

Cockpit and Cliq Collections Documentation

Introduction

This documentation supports the new Cliq Collections module for the Cockpit CMS. At Artur's suggestion we moved the existing Cockpit Collections module from Core to Addons and then worked on the version in Addons.

If a Developer wishes to implement the Cliq Collections module, they should delete or archive the existing Collections module directory in \core and copy the Cliq Collections module into \addons. No additional configuration is required.

The main purpose for the development of the Cliq Collections module was the requirement to provide support for a better display of Collections entries using the well respected DataTables jQuery plugin provided by Allan Jardine of Spry Media. Although there many reasons for choosing DataTables as opposed to other possibilities is the fact that DataTables now support UIKit and thus the result achieved with DataTables is almost indistinguishable from a standard Cockpit table.

This is the second attempt at providing a table facility that we have produced over the last few months. The Cliq Collections module supports our simple modifications to an existing Cockpit table as a selectable alternative to the use of Datatables.

This is the first release of the Cliq Collections module and there are a number of improvements that have already been visualised, designed and planned. However your feedback would be most welcome.

This Documentation supports the release of the Cliq Collections Module and also provides additional information about the entire subject of Fields and their configuration. However this Documentation about DataTables should be read and understood in conjunction with the Guide, Examples and Tutorial on the DataTables website at <https://datatables.net/>.

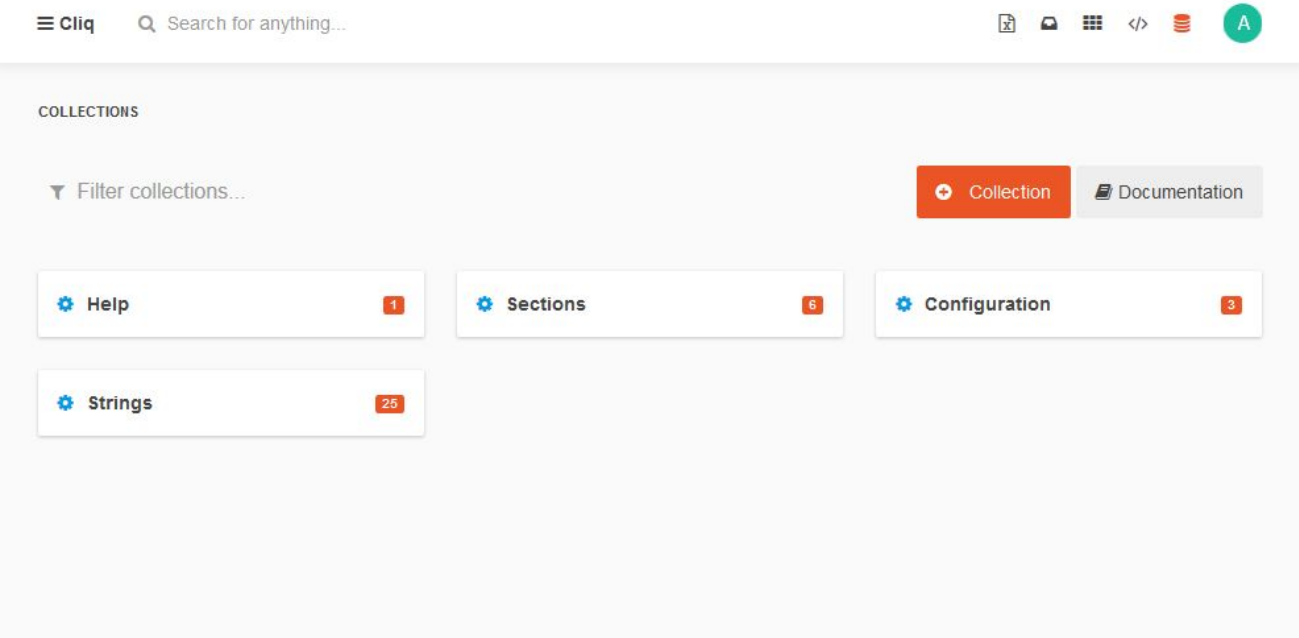
Finally, the Cliq Collection module only really comes to life when it is supporting a significant number of tables and these tables have a large amount of data.

