DCG Documentation

Release 2.1.0

Team DCG

25.10.2020

Inhaltsverzeichnis

1 Unterstützung und Spenden 2

2 Handbuch 3

3 Kochbuch 9

4 Referenz 10

Dies ist die offizielle Dokumentation des DC_General, eine Erweiterung für das Contao CMS.

Das Handbuch richtet sich an Entwickler, die Erweiterungen für Contao programmieren oder bestehende Erweiterungen auf Basis des DC_General anpassen möchten.

Die Erweiterung DC_General ist eine Alternative für den im Contao-Core enthaltenen DC_Table. Der DC_Table ist als „Driver“ in erster Linie für die Datenmanipulation wie z.B. Datensatz speichern, kopieren und löschen zuständig - zudem ist im DC_Table auch das Mehrfachbearbeiten eigenständig implementiert. Weiterhin kümmert sich der DC_Table um die Anzeige der Daten im Backend z.B. für die Listenansichten oder die Eingabemasken.

Der aktuelle DC_Table ist im Verlauf seiner Entwicklung zu einem recht „monolithischem Gebilde“ herangewachsen, so dass die Erweiterung und Modernisierung sehr schwierig bis unmöglich ist. Zudem ist es nicht möglich, den Code per Unittest zu prüfen.

Aufgrund dieser Schwierigkeiten wurde der DC_General als moderner „Data container“ entwickelt und nutzt die Möglichkeiten, die sich aus einer modernen OOP sowie dem Einsatz von Symfony ergeben, bestens aus. Der DC_General ist gegenüber dem DC_Table „Event driven“, hat eine Abstraktion der Datenquelle sowie eine verbesserte Konfiguration der Abhängigkeiten zwischen Datacontainers.

Wird der DC_General in eigene Erweiterungen implementiert stehen vielfältige Manipulationsmöglichkeiten für die Daten und das Contao-Backend über Events zur Verfügung. Mehr zu den zu den Funktionen unter…

Der DC_General ist in einigen großen Erweiterungen wie MetaModels, Avisota, syncCto oder Language2File implementiert.

Die gängige Abkürzung für den DC_General ist ‚DCG‘.

Diese Dokumentation gliedert sich in drei Bereiche:

Handbuch - Hier werden generelle Sachen des DCG dokumentiert.

Kochbuch - Workflows und ‚Best practice‘-Beispiele um mit dem DCG zu arbeiten.

Referenz - Dokumentation über Events, Schnittstellen und vielem mehr.

KAPITEL 1

Unterstützung und Spenden

Die Erweiterung DC_General ist eine kostenfreie Erweiterung auf der Basis von OpenSource und benötigt für eine kontinuierliche Weiterentwicklung die Unterstützung einer aktiven Community. Mitarbeit in Form von Programmierungen oder die Meldung von Fehlern sowie eigenen Workflows werden gern angenommen.

KAPITEL 2

Handbuch

2.1 Vorstellung des DCG

Das Handbuch richtet sich an Entwickler, die Erweiterungen für Contao programmieren oder bestehende Erweiterungen auf Basis des DC_General anpassen möchten.

Die Erweiterung DC_General ist eine Alternative für den im Contao-Core enthaltenen DC_Table. Der DC_Table ist als „Datencontainer-Treiber“ in erster Linie für die Datenmanipulation wie z.B. Datensatz speichern, kopieren und löschen zuständig - zudem ist im DC_Table auch das Mehrfachbearbeiten eigenständig implementiert. Weiterhin kümmert sich der DC_Table um die Anzeige der Daten im Backend z.B. für die Listenansichten oder die Eingabemasken.

2.1.1 Der DCG im Vergleich zum DC_Table

Der DCG hat im Vergleich zum DC_Table folgende Vorteile:

- Event Driven
- Objektbasierte Abstraktion der Definitionen
- Abstraktion der Datenquelle
- Verbesserte Konfiguration der Abhängigkeiten zwischen Datacontainers
- modularer Aufbau
- Prüfung der Daten vor Speicherung - nur wenn Daten konsistent sind, wird gespeichert
- Deep-Delete ohne anladen der referenzierten Datacontainer, mehr Variabilität da nicht abhängig von ptable-Bezug

Der DCG unterstützt alle *Callback-Aufrufe* von Contao und leitet diese in die eigenen Events um.

2.1.2 Ressourcen

- DCG auf Github
- DCG Handbuch auf Github

2.2 DCG installieren und aktualisieren

Der DCG wird üblicher Weise in einem eigenen Projekt über die Angabe „require“ in der composer.json eingebunden:

```
...
"require": {
    "php": "^7.1",
    "contao-community-alliance/dc-general": "^2.1",
    ...
}
...
```

Es ist aber auch möglich, den DCG manuell über den Contao Manager oder über die Konsole zu installieren. Für die Konsole erfolgt die Installation mit

```
php web/contao-manager.phar.php composer require
contao-community-alliance/dc-general
```

Über den Contao Manager bzw. „composer update“ wird der DCG aktuell gehalten.

