

DynamicStructure

Version 1.0 – april 2002

©Osmose Éditeur - techsupport@osmose.net



Osmose Éditeur
33, avenue Jean Monnet
F-13410 Lambesc
Tel.: +33 (0)4 42 92 83 55
Fax.: +33 (0)4 42 92 83 27
<http://www.osmose.net>

DynamicStructure is a plugin that lets the 4D developer get and set dynamically the properties and contents of objects in the Structure.

It is a very powerful tool that is very useful to developers : from automatic documentation to dynamic manipulation of objects, DynamicStructures covers a lot of features that can't be accessed with the 4D language.

Compatibility

DynamicStructure is compatible with the following 4D versions and operating systems :

- 4D 6.7 : Mac OS 8.6, Mac OS 9.x, Windows (98/NT/XP). And Mac OS X in the « Classic » environment
Use at least 4D 6.7.2, but avoid 6.7.3 which has known bugs regarding plugins.
- 4D 6.8 : Mac OS 9.2.2, Mac OS X and Windows (98/NT/XP).
- **Version 6.5:** *The plugin is not supposed to be used under 4D 6.5, and Osmose Éditeur can not offer tech support when the plugin is used with this version of 4D (in version 1.0 of the plugin). We can say anyway that the beta-test has shown that most of the routines are OK with 4D 6.5, and in a near future it is likely that version 6.5 will be officially supported. If you use the plugin under 4D 6.5, please, let us know which routines work, and which don't : we'll fix any problem - if possible - in order to support 4D 6.5. Use at least 4D 6.5.9.*

Installation

Drop the plugin in the appropriate Mac4DX or Win4DX folder.

- 4D 6.7 on Mac OS as on Windows :
 ➔ Use DynamicStructure.4DX (and DynamicStructure.RSR on Windows)
- 4D 6.8
 - On Windows, use DynamicStructure.4DX and DynamicStructure.RSR
 - On Mac (OS 9.2 and OS X), use the carbonised version : DynamicStructure.4CX

Register the plugin - Demo mode

Once installed, the plugin runs in demo mode until you enter a valid license number, using the `ds_Register` routine :

`ds_Register`(selector;licence) -> errorCode

selector	Integer	Kind of license
Licence	String	License number

Possible values for Selector

- 1 : Developer license for Mac OS
- 2 : Developer license for Windows
- 3 : Compiled Runtime Deployment
- 4 : Unlimited For One Application Compiled Deployment

If the license is valid, the routine returns 0 (no error). Else, it returns error code -30000 and still runs in demo mode.

There are 3 kinds of license :

- The *developer license*, which allows the development (for one platform) in interpreted mode, and let the developer do some testing in compiled mode. One license per machine is needed. You can use the same license for every structure file you develop on this machine.
- The *compiled runtime license*, which runs only in compiled mode, and does nothing in interpreted mode. This license is the one you must use if you use DynamicStructure at runtime, compiled. This license goes for one machine too.
- At least, the *unlimited runtime license for one app*, which goes with only one application and for one platform. You need this one if you want to distribute several copies of one compiled application.

Under client/server, you must have one licence per machine that uses the plug-in.

Mac OS = one platform, whatever the OS is : OS 9 or OS X.

Examples :

- A developer uses DynamicStructure only for optimising its developments, doing better technical documentaion, ... DynamicStructure is not used in its developments once installed at his customers's offices.
-> He must buy one *developer license*.
- This developer uses the plugin in some compiled product :

-> He must buy one *developer license* and as much as *runtime compiled* licenses than machines that will use the plugin.

- A developer uses the plug-in on his Mac and on his PC : he must buy 2 *developer licenses* (one per platform).
- DynamicStructure is used with 4D standalone Web Server, in a compiled structure : only one *runtime compiled license* is needed.

For any question, please contact infos@osmose.net, check our web site <http://www.osmose.net>, or ask your distributor.

Limitations of the demo mode are the following :

- Only 20 "Set" and 50 "Get" routines can be used. After that, the plugin does nothing but returning the "not registered" error code.
- Routines returning arrays of information truncates the array
- When no error occurred when calling a plugin's routine, error -30 000 ("not registered") is returned instead of 0.

Example of registration method :

```

C_INTEGER($err)
C_STRING(255;$license)
`
$license:="" ` <- Put your license number here
If (Not(Compiled Application)) ` This is a developer license
  If (wBB_IsMac )
    $err:=ds_Register (1;$license) ` selector 1 = Mac developer license
  Else
    $err:=ds_Register (2;$license) ` selector 2 = Windows developer license
  End if
Else "Compiled Runtime", license
  ` If the license starts with "UNL", it is an Unlimited license for one application". Else, it is
  ` a usual Compiled Runtime Deployment license.
  If ($license="UNL@")
    $err:=ds_Register (4;$license)
  Else
    $err:=ds_Register (3;$license)
  End if
  `
  ` NOTICE : the developer license lets the developer test in compiled mode :
  ` If (wBB_IsMac )
  `   $err:=ds_Register (1;"DEV....")
  ` Else
  `   $err:=ds_Register (2;"DEV....")
  ` End if
  `
End if
If ($err=-30000) ` -30 000 = invalid license
...
End if

```

How to use DynamicStructure - important things

Routines that only return informations (the « get » routines) *never* modify the structure file. So, they can be used freely, except for special routines : for instance, it is not possible to read the database properties from a 4DClient or to get the text of a method in a compiled database.

