype:*

(NSString \*)get\_base\_url;

*Description:*

Gets the base URL to Composr API Endpoint, defined for the project in CMS\_Constants.h.

*Usage:*

[CMS\_HTTP get\_base\_url]

## build\_url

*Prototype:*

(NSString \*)build\_url:(NSDictionary \*)params :(NSString \*)zone;

*Description:*

Returns a URL with the given parameters and zone name using the format

<base\_url>/<zone>/index.php?<params...>

*Usage:*

[CMS\_HTTP build\_url:<params> :<zone\_name>]

*Example*:

Calling, [CMS\_HTTP build\_url:@{@"param1":@"value1",@"param2":@"value2"} :@“testZone"]will return:

@“http://<base\_url>/testZone/index.php?&param1=value1&param2=value2”

## http\_download\_file

*Prototype:*

(void)http\_download\_file:(NSString \*)url :(BOOL)triggerError :(NSDictionary \*)postParams :(int)timeoutInSeconds :(CMSCompletedDownloadBlock)completionHandler;

*Description:*

Downloads the response from the web service and returns as a response in a completion handler. If the triggerError is true, then a failure will be displayed as a generic error alert. If postParams is not nil then it will be an HTTP post.

*Usage:*

[CMS\_HTTP http\_download\_file:<web service url> :<trigger\_error ?> :<post parameters> :<custom\_timeout\_value> :<completion\_handler>];

*Example:*

Calling, [CMS\_HTTP http\_download\_file:<url> :YES :nil :kHTTPTimeout :^(NSString \*response) {

 NSLog(@"%@",response);

 }]; will print the response of the resulting web service call.

## has\_network\_connection

*Prototype:*

(BOOL) has\_network\_connection;

*Description:*

Check and return a boolean value based on the internet connectivity available for the app. The utility will check whether the app is connected to the internet or not.

*Usage:*

[CMS\_HTTP has\_network\_connection]

# CMS\_Users

The CMS\_Users utility provides user related functionalities such as retrieve member, usergroup, and privilege information from Composr to the app and that are synced during login.

When a successful login is executed using CMSNetworkManager, each value from the resultant JSON response is stored in the UserDefaults. Also, the values inside user\_data key is extracted and saved into UserDefaults. The CMS\_Users class helps you with utilities that you will require to fetch these values.

This class contains the following utilities:

## has\_page\_access

*Prototype:*

(BOOL)has\_page\_access;

*Description:*

Returns a boolean value if the user has page access in Composr.

*Usage:*

[CMS\_Users has\_page\_access]

## has\_privilege

*Prototype:*

(BOOL)has\_privilege:(NSString \*)privilegeName;

*Description:*

Returns a boolean value if the user has any privileges on the account in Composr.

*Usage:*

[CMS\_Users has\_privilege:<priviege\_name>]

## has\_zone\_access

*Prototype:*

(BOOL)has\_zone\_access:(NSString \*)zoneName;

*Description:*

Returns a boolean value after checking whether the user has the access to a specific zone in Composr.

*Usage:*

[CMS\_Users has\_zone\_access:<zone\_name>]

## is\_staff

*Prototype:*

(BOOL)is\_staff;

*Description:*

Returns a boolean value if the user is staff or not in Composr.

*Usage:*

[CMS\_Users is\_staff]

## is\_super\_admin

*Prototype:*

(BOOL)is\_super\_admin;

*Description:*

Returns a boolean value if the user is a super admin or not in Composr. But will return the value as NO if the user is a staff but super admin was true.

*Usage:*

[CMS\_Users is\_super\_admin]

## get\_member

*Prototype:*

(int)get\_member;

*Description:*

Returns the member id of the user in Composr.

*Usage:*

[CMS\_Users get\_member]

## get\_session\_id

*Prototype:*

(int)get\_session\_id;

*Description:*

Returns the session id of the user in Composr.

