Inline Relational Record Editing

D i p l o m a   T h e s i s

supervised by the
University of Applied Sciences Hof
Faculty of IT and Engineering
Applied Computer Science

Supervised by
Prof. Dr. Jürgen Heym
Alfons-Goppel-Platz 1
95028 Hof
Germany

Submitted by
Oliver Hader
Untere Siedlung 2
95131 Schwarzenbach a.W.
Germany

Hof, 26th February 2007

Foreword

During the last five years TYPO3 was fast-paced as Content Management Framework. TYPO3 is distributed as open source project under the General GNU Public License. The commitment and the decision of Dane Kasper Skårhøj made it possible to pool development. This also enforced improvement of the project.

To support the vision of TYPO3 - “inspiring people to share” - the results of this diploma thesis are also explicit available under the General GNU Public License and GNU Free Document License.

I am indebted to everybody who supported me in all aspects to realise this considerably project.

— Oliver Hader

TYPO3 Core Team Member

Table of Contents

List of Abbreviations

AJAX	Asynchronous JavaScript and XML
CMS	Content Management System
CSS	Cascading Style Sheets
CVS	Concurrent Versions System
DOM	W3C Document Object Model
EXT	TYPO3 Extension
GFDL	GNU Free Document License
GNU	GNU is not Unix (recursive acronym)
GPL	General GNU Public License
JSON	JavaScript Object Notation
OASIS	Org. for the Advance of Structured Information Standards
PHP	PHP Hypertext Preprocessor (recursive acronym)
PID	Page Identifier
SVN	Subversion
TCA	TYPO3 Table Configuration Array
TCE	TYPO3 Core Engine
UID	Unique Identifier
W3C	The World Wide Web Consortium

List of figures

1. Introduction

In consideration of functionalities and possibilities, TYPO3 [1] is an enormous and powerful tool for administrating content data on the internet and in intranets.

Complex scenarios can be solved and implemented by TYPO3 experts with an accordant effort. Unfortunately those solutions cannot always be applied easily and intuitionally. For instance, most editors do not have any engineering insight.

My original motivation is built on a realistic scenario. This case-study will be shown, analysed and enhanced in the following chapters.

1.1. Case-Study

The website of a travel business contains contents of about 300 different hotels. It has to be possible to create and manage this data using the TYPO3 back-end. Furthermore a hotel consists of different offers, such as a "wellness special" or "relax weekend". Additionally every single one of these offers can vary in price depending on the season customers chooses to book. Finally this results in a structure spanning three levels, beginning with one hotel as parent, many offers as first generation and also many prices as second generation. Each generation mentioned depends on the previous generation. This scenario is known as a composition of the cardinality 1:n.

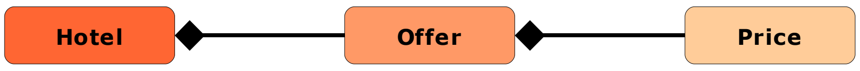


Figure 1: Composition of involved entities

1.2. General purpose

The general purpose of this work is to simplify and improve workflows of editors by adequate modifications. Avoidable clicks and changes of the view have to be disclaimed.

Furthermore, a technique shall be developed which offers the possibility to collect and handle hierarchical information spread over multiple levels.

2. Comparison

According to the general purpose this chapter will show the possibilities to handle multidimensional structures with TYPO3 4.0.x and outline the disadvantages. Finally, this results in a list of general requirements concerning improvement.

2.1. TYPO3 Core

TYPO3 was initially developed in 1997 by Dane Kasper Skårhøj. That initial version cannot be compared to nowadays state however. The obtainable TYPO3 distribution is the software core. It covers the basic concerns of the system regarding to content handling more than satisfactory. In the course of time the possibility to broaden functionalities to a specific demand was conceived by the so-called Extension Manager. Hence an adjustment to the core package was not necessary anymore. These extensions can reuse and build on the functionalities of the base by using a certain degree of object-orientation and inheritance. Since TYPO3 is implicit expandable by extensions of other parties it augmented is called a Content Management Framework. It establishes and defines certain basic conditions but nevertheless stays absolutely flexible for enhancements.

2.1.1. Structures using regular database models

The regular database model stores data in each field of a table. Each field and its usage is defined in the Table Configuration Array (TCA). TYPO3 offers two TCA types for handling relational data – "select" and "group" with the subtype "db".

The type "select" offers a selector box that shows records of another table, the so-called "foreign table". Thus it is possible to select a subset out of a superset of records. Additionally the TCA configuration property “foreign_table_where” allows to reduce this superset by a regular SQL “WHERE” clause. However the related records do not fully depend on their parent. Thus other parents could create relations to the same child records again. The creation of new child records can only be achieved by using a wizard. The standard TYPO3 “add wizard” opens a pop-up window that allows to enter data for the new record. The treatment of these sub-records works similar by using the “edit wizard”.

The type “group” with the subtype “db” resembles “select”. Instead of showing a selector box, a pop-up window opens that allows to select a record from anywhere in the page structure. The TCA property “allowed” defines which database tables could be used to get related. Creating new and editing available records can also be done using wizards.

Both TCA types store the relations as a comma separated sequence of UIDs. Due to this behaviour the parent record always has to be fetched from database. After extracting again the UIDs out of this list the child records can be fetched. Another disadvantage is that related children do not even know that they have relationships to parental elements.

Both types offer the possibility to use an intermediate table by setting the TCA property “MM”. Records of this intermediate table store the UID of the source element and the UID of the destination element. Furthermore it is possible to define individual sorting for each side of the relationship. This allows the disposal of direct SQL queries. The disadvantage is, that MM tables do not have a UID or PID and are not defined via TCA. The consequence of which is that regular MM relations cannot have different versions and cannot be reverted by using the history/undo functionality.

To edit a structure of three levels (e.g. hotel, offer and price) the pop-up window containing an offer item opens another pop-up window displaying a price item. To handle this, there have to be as many different windows as structural levels exist.

2.1.2. Structures using XML (“FlexForms”)

XML offers the possibility to build up multidimensional structures by default. The emerging tree with nodes of parents, children and grandchildren allows to be extended for any demand.

FlexForms support all available TCA types except itself and create a form-in-a-form environment. Nested FlexForms are not possible at all.

The disadvantages are, that this XML data is stored as a string. To fetch data of several sub nodes it is always necessary to parse the whole XML hierarchy and to extract the required elements. Also, the space of a blob field on the database is limited. If an error occurs during saving XML data maybe the whole structure is damaged.

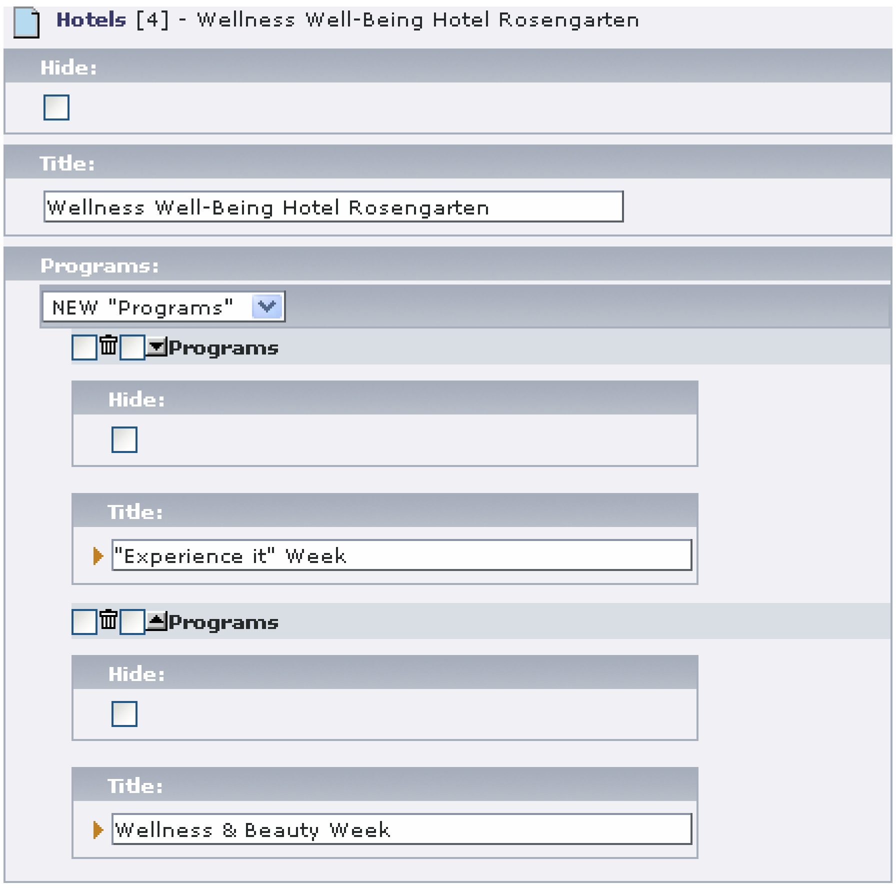
Furthermore, it is impossible to extend FlexForms dynamically. This disables the aptitude to create new child records on demand. Sorting elements inside XML is not implemented in TYPO3 by default.

2.2. TYPO3 Extensions

There are two extensions available on the TYPO3 Extension Repository that solve some of the disadvantages mentioned before. Thus, they allow to improve usability and flexibility.

2.2.1. Extension kb_tca_section

The extension "kb_tca_section" [2] by Bernhard Kraft addresses to regular database models. It provides a form-in-a-form solution that could dynamically create, move and delete child records. The handling of a third level of grandchildren is possible in just one window.



Hotels [4] - Wellness Well-Being Hotel Rosengarten

Hide:

Title: Wellness Well-Being Hotel Rosengarten

Programs: NEW "Programs"

Programs
Hide:
Title: "Experience it" Week

Programs
Hide:
Title: Wellness & Beauty Week

Figure 2: Using kb_tca_section for handling relational data

To create, delete or move child records it is necessary to choose an accordant action by clicking a check box or selecting an item of a selector box and save the parent record. After the view has been reloaded again a child record was created, deleted or moved by one position.

The relation itself is stored on the child record using a pointer field. This field contains the UID of its accordant parent record and fits the requirements of database normalization.

2.2.2. Extension dynaflex

The extension "dynaflex" [3] by Thomas Hempel addresses to the usage of XML to handle the mentioned FlexForms. In dependence on TCA dynaflex defines his own object, the Dynaflex Configuration Array (DCA). It can be dynamically changed during runtime.

This software package offers various helper techniques to dynamically manipulate a static XML structure. Thus it is possible to integrate new child elements or remove them. Sorting has to be implemented individually for the required demand by extension developers. The parent record has to be saved for each action so that changes will be applied to the hierarchy.

2.3. Requirements

The advantages and disadvantages of the mentioned available techniques will be compared to each other in a feature overview. Finally this results in a list of demands and requirements that shall be implemented in Inline Relational Record Editing.