2.3 Data-Container erstellen

Der Data-Container mit dem DCG wird analog wie mit dem DC_Table über die Konfiguration des DCA erstellt.

Als erster Punkt wird im Knoten `config` der Data-Container auf den DCG per `General` umgestellt.

```
$GLOBALS['TL_DCA']['tl_my_table'] = array
(
    // Config
    'config' => array
    (
        // Replace the data container Table with General.
        'dataContainer'    => 'General'
        'notCopyable'      => true, // wird erst ab DCG 2.2 untertützt
        'enableVersioning' => true, // wird erst ab DCG 2.2 untertützt
        'sql' => array
        (
            'keys' => array
            (
                'id' => 'primary'
            )
        )
        // *_callback per Event
    ),
    ...
```

Die übrigen, möglichen Parameter bei `config` können wie bei DC_Table eingesetzt werden. Der Knoten `ctable` für Kindtabellen ist nicht notwendig und wird dafür im Knoten `data_provider` behandelt (s.u.).

Die Callbacks könnten wie gehabt in `config` eingetragen werden und werden von dem DCG-Legacy-Builder mit in die Abarbeitung übernommen - besser ist es, die Aufgaben einen entsprechenden Event-Listener zu übergeben (siehe *Callbacks als Event*).

Der Standard-Datenprovider (Datentabelle) wird Knoten `data_provider` in `data_provider` angegeben.

```
$GLOBALS['TL_DCA']['tl_my_table'] = array
(
    ...
    // Add the data container configuration.
    'dca_config' => array
    (
        // Configure the data provider and all child data provider.
        'data_provider' => array
        (
            // The default data provider, for this data container.
            'default' => array
            (
                'source' => 'tl_my_table'
            ),
```

Gibt es Kindtabellen, werden ebenfalls diese im Knoten `data_provider` angegeben - beim DC_Table würde das in `ctable` erfolgen.

```
$GLOBALS['TL_DCA']['tl_my_table'] = array
(
    ...
    // Add the data container configuration.
    'dca_config' => array
    (
        // Configure the data provider and all child data provider.
        'data_provider' => array
        (
            ...
            // The child data provider for all children are has related␣
↪to this and has their child relation.
            // This must configure so when you delete this theme are all␣
↪relations deletes too. (deep delete)
            'tl_my_child' => array
            (
                'source' => 'tl_my_child'
            ),
            ...
```

Mit der Konfiguration der Kindtabelle(n) wird automatisch ein *deep delete* konfiguriert, d.h. wenn der Elterndatensatz gelöscht wird, werden automatisch auch alle Kind-Datensätze gelöscht.

Die Beziehung zwischen einer Kind- zur Eltern-Tabelle wird in den `childCondition` definiert.

```
$GLOBALS['TL_DCA']['tl_my_table'] = array
(
    ...
    // Add the data container configuration.
    'dca_config' => array
    (
    ...
     // Add the child condition. This will announce the relations.
     'childCondition' => array
     (
        array
        (
            'from'    => 'tl_my_table',
            'to'      => 'tl_my_child',
            'setOn'   => array
            (
                array
                (
                    'to_field'   => 'pid',
```

(continues on next page)

(continued from previous page)

```
                    'from_field' => 'id',
                ),
            ),
            'filter'  => array
            (
                array
                (
                    'local'     => 'pid',
                    'remote'    => 'id',
                    'operation' => '=',
                ),
            ),
            'inverse' => array
            (
                array
                (
                    'local'     => 'pid',
                    'remote'    => 'id',
                    'operation' => '=',
                ),
            )
        ),
        ...
```

Der Knoten `setOn` definiert die Relation zwischen Eltern- zu Kindtabelle.

Der Knoten `filter` definiert ein Array von möglichen Filterungen, um die Kinddatensätze einzugrenzen - eine Filterung ist Pflicht.

Der Knoten `inverse` ist optional, aber beschleunigt die Datenbankabfrage für eine Abfrage vom Kind- zur Elterntabelle.

