# ogDevTools

# User's guide

**Component's user data sheet**

| Release | Version | Date | Rédacteur | Commentaires |
|---|---|---|---|---|
| Initial | | 4 mars 2023 | OG | First version |
| R1 | | 5 mai 2023 | OG | Added Snippets, and Gets #DECLARE, #Compiler_ |
| | | | | |
| | | | | |
| | | | | |

**from v19 R6, for project mode**

For 4D, better in project mode, for programming language set to EN.

Protée
IT, app design

Protée sarl
31, rue Sainte-Marthe
31000 TOULOUSE

Tél : 0970 46 56 46
Mob : 06 3718 5941
www.protee.org / info@protee.org

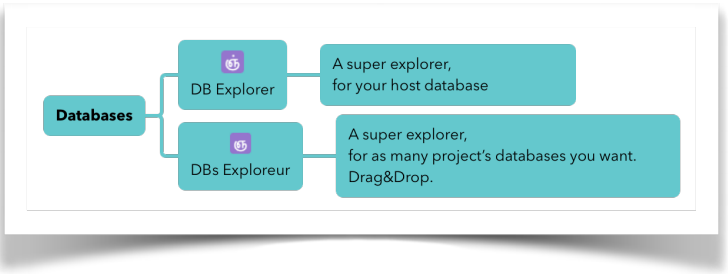
# Overwiew

## Introduction

The ogDevTools component helps the developer in its daily routine with they 4D products. It gives some meta interfaces, with high level interaction with your database. It is divided into actions and modules for Databases, Methods, Forms, and Miscellaneous.

### Databases

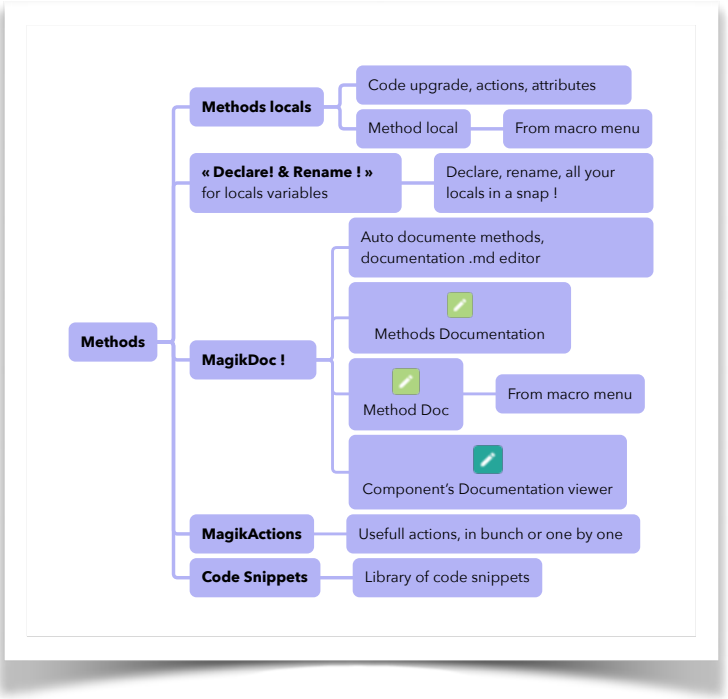
The goal is to explore (search widget), see, analyse and drag&drop all the items between databases. It integrates **Gnânam**, a new widget and Explorer for your host project databases.
You have one DB Explorer for your current host, or DBs Explorer that opens a Form with as many explorers for the databases you want, and make Drag&Drop easily.




### Methods

With **Methods Locals**, you can see and filter methods from how many Undeclared, Unused, and Unnamed.
When you begin to use this module, you can't stop to use it after all, it is so powerful.

With **Declare&Rename!**, we wanted to propose, with naming spaces, an easy way to automatically declare from locals based on they names, and rename based on they declaration's types. Based on naming space's conventions (3 available), modifies the local's names for your methods in a snap. You can also convert all the variables of a selection of methods in a project from one space to an other: just blowing mind!



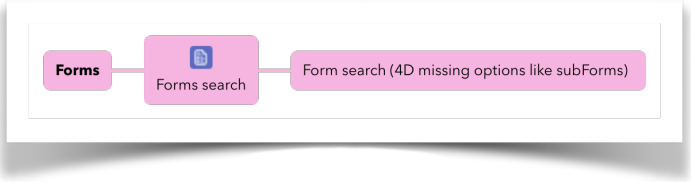

With **MagikDoc**, the goal was to get an instant documentation for databases, but specifically for component: if you get the call, the parameters, with the alias name and the type, you have almost all!
You generate your markdown documentation for your methods in a snap, with different tags you can modify and update. A dedicated « Component's Documentation Viewer », thru 4DPop, proposes all the components found with some .md in them, and opens a Form with all methods with .md and the markdown in a click. Just astonish.

You have **MagikActions**, a set of predefined actions you can do on methods, like convert C_XXX() to Var xxx: Type, or create a #DECLARE based on $vLocal:=$n or $0:=$vAnswer…
All of this is accessible for batch actions or macro individual method.

**Code Snippets** gives you a place where to put and get snippets of code. You can also easily share code between your databases, the synch is instantly made for added, modified and deleted items.

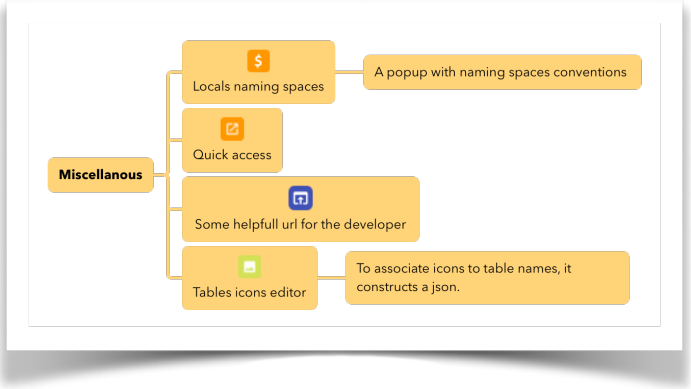
## Forms

This interface is helpful to find the occurrences of subForm basic name, making easy to find your component's widgets without restless nights…



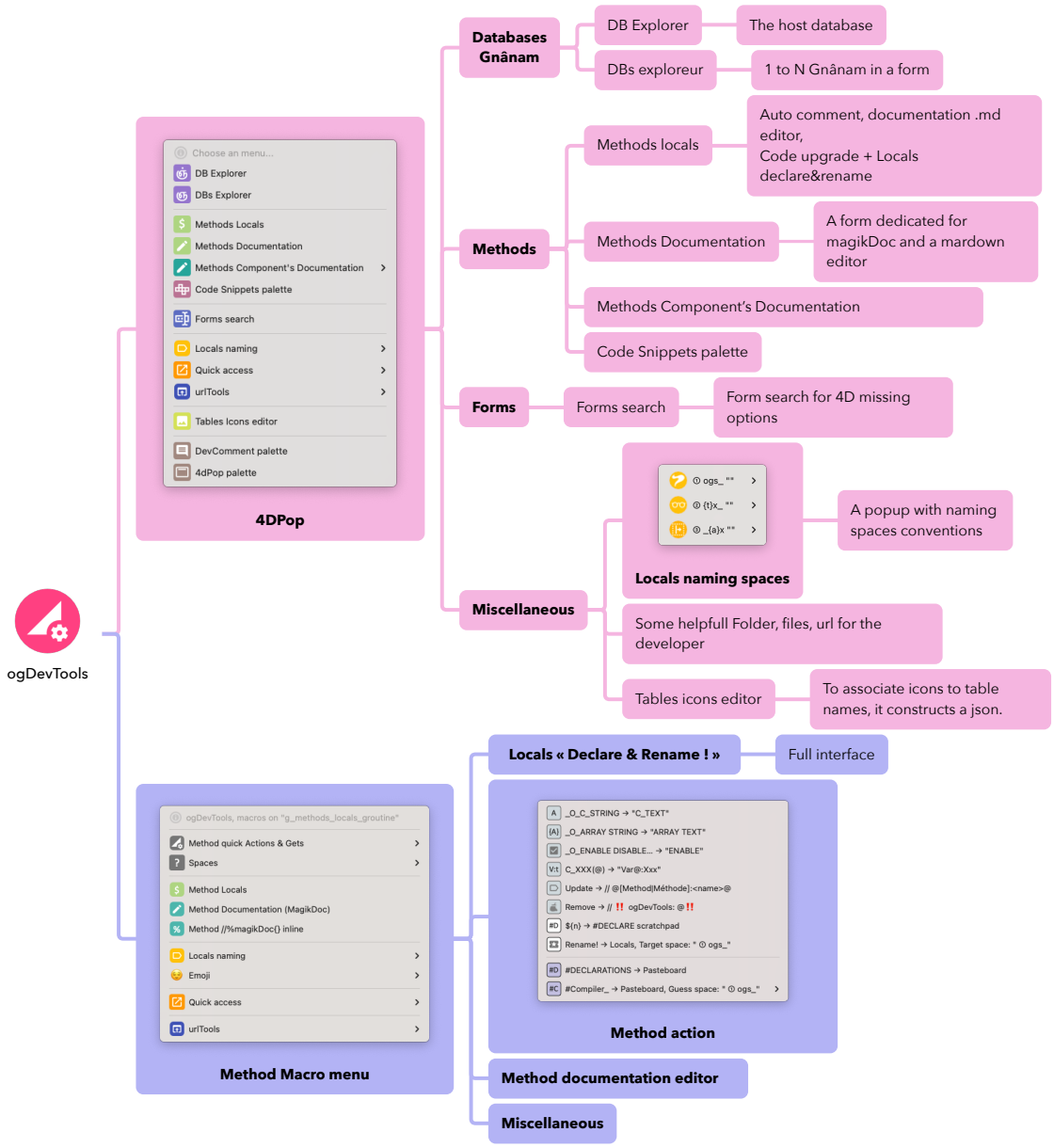

## Miscellaneous

Some common quick actions, to see files, folders, open doc…



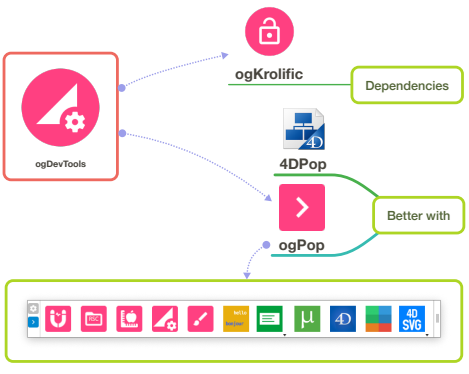

It is worth a try rather than speaking about it!

# Access




## Dependencies

To get **ogDevTools** works, you need to have it in your component's folder, with **ogKrolific** (Protée's license manager) installed too (aliases work). Don't forget to exclude them from your build's settings if you don't need them in the compiled target. Best to use with ogPop, but you need 4DPop installed too.



## Register

To register ogDevTools, if you got one, you must execute this code on startup — here shown with a trial key till 1/04/2023 :

```
If (Not(Is compiled mode))
      ARRAY TEXT($aT_components; 0)
      COMPONENT LIST($aT_components)
      If (Find in array($aT_components; "ogPop")>0)
            EXECUTE METHOD("ogPop_palette")
      End if
If (Find in array($aT_components; "ogDevTools")>0)
      // ***** ogDevTools
      // *
      // * Version: 2
      // * Trial till: 01/03/23
      // * Licenses: 1
      // * Modules: is_win, is_mac, option1, option2, option3, …
      EXECUTE METHOD("wod_initRegister"; $isOk; "ogDevTools"; "vvwT7zS88BS03ErvxZ/0003X")
      // *
      // *****
      End if
End if
```

In demo mode, you get 30 minutes to work on **ogDevTools** with some annoying popups.

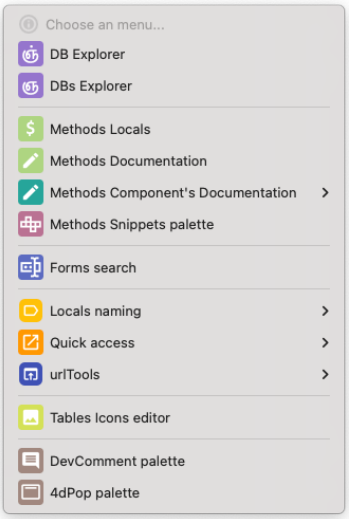
## Open Form in host

The new 4D command FORM EDIT is not totally finished for component, and is not able to target an host database. In order to get it works from ogDevTools, you must create the method **wod_FORM_EDIT**, and check « shared by components and host project ».