Very Important Notice :

- Every routine that set an information writes *immediatly* the structure file. Modifications are not restricted to the current process, they are « definitive » until the next change, just like if the object was modified in Design mode.
- Because of that, it is *very important* that the 4D developer takes care himself of the consequences of the modifications he make, and takes care himself of multiple access to an object. It is very important in Client/Server environment and/or if important objets were modified, such as Trigger or a Database method... Because changes are immediatly applied to all the database.
- At last, while developping, you must use copies if you develop anything that - using the plugin - may change the current structure file (ds_TextToMethod, ds_DuplicateForm, ...)

Under Client/Server, modified objects are locked during the modifications. If they were used by another Client, the plugin returns the error : 30200 (Locked Objet). There is no object locking under 4D standalone.

Known plugin problems : 4D interpreter

In interpreted mode, some variables are retyped after a call to a plugin. In their routines definitions, plugins tell 4D what kind of parameters they expect, and 4D changes the type of parameters according to that. This concerns every plugin, not only DynamicStructure.

For instance : the routine `RoutineOfThePlugin` accepts a Long Integer as parameter #1 :

C_REAL(\$aReal)

RoutineOfThePlugin(\$aReal)

=> After the call, \$aReal has become a Long Integer.

You may loose information, since a long integer is 4 bytes long, and a real is 8 bytes long. In fact, if after the call you write in \$aReal a number greater than MAXLONG, 4D will « truncate » the value to fit in a Long Integer (4 bytes).

It could be really problematic in case of array : imagine a routine that expects an integer as parameter. You want to use the current selected element of an array as the parameter, so you write something like :

ARRAY TEXT(theArray ;10)

... later in the code ...

RoutineOfThePlugin(theArray)

=> after the call, theArray became a Long Integer !

The array theArray has become a Long Integer variable : we have lost the acces to its previous content.

In such situation, it could be very usefull to use a temporary buffer :

ARRAY TEXT(theArray ;10)

... later in the code ...

\$buffer := theArray

RoutineOfThePlugin(\$buffer)

Please, note that when a plugin needs an array, and not a fixed-size variable, there is no typecasting : an Integer array doesn't become a Text array or a something else.

Also, note that in compiled mode, there is no problems.

Known limitations of particular routines

(see the documentation of the routines)

4D must be reloaded after using these routines:

ds_NewMethod
ds_SetMethodName
ds_DuplicateForm

Not allowed in compiled environment :

ds_NewMethod
ds_SetMethodName
ds_MethodToText
ds_TextToMethod
ds_DuplicateForm
ds_GetObjectComments
ds_SetObjectComments

Not allowed on 4D Client:

ds_GetDatabaseProperties
ds_SetDatabaseProperties
ds_NewMethod
ds_SetMethodName
ds_DuplicateForm

Not allowed on 4D Server:

ds_NewMethod
ds_SetMethodName
ds_DuplicateForm

About every routine of the plug-in

- All routines of the plugin are functions that return an error code : « errorCode », of type Long Integer. A 0 value means that no problem occurred. Any other value is a system error, a 4D error, or a plugin error (see : Error Codes)
- When the parameter expected is a Numeric Array you can use as you want an Integer Array, a Long Integer Array, a Real Array.

- When the parameter expected is a Text Array you can use as you want a Text Array, or a string Array. If you choose a too small string lenght for the string Array, strings will be troncated to fill the space.
- When the parameter expected is a Numeric you can use as you want an Integer, a Long Integer, or a Real. See the section « The 4D interpreter ».
- The plugin cannot get or set a component's object information, it returns an error instead.

DynamicStructure : Methods

With DynamicStructure, you can manipulate any kind of methods : project, object, form, database, triggers. You usually get the ID of the method to manipulate using one routine of the plugin (ds_GetMethodNames, ds_GetFormMethod, ...), then you manipulate it : get/set the code, get/set the comment, or open the method in design mode.

ds_GetMethodNames(names;filter{;visible{;IDs}}) ← errorCode

Names	Text Array	←	Method names array
Filter	String	→	Name search filter
Visible	Numeric Array or Boolean	←	Visibility
Ids	Numeric Array	←	Methods IDs
ErrorCode	Long Integer	←	Error Code (0 = no error)

Load synchronised Arrays :

- All project methods names
- Their visibility
- Their IDs

The filter parameter allows to load only methods which starts by the filter value.

Arrays Visible and Ids, and Filter are optionals, they may be omit when calling the routine. The visibility of a method is set in Design Mode, when the method editor is opened : use the "Get method properties..." of the "Method" menu. A method that is invisible is not listed in 4D Editors ("Execute method", Quick Report, ...).

Beware: IDs are not the same in an interpreted structure once compiled.

ds_MethodToText(methodID;code) ← errorCode

methodID	Integer	→	Method ID
Code	Text	←	The Code
ErrorCode	Long Integer	←	Error Code (0 = no error)

Returns the Text of the method which ID is passed.

ds_TextToMethod(newCode;methodID) ← errorCode

newCode	Text	→	Method code
methodID	Integer	→	Method ID
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the code of a method which ID is passed.

NOTE : if the method is currently opened in Design Mode, its content is not redrawn in its window. If you close the window, it is the actual text in the window that becomes the code. To update the window, choose « Revert to saved » in the File menu.

ds_OpenMethod(methodName{ ;lineNum) ← errorCode

MethodName	String	→	Method name
lineNum	Integer	→	N° of the line to underline
ErrorCode	Long Integer	←	Error Code (0 = no error)

Open the method in Design Mode. If the line number is valid, the line number lineNum is selected. If the current user doesn't have access to the structure mode or if the method is protected (component) an error is returned. You can only open project methods with this routine. To open an other kind of method, use ds_OpenMethodByID.

ds_OpenMethodByID(methodID{ ;lineNum) ← errorCode

methodID	Integer	→	ID of the method
lineNum	Entier	→	N° of the line to underline
ErrorCode	Long Integer	←	Error Code (0 = no error)

Open the method in Design Mode. If the line number is valid, the line number lineNum is selected. If the current user doesn't have access to the structure mode or if the method is protected (component) an error is returned.