"Usability" rates the possibility of intuitive handling of multidimensional structures of user's perspective. "Extensibility" rates the ability of extending existing structures. "Performance" rates the costs of handling many data and extracting only a subset of information. "Consistency" describes the resistance concerning errors on writing transactions and the ability to make changes undone.

	Regular database models	Regular FlexForms (XML)	Extension kb_tca_section	Extension dynaflex
Usability	--	---	-	-
Extensibility	-	+	+	++
Performance	-	---	++	---
Consistency	-	---	++	---

Figure 3: Comparison of features available for TYPO3 4.0.x

The conclusion of the comparison is to allocate a technique with the following abilities:

- Usability: Child records can be edited, created, removed or sorted easily. Reloading the whole frame must be avoided when it is not necessary. It is furthermore required that multidimensional structures can be administrated in just one form without opening several pop-up windows.
- Extensibility: Extending structures for developers has to be possible by using TYPO3 standards such as the TCA. Further individual programming for base functionalities have to be avoided. The new feature has to be available for all other TYPO3 related modules and extensions. Thus the development is done at the

TYPO3 core instead of creating a new extension.

- Performance: For selecting subsets out of a superset it is required to have a normalized database model. Fetching parental information must be avoided when it is not necessary.
- Consistency: It has to be possible to revert changes by a roll-back function such as "history/undo". Furthermore the solution widely has to be resistant against errors. If accidentally an error occurs during writing transactions it should not lead to a chain of errors affecting all other records and data.

2.4. Engaged technologies

To fulfil the mentioned requirements some additional technologies have to be engaged. AJAX [4] will be used to dynamically execute programs on the server environment. The JavaScript framework "prototype.js" [5] offers functions to easily perform cross-browser DOM actions, AJAX calls or event-handing. AJAX' intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes. The result is that the web page does not have to be reloaded each time the user requests a change. The prototype framework is the base of several other JavaScript frameworks.

For the reason that e.g. JavaScript structural data has to be exchanged besides regular HTML/DOM data, the JSON [6] container will be used. JSON is a subset of the object literal notation of JavaScript and is commonly used with that language.

3. Compendium: Case-Studies

This chapter is written as a handbook for the TYPO3 community. It tells developers how to implement extensions using Inline Relational Record Editing. It is published in the TER as “irre_tutorial” [7].

3.1. Reusing “select” type for 1:n relations

Simple 1:n relations, between one parent and many other children have formerly being realised using the TCA type “select”. The UIDs were stored in a list in one field on the parent table, separated by commas. The sequence of numbers in this list was the final sorting order of all affected child records.

```
1:  $TCA["tx_irretutorial_1ncsv_hotel"] = Array (
2:      ...
3:      "columns" => Array (
4:          "title" => Array (
5:              "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.title",
6:              "config" => Array (
7:                  "type" => "input",
8:                  "size" => "30",
9:              )
10:         ),
11:         "offers" => Array (
12:             "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.offers",
13:             "config" => Array (
14:                 "type" => "select",
15:                 "foreign_table" => "tx_irretutorial_1ncsv_offer",
16:                 “foreign_table_where” =>
17:                     “AND tx_irretutorial_1ncsv_offer.pid=###CURRENT_PID###
18:                     ORDER BY tx_irretutorial_1ncsv_offer.title”,
19:                 "maxitems" => 10,
20:             )
21:         ),
22:     ),
23:     ...
24: );
```

Figure 4: TCA type “select” for custom relations

The TCA type “select” (line 14 in code) defines the table of children by using the property “foreign_table” (line 15 in code). The disadvantage is that child records had to exist before they could be used to build a relationship to a parent record.

The new TCA type "inline" contains the functionalities of Inline Relational Record Editing. It now allows one to create new child records even if the parent record is a new one and was not saved before.

```
1:  $TCA["tx_irretutorial_1ncsv_hotel"] = Array (
2:      ...
3:      "columns" => Array (
4:          "title" => Array (
5:              "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.title",
6:              "config" => Array (
7:                  "type" => "input",
8:                  "size" => "30",
9:              )
10:         ),
11:         "offers" => Array (
12:             "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.offers",
13:             "config" => Array (
14:                 "type" => "inline",
15:                 "foreign_table" => "tx_irretutorial_1ncsv_offer",
16:                 "maxitems" => 10,
17:             )
18:         ),
19:     ),
20:     ...
21: );
```

Figure 5: TCA for 1:n relations with sequential list

Only the type definition changed from "select" to "inline" (line 14 in code). The disposal of the property "foreign_table" equals TCA type "select". This is the simplest way to migrate to Inline Relational Record Editing without losing old relations.

3.2. Normalized data structure for 1:n relations

On a database, relations in a list of UIDs cannot be selected directly by a query. Thus it is even better to use a normalized way of storing the relational information. In the sequential approach of the previous section the child records do not even know the UID of their parent. To get a subset of children, first the parent has to be loaded and processed. After that step all related child records have to be fetched and finally these results could be narrowed to get the designated subset.

In a normalized 1:n context, the UID of the parent record is stored in a separate field on every child record. This allows to directly acquire a subset of children related to an ancestor. Before Inline Relational Record Editing was available this could be done by using an individual wizard. The wizard opened a pop-up window and offered the possibility to create a new child record. The UID of the parent was written to a defined field in the database table by clicking the save-button.

This solution comes close to Inline Relational Record Editing, but unnecessarily opens new windows. Imagine a structure of four different levels. To initially create top down the required entities there would be one main back-end view and additionally three pop-up windows.

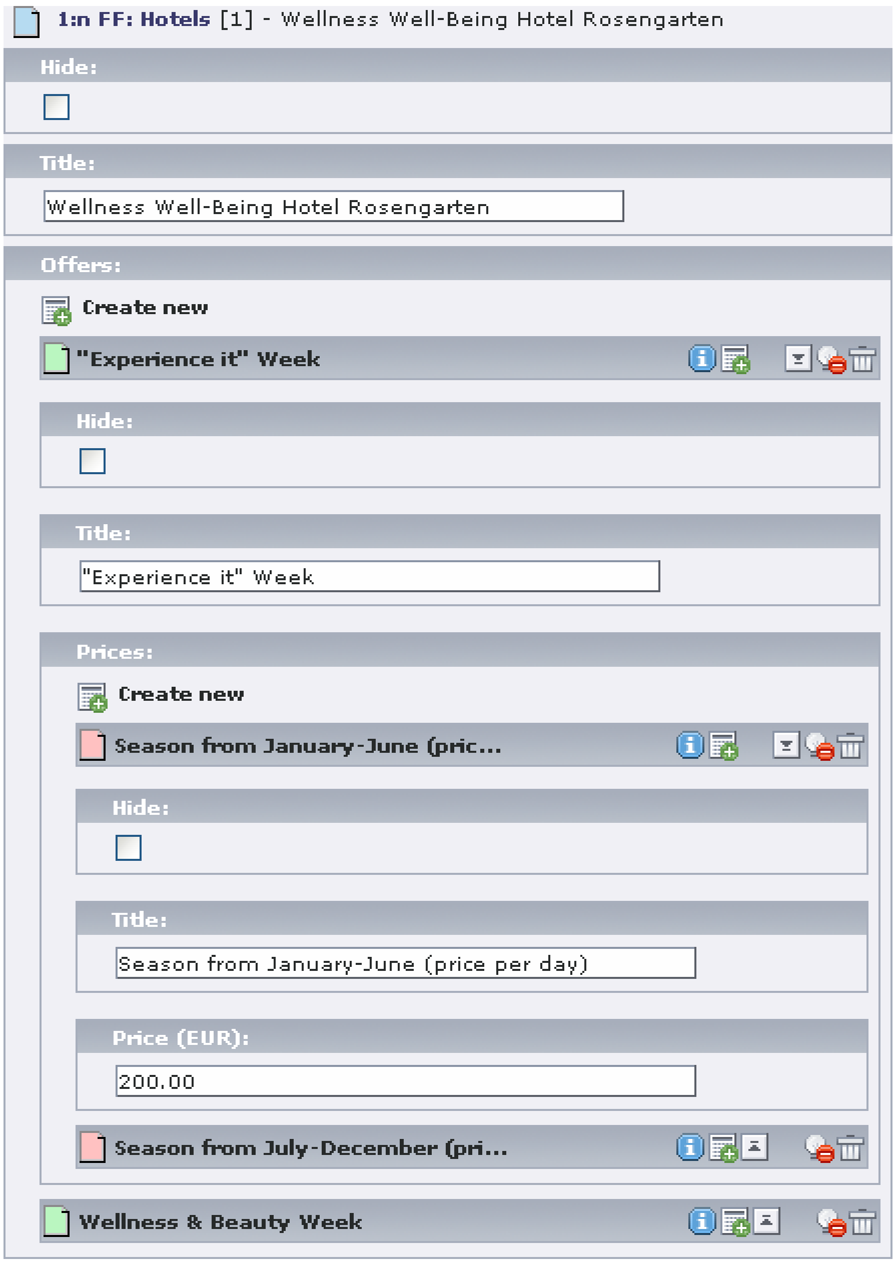
```
$TCA["tx_irretutorial_1nff_hotel"] = Array (
    ...
    "columns" => Array (
        "title" => Array (
            "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.title",
            "config" => Array (
                "type" => "input",
                "size" => "30",
            )
        ),
        "offers" => Array (
            "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.offers",
            "config" => Array (
                "type" => "inline",
                "foreign_table" => "tx_irretutorial_1nff_offer",
                "foreign_field" => "parentid",
                "foreign_table_field" => "parenttable",
                "maxitems" => 10,
            )
        ),
    ),
);
```

```
23: $TCA["tx_irretutorial_1nff_offer"] = Array (
24:     "columns" => Array (
25:         "parentid" => Array (
26:             "config" => Array (
27:                 "type" => "passthrough",
28:             )
29:         ),
30:         "parenttable" => Array (
31:             "config" => Array (
32:                 "type" => "passthrough",
33:             )
34:         ),
35:         "title" => Array (
36:             "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_offer.title",
37:             "config" => Array (
38:                 "type" => "input",
39:                 "size" => "30",
40:             )
41:         ),
42:     ),
43: );
44:
```

Figure 6: TCA for normalized 1:n relations

The field “offers” (line 11 of code) defines that of the accordant “foreign_table” (line 15). The property “foreign_field” (line 16) tells the system to store the UID of the parent record in the defined field on the child record. The key “foreign_table_field” (line 17) defines the field on the child side that stores the table name of the parent record. In this case it would be “tx_irretutorial_1nff_hotel”.

When the parent record is completely identified by its UID and table name on the child side this is also called a “weak entity”. Thus, the accordant child record knows its parent.



1:n FF: Hotels [1] - Wellness Well-Being Hotel Rosengarten

Hide: ☐

Title: Wellness Well-Being Hotel Rosengarten

Offers:

Create new

"Experience it" Week

Hide: ☐

Title: "Experience it" Week

Prices:

Create new

Season from January-June (pric...