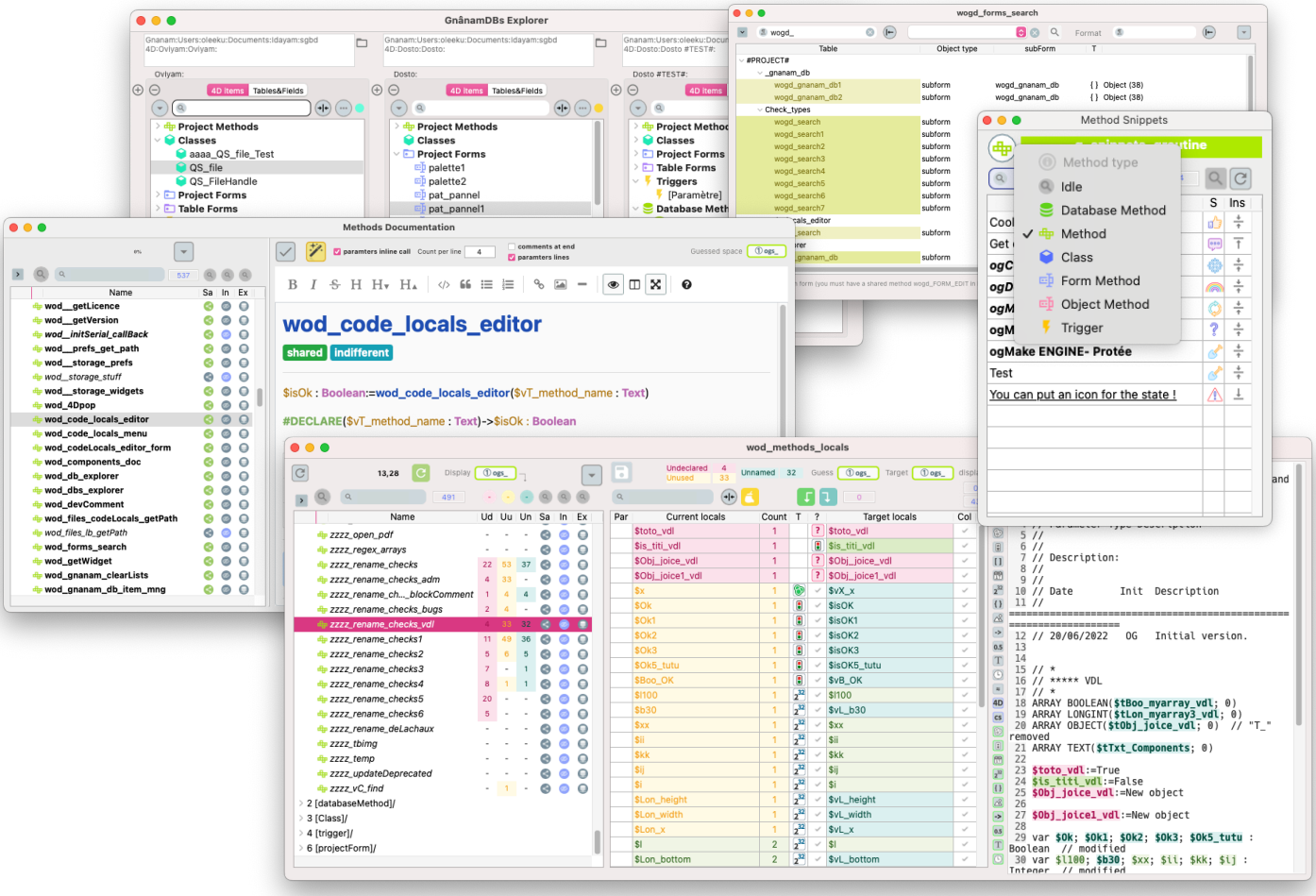
```
// *
// ***** wod_FORM_EDIT in host, shared
// *
var $vL_noTable : Integer
var $vT_form : Text
$vL_noTable:=$1
$vT_form:=$2
If ($vL_noTable=0)
      FORM EDIT($vT_form)
Else
      FORM EDIT(Table($vL_noTable)->; $vT_form)
End if
```

# Menu 4DPop/ogPop

The ogDevTools is also compatible and mounted with 4DPop and better, ogPop. Click on the icon and you will get this menu. You can also get it with a call to the method **wod_4Dpop**.

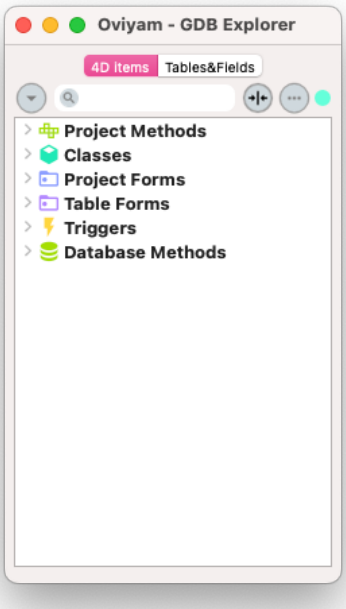



- « DB Explorer » explore the host project database
- « DBs Explorer » open a form with as many « DB explorer » you need in it, allowing easy drag&drop between databases.
- « Methods Locals » batch modifications for attributes, and central access to module « Method locals » where you can Declare & Rename based on naming spaces.
- « Methods Documentation » to easily edit and magikDoc! your methods, in a bunch or one by one.
- « Methods Component's Documentation » allows you to consult all .md files found for each of your host component's database.
- « Methods Snippets palette » is a full shared library between your projects.
- « Forms search » subForm source name, on type, and format.
- « Local naming » for reference, with for now three spaces
- « Quick access » for useful folders and files
- « urlTools » some very useful links to 4D web
- « Table icons editor: associate from icons in Resources/tables/ to tables to get an icon by programming.
- « DevComment palette » places a palette to know where you are when more than one 4D are opened at a time.
- « 4dPop palette » open a palette window with a button on the last action selected. Right click to change it.



# Gnânam DB Explorer

Gnânam means wise in tamil. As this module is to know all about any 4D project database giving a path to it, this is a good name! Gnânam is basically a widget you can implement in your own Forms, see the « Programmer's guide » for that.




Here, Gnânam DB Explorer displays for the actual host, all Project Methods, Classes, Project Forms, Table Forms, Triggers, and Database Methods.

This works only for Project mode database, as it is wandering all the database items from a project database on disk. The advantage is the possibility to explore a database that is not the current launched.

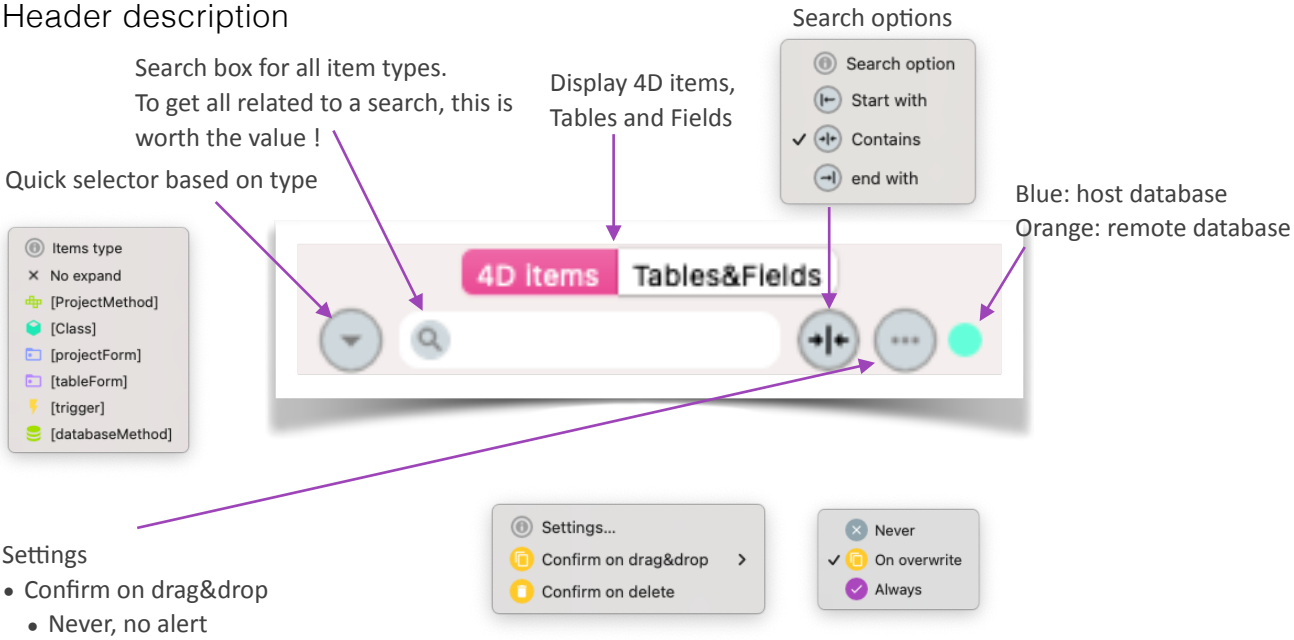
## Keyboard shortcut — here Command/Option/Shift g

To add a keyboard shortcut to invoke Gnânam Explorer, you can use this code.

```
var $vT_keyboard:Text
If (Is compiled mode)
	$vT_keyboard:="Z_keyboard"  // Keyboard events for compiled, yours
Else
	$vT_keyboard:="Z_keyboard_dev"  // Keyboard events for non-compiled
End if
ON EVENT CALL($vT_keyboard)

// *
// ***** Z_keyboard_dev
// *
If ((Macintosh command down | Macintosh option down) & Shift down)
	If (KeyCode=223)  // g
		FILTER EVENT
		var $vL_gnanamDB_P : Integer
		$vL_gnanamDB_P:=v_find_process("$wod_db_explorer")
		If ($vL_gnanamDB_P>0)
			BRING TO FRONT($vL_gnanamDB_P)
		Else
			$vL_gnanamDB_P:=New process("wod_db_explorer"; 0; "$wod_db_explorer")
		End if
	End if
Else
	// ... your own
End if
```

## Header description



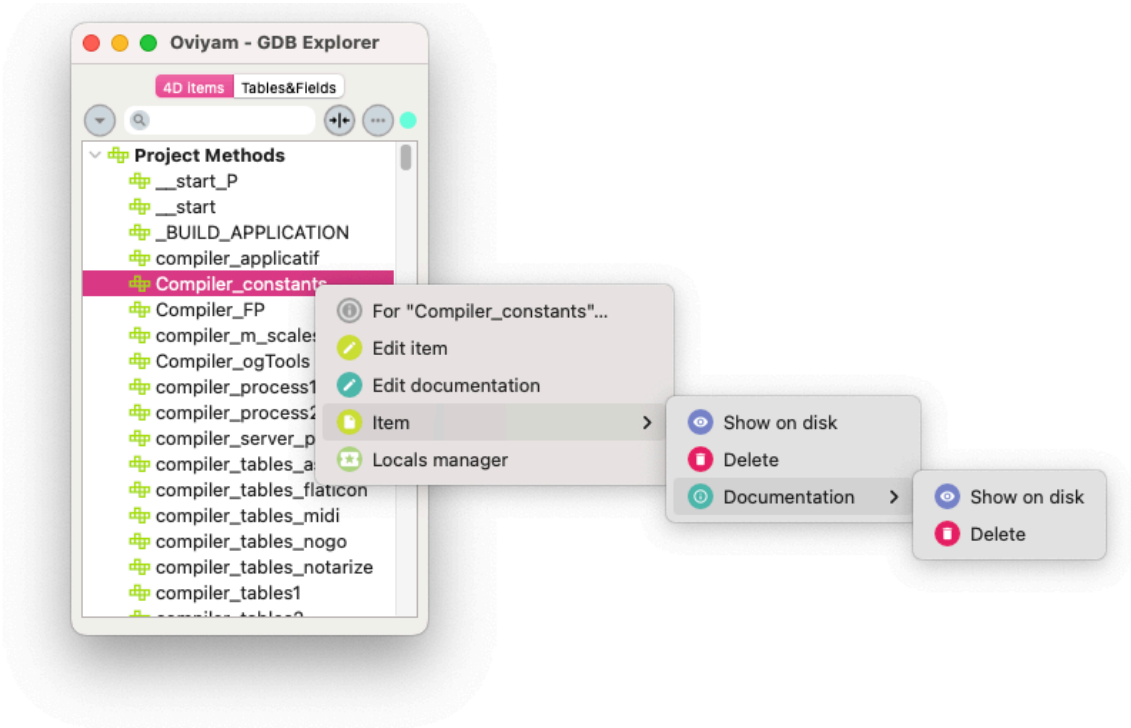

Settings

- Confirm on drag&drop
  - Never, no alert
  - On overwrite, only if a target exists
  - Always
- Confirm on delete, checked a confirmation is made for each

Drag & Drop is managed in text, and between several Gnânam, making easy to transfer methods, forms, and so one.

## Right click on items

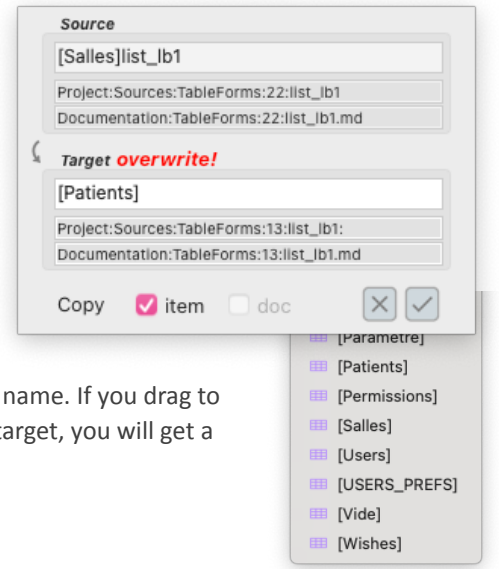
The options dynamically depend from the type of the clicked item.




You can:

- Edit item, in case it's the host database, it opens with 4D itself. In case of a remote database, it opens with Atom (we recommend to install the plugin by Miyako to get the text colorized).
- Edit Documentation, to edit the md file with our embedded offline md editor.
- The Item delete, delete the file or folder and the doc if any.
- Locals manager, see the chapter on it.

## Drag&Drop




In case of the right settings, a confirm dialog might open to explain all what will be done. You can selectively uncheck item and/or doc copy.
The copy is for the full item, so:

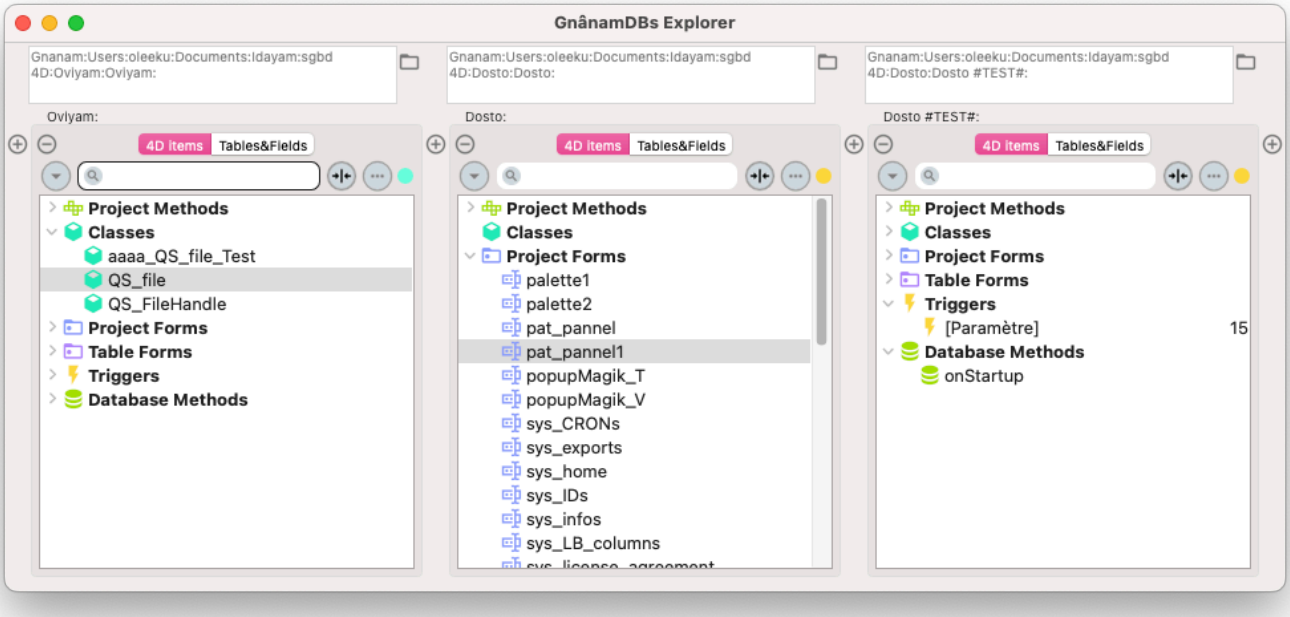
- Project Methods, Classes, Triggers, and Database Methods => file
- Project Forms, Table Forms => folder

**Table management**

For Table Forms, if you drag a Form inside a target Table, or Form, it will be copied in it based on the new table number, regardless the table name. If you drag to « Table Forms », it will be based on the table name and if not found on target, you will get a popup to choose in which table to do the drop.