*Usage:*

[CMS\_Users get\_session\_id]

## get\_username

*Prototype:*

(NSString \*)get\_username;

*Description:*

Returns the username of logged in user.

*Usage:*

[CMS\_Users get\_username]

## get\_members\_groups

*Prototype:*

(NSArray \*)get\_members\_groups;

*Description:*

Returns all the details of member groups a user has in Composr.

*Usage:*

[CMS\_Users get\_members\_groups]

## get\_members\_groups\_names

*Prototype:*

(NSArray \*)get\_members\_groups\_names;

*Description:*

Returns only names of the member groups a user had in Composr.

*Usage:*

[CMS\_Users get\_members\_groups\_names]

## get\_value

*Prototype:*

(NSString \*)get\_value:(NSString \*)key;

*Description:*

Returns value for a key from user defaults.

*Usage:*

[CMS\_Users get\_value:<key>]

## get\_password

*Prototype:*

(NSString \*)get\_password;

*Description:*

Returns the password of the user saved in user defaults

*Usage:*

[CMS\_Users get\_password]

# CMS\_Flow

The CMS\_Flow utility provides methods that you can use to control the flow of your app. This class contains the following utilities:

## access\_denied

*Prototype:*

(void)access\_denied;

*Description:*

Dismisses any screen that is shown now and presents the login page as root controller.

*Usage:*

[CMS\_Flow access\_denied];

## attach\_message

*Prototype:*

(void)attach\_message:(NSString \*)msg;

*Description:*

Pops up an alert with the given message. The alert title will be “Message”.

*Usage:*

[CMS\_Flow attach\_message:<some\_text>];

## inform\_screen

*Prototype:*

(void)inform\_screen:(NSString \*)msg :(UIViewController \*)viewController;

*Description:*

Pops up an alert with the given message. The alert title will be “Message”. Automatically dismisses the calling view controller.

*Usage:*

[CMS\_Flow inform\_screen:<some\_text> :<presenting\_controller\_instance>];

## warn\_screen

*Prototype:*

(void)warn\_screen:(NSString \*)msg :(UIViewController \*)viewController;

*Description:*

Pops up an alert with the given message. The alert title will be “Warning”. Automatically dismisses the calling view controller.

*Usage:*

[CMS\_Flow warn\_screen:<some\_text> :<presenting\_controller\_instance>];

## redirect\_screen

*Prototype:*

(void)redirect\_screen:(UIViewController \*)sourceViewController :(UIViewController \*)viewController;

*Description:*

Open a new view controller modally from the sourceViewController.

*Usage:*

[CMS\_Flow redirect\_screen:<source\_controller\_instance> :<destination\_controller\_instance>];

# CMS\_Forms

The forms related functionalities are provided in the CMS\_Forms class utility. This works like as a form builder. CMS\_Forms class itself is a subclass of UIView class and can be added as a subview into any view. All the methods in this class are instance methods. So, you will need an instance of the class to access the features.

For including the form in full screen on a controller, use:

CMS\_Forms \*form = [[CMS\_Forms alloc] initWithFrame:self.view.frame];

[self.view addSubview:form];

The class has a delegate variable which can be used to get responses of form submissions and such communications.

All the form field adding methods has following common parameters:

* PrettyName: Text that is displayed as the form field placeholder.
* Description:String that is shown as an instruction text of the field above it.
* ParamName: This will be parameter name of the value for the particular form field when sending an HTTP request.
* DefaultValue: The default value that will be populated in the form field
* isRequired: A boolean value used to set whether a form field is mandatory or optional. When submitting a form, the unfilled form fields will be shown in red and the delegate will be notified through the preSubmitCallback.

The class contains the following utilities:

## form\_input\_field\_spacer\_withHeading

*Prototype:*

(void)form\_input\_field\_spacer\_withHeading:(NSString \*)heading withText:(NSString \*)text;

*Description:*

Adding a heading into the form.