Hide: ☐

Title: Season from January-June (price per day)

Price (EUR): 200.00

Season from July-December (pri...

Wellness & Beauty Week

Figure 7: Example view of 1:n relations

3.3. Intermediate tables for m:n relations

It could be required for child records to be associated with different parent records and vice versa. If this is the case, then an intermediate table containing only the relational information will be necessary.

Before IRRE was integrated TYPO3 supported the so-called MM-Relations (many-to-many) to handle this. MM relations have a very limited structure. It is not possible to edit a record of a MM table directly. Furthermore no history/undo function is available for MM relations.

IRRE offers a new way to get a more flexible approach. Intermediate tables are now used to have a regular TCA definition. This includes the existence of UID, PID, timestamp and some other default fields. Thus, modifications on a relationship can be reverted to a former state.

To apply this functionality an additional table is required – the intermediate table. Auxiliary the involved entities have to refer to that foreign table and must define in which field their UID information shall be lodged.

The standard deployment on intermediate tables are bidirectional relations. "Bidirectional" describes the way a relationship between two objects can be visualized. In this case a parent record can see its children and a child knows about its parents.

3.3.1. Bidirectional asymmetric m:n relations

Asymmetric behaviour of relations is the most common rule. The condition for a relation to be asymmetric is that two different tables are affected. The opposite is a symmetric relation which is described in the following section. To get a step-by-step introduction we just relate hotels to offers and care about prices later on.

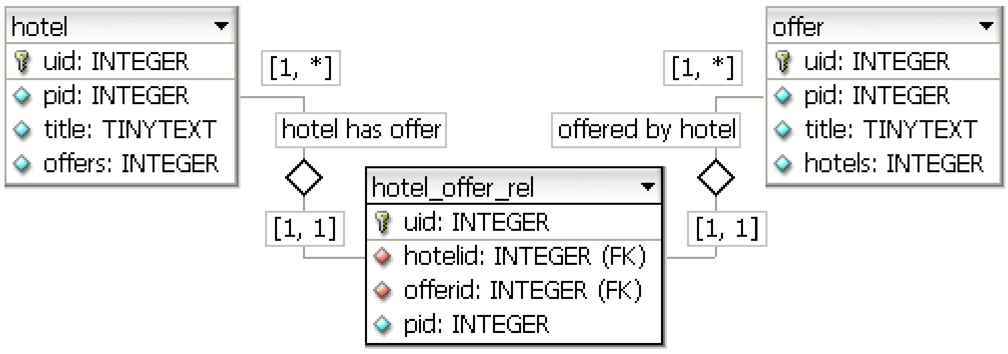


Figure 8: UML scheme of simple bidirectional asymmetric relations

The prefix “tx_irretutorial_...” of each table is missing in the UML scheme to get a better overview. “hotel_offer_rel” is the intermediate table with the information which hotel is related to which offer and vice versa.

```
$TCA["tx_irretutorial_mnasym_hotel"] = Array (
    ...
    "columns" => Array (
        "title" => Array (
            "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.title",
            "config" => Array (
                "type" => "input",
                "size" => "30",
            )
        ),
        "offers" => Array (
            "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.offers",
            "config" => Array (
                "type" => "inline",
                "foreign_table" => "tx_irretutorial_mnasym_hotel_offer_rel",
                "foreign_field" => "hotelid",
                "foreign_sortby" => "hotelsort",
                "foreign_label" => "offerid",
                "maxitems" => 10,
            )
        ),
    ),
);
$TCA["tx_irretutorial_mnasym_hotel_offer_rel"] = Array (
    "columns" => Array (
        "hotelid" => Array (
            "label" => "LLL:EXT:[locallang-path]:tx_irretutorial_hotel_offer_rel.hotelid",
            "config" => Array (
                "type" => "select",
                "foreign_table" => "tx_irretutorial_mnasym_hotel",
                "maxitems" => 1,
            )
        ),
        "offerid" => Array (
            "label" => "LLL:EXT:[locallang-path]:tx_irretutorial_hotel_offer_rel.offerid",
            "config" => Array (
                "type" => "select",
                "foreign_table" => "tx_irretutorial_mnasym_offer",
                "maxitems" => 1,
40:             )
41:         ),
42:         "hotelsort" => Array (
43:             "config" => Array (
44:                 "type" => "passthrough",
45:             )
46:         ),
47:         "offersort" => Array (
48:             "config" => Array (
49:                 "type" => "passthrough",
50:             )
51:         ),
52:     ),
53: );
54: $TCA["tx_irretutorial_mnasym_offer"] = Array (
55:     ...
56:     "columns" => Array (
57:         "title" => Array (
58:             "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_offer.title",
59:             "config" => Array (
60:                 "type" => "input",
61:                 "size" => "30",
62:             )
63:         ),
64:         "hotels" => Array (
65:             "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_offer.hotels",
66:             "config" => Array (
67:                 "type" => "inline",
68:                 "foreign_table" => "tx_irretutorial_mnasym_hotel_offer_rel",
69:                 "foreign_field" => "offerid",
70:                 "foreign_sortby" => "offersort",
71:                 "foreign_label" => "hotelid",
72:                 "maxitems" => 10,
73:             )
74:         ),
75:     ),
76: );
```

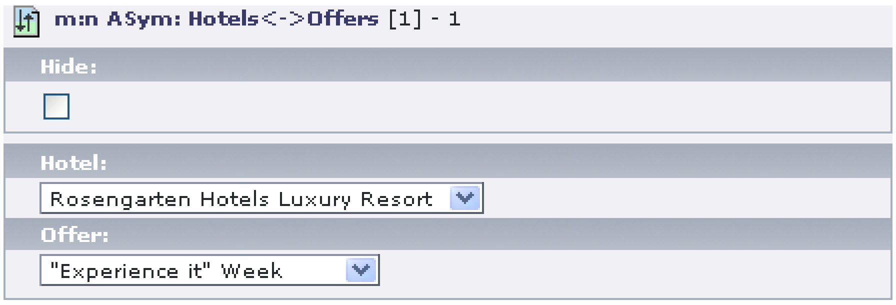
Figure 9: TCA for defining simple bidirectional asymmetric relations

The TCA source code example from above implements the structure shown in Figure 8. The field to inherit the functionality of Inline Relational Record Editing (line 11) is set to manage children of the intermediate table by "foreign_table" (line 15) instead of the offer table directly. The UID of the parent record will be written to the "foreign_field" on the intermediate table (line 16). Furthermore manual sorting and the field to hold this information is defined (line 17).

"foreing_label" tells TCEforms from which field the specific record title should be generated (line 18). In this case the field "offerid" of the intermediate table is used, which holds records of the entity "offer". Thus, the title of the related offer record will be shown inside the hotel record.

If the optional setting “foreign_label” is missing, TCEforms shows the default label defined in $TCA[<table>]['ctrl']['label'].

The field “hotelid” on the intermediate table stores the UID of the parent hotel. It is set to be of TCA type “select” and is pointing back to its own table (line 26). The field “offerid” works similar (line 34). But it is very important that these two fields only allow a maximum of one item to be selected (“maxitems” must be set to one). Thus, it would also be possible to edit the record of the intermediate table directly.


m:n ASym: Hotels<->Offers [1] - 1

Hide:

☐

Hotel:

Rosengarten Hotels Luxury Resort

Offer:

"Experience it" Week

Figure 10: Editing relation on intermediate table directly

Figure 10 shows the possibility of directly selecting one hotel and one offer that should be related to each other. Of course, it doesn't make any sense to edit the intermediate table directly. This is just to transfer the idea and behaviour behind Inline Relational Record Editing.

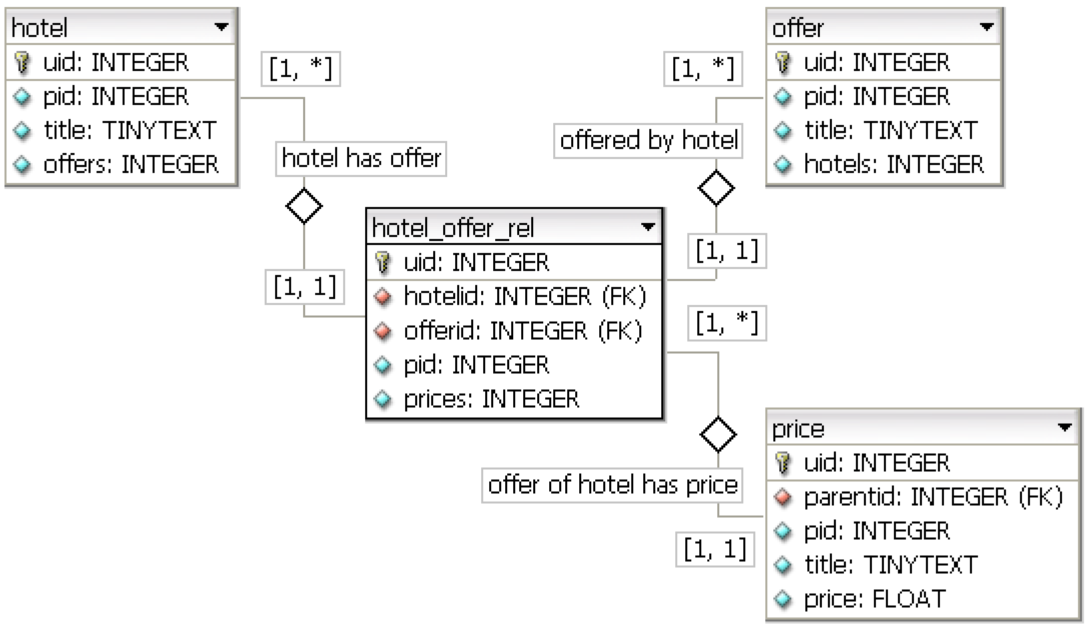


Figure 11: UML scheme of extended bidirectional asymmetric relations

Since the price depends on the offer of a hotel it only concerns the relationship. The solution to integrate prices in this scheme is to define them as an attribute on the intermediate table used for hotels and offers.

```
$TCA["tx_irretutorial_mnasym_hotel_offer_rel"] = Array (
    ...
    "columns" => Array (
        ...
        "prices" => Array (
            "label" => "LLL:EXT:[locallang-path]:tx_irretutorial_hotel_offer_rel.prices",
            "config" => Array (
                "type" => "inline",
                "foreign_table" => "tx_irretutorial_mnasym_price",
                "foreign_field" => "parentid",
                "maxitems" => 10,
            )
        ),
        ...
    ),
);
$TCA["tx_irretutorial_mnasym_price"] = Array (
    ...
    "columns" => Array (
        "parentid" => Array (
            "config" => Array (
                "type" => "passthrough",
            )
        ),
        "title" => Array (
            "exclude" => 1,
            "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_price.title",
            "config" => Array (
29:                 "type" => "input",
30:                 "size" => "30",
31:                 "eval" => "required",
32:             )
33:         ),
34:         "price" => Array (
35:             "exclude" => 1,
36:             "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_price.price",
37:             "config" => Array (
38:                 "type" => "input",
39:                 "size" => "30",
40:                 "eval" => "double2",
41:             )
42:         ),
43:     ),
44: );
```

Figure 12: TCA for extended bidirectional asymmetric relations

The field "price" (line 5 of code) is used as new attribute on the intermediate table used for hotels and offers. The utilization of attributes in general is explained in the following section "Attributes for each m:n relation".