# Gnânangal DBs Explorer

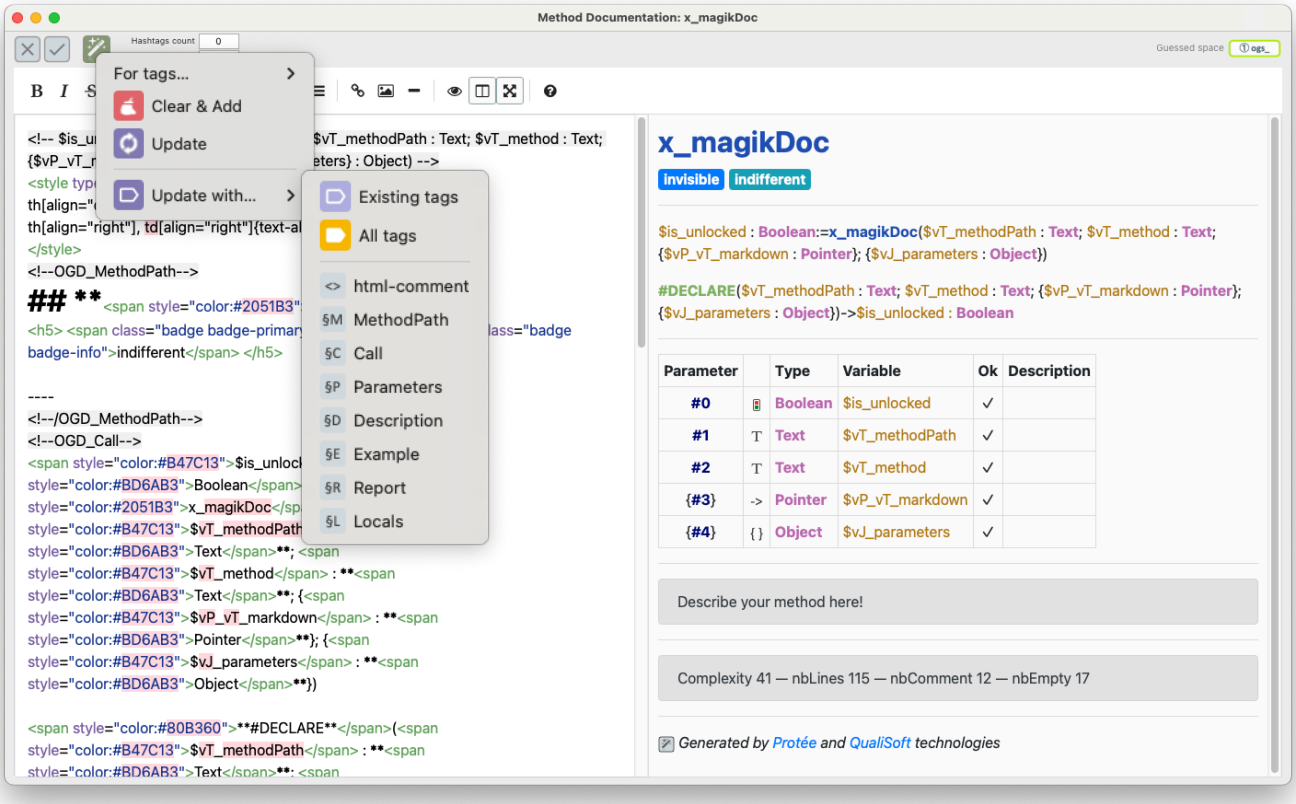
Gnânangal is Gnânam at the plural form: this interface allows you to add (and remove) as many Gnânam widget you want, and your actual settings are saved.
You can remove a Gnânam, or add one before or after an existing one.
Then you must indicate the database by clicking on the folder button — Command click to automatically affect to the actual host database, getting a blue indicator.
This interface is fully adapted to Drag&Drop and databases comparison.



# Method Documentation

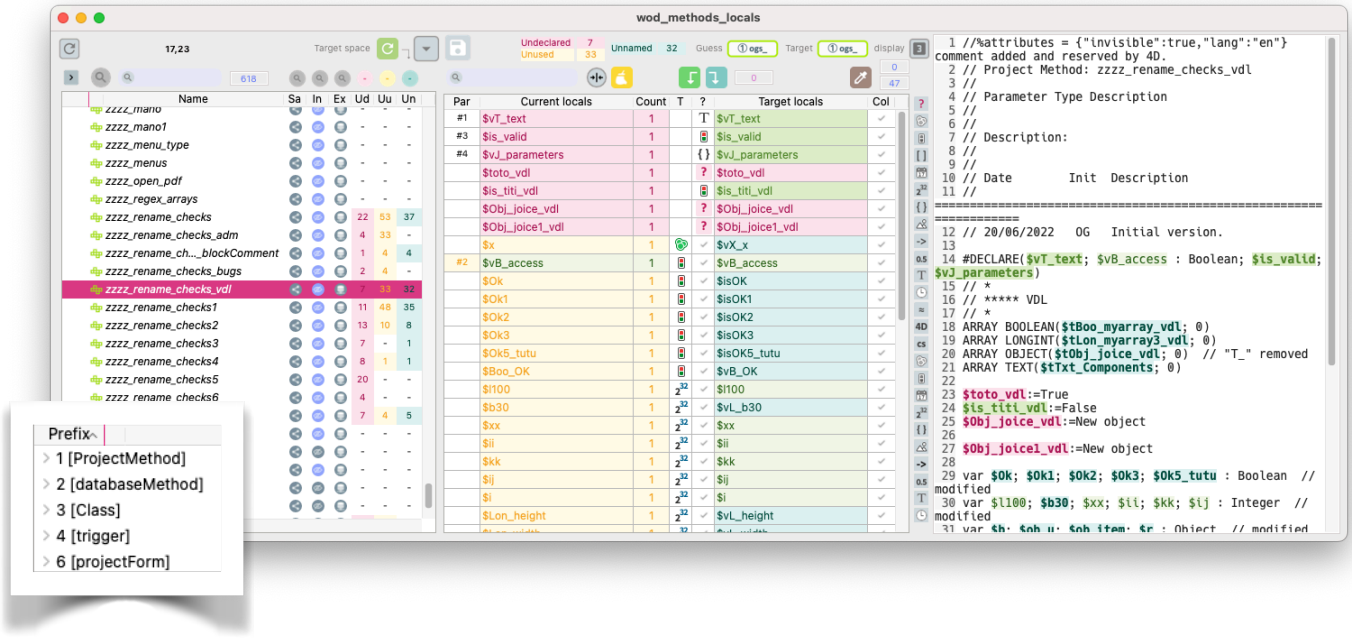
It is a markdown editor, that uses a web area, without the need to start the 4D web server or to be online. All is files based with a folder in the Ressources folder. This editor is based on the open source project named « easyMDE » (a fork of simpleMDE with some bugs corrected).
This Form contains two zones, the one on the left to type in markdown syntax, the right displays the final result.
This Form is opened in a new process, so even in a macro, the method is not stuck and you can continue to consult or edit it.

Dawn yourself in the automatic generation based on tags, by clicking the MagikDoc! button. See the right chapter below « Methods Documentation » for a full description.



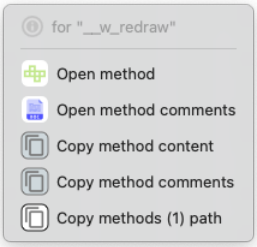
# Methods locals

This manager displays at left all the methods found in the host database.



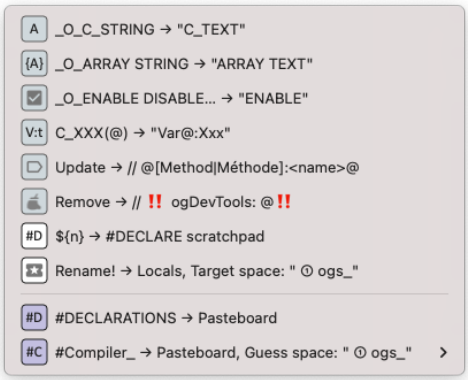

At startup, the progress bar is running to calculate for each method, the « Undeclared », « Unused », « Unnamed » informations, or « Ud », « Uu », « Un », available for the current method on the middle top, and within the listbox (it takes less than a minute to process and display it). Those three informations have a two states associated filter just above the columns.
On the right, you get two zones for the « Method Locals », when you click on a method, you get all information about that method. See the chapter « Method Locals » on how to use it.

The listbox displays method's properties « Shared », « Invisible », « Execute on server » or « Sa », « In », « Ex ». Those three properties have a three states associated filter just above the columns.
All methods are sorted by type — Project, Database, Class, Trigger, ProjectForm. You can double click on a method to open it.




If you right click on a method, you get this menu to open or copy.

You can apply bunk « quick actions & gets » and MagikDoc or Attributes setting to the selected methods in the listbox. See the macro menu for more explanations.




**Actions**

- _O_C_STRING → « C_TEXT »
- _O_ARRAY STRING → « ARRAY TEXT »
- _O_ENABLE DISABLE... → « ENABLE »
- C_XXX(@) → « Var@:type »
- Update → // @[Method|Méthode]:<name>@ »
- Remove → « !! ogDevTools-@ !! » for comments added by ogTools
- ${n} → #DECLARE from your $n declarations
- Rename → Locals « ① ogs_ » in a target space

**Gets**

- #DECLARE: Concatenate the declarations of all your selected method into the pasteboard
- #Compiler_: Concatenation of all your selected methods ready to paste into the pasteboard or a « compiler_ » method. The « Guess space » is used for undeclared variables to guess the type.

**Others**

- Magik comments
- Attribute, set to… (shared, invisible, ExecServer)


## Obvious actions

- _O_C_STRING → « C_TEXT »
- _O_ARRAY STRING → « ARRAY TEXT »
- _O_ENABLE DISABLE… → « ENABLE »
- Attribute, set to…

Do the stuff !

## « Var@:type » → C_type(@)

Will convert the old syntax C_LONGINT, C_TEXT… to the modern Var :Integer, Var :Text.
If variables are interprocess, they are not converted, and if a line is a mixture of types, the line is split into two, one in the old syntax with interprocess only and one in the new syntax with locals only.

## Update → // @[Method|Méthode]:<name>@

Easy but powerful: who have the comment with method name in the methods up to date ? So this will change the method name in <name>, for the first occurence found in following patterns, for EN and FR, spaces are any.
// @Method : <name>@ or // @Méthode : <name>@

## Remove « ‼ ogDevTools-@‼ »

The next « Rename! » action might write at the beginning of methods a comment tag to describe some eventual problème, like name collisions. All comment Tag are on the form: // ‼ ogDevTools-@‼, with @ the real description of the problème. This command will remove all ogDevTools added tags for the selection.

## ${n} → #DECLARE

This action does something only for methods without a #DECLARE in it. The target method is parsed using the « Declare&Rename! » module, and get a list of all local variables found with a direct affectation to a parameter.

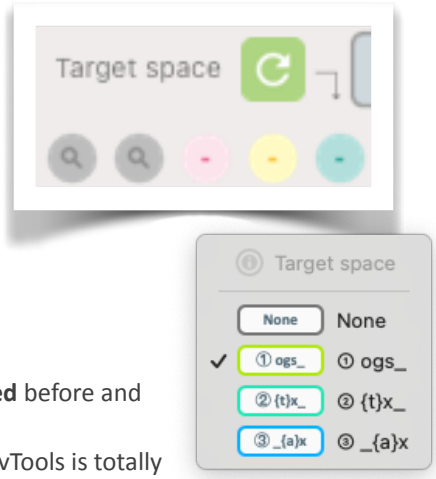
```
var $vC_names : Collection
var $vL_toto : Integer
var $vT_stack : Text
$vL_toto:=$1
$vT_stack:=$4
$vC_names:=$6
var $vT_text : Text
$vT_text:=$0
```

As you can see, you can have holes in the definitions: they will be filled in the result with local's name $vV_inex<n> : Variant. The #DECLARE will be added before the first line of code, and it is your responsibility to remove all the old unused code, here just removing all previous code. This preparation to migrate to #DECLARE really helps, as manually,

you need to do plenty of copy/paste for locals and type. To indicate what are optional parameters for « Declare&Rename! » and MagikDoc!, just add at the end of the declaration // {#3}, if optionals are from 3 included. Here the result is:

```
#DECLARE($vL_toto : Integer; $vV_inex2 : Variant; $vV_inex3 : Variant; $vT_stack : Text; $vV_inex5 :
Variant; $vC_names : Collection)->$vT_text : Text  // {#3}
var $vC_names : Collection
var $vL_toto : Integer
var $vT_stack : Text
$vL_toto:=$1
$vT_stack:=$4
$vC_names:=$6
var $vT_text : Text
$vT_text:=$0
```

## Rename! Locals, space:




Relaunch the worker that calculate « Ud », « Uu », « Un » for all methods in the listbox, based on « Target » naming space.

In Target, you choose the naming space you want to display, and use for the action Locals « Rename! ». At this time, three spaces are available, « ①ogs_ » and « ②{t}x_ » and «③ _{a}x ». See the correspondant chapter below for more details.

This action is **totally safe as the collision of variable's renamed names are checked** before and in case this is happening, the corresponding variables remain untouched.