If you wish to see how we have combined our Cliq web application framework with Cockpit CMS, please visit <https://github.com/webcliq/Cliq>. We anticipate making this demonstration system available from the beginning of May 2016.

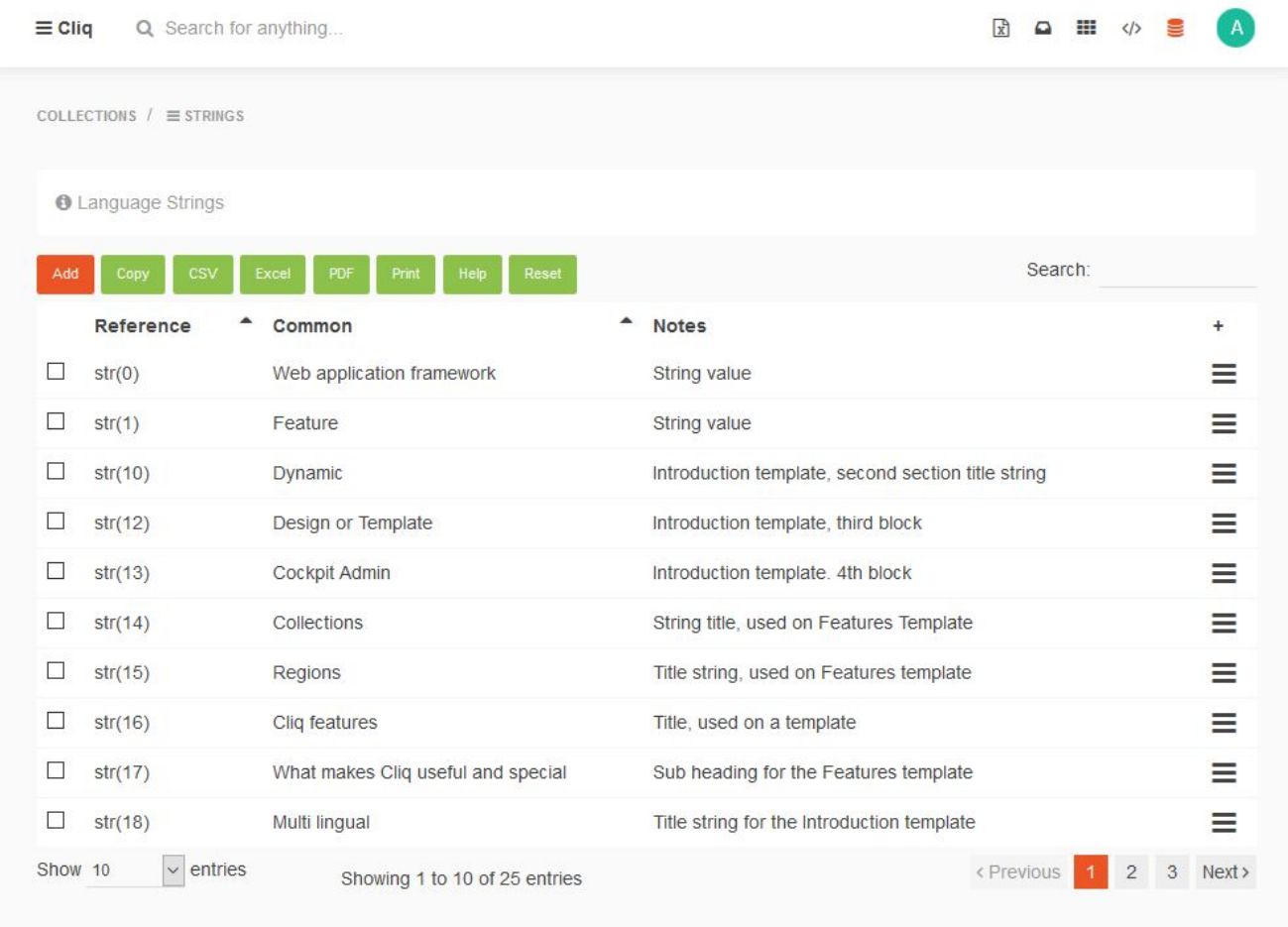
Cockpit and Cliq Collections Documentation

Collections

Online access to this Documentation is available from the Cliq Collections module main page.