The field "price" is of the TCA type "inline" again (line 8) to allow the integration of several child records. It is used to be a normalized 1:n relation as described in "Normalized data structure for 1:n relations" in a previous section (lines 9-10 and 20).

3.3.2. Bidirectional symmetric m:n relations

Special cases of bidirectional asymmetric relations are symmetric relations. The adjective symmetric describes the self referencing property to the same table. It could be demanded that entities of the same type have a relationship. Symmetric m:n relations handle this situation.

The travel business case-study was a good example: A relationship between one or more hotels to each other as branch offices shall be achieved. Indeed, a better matching task would be the representation of a family. The husband is married to its spouse and both have two children. All involved objects are of the same type – they are humans or persons. Every relation is bidirectional symmetric.

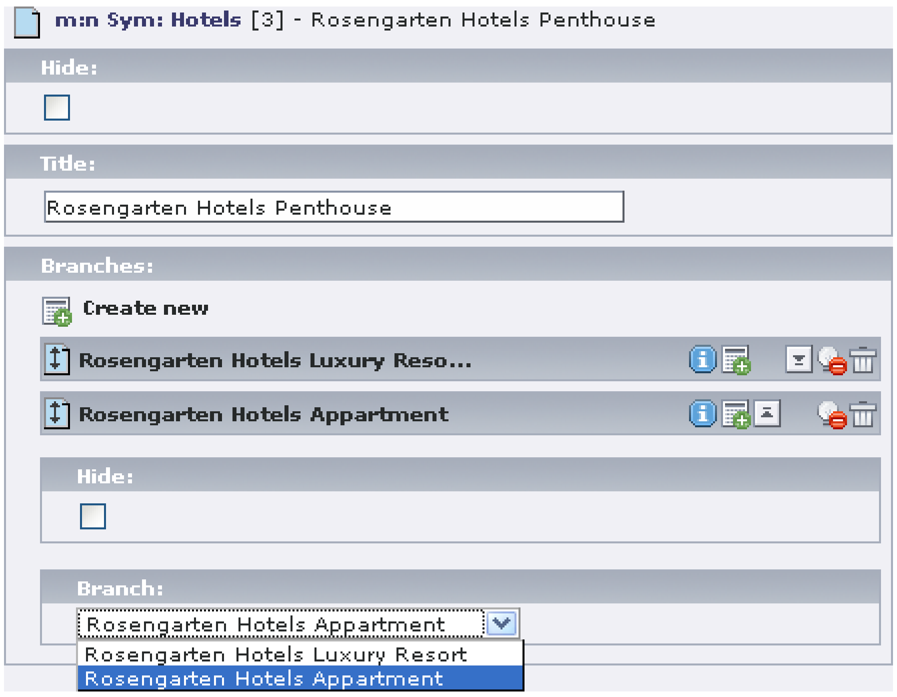
```
$TCA["tx_irretutorial_mnsym_hotel"] = Array (
    ...
    "columns" => Array (
        "title" => Array (
            "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.title",
            "config" => Array (
                "type" => "input",
                "size" => "30",
            )
        ),
        "branches" => Array (
            "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.branches",
            "config" => Array (
                "type" => "inline",
                "foreign_table" => "tx_irretutorial_mnsym_hotel_rel",
                "foreign_field" => "hotelid",
                "foreign_sortby" => "hotelsort",
                "foreign_label" => "branchid",
                "symmetric_field" => "branchid",
                "symmetric_sortby" => "branchsort",
                "symmetric_label" => "hotelid",
                "maxitems" => 10,
            )
        ),
    ),
);
$TCA["tx_irretutorial_mnsym_hotel_rel"] = Array (
    ...
    "columns" => Array (
        "hotelid" => Array (
            "label" => "LLL:EXT:[locallang-path]:tx_irretutorial_hotel_rel.hotelid",
            "config" => Array (
                "type" => "select",
                "foreign_table" => "tx_irretutorial_mnsym_hotel",
                "maxitems" => 1,
            )
        ),
        "branchid" => Array (
            "label" => "LLL:EXT:[locallang-path]:tx_irretutorial_hotel_rel.branchid",
            "config" => Array (
                "type" => "select",
                "foreign_table" => "tx_irretutorial_mnsym_hotel",
                "maxitems" => 1,
            )
        ),
        "hotelsort" => Array (
            "config" => Array (
                "type" => "passthrough",
            )
        ),
        "branchsort" => Array (
            "config" => Array (
                "type" => "passthrough",
            )
        ),
    ),
);
```

Figure 13: TCA for defining bidirectional symmetric relations

The field "branches" (line 11) links to the intermediate table but defines the two pointer fields "foreign_field" and "symmetric_field" (line 16 and 19). The UID of the current main record will be set to one of these fields according to which record initially created the record in the intermediate table. If we edit the creator the "foreign_field" is used. If we edit the related record the "symmetric_field" is used.

The disposal of "foreign_sortby" and "symmetric_sortby" (lines 17 and 20) equals the previously mentioned behaviour.

The pointer fields "hotelid" and "branchid" resemble the bidirectional asymmetric approach. However, both fields are pointing back to the same table of the entity "hotel" in this case.


m:n Sym: Hotels [3] - Rosengarten Hotels Penthouse

Hide:

Title:
Rosengarten Hotels Penthouse

Branches:
Create new
Rosengarten Hotels Luxury Reso...
Rosengarten Hotels Appartment

Hide:

Branch:
Rosengarten Hotels Appartment
Rosengarten Hotels Luxury Resort
Rosengarten Hotels Appartment

Figure 14: Bidirectional symmetric m:n relations of hotels

3.3.3. Attributes for each m:n relation

The utilization of standard TCA instead of the standard MM style to implement intermediate tables offers the possibility to use custom attributes for each relation. A new attribute simply has to be added in the TCA of the intermediate table and the SQL table definition. Rendering and data handling is again done by the TYPO3 Core Engine.

In the following example the relation between hotels and offers is attributed by a select field to define quality and a check box to denote an all-inclusive offer.

```
1:  $TCA["tx_irretutorial_mnattr_hotel_offer_rel"] = Array (
2:      ...
3:      "columns" => Array (
4:          "hotelid" => Array (...),
5:          "offerid" => Array (...),
6:          "hotelsort" => Array (...),
7:          "offersort" => Array (...),
8:          "quality" => Array (
9:              "label" => "LLL:EXT:[locallang-path]:tx_irretutorial_hotel_offer_rel.quality",
10:             "config" => Array (
11:                 "type" => "select",
12:                 "items" => Array (
13:                     Array("(1 star) *", "1"),
14:                     Array("(2 stars) **", "2"),
15:                     Array("(3 stars) ***", "3"),
16:                     Array("(4 stars) ****", "4"),
17:                     Array("(5 stars) *****", "5"),
18:                 ),
19:             )
20:         ),
21:         "allincl" => Array (
22:             "label" => "LLL:EXT:[locallang-path]:tx_irretutorial_hotel_offer_rel.allincl",
23:             "config" => Array (
24:                 "type" => "check",
25:             )
26:         ),
27:     ),
28: );
```

Figure 15: TCA for adding attributes to each relation

The fields "quality" and "allincl" (lines 8 and 21) define regular TCA types to be used as attributes on each relationship between hotels and offers. The new TCA type "inline" could also be used in this case (see the example to integrate prices in the previous section "Bidirectional symmetric m:n relations").