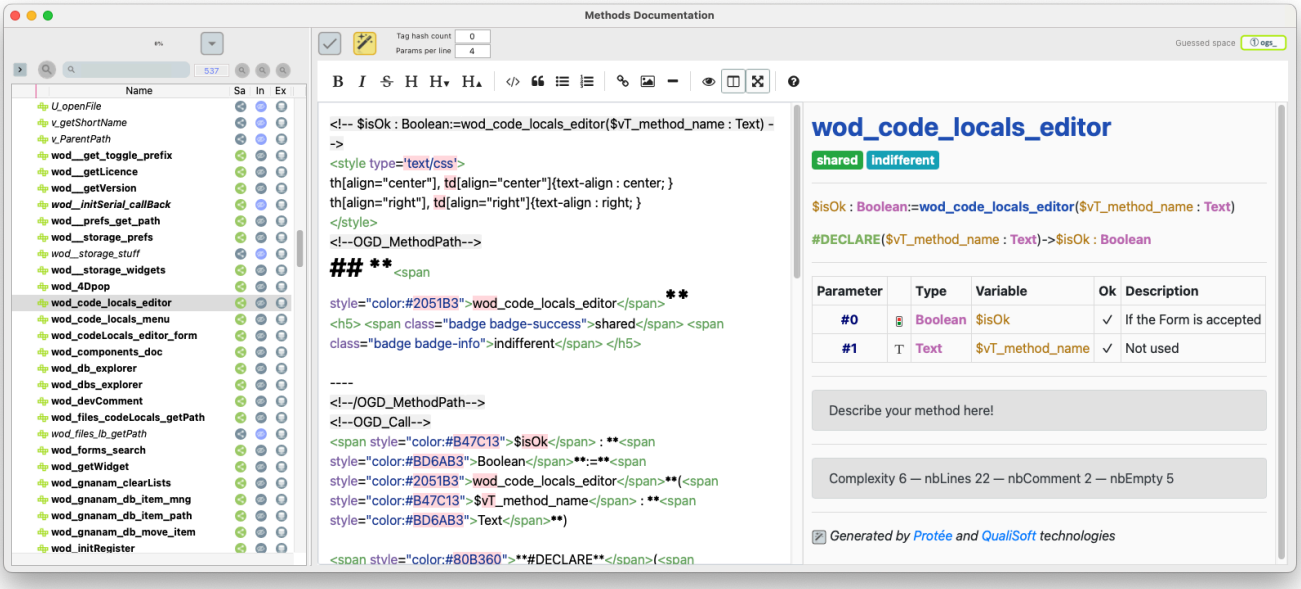
- ☆ Take a particular attention on methods with « Get Pointer » command : ogDevTools is totally unable to guess what variables will be addressed that way ! So the use of this command can break the correct behaviour.
- ☆ This is not possible to automate the Declare! action, as based on naming spaces, it guesses the type of undeclared locals, and it is too dangerous to take it as granted!
- ☆ Soon, an « Undeclare! unused » will be available.

## Locals « Save » behaviour

When you deal with locals on the right part of the panel, you must explicitly save your changes. If there is something to save, the « Save » button is enabled green. But to be able to save, ogDevTools needs to write back to the method, that must not be open for the user! If the method is already open, that method's window goes frontend and this button becomes orange to warn you that you just have to close the method, and click to this orange « Save » button again! The « save » button shortcut is Cde/Ctrl S.

# Methods Documentation

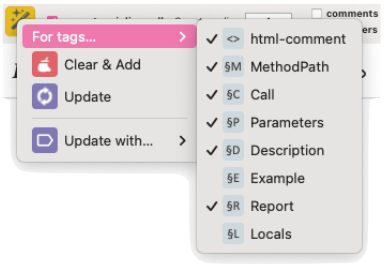
This manager displays at left all the methods found in the host database, and at right the markdown editor.
This editor is based on easyMDE, and H and bootstrap is available.
The magikDoc! can generate 8 tag's zones in the markdown, and can be updated if you changed something in your call, parameters, and so one. For some tags, the update is non destructive for the user's datas, like the description for the parameter's list.



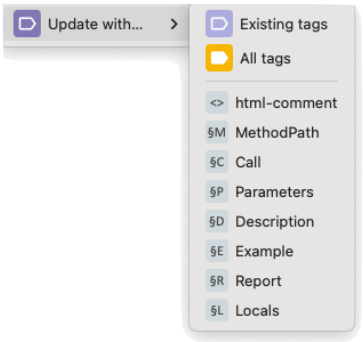

## Actions

For one method, you can execute an action from magikDoc! with this menu.
The two first items will use the tags checked in For tags… sub menu. Choose one to toggle the check.

- Clear & Add: clear the markdown before, then add the checked tags.
- Update: add/update the checked tags.




- Update with…: you can choose « Existing tags », « All tags », or choose one tag to add/or update. With one tag update, and only with, the eventual mask given in the method is ignored and the update is done.




## What is a magikDoc tag?

A magikDoc tag is a zone in the markdown, bounded with html comment tags, one at start and one at end. The start tag might contain an optional heading (preference « Hashtags count, 0 for none).
ie for magikDoc tag « Call », with a count of 3, they are:

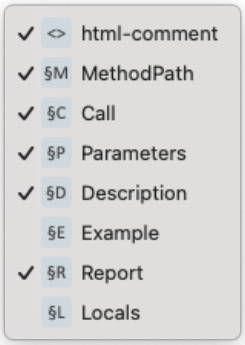
- <!--OGD_Call-->\n### Call
- \nThe content\n
- ----\n <!—/OGD_Call-->\n.

## What means update?

A magikDoc tag update is a search in the markdown for the magikDoc tag with this regex « <!--OGD_TAG-->(\n#* *TAG)? ». So a tag is found with its comment part, and the eventual Header part with as many « # ».
Some tag are « write only », that means the update will replace everything in the tag. Some others are user editable, and the content somehow will be preserved. Updating a tag is:

- find if the start tag exists. If yes, find if the end tag exists, if yes too the content is replaced, else it is added before the start tag found.
- if it do not exist, the tag is added before the next existing tag supposed to follow. ie for « Call », it will check all the following, « Parameters », « Description »… till end and the first found will have the tag inserted before.

This garanties the tag is always at its right predefined place.




## Tags

The MagikDoc! module is managing up to 8 zones, each zone for specific informations, some for generation only, some to be edited after and keep informations on eventual updates. You can select which tags being used for the two first actions. For tags Call and Parameters, the output is not just a text output of #DECLARE: it is made after a deep analysis of the method, the same used in Declare&Rename, collecting all $n affectations, and try to find all declarations.

#### HTML_COMMENT

It generates the first html comment to display as the method mouse over tip. The content is like the Call parameter's first line, the method call syntax. Fully cleared on update.

#### METHODPATH

This is the method name and attributes tag zone. Fully cleared on update.

#### CALL

This display the method call line and the #DECLARE for the method, directly from the line itself if it exists, or deducted from $n affectations. Only the named parameters appears, from aliases in #DECLARE or affectations with locals. When optional, the parameters are placed into curly brackets { }. When generic, the parameters are following with « {;…} ».
Fully cleared on update.

#### PARAMETERS

This display all the named parameters found into the method. Parameter, Type, Variable, Ok and Description.
For parameters, #n means from a #DECLARE, $n from a found affectation.
The Ok column indicates if the type is in collision🚫, guessed from its name⚠️, or declared✓.
The « Description » column is preserved from update: The content is put back into the new added lines if the variable's name is the same, or if the parameter is the same.
When optional, the parameters are placed into curly brackets { }. When generic, the parameter's numbers are placed into curly brackets.

#### DESCRIPTION

This is a user zone, and if not empty, it is kept unchanged. It use the following bootstrap code:
Describe your method here!

#### EXAMPLE

This is a user zone, and if not empty, it is kept unchanged.

#### REPORT

Gives some informations about the method. It will be enhanced later. Fully cleared on update.

#### LOCALS

This display all the locals found in the method, with information on parameters as for the Parameters tag.
The « Description » column is preserved from update: The content is put back into the new added lines if the variable's name is the same, or if the parameter is the same.
When optional, the parameters are placed into brackets { }.

## Colors used

With wod__storage_prefs you get an object with ogDevTools preferences. The following is used in magikDoc! :

```
$vJ_colors_md.method:="#2051B3"
$vJ_colors_md.local:="#B47C13"
$vJ_colors_md.type:="#BD6AB3"
$vJ_colors_md.command:="#80B360"
$vJ_colors_md.param:="#000B76"
$vJ_prefs.j_magikDoc_color:=OB Copy($vJ_colors_md; ck_shared; $vJ_prefs)
```

Those values are not saved, it is you responsibility to initialise them.

## Method Prefs overwrite

You might specific preferences for some methods. For that, you can add this line at the beginning of the method.
//%magikDoc { "property":value , etc }

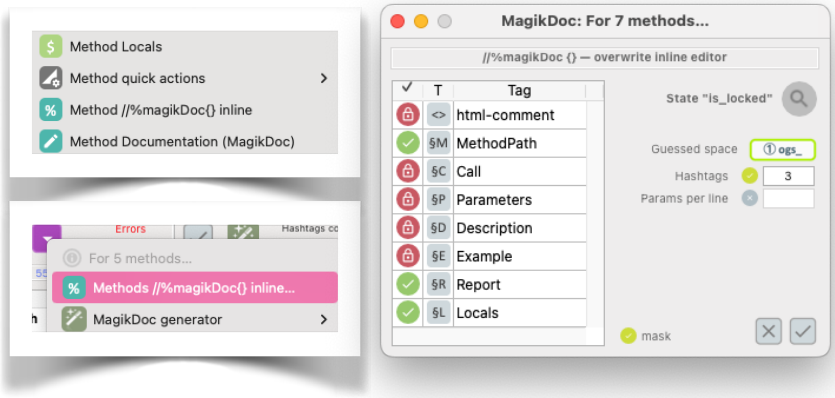
What is inside will be treated as a json file. The following properties are used:

- is_locked, lock the method from any further modification with magikDoc, never ever.
- guessSpace, specific guessSpace for magikDoc
- l_hashtags, number of # for Headers, 0 means no headers
- l_paramsPerLine, used for the html and Call Tags, to force a Carriage Return every n parameters.
- mask, to mask specific Tags for automatic update. In case of a one tag update, this mask is ignored and the update is made.

This will overwrite the prefs for magikDoc, but only for the current method - the global prefs remains untouched.
This is very useful for sophisticated methods, where you likely finish the doc with your dirty hands, and don't want to have it erased with a magikDoc by inadvertance! For that, you just put in the first lines of your method:
//%magikDoc { "is_locked":true}
Example with all the parameters:
//%magikDoc {« is_locked":true,"guessSpace":1,"mask":2,"l_hashtags":4,"l_paramsPerLine":5}

## Method Prefs set

You can edit the %magikDoc line of your method using the menu in the macro menu, or in bunch with the action menu using the Bunch actions menu.
You will get the %magikDoc{} inline editor.
All the properties you validate, ie no glass for popup, or a green stamp nearby, will overwrite for the method the global parameters.




## #DECLARE added options

You might tell to ogDevTools, for each method, specific values for where begin the optional parameters, and where (usually after the last) begin the generic parameters, if any. There is no way to fully know it by analysing the code, and more specifically with the #DECLARE and the initiated values for not given parameters.
For that, you can add a comment at the end of the #DECLARE, with two syntaxes to indicate in Declare&Rename! and magikDoc!:

- optional: add « {#n} and n will be the starting for optional parameters.
- generic: add « #{m} and m will be the starting for generic parameters. To be sure you know about the type of the first generic parameter, usually being the last, please add it to the #DECLARE at end. Following is $vT_key : Text.

Example :

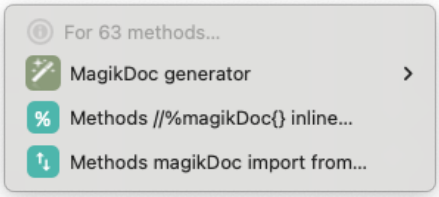
```
#DECLARE($vJ_source : Object; $vT_formName : Text; $vT_key : Text)  // {#2} optional #{3} generic
```

Will generate through magikDoc:

```
wod_setWidgetProperties($vJ_source : Object; {$vT_formName : Text}; {$vT_key : Text{;...}})
#DECLARE($vJ_source : Object; {$vT_formName : Text}; {$vT_key : Text{;...}})
```

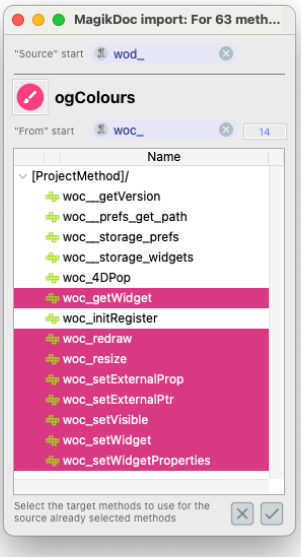
## magikDoc Import from…

The MagikDoc! bunch menu allows you to import already existing documentation in a Component installed to your database.
You might have some generic methods with a little different prefix but with the same action and doc ?
Example for ogDevTools: wod_getWidget, wod_redraw, wod_resize, wod_setVisible… are all the same but the prefix for ogColours woc_getWidget, woc_redraw, woc_resize, woc_setVisible…




For import and copy into your source from a target component, just select the methods you want to use, then invoque this action, it opens a Form where you have to give:

- the source prefix (it will be changed to)
- the target prefix (it will search with it).




The listbox displays all found method will same remaining as source with source prefix exchanged by target prefix. Select the methods in target you want to incorporate into source, accept and you're done.