Die Konfiguration für eine Kindtabelle ist analog der Elterntabelle. Beim `data_provider` wird statt `default` die Tabelle für `parent` angegeben.

```
$GLOBALS['TL_DCA']['tl_my_child'] = array
(
    // Config
    'config' => array
    (
        'dataContainer'               => 'General',
    ),
    // Add the data container configuration.
    'dca_config' => array
    (
        // Configure the data provider and all child data provider.
        'data_provider' => array
        (
            // The default data provider, for this data container.
            'parent' => array
            (
                'source' => 'tl_my_table'
            )
        ),
        // Add the child condition. This will announce the relations.
        'childCondition' => array
        (
            array
            (
                'from'    => 'tl_my_table',
                'to'      => 'tl_my_child',
                'setOn'   => array
```

(continues on next page)

(continued from previous page)

```
                (
                    array
                    (
                        'to_field'   => 'pid',
                        'from_field' => 'id',
                    ),
                ),
                'filter'  => array
                (
                    array
                    (
                        'local'     => 'pid',
                        'remote'    => 'id',
                        'operation' => '=',
                    ),
                ),
                'inverse' => array
                (
                    array
                    (
                        'local'     => 'pid',
                        'remote'    => 'id',
                        'operation' => '=',
                    ),
                )
            )
        )
    ),
```

Die übrigen Parameter im DCA werden analog dem üblichen Vorgehen wie bei einem „DC_Table-Projekt“ vorgenommen. Die Einstellungen können an einer Beispielkonfiguration für tl_theme nachvollzogen werden.

2.4 Mehrfachbearbeitung (edit/overrideAll)

Der DCG bringt eine eigene Mehrfachbearbeitung mit, welche einen erweiterten Funktionsumfang gegenüber der Standardmethode von Contao hat. Die Mehrfachbearbeitung wird automatisch mit der Definition *general* im DCA aktiviert (siehe *Data-Container erstellen*).

Die Vorteile sind:

- Vor-, Zurück- und Beenden-Button
- es werden nur die Properties für die Bearbeitungsauswahl angezeigt, die für die Bearbeitung relevant sind; werden verschiedenartige Widgets ausgewählt, ist nur noch die gemeinsame Schnittmenge an Properties sichtbar
- es stehen zwei weitere *eval*-Parameter für das DCA zur Verfügung als *doNotEditMultiple* und *doNotOverrideMultiple* für Ansteuerung der Sichtbarkeit für den jeweiligen Bearbeitungstyp.

Die zusätzlichen DCA-Parameter für die Mehrfachbearbeitung haben folgende Einsatzmöglichkeit:

- *doNotEditMultiple* - verhindert für ein Property das Editieren in der Mehrfachberabeitung
- *doNotOverrideMultiple* - verhindert für ein Property das Überschreiben in der Mehrfachberabeitung; z.B. Properties, die Unique bleiben müssen wie Spaltennamen für Tabellen von MetaModels.

Ein Beispiel für einen DCA-Eintrag:

```
$GLOBALS['TL_DCA']['tl_my_table'] = array
(
   ...
    'fields' => array
    (
        'my_field' => array
        (
            ...
            'eval' => array(
                'doNotEditMultiple'     => true, // Hide at editAll.
                'doNotOverrideMultiple' => true, // Hide at overrideAll.
           ...
```

KAPITEL 3

Kochbuch

3.1 DCG „Kochbuch“

In dem MetaModels „Kochbuch“ sind verschiedene Snippets, Tipps und Tricks rund um den Einsatz mit dem DCG zusammengefasst.

In die Auflistung können gern interessante oder ungewöhnliche Lösungen aufgenommen werden - bitte eigene „Rezepte“ oder Links zum Forum bzw. andere Webseiten an die folgende E-Mail senden:

3.1.1 Eingabemaske ohne Tabelle

Wir erstellen eine Eingabemaske, die direkt durch einen Navigationslink im Backend aufgerufen wird und die Daten aber in keiner Tabelle gespeichert werden.

In unserem Beispiel nutzen wir das für eine eigene Importmöglichkeit z.B. von CSV-Dateien.

Als erstes benötigen wir einen Navigationslink in der Backend-Navigation in der *config.php*. Anschließen manipulieren wir den Link, so dass wir direkt in einer Eingabemaske landen. Die Manipulation erfolgt über einen Hook von Contao *getUserNavigation* und binden den Hook als Service über die ‘service.yml‘ein.

Im letzten Schritt erstellen wir die Widgets für die Eingabemaske per DCA-Definition in einer *dcaconfig.php* und verarbeiten den Aufruf in einem EventListener. Das Besondere an der DCA ist, dass hier der *NoOpDataProvider* zum Einsatz kommt, der die übliche Speicherung in einer zugehörigen Tabelle unterbindet.

Die Dateien sind in einem Beispiel zusammengefasst.

KAPITEL 4

Referenz

4.1 DCG Referenz

Die Referenzen sind für Entwickler gedacht, die Funktionen aus dem DCG nutzen möchten.

4.1.1 DC_General API

Die DC_General API bildet die Schnittstelle zur eigenen Programmierung und Erweiterung.