This routines enables you to open method that have "no name", such as Object methods, forma methods, ... Those lds ca be get using other routines of the plugin such as ds_GetFormMethod.

ds_NewMethod (newMethodName{ ;ID})	←	errorCode
newMethodName	String	→ Name of the new method
ID	Integer	→ ID of the method
		←
ErrorCode	Long Integer	← Error Code (0 = no error)

Creates a new method named newMethodName.

ID is the ID of the newly created method. Pass 0 or −1 to force 4D to calculate a unique ID for you. If you pass an other value, 4D will use this value as ID for the method. If the ID is already used by an other method, an error is returned.

IMPORTANT NOTICE : Actual versions of 4D (67/68) don't enable a plugin to inform 4D that a new method has been created : It is important to reload 4D after having created a method this way. What can be done is to create several methods, then quit. Don't change the code of method if the new code includes names of newly created method, the tokenization will not recognize the new methods. The following code does this : creates 50 methods named « aMethod_01 » to « aMethod_50 », each method will contain a comment saying that it was created using DynamicStructure :

```
` Create_50_Methods
C_LONGINT($i;$err;$id)
C_TEXT($prefix;$code;$name)
$i:=0
$prefix:="aMethod_"
$code:="" Created using ds_NewMethod, "+String(Current date)
$code:=$code+" - "+String(Current time;HH MM )+" "+(" "*50)
For ($i;1;50)
  $name:=$prefix+String($i;"00")
  $id:=-1
  $err:=ds_NewMethod ($name;$id)
  If ($err#0)
    ALERT("New method #"+String($i)+" => error #"+String($err))
  $i:=50
Else
  $err:=ds_TextToMethod (" "+$name+$code;$id)
```

```

If ($err#0)
  ALERT("ds_TextToMethod#" + String($i) + " => error #" + String($err))
  $i:=50
End if
End if
End for
,
QUIT 4D
,
,

```

ds_SetMethodName(oldName ;newName) ← errorCode

oldName	String	→	Method that we want to change the name
newName	String	→	New method name
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the name of the method oldName. If oldName is not a valid projet method, or if the newName is an existant method or if newName is an invalid name (empty or more than 31 characters), the routine does nothing an returns an error.

IMPORTANT NOTICE : Actual versions of 4D (67/68) don't enable a plugin to inform 4D that a method name has been modified: It is important to reload 4D after having changed a method name this way.

ds_SetMethodVisible(methodName;isVisible) ← errorCode

methodName	String	→	Name of the method to modify
isVisible	Numérique	→	New visibility (1 = visible, 0 = invisible)
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the visibility of the method methodName. An invisible method doesn't appear in 4D dialogs : « Execute a method », in User mode, formula editor, ...

ds_GetObjectMethodIDs(IDs ;varNames ;formIDs) ← errorCode

Ids	Numeric Array	←	IDs of object methods
varNames	String/Text Array	←	Names of variables
formIDs	Numeric Array	←	Ids of forms

ErrorCode	Long Integer	←	Error Code (0 = no error)
-----------	--------------	---	---------------------------

This routine loads in synchronised arrays informations about all objects methods of the structure file :

- Their IDs (usable with ds_MethodToText fo instance)
- The name of the variable/Field that holds the method
 - For fields, the name
- IDs of forms which contain the variables

NOTICE : with version 1.0 of DynamicStructure fields names are formatted using this convention : number of the table on 3 chars + number of the field on 3 chars. When the table is the table of the form, its number is 0 (it is formatted "000")

Examples :

Field #12 of the table of the form : "000012"

Field #8 of table #36 in a form of an other table : "036008"

It is up to the developer to extract the numbers and build the full name of the field (in version 1.0)

ds_TokenizeStatement(toTokenize;tokens) ← errorCode

toTokenize	Text	→	Text
Tokens	Text	←	Tokens
ErrorCode	Long Integer	←	Error Code (0 = no error)

Tokenize the Text toTokenize. Only one line can be tokenised.

ds_ByteSwapTokens(tokens;swapped) ← errorCode

Tokens	Text	→	OriginalsTokens
Swapped	Text	←	Tokens Byte swapped
ErrorCode	Long Integer	←	Error Code (0 = no error)

Transform a Mac token to a PC token, or a PC token to a Mac token. It makes it possible for you to transport tokens from one plateform to the other.

ds_DetokenizeStatement(tokens;text) ← errorCode

Tokens	Text	→	Tokens
Text	Text	←	Text
ErrorCode	Long Integer	←	Error Code (0 = no error)

Transform a token to as readable Text.

ds_ExecuteToken(tokens) ← errorCode

Tokens	Text	→	Tokens
ErrorCode	Long Integer	←	Error Code (0 = no error)

Execut the tokenised code. To execute a token is faster than calling «EXECUTE ». if you have to execute several times the same instruction, it will be faster to tokenise the expression and then execute the tokens.

DynamicStructure : Forms

ds_GetFormNames(names;tableNum{;IDs{;formKinds{;inputName{;outputName{}}}) ←
errorCode

Names	Text Array	←	Forms name
TableNum	Integer	→	Table number
IDs	Numeric Array	←	Ids of the forms
formKinds	Numeric Array	←	Kinds of the forms
inputName	String/Text	←	Name of the Input form
oututName	String/Text	←	Name of the Output form
ErrorCode	Long Integer	←	Error Code (0 = no error)

Load forms names of the table tableNum.