Even the eventual method's label inside the target doc are renamed into the source.
<prefix_target>method -> <prefix_source>method

Easy and efficient. Once imported, you can use « Methods //%magikDoc{} inline to lock what you have done from regular updates.

ie, imported, renamed, from woc_redraw (ogColours) to ogDevTools:

# wod_redraw

shared indifferent

wod_redraw({$vT_widget : Text})

#DECLARE({$vT_widget : Text})

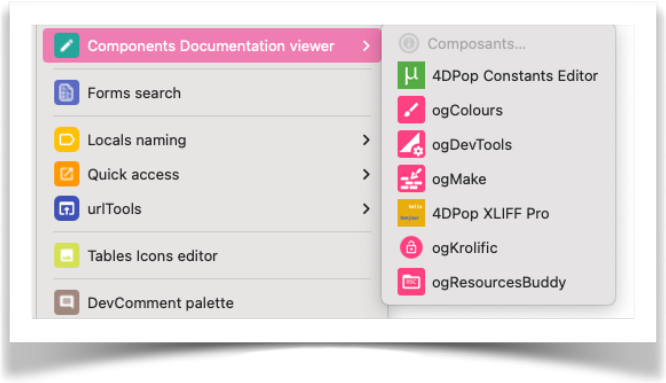
| Parameter | | Type | Variable | Ok | Description |
|---|---|---|---|---|---|
| {#1} | T | Text | $vT_widget | ✓ | Widget name, or current widget if empty |

Redraw the given widget

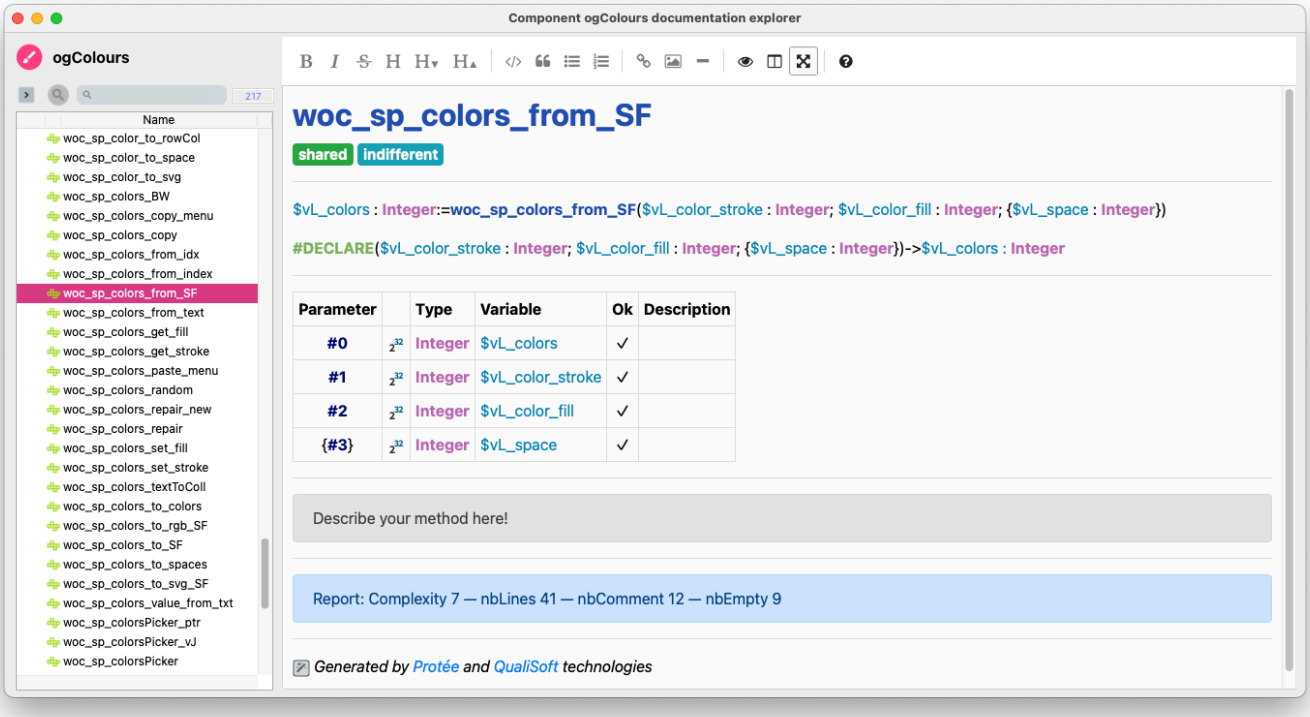
# Component's Documentation viewer

This ogPop menu's item proposes to you the components found in the host's components folder.
You can select one to open then the Form viewer.

From the viewer itself you can later change the component used, by clicking the top left component's icon — the same « Components » right menu opens.



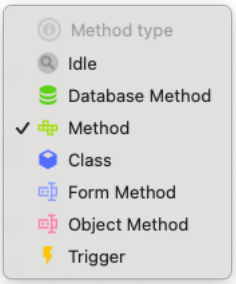

Here from our **ogColours** component:




As you can see, the documentation is on your finger's click! We just added the Description for parameters.

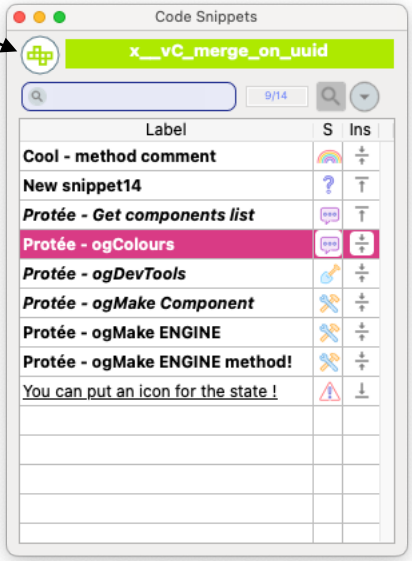
Remark: even if the toolbar is still present (we can't remove it without side effect with easyMDE), this is totally waste to edit anything in this Form as there is no save button!!! This is for consultation only.

# Code Snippets

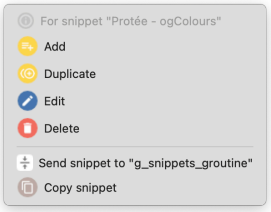



This module is to get on screen a palette with snippets of code, with a global storage between your databases. The palette is contextual and the selection changes based on the frontmost opened windows. The type is fixed based on the window's title, and if in the Dev process. The menu on the right shows the managed types, based on the localisation of your 4D (right now EN and FR).

In the palette, you find a popup button that indicates the type of the method opened, and a coloured field with the method path.
If the type is not recognised, as for a non-method form, a user form… it is indicated with the type set to « Idle » and the list is not filtered.
If the type is recognised, the type button is set and the list filtered to that type.



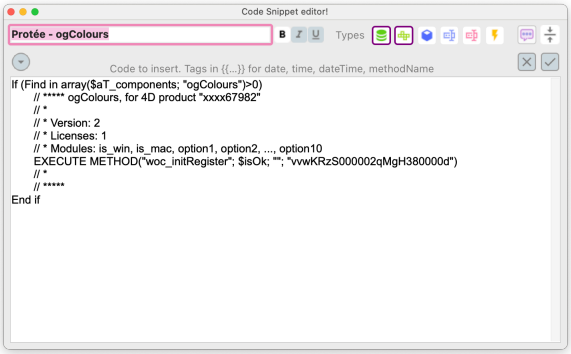

A state button allows to filter on this data shown by an icon « S ».
The « Ins » column indicate where to copy the snippet in the method (top, cursor, bottom).
A right clic to a snippet line opens the Snippet's menu, where you can Add, or Duplicate, Edit, Delete a snippet, and send or copy.



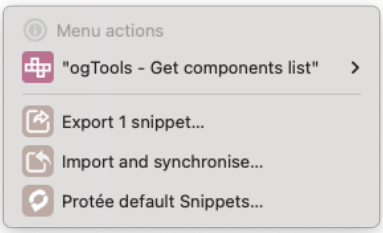

The snippet is sent to its given « Insert mode ». Much better, you can drag&drop a snippet's line to your method and the code is inserted there, regardless the « insert » information of the snippet.

A double click or the choice of « Edit » opens the snippet Form.
The datas are the label, the state icon in a list, the types in multi choice, the Insert mode, and the style (bold, italic, underline), and obviously the code to be inserted!

- A button on the Top left proposes to « smart remove » the tabs found at the beginning of the code.
- You can add dynamic tags in the code within this list: methodType, methodName, dateTime, date, time, and by encapsulating the tag with {{<tag>}}. They are preprocessed before the code to be inserted.



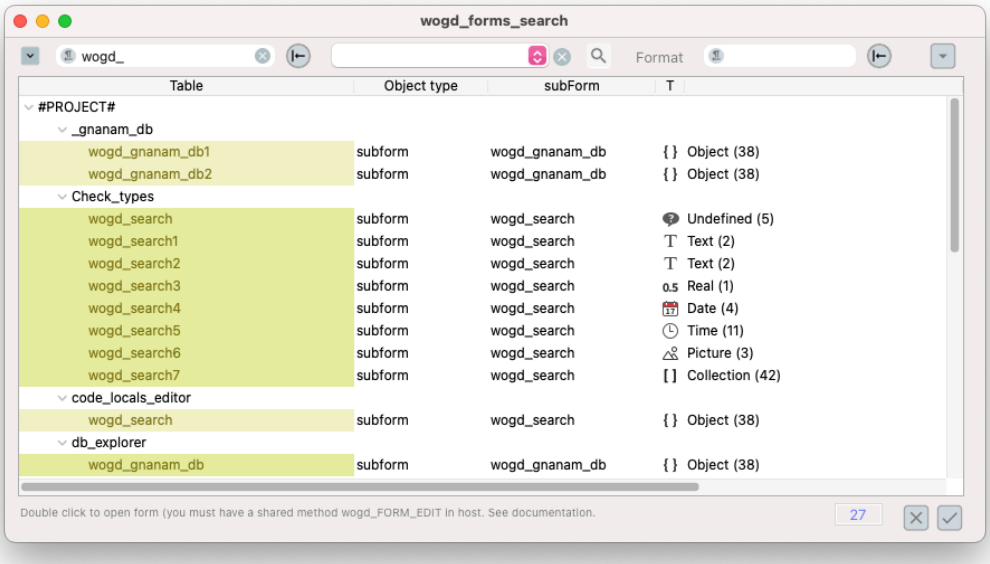

A action menu is available with a other access to the Snippet's menu, and action for Export the selected lines, Import and synchronise based on uuid for new and modified, and Protée Import and synchronise default Snippets to your list (useful code from Protée©).

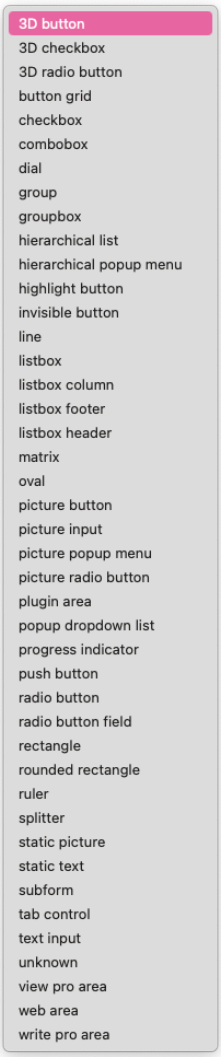



At last, the snippets are synched between all the instanced opened in more than one databases, from insertions, deletions, and last modifications. You're sure when you work in a palette that your modifications will be taken into account in other databases.

# Forms search

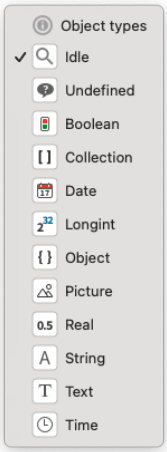
First of all, this manager was created to deal with a lack in the 4D project search that is not able to find subForm's name. If you need to parse all your « wogd_@ » widget instances, now you can!



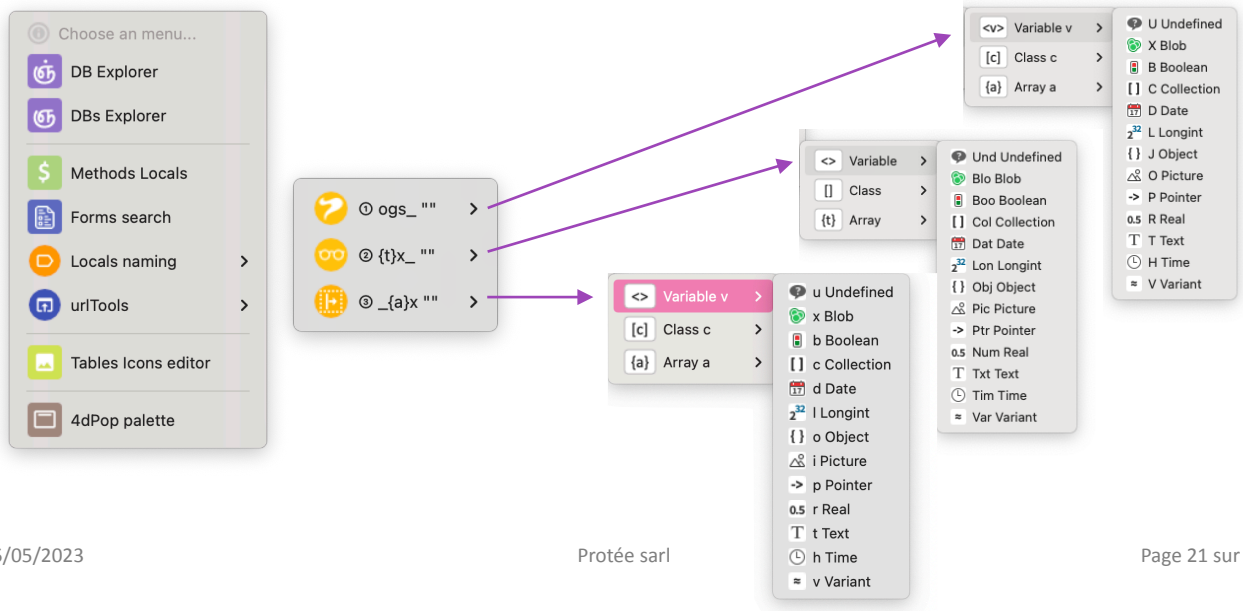

Optionally, in the search box, you enter a name for the object name or the subForm name, select one in the popup for the type, select an object type in the menu button.
You can also do a search based on a format, this is useful when you need to change all containing « , » and change them to « . »...

A double-click on a line opens the corresponding Form.

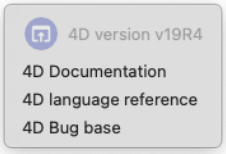



# Local naming

This call in the ogDevTools menu has no action and just gives information on the existing name spaces in ogDevTools.