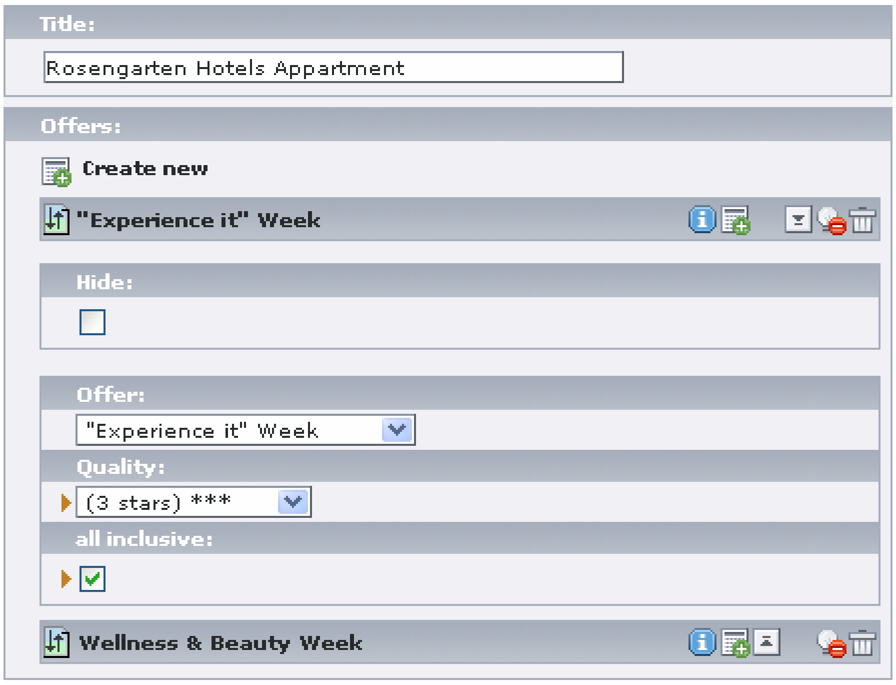


Figure 16: Edit attributes for each relation

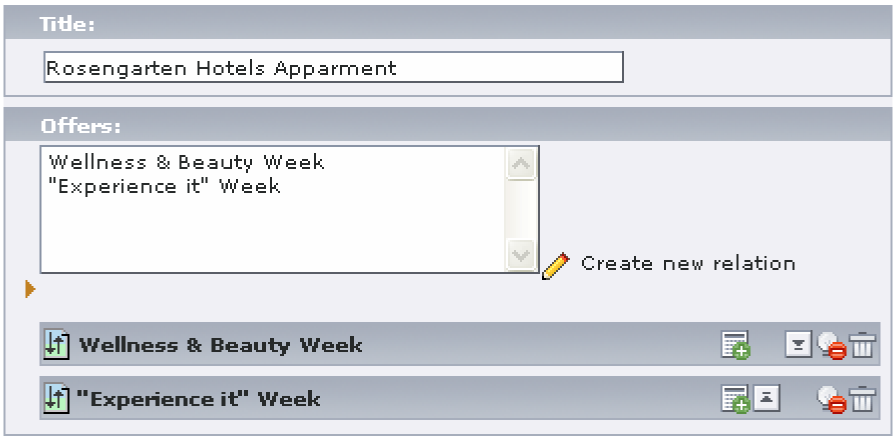
3.3.4. Using a record selector

Record selectors support the editor to create relations to a given set of elements. A selector could be a simple selector-box or an element browser. The selector-box implements the TCA type "select" and shows records which are globally available. The element browser uses the TCA type "group" with the subtype "db". It opens a new pop-up window which shows the page structure and enables to select records from the accordant page.

```
$TCA["tx_irretutorial_mnasym_hotel"] = Array (
    ...
    "columns" => Array (
        ...
        "offers" => Array (
            "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.offers",
            "config" => Array (
                "type" => "inline",
                "foreign_table" => "tx_irretutorial_mnasym_hotel_offer_rel",
                "foreign_field" => "hotelid",
                "foreign_selector" => "offerid",
                "foreign_sortby" => "hotelsort",
                "foreign_label" => "offerid",
14:              "maxitems" => 10,
15:           )
16:        ),
17:     ),
18: );
19: $TCA["tx_irretutorial_mnasym_hotel_offer_rel"] = Array (
20:     "columns" => Array (
21:         "hotelid" => Array (
22:             "label" => "LLL:EXT:[locallang-path]:tx_irretutorial_hotel_offer_rel.hotelid",
23:             "config" => Array (
24:                 "type" => "select",
25:                 "foreign_table" => "tx_irretutorial_mnasym_hotel",
26:                 "maxitems" => 1,
27:             )
28:         ),
29:         "offerid" => Array (
30:             "label" => "LLL:EXT:[locallang-path]:tx_irretutorial_hotel_offer_rel.offerid",
31:             "config" => Array (
32:                 "type" => "select",
33:                 "foreign_table" => "tx_irretutorial_mnasym_offer",
34:                 "maxitems" => 1,
35:             )
36:         ),
37:     ),
38: );
```

Figure 17: TCA for a record selector

The most significant change is the inclusion of the parameter “foreign_selector” (lines 11 and 29 in code). In this case the field “offerid” is part of the intermediate table which was defined to be the “foreign_table” (line 9). Thus, the record-selector will show available entries of the “offer” table (line 33).



Title:
Rosengarten Hotels Apparment

Offers:
Wellness & Beauty Week
"Experience it" Week

Create new relation

Wellness & Beauty Week
"Experience it" Week

Figure 18: Using a record selector to create relations

3.3.5. Define uniqueness on relations

Relations between entities should possibly be unique. Concerning our case-study it doesn't make any sense to build a relation from a hotel to the same offer more than once. In this case it is demanded to define uniqueness. The value of the field that is used to be unique can therefore only appear once as relation. If the editor still tries to create multiple relationships to the same child record he will be notified.

```
1:  $TCA["tx_irretutorial_mnasym_hotel"] = Array (
2:      ...
3:      "columns" => Array (
4:          ...
5:          "offers" => Array (
6:              "label" => "LLL:EXT:irre_tutorial/locallang_db.xml:tx_irretutorial_hotel.offers",
7:              "config" => Array (
8:                  "type" => "inline",
9:                  "foreign_table" => "tx_irretutorial_mnasym_hotel_offer_rel",
10:                 "foreign_field" => "hotelid",
11:                 "foreign_selector" => "offerid",
12:                 "foreign_unique" => "offerid",
13:                 "foreign_sortby" => "hotelsort",
14:                 "foreign_label" => "offerid",
15:                 "maxitems" => 10,
16:             )
17:         ),
18:     ),
19: );
```

Figure 19: TCA for defining uniqueness handling

The "foreign_unique" property (line 12 in code) points to the same field on the intermediate table as the "foreign_selector". But it's also possible to use uniqueness handling without any selector.

3.4. Appearance settings for TCEforms

Appearance configuration allows to define how relational sections should be shown and behave in the back-end view.

3.4.1. Define visibility of child forms

The boolean keys “collapseAll” and “expandSingle” allow to define how the input forms of children are displayed to the editor.

If the collapseAll property is enabled, all child records are represented by their header-bar, showing an icon, the title and the control-items. On clicking the icon or title, the accordant element will expand and exhibit its secrets. This is a good way to save space on the screen and offer a quick overview for child records on the next level.

The expandSingle behaviour ties in with the previous, but takes care that only one child record in the same level is expanded. So, if someone edits data in any element, clicking another one of the same type, results in collapsing the current and expanding the next.

```
$TCA["tx_irretutorial_1nff_hotel"] = Array (
    ...
    "columns" => Array (
        "offers" => Array (
            "config" => Array (
                "type" => "inline",
                ...
                “appearance” => Array (
                    “collapseAll” => 1,
                    “expandSingle” => 1,
                ),
            )
        ),
    ),
);
```

Figure 20: TCA for setting visibility of child records (appearance)

3.4.2. Define positions and style of new record link

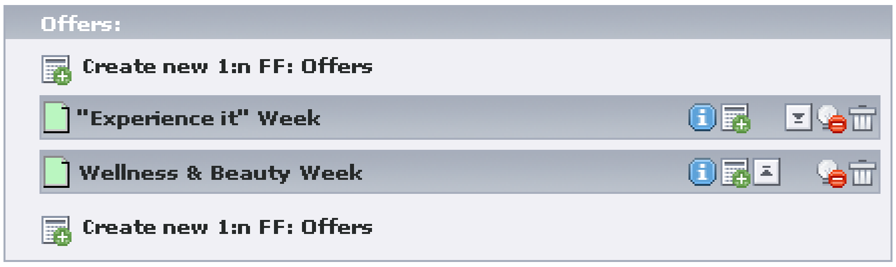
Another appearance property "newRecordLinkPosition" allows to define the position of the link to create new child records. Regarding to the children themselves the link can appear at the top or bottom, at both positions or not at all.

The boolean property "newRecordLinkAddTitle" appends the name of the child table at the new record link. This helps to determine of which type new child records will be created.

```
1:  $TCA["tx_irretutorial_1nff_hotel"] = Array (
2:      "columns" => Array (
3:          "offers" => Array (
4:              "config" => Array (
5:                  "type" => "inline",
6:                  ...
7:                  "appearance" => Array (
8:                      "newRecordLinkAddTitle" => 1,
9:                      "newRecordLinkPosition" => "both",
10:                 ),
11:             )
12:         ),
13:     ),
14: );
```

Figure 21: TCA for changing appearance of new record links

Both keys (lines 8-9 in code) define the appearance during editing a parent record. "newRecordLinkAddTitle" and "newRecordLinkPosition" can be used independent to each other.


Offers:

Create new 1:n FF: Offers

"Experience it" Week

Wellness & Beauty Week

Create new 1:n FF: Offers

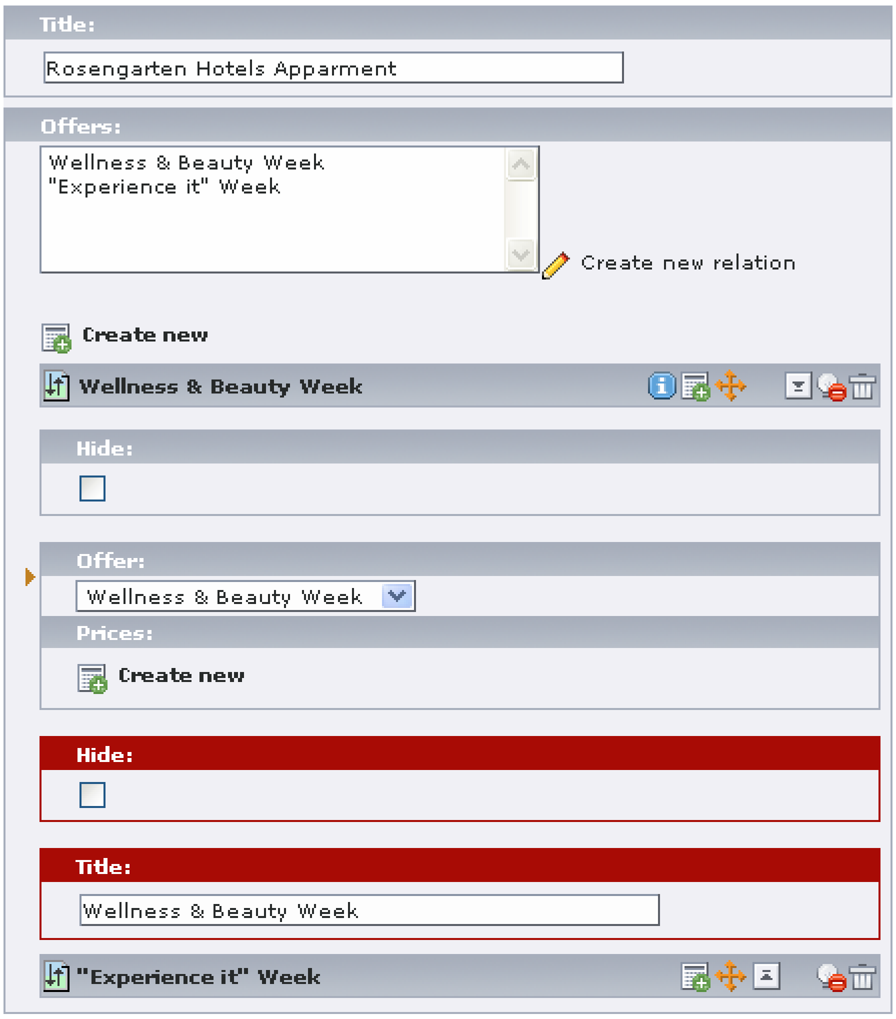
Figure 22: Changing appearance of new record links

3.4.3. Use combined mode for complex relational editing

In situations where an intermediate table is used normally, it is only possible to select a child record on the parent side what also narrows the editing process to the parent record only.

The combined mode offers a possibility to manipulate both sides of a relationship directly. If a field on the child table is defined by the property "foreign_selector" and the appearance "useCombination" is enabled the whole child record can be rendered and edited in the back-end view.

Manipulating the displayed child record affects it directly and not just a copy. Thus, the block of the child record is shown with a red border.


Title: Rosengarten Hotels Apparment

Offers:
Wellness & Beauty Week
"Experience it" Week

Create new relation

Create new

Wellness & Beauty Week

Hide: ☐

Offer: Wellness & Beauty Week

Prices:
Create new

Hide: ☐

Title: Wellness & Beauty Week

"Experience it" Week

Figure 23: Use combined editing mode

```
1:  $TCA["tx_irretutorial_mnasym_hotel"] = Array (
2:      ...
3:      "columns" => Array (
4:          "offers" => Array (
5:              "config" => Array (
6:                  "type" => "inline",
7:                  "foreign_selector" => "offerid",
8:                  ...
9:                  "appearance" => Array (
10:                     "useCombination" => 1,
11:                 ),
12:             )
13:         ),
14:     ),
15: );
```

Figure 24: TCA for enabling the combined editing mode

The required configuration properties to enable the combined mode are the “foreign_selector” (line 7 of code) and the appearance setting “useCombination” (line 10). The child records to be displayed combined to the parent are defined by the field on the intermediate table the “foreign_selector” is set to.