IDs is optionnal and may not be passed to the routine.

if formKinds is passed, it is filled with the kind of each form. Possible values are :

- 0 : no particular kind
- 1 : Detail form
- 2 : List form
- 3 : Detail form for printing
- 4 : List form for printing

ds_DuplicateForm(tableNum;nameOfOriginal;nameOfCopy) ← errorCode

TableNum	Integer	→	Table number
NameOfOriginal	String	→	Source form name
NameOfCopy	String	→	Created form name
ErrorCode	Long Integer	←	Error Code (0 = no error)

Duplicate the form nameOfOriginal. The copy is placed on the same table. The form method and every object method method are duplicated, likewise form properties (resizing options, event, inherited, ...)

IMPORTANT NOTE : after calling this routine, you must quit 4D, because the list of the forms of the table is not automatically updated.

ds_GetFormEvents(formID;events) ← errorCode

FormID	Integer	→	Form ID
Events	Long Integer	←	List of events
ErrorCode	Long Integer	←	Error Code (0 = no error)

Returns in Events the liste of checked events on the form. Events is return as a Long Integer, every bit tells if the event is checked or not. You must use the 4D constants of the theme « form events» to get the state of a particular event. A value of -1 means « all events checked ».

Sample : how to know is the event « On Load » is checked ?

```
$err :=ds_GetFormEvents(formID;$events)
```

```
if($events ?? On Load)
```

```
    ... On Load is checked ...
```

ds_SetFormEvents(formID;newEvents) ← errorCode

FormID	Integer	→	Form ID
Events	Long Integer	→	New events values
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the checked events of the form. See : ds_GetFormEvents.

ds_GetFormRulers(tableNum;formName;header;detail;break;footer;
moreHeaders;moreBreaks) ← errorCode

TableNum	Integer	→	Table number
FormName	String	→	Form name
Header	Integer	←	Main Header position
Detail	Integer	←	Detail position
Break	Integer	←	Main Break positiposition
Footer	Integer	←	Footer position
MoreHeaders	Numeric Array	←	Other headers
MoreBreaks	Numeric Array	←	Other breaks
ErrorCode	Long Integer	←	Error Code (0 = no error)

Return the values of form rulers. If there are no more than 1 header or 1 footer, the size of the arrays moreHeaders and moreBreaks will be 0.

ds_SetFormRulers(tableNum;formName;header;detail;break;
 footer;moreHeaders;moreBreaks) ← errorCode

TableNum	Integer	→	Table number
FormName	String	→	Form name
Header	Integer	←	Main Header position
Detail	Integer	←	Detail position
Break	Integer	←	Main Break position
Footer	Integer	←	Footer position
MoreHeaders	Numeric Array	←	Other headers
MoreBreaks	Numeric Array	←	Other breaks
ErrorCode	Long Integer	←	Error Code (0 = no error)

Define form rulers. The routine returns an error if you passed bad parameters (Header higher than Footer, ...).

You can't pass more than 9 more breaks and 10 more headers.

ds_GetFormMenubar(tableNum;formName;menuBar) ← errorCode

TableNum	Integer	→	Table number
FormName	String	→	Form name
Menubar	Integer	←	Number of associated menu bar
ErrorCode	Long Integer	←	Error Code (0 = no error)

Get the menu bar associated to a form. A number negative means that the check box « Menu Bar Active » is checked.

ds_SetFormMenubar(tableNum;formName;newMenuBar) ← errorCode

TableNum	Integer	→	Table number
FormName	String	→	Form name
newMenuBar	Integer	→	Number of associated menu bar
ErrorCode	Long Integer	←	Error Code (0 = no error)

Define the menu bar associated to a form. Giving a negative value is as to check the box « Menu Bar Active ».

ds_GetFormMethod(tableNum;formName;methodID) ← errorCode

TableNum	Integer	→	Table number
FormName	String	→	Form name
MethodID	Integer	←	Form method ID (0 = no form method)
ErrorCode	Long Integer	←	Error Code (0 = no error)

Retuns in methodID the ID of the form method associated to the form formName of the table tableNum. If this form doesn't have any form method, methodID is set to 0. After getting the form method ID, you can use ds_MethodToText and ds_TextToMethod.

ds_SetFormMethod(tableNum;formName;newMethodID) ← errorCode

TableNum	Integer	→	Table number
FormName	String	→	Form name
MethodID	Integer	→	Method ID
ErrorCode	Long Integer	←	Error Code (0 = no error)

Change the form method associated to the form FormName.

IMPORTANT NOTICE: The routine doesn't verify that the newMethodID is an existing method ID. To stop associating a from method to a form you just have to pass 0 in methodID. Notice that the old form method, if any and if it was not a valid project method ID, stays in the structure, as an orphan.

ds_GetFormInfos(formID;
 resizable;fixedWidth;fixedHeight;
 heightMini;heightMaxi;hauteurMini;hauteurMaxi;
 hMarginOrWidth;vMarginOrHeight;
 nomObjetDelimiteur>windowTitle;platForm
 inheritedTableNum ;inheritedFormName) ← errorCode

FormID	Integer	→	Form number
Resizable	Integer	←	Resizeable
FixedWidth	Integer	←	Fixed Width

FixedHeight	Integer	←	Fixed Height
HeightMini	Integer	←	height minimum
HeightMaxi	Integer	←	height maximum
HauteurMini	Integer	←	Width minimum
HauteurMaxi	Integer	←	Width maximum
HmarginOrWidth	Integer	←	horizontal Margin or Height
VmarginOrHeight	Integer	←	vertical Marge or Width
NomObjetDelimiteur	String	←	Object delimiting
WindowTitle	String	←	Window Title
PlatForm	Integer	←	Window Style (plateform)
inheritedTableNum	Integer	←	Number of table on inherited form
inheritedFormName	String	←	Inherited form name
ErrorCode	Long Integer	←	Error Code (0 = no error)

This routine makes it possible for you to load most of form properties. Dues to some options are checked or not, some parameters may be negative.