# urlTools




- 4D Documentation opens the root url for all docs
- 4D Language reference opens the Language for the current running version
- 4D Bug base opens it for the current running version

# Table icons editor

This interface supposes that

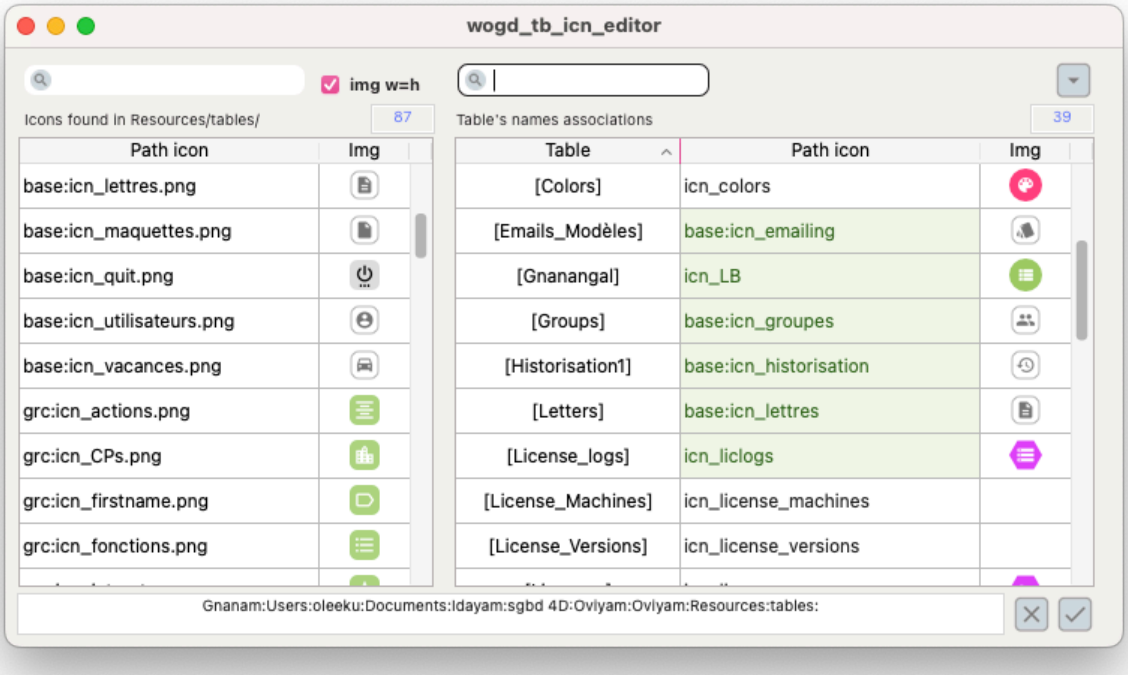
- you have all your icons in RESOURCES/tables/ and in eventual sub-folders
- To use an icon for a table, you use RESOURCES/tables/icn_<tableName>.png

With this interface, you will have all the icons at left, and all the tables at right.
Then, using drag&drop, you can assign an icon on the left to a table on the right, and this will be saved in a json file located at :

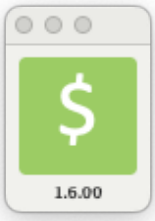
- RESOURCES/tables/tb_to_icn.json
- This will be an object with table number in string for property, and the icon name or sub-path.

{“14“: “icn_android_texts“,“36“: “base:icn_android“,… }

With this, all tables have an associated icon name, based on table name or not.
The white « Path icon » lines are default based line, not declared in the file.
The green « Path icon » lines are names declared in the file, that are different than default based on name.
See the Programer’s guide for more informations about it.




# 4dPop palette

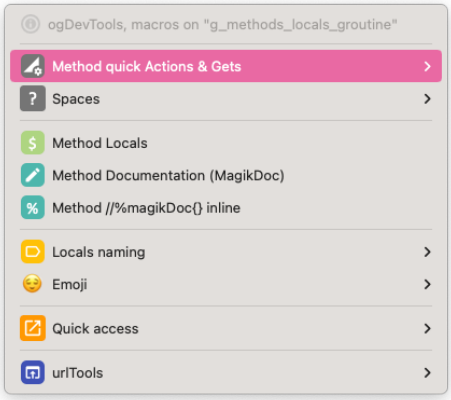



This open a floating palette with a button to open one interface. If you

- click on the button to get the menu to open the corresponding interface and change the button accordingly.
- right-click on the button to open the corresponding interface — or passes it at front.

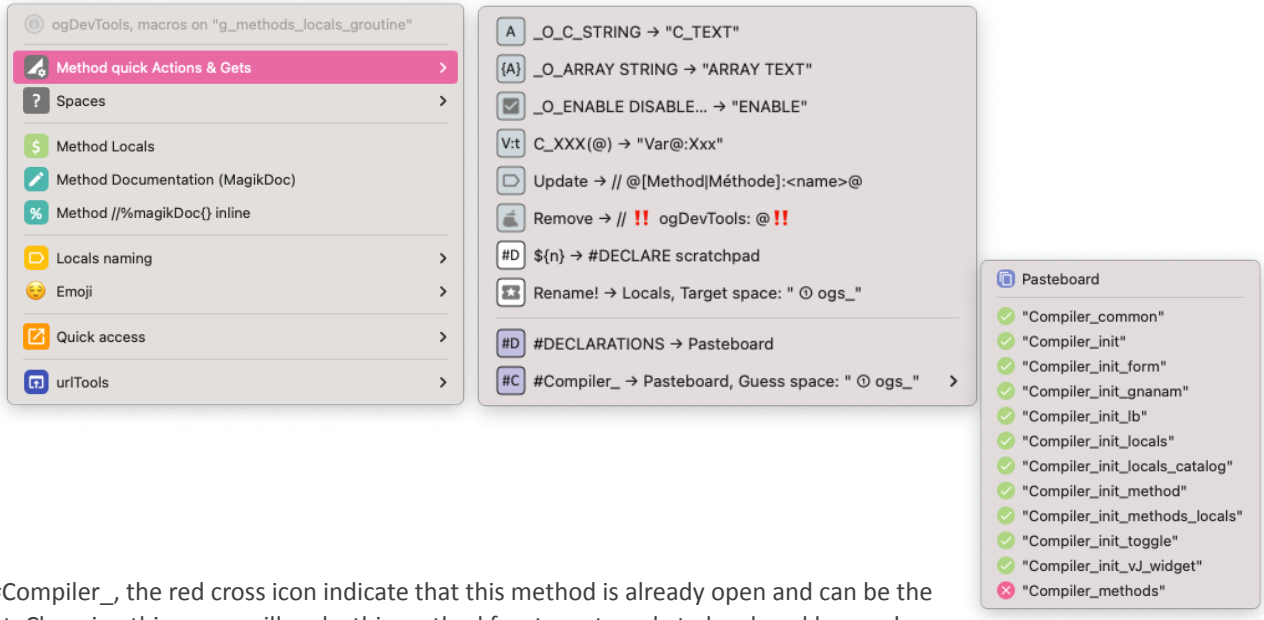
# Menu Macro

The ogDevTools contains a macro file automatically mounted in the macro menu.



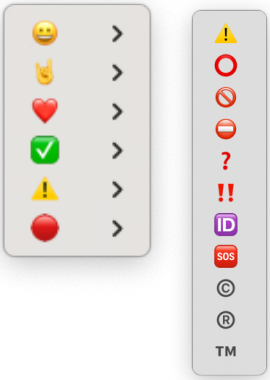

For the current method, you can…

- Method quick actions & Gets: apply a quick action — see « Methods Locals », or get values in the Pasteboard.
- Spaces allows you to see and change Guess and Target spaces without going into the locals interfaces, for the proper Actions & Gets menu.
- Method Locals: open the « Method locals » form
- Method Documentation: edit the doc with the offline embedded easyMDE editor. Opens in a new process.
- Locals naming: check and rename the selected local variable, using one of the three naming Spaces, « ogs_ », « {t} Xxx »; « _{a}x ⚠️ Only the selection is renamed, not all the occurrences! (will be improved in future)
- urlTools: some useful links for 4D
- Emoji: open a hierarchical menu for some useful emoji for your code




For #Compiler_, the red cross icon indicate that this method is already open and can be the target. Choosing this menu will make this method front most ready to be closed by you !
Then you can call this menu again. The method or all the selected methods are inserted into the selected « Compiler_ » method, replacing an existing declaration, or placing the new one in the better found position to let the method sorted by method's name.

Emoji



# Naming spaces

## Introduction

This module is to automatically declares undeclared locals, and to propose a rename based on a selected target space.
At this time, three spaces are available:

- Prefix spaces: « ① ogs_ » and « ② {t}x_ »
- Postfix space: « ③ _{a}x ».

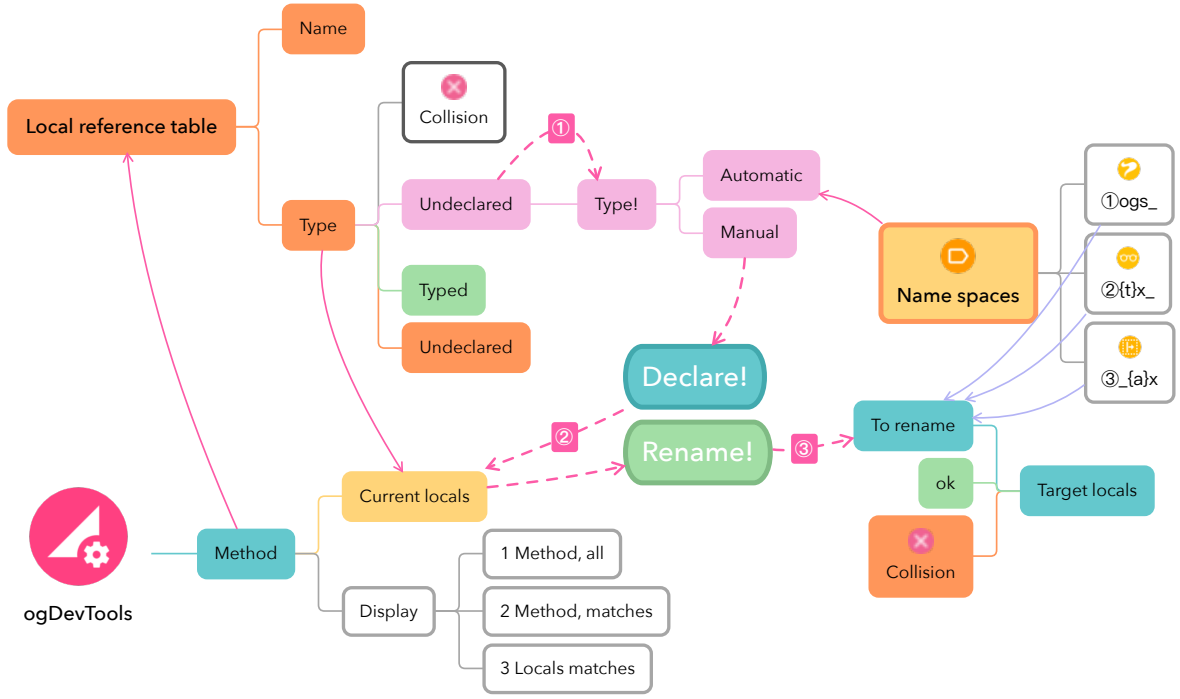
When ogDevTools renames, it first tries based on the current type, to remove the prefix/postfix if one found, then rename with the selected space. All the exceptions in naming are managed for all spaces at any time.

---

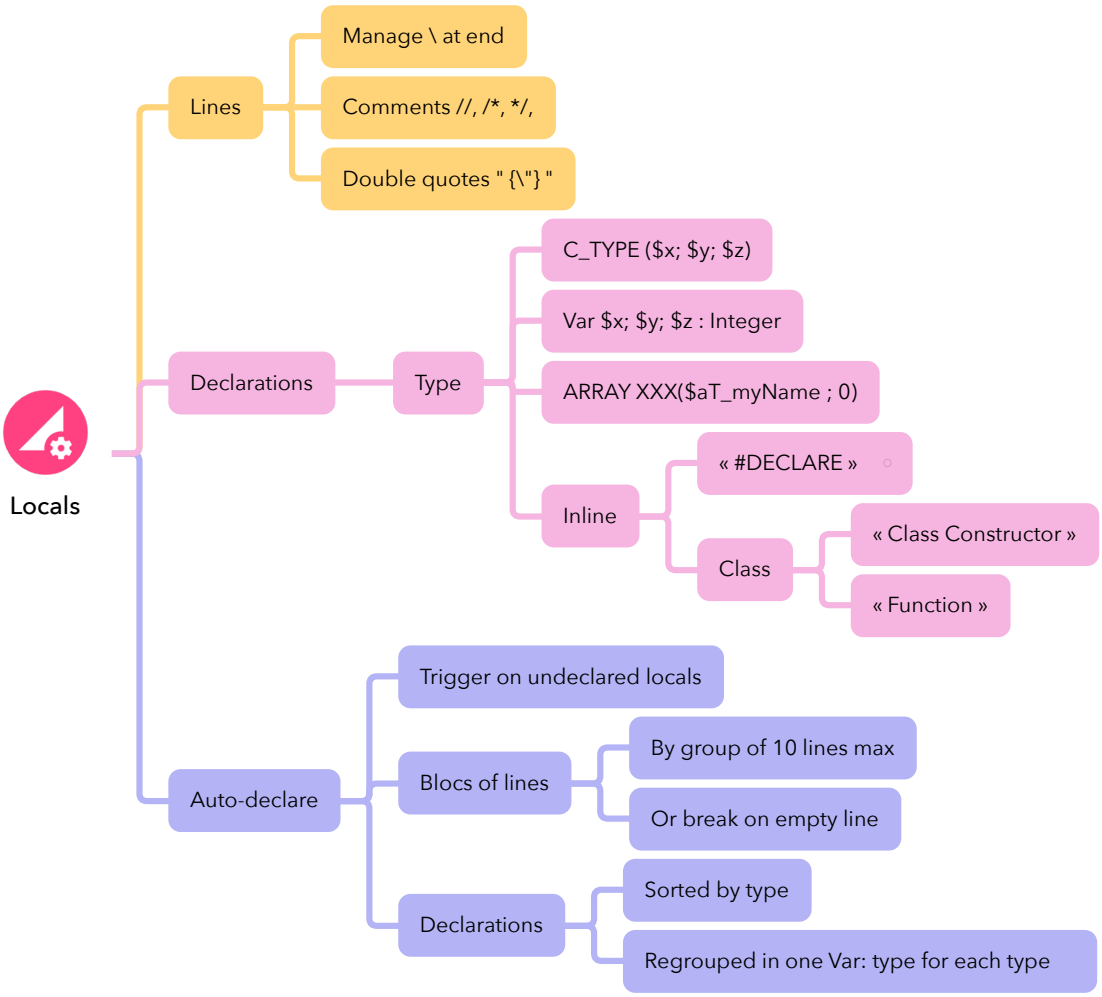
## Schematics

A method contains local variables, or locals, that can be undeclared or declared.
All the job is to Type! the undeclared, then ask ogDevTools to Declare! with a special algorithm, then Rename! based on the selected target name space.
The steps to manage the locals are following:

1) **Type!** manually the undeclared locals, and check the automatically typed locals.
2) **Declare!** the typed locals undeclared.
3) **Rename!** the locals based on a target name space, and change names for collisions if any.