3.4.4. Use drag'n'drop functionality to speed up sorting

Imagine you have a list of many records that shall be sorted manually. It could take some time to put the first item to the end of that list by clicking the move-down-button for each step.

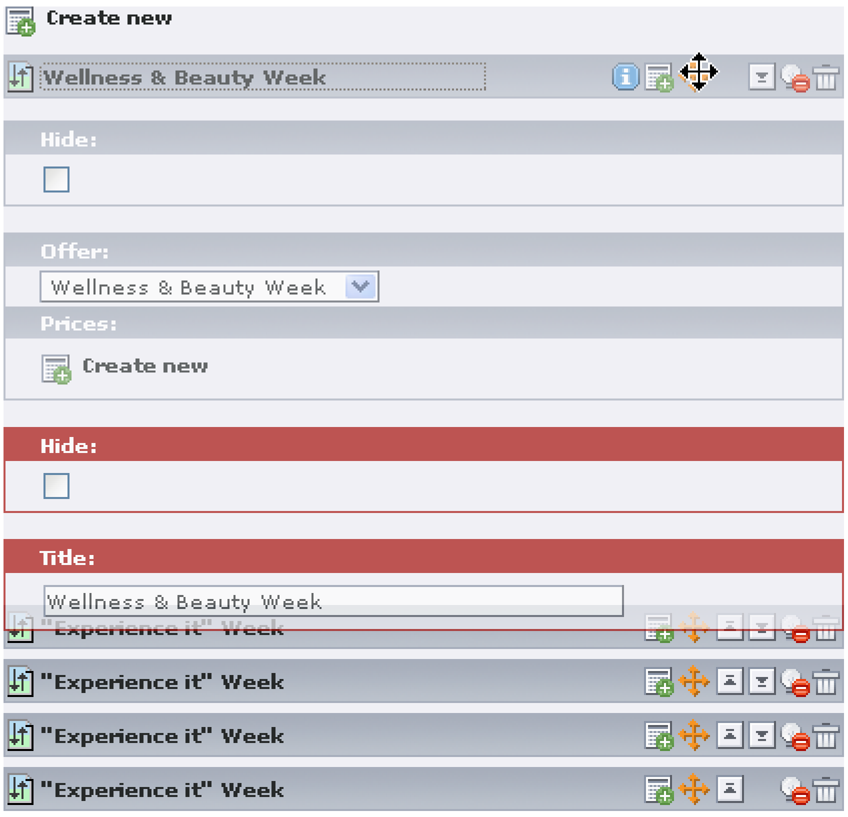
To speed this case up the script.aculo.us framework is used to offer drag'n'drop sorting. If a record should be moved from the top to the bottom of a list, it just has to be dragged at the beginning and dropped at the end. This feature is enabled by the appearance property “useSortable”. “Sortable” is deduced of the JavaScript object of script.aculo.us with the same name.

Of course it only works if there is at least a sorting column defined in TCA for that table which stores the information. Thus, there has to be at least a “sortby” in the ctrl-section, “foreign_sortby” or “symmetric_sortby” in the field configuration.

```
1:  $TCA["tx_irretutorial_1nff_hotel"] = Array (
2:      ...
3:      "columns" => Array (
4:          "offers" => Array (
5:              "config" => Array (
6:                  "type" => "inline",
7:                  ...
8:                  "appearance" => Array (
9:                      "useSortable" => 1,
10:                 ),
11:             )
12:         ),
13:     ),
14: );
```

Figure 25: TCA for enabling drag'n'drop sorting by script.aculo.us

Drag'n'drop sorting of child records is enabled by by the appearance property "useSortable" (line 9 of code). Thus, the script.aculo.us feature "Sortable" will be enabled.


Create new

Wellness & Beauty Week

Hide: ☐

Offer: Wellness & Beauty Week

Prices:

Create new

Hide: ☐

Title: Wellness & Beauty Week

"Experience it" Week

"Experience it" Week

"Experience it" Week

"Experience it" Week

Figure 26: Using drag'n'drop sorting

The child records can be sorted by clicking the orange "move" icon and dragging the selected block.

3.5. Use Page TSconfig to override TCA settings

Page TSconfig allows to override TCA field configuration since TYPO3 4.1. This offers a flexible opportunity to reuse tables and TCA definitions but adapt it to the visual demands. The "Page TSconfig" can be set and modified in the "page properties".

```
TCEFORM.<table>.<field>.config {
  <property1> = <key1>
  <property2>.<subProperty> = <key2>
}
```

Figure 27: General Page TSconfig to override TCA settings

```
$TCA[<table>]['columns'][<field>]['config'][<property1>] = <key1>;
$TCA[<table>]['columns'][<field>]['config'][<property1>][<subProperty>] = <key2>;
```

Figure 28: General effect in TCA due to overriding by Page TSconfig

By default it is impossible to override all TCA properties. Thus, trying to change the type from "inline" to "input" has no effect. For the TCA type "inline" the allowed properties to be overridden are appearance, foreign_label, foreign_selector, foreign_unique, maxitems, minitems, size, autoSizeMax and symmetric_label. The complete whitelist can be found in doc_core_tsconfig, section "Page TSconfig". The evaluation of allowed properties is done in TCEforms. Thus, extension developers with special demands to this issue can influence and extend the whitelist by modifying the array t3lib_TCEforms::$allowOverrideMatrix.

```
TCEFORM.tx_irretutorial_mnasym_hotel.offers.config {
  foreign_selector = offerid
  size = 5
}
TCEFORM.tx_irretutorial_mnasym_offer.hotels.config {
  foreign_selector = hotelid
  size = 5
}
```

Figure 29: Page TSconfig to override TCA configuration

The example shown above enables the assignment of record selectors for the entities hotel and offer. This method is used in EXT "irre_tutorial" on the pages "m:n asymmetric selector" and "m:n asymmetric combo".

4. Implementation

This chapter shows how Inline Relational Record Editing interacts with existing modules. The intention is to get a quick overview of the concept and the behaviour of the system for each part of the TYPO3 Core.

4.1. TCEforms

Since Inline Relational Record Editing is a method to dynamically create and manipulate data inside a given multidimensional structure the most action happens in the form renderer “TCEforms”.

The implementation is kept in a single class t3lib_TCEforms_inline which is automatically instantiated by the constructor of t3lib_TCEforms.

4.1.1. Namespace hierarchy

IRRE handles parents, children, grandchildren, grandchildren of children ad infinitum. It is necessary to identify each of these elements in the whole structure and to know which branch belongs to which parent. Therefore a new naming scheme was created. The names are set as CSS identifier to each involved DOM object (e.g. <div id=”<irre-ident>”>).

```
<irre-ident>      ::=   'data[' <pid> ']' <level-ident> { <level-ident> }
                        [ <level-role> | <partial-ident> ]
<level-ident>     ::=   '[' <tablename> '][' <uid> '][' <field> ']'
<level-role>      ::=   '_records'
<partial-ident>   ::=   '[' <tablename> '][' <uid> ']' <partial-role>
<partial-role>    ::=   '_' ( 'div' | 'header' | 'label' | 'disabled' | 'fields' )
```

Figure 30: Backus-Naur form for object identifiers used by IRRE

A complete level in the structure is always described by a triple of tablename, record UID and fieldname. Partially available information (e.g. just tablename and record UID) describe a child record.

We come back to the case-study of relating hotels to offers and offers to prices. The following identifiers will be used for this scenario:

Identifier string	**Denotation**
data[2][hotel][4][offers]	a section containing generally all child related data
data[2][hotel][4][offers]_records	a section containing all child records
data[2][hotel][4][offers][offer][6]_div	a section containing generally all data of exact one child (here: one offer)
data[2][hotel][4][offers][offer][6]_fields	a section containing all form fields rendered by getMainFields() in TCEforms
data[2][hotel][4][offers][offer][6][prices][price][8]_div	a section containing generally all data of exact one child (here: one price)

Figure 31: Examples of some object identifier

The structural scenario of data objects can be rebuild by a complete identifier string as shown above. Thus, every involved record could be fetched from storage as well as its TCA and TSconfig configurations. Since some actions are only executed in JavaScript at runtime in the browser this improves handling the whole structure.

4.1.2. Edit a parent record

Imagine a back-end user starts to edit a parent record. Creating a new parent record resembles editing an existing one – however the UID is a string with the token “NEW” prepended instead of an integer. The record will be fetched from storage using t3lib_transferdata and finally the forms are rendered in t3lib_TCEforms::getMainFields(). Each field type, such as input, text or inline has own functionalities to render the information.

If records implementing the Inline Relational Record Editing are involved the function t3lib_TCEforms_inline::getSingleField_typeInline() is called. This method fetches all related child records using t3lib_transferdata and handles the internal identifiers of the namespace hierarchy. The rendering of the fields of a child record is again done by t3lib_TCEforms::getMainFields(). Thus, the possibility to recursively execute the parental rendering function is integrated. This offers the opportunity to have complex multidimensional data structures with Inline Relational Record Editing.

There are two methods in t3lib_TCEforms_inline – pushStructure() and popStructure(). Both are used to build up a stack of levels for records using IRRE. If the systems walks down the structural tree, pushStructure() puts the current relative parent element on top of the stack. If the level has been completely processed, popStructure() removes the element from the top of the stack. Thus, it allows recursions and every level knows about its parent and all grandparents.