DynamicStructure : Tables

ds_GetTriggerInfos(tableNum;triggerID;onSaveNew;onSave;onDelete;onLoad)

← errorCode

TableNum	Integer	→	Table number
TriggerID	Integer	←	Trigger method ID
OnSaveNew	Integer	←	<div style="display: inline-block; vertical-align: middle;"> <div style="display: inline-block; width: 10px; height: 100px; border-left: 1px solid black; margin-right: 5px;"></div> <div style="display: inline-block; vertical-align: middle;"> Trigger enabled (1) or disabled (0) </div> </div>
OnSave	Integer	←	
onDelete	Integer	←	
OnLoad	Integer	←	
ErrorCode	Long Integer	←	Error Code (0 = no error)

Returns in triggerID the ID the trigger method (0 if there isn't any for this table). This ID can be used with other routines of the plugin (ds_MethodToText, ds_GetObjectComments, ...). If there is no trigger method, triggerID is set to -1.

These values inform you about the activation state of the trigger for some database events. If the value is 1 the trigger is enabled, else if the value is 0 the trigger is disabled. Notice that since the 4D 6.7, those informations (except the trigger method ID) are already accessible using the 4D language.

ds_SetTriggerInfos(tableNum;triggerID;onSaveNew;onSave;onDelete;onLoad)

← errorCode

TableNum	Integer	→	Table number
TriggerID	Integer	→	new Trigger method ID, or 0 or -1.
OnSaveNew	Integer	→	<div style="display: inline-block; vertical-align: middle;"> <div style="display: inline-block; width: 10px; height: 100px; border-left: 1px solid black; margin-right: 5px;"></div> <div style="display: inline-block; vertical-align: middle;"> enabled(1) or disabled(0) the trigger -1 : do not modify </div> </div>
OnSave	Integer	→	
onDelete	Integer	→	
OnLoad	Integer	→	
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify :

- The method to execute when the trigger is called. 0 means « no trigger method »
- The state enable/disable of this or that database event.

A value of -1 in any parameter means « do not modify this parameter ».

IMPORTANT NOTICE: The routine doesn't verify that triggerID is an existing method ID. To stop associating a method to a trigger you just have to pass 0 in triggerID. Notice that the old trigger method, if any and if it was not a valid project method ID, will still exist in the structure, as an orphan.

ds_TablesIDs(IDs) ← errorCode

Ids	Numeric Array	←	List of table IDs
ErrorCode	Long Integer	←	Error Code (0 = no error)

Returns a list of tables IDs. Used only (on this version of the plugin) to get global comments on the table (see ds_GetObjectComments).

ds_SetTableName(tableNum;newName) ← errorCode

TableNum	Entier	→	Table number
newName	Alpha	→	New table name
ErrorCode	Entier Long	←	Error Code (0 = no error)

Changes the name of the table number tableNum.

Special: the change is immediately available everywhere but in the explorer in Design Mode.

ds_SetTableVisible(tableNum;isVisible) ← errorCode

tableNum	Integer	→	Table number
isVisible	Integer	→	Set visible (1) or invisible(0)
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the visibility of the table tableNum.

To know if a table is visible, use 4D Pack (This plugin is shipped with 4D).

ds_SetTableDeletion(tableNum;physicallyDeleted) ← errorCode

tableNum	Integer	→	Table number
physicallyDeleted	Integer	→	physically deleted(1) or no(0)

ErrorCode Long Integer ← Error Code (0 = no error)

Modify the state « physically Deleted » of the table.

To know if a table is visible, use 4D Pack (This plugin is shipped with 4D).

ds_GetTablePosition(tableNum;top;left;bottom;right) ← errorCode

TableNum	Integer	→	Table number
Top	Integer	←	Higher point position
Left	Integer	←	Left point position
Bottom	Integer	←	Lower point position
Right	Integer	←	Right point position
ErrorCode	Long Integer	←	Error Code (0 = no error)

Return the position (in the structure window) of the table TableNum.

ds_SetTablePosition(tableNum;top;left;bottom;right) ← errorCode

TableNum	Integer	→	Table number
Top	Integer	→	Higher point position
Left	Integer	→	Left point position
Bottom	Integer	→	Lower point position
Right	Integer	→	Right point position
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the position (in the structure window) of the table TableNum.

ds_GetTableColor(tableNum;isAutomatic;colorIndex) ← errorCode

TableNum	Integer	→	Table number
IsAutomatic	Integer	←	Default color(1) or no(0)
ColorIndex	Integer	←	Color Index if isAutomatic = 0
ErrorCode	Long Integer	←	Error Code (0 = no error)

Returns informations about the table color (in the structure window). If isAutomatic is 1, the Color is « Default ». Else, colorIndex is the color in the 256 colors palette of 4D.