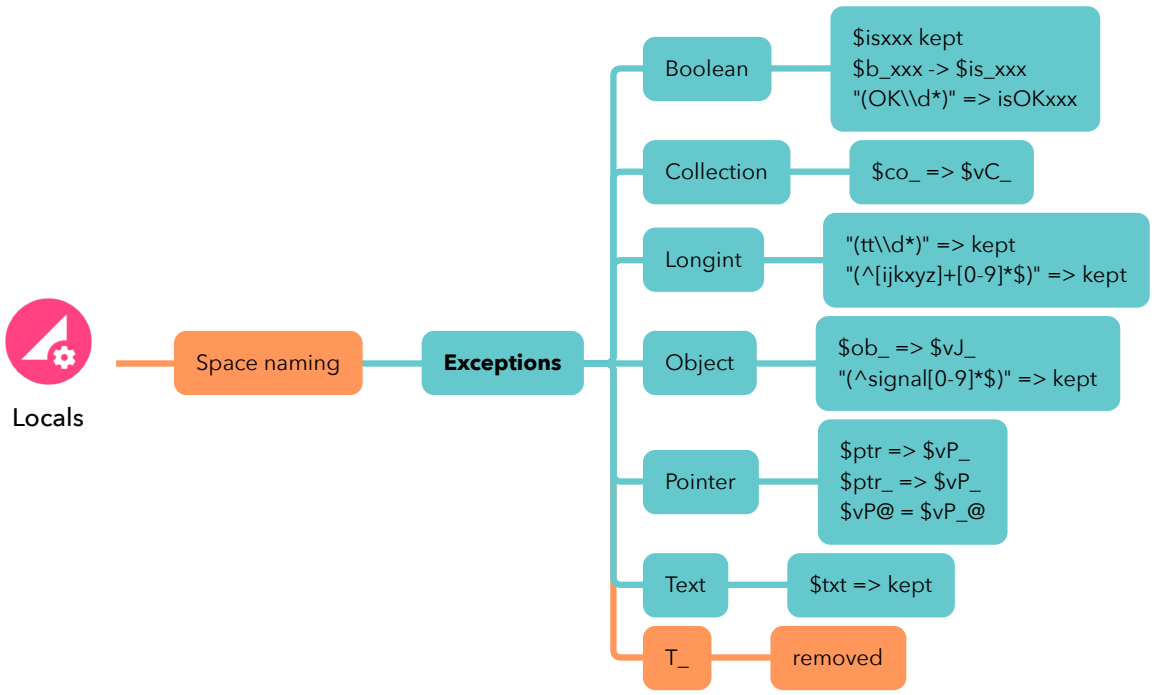


# Naming spaces — Common



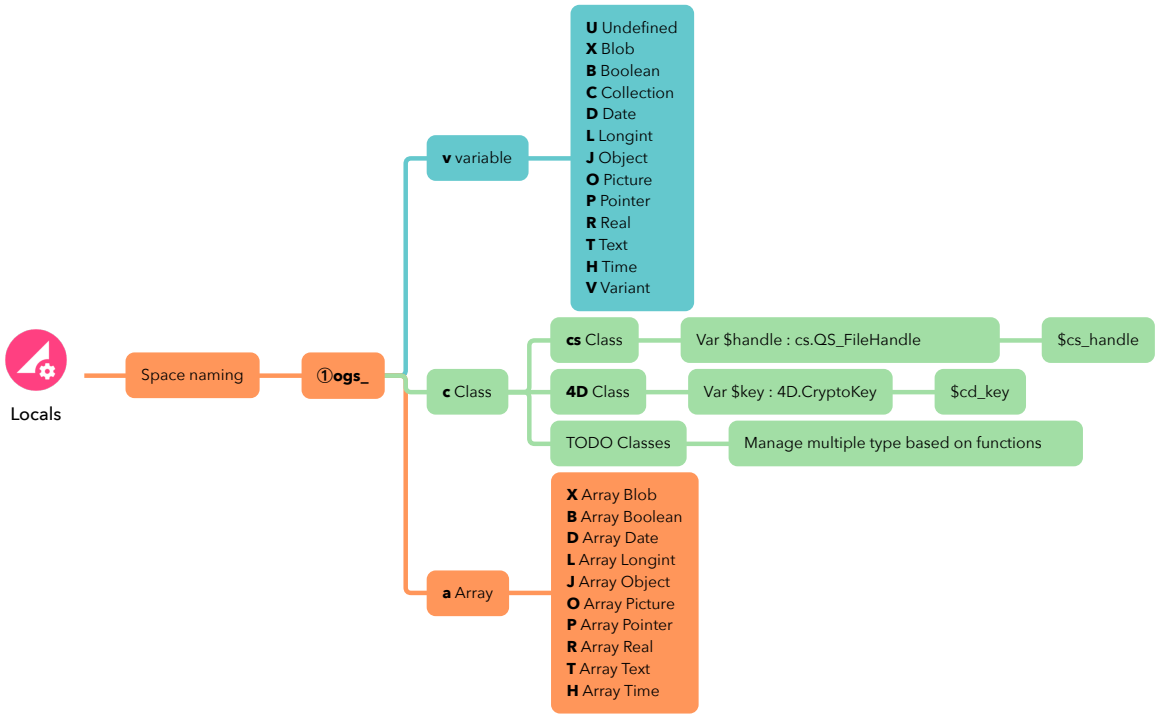
## Naming spaces — Exceptions

Based on my experience, those are exceptions used for all name spaces.



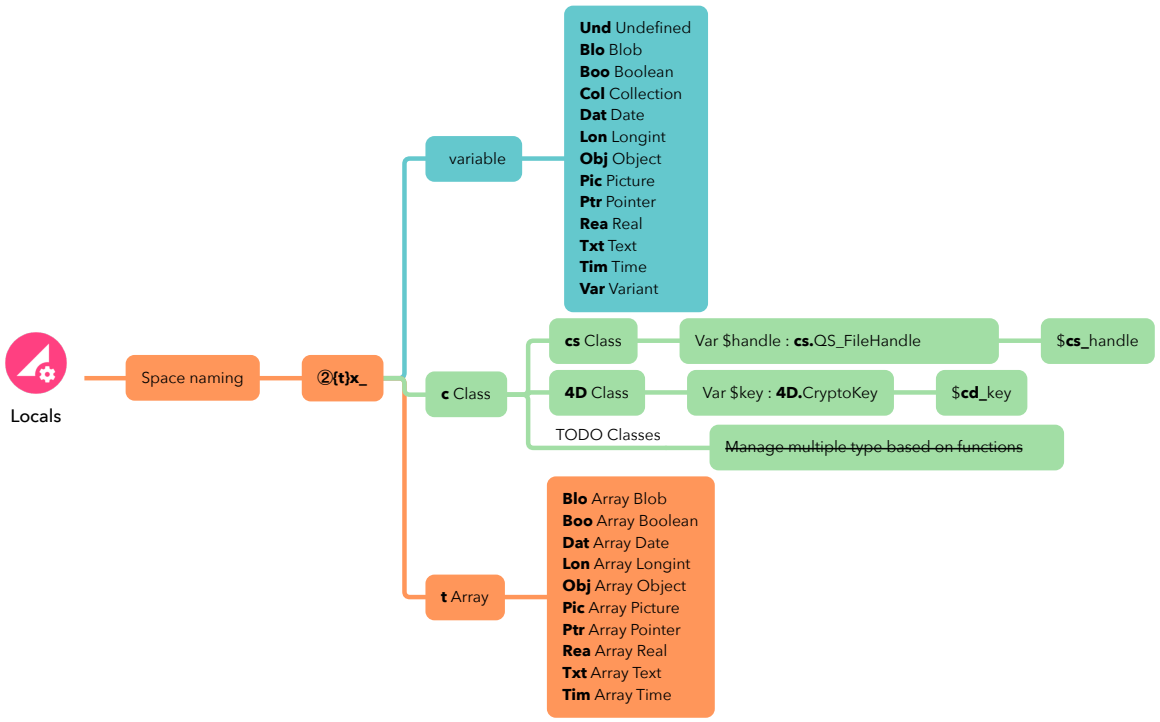

## Naming spaces — « ①ogs_ » — say « ogs »

This space is prefix based used by Protée, and initially the one used by Genie Solutions, Australia. I like the simplicity and shortness of this convention.



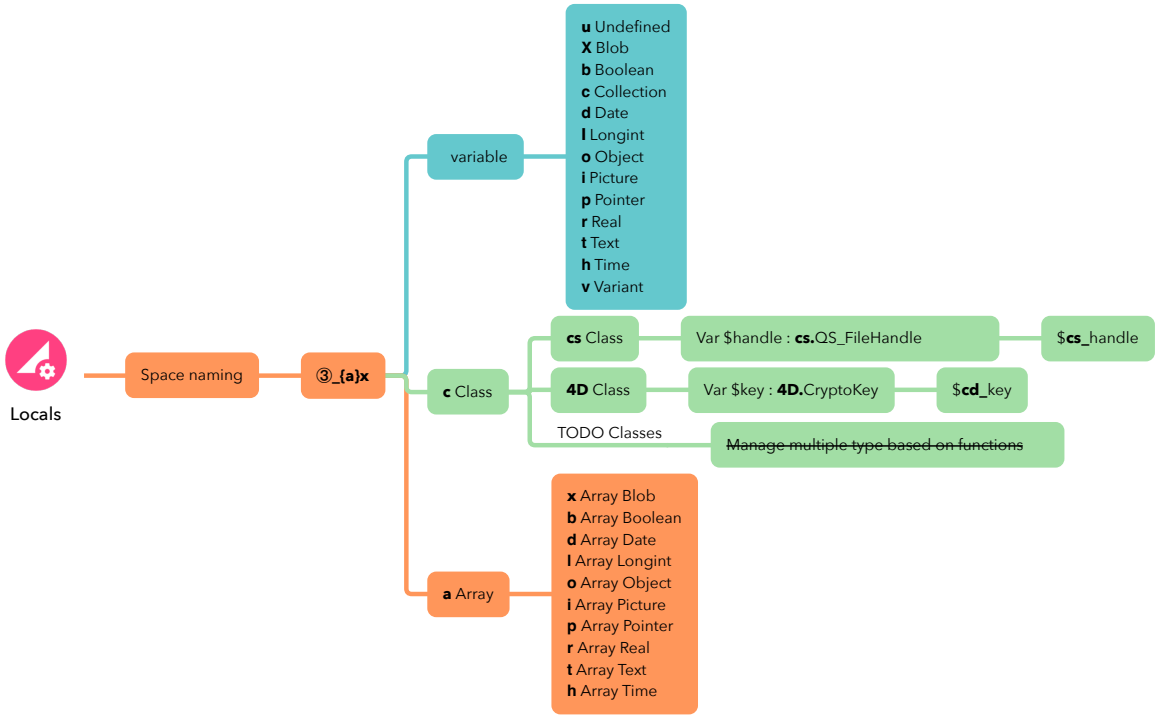
## Naming spaces — « ②{t}x_ » — say « tx »

This space is prefix based and is basically the one used by Vincent De Lachaux.



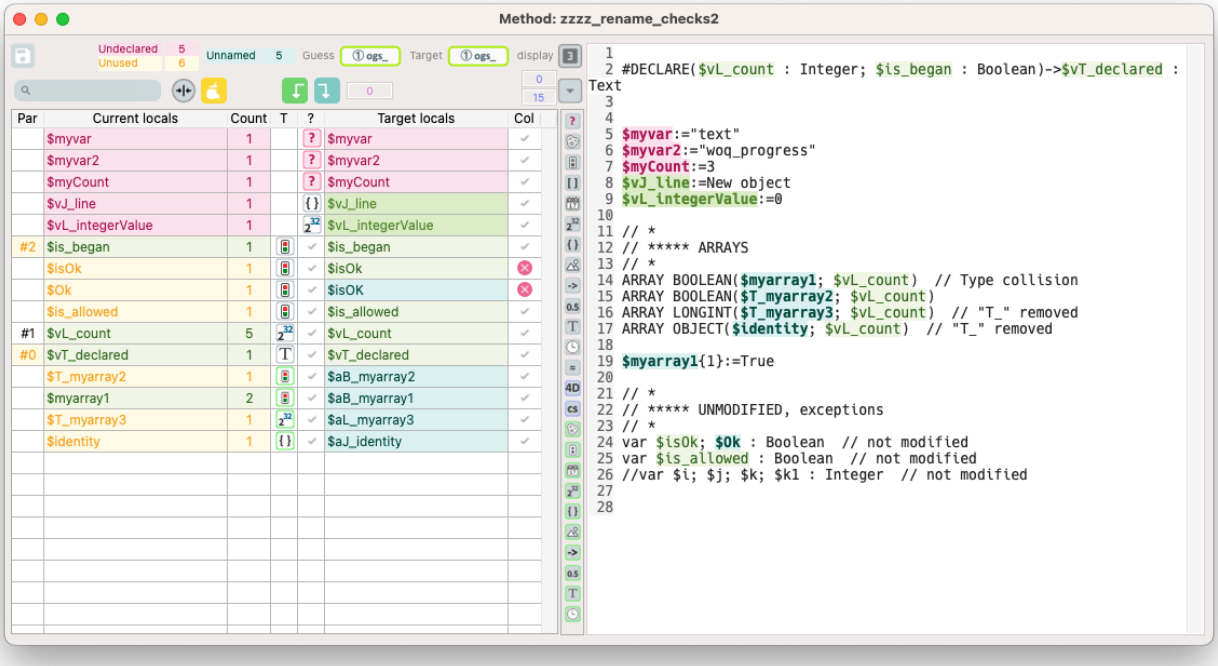

## Naming spaces — « ③_{a}x » — say « ax »

This space is postfix based and is the one used ie by Arnaud de Montard.