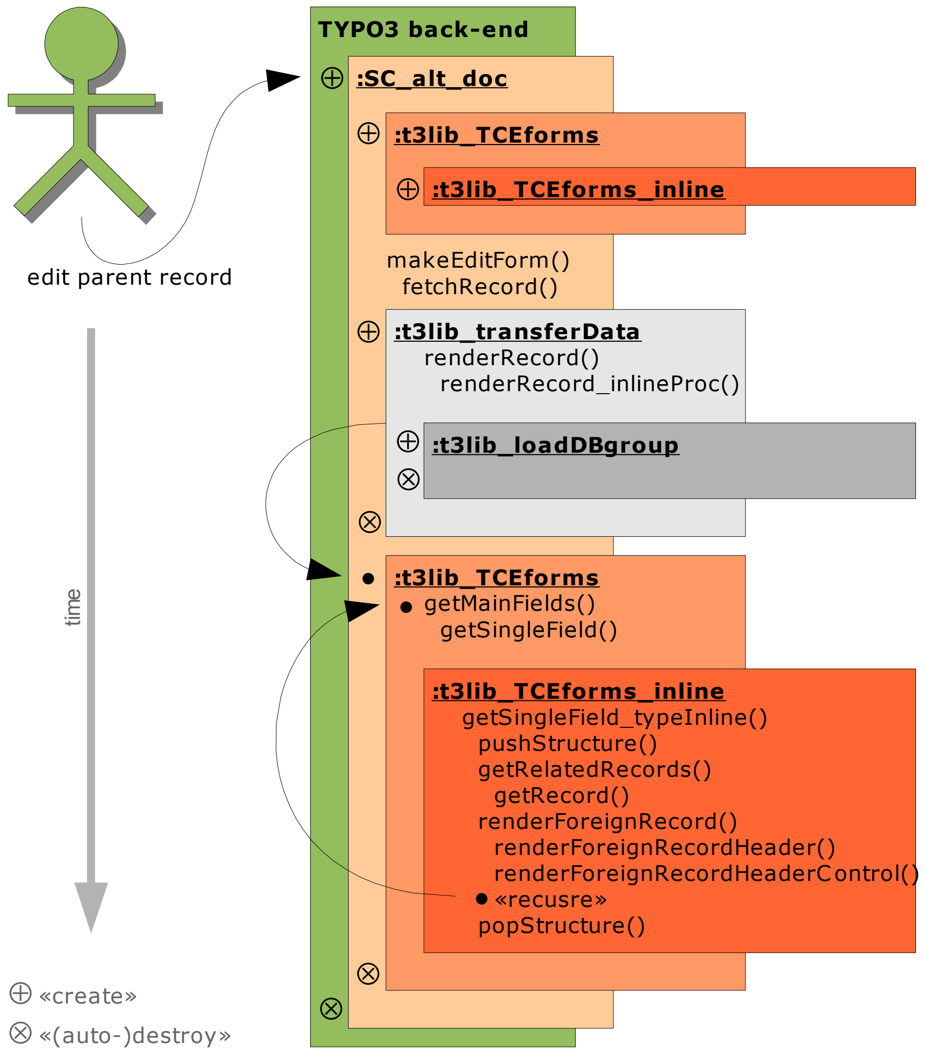


Figure 32: Sequence overview diagram: Edit a parent record

4.1.3. Create new child records

Let's assume the forms of a parent record are completely rendered and loaded. The back-end editor now wants to dynamically create new child records by clicking on the "Create new record" link. After this event has been triggered a sequence of actions follows:

The JavaScript functions of IRRE in t3lib/jsfunc.inline.js create a new AJAX request to the PHP script entry handler typo3/alt_doc_ajax.php and passes two arguments. The first one is the string "createNewRecord" and just tells which action should be performed. The second argument contains the object identifier as described in Figure 31 above.

Now the processing and rendering is delegated to the TYPO3 Core. SC_alt_doc_ajax checks for security purposes if the first argument passed by the AJAX call is allowed. If so the corresponding method in t3lib_TCEforms_inline (in this case "createNewRecord(...)") will be called with the remaining arguments delivered by the AJAX request.

t3lib_TCEforms_inline::createNewRecord reproduces the structural level stack by parsing the object identifier. Also the TCA and TSconfig configurations for the affected tables and fields are loaded. Afterwards the workflow is similar to the previous section. A new blank record is virtually being created using t3lib_transferdata and t3lib_loadDBgroup. The forms again are rendered by calling t3lib_TCEforms::getMainFields().

The results are pushed into an associative array of two keys. The first key "data" is filled with the HTML of the new child record. The second key "scriptCall" is an indexed array which stores several JavaScript actions that should be performed in the browser on the client-side. For example, there is anyway a call that integrates the HTML data into the DOM structure. If input fields are required to be filled (TCA field configuration "required"), this information is also sent using an additional "scriptCall".

Finally, if "data" and "scriptCall" are completely set the mentioned array is converted to JSON and returned to the browser. A client-side JavaScript processes the “scriptCall” section of the response and performs the requested actions.

```
{   "data":"<div id=\"<irre-identifier>\">[HTML data]<\/div>",
    "scriptCall":[
        "inline.domAddNewRecord('bottom','<irre-identifier>','<irre-identifier>',json.data);",
        "inline.memorizeAddRecord(...);",
        "inline.addToDataArray({\"map\":{...},\"config\":{...});",
        "typo3form.fieldSet(...,'required','',0,'');",
        "Element.scrollTo('<irre-identifier>_div');",
        "inline.fadeOutFadeIn('<irre-identifier>_div');"
] }
```

Figure 33: Example of JSON output upon AJAX call

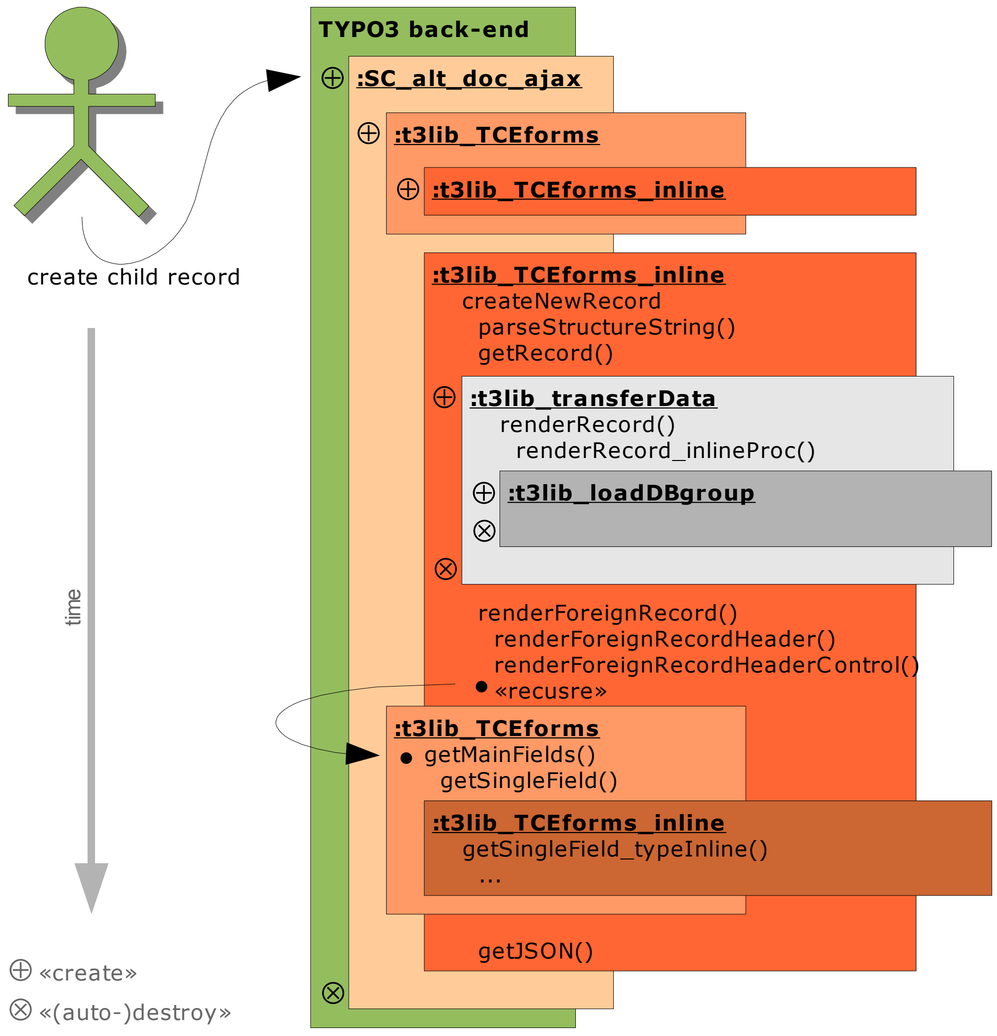


Figure 34: Sequence overview diagramm: Create a child record

4.2. TCEmain

TCEmain takes care about several actions that could be done with data structures. These actions are: create, manipulate, copy, move and delete. Due to inline related records build dependencies between parents and children this also has to be considered in this case. That's why TCEmain was changed to recursively descend in a structure tree and perform the same actions as requested for the parent node.

4.2.1. Saving data after editing

When the back-end user clicks on the save button, all data of the form fields are sent to SC_alt_doc as multidimensional array. The first dimension holds the name of the tabel, the second the UID of the record and the third one the field to write data in. This data array is forwarded to t3lib_TCEmain and processed by the method process_datamap().

Since the parent record could be new and therefore without a proper UID child records could not really build a relation to their parent. Therefore a remap stack was integrated which performs a post-processing of new records and updates all affected UIDs to their new proper integer values. The remap stack will be extended in t3lib_TCEmain::checkValue_inline and t3lib_TCEmain::checkValue_group_select. These are also the locations in source code where relational data is handled.

If intermediate tables are affected a special handling of relational records is necessary. It works similar to methods used by MM relations but is improved to use regular TCA tables and to support bidirectional asymmetric and symmetric relations.

The action for m:n relations happens in t3lib_loadDBgroup. For reading relations the method readForeignField() will be used. Writing relational information to the persistent storage happens in writeForeignField().

4.2.2. Copy inline related records

When a record is copied, normally just this one record has to be duplicated. If records that implement IRRE are affected the dependencies have to be duplicated as well. Thus, if a parent record is copied all direct related 1:n child records will be copied as well. If an intermediate table is used only the relational information is affected and not the semantic related child record itself.

After all records have been duplicated the relational information has to be updated. So the pointer fields defined by "foreign_field" have to be set to the new UID of the accordant parent record. This is achieved by using a remap function similar to saving data structures.

4.2.3. Move inline related records

When a parent record is moved, also the direct affected child records will get moved to the new page. But in this case only the field storing the PID of each affected record will be updated. The relational information is untouched.

Sorting on the same page via Web>List module also uses the moving functionalities of TYPO3. The passed value that normally indicates the PID a record should be moved to is used different here:

If the value is a positive integer and equals the current PID, the record is moved to the first place in the sorting order. If the value is a negative integer the record will be moved after the position of the record with the absolute UID of the passed value. But this is standard behaviour of TYPO3 and wasn't changed by Inline Relational Record Editing at all.

4.2.4. Delete inline related records

If a parent record shall be deleted the behaviour resembles copying of records – all direct affected child records will be deleted as well. If an intermediate table is used only the relational information is removed and not the semantic related child record itself.

Example:

Imagine hotels and offers are related using an intermediate table (see the "Bidirectional asymmetric m:n relations" in the compendium with table prefix "tx_irretutorial_mnasym"). Now a hotel is going to be deleted. Thus, all records of the intermediate table ("..._hotel_offer_rel") which were related to that hotel are removed. The offers stay completely untouched.

5. Quality Assurance

TYPO3 is known to be extremely stable. This requires a few rules to assure not to loose reliability by fixing bugs or implementing new features.

Misconceptions could possibly influence the system in a negative way. That's why every change to the core of TYPO3 has to be reviewed. The review team consists of at least two members of the TYPO3 core development team.