*Usage:*

[form\_instance form\_input\_field\_spacer\_withHeading:<heading text>withText:<sub\_heading text>];

## form\_input\_hidden\_withParamName

*Prototype:*

(void)form\_input\_hidden\_withParamName:(NSString \*)paramName withParamValue:(NSString \*)paramValue;

*Description:*

Insert a hidden form parameter which is only considered when submitting the form. This hidden form will not appear anywhere in the UI.

*Usage:*

[form\_instance form\_input\_hidden\_withParamName:<param\_name> withParamValue:<param\_value>];

## form\_input\_integer\_withPrettyName

*Prototype:*

id)form\_input\_integer\_withPrettyName:(NSString \*)prettyName withDescription:(NSString \*)description withParamName:(NSString \*)paramName withDefaultValue:(NSString \*)defaultValue isRequired:(BOOL)isRequired;

*Description:*

Inserts an integer only input control. The utility will not accept floating points, alphabets and special characters.

*Usage:*

[form\_instance form\_input\_integer\_withPrettyName:<pretty\_name>

 withDescription:<description>

 withParamName:<param\_name>

 withDefaultValue:<default\_value>

 isRequired:<is\_required ?>];

## form\_input\_float\_withPrettyName

*Prototype:*

(id)form\_input\_float\_withPrettyName:(NSString \*)prettyName withDescription:(NSString \*)description withParamName:(NSString \*)paramName withDefaultValue:(NSString \*)defaultValue isRequired:(BOOL)isRequired;

*Description:*

Inserts a floating point input control and will not accept alphabets and special characters.

*Usage:*

[form\_instance form\_input\_float\_withPrettyName:<pretty\_name>

 withDescription:<description>

 withParamName:<param\_name>

 withDefaultValue:<default\_value>

 isRequired:<is\_required ?>];

## form\_input\_line\_withPrettyName

*Prototype:*

(id)form\_input\_line\_withPrettyName:(NSString \*)prettyName withDescription:(NSString \*)description withParamName:(NSString \*)paramName withDefaultValue:(NSString \*)defaultValue isRequired:(BOOL)isRequired;

*Description:*

Inserts a string input control similar to an HTML input box. There are no validations and will accept any type of characters.

*Usage:*

[form\_instance form\_input\_line\_withPrettyName:<pretty\_name>

 withDescription:<description>

 withParamName:<param\_name>

 withDefaultValue:<default\_value>

 isRequired:<is\_required ?>];

## form\_input\_list\_withPrettyName

*Prototype:*

(id)form\_input\_list\_withPrettyName:(NSString \*)prettyName withDescription:(NSString \*)description withParamName:(NSString \*)paramName withOptions:(NSDictionary \*)options withDefaultValue:(int)valueIndex isRequired:(BOOL)isRequired;

*Description:*

Inserts a combo-box type control. Options is a map between actual values and the displayed labels. The functionality is as same as the “<option>” in a web “<select>” control. The default value is the index of option to be selected by default.

*Usage:*

[form\_instance form\_input\_list\_withPrettyName:<pretty\_name>

 withDescription:<description>

 withParamName:<param\_name>

 withOptions:<options\_map>

 withDefaultValue:<default\_value>

 isRequired:<is\_required ?>];

*Example:*

[form\_instance form\_input\_list\_withPrettyName:@"gender"

 withDescription:@"Gender"

 withParamName:@"gender"

 withOptions:@{

 @"male":@"Male",

 @"female":@"Female"

 }

 withDefaultValue:1

 isRequired:YES];

## form\_input\_text\_withPrettyName

*Prototype:*

(id)form\_input\_text\_withPrettyName:(NSString \*)prettyName withDescription:(NSString \*)description withParamName:(NSString \*)paramName withDefaultValue:(NSString \*)defaultValue isRequired:(BOOL)isRequired;

*Description:*

Inserts a multiline text input control.