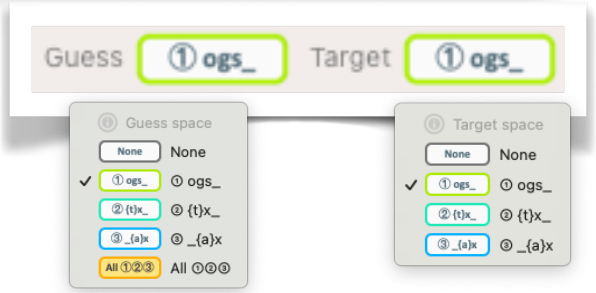


# Method Locals, « Declare! & Rename! »

A « Locals table reference » for all locals in the method is first constructed, and used everywhere in the display.
On the left, it displays the Table reference and on the right, depending on the display mode, variables in the method's context.




First, select the « Guess » and « Target » naming space you want! All the undeclared are sorted at top, and a guessed type icon. All the declared have a type icon at left.




The **Current locals** column lists all the locals:
- in red for the undeclared,
- in yellow for the declared unused,
- in light green for the declared.

In the **Target locals** column, you find:
- for undeclared, in red, or in green if a type has been guessed from its name and the selected « Guess » space. A click on the type icon opens a menu where you can choose or change the type — ie type the unguessed, or the guessed type is wrong.
- for declared, in green when the name comply to the space name, or in teal if the corrector suggest an other name than the current, based on « Target » type.

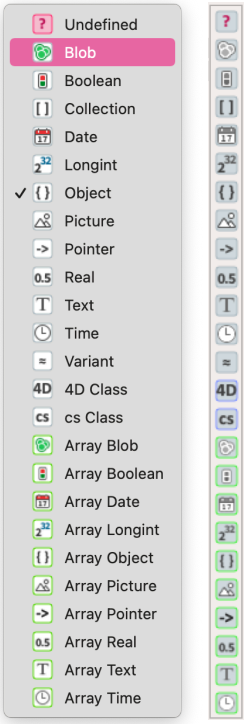
In the **Par** column, you will see the parameters of that method. They are detected that way:
- with **#DECLARE()**, they are display as « #n »
- with a direct affectation like $vT_text:=$n or $0:=$isOk, displayed « $n »

Tip: Unused locals parameters in #DECLARE are never counted as unused, as the programmer is supposed to know what he/she does! But this fact is displayed in yellow in this column to warn the reader (**Current locals** stays as used).

By construction, « #n » are always optional. But for direct affectation, the **Count parameters** command is parsed and the option threshold « t » is calculated based on >= t. Then, optional parameters are displayed « ${n} ».

The **Col** column display where after a rename there will be a collision. The Rename! action will never rename locals if a collision will result, as this situation would be a nightmare to go back to normal : **this is safe**!

## How to use the interface

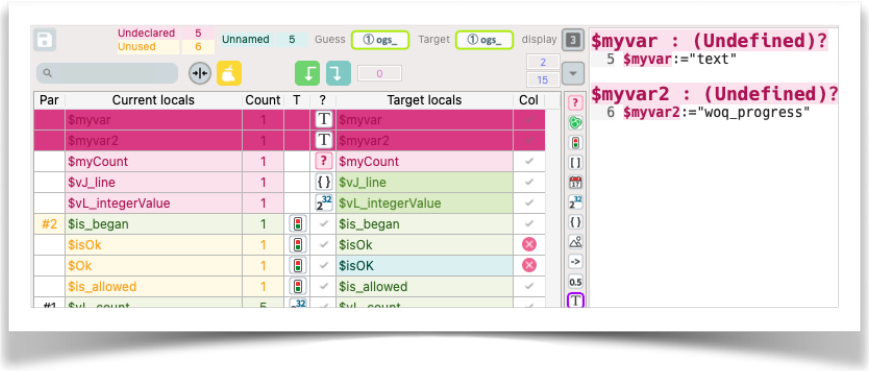



You do a multiple selection, and you click on the proper type button to change the type for all the selected items. Only undefined items will be changed in the selection.
You can also click directly on the type icon in the middle of the listbox to get a menu.

When you select line(s), the zone displayed at right for the method allows you 3 modes of display:

1) Method, all variables highlighted
2) Method, lines match with selection
3) Locals matches (Like that following)

Here, it is easy to type the two undeclared untyped locals: select them, you see there uses at left, and you understand there are « Text ».
You can type one by one clicking in the column on the left that **Target locals**.
Or you can do a selection and use the Type buttons in the right widget:

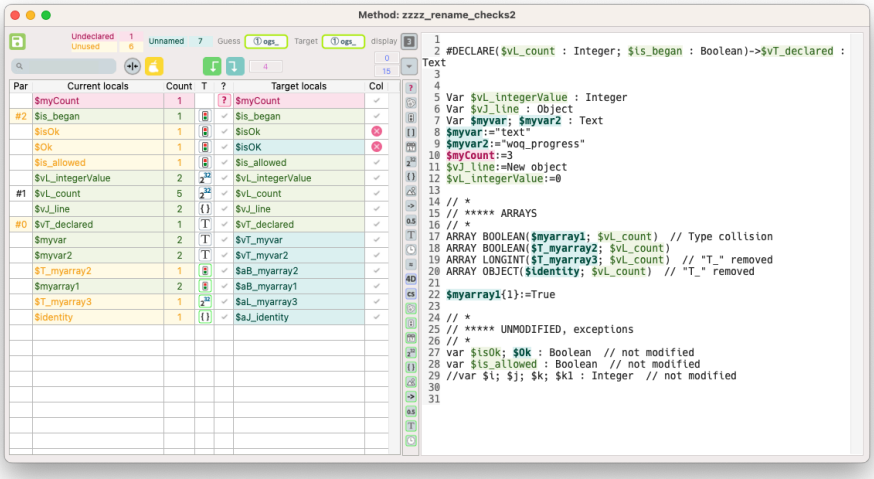



Here, you selected two lines and choose « Text »
So now you can go ahead with two attitudes: quick on all, or slow on the selection with the use of the buttons.

⚠ *IMPORTANT: you can't damage your method! if you ask to declare an already declared local, nothing appends, if you want to rename something with no new proposal nothing append. More than that, if a rename generates a collision (variables that will be merged in one), they keep untouched. You can see this situation with the last right « Col » (collision) column.*

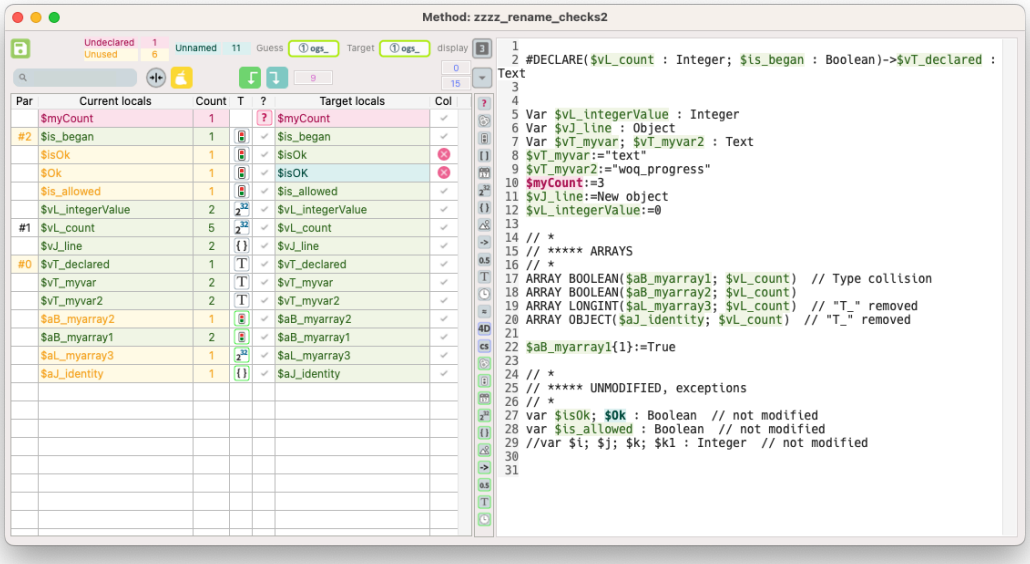
**Quick**

Command D: Declare all what is declarable (undeclared, with a defined guessed type). Here, our two new text variables are in declared texts section. They are in Teal because the target space defined proposes a correct rename.




Now on the Target locals column, you get green for names that are ok, and teal for suggested renames for the selected target space.

Command R: Rename all what is suggested, but let the names in collision untouched. The collisions have to be solved by dirty hands. In the **Target locals** column, before to Rename!, you can repair collision by changing the name to something else, and the collision indicator is re-calculated when you leave the cell (tab or click).

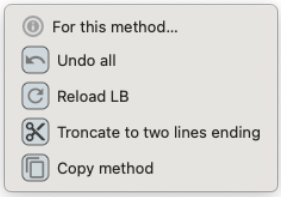



**Slow**
You just select what you want to change and click on the following buttons:

Declare! — or Command D, regardless of the selection

Rename! — or Command R, regardless of the selection

Quick actions




- Undo all restore the method at opening of this panel.
- Reload LB for test purpose, reload the listbox
- Troncate to two lines ending: automatic now
- Copy method copy the actual state of the methodic this Form to the pasteboard

Cancel: button or Cde/Ctrl W
Accept: button or Cde/Ctrl S
If you Cancel, the method is unchanged, whatever you have done on it.
You have to Accept to get the method changed.

This module is available in two places in ogDevTools.
- From the method itself and the macro menu (keyboard shortcut Command $).
- From the « Methods locals » interface, where you see all the « Ud », « Uu » and « Un » in the list, and edit the same way on right.

In short, you have to
- for untyped, choose a type for them, check and keep/change the guessed type,
- do Cde/Ctrl D,
- then Cde/Ctrl R,
- then Cde/Ctrl S to Accept and close.

## Favorites

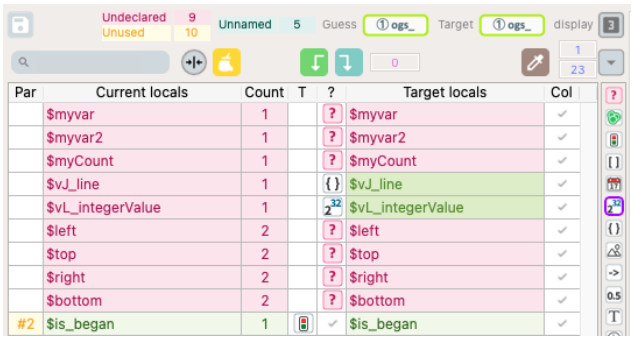
You might have some recurrent and safe variable names you know perfectly the type, even with the naming space not able to guess them, ie with

```
OBJECT GET COORDINATES(*;"myButton"; $left; $top; $right; $bottom)
```

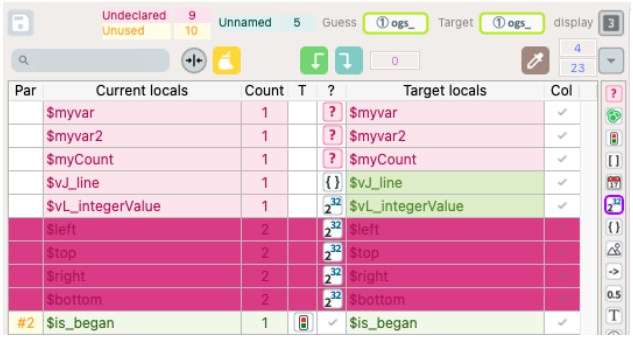
where you always use them in this context (here Longint). To work efficiently, you can manage « favorites » and create exceptions. For that, a tool is available to add or remove {name, type} from the favorites.
It works this way

- only undeclared lines are taken into account. The others, even selected, do not apply
- selecting lines, the variable names are splitted into two sections in the menu
  - the ones not in the favorite's prefs, proposed to add to them
  - the ones already into the favorite's prefs, then proposed to remove from them

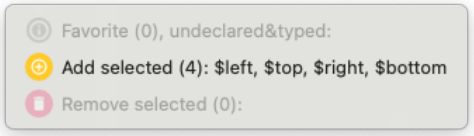
In our case, for the four lines for our locals:



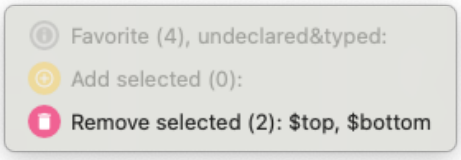

Select them and type them to longint with the icon on the right:




Click the pipette button to get this menu:
Add, and here we are : next time you open a method with those names undeclared, the guess types are automatically set to Longint.




If later on you want to remove from the favorites $top and $bottom, just select those lines, and click again to open the menu and get the option to remove what is already stored:



## Ways of programming

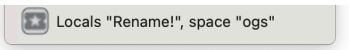
You can use this tool in two major ways, and/or a mixture of them.

A) you do coding providing correct name for locals, based on the chosen naming spaces, then you use the tool to declare, and if you worked well, all undeclared are automatically pre-typed, declarations is a snap!
B) you do coding with just names without any naming convention but you declares the locals, so the tool will propose you to rename to your target naming space. Mmmmh, works but not very elegant, and the proposed types are to be carefully checked before « Declare! ».
C) you declare and give correct typed names, and the tool is just to check and correct mistakes.
D) you open a database written with no naming space or with one implemented, and you do a bulk « Rename! » of your database to the target naming space you like: a must a do!

I love to work the A) way!

The D) way is useful to open databases provided by someone else, or your old code !!! and convert them in your favorite naming space.

## Quick actions




In the quick actions, you can't « declare! » as the type is only guessed based on name, and you might have some miss interpretations, and untyped too, making it too dangerous to just proceed blind eyes. That is why only the « Rename! » action exists, and will rename all declared locals based on space and type. And all eventual collisions will be unchanged, to keep your method safe and with no miss after.

## Limitations

Right now, for classes, functions are not encapsulated, ie you can't have the same variable $toto declared once as text and once as longint, as ogTools will indicate a conflict even in two different Functions.

# Thanks

A big thanks to Genie Solutions and the collaboration we had during 2021, and where I discovered this naming space and I wasn't to guess it will become my habits for future.

Thanks to Patrick Emanuel for its long-term assistance and help during the tests and the regex definitions.

oooOOOooo