Numbers start at 1 and end at 255.

ds_SetTableColor(tableNum;automatic;newColorIndex) ← errorCode

TableNum	Integer	→	Table number
IsAutomatic	Integer	→	Default color(1) ou no (0)
ColorIndex	Integer	→	Color
ErrorCode	Long Integer	←	Error Code (0 = no error)

Define informations about the table color (in the structure window). If isAutomatic is 1, the Color is « Default ». Else, colorIndex is the color in the 256 colors palette of 4D. Numbers start at 1 and end at 255.

ds_GetFieldColor(tableNum;fieldNum; isAutomatic;colorIndex) ← errorCode

TableNum	Integer	→	Table number
FieldNum	Integer	→	Field number
isAutomatic	Integer	←	Default color(1) ou no (0)
ColorIndex	Integer	←	Color
ErrorCode	Long Integer	←	Error Code (0 = no error)

Returns informations about the field color (in the structure window). If isAutomatic is 1, the Color is « Default ». Else, colorIndex is the color in the 256 colors palette of 4D. Numbers start at 1 and end at 255.

ds_SetFieldColor(tableNum;fieldNum;automatic;newColorIndex) ← errorCode

TableNum	Integer	→	Table number
FieldNum	Integer	→	Field number
isAutomatic	Integer	→	Default color(1) ou no (0)
ColorIndex	Integer	→	Color
ErrorCode	Long Integer	←	Error Code (0 = no error)

Changes informations about the field color (in the structure window). If isAutomatic is 1, the Color is « Default ». Else, colorIndex is the color in the 256 colors palette of 4D. Numbers start at 1 and end at 255.

ds_SetFieldAttributes(tableNum;fieldNum;isMandatory;isEnterable;noModif;
indexUnique;invisible) ← errorCode

TableNum	Integer	→	Table number
FieldNum	Integer	→	Field number
isMandatory	Integer	→	1 = mandatory, 0 = not mandatory
isEnterable	Integer	→	1 = enterable, 0 = not enterable
noModif	Integer	→	1 = is modifiable, 0 is not modifiable
IndexUnique	Integer	→	1 -> field is indexed-unique
Invisible	Integer	→	1 -> Set invisible
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the field attributes. Current values can be get with 4D commands. It is not possible to change the attributes of a field of kind subTable.

ds_SetFieldRelation(tableNum;fieldNum;relatedTable;relatedField) ← errorCode

TableNum	Integer	→	Table number
FieldNum	Integer	→	Field number
RelatedTable	Integer	→	Related table number
RelatedField	Integer	→	Related field number
ErrorCode	Long Integer	←	Error Code (0 = no error)

Change relation of the field.

If the types are invalids, the routine does nothing (ie, it is impossible to link a Long Integer field with a field which is of an other type).

ds_SetFieldProperties(tableNum;fieldNum;newName;newType;
newStringSize;newList) ← errorCode

TableNum	Integer	→	Table number
FieldNum	Integer	→	Field number
NewName	String	→	New field name. "" = don't change
NewType	Integer	→	New type
NewStringSize	Integer	→	Length (2-80) for String fields. -1 = don't change
NewList	String	→	Associated list name. "" = don't change
ErrorCode	Long Integer	←	Error Code (0 = no error)

Change a field properties.

A subtable field cannot be changed, and it is not allowed to change a « regular » field to a subtable.

If newName is not "" and is already used by an other field of the table, the routines returns an error code and does nothing.

Pass in newType a value of the theme « Constants : Field and Variable Types » :

0	Is Alpha Field
1	Is Real
2	Is Text
3	Is Picture
4	Is Date
6	Is Boolean
8	Is Integer
9	Is LongInt
11	Is Time
30	Is BLOB

DynamicStructure : Menus

ds_MenuBarsID(IDs) ← errorCode

Ids	Numeric Array	←	Menu bars IDs
ErrorCode	Long Integer	←	Error Code (0 = no error)

Return the list of menu bars IDs. The array is synchronised with the bar number. This ID can used with other routines of the plugin (ds_GetObjectComment, ds_MenusIDs, ...)

ds_MenusIDs(menuBarID;menuIDs) ← errorCode

menuBarID	Long Integer	→	menubar ID
menuIDs	Numeric Array	←	array of menu IDs
ErrorCode	Long Integer	←	Error Code (0 = no error)

Returns the list of the IDs of the menus of the menubar ID menuBarID.

ds_GetMenuBarPICT(menuBarNum;menuBarPict) ← errorCode

menuBarNum	Long Integer	→	menubar range
menuBarPict	Picture	←	Menu bar picture
ErrorCode	Long Integer	←	Error Code (0 = no error)

Get the picture of the menu bar menuBarNum.

If no image is found for this menu bar, menuBarPict is returned empty. This means that 4D will draw it's own picture, wkich is stored in its own resource as PICT ID 5002.

ds_SetMenuBarPICT(menuBarNum;newPict) ← errorCode

menuBarNum	Long Integer	→	menubar range
newPict	Image	→	New picture for this bar.
ErrorCode	Long Integer	←	Error Code (0 = no error)

Set the picture of the menu bar menuBarNum. If you pass an empty picture, the previous pictyure (if any) is removed, so that 4D will draw its own picture.

ds_GetMenuInfos(menuBarNum;menuRange;menuTitle;labels;methods) ←
 errorCode

menuBarNum	Long Integer	→	menubar range
menuRange	Long Integer	→	Menu range
MenuTitle	Text	←	Menu title
labels	Text Array	←	Labels
methods	Text Array	←	Methods
ErrorCode	Long Integer	←	Error Code (0 = no error)

Returns informations about the menu number menuRange in the menu bar number menuBarNum. Labels and methods are synchronised arrays returning item labels and 4D associated methods.

ds_SetMenuInfos(menuBarNum;menuRange;menuTitle;labels;methods) ←
 errorCode

menuBarNum	Long Integer	→	menubar range
menuRange	Long Integer	→	Menu range
MenuTitle	Text	→	New menu title (" " = do not modify)
labels	Text Array	→	Labels
methods	Text Array	→	Methods
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify informations of the menu number menuRange is the menu bar number menuBarNum.

