Protée sarl
31 rue Sainte-Marthe
31000 TOULOUSE

# zen_Nucleus

# RECORDS – Programmer's guide

| Release | Version | Date | Rédacteur | Commentaires |
| --- | --- | --- | --- | --- |
| v3.0.00 | | 13 octobre 2025 | OG | First version |
| | | | | |

Sēmippān BANKS record

# SUMMARY

**SUMMARY** **2**

**Introduction** **3**

**Records 3**

Architecture 3

Description 4

Record « Form object » 5

How to manage « orda » entities 6

Components 7

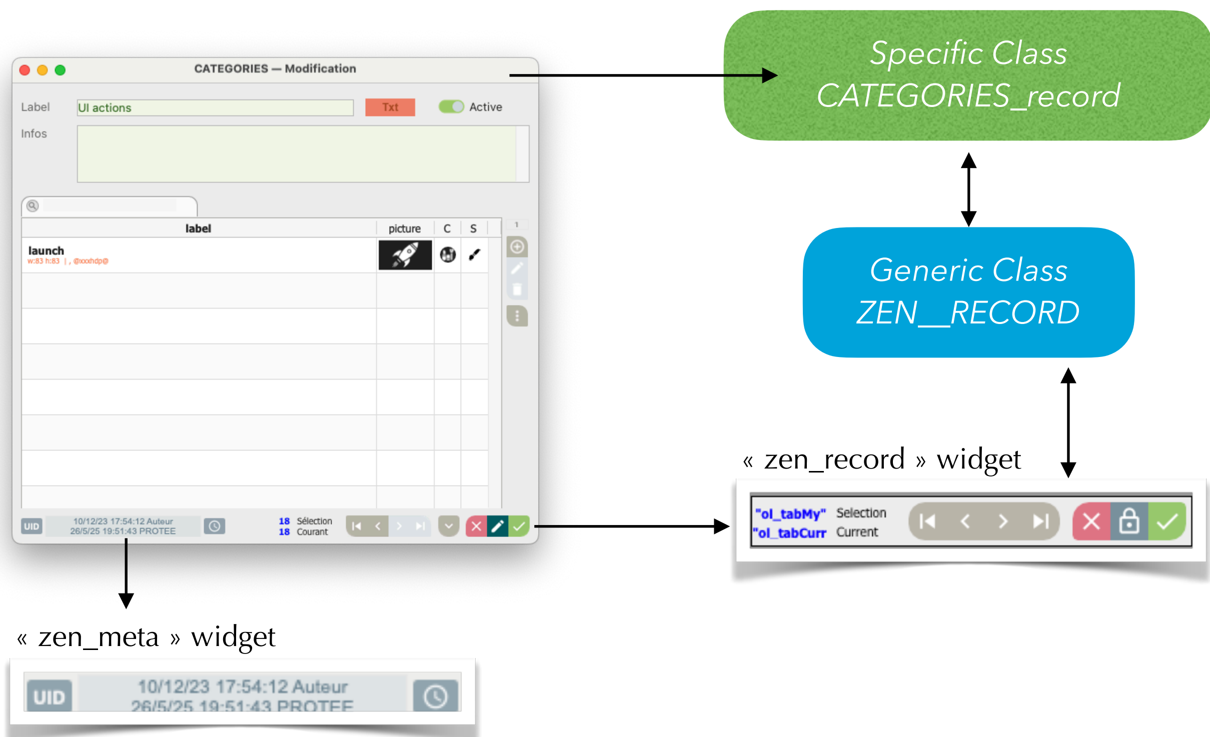
# INTRODUCTION

This documentation is intended to explain how the record's manager is conceived, how to use it quickly.

# RECORDS

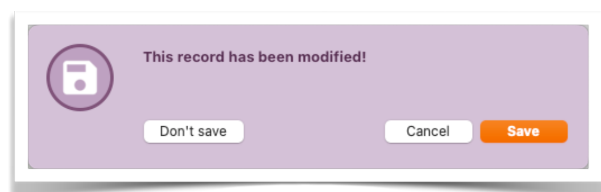
## Architecture

All the records are made using this framework.




The navigation flow always check for the « Touched » attribute, and in case of True, a regular process is proposed to the user: « Don't save », « Cancel », « Save ».

This check is made for the close box, the four navigation's buttons (first, previous, next, last), and the read/write button (locked/unlocked). The « Cancel » and « Accept » buttons always do a deterministic action.



## Description

The <TABLE>_RECORD class is dedicated for the records of one table - ie here table « CATEGORIES ».

The Form method is basic : On Load create an instance of the class, extending the following generic class « ZEN__RECORDS ». The attached class is referenced and accessible with « Form.fc.<member>() ». All other events are going to the dispatcher « Form.fc.form_events() ».

```
var $vL_event_code : Integer
$vL_event_code:=Form event code
Case of
	: ($vL_event_code=On Load)
		Form.fc:=cs.CATEGORIES_record.new()
		
	Else
		Form.fc.form_events()
End case
```

The Form includes the « zen_meta » and « zen_record » widgets that load they generic classes.

The generic class « ZEN__RECORDS » gives all the stuff needed for the record's management. It calls back the <TABLE>_RECORD Class for some functions like:

```
Form.fc.record_save()
Form.fc.record_touched($c4E_record)
Form.fc.record_checkout()
```

When there is no need of specific cases, those calls are sending back to the callers to the associated generic functions. Just remove them or use Super() in them.

Each time record needs to be loaded, the Form's Dialog is quit and the navigation action is made in the the calling method, then goes into the Dialog again. This ensure that all components will have the proper context for all they On Load event, allowing to bind the value to the Entity (Form.c4E) and Properties. Those actions are for now: first, previous, next, last, modify, close box.

So in a specific <TABLE>_RECORD Class manager, you have to deal with:

- the **Class Constructor** to initialise what is common for the record, and the proper initialisation for fields if it is a new record. Call first Super() (with « True » to be in transaction mode).
- Function **record_load_upd**() -> will update the interface based on the record.
- Function **record_touched**($c4E_entity : 4D.Entity)->$is_touched : Boolean
  - To implement your own « is_touched » function if subTables are present, or for a 4DWrite zone etc.
- Function **record_checkout**()->$isOk : Boolean
  - This is where you implement all the checks for mandatory fields or specific coherency to the data before save. If your answer is False, all the record Save process is aborted.
- Function **form_events**() -> this is where you dispatch your form events for your bizz.

## Record « Form object »

The Record's form include precious informations: this is the data context for a proper ORDA mechanism. Here are the properties.

| Property | Type | Description |
|---|---|---|
| **is_write** | Bool | Indicate if the table is writable |
| **is_new** | Bool | Indicate if the record is a new created record |
| **is_dup** | Bool | True if the record is a duplication made by zen |
| **is_editing** | Bool | Indicate the initial write status of the record |
| **is_local** | Bool | Indicate if this table is local to zen_Nucleus |
| **is_transaction** | Bool | Indicate if the record is in transaction |
| **l_pid** | Integer | Process id |
| **t_comingFrom** | Text | Table where Form is opened, empty if coming from a list |
| **t_form** | Text | Name of the Form to use |
| **t_table** | Text | Table name |
| **l_table** | Integer | Table number |
| **t_view** | Text | View of the table |
| **l_winRef** | Integer | Window reference from parent for any action like CALL FORM |
| **l_winRef_record** | Integer | Window reference for the current Form |
| **l_position** | Integer | Position of the record in the ES (1->N) |
| **c4ES** | c4ES | Entity selection of the current record |
| **c4E** | c4E | Entity of the current record. It is calculated once in the process, from c4ES and l_position |
| **_j_form** | Object | Preference properties you can manage you own. Session scope. |
| **j_prefs** | Object | Used by the Move Window class. With « is_moved ». |

So your record's reference is in « c4E » ready for use !

All your objects are Enabled/Enterable regarding the is_modify flag. To remove objects from this automatic action done in **Super.record_load_upd**(), just prepare a collection of object you want out of this process:

```
var $vC_at_objects_nc : Collection
$vC_at_objects_nc:=New collection
$vC_at_objects_nc.push("oT_prefix"; "wog_banner")
This.at_objects_nc:=$vC_at_objects_nc
```

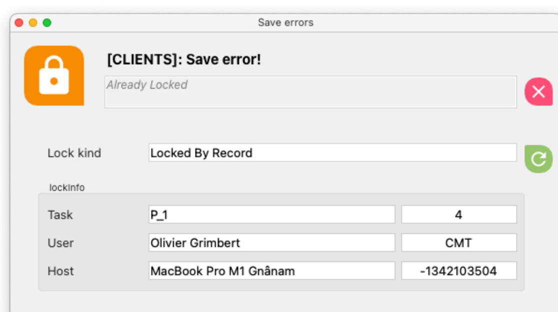
If you want the generic class to manage the enabled state of your buttons (based on is_write), just name them « btn_<yourname> ». If you don't want the enabled state being managed by zen, name them « bt_<yourname>.

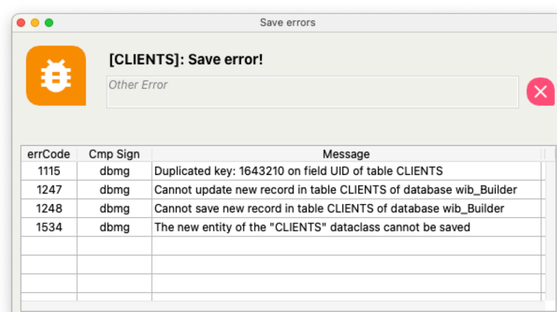
## How to manage « orda » entities

In orda, the entities and errors are managed with .save() command. It is always possible to take a record as there is no question of WRITE state and lock, if you don't ask explicitly for it.

| orda | zen methods | |
|---|---|---|
| .lock() | ***zen_entity_lock*** | `($c4E : 4D.Entity; $vL_mode : Integer; $is_silent : Boolean)->$vJ_status : Object` |
| .new() | ***zen_entity_new*** | `($c4DC : 4D.DataClass)->$c4E_entity : 4D.Entity` |
| .reload() | ***zen_entity_reload*** | `($c4E : 4D.Entity; $is_silent : Boolean)->$vJ_status : Object` |
| .save() | ***zen_entity_save*** | `($c4E : 4D.Entity; $vL_mode : Integer; $is_silent : Boolean)->$vJ_status : Object` |
| .unlock() | ***zen_entity_unlock*** | `($c4E : 4D.Entity)->$is_success : Boolean` |
| | ***zen_entity_errors*** | `($c4E : 4D.Entity; $vJ_status : Object; $is_silent : Boolean)->$is_retry : Boolean` |

Those methods allows you to use orda with a higher level to manage errors.
By default, $vL_mode = dk reload if stamp changed / dk auto merge for .save().
When $is_silent is false, in case of « success = false » and for all kind of « status », a Form opens and displays the orda error to the user.
The ***wib_entity_save*** command includes a retry of 6 times 10 ticks delay. After this, the error is given back to $vJ_status and eventually the error's form is opened.



Those methods manage a field « meta » if any, with dedicated fields in it:

## Components

This is a good habit to have no code in form's objects, and use an event's dispatcher for that.
But for components, this is not possible for 4D reasons.

- First, the On Load is made before the main Form On Load event. You can initialise a binding for all components easily as you are always with the proper context in On Load ie Form.c4E is correct.
- Second, the object events made inside a component are propagated to the host (4D's fault).
- Third, the events generated inside a widget like negative events or k_OnDataChange are not propagated to the main Form events (4D's fault). And there is no information about an event source in case of a subForm.

The goal is to allow the programmer to have a fully encapsulated code inside the components – as we can't avoid to have code in them.

That is why all the component from Protée allows to bind the data source with the use of a pointer, or to an Object/property, or an object field. That means all subsequent changes are reported to the data without any external code.

Obviously if you need some actions based on a data change, you have to call the form class Form.fc.myFunction() to do your stuff.

```
: ($vL_event_code=k_OnDataChange)
    Form.fc._sets_dcox_chgt()
```

Have a look on available class's members in ZEN__WIDGETS, like
.load(), .resize(), .redraw(), .set_widget_name()
.bind_to(), .bind_to_c4E(), .bind_to_c4E_vJ()