*Usage:*

[form\_instance form\_input\_text\_withPrettyName:<pretty\_name>

 withDescription:<description>

 withParamName:<param\_name>

 withDefaultValue:<default\_value>

 isRequired:<is\_required ?>];

## form\_input\_tick\_withPrettyName

*Prototype:*

(id)form\_input\_tick\_withPrettyName:(NSString \*)prettyName withDescription:(NSString \*)description withParamName:(NSString \*)paramName withDefaultValue:(BOOL)defaultValue;

*Description:*

Inserts a switch control. This is equivalent to a two options only radio box. Form value will be either 0 or 1.

*Usage:*

[form\_instance form\_input\_tick\_withPrettyName:<pretty\_name>

 withDescription:<description>

 withParamName:<param\_name>

 withDefaultValue:<default\_value>];

*Example:*

[form\_instance form\_input\_tick\_withPrettyName:@"subscribe for newsletter ?"

 withDescription:@""

 withParamName:@"subscribe"

 withDefaultValue:NO];

## form\_input\_uploaded\_picture\_withPrettyName

*Prototype:*

(id)form\_input\_uploaded\_picture\_withPrettyName:(NSString \*)prettyName withDescription:(NSString \*)description withParamName:(NSString \*)paramName isRequired:(BOOL)isRequired;

*Description:*

Inserts a photo upload control. Picks a photo from user photo library or shows an option to capture photo using camera.

*Usage:*

[form\_instance form\_input\_uploaded\_picture\_withPrettyName:<pretty\_name>

 withDescription:<description>

 withParamName:<param\_name>

 isRequired:<is\_required ?>];

*Example:*

[form\_instance form\_input\_uploaded\_picture\_withPrettyName:@"profilePic"

 withDescription:@"upload your profile pic"

 withParamName:@"profilePic"

 isRequired:YES];

## form\_input\_date\_withPrettyName

*Prototype:*

(id)form\_input\_date\_withPrettyName:(NSString \*)prettyName withDescription:(NSString \*)description withParamName:(NSString \*)paramName isRequired:(BOOL)isRequired withDefaultValue:(NSString \*)defaultValue includeTimeChoice:(BOOL)includeTimeChoice;

*Description:*

Inserts a date picker control. includeTimeChoice determines whether the date picker should show time or not.

*Usage:*

[form\_instance form\_input\_date\_withPrettyName:<pretty\_name>

 withDescription:<description>

 withParamName:<param\_name>

 isRequired:<is\_required ?>

 withDefaultValue:<default\_value>

 includeTimeChoice:<show time ?>];

*Example:*

[form\_instance form\_input\_date\_withPrettyName:@"DOB"

 withDescription:@"Provide your DOB :"

 withParamName:@"dob"

 isRequired:YES

 withDefaultValue:@""

 includeTimeChoice:NO];

## get\_input\_date

*Prototype:*

(int)get\_input\_date:(NSString \*)paramName;

*Description:*

Get UNIX timestamp from a date input field in the form with the given paramName.

*Usage:*

[form\_instance get\_input\_date:<paramName>];

*Example:*

[form\_instance get\_input\_date:@“DOB”]; returns timestamp of the DOB date input field. The previous example mentioned for form\_input\_date\_withPrettyName is relevant.

## post\_param

*Prototype:*

(NSString \*)post\_param:(NSString \*)paramName;

*Description:*

Gets the string value from an input field in a form with the given paramName.

*Usage:*

[form\_instance post\_param:<paramName>];

## post\_param\_integer

*Prototype:*

(int)post\_param\_integer:(NSString \*)paramName;

*Description:*

Gets int value from an input field in a form with the given paramName.

*Usage:*

[form\_instance post\_param\_integer:<paramName>];

## set\_url

*Prototype:*

(void)set\_url:(NSString \*)url;

*Description:*

Sets the URL to which the form is submitted, the function is similar to action on HTML form.