A fully populated display of a Collection.



Cockpit and Cliq Collections Documentation

The amended Collection entry screen.

COLLECTIONS / COLLECTION

Name
string

Label
Strings

Description
Language Strings

Options

object {4}
 fields [4]
 menu {3}
 topbuttons [8]
 buttons : treetable,d

Fields

reference ↔ 1- [list] [settings] [delete]

common ↔ 2-3 [list] [settings] [delete]

text ↔ 1-2 [list] [settings] [delete]

notes ↔ 1-2 [list] [settings] [delete]

+ Add field

Save Show entries Close

Display in Datatable

Show in system menu

We have made two obvious changes to the Collection Entry screen. We have converted the Options textarea into a fully featured JSON editor and the button entitled Sortable on a standard Collection entry screen now invokes the DataTables display.

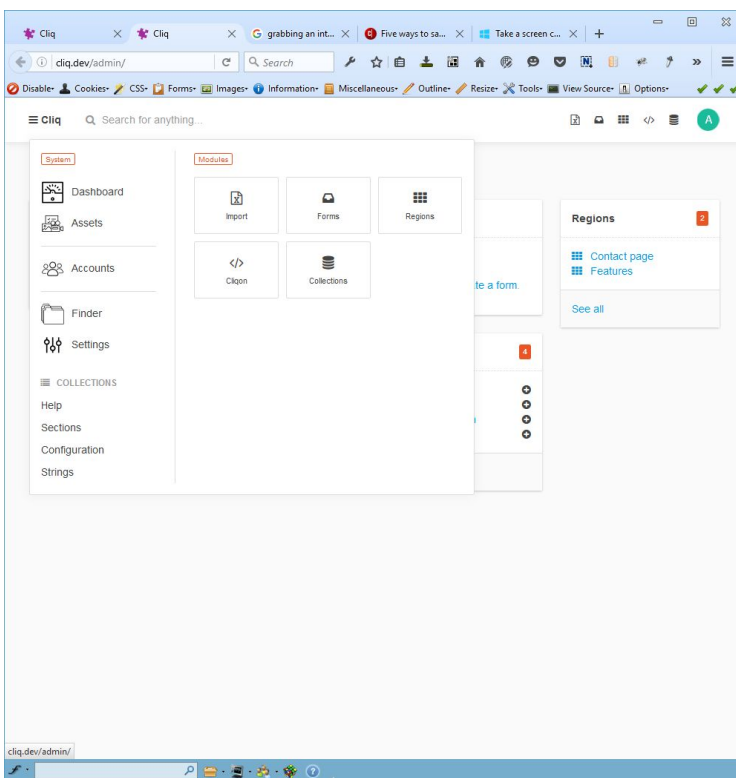
However for completeness, we will describe all the activities performed on the Collection create and Edit activity including subsequent reference to the database record and parameter files generated by the activity.

Cockpit and Cliq Collections Documentation

When you create a Collection, you are required to give it a name. In our opinion this name should be in lowercase, must be unique among Collections and should definitely not contain any spaces. The name is subsequently used to create the first part of a filename by which the Collection parameter file is known and is also the first part of a reference name in the database.

The parameter file names become collectionname.collection.php and are stored in `\cockpit\storage\collections`.

You are then asked to enter a single language Label or Title for the field which will be used in most places throughout Cockpit Admin. A textarea is provided to enter a longer single language description of the Collection.



The JSON / Code formatted textarea is replaced in Cliq Collections with the JSON Editor authored by [Jos de Jong](#). This is a sophisticated JSON editor that provides support for creating a JSON array as code or as a form. In a system such as Cockpit, which relies heavily on the use of JSON, the introduction of a fully featured JSON Editor wherever possible is extremely valuable.