There is a strict coding guideline for every developer that must be followed. Furthermore it could happen, that some pieces of code were not written clearly and are used redundantly or manipulate performance in a negative way. In the worst case vulnerabilities could allow remote command execution or injection of harmful data. These are the basic conditions of a review process that shall avoid serious mistakes.

If the review of new code is done and evaluated as suitable a patch file containing the code of all changes and extensions to current version is announced on the core development mailing list. So other developers have the possibility to get in touch with the changes and approve or decline them.

If nobody objects the changes finally are send (“committed”) to a version control system like SVN or CVS. At that time the enhancements are obtainable to everyone.

Inline Relational Record Editing was reviewed during a weekend mid of November 2006 by TYPO3 Core Team members [8] Ingmar Schlecht and Sebastian Kurfürst at the University of Applied Sciences Hof. Some days after this meeting a patch file with all changes was announced on the mentioned mailing list. The first release of Inline Relational Record Editing was published with TYPO3 4.1-beta1.

6. Perspective and further Development

6.1. Integration in TYPO3 Core

6.1.1. Integration in TYPO3 Core 4.1

Inline Relational Record Editing was initially developed as a proof-of-concept extension in September 2006. This extension is depreciated but still available in the TYPO3 Extension Repository as "Dynamic Backend Editing (dynbeedit)" [9].

Inline Relational Record Editing was developed from scratch in October 2006. The main focus was to create a version that could be integrated into the TYPO3 core package later.

TYPO3 4.1 was pre-released with three beta versions and two release candidates from 24th November 2006 to 21st February 2007. All builds contained the new IRRE feature and were improved from version to version. The final release of TYPO3 4.1 was scheduled to 6th March 2007.

6.1.2. Integration in TYPO3 Core 5.0

TYPO3 5.0 [10] is a ground breaking change in comparison with all versions and branches before. It's not possible to give a definitive guide, how Inline Relational Record Editing could be integrated into prospective TYPO3. But Web 2.0 or better Web x.0 discussions will proceed. This diploma thesis shows how it is currently implemented and allows to transport the concept. It is also imaginable that the whole editing process is covered by AJAX or any other futuristic functionality. So it could feel like a desktop application but is still a flexible and cyclopaedic web application framework.

When the base architecture of TYPO3 5.0 is almost finished, the time comes to adapt the 4.x concept of IRRE.

6.2. Development for TYPO3 4.x

There are some topics in scope of Inline Relational Record Editing concerning the TYPO3 core that could also be called "missing features". The implementation of this features will be done during the further development for the TYPO3 versions 4.2 and 4.3

6.2.1. Adjustments in TCEforms

Some adjustments concerning only TCEforms and thus usability are clipboard functions and inline localization possibilities.

The clipboard actions cut, copy and paste offer an easy way to handle data. This should also be part of Inline Relational Record Editing in future TYPO3 versions.

The localization virtually extends the structure of each record. Translations to other languages of a default record (e.g. in English) could be seen as its child records. Translations are always in context of a 1:n composition and depend fully on their parent. This scenario fits very good into the concept of Inline Relational Record Editing.

6.2.2. FlexForms

FlexForms offer the possibility to handle structured data using a XML tree. Actually this is an antagonism because Inline Relational Record Editing offers the feature to hold data on a normalized data structure instead of pushing everything into a single XML string.

But to be consequent, it is required that all TCA types are also available for FlexForms. Thus, Inline Relational Record Editing will integrate the missing FlexForms feature not later than by the release of TYPO3 4.2.

6.2.3. Workspaces & Versioning

Workspaces and versioning have the ability to create different states of a set of data at a given time. It could also be seen as a snapshot of records. Currently workspaces only support non-relational data.

If a parent record is versionized all affected child records also have to be versionzed and this is currently not possible. TYPO3 Core Team member Dmitry Dulepov is currently working on a solution to allow versioning for the whole page. Thus, the snapshot inherits all records on a given page instead of only affected ones. Pages can be versionized by default.

6.2.4. Import & export of data structures

The system extension tx_impexp allows to export a page or data structure to a file and import it again at another TYPO3 installation. This module uses the sys_refindex table which keeps information about relationships between elements. If structures implementing the IRRE feature shall be exported this could lead to an endless recursion.

Furthermore the record a relation is pointing to could exist on a different page than the one to be exported. This cross-linking causes a conceptual problem, because it's only requested to export the selected page and nothing else. If the structure is imported again it might happen that this produces redundancies or duplicate relations.

6.3. Implementation of IRRE in extensions

Furthermore it might make sense to integrate Inline Relational Record Editing to some existing extensions. These extensions are mostly focused on providing techniques for handling 1:n or m:n related data.

6.3.1. Integration in Party Information Framework

Party is the umbrella term for persons and organizations (e.g. companies, associations, teams, etc.). Thus, the "Party Information Framework" [11] by David Brühlmeier offers the relational management of all data concerning parties. It implements the OASIS [12] definitions to describe and handle information for addresses, customers and relationships.

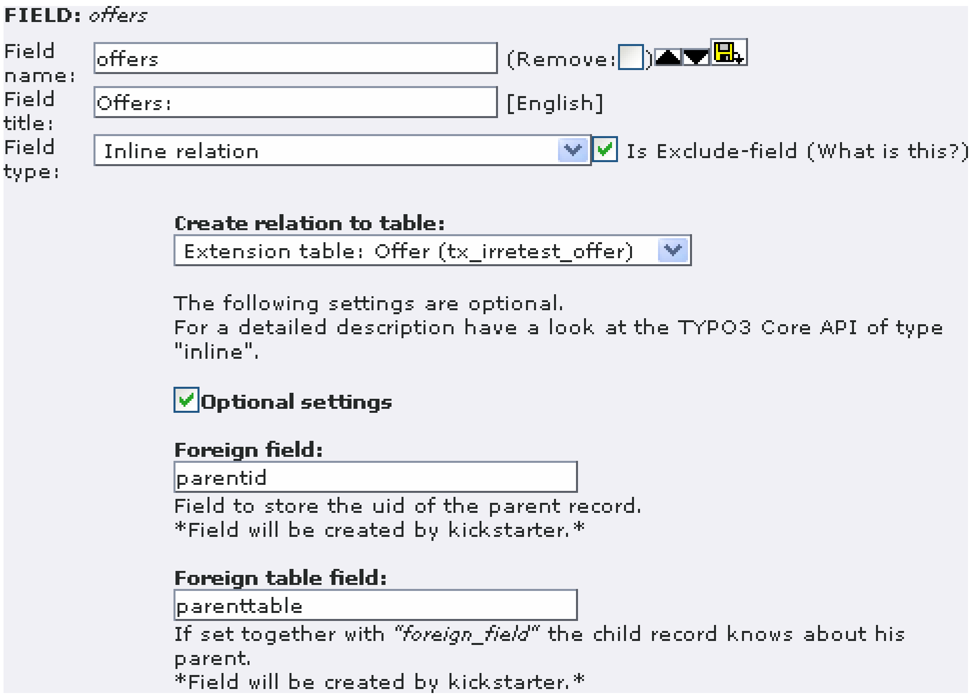
Simple examples of use are the development of a family tree or a reproduction of a business network like OpenBC/XING [13].

Inline Relational Record Editing offers the most techniques that are required for the Party Information Framework, e.g. bidirectional asymmetric and symmetric relations and the possibility to reuse entities using an intermediate table.

Party Information Framework was formerly also know as Partner Framework. This extension is available in the TER as "tx_partner" [14].

6.3.2. Integration in tx_kickstarter

The "Extension Kickstarter" [15] by Ingo Renner offers a possibility to create table structures and thus an initial framework for new extensions using a form wizard. Peter Foerger has volunteered to extend tx_kickstarter to support Inline Relational Record Editing. A draft version of his work looks promising and allows simply to implement IRRE by just clicking.


FIELD: *offers*

Field name:	offers	(Remove: ☐)
Field title:	Offers:	[English]
Field type:	Inline relation	✓ Is Exclude-field (What is this?)

Create relation to table:
Extension table: Offer (tx_irretest_offer)

The following settings are optional.
For a detailed description have a look at the TYPO3 Core API of type "inline".

✓ **Optional settings**

Foreign field:
parentid
Field to store the uid of the parent record.
Field will be created by kickstarter.

Foreign table field:
parenttable
If set together with *"foreign_field"* the child record knows about his parent.
Field will be created by kickstarter.

Figure 35: Extract of tx_kickstarter implementing IRRE

7. Synopsis

Inline Relational Record Editing bridges the gap between the repetitive elements capabilities of FlexForms and the old field type of TCEforms implementing relations. This is generally achieved by saving information in nice separated database records with 1:n or m:n relationships what FlexForms stored in just one big XML string.

Inline Relational Record Editing integrates a new TCA type, several new dynamic techniques for handling and storing data into the Core of version TYPO3 4.1. The different TYPO3 releases of the 4.1 development branch have been downloaded about 15.000 times over a period of eleven weeks.

The development of Inline Relational Record Editing and other related new features inside TYPO3 is not yet finished. Members of the TYPO3 community can contribute their ideas and desires by using the bugtracker or the project mailing list and follow the mission “to jointly innovate free software enabling people to communicate”.

CD-ROM

The enclosed CD-ROM contains three directories:

- root directory
    - Contains the GNU General Public License Version 2 (GPL.txt)
- “diploma_thesis”
    - Contains this document as PDF file.
- “publication”
    - Contains an article about Inline Relational Record Editing in German language published in “T3N Magazin für Open Source und TYPO3”, yeebase media solutions GbR, Hannover, DE [16]
- “typo3_ext”
    - Contains the extensions “irre_tutorial” and “irre_hotel”.
- “typo3_src”
    - Contains all development versions of TYPO3 4.1, three beta versions and two release candidates as Unix tarballs as well as zipped archives. The final release can be obtained from the TYPO3 website [1].

Bibliography

[1] http://www.typo3.org/

[2] http://typo3.org/extensions/repository/view/kb_tca_section/0.0.1/

[3] http://typo3.org/extensions/repository/view/dynaflex/1.7.2/

[4] http://www.adaptivepath.com/publications/essays/archives/000385.php

[5] http://www.prototypejs.org/

[6] http://www.json.org/

[7] http://typo3.org/extensions/repository/view/irre_tutorial/

[8] http://typo3.org/teams/core/members/

[9] http://typo3.org/extensions/repository/view/dynbeedit/0.0.3/

[10] http://5-0.dev.typo3.org/

[11] http://wiki.typo3.org/index.php/Party_Information_Framework

[12] http://www.oasis-open.org/

[13] http://www.xing.com/

[14] http://typo3.org/extensions/repository/view/partner/

[15] http://typo3.org/extensions/repository/view/kickstarter/0.3.8/

[16] Oliver Hader: "Einfach irre!", T3N Magazin für Open Source und TYPO3 (Nr. 7), 2007

Declaration of Authorship

I certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, done with no other than the mentioned utilities and has not been published or submitted, either in part or whole, for a degree at this or any other University.

Oliver Hader

Hof, 26th February 2007