4.1.2 DC_General Events

4.1.3 Callbacks als Event

Die Callbacks aus DC_Table werden über den Legacy-Builder abgefangen und im DCG verarbeitet. Es ist aber zu Empfehlen, statt des Callback- Aufrufes den entsprechenden Event direkt zu verwenden.

Die Events werden als Service angesprochen und können damit auch mit einer Priorität der Verarbeitungsreihenfolge versehen werden.

Folgend eine Auflistung, welcher Callback mit welchem Event seine Ersetzung hat:

Callback in `config`

Callback	Event
onload_callback	dc-general.factory.create-dc-general
onsubmit_callback	dc-general.model.post-persist
ondelete_callback	dc-general.model.post-delete
oncut_callback	dc-general.model.post-paste
oncopy_callback	dc-general.model.post-duplicate

Callback in `list/sorting`

Callback	Event
header_callback	dc-general.view.contao2backend.get-parent-header
paste_button_callback	dc-general.view.contao2backend.get-paste-root-button
paste_button_callback	dc-general.view.contao2backend.get-paste-button
child_record_callback	dc-general.view.contao2backend.parent-view-child-record

Callback in `list/label`

Callback	Event
group_callback	dc-general.view.contao2backend.get-group-header
label_callback	dc-general.view.contao2backend.model-to-label

Callback in `list/global_operations`

Callback	Event
button_callback	dc-general.view.contao2backend.get-global-button

Callback in `list/operations`

Callback	Event
button_callback	dc-general.view.contao2backend.get-global-button

Callback in `fields`

Callback	Event
load_callback	dc-general.view.contao2backend.decode-property-value-for-widget
save_callback	dc-general.view.contao2backend.encode-property-value-from-widget
options_callback	dc-general.view.contao2backend.get-property-options
input_field_callback	dc-general.view.contao2backend.build-widget
wizard	dc-general.view.contao2backend.manipulate-widget

4.1.4 DCA Mapping

In der folgenden Auflistung sind alle bekannten Angaben zum DCA aufgeführt:

```
$GLOBALS['TL_DCA']['tl_example
↪'] = array
(
    // Config
    'config' => array
    (
        'label'                 ␣
↪=> &$GLOBALS['TL_LANG']['tl_
↪example']['headline'],
        'dataContainer'         ␣
↪=> 'General',
        'ptable'                ␣
↪=> 'tl_parent',
        'dynamicPtable'         ␣
↪=> true, // require 'ptable
↪'=>''
        'ctable'                ␣
↪=> array('tl_child1', 'tl_
↪child2'),
        'validFileTypes'        ␣
↪=> 'jpg,png,gif',
        'uploadScript'          ␣
↪=> '',
        'closed'                ␣
↪=> true,
        'notEditable'           ␣
↪=> true,
        'notDeletable'          ␣
↪=> true,
        'switchToEdit'          ␣
↪=> true,
        'enableVersioning'      ␣
↪=> true,
        'doNotCopyRecords'      ␣
↪=> true,
        'doNotDeleteRecords'␣
↪=> true,
        'onload_callback'       ␣
↪=> array
        (
            array('<class␣
↪name>', '<method name>')
        ),
        'onsubmit_callback'     ␣
↪=> array
        (
            array('<class␣
↪name>', '<method name>')
        ),
        'ondelete_callback'     ␣
↪=> array
        (
            array('<class␣
↪name>', '<method name>')
        ),
        'oncut_callback'        ␣
↪=> array
        (
            array('<class␣
↪name>', '<method name>')
        ),
        'oncopy_callback'       ␣
↪=> array
        (
            array('<class␣
↪name>', '<method name>')
        ),
        'sql'                   ␣
```

```
-> `Data provider mapping`_
-> `Listing mapping`_
-> *ignored*
-> `Data provider mapping`_
-> *ignored*
-> *ignored*
-> *ignored*
-> *ignored*
->
->
->
->
->
->
-> *ignored*
->
   ^
   ^
   ^
->
   ^
   ^
   ^
->
   ^
   ^
   ^
->
   ^
   ^
   ^
->
   ^
   ^
   ^
-> *ignored*
   ^
   ^
   ^
   ^
   ^
   ^
   ^
   ^
->

-> `Deprecated DcGeneral␣
↪config`_
-> `Deprecated DcGeneral␣
↪config`_
-> `Deprecated DcGeneral␣
↪config`_
-> `Data provider mapping`_
   ^
   ^
   ^
```

4.2 Impressum, Datenschutz, Lizenz, Quellenangaben

4.2.1 Impressum

Die Angaben zum Impressum und rechtliche Hinweise finden Sie auf der Seite https://c-c-a.org/impressum

4.2.2 Datenschutz

Die Angaben zum Datenschutz finden Sie auf der Seite https://c-c-a.org/datenschutz

4.2.3 Lizenz der Dokumentation