Below the JSON Editor are two radio buttons. The first, entitled “Datatable” provides the switch to select whether the entries in this Collection are displayed as a DataTable or a simple pageable tabular layout. The second button selects whether this collection shall appear in the system menu.

This completes the Header for the new Collection and it might be a good idea to save your entries at this time.

Collection Fields

Click on “Add Field” to start the process of adding fields and to add new fields. When you click on the link, a new row tile containing the defaults for a new Row is created.

Name

Like the name for the Collection, it makes sense to use one word, in lower case and with no spaces. Obviously the name must be unique within that Collection but may be the same as another name in another Collection. One adds or edits the name of the Collection, in place, upon its row tile.

Cockpit and Cliq Collections Documentation

Width

When a row is created, the default width is 1-1. This is the nomenclature used within UIKit to distinguish elements of a Grid. If you click on the "<--> 1-1" area, a small dropdown is revealed with options to select various fractions of a row. As an example, if one wanted two fields to appear side by side on the Collections entry form, one would select 1-2 for both fields.

Visible on Pageable list

A list icon is displayed that is either selected (green) or not selected (grey). Setting the icon / colour to selected indicates that the field will appear on the Pageable listing. The controls and options for a DataTables field have to be selected in other ways.

Invoke popup modal for additional field definitions

Click on the Cog icon causes a modal popup to be displayed with the Form Field additional Definitions form.

The first element on the form is a drop-down list of Field Types. A list of all the Field Types is provided later in this documentation. In a future release of this Documentation, a comprehensive tutorial and list of examples will be provided.

The second field represents the Column Header label and Form Field caption and the third field is a longer description which is displayed on the Admin Collections form and in certain places as a roll-over.

Below the Description field is a text area entitled Options JSON. In order to take advantage of this field, it is necessary to create whatever you want to enter as a valid JSON array. Just in case the person entering text at this point, has concerns about generating a valid JSON array, they may like to visit jsonlint.com in order to compose and/or check their proposed entry.

A helper with what is essential and/or suitable as array entries is detailed below. Here is a simple example that would populate a Select drop-down:

```
{
  "options": "label1,label2,label3"
}
```

Finally, you can check checkboxes, to say that the field is "Required" and that it should be "Localized". The latter option causes the Collection Entry form to display a drop-down language select box which will be relevant for any Collection fields that have been "localized".

The Cliq Documentation explains how the localization process works, both in terms of administration and at the front-end.

Delete this field

This icon does the obvious and removes the field row from the display.

Cockpit and Cliq Collections Documentation

Save, Show entries or Close

It makes sense to save your activities on a regular basis. If you click on Close, you are returned to the main Collections screen. If you click on Show Entries, you are taken to the display of entries.

Parameter Files

At this time the parameters for a Collection (or Region and Form) are not stored in the database but are stored as an array in script files with a .PHP extension within the \cockpit\storage\collections\ sub-directory¹.

The script file is created, updated or deleted by the Save action on the Collections definition form.

The layout of the parameter file is intuitive and may be edited by hand or generally copied about. The combination of the SQLITE database and tables plus these parameter constitutes the definition of a system.

```
return array (
  'name' => 'help',
  'label' => 'Help',
  '_id' => 'help571e3d864c4f7',
  'fields' =>
  array (
    0 =>
    array (
      'name' => 'reference',
      'label' => 'Reference',
      'type' => 'text',
      'default' => '',
      'info' => 'Cross reference to Help page',
      'localize' => false,
      'options' =>
      array (
        'placeholder' => 'Table name',
        'required' => true,
      ),
      'width' => '1-3',
      'lst' => true,
      'required' => true,
    ),
    ....
    ....
    'sortable' => true,
    'in_menu' => true,
```

¹ We believe that we could and should convert these physical files to JSON arrays stored in the database.