IMPORTANT NOTICE : Labels and methods must be synchronised and be exactly of the same size as the originals. It is not possible, with this routine, to increase or to decrease the number of items of the menu.

DynamicStructure : Database

ds_GetDatabaseProperties(selector;value) ← errorCode

Selector	Long Integer	→	Selector
Value	Long Integer	←	value
ErrorCode	Long Integer	←	Error Code (0 = no error)

Return the value of a database parameter. These informations are the one that the developer can modify in the « Database properties » dialog.

Possible values of the selector are defined in the plugin as constants :

Constant	Value	Returned value
kds_StartupEnvironment	1	1 = Start in Design mode 2 = Start in User mode mode 3 = Start in Custom menus mode
kds_ProgressIndicator	9	0 = Number 1 = Thermometers
kds_DragDropHighLight	10	0 = No effect 1 = Frame only 2 = Pattern 3 = Both
kds_MandatoryLogFile	5	0 = Use Log File 1 = Dont use Log File
kds_DeletionControl	3	0 = Disable Deletion Control 1 = Enable Deletion Control
kds_AutomaticTransaction	4	0 = Disable automatic transactions 1 = Enable automatic transactions
kds_ArobaselsChar	11	0 = Don't use @ as a char 1 = Use @ as a char
kds_4DOpenAllowed	2	0 = Disable 4D Open connexions 1 = Enable 4D Open connexions
kds_UserList	6	0 = Dont show user list 1 = Show user list
kds_SortUserList	7	0 = Don't sort user list 1 = Sort user list

kds_FlushData	8	Delay to flush buffers, in ticks
kds_CacheMin	12	Minimum cache
kds_CacheMax	13	Maximum cache
kds_UseNewMemOnMac	14	0 = Disable new memory scheme on Mac 1 = Enable it
kds_WinBlockCount	15	Coun of memory blocks (Windows only)
kds_WinOneBlockSize	16	One memory block soze (Windows only)

Calling this routine on Mac with selectors kds_WinBlockCount or kds_WinOneBlockSize generates an error.

ds_SetDatabaseProperties(selector;newValue) ← errorCode

Selector	Long Integer	→	Selector
newValue	Long Integer	→	New Selector value
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the value of a database parameter.

See ds_GetDatabaseProperties for the possible selector values.

« kds_ArobaselsChar » value can't be set.

Calling this routine on Mac with selectors kds_WinBlockCount or kds_WinOneBlockSize generate an error.

ds_GetDatabaseMethodIDs(onStartUp;
 onServerStart;
 onExit;
 onServerShutDown;
 onServerOpenConnexion;
 onWebConnexion;
 onServerCloseConnexion;
 onWebLog) ← errorCode

OnStartUp	Integer	←	Database « On startup » method ID
OnServerStart	Integer	←	Database « On Server Startup » method ID

OnExit	Integer	←	Database « On exit » method ID
OnServerShutDown	Integer	←	Database « On Server shut down » method ID
OnServerOpenConnexion	Integer	←	Database « On Server Open Connexion » method ID
OnWebConnexion	Integer	←	Database « On Web Connexion » method ID
OnServerCloseConnexion	Integer	←	Database « On Server Close Connexion » method ID
OnWebLog	Integer	←	Database « On Web Log » method ID
ErrorCode	Long Integer	←	Error Code (0 = no error)

Get all database methods IDs.

A value of -1 means « No Such Database method ».

```

ds_SetDatabaseMethodIDs( onStartUp;
                        onServerStart;
                        onExit;
                        onServerShutDown;
                        onServerOpenConnexion;
                        onWebConnexion;
                        onServerCloseConnexion;
                        onWebLog) ←   errorCode

```

OnStartUp	Integer	→	New database « On startup » method ID
OnServerStart	Integer	→	New database « On Server Startup » method ID
OnExit	Integer	→	New database « On exit » method ID
OnServerShutDown	Integer	→	New database « On Server shut down » method ID
OnServerOpenConnexion	Integer	→	New database « On Server Open Connexion » method ID
OnWebConnexion	Integer	→	New database « On Web

OnServerCloseConnexion	Integer	→	Connexion » method ID New database « On Server Close Connexion » method ID
OnWebLog	Integer	→	New database « On Web Log » method ID
ErrorCode	Long Integer	←	Error Code (0 = no error)

Set the database methods IDs.

–1 means « no Database method ». Be careful: if you pass –1 as database method, the previous code will be not be deleted and will become, if any and if the ID is not a Project method, « orphan» in the structure.

```
ds_GetStyleSheets(names;macFontNames;macFontSizes;macFontFaces;
                    ntFontNames;ntFontSizes;ntFontFaces,
                    winFontNames;winFontSizes;winFontFaces) ← errorCode
```

Names	Text Array	←	Style sheets names
MacFontNames	Text Array	←	Mac fonts names
MacFontSizes	Numeric Array	←	Mac fonts sizes
MacFontFaces	Numeric Array	←	Mac fonts faces
NtFontNames	Text Array	←	WinNT fonts names
NtFontSizes	Numeric Array	←	WinNT fonts sizes
NtFontFaces	Numeric Array	←	WinNT fonts faces
WinFontNames	Text Array	←	Win98 fonts names
WinFontSizes	Numeric Array	←	Win98 fonts sizes
WinFontFaces	Numeric Array	←	Win98 fonts faces
ErrorCode	Long Integer	←	Error Code (0 = no error)

Get all Style sheets and their specifications. Arrays are synchronised.

```
ds_SetStyleSheet(name;newName;
                  macFontName;macFontSize;macFontFace;
                  ntFontName;ntFontSize;ntFontFace;
                  winFontName;winFontSize;winFontFace) ← errorCode
```

Name	String	→	Style sheet to modify
NewName	String	→	New name for the style sheet
MacFontName	String	→	Mac font name
MacFontSize	Integer	→	Mac font size
MacFontFace	Integer	→	Mac font face
NtFontName	String	→	WinNT font name
NtFontSize	Integer	→	WinNT font size
NtFontFace	Integer	→	WinNT font face
WinFontName	String	→	Win98 font name
WinFontSize	Integer	→	Win98 font size
WinFontFace	Integer	→	Win98 font face
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the Style sheet properties.

ds_GetFilters(names;values) ← errorCode

Names	Text Array	←	Names of ormats/filters
Values	Numeric Array	←	Format/filters values
ErrorCode	Long Integer	←	Error Code (0 = no error)

Get the list of all formats/filters and their values in synchronised arrays.

ds_SetFilter(filterName;newName;newValue) ← errorCode

filterName	String/Text	→	Format name/filter to modify
Newname	String/Text	→	New format name/filter name
Value	String/Text	→	Filter value
ErrorCode	Long Integer	←	errorCode

Modify the format/filter filterName.

IMPORTANT NOTE : filters are stored *by name* in forms. Changing a filter's name will not be dispatched in every form that uses the form.

DynamicStructure : Miscellaneous

ds_GetListNames (names{;IDs})		←	errorCode
Names	Text Array	←	List names Array
Ids	Numeric Array	←	Lists IDs Array
ErrorCode	Long Integer	←	Error Code (0 = no error)

Get name et les ID of every list of the structure.

IDs is optionnal and may not be passed to the routine.

ds_GetListUserModifiable (listID;isModifiable)		←	errorCode
ListID	Integer	→	List ID
IsModifiable	Integer	←	1 = List is modifiable
ErrorCode	Long Integer	←	Error Code (0 = no error)

Return 1 in isModifiable if the list ID listID is modifiable, 0 in other case.

ds_SetListUserModifiable (listID;isModifiable)		←	errorCode
ListID	Integer	→	List ID
IsModifiable	Integer	→	1 = List is modifiable, 0 = not modifiable
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the status of a list (modifiable or not).

ds_GetComponentNames (tabNoms;tabIDs)		←	errorCode
TabNoms	Text Array	←	Components names array
TabIDs	Numeric Array	←	Components IDs array
ErrorCode	Long Integer	←	Error Code (0 = no error)

Get all installed component names and IDs.

ds_GetObjectComments(selector;id;fieldID;comment) ← errorCode

Selector	Integer	→	Selector
Id	Integer	→	Object ID
FieldID	Integer	→	Field number
Comment	Text	←	Comment
ErrorCode	Long Integer	←	Error Code (0 = no error)

Return the object comment. It is a global comment, written in insider or in 4D.
Selector values are :

kds_DataBaseComment	0
kds_GroupComment	1
kds_ObjectComment	2
kds_TableComment	3
kds_FieldComment	4
kds_FormComment	5
kds_MenuBarComment	6
kds_MenuComment	7

The ID expected is the one returned by other routines of the plugin such as ds_GetMethodNames, ds_GetFormNames, ds_TableIDs, ...

To get a field comment, you must pass the table ID and the field number.

ds_SetObjectComments(selector;ID;fieldID;newComment) ← errorCode

Selector	Integer	→	Selector
Id	Integer	→	Object ID
FieldID	Integer	→	Field number
Comment	Text	→	Comment
ErrorCode	Long Integer	←	Error Code (0 = no error)

Add a comment to an object. To remove the comment, pass an empty string.

See `ds_GetObjectComments` to have the different possible values for the selector.

IMPORTANT NOTICE : It is not possible to pass stylised text to this command. If the previous comment was stylised the style is lost.

ds_GetTipList(tipNames;tipIDs) ← errorCode

TipNames	Text Array	←	Tips Array
TipIDs	Numeric Array	←	Tips Ids Array
ErrorCode	Long Integer	←	Error Code (0 = no error)

Get Tips list and IDs.

ds_SetTip(tipID;newName;newValue) ← errorCode

TipID	Integer	←	Tip ID
NewName	String	→	New name
newValue	String	→	New text
ErrorCode	Long Integer	←	Error Code (0 = no error)

Modify the tip.

DynamicStructure : Errors

-15000	Bad Array Type
-30000	Not Registered
-30001	Time Has Bombed
-30002	Bad Version Of 4D
-30003	Need At Least 67
-30004	Could Not Load Struct
-30005	Not On Windows
-30006	Not On Mac
-30007	Not On 4D Client
-30008	Not On Compiled
-30009	Bad Selector
-30100	Out Of Range Parameter
-30101	Parameter Error
-30200	Object Is Locked
-30201	Invalid Name
-30300	Bad Style Sheetl D
-30301	Method Name Exists
-30302	Method Name Not Found
-30303	No Flow Chart
-30304	Invalid String Field Size
-30305	Invalid Field Kind
-30306	Get Var Only Compiled
-30307	Form Not Found
-30308	Error Reading Form
-30309	Bad Rulers Params
-30310	Duplicate Form Name
-30311	Unknown Form Version
-30312	Form Parser Error
-30313	ID Already Used
-30400	Can't Change Component Object
-30401	Object Not Found
-30402	Cant Get ComponentObject