Deprecated and Removed Features

Rev. January 2017

About this manual

For over 30 years, our main goal has been to enhance our product (using new concepts and technologies) while ensuring the compatibility of 4D applications. As early adopters of 4D know, we have always put a lot of effort into compatibility, and we can find 4D applications all over the world that were created years and years ago, with old versions of 4D and of an OS, that are still working with the latest revisions of the product.

Unfortunately, it sometimes becomes too difficult to mix old technologies with new:

- 4D must bring new technologies, new APIs, and new paradigms to developers.
- OSes change every day, and sometimes deprecate their own old APIs

This is why 4D sometimes needs to tag some commands and features as deprecated, meaning that one day, they will be removed from the language in a future major version.

Letting developers know what is deprecated and what kind of replacement can be used instead makes it much more comfortable for them to implement the change in their code: There is no urgency, no pressure, and the developer has plenty of time to make any necessary changes.

We start by covering all the functions that are currently deprecated or that have been removed in 4D v16 R2, followed by a list of deprecated commands along with their current status. We also provide a quick summary table. You can also access the equivalent documents generated for previous major versions of 4D.

Deprecated or removed features in v16 product range

About 4D 64-bit versions

64-bit versions of 4D favor recent technologies and usually do not support those that have been declared obsolete in previous versions of 4D. For a complete list of functions that are not supported in the 64-bit product range of 4D, please refer to the **Specific features of 64-bit versions** section in the *Design Reference* manual.

XSLT commands deprecated

The XSLT language, inspired by functional programming concepts, transforms XML data to any format (XML, HTML, or any other type). All major Web browsers as well as 4D software have implemented the XSLT 1.0 specification.

Currently the XSLT trend is in decline because developers consider that it is difficult to use and debug. Following this trend as well as feedback from developers, we decided that the XSL transformation feature will not be developed for 4D 64-bit versions. However, to support our customers still using XSLT in 4D, we made the choice to rely on the PHP XSL library, which provides a comprehensive API allowing you to perform all operations necessary for your XSL transformations. This library is an efficient tool which can easily replace the **_o_XSLT APPLY TRANSFORMATION**, **_o_XSLT SET PARAMETER** and **_o_XSLT GET ERROR** commands after their removal. 4D has produced a specific document to help you use PHP XSL as a replacement for the 4D XSLT commands: Download XSLT with PHP technical document (PDF).

We also suggest that you consider using 4D tags when dealing with the dynamic generation of HTML pages, since in most cases it is easier if you handle HTML code as unformatted text (see also the **PROCESS 4D TAGS** command).

Note: Trends of XSLT search in Google: #command_6

For compatibility, XSL transformations are still supported in 4D, but their use is now discouraged. Support for XSLT processing will be removed in future 4D releases.

Note for 64-bit versions: XSLT is not available with 4D 64-bit (OS X and Windows) and 4D Server 64-bit for OS X. Consequently, calling one of the XSLT commands from these applications will generate an error 33 "Unimplemented command or function".

Pictures in PICT format

The PICT format will not be supported in the next major releases of 4D and you must no longer use it in 4D. The **GET PICTURE FORMATS** command helps you detect and filter pictures using the PICT format in your data file (the **_o_AP Is Picture Deprecated** function of 4D Pack is now obsolete).

Note: The Mac "PICT" format has been deprecated by Apple since several prior Mac OS versions (see the description of PICT format on Wikipedia).

The 'PICT' format is a very old Mac format. Prior to version 11, 4D stored all pictures in this format, even on Windows. The PICT format has been deprecated since QuickDraw was deprecated in 2005.

There is one important thing to understand about PICT. It can store (read "encapsulate") 2 main kinds of information:

- the drawing primitives themselves (either bitmap or vectorial), or
- a more modern format (JPEG for example) stored in a PICT using QuickTime. (Usually the developer was calling **_o_QT COMPRESS PICTURE** with the QT Photo compressor constant).

This means that even back when all pictures stored in the data files were PICT, those PICTs could, in fact, contain JPEGs (or other formats). It's important for our customers to stop using PICT, not only because it is obsolete, but

also because 4D needs Altura (+ QuickTime if **_o_QT COMPRESS PICTURE** was used) to read PICT on Windows. This is not efficient, and it requires QuickTime to be installed.

When migrating data from versions prior to v11, developers should apply the **CONVERT PICTURE** command to every picture field of the data. When converting data from more recent versions, we recommend using the **GET PICTURE FORMATS** command to find pictures in your data file that need to be converted.

Detecting PICT format in your database structure

Starting with v16, you can detect pictures that use the deprecated PICT format in your database structure by means of the **Maintenance and security center** (**MSC**). When you use the **Verifying the application** feature, the log file produced includes warnings indicating any pictures found which use or contain the PICT format. These warnings may concern static pictures, as well as pictures found in the picture library or in form objects.

Note: It is up to you to either remove or replace pictures that use the deprecated PICT format. Using the **MSC** to perform a **Repair the structure file** operation does not have any effect on "deprecated" pictures and the same warnings will appear in its log file.

QuickTime

Support for the picture codecs related to QuickTime is now obsolete.

By default, the use of QuickTime is disabled since 4D v14. However for compatibility reasons, you can enable it using the new QuickTime support option of the **SET DATABASE PARAMETER**, **Get database parameter** commands (except in 64-bit versions of 4D, where QuickTime is not supported).

QuickTime image formats under Windows

For several years, image handling under the Windows version of QuickTime has not evolved (only the video part is evolving). We plan to remove support for these specific APIs in the next release.

4D for Windows natively supports all major formats (JPEG, PNG, GIF, TIFF, etc.), and also supports WIC (Windows Imaging Component). If, in your data, you have some pictures saved–under Windows–in a specific format known only by QuickTime, you can convert them (**CONVERT PICTURE**).

We also remind you that the support for QuickTime picture formats has been removed from the 64-bit version of 4D Server for Windows as of 4D v12.

Dynamic assignment of variables received through HTTP

In previous versions of 4D, the Web server automatically recopied the value of variables sent through a Web form or a URL into 4D variables when they had the same name.

For reasons of optimization and control, this principle is not maintained starting with 4D v14: the value of Web variables are no longer automatically assigned to the 4D variables. To recover variables sent using a POST or a GET, you must use the **WEB GET VARIABLES** command exclusively. To recover the posted files, you must use the **WEB GET BODY PART**/**WEB Get body part count** commands.

Note: Dynamic assignment is also disabled by default in 4D databases created beginning with version 13.4.

However, for compatibility, this mechanism is maintained by default in databases created with a version of 4D earlier than 13.4. In this case, you can disable it using the **Automatic variable assignment** compatibility option on the Compatibility page of the Database Settings.

Since this mechanism is obsolete, we strongly recommend that you uncheck this option in your converted databases (and adapt your code if necessary) so as to facilitate future evolutions.

Mac OS QuickDraw fonts no longer supported

QuickDraw fonts (e.g. Geneva, Chicago) are now deprecated and you should no longer use ID numbers to designate fonts. The **_o_Font number** and **_o_Font name** commands are kept in 4D v15 and higher for compatibility but will be removed in subsequent versions. The **OBJECT SET FONT** command now only accepts font names.

Altura Mac2Win

Altura Mac2Win was used to port 4D to Windows. It is a set of APIs that helped porting Mac OS (pre OS X) code to Windows, by translating APIs: filesystem, QuickDraw, Resources, PICT, etc. It was very useful and helped a lot (Mac plug-in developers, for example, could move their plug-ins to Windows more easily), but it translates old (read “deprecated”) Mac OS APIs, and doesn’t use modern native Windows APIs: 4D must remove Mac2Win from its code as much as possible. This is very long and hard work, and in each version of 4D, some dependencies are removed (and replaced by modern APIs).

Right now, 4D still depends on it in part, mostly to be able to handle compatibility of old databases: Resources, PICT, part of the user events handling, support for third party plug-ins that are built using Altura, etc.

By removing resources in the .RSR file to separate files in the “Resources” folder, and by converting (**CONVERT PICTURE**) to not-PICT, 4D developers will be ready once 4D has removed Altura. But the first people concerned by this huge step are plug-in developers. They must stop using Altura as soon as possible, which means they must rewrite some parts of their Windows source code. (We have already been warning them for several years now.)

Subtables

Over several major versions, 4D has warned developers against the use of subtables and since 4D v11, it is no longer possible to create a field of the SubTable type. Subrecords have several known limitations. For example, they are always loaded in memory; they are not handled by the **SEND RECORD** or **DUPLICATE RECORD** commands.

We do not plan to remove support for subtables in the near future, but it’s really time for developers to convert their subtables to regular N-> tables because we do plan to remove them in a future major version of 4D. Developers who used subtables for performance reasons (certain specific situations where loading related records was slow) can be reassured, especially since v12: the speed is here and using classic N<->1 relations is very fast.

Basically, there are two main ways to remove subtables (note: the following is not a full tech tip; just a quick overview):

- Before conversion from a pre-v11 structure: in 2004, create the appropriate N table and the ID field in the 1 table (if not already there). Then change the code everywhere it is needed (see below).
- After conversion: in this situation, 4D has replaced the subtable with a N table using a special relation, that allows the language to work with the subselection and the subrecords. The 4D developer needs to remove this special relation, replace it with a normal relation and change the code everywhere if it is needed (see below).

What we mean by “change the code everywhere if it is needed” is, basically:

- Create the new forms, update included forms
- In the methods (project, form, object, etc.):
  - Replace all commands of the “SubRecords” theme with the corresponding Selection or Record command (for example, replace **_o_CREATE SUBRECORD** with **CREATE RECORD**, filling the ID fields)
  - Explicitly load the N records when needed

Note: Starting with 4D v14 R3, you can assign values to the special "id_added_by_converter" fields that are automatically added by 4D when it converts a database containing subtables. This allows you to keep the "subtable relation" link, and add or modifiy related records, without needing to use deprecated commands such as **_o_CREATE SUBRECORD**. Once you have updated your methods, these special relations can be replaced with standard ones with no change in your code.

Non-Unicode mode

Supporting ASCII mode (synonym for “non-Unicode mode”) leads to poor performance when manipulating text because it must be converted to and from Mac-Roman every time it is used in the legacy-converted structure. We plan to remove ASCII mode in future major versions.

Note that support for ASCII mode was already removed for compiled structures running under 4D Server 64-bit for Windows.

4D developers should – for converted structures – activate the Unicode mode. The Conversion to 4D v14 PDF

document gives hints about this topic.

Note for 64-bit versions: ASCII mode is not supported in 64-bit versions of 4D and 4D Server.

Mac Resources

This is another old Mac OS technology, deprecated since Mac OS X 10.4 (Tiger, 2005). Resources are used to store structured data such as text and strings (localization), as well as icons, etc. Basically, we can say that it's not the resources that are deprecated, it's their on-disk support, known as the resource fork. The resource fork is part of the Mac OS file system, and since the beginning of Mac OS X, Apple has tried to remove this support as it is not compatible with other file systems (Unix, Windows), and is the source of a lot of problems when files are transferred via the network.

On Windows, this mechanism is emulated and Mac Resources reside in a .RSR file.

But even if there are still APIs to handle resources (and Mac OS transparently handles resources stored in a data fork), it is no longer recommended to use this old mechanism for several reasons:

- Text and strings are Mac-Roman. You can't store Unicode in resources of type TEXT or STR#
- PICT resources store PICTs: not modern, deprecated, no transparency, etc. (See the "Pictures in PICT format" topic above.)
- The count of resources and the size of the resources are limited (about 2700 resources or 16 MB)

We have removed support for commands that write/create resources.

The vast majority of 4D applications using resources are in fact using the "Strings List" resources, 'STR#'. 4D provides tools to easily move from STR# to XLIFF:

- The 4D Pop component can automatically create the XLIFF files by reading and transferring the content of the STR#.
- All the routines and expressions that reference STR# work with no change with XLIFF. For example, if the label of a button or a menu was ":15000,3" (meaning "get the third item of STR# ID 15000"), 4D will load the appropriate XLIFF (if it exists).

For other kinds of resources:

- Put resources in separate files inside the Resources folder (create sub-directories if needed):
  - Save 'TEXT' resources in XLIFF or .txt files
  - Save 'PICT' resources as separate .jpg/.png/etc. files
  - Save 'PICT' + MASK' resources as png files
  - Use (on Mac) icns instead of ICON or colored icons
  - Save any private resources as appropriate for you (typically: save as a binary file with a specific extension)
- Use the "Resources" folder to store your resources. Use **Get 4D folder(Current resources folder)** to dynamically get the parent path for your resources.

API QuickDraw for plug-ins

There are two types of plug-ins: those using the new plug-in API, and those that still use the old one (with QuickDraw).

For plug-ins using the old tool box (with QuickDraw): to maintain compatibility, the drawing/rendering is no longer done directly in a QuickDraw port, as in previous versions, but instead through a GWorld QuickDraw offscreen area dedicated to the plugin.

Consequently, you have to respect a few rules, like plugins must not modify the current port set by the container (form object).

For plug-ins using the new tool box: only this new tool box is used and not QuickDraw (see http://sources.4d.com/trac/4d_4dpluginapi/wiki/native_drawing)

4D Pack

Over the course of different versions, the most useful 4D Pack routines have been progressively integrated into 4D itself, while those that became obsolete have been removed. 4D Pack v16 now contains only a very small number of routines and will no longer evolve. Starting with 4D v16, the 4D Pack plug-in as a whole is deprecated and will no longer be provided in future versions of 4D. Refer to the table below to find out the replacement solutions available (if any) for the remaining routines.

Language: deprecated and/or removed commands

Command	Replaced with	Obsolete since	Current status
4D Environment theme:			
_o_ADD DATA SEGMENT	-	v11	Deprecated
_o_DATA SEGMENT LIST	-	v11	Deprecated
Backup theme:			
_o_INTEGRATE LOG FILE	**INTEGRATE MIRROR LOG FILE**	v16	Deprecated
Compiler theme:			
_o_ARRAY STRING	**ARRAY TEXT**	v12	Deprecated
_o_C_GRAPH	(use SVG with the **GRAPH** command)	v12	Deprecated
_o_C_INTEGER	**C_LONGINT**	v12	Deprecated
_o_C_STRING	**C_TEXT** (as soon as database is in Unicode)	v12	Deprecated
Data Entry theme:			
_o_ADD SUBRECORD	**ADD RECORD** in the n table of a N->1 relation	v12	Deprecated
_o_MODIFY SUBRECORD	**MODIFY RECORD** in the n table of a N->1 relation	v12	Deprecated
Form Events theme:			
_o_During	Replace with **Form event** and the appropriate event	v12	Deprecated
Graphs theme:			
GRAPH (using 4D Graph Area)	Use an SVG picture instead	v12	Deprecated
_o_GRAPH TABLE	Build the data in arrays and call **GRAPH** in a SVG picture	v13	Disabled since 4D v14
Hierarchical Lists theme:			
_o_REDRAW LIST	Remove in code (does nothing since v11)	v11	Deprecated
Objects (Forms) theme:			
_o_DISABLE BUTTON/**_o_ENABLE BUTTON**	**OBJECT SET ENABLED**	v12	Deprecated
Pictures theme:			
_o_PICTURE TYPE LIST	**PICTURE CODEC LIST**	v12	Deprecated
_o_QT COMPRESS PICTURE	**CONVERT PICTURE**	v12	Deprecated
_o_QT COMPRESS PICTURE FILE	**WRITE PICTURE FILE**/**PICTURE TO BLOB**	v12	Deprecated
_o_QT LOAD COMPRESS PICTURE FROM FILE	**READ PICTURE FILE**/**CONVERT PICTURE**	v12	Deprecated
_o_SAVE PICTURE TO FILE	**WRITE PICTURE FILE**	v12	Deprecated
Resources theme: all commands that			

write/create resources, i.e.:			
_o_ARRAY TO STRING LIST	-	v12	Deprecated
_o_Create resource file	-	v12	Deprecated
_o_DELETE RESOURCE	-	v12	Deprecated
_o_Get component resource ID	-	v12	Deprecated
_o_SET PICTURE RESOURCE	-	v12	Deprecated
_o_SET RESOURCE	-	v12	Deprecated
_o_SET RESOURCE NAME	-	v12	Deprecated
_o_SET RESOURCE PROPERTIES	-	v12	Deprecated
_o_SET STRING RESOURCE	-	v12	Deprecated
_o_SET TEXT RESOURCE	-	v12	Deprecated
SQL theme:			
_o_USE EXTERNAL DATABASE	**SQL LOGIN**	v12	Deprecated
_o_USE INTERNAL DATABASE	**SQL LOGOUT**	v12	Deprecated
String theme:			
_o_Convert case	**CONVERT FROM TEXT**/**Convert to text** when necessary.	v11	Deprecated
_o_ISO to Mac	Just remove the command from the method if conversion is not necessary	v11	Deprecated
_o_Mac to ISO	(which means the database runs in Unicode mode)	v11	Deprecated
_o_Mac to Win		v11	Deprecated
_o_Win to Mac		v11	Deprecated
Subrecords theme: all commands	Replace “nnn SUBRECORD” and “nnn SUBSELECTION” with an action on the N record or N-selection of the N-table in a N->1 relation	v12	Deprecated
System Documents theme:			
Document type	-	v12	Deprecated
System Environment theme:			
_o_Font name	Use font identifiers	v14	Deprecated
_o_Font number	QuickDraw is deprecated, so the **_o_Font name** and **_o_Font number** commands are deprecated. The **OBJECT SET FONT** command no longer accepts a LongInt parameter for the font: this parameter is now a String and you must specify the font name.	v14	Deprecated
User Interface theme:			
_o_Get platform interface/**_o_SET**	Can be used only for converted application; with the	v12	Deprecated

interface/_o_SET PLATFORM INTERFACE	Automatic Platform constant	v12	Deprecated
Windows theme:			
_o_Open external window	Not supported in 4D 64-bit versions	v16	Deprecated
XML theme:			
_o_XSLT APPLY TRANSFORMATION	Use PHP *libxslt* module or the **PROCESS 4D TAGS** command	v14 R4	Deprecated
_o_XSLT GET ERROR	Use PHP *libxslt* module or the **PROCESS 4D TAGS** command	v14 R4	Deprecated
_o_XSLT SET PARAMETER	Use PHP *libxslt* module or the **PROCESS 4D TAGS** command	v14 R4	Deprecated
4D Pack commands:			
_o_AP ShellExecute	**LAUNCH EXTERNAL PROCESS**	v11	Removed
_o_AP Save BMP 8 bits	Use 4D commands of the "**Pictures**" theme	v14 R5	Removed
_o_AP FCLOSE, _o_AP fopen, _o_AP FPRINT, _o_AP fread	-	v14 R5	Removed
_o_AP Get file MD5 digest	**Generate digest**	v14 R5	Removed
_o_AP BLOB to print settings	**BLOB to print settings**	v16	Deprecated
_o_AP Print settings to BLOB	**Print settings to BLOB**	v16	Deprecated
_o_AP Is picture deprecated	**GET PICTURE FORMATS**	v16	Deprecated
_o_AP NORMAL SCREEN, _o_AP FULL SCREEN	-	v16	Deprecated
_o_AP Get field infos, _o_AP Get table infos	-	v16	Deprecated
_o_AP Get tips state, _o_AP SET TIPS STATE	-	v16	Deprecated

Obsolete commands renamed and hidden

For better clarity in the 4D language, starting with 4D v15, every obsolete command has been prefixed by "_o_", if this was not already the case and are no longer available in 4D lists (code editor, type-ahead feature, etc.).
They will not be removed from existing code and will continue to work normally as long as they are supported. It is still possible (but not recommended) to add an obsolete command in a method by simply entering its name prefixed by "_o_"; it will be interpreted correctly.

_o_XSLT APPLY TRANSFORMATION

_o_XSLT APPLY TRANSFORMATION (xmlSource ; xslSheet ; result {; compileSheet})

Parameter	Type		Description
xmlSource	String, BLOB	⇒	Name or access path of XML source document, or BLOB containing the XML source
xslSheet	String, BLOB	⇒	Name or access path of document containing XSL stylesheet, or BLOB containing the XSL stylesheet
result	String, BLOB	⇒	Name or access path of the document receiving the result of the XSLT transformation, or BLOB receiving the result of the XSLT transformation
compileSheet	Boolean	⇒	True = Optimize XSLT transformation False or omitted = No optimization, remove the compiled XSL file (if any)

Compatibility note

Starting with 4D v14 R4, XSL transformation commands are obsolete. For compatibility, they are still supported in 4D but we strongly recommend that you discontinue using them. In future versions of 4D, it will no longer be possible to use XSLT technology. For more information, please refer to the **Overview of XML Utilities Commands***.*

PROCESS 4D TAGS

PROCESS 4D TAGS (inputTemplate ; outputResult {; param}{; param2 ; ... ; paramN})

Parameter	Type		Description
inputTemplate	Text, BLOB	⇒	Data containing tags to process
outputResult	Text, BLOB	⇐	Result from template execution
param	Text, Number, Date, Time, Pointer	⇒	Parameter(s) passed to template being executed

Description

The **PROCESS 4D TAGS** command causes the processing of 4D transformation tags contained in the *inputTemplate* parameter (field or variable of the BLOB or Text type) while (optionally) inserting value(s) using the *param* parameters and returns the result in *outputResult*. For a complete description of these tags, refer to the **4D Transformation Tags** section.

This command lets you execute a "template" type text containing tags and references to 4D expressions and/or variables, and to produce a result depending on the execution context and/or the values passed as parameters. For example, you can use this command to generate and save HTML pages based on **semi-dynamic pages** containing 4D transformation tags (without it being necessary for 4D's Web server to be started). You can use it to send e-mail messages in HTML format that contain processing of and/or references to data contained in the database via the 4D Internet Commands. It is possible to process any type of data based on text, such as XML, SVG or multi-style text.

Pass the data containing the tags to be processed in the *inputTemplate* parameter. This parameter can be a field or variable of the BLOB or Text type. The Text type is usually sufficient (parameters can receive up to 2 GB of text).

Compatibility note: Beginning with version 12 of 4D, when you use BLOB type parameters, the command automatically considers that the character set used for BLOBs is MacRoman. For better efficiency, it is strongly recommended to use Text type parameters for which processing is carried out in Unicode mode.

All the transformation tags of 4D are supported (*4DTEXT*, *4DHTML, 4DSCRIPT*, *4DLOOP*, *4DEVAL*, etc.).

Note: When using the *4DINCLUDE* tag outside the framework of the Web server (Web process):

- with 4D in local mode or 4D Server, the default folder is the folder containing the database structure file,
- with 4D in remote mode, the default folder is the folder containing the 4D application.

The **PROCESS 4D TAGS** command supports an indefinite number of *param* parameters that can be inserted into the executed code. As with project methods, these parameters can contain scalar values of varied types (text, date, time, longint, real, etc.). You can also use arrays, by means of array pointers. Inside the code processed by the 4D tags, these parameters can be accessed by means of standard arguments ($1, $2, etc.), just like in 4D methods 4D (see example).
A dedicated set of local variables is defined in the execution context of the **PROCESS 4D TAGS** command. These variables can be written or read during processing.

Compatibility note: In previous versions of 4D, local variables defined in the calling context could be accessed in the **PROCESS 4D TAGS** execution context in interpreted mode. Beginning with 4D v14 R4, this is not the case anymore.

After command execution, the *outputResult* parameter receives the execution result of the *inputTemplate* parameter, along with the result of the processing of any 4D tags that it contains, when applicable. If *inputTemplate* does not contain any 4D tags, the contents of *outputResult* is identical to that of *inputTemplate*.
The *outputResult* parameter may be a field or a variable, but it must be of the same type as that of the *inputTemplate* parameter.

Note: This command never calls the **On Web Authentication database method**.

Example 1

This example loads a 'template' type document, processes the tags it contains and then stores it:

```
C_BLOB($Blob_x)
C_BLOB($blob_out)
C_TEXT($inputText_t)
C_TEXT($outputText_t)

DOCUMENT TO BLOB("mytemplate.txt";$Blob_x)
$inputText_t:=BLOB to text($Blob_x;UTF8 text without length)
PROCESS 4D TAGS($inputText_t;$outputText_t)
TEXT TO BLOB($outputText_t;$blob_out;UTF8 text without length)
BLOB TO DOCUMENT($document;$blob_out)
```

Example 2

This example generates a text using data of the arrays:

```
ARRAY TEXT($array;2)
$array{1}:="hello"
$array{2}:="world"
$input:="<!--#4DEVAL $1-->"
$input:=$input+"<!--#4DLOOP $2-->"
$input:=$input+"<!--#4DEVAL $2->{$2->}--> "
$input:=$input+"<!--#4DENDLOOP-->"
PROCESS 4D TAGS($input;$output;"elements = ";->$array)
 // $output = "elements = hello world"
```

GET PICTURE FORMATS

GET PICTURE FORMATS (picture ; codecIDs)

Parameter	Type		Description
picture	Picture	⇨	Picture field or variable to analyze
codecIDs	Text array	⇦	Picture codec IDs

Description

The **GET PICTURE FORMATS** command returns an array of all the codec IDs (picture formats) contained in the *picture* passed as parameter. A 4D picture (field or variable) can contain the same picture encoded in different formats, such as PNG, BMP, GIF, etc.

In the *picture* parameter, you pass a picture field or a picture variable whose included formats you want to be returned in the *codecIDs* array.

The codec IDs returned are established by 4D in exactly the same way as for the **PICTURE CODEC LIST** command. They can be returned in the following forms:

- As extensions (for example, ".gif")
- As Mime types (for example, "image/jpeg")
- As 4-character QuickTime codes

Notes:

- The following codecs, handled internally by 4D, are always returned as extensions: JPEG, PNG, TIFF, GIF, BMP, SVG, PDF, EMF.
- 4-character QuickTime codes may be returned in databases where the QuickTime support compatibility option has been set (using the **SET DATABASE PARAMETER** command). However, QuickTime is no longer supported in 4D and we do not recommend using QuickTime codecs.

For more information about picture codec IDs, refer to the **Pictures** section.

Example

You want to know the picture formats stored in a field for the current record:

```
ARRAY TEXT($aTPictureFormats;0)
 //Get all the formats saved
GET PICTURE FORMATS([Employees]Photo;$aTPictureFormats)
```

_o_QT COMPRESS PICTURE

_o_QT COMPRESS PICTURE (picture ; method ; quality)

Parameter	Type		Description
picture	Picture	⇒	Picture to be compressed
		⇐	Compressed picture
method	String	⇒	4-character string compression method
quality	Longint	⇒	Compression quality (1..1000)

Compatibility note

This command calls for obsolete mechanisms and must be replaced by the **CONVERT PICTURE** command.

CONVERT PICTURE

CONVERT PICTURE (picture ; codec {; compression})

Parameter	Type		Description
picture	Picture	➡	Picture to be converted
		⬅	Converted picture
codec	String	➡	Picture Codec ID
compression	Real	➡	Quality of compression

Description

The **CONVERT PICTURE** command converts *picture* into a new type.

The *codec* parameter indicates the type of picture to be generated. A Codec can be an extension (for example, ".gif") or a Mime type (for example, "image/jpeg"). You can get a list of Codecs that are available using the **PICTURE CODEC LIST** command.

If the *picture* field or variable is a compound type (if, for example, it is the result of a copy-paste action), only the information corresponding to the codec type are preserved in the resulting picture.

Note: If the type of *codec* requested is the same as the original type of the *picture*, no conversion is carried out and the picture is returned "as is" (except when the *compression* parameter is used, see below).

The optional *compression* parameter, if passed, can be used to specify the compression quality to be applied to the resulting picture when a compatible Codec is used. In *compression*, pass a value between 0 and 1 to specify the quality of the compression, where 0 is the most mediocre quality (high compression) and 1 the best quality (low compression). This parameter is only taken into account when the Codec supports compression (for example JPEG or HDPhoto) and is supported by the WIC and ImageIO APIs. For more information about picture management APIs in 4D, please refer to the **Pictures** section. By default, if you omit the *compression* parameter, the best quality is applied (compression =1).

Example 1

Conversion of the vpPhoto picture to the jpeg format:

```
CONVERT PICTURE(vpPhoto;".jpg")
```

Example 2

Conversion of a picture with 60% quality:

```
CONVERT PICTURE(vPicture;".JPG";0.6)
```

Maintenance and security center

SET DATABASE PARAMETER

SET DATABASE PARAMETER ({aTable ;} selector ; value)

Parameter	Type		Description
aTable	Table	⇒	Table for which to set the parameter or, Default table if this parameter is omitted
selector	Longint	⇒	Code of the database parameter to modify
value	Real, String	⇒	Value of the parameter

Description

The **SET DATABASE PARAMETER** command allows you to modify various internal parameters of the 4D database.

The *selector* designates the database parameter to modify. 4D offers predefined constants, which are located in the "**Database Parameters**" theme. The following table lists each constant, describes its scope and indicates whether any changes made are kept between two sessions:

Constant	Type	Value	Comment
Direct2D disabled	Longint	0	See selector 69 (Direct2D Status)
Direct2D hardware	Longint	1	See selector 69 (Direct2D Status)
Direct2D software	Longint	3	See selector 69 (Direct2D Status)
Minimum Web process	Longint	6	**Scope**: 4D local, 4D Server **Kept between two sessions**: Yes **Possible values**: 0 -> 32 767 **Description**: Minimum number of Web processes to maintain in non-contextual mode with 4D in local mode and 4D Server. By default, the value is 0 (see below).
Maximum Web process	Longint	7	**Scope**: 4D local, 4D Server **Kept between two sessions**: Yes **Possible values**: 0 -> 32 767 **Description**: Maximum number of Web processes to maintain in non-contextual mode with 4D in local mode and 4D Server. By default, the value is 10. In non-contextual mode, for the Web server to be reactive, 4D delays the Web processes for 5 seconds and reuses them to execute any possible future HTTP queries. In terms of performance, this is actually more advantageous than creating a new process for each query. Once a Web process is reused, it is delayed again for 5 seconds. When the maximum number of Web processes has been reached, the web process is then aborted. If no query has been attributed to a Web process within the 5 second delay, the process is aborted, except if the minimum number of Web processes has been reached (in which case the process is delayed again). These parameters allow you to adjust how your Web server operates in relation to the number of requests and the memory available as well as other parameters.
_o_Web conversion mode	Longint	8	**** *Selector disabled* ****
_o_Database cache size	Longint	9	**Scope**: 4D application **Kept between two sessions**: - **Description**: *Constant obsolete (kept for compatibility reasons only)*. We now recommend using the **Get cache size** command.
4D Local mode scheduler	Longint	10	**Scope**: 4D application **Kept between two sessions**: Yes **Description**: see selector 12
4D Server scheduler	Longint	11	**Scope**: 4D application **Kept between two sessions**: Yes **Description**: see selector 12
			Scope: 4D application **Kept between two sessions**: Yes **Possible values**: for selectors 10, 11 and 12, the *value* parameter is expressed in hexadecimal *0x00aabbcc* as follows: *aa* = minimum number of ticks per call to the system (0 to 100 included). *bb* = maximum number of ticks per call to the system (0 to 100 included). *cc* = number of ticks between calls to the system (0 to 20 included). If one of the values is out of range, 4D sets it to its maximum. You can pass one of the following preset standard values in the *value* parameter: • *value* = -1: maximum priority allocated to 4D,

4D Remote mode scheduler	Longint	12	• *value* = -2: average priority allocated to 4D, • *value* = -3: minimum priority allocated to 4D. **Description**: This parameter allows you to dynamically set the 4D system internal calls. Depending on the Selector, the scheduler value will be set for: • 4D local mode when the command is called from a 4D single-user application (*selector=10*). • 4D Server when the command is called from 4D Server (*selector=11*). • 4D remote mode when the command is called from a 4D connected to 4D Server (*selector=12*). **Note:** The operation of selector 12 (4D Remote Mode Scheduler) differs according to whether the **SET DATABASE PARAMETER** command is executed on the server machine or on the client machine: - If the command is executed on the server machine, the new value will be applied to all the client machines that connect to it subsequently. - If the command is executed on the client machine, the new value is applied to the client machine immediately as well as to all the client machines that connect to the server subsequently. You can use this operation to implement a dynamic and individualized management of priority for each client machine. This consists in executing the command initially on the client machine to be configured, then a second time on the server machine using the default value, which will then be used for the client machines that connect to it subsequently. This operation is in effect in 4D starting with versions 6.8.6, 2003.3 and 2004. **Warning**: Configuring these selectors inappropriately can cause serious degradation of application performance. It is recommended to only modify the default values with full knowledge of the facts.
4D Server timeout	Longint	13	**Scope**: 4D application if *value* positive **Kept between two sessions**: Yes if *value* positive **Possible values**: 0 -> 32 767 **Description**: Value of the 4D Server timeout. The default 4D Server timeout value is defined on the "Client-Server/Network options" page of the Database settings dialog box on the server side. The server timeout sets the maximum period "authorized" to wait for a client response, for example when it is executing a blocking operation. After this period, 4D Server disconnects the client. The 4D Server Timeout selector allows you to set, in the corresponding *value* parameter, a new timeout expressed in minutes. This feature is particularly useful to increase the timeout before executing a blocking and time-consuming operation on the client, such as printing a large number of pages, which can cause an unexpected timeout. You also have two options: • If you pass a **positive** value in the *value* parameter, you set a global and permanent timeout: the new value is applied to all process and is stored in the preferences of the 4D application (equivalent to change in the Preferences dialog box). • If you pass a **negative** value in the *value* parameter, you set a local and temporary timeout: The new value is applied to the calling process only (the other processes keep the default values) and is reset to default as soon as the server receives any signal of activity from the client — for example, when the operation is finished. This option is useful for managing long operations initiated by 4D plug-ins. To set the "No timeout" option, pass 0 in *value*. See example 1. **Scope** (legacy network layer only): 4D application if *value* positive

4D Remote mode timeout	Longint	14	**Kept between two sessions**: Yes if *value* positive **Description**: To be used in very specific cases. Value of the timeout granted by the remote 4D machine to the 4D Server machine. The default timeout value used by 4D in remote mode is set on the "Client-Server/Network options" page of the Database settings dialog box on the remote machine. The 4D Remote mode timeout selector is only taken into account if you are using the legacy network. It is ignored when the *ServerNet* layer is activated: this setting is entirely managed by the 4D Server timeout (13) selector.
Port ID	Longint	15	**Scope**: 4D local, 4D Server **Kept between two sessions**: No **Description**: TCP port ID used by the 4D Web server with 4D in local mode and 4D Server. The default value, which can be set on the "Web/Configuration" page of the Preferences dialog box, is 80. You can use the constants of the **TCP Port Numbers** theme for the *value* parameter. The Port ID selector is useful for 4D Web Servers compiled and merged with 4D Desktop (in which there is no access to the Design mode). For more information about the TCP port ID, refer to the **Web Server Settings** section.
IP Address to listen	Longint	16	**Scope**: 4D local, 4D Server **Kept between two sessions**: Yes **Description**: *Constant obsolete (kept for compatibility reasons only).* We now recommend using the **WEB SET OPTION** and **WEB GET OPTION** commands for configuring the HTTP server.
Character set	Longint	17	**Scope**: 4D local, 4D Server **Kept between two sessions**: Yes **Description**: *Constant obsolete (kept for compatibility reasons only).* We now recommend using the **WEB SET OPTION** and **WEB GET OPTION** commands for configuring the HTTP server.
Max concurrent Web processes	Longint	18	**Scope**: 4D local, 4D Server **Kept between two sessions**: Yes **Description**: *Constant obsolete (kept for compatibility reasons only).* We now recommend using the **WEB SET OPTION** and **WEB GET OPTION** commands for configuring the HTTP server.
Client minimum Web process	Longint	19	**Scope**: All 4D remote machines **Kept between two sessions**: Yes **Possible values**: See selector 6 **Description**: Used to specify this parameter for all the remote 4D machines used as Web servers. The values defined using these selectors are applied to all the remote machines used as Web servers. If you want to define values only for certain remote machines, use the Preferences dialog box of 4D in remote mode.
Client maximum Web process	Longint	20	**Scope**: All 4D remote machines **Kept between two sessions**: Yes **Possible values**: See selector 7 **Description**: Used to specify this parameter for all the remote 4D machines used as Web servers. The values defined using these selectors are applied to all the remote machines used as Web servers. If you want to define values only for certain remote machines, use the Preferences dialog box of 4D in remote mode.
Client Max Web requests size	Longint	21	**Scope**: All 4D remote machines **Kept between two sessions**: Yes **Possible values**: See selector 27 **Description**: Used to specify this parameter for all the remote 4D machines used as Web servers. The values defined using these selectors are applied to all the remote machines used as Web servers. If you want to define values only for certain remote machines, use the Preferences dialog box of 4D in

			remote mode.
Client port ID	Longint	22	**Scope**: All 4D remote machines **Kept between two sessions**: Yes **Possible values**: See selector 15 **Description**: Used to specify this parameter for all the remote 4D machines used as Web servers. The values defined using these selectors are applied to all the remote machines used as Web servers. If you want to define values only for certain remote machines, use the Preferences dialog box of 4D in remote mode.
Client IP address to listen	Longint	23	**Scope**: All 4D remote machines **Kept between two sessions**: Yes **Possible values**: See selector 16 **Description**: Used to specify this parameter for all the remote 4D machines used as Web servers. The values defined using these selectors are applied to all the remote machines used as Web servers. If you want to define values only for certain remote machines, use the Preferences dialog box of 4D in remote mode.
Client character set	Longint	24	**Scope**: All 4D remote machines **Kept between two sessions**: Yes **Possible values**: See selector 17 **Description**: Used to specify this parameter for all the remote 4D machines used as Web servers. The values defined using these selectors are applied to all the remote machines used as Web servers. If you want to define values only for certain remote machines, use the Preferences dialog box of 4D in remote mode.
Client max concurrent Web proc	Longint	25	**Scope**: All 4D remote machines **Kept between two sessions**: Yes **Possible values**: See selector 18 **Description**: Used to specify this parameter for all the remote 4D machines used as Web servers. The values defined using these selectors are applied to all the remote machines used as Web servers. If you want to define values only for certain remote machines, use the Preferences dialog box of 4D in remote mode.
Maximum Web requests size	Longint	27	**Scope**: 4D local, 4D Server **Kept between two sessions**: Yes **Description**: *Constant obsolete (kept for compatibility reasons only).* We now recommend using the **WEB SET OPTION** and **WEB GET OPTION** commands for configuring the HTTP server.
4D Server log recording	Longint	28	**Scope**: 4D Server, 4D remote **Kept between two sessions**: No **Possible values**: 0 or from 1 to X (0 = do not record, 1 to X = sequential number, added to the file name). **Description**: Starts or stops the recording of standard requests received by 4D Server (excluding Web requests). By default, the value is 0 (requests not recorded). 4D Server lets you record each request received by the server machine in a log file. When this mechanism is enabled, two files are created in the Logs folder of the database, next to the database structure file.They are named 4DRequestsLog_X.txt and 4DRequestsLog_ProcessInfo_X.txt, where X is the sequential number of the log. Once the file 4DRequestsLog has reached a size of 10 MB, it is closed and a new one is generated, with an incremented sequential number. If a file of the same name already exists, it is replaced directly. You can set the starting number of the sequence using the *value* parameter. These text files store various information concerning each request in a simple

			tabbed format: time, process number, size of request, processing duration, etc. This information is particularly useful during the development phase of the application or for statistical purposes. It can be imported, for example, into a spreadsheet software in order to be processed.
_o_Web Log recording	Longint	29	**Scope**: 4D local, 4D Server **Kept between two sessions**: Yes **Description**: *Constant obsolete (kept for compatibility reasons only).* We now recommend using the **WEB SET OPTION** and **WEB GET OPTION** commands for configuring the HTTP server.
Client Web log recording	Longint	30	**Scope**: All 4D remote machines **Kept between two sessions**: Yes **Possible values**: 0 = Do not record (default), 1 = Record in CLF format, 2 = Record in DLF format, 3 = Record in ELF format, 4 = Record in WLF format. **Description**: Starts or stops the recording of Web requests received by the Web servers of all the client machines. By default, the value is 0 (requests not recorded). The operation of this selector is identical to that of selector 29; however, it applies to all the 4D remote machines used as Web servers. The "logweb.txt" file is, in this case, automatically placed in the Logs subfolder of the remote 4D database folder (cache folder). If you only want to set values for certain client machines, use the Preferences dialog box of 4D in remote mode.
Table sequence number	Longint	31	**Scope**: 4D application **Kept between two sessions**: Yes **Possible values**: Any longint value. **Description**: This selector is used to modify or get the current unique number for records of the table passed as parameter. "Current number" means "last number used": if you modify this value using **SET DATABASE PARAMETER**, the next record will be created with a number that consists of the value passed + 1. This new number is the one returned by the **Sequence number** command as well in any field of the table to which the "Autoincrement" property has been assigned in the Structure editor or via SQL. By default, this unique number is set by 4D and corresponds to the order of record creation. For additional information, refer to the documentation of the **Sequence number** command.
_o_Real display precision	Longint	32	**** *Selector disabled* ****
			Scope: 4D application **Kept between two sessions**: No **Description**: Starts or stops the sequential recording of events occurring at the 4D programming level in the *4DDebugLog* file, which is automatically placed in the Logs subfolder of the database, next to the structure file. A new, more compact, tabbed text format is used in the event log file "4DDebugLog[_n].txt" starting with 4D v14 (where _n is the segment number of the file). **Possible values**: Longint containing a bit field: value = bit1(1)+bit2(2)+bit3(4)+bit4(8)+⋯). - Bit 1 (value 1) requests to enable the file (note that any other non-null value also enables it as well) - Bit 2 (value 2) requests call parameters to methods and commands. - Bit 3 (value 4) enables new tabbed format. - Bit 4 (value 8) disables immediate writing of each operation on disk (enabled by default). Immediate writing is slower but more effective, for example for investigating causes of a crash. If you disable this mode, the file contents are more compact and are generated more quickly. - Bit 5 (value 16) disables recording of plug-in calls (enabled by default).

Debug log recording	Longint	34	In the (former) non-tabbed format, execution times are expressed in milliseconds and the "< ms" value is displayed when an operation lasts less than one millisecond.
			In the new tabbed format, execution times are expressed in microseconds.
			Examples:
			SET DATABASE PARAMETER (34;1) // enables mode v13 file without parameters, with runtimes
			SET DATABASE PARAMETER (34;2) // enables mode v13 file with parameters and runtimes
			SET DATABASE PARAMETER (34;2+4) // enables file with v14 format, with parameters and runtimes
			SET DATABASE PARAMETER (34;0) // disables file
			To avoid having a file record too much information, you can restrict the 4D commands that are examined by using selector 80, Log Command list.
			This option can be enabled for any type of 4D application (4D all modes, 4D Server, 4D Volume Desktop), in interpreted or compiled mode.
			Note: This option is provided solely for the purpose of debugging and must not be put into production since it may lead to deterioration of the application performance and saturation of the hard disk. For more information about this format and on the use of the 4DDebugLog[_n].txt file, please contact the Technical Support of 4D Inc.

Client Server port ID	Longint	35	**Scope**: Database **Kept between two sessions**: Yes **Possible values**: 0 to 65535 **Description**: TCP port number where the 4D Server publishes the database (bound for 4D remote machines). By default, the value is 19813. Customizing this value means that several 4D client-server applications can be used on the same machine with the TCP protocol; in this case, you must indicate a different port number for each application. The value is stored in the database structure file. It can be set with 4D in local mode but is only taken into account in client-server configuration. When you modify this value, it is necessary to restart the server machine in order for the new value to be taken into account.
Invert objects	Longint	37	**Scope**: Database **Kept between two sessions**: Yes **Possible values**: 0, 1 or 2 (0 = mode disabled, 1 = automatic mode, 2 = mode enabled). **Description**: Configuration of the "object inversion" mode which is used to invert forms, objects, menu bars, and so on, in Application mode when the database is displayed under Windows in a right-to-left language. This mode can also be configured on the Interface/Right-to-left languages page of the Database Settings. • Value 0 indicates that the mode is never enabled, regardless of the system configuration (corresponds to the Never value in the Database Settings). • Value 1 indicates that the mode is enabled or disabled depending on the system configuration (corresponds to the Automatic value in the Database Settings). • Value 2 indicates that the mode is enabled, regardless of the system configuration (corresponds to the Always value in the Database Settings). For more information, refer to the *Design Reference* manual of 4D.
HTTPS Port ID	Longint	39	**Scope**: 4D local, 4D Server **Kept between two sessions**: Yes **Description**: *Constant obsolete (kept for compatibility reasons only).* We now

ID			recommend using the **WEB SET OPTION** and **WEB GET OPTION** commands for configuring the HTTP server.
Client HTTPS port ID	Longint	40	**Scope**: All 4D remote machines **Kept between two sessions**: Yes **Possible values**: 0 to 65535 **Description**: TCP port number used by the Web servers of the client machines for secure connections via SSL (HTTPS protocol). By default, the value is 443 (standard value). This selector can be used to modify by programming the TCP port used by the Web servers of the client machines for secure connections via SSL (HTTPS protocol). By default, the value is 443 (standard value). This selector operates exactly the same way as selector 39; however, it applies to all the 4D remote machines used as Web servers. If you only want to modify the value of certain specific client machines, use the Preferences dialog box of the remote 4D.
Unicode mode	Longint	41	**Scope**: Database **Kept between two sessions**: Yes **Possible values**: 0 (compatibility mode) or 1 (Unicode mode) **Description**: Current database operating mode, with regards to the character set. 4D supports the Unicode character set but can function in “compatibility” mode (based on the Mac ASCII character set). By default, converted databases are executed in compatibility mode (0) and databases created with version 11 or higher are executed in Unicode mode. The execution mode can be controlled via an option in the Preferences and can also be read or (for testing purposes) modified via this selector. Modifying this option requires the database to be restarted in order for it to be taken into account. Note that within a component it is not possible to modify this value, but only to read it.
SQL Autocommit	Longint	43	**Scope**: Database **Kept between two sessions**: Yes **Possible values**: 0 (deactivation) or 1 (activation) **Description**: Activation or deactivation of the SQL auto-commit mode. By default, the value is 0 (deactivated mode) The auto-commit mode is used to strengthen the referential integrity of the database. When this mode is active, all **SELECT**, **INSERT**, **UPDATE** and **DELETE** (SIUD) queries are automatically included in ad hoc transactions when they are not already executed within a transaction. This mode can also be set in the Preferences of the database.
SQL Engine case sensitivity	Longint	44	**Scope**: Database **Kept between two sessions**: Yes **Possible values**: 0 (case not taken into account) or 1 (case-sensitive) **Description**: Activation or deactivation of case-sensitivity for string comparisons carried out by the SQL engine. By default, the value is 1 (case-sensitive): the SQL engine differentiates between upper and lower case and between accented characters when comparing strings (sorts and queries). For example “ABC”= “ABC” but “ABC” # “Abc” and "abc" # "âbc." In certain cases, for example so as to align the functioning of the SQL engine with that of the 4D engine, you may wish for string comparisons to not be case-sensitive (“ABC”=“Abc"="âbc"). This option can also be set on the **SQL page** of the Database settings.
			Scope: Remote 4D machine **Kept between two sessions**: No **Possible values**: 0 or from 1 to X (0 = do not record, 1 to X = sequential number, attached to file name). **Description**: Starts or stops recording of standard requests carried out by the 4D client machine that executed the command (excluding Web requests). By default, the value is 0 (no recording of requests).

Client log recording	Longint	45	4D lets you record the log of requests carried out by the client machine. When this mechanism is activated, two files are created on the client machine, in the Logs subfolder of the local folder of the database. They are named 4DRequestsLog_X.txt and 4DRequestsLog_ProcessInfo_X.txt, where X is the sequential number of the log. Once the file 4DRequestsLog has reached a size of 10 MB, it is closed and a new one is generated, with an incremented sequential number. If a file with the same name already exists, it is directly replaced. You can set the starting number for the sequence using the value parameter. These text files store various information concerning each request in a simple tabbed format: time, process number, size of request, processing duration, etc. This information is particularly useful during the development phase of the application or for statistical purposes
Query by formula on server	Longint	46	**Scope**: Current table and process **Kept between two sessions**: No **Possible values**: 0 (use database configuration), 1 (execute on client) or 2 (execute on server) **Description**: Execution location of **QUERY BY FORMULA** and **QUERY SELECTION BY FORMULA** commands for the *table* passed in the parameter. When using a database in client-server mode, the query "by formula" commands can be executed either on the server or on the client machine: • In databases created with 4D v11 SQL, these commands are executed on the server. • In converted databases, these commands are executed on the client machine, as in previous versions of 4D. • In converted databases, a specific preference (Application/Compatibility page) can be used to globally modify the execution location of these commands. This difference in execution location influences not only application performance (execution on the server is usually faster) but also programming. In fact, the value of the components of the formula (in particular variables called via a method) differ according to the execution context. You can use this selector to punctually adapt the operation of your application. If you pass 0 in the *value* l'parameter, the execution location of query "by formula" commands will depend on the database configuration: in databases created with 4D v11 SQL, these commands will be executed on the server. In converted databases, they will be executed on the client machine or the server according to the database preferences. Pass 1 or 2 in *value* to "force" the execution of these commands, respectively, on the client or on the server machine. Refer to example 4. **Note:** If you want to be able to enable "SQL type" joins (see the QUERY BY FORMULA Joins selector), you must always execute formulas on the server so that they have access to the records. Be careful, in this context, the formula must not contain any calls to a method, otherwise it will automatically be switched to the remote machine.
Order by formula on server	Longint	47	**Scope**: Current table and process **Kept between two sessions**: No **Possible values**: 0 (use database configuration), 1 (execute on client) or 2 (execute on server) **Description** : Execution location of **ORDER BY FORMULA** command for the table passed in the parameter. When using a database in client-server mode, this command can be executed either on the server or on the client machine. This selector can be used to specify the execution location of this command (server or client). This mode

server			can also be set in the database preferences. For more information, please refer to the description of selector 46, Query By Formula On Server. **Note:** If you want to be able to enable "SQL type" joins (see the QUERY BY FORMULA Joins selector), you must always execute formulas on the server so that they have access to the records. Be careful, in this context, the formula must not contain any calls to a method, otherwise it will automatically be switched to the remote machine.
Auto synchro resources folