

GOALS

Show the capabilities and uses related to the manipulation of the `config_form_fields` table with the intention of customizing the forms displayed in QGIS for the network elements (node, connec, gully, arc). The labels and tooltips of all the widgets can be translated, as well as moving or hiding them if we are interested.

INTRODUCTION

Since improper manipulation can cause major damage to the configuration table, there is a backup function of any table in the database that can be very useful for this process. Its use consists of:

CREATION OF THE BACKUP:

```
SELECT gw_fct_admin_manage_backup ($${"data":{"action":"CREATE",
"backupName":"test6", "table":"config_form_fields"}}$)
```

RECOVER OF THE BACKUP:

```
SELECT gw_fct_admin_manage_backup ($${"data":{"action":"RESTORE", "backupName":"test6",
"newBackupName":"test5", "table":"config_form_fields"}}$)
```

DELETE OF THE BACKUP:

```
SELECT gw_fct_admin_manage_backup ($${"data":{"action":"DELETE",
"backupName":"test4", "table":"config_form_fields" }}$)
```

COMMENTS:

The creation process stores all the information in the `temp_table` table with `fprocesscat_id = 111`

To avoid possible mishandling, a double safety factor strategy is used. In this sense:

- It is mandatory to always put the two keys (backupName and table)
- Before restoring the backup, a backup of the existing data in the table is saved with the name of newBackupName.

DESCRIPTION

To modify **config_form_fields** to customize the forms.

Notes to keep in mind. The primary key of the table is:

formname & formtype & tabname & columnname

The only rows that need to be manipulated to customize the element forms are the ones prefixed in the formname column:

```
ve_node_*      ve_arc_*
ve_connec_*    ve_gully_*
```

Normally, there must be records for all the fields of the elements that we find in the `cat_feature` table (they must also coincide with the *child* views that we have created in our schema).

So, in the `config_form_fields` table, we will have for each *child* view a number of rows that match all the fields that this view has, in order to configure them one by one.

Of the many columns in `config_form_fields`, each and every one of them is described to achieve the desired degree of configuration:

[POSITION]

tabname: To manage the forms of the different elements of the network, the widgets are managed according to the tab where they are located. As we see in the image, there are different tabs, and each one will have the corresponding name: **data**, element, document, etc.

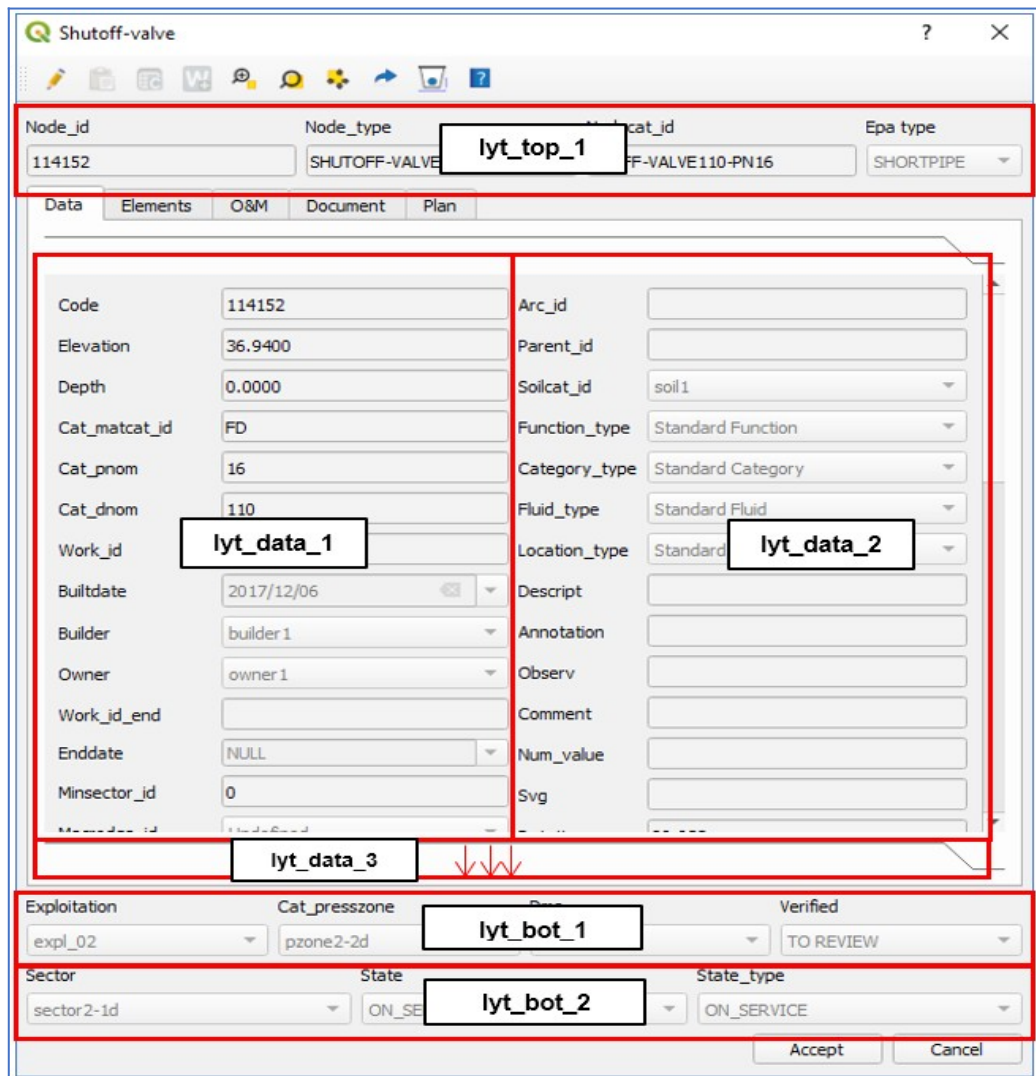
There are two types of tabs, they are differentiated by the layout of the layouts. Most have one or two layouts, but those in the feature_info have all 3 verticals layouts (except the tab data).

When there are values in config_form_fields that refer to a form without tabs, the value for this column will be: **main**.

layoutname: For all the tabs there are three layouts (1,2,3). The name follows the premotechnical rule of `lyt _ tabname _ (1,2,3)`

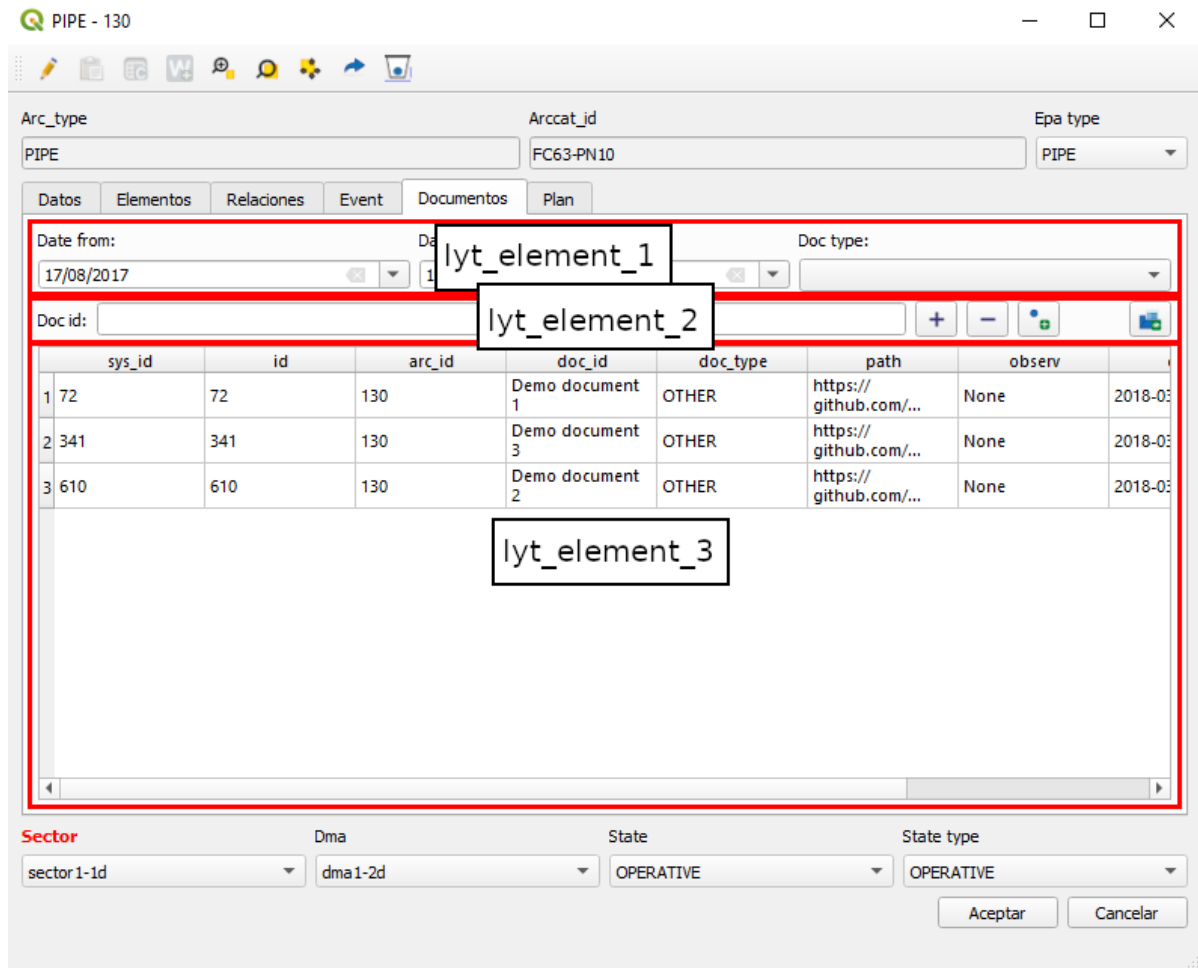
Additionally we have the layouts `lyt_top_1`, `lyt_bot_1` (top row), `lyt_bot_2` (bottom row). Check image.

layoutorder: order of the field within its corresponding layout. They will be sorted in ascending order using the value they have in this field. If two fields are found with the same value of `layoutname` and `layout_order`, obviously they will overlap within the form.



The image shows a software window titled "Shutoff-valve" with a toolbar and a form. The form has several tabs: "Data", "Elements", "O&M", "Document", and "Plan". The "Data" tab is active. The form contains various input fields and dropdown menus. Red boxes and labels highlight specific layout areas:

- lyt_top_1:** The top header area containing fields like "Node_id", "Node_type", "Cat_id", and "Epa type".
- lyt_data_1:** The left data panel containing fields like "Code", "Elevation", "Depth", "Cat_matcat_id", "Cat_pnom", "Cat_dnom", "Work_id", "Builddate", "Builder", "Owner", "Work_id_end", "Enddate", and "Minsector_id".
- lyt_data_2:** The right data panel containing fields like "Arc_id", "Parent_id", "Soilcat_id", "Function_type", "Category_type", "Fluid_type", "Location_type", "Descript", "Annotation", "Observ", "Comment", "Num_value", and "Svg".
- lyt_data_3:** The bottom data panel containing fields like "Exploitation", "Cat_presszone", "Verified", "Sector", "State", and "State_type".
- lyt_bot_1:** The bottom status panel containing fields like "Exploitation", "Cat_presszone", "Verified", "Sector", "State", and "State_type".
- lyt_bot_2:** The bottom action panel containing "Accept" and "Cancel" buttons.



[BASIC CHARACTERISTICS]

Datatype. Type of data. It does not apply to combo-type elements. Possible values are: ["string", "double", "date", "bytea", "boolean", "text", "integer", "numeric"]

Widgettype Type of widget. The possible values are: ["datetime", "label", "nowidget", "text", "image", "typeahead", "button", "check", "combo", "hyperlink", "divider", "list", "spinbox", "hspacer", "tableview"]

typeahead & combo: they are filled using the [VALUE DOMAIN MANAGEMENT] with some limitation for typeahead. The combo can also use the enableWhenParent key to be active only in certain values of parent.

list: The list defined in the field linkedobject is used taken from the table config_form_list

spinbox: The number of decimal places in the key is defined 'spinboxDecimals' of widgetcontrols.

image: the image defined in the field linkedobject is used taken from the table sys_image

button: The front-end function defined in widgetfunction is assigned

link: The element will be in linkable format. Front-end function should be assigned in widgetfunction.

hspacer: The element will be a separator to give space between other widgets.

tableview: The element will be a table with the data defined in the config_form_list table

label: Label of the field in the form and the attribute table. Fully customizable.

PROTOCOL DOCUMENT

P-09

Configuration of form_fields (3.5)

hidden: TRUE / FALSE – displayed / not displayed in form and attribute table

tooltip: text that is displayed if you are above the field label. Fully customizable.

(*) *WE RECOMMEND: in case of label translation, put at the beginning of the tooltip the real name of the field (English) to be able to know it easily*

placeholder: Example value to display when the field is empty.

iseditable: TRUE / FALSE - the field can / cannot be edited in the form and the attribute table.

ismandatory: TRUE / FALSE - in case of TRUE, this field must necessarily have a value.

isparent TRUE / FALSE When a widget is the parent of another, it allows to reload combos of the children that have this widget identified as their parent (dv_parent_id)

isautoupdate TRUE / FALSE It triggers the update of the form without waiting for the user's okay. Valid for fields that need to recalculate things like depths or others: *This option is not possible for typeahead type widgets.*

isfilter TRUE / FALSE When we have a list widget type, it can be filtered by widgets that are in the same tab. These widgets can be anything, but they will have the isfilter=true attribute. For them, the keys 'vdefault' and 'listFilterSign' of widgetcontrols are of special interest.

[MANAGEMENT OF VALUE DOMAINS FOR COMBO & TYPEAHEAD]

In combination with typeahead widgets or combo.

dv_querytext: Query text always with the same id criteria / idval. *If the widgettype is "typeahead", it is mandatory that id & idval for querytext must be the same field*

dv_orderby_id: TRUE / FALSE Allows to order by id instead of idval

dv_isnullvalue: TRUE / FALSE Allows null values in querytext

dv_parent_id: For those widgets that have a parent

dv_querytext_filterc: Additional query text filtering by the value of parent id

[ADVANCED CHARACTERISTICS]

stylesheet: json type field that allows a graphic customization of the label. See FAQs for examples of this field.

widgetcontrols: They allow advanced widget control with the following options:

autoupdateReloadFields – instantly reload other fields in case one is modified. Works in combination with isautoupdate.

```
UPDATE config_form_fields SET widgettype = 'combo' , isreload=true, widgetcontrols =
gw_fct_json_object_set_key(widgetcontrols, 'autoupdateReloadFields' ,["cat_matcat_id",
"cat_dnom", "cat_pnom"]::json) WHERE column_id IN ('arccat_id', 'nodecat_id', 'connecat_id')
```

enableWhenParent – enables a combo only in case the parent field has certain values.

```
UPDATE config_form_fields SET widgetcontrols = gw_fct_json_object_set_key
(widgetcontrols,'enableWhenParent',['1, 2']::json) WHERE column_id IN ('state_type')
```

regexControl – Control of what the user can write through regular expression in widgets type free text

```
UPDATE config_form_fields SET hidden=false, datatype = 'text', widgetcontrols =
gw_fct_json_object_set_key(widgetcontrols,'regexControl','[\d]+:[0-5][0-9]:[0-5][0-9]::text)
WHERE column_id = 'observ'
```

Additional Note: Since the character '\' is system reserved for PostgreSQL, the update must be done with a '\\' so that two appear in the row, so that the syntax stored and with which to work will be [\d]+:[0-5][0-9]:[0-5][0-9]

maxMinValues – establishes a maximum value for the numerical fields in free text widgets

```
UPDATE config_form_fields SET widgetcontrols = gw_fct_json_object_set_key
(widgetcontrols,'maxMinValues',{'min':0.001, "max":100}::json) WHERE column_id = 'descript'
```

setMultiline – establishes the possibility of multi-line fields for writing with enter

spinboxDecimals – establishes a concrete number of decimal places for the spinbox widget (vdef 2)

```
UPDATE config_form_fields SET widgetcontrols = gw_fct_json_object_set_key (widgetcontrols,
'spinboxDecimals', '3') WHERE column_id = 'descript'
```

widgetdim – Dimensions for the widget

vdefault value – Default value of the widget. It makes sense for those widgets that do not belong to the data of a feature, since the default values are defined in those that the user already has set in config_param_user. Of special interest for the filter widgets.

vdefault querytext – Default value of the widget obtained from the query. It makes sense for those widgets that do not belong to the data of a feature, since the default values are defined in those that the user already has set in config_param_user. Of special interest for the filter widgets.

listFilterSign – Sign (LIKE, ILIKE, =, >, <) for the fields filter type. In case of omission, ILIKE will be used for tableview-type lists e= for tab-type lists,

skipSaveValue – If this value is set to true, the changes made in the corresponding widget will not be saved. By default it is not necessary to put anything because it is understood as true

labelPosition:- If this value is defined [top, left, none], the label will occupy the relative position with respect to the widget. By default it is understood as left. If the label field is empty, labelPosition is omitted.

PROTOCOL DOCUMENT

P-09

Configuration of form_fields (3.5)

widgetfunction: The name of the Python function to be executed is defined, and if there are the characteristics of the additional parameters. You can define the file to use with the key "module", by default it is understood as the file core/utlis/tools_backend_calls.py. To use a file other than tools_backend_calls.py it will have to be imported into tools_gw.py.

```
{"functionName": "add_document", "module": "info", "parameters": {"sourcewidget", "targetwidget"}}
```

linkedobject:

For widgettype list: Name of the list located in the config_form_list table to be linked. This table configures the query to be used (querytext) and the client with which it will be called. There are two fields in the table that do not have an associated code at the moment, such as:

listtype: Refers to how the list is displayed: tab (vertical elements for a narrow tab) or in attributetable (tableview elements for a larger width).

listclass: Class of elements shown in the list (icon, icons, gallery type or list).