Cockpit and Cliq Collections Documentation

```
'_created' => 1461599622,
'_modified' => 1461608352,
'description' => 'Help for the admin system',
'options' =>
array (
  'fields' =>
  array (
    0 => '_id', 1 => 'reference',
    2 => 'common', 3 => 'notes',
  ),
  'menu' =>
  array (
    'edit' => 'Edit',
    'delete' => 'Delete',
  ),
  'topbuttons' =>
  array (
    0 => 'add', 1 => 'copy', 2 => 'csv', 3 => 'excel', 4 => 'pdf',
    5 => 'print', 6 => 'help', 7 => 'reset',
  ),
  'buttons' => 'treetable,datatable,pageable',
),
);
```

Cliq Collections DataTable display

In order to implement DataTables in the Cliq Collections module, we did several things. Firstly we organised to use Views Partial “entries.datatable.php” for the DataTable and “entries.pageable.php” for our modified standard offering.

We moved the Script block out of \views\entries.php and rehomed the scripts in their respective partials view scripts.

Thus all of the Javascript coding that is required for DataTables, will be found “entries.datatable.php”. The blocks of code have been titled and commented, so that anyone who is familiar with DataTables will be able to understand and modify the Code.

We deliberately did not change the way in which the Collections data is read into “data.result” by the App.callmodule() and chose to process the data (on the Client side) to become suitable for DataTables. Perhaps in the future this might be tidied up. Thus, the Cliq Collections module does not use Server-side processing”. The entire data set is made available to the DataTables script and relies on the script to search through it, filter it, paginate it and so on. This will impose a practical top limit on the amount of Data that can be stored but realistically that is not a limit that would concern a user of Cockpit CMS.

Cockpit and Cliq Collections Documentation

Options for the Table as a whole and for the individual columns are stored in the Collections parameter file.

We have provided our Cliq demonstration system Collections parameter files as examples of how the module works. Please copy the PHP files to [\cockpit\storage\collections](#) from [\cockpit\modules\addons\Collections\assets\parameter_files](#).

Options for the Cliq Collections Datable

This is a summary list of the options that have been configured for DataTables via the Collections JSON Option Editor or the individual field JSON editor.

Please read and understand these options in conjunction with the Documentation, Guide, API and examples on the DataTables.Net website.

General Options

All data is loaded by the routine and formatted client-side to be used by DataTables.

JSON Array key (JAK) "fields" is the array of columns that will be used in the DataTable. The array must include the entry "_id" at position 0;

JAK "menu" is an array of keys and labels where the key equals a supported action that has been programmed in the DataTable Script.

JAK "topbuttons" is an array of entries which determines which buttons and facilities will displayed above the DataTable.

Add = Add new Record

Copy = DataTable function to copy all the table data to the clipboard as a set of textual records in CSV format

CSV = DataTable function to create and save all the table data as a set of textual records in CSV format to a named file

Excel = DataTable function to create and save all the table data as a XLSX format spreadsheet

PDF = DataTable function to save all the data to a PDF file that can be downloaded

Print = DataTable function to display all the data of the table in a printable format so that it can be printed.

Help = Displays a popup modal dialog contain the contents of a help record for this named collection. Thus it is necessary, before this option will work that you create suitable content in the Help Collection, using the name of this collection as the reference for the help entry. Thus Collection entitled "string" has a row in the Collection entitled "help" with a field called reference where the data reads "string". The explanation is then stored in field "text" which is also localized. Finally Reset reloads the Table.

Field Options

Please read this section in conjunction with the section below about the changes that the Collection module makes to the overall availability of additional options for a field.

The most likely options to be added to a Field from a DataTables point of view are things like

Cockpit and Cliq Collections Documentation

“orderable” and “filterable”. However a good read of the DataTables columns documentation will reveal the host of possibilities.

We have indicated in the Partial's where you would be able to amend the scripts for yourselves to add more options.

Cliq Collections Pageable display

The Cliq Collections pageable display is a standard Cockpit table display with the addition of a pager function to make the display tidier. In future versions of the Module, additional appropriate facilities may be added.

Roadmap

From a Cliq Collections Module point of view, we would like to introduce the ability to display the data as a Treetable.

Cockpit and Cliq Collections Documentation

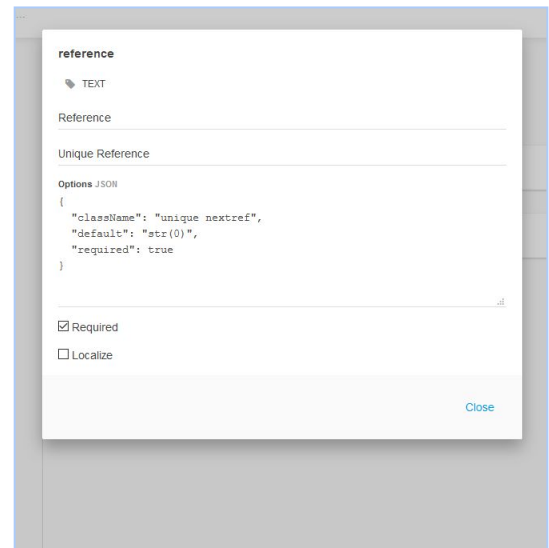
The use of Options in Field Types

In the Cliq Collections module, the Field types are augmented. However in order to explain the additions it is necessary to explain all the different processes involved in their configuration and implementation.

The Cockpit CMS Administrator is responsible for creating or changing a Field within a Collection. Admin clicks on the cog icon and the additional field definition form is displayed in the popup modal.

The modal popup displays a form which contains the following fields:

- The unlabeled Field Type selector.
- A non multi-lingual Caption for the field²
- A non multi-lingual longer descriptive title.
- A text area with the opportunity to enter options in a JSON format
- And two checkboxes indicating whether the field is multi-lingual and is a required field.



The screenshot shows a modal window titled 'reference' with a 'TEXT' icon. It contains the following fields: 'Reference' (text input), 'Unique Reference' (text input), 'Options JSON' (text area with a JSON object: `{ "className": "unique nextref", "default": "str(0)", "required": true }`), 'Required' (checked checkbox), and 'Localize' (unchecked checkbox). A 'Close' button is at the bottom right.

The two areas of interest to the Cliq Collections Module are the Field Types and changes to the JSON Options.

Field Types

Clicking on the Tag item or existing Field Type Choice, displays a drop-down select box with the following list of possible Field Types.

Boolean, Code, Color, Date, File, Gallery, HTML, Image, Location, Markdown, Multipleselect Object, Password, Radio, Rating, Repeater, Select, Set, Tags, Text, Textarea, Time, Wysiwyg

Developers should read the documentation about Riot.js to understand how the entries on the Select drop-down list have been generated.

The Select list elements are a summary of all the Field Tag files that have been compiled into Component.js by a Riot Build procedure using Node before the Cockpit system was uploaded to Github. Thus it is not possible to make “on the fly” alterations to these Tag files and expect the changes to be implemented immediately in the running Code. Nor would it be currently possible to make changes in an Addon Module and have them enhance or overwrite the base Cockpit system.

Although the manner in which Artur Heinz, Author of Cockpit has programmed the Fields system has created some immutable areas, it still leaves a great deal of room for maneuver.

² To be improved in future release of the Cliq Collections module

Cockpit and Cliq Collections Documentation

When one is creating a form and form field building system, it is necessary to provide all the possible field types that can be envisaged. With the introduction of the newer HTML5 form field types, the list of types now exceeds 30 in number.

In the next few sections we will detail all the active and essential settings for the current list of fields. All of these settings are implemented using key:value pairs in the JSON formatted options for each field.

An example that might be entered into the JSON options for the current Field type: TEXT:

```
{
  placeholder: "person@email.com",
  type: "email"
}
```

Current Field Tag options

Field **Boolean** is a UIKit button with Toggle and implements:

Value
Cls

Field **Code** is a Codemirror implementation which implements:

Syntax *// Default is Text (JSON or Javascript would be better)*

Field **Color** is a standard Input field but adds support for a colour picker through the use of Tinycolorpicker. It implements:

Bind *// Provides the opportunity to set a new data binding relationship*
Type *// Change Type=text to Type=email ??*
Placeholder
Cls
Required

Field **Date** is a standard Input field but adds support for the UIKit datepicker. It implements:

Bind
Placeholder
Cls
Required

Cockpit and Cliq Collections Documentation

Field **File** is a standard Input field but adds an additional button to provide for the uploading of files and other media. It implements:

Bind
Placeholder
Cls

Field **Gallery** is a complex facility that does not implement any options.

Field **HTML** is a standard Textarea but adds a simple rich text editor based on Codemirror and UIKit. It does not implement any options.

Field **Image** is a complex field that has both image display and image file upload functions. It does not implement any options.

Field **Location** displays a Google map and supports many functions for selecting and displaying position and markers. It does not implement any options.

Field **Markdown** does not appear to have been fully implemented.

Field **Multipleselect** is a hybrid type relying on UIKit to make it appear as a Select facility. It permits the selection of multiple options. It implements:

Options *// A comma separated list of labels/values (both label and value are the same)*

Field **Object** is a standard Textarea but adds support for Behave.js. Behave is a lightweight library for adding IDE style behaviors to plain text areas. See <http://jakiestfu.github.io/Behave.js/> for full details. It implements:

Placeholder
Cls
Required

Field **Password** is a standard Input field but with type = password. It adds the facility to show the text of a password through the implementation of the UIKit Passord script. It implements:

Bind
Cls

Field **Radio** uses the UIKit mechanism for displaying a Radio field as a button. The field supports multiple actions but does not implement Radio Group. It implements:

Buttons *// A comma separated list of labels/values (both label and value are the same)*

Cockpit and Cliq Collections Documentation

Field **Rating** is a type of range field permitting the selection of a numeric value. It implements:

Minimum *// Numeric, default is 0*
Maximum *// Numeric, default is 5*
Precision *// Numeric, default is 0 (no decimals)*
Icon *// Default is a star*
Remove

Field **Repeater** is a sub form or form within a form. By default the input fields of the repeating section are type = Text. The Repeater implements:

Options
Field *// Field type*

Field **Select** is a standard Select field. It implements:

Options *// A comma separated list of labels/values (both label and value are the same)*
Bind
Required
Cls

Field **Set** is also a repeater. It relies on the fact that “fields” have been defined and will generate an error if they are not configured. Thus it implements:

Fields *// Fields is an array of sub fields, where all the characteristics of a field are given, such as name, label, type etc.*
Multiple *// Is an array*
Bind *// Only one Bind, thus all the values in the fields must be concatenated, rather like tags*

Field **Tags** is an input field supporting the display of multiple inputs in the form of tags. It implements:

Autocomplete *// Is the value for the “source” attribute of a UIKit autocomplete scriptlet*
Placeholder

Field **Text** is a standard text input field. It implements:

Placeholder
Bind
Type *// Supports HTML5 modifiers such as email and url*
Cls

Cockpit and Cliq Collections Documentation

Field **Textarea** is a standard textarea field. It implements:

Placeholder

Bind

Rows

Cls

Required

Field **Time** is a standard Input field but adds support for the UIKit timepicker. It implements:

Bind

Cls

Required

Field **Wysiwyg** is a Textarea overlaid with a comprehensive Rich Text Editor utilising TinyMCE Version 4. It implements:

Cls

Rows

Editor *// permits the addition of extra options to the TinyMCE initialisation.*

Standard additional field options

In addition to the Options detailed previously on a field type by type basis. The standard Collections module implements:

Value *// A default value for the field*

It can also turn any field into a multi-lingual field.

Additional field options provided by the Cliq Collections module

In order to support the requirements of support for DataTables, it was necessary to add support for more options. As a result of programming support for additional options not catered for in the Riot Field Tags, we open up the possibility (that must be used with knowledge, care and common sense) to add a whole range of additional and necessary options to fields in general.

An example is the provision of support for a Class for a field (e.g. `className: "uk-text-primary"`), or validation rules and support for masking. In `entry.php` an additional function has been added to the Script after the field is mounted, utilising jQuery, to add the options as additional attributes to the field.