*Usage:*

[form\_instance set\_url:<url>];

## set\_button\_withName

*Prototype:*

(void)set\_button\_withName:(NSString \*)name preSubmitGuard:(SEL)preSubmitGuard postCallback:(SEL)postCallback autoSubmit:(BOOL)autoSubmit;

*Description:*

Sets the form’s submit button. preSubmitGuard is the selector that will be called in case of validation failure while sending the form. postCallback is the selector that is called after the form submit is finished. All selectors will be executed in the delegate.

* By default, the submit button will be executing the validation. If the validation fails, then the preSubmitGuard is called and if succeeds, the postCallback will be called. This feature can be used if the form is not needed to be submitted on an HTTP URL. This feature is available in the CMS\_SDK\_Sample app.
* If autoSubmit is true, then the submit button will automatically trigger the HTTP form submit if the validation is successful.

*Usage:*

[form\_instance set\_button\_withName:@"Done"

 preSubmitGuard:<validation\_error\_callback\_selector>

 postCallback:<submit\_success\_callback\_selector>

 autoSubmit:<autoSubmit ?>];

*Example:*

[form\_instance set\_button\_withName:@"Done"

 preSubmitGuard:@selector(validationError)

 postCallback:@selector(callback)

 autoSubmit:YES];

## do\_http\_post\_request

*Prototype:*

(void)do\_http\_post\_request;

*Description:*

Execute the form HTTP submit. Will call the preSubmitGuard if validation fails and the submit process will be aborted. If the validation succeeds, then will call postCallback after HTTP submit is completed. Screen will be blocked automatically with a loader when the HTTP submit happens.

*Usage:*

[form\_instance do\_http\_post\_request];

## getFormValues

*Prototype:*

(NSDictionary \*)getFormValues;

*Description:*

Returns a map of all the values of the form with paramName as key and entered value as value.

*Usage:*

[form\_instance getFormValues];

# CMS\_Database

This utility provides the methods that you will need to access a SQLite database. The methods functionalities are similar to the PHP database utility methods. So, you can access the database the same way you access your SQL database on the Composr. You can set up the custom database upgrading code (this is not possible out of the box with SQLite).

The utility methods are:

## createCopyOfDatabaseIfNeeded

*Prototype:*

(void)createCopyOfDatabaseIfNeeded;

*Description*:

Looks for “cms\_DB.sqlite” named file in the project bundle. If a similar file does not exist in the documents directory then this file is copied into it. If the project is run with the flag “database\_reset”, the file in documents directory is forcefully rewritten. This can be used to create a prototype of the database and pack with your project.

*Usage*:

[CMS\_Database createCopyOfDatabaseIfNeeded];

## initializeDatabase

*Prototype:*

(void)initializeDatabase;

*Description:*

This method opens the connection to the database. Should be called before executing any of the below utility methods to ensure that the connection to the database is open.

*Usage:*

[CMS\_Database initializeDatabase];

## add\_table\_field

*Prototype:*

(void)add\_table\_field:(NSString \*)tableName :(NSString \*)fieldName;

*Description:*

Adds a new column to the table.

*Usage:*

[CMS\_Database add\_table\_field:<table\_name> :<field\_name>];

## rename\_table\_field

*Prototype:*

(void)rename\_table\_field:(NSString \*)tableName :(NSString \*)oldFieldName :(NSString \*)newFieldName;

*Description:*

Renames a table column name

*Usage:*

[CMS\_Database rename\_table\_field:<table\_name> :<old\_field\_name> :<new\_field\_name>];

## delete\_table\_field

*Prototype:*

(void)delete\_table\_field:(NSString \*)tableName :(NSString \*)fieldName;

*Description:*

Delete a column from the table.

*Usage:*

[CMS\_Database delete\_table\_field:<table\_name> :<delete\_field\_name>];

## create\_table

*Prototype:*

(void)create\_table:(NSString \*)tableName :(NSArray \*)fieldNames;

*Description:*

Create a table

*Usage:*

[CMS\_Database create\_table:<table\_name> :<field\_names\_array>];

*Example:*

NSString \*tableName = @"testTable";

NSArray \*fields = @[

@"field1",

@"field2"

];

[CMS\_Database create\_table:tableName :fields];

## drop\_table\_if\_exists

*Prototype:*

(void)drop\_table\_if\_exists:(NSString \*)tableName;

*Description:*

Delete a table

*Usage:*

[CMS\_Database drop\_table\_if\_exists:<table\_name>];

## db\_escape\_string

*Prototype:*

(NSString \*)db\_escape\_string:(NSString \*)value;

*Description:*

Escapes any single quote strings in the SQL.

*Usage:*

[CMS\_Database db\_escape\_string:<string\_to\_be\_escaped>]

## query

*Prototype:*

(NSArray \*)query:(NSString \*)query;

*Description:*

Executes the query and returns an array of resultant rows. Each row is represented as a map. Values in the row will be saved against column name as key.

*Usage:*

[CMS\_Database query:<query>];

## query\_delete

*Prototype:*

(void)query\_delete:(NSString \*)tableName :(NSDictionary \*)whereMap;

*Description:*

Delete a specific set of rows based on equal conditions provided in whereMap. i.e. each key value pair in whereMap will be converted in to key=value in the sql. No other conditions are supported.

*Usage:*

[CMS\_Database query\_delete:<table\_name> :<where\_map>];

*Example:*

NSString \*tableName = @“testTable";

NSDictionary \*deleteWhereMap = @{

@"field2":@"value22"

};

Executing

[CMS\_Database query\_delete:tableName :deleteWhereMap];

is equivalent to running - “DELETE FROM testTable WHERE field2=‘value22’; ”

## query\_insert

*Prototype:*

(void)query\_insert:(NSString \*)tableName :(NSDictionary \*)valueMap;

*Description:*

Insert values into a table. Value map will contain the values to be inserted saved in a dictionary against the column names as keys.

*Usage:*

[CMS\_Database query\_insert:<table\_name> :<insert\_map>];

*Example:*

NSString \*tableName = @“testTable";

NSDictionary \*insertMap = @{

@"field1":@"value11",

@"field2":@"value12"

};

[CMS\_Database query\_insert:tableName :insertMap];

is equivalent to running - INSERT INTO testTable (field1, field2) VALUES (‘value11’, ‘value12’);

## query\_select

*Prototype:*

(NSArray \*)query\_select:(NSString \*)tableName :(NSArray \*)selectList :(NSDictionary \*)whereMap :(NSString \*)extraSQL;

*Description:*

 Running a select query. “selectList” is the array of fields that needs to be selected. “whereMap” contains the conditions to be matched. Any extra SQL can be specified in “extraSQL”.

*Usage:*

[CMS\_Database query\_select:<table\_name> :<selectFields> :<whereMap> :nil];

*Example:*

NSString \*tableName = @“testTable";

[CMS\_Database query\_select:tableName :@[@"field1"] :@{@"field2":@"value22"} :nil];

is equivalent to running - SELECT field1 from testTable where field2=‘value22’ ;

## query\_select\_value

*Prototype:*

(NSString \*)query\_select\_value:(NSString \*)tableName :(NSString \*)selectFieldName :(NSDictionary \*)whereMap :(NSString \*)extraSQL;

*Description:*

 Selecting a single string value from the database. If more than one rows are found in the result of the query, the value in the first row is returned. If no value, blank string will be returned.

*Usage:*

[CMS\_Database query\_select\_value:<table\_name> :<selectField> :<whereMap> :nil];

## query\_select\_int\_value

*Prototype:*

(int)query\_select\_int\_value:(NSString \*)tableName :(NSString \*)selectFieldName :(NSDictionary \*)whereMap :(NSString \*)extraSQL;

*Description:*

 Selecting a single integer value from the database. If more than one rows are found in the result of the query then the value in the first row is returned. If no value, -1 is returned. If the value was not an integer convertible, 0 is returned.

*Usage:*

[CMS\_Database query\_select\_int\_value:<table\_name> :<selectField> :<whereMap> :nil];

## query\_update

*Prototype:*

(void)query\_update:(NSString \*)tableName :(NSDictionary \*)valueMap :(NSDictionary \*)whereMap;

*Description:*

 Update rows in a table based on a particular condition. “valueMap” is Values to be updated in a map saved against field name as keys.

*Usage:*

[CMS\_Database query\_update:<table\_name> :<value\_map> :<where\_map>];

*Example:*

NSString \*tableName = @"testTable";

[CMS\_Database query\_update:tableName :@{@"field2":@"40"} :@{@“field1":@"value31"}];

is equivalent to executing - UPDATE testTable SET field2=’40' WHERE field1=‘value31’;

## getFieldNamesForTable

*Prototype:*

+ (NSArray \*) getFieldNamesForTable:(NSString \*)tableName;

*Description:*

Get the list of column names of a table

*Usage:*

[CMS\_Database getFieldNamesForTable:<table\_name>];

## upgradeDatabaseIfRequired

*Prototype:*

(void)upgradeDatabaseIfRequired:(CMSDatabaseUpgradeBlock)upgradeBlock;

*Description:*

Upgrade database based on the codes written in the code handler provided. The code handler has database object as parameter. So, you can directly execute SQLite queries on it. The latest database version is saved in the user defaults and step by step upgrades will be executed.

*Usage:*

[CMS\_Database upgradeDatabaseIfRequired:<upgrade\_handler>];

*Example:*

[CMS\_Database upgradeDatabaseIfRequired:^(sqlite3 \*dbInstance){

NSUserDefaults \*defaults = [NSUserDefaults standardUserDefaults];

NSString \*previousVersion = [defaults objectForKey:k\_DBVERSION];

NSString \*currentVersion = [self versionNumberString];

if (previousVersion==nil){

// do nothing. first install.

}

else if([previousVersion compare:currentVersion options:NSNumericSearch] == NSOrderedAscending) {

// previous < current

// place upgrade codes here for each versions in their corresponding if structures.

// You can use the "dbInstance" variable to access the db.

int prevVersion = [previousVersion intValue];

if (prevVersion < 1) {

// place all codes for upgradation to version 1

prevVersion = 1;

}

if (prevVersion < 2){

// place all codes for upgradation to version 2

prevVersion = 2;

}

}

[defaults setObject:currentVersion forKey:k\_DBVERSION];

[defaults synchronize];

}];

# CMS\_Notification

This utility provides all the methods required to handle push notifications in an iPhone app. The utility methods are:

## registerForRemoteNotifications

*Prototype:*

(void)registerForRemoteNotifications;

*Description:*

Register the app for permission to push notifications

*Usage:*

[CMS\_Notification registerForRemoteNotifications];

## parseDeviceToken

*Prototype:*

(NSString \*)parseDeviceToken:(NSData \*)deviceToken;

*Description:*

Parse device token that has received from the APNS server

*Usage:*

[CMS\_Notification parseDeviceToken:<NSData received from the server did register for remote notifications>

## notifyDeviceTokenToServer

*Prototype:*

(void) notifyDeviceTokenToServer:(NSString \*)deviceToken;

*Description:*

Send the device token to Composr Server.

*Usage:*

[CMS\_Notification notifyDeviceTokenToServer:<parsed device token as string>];

## showNotification

*Prototype:*

(void) showNotification:(NSDictionary \*)userInfo;

*Description:*

Parse a push notification received from the Composr server and show the alert.

*Usage:*

[CMS\_Notification showNotification:<payload received as notification>];

# CMSNetworkManager

CMSNetworkManager is a singleton network manager that is built on AFNetworking 2.0 . A JSON parser is mount as response parser by default. So, if you need to parse any other kind of responses like XML, plain/text or so, you will need to modify the response serialiser.

The following APIs are already built in to the CMSNetworkManager:

1. Login
	1. The response received is saved in the UserDefaults
	2. Cookie and session variables are automatically set in the header all the API calls after a successful login
2. Register
3. Recover Password
4. Feedback/Contact Us
5. Save push token to Composr Server

More APIs can be integrated as required. The above written API calls use a standard format. All the API call methods have a completion handler attached and the CMSNetworkManager also has a delegate. If completion handler is available, the callback is executed using the completion handler. If the completion handler is nil, the delegate methods are called. The same case goes for failures too.

Initiating the CMSNetworkManager singleton class:

Initialize a generic network manager:

[CMSNetworkManager sharedManager];

Initialize network manager with a specific base url :

[[CMSNetworkManager sharedManager] initWithBaseURL:<domain\_url>];

The class has got two generic API methods one for GET and other for POST.

## GET

*Prototype:*

(void)executeGETWithParams:(NSDictionary \*)params forURL:(NSString \*)url onCompletion:(CMSResponseBlock)objectBlock onFaillure:(CMSErrorBlock)failBlock showLoader:(BOOL)showLoader;

*Description:*

Executes a get request on the given URL

 - params -> map of parameters to be sent as key value pair along with URL

 - URL -> URL to connect to

 - completion block

 - failure block

 - showLoader - Should a loader be shown blocking the screen when executing the API call.

*Usage:*

[self executeGETWithParams:<params\_map>

 forURL:<url>

 onCompletion:<completion handler>

 onFaillure:<failure\_handler>

 showLoader:<show loader ?>];

## POST

*Prototype:*

(void)executePOSTWithParams:(NSDictionary \*)params forURL:(NSString \*)url onCompletion:(CMSResponseBlock)objectBlock onFaillure:(CMSErrorBlock)failBlock showLoader:(BOOL)showLoader

*Description:*

Executes a post request on the given URL

 - params -> map of parameters to be sent as key value pair in body

 - URL -> URL to connect to

 - completion block

 - failure block

 - showLoader - Should a loader be shown blocking the screen when executing the API call.

*Usage:*

[self executePOSTWithParams:<params\_map>

 forURL:<url>

 onCompletion:<completion handler>

 onFaillure:<failure\_handler>

 showLoader:<show loader ?>];

Apart from this, the CMSNetworkManager has utility methods within it for parsing response, error, from a Composr API like:

1. isResponseValid
2. getResponse
3. getError

# More Utilities

Apart from the above utility methods, the Composr Mobile SDK contains some custom generic utility views that it has used to build the form builder. These methods can be reused separately if required. They are:

## NumberInput

Number only entry text field.

*Usage:*

NumberInput \*txtInput = [[NumberInput alloc] initWithFrame:<frame> defaultValue:<default\_value> placeHolder:<placeholder> supportFloat:<support floating point ?>];

## ComboBox

Textfield linked with a picker as accessory automatically.

*Usage:*

ComboBox \*txtInput = [[ComboBox alloc] initWithFrame:<frame>];[txtInput setOptions:<values of options>];[txtInput setKeys:<the param name or key of options>];[txtInput setDefaultValue:<default value to be selected>];

## DatePicker

Textfield linked to a date picker automatically.

*Usage:*

DatePicker \*txtInput = [[DatePicker alloc] initDatePickerWithTime:includeTimeChoice frame:<frame>];

## PhotoUpload

A control that acts as a photo upload form control by itself. Has option to take image from photo gallery as well as camera.

*Usage:*

PhotoUpload \*photoView = [[PhotoUpload alloc] initWithFrame:<frame>];[photoView setImage:<a no image placeholder image>];