It is recommended that the lists have the name list_* in the definition of the table where they are created.

For widgettype image: Name of the image located in the sys_image table to be linked. It is recommended that the images have the name img_*

For widgettype [text/check/combo/typeahead]: action (optional) linked to the widget (getcatalog for example) that is available in the dialog, configured in config_form_tabs. It is recommended that the actions have the name action_*

For widgettype button: Name of an icon (optional) to set on the button with the associated image found in the plugin folder icons/backend/20x20. It is recommended that the names of the icons be simple such as numbers.png

FAQS

Any useful queries to make massive replacements?

Replace for **all element types** the **label** of a **specific field**. In this case, the *label* of the *presszonecat_id* field is replaced by Pressure zone for all elements, whether they are *node*, *connec*, *gully* or *arc*.

```
UPDATE config_form_fields SET label='Zona de presión' WHERE columnname = 'presszonecat_id' AND formtype='form_feature' AND formname LIKE 've_%';
```

Hide a **specific field for all element types**. In this case, the *verified* field is hidden for all elements.

```
UPDATE config_form_fields SET hidden=true WHERE columnname = 'verified' AND formtype='form_feature' AND formname LIKE 've_%';
```

Hide a **specific field** only for **node type elements**. In this case, the *code* field is hidden for *node* type elements.

```
UPDATE config_form_fields SET hidden=true WHERE columnname = 'code' AND formtype='form_feature' AND formname LIKE 've_node_%';
```

Hide a **specific field** only for some type of **specific element**. In this case, the *parent_id* field is hidden for the node types: *pump*, *reduction*, and *tank*.

```
UPDATE config_form_fields SET hidden=true WHERE columnname = 'parent_id' AND formtype='form_feature' AND formname IN ('ve_node_pump', 've_node_reduction', 've_node_tank');
```

Can I add a field that I want in the form?

Yes, you can add the field you want ALWAYS AND WHEN IT IS ONLY A QUERY and it is not an additional field (*addfield*). For this you have to:

Modify the *ve_node_** / *ve_arc_** / *ve_connec_** / *ve_gully_** adding all those fields that you want to visualize (only query. Editing is not allowed since the trigger will not work).

The modification can be with *DROP* or *CREATE OR REPLACE*, we recommend the latter method. While for *DROP* we must keep the trigger and redo it, if we do it with *CREATER OR REPLACE* we must add the new fields *MANDATORY* at the end of the view, they cannot be in between.

Add in *config_form_fields* a new row referring to the created field.

EXAMPLE: I want to add the descriptor of the material in the form of *ve_arc_ownpipes*, since in the form and in the corresponding view there is only the id:

I modify the ve_arc_ownpipe by making a JOIN to cat_mat and adding the cat_mat.descript field.

Add in the *config_form_fields* a new row referring to the created field, in this case *descript*. If I use an alias for the name I will have to put the alias.

This is a complex environment. Can you manage backups?

Check the INTRODUCTION of the protocol.

What values can I put in the stylesheet field to modify the colors of the label?

Put color and bold in a field.

- Value in *stylesheet* - {"label":"color:blue; font-weight:bold"}
- Result

Code	1092
Elevation	23.2100

Color in HTML and change letter size.

- Valor en *stylesheet* - {"label":"color:#ff3300; font-size: 15px"}
- Result

Code	1092
Elevation	23.2100

Put color in the background.

- Value in *stylesheet* - {"label":"color:yellow; font-weight:bold; background-color:#9999ff"}
- Result

Code	1092
Elevation	23.2100

REFERENCES

REVIEWS:

Action	User	Date
Created	Albert	11/11/2019
Updated	Xavier T.	13/11/2019
Updated	Albert B.	07/04/2020
Updated	Xavier T.	07/07/2020
Updated	Xavier T.	21/10/2020
Updated	Xavier T.	08/04/2021
Updated	Néstor	09/04/2021
Updated	Albert B.	12/04/2021
Updated	Albert B	03/09/2021
Updated	Sergi Maspi	17/09/2021
Updated	Xavier Torret	01/12/2021
Updated	Albert B	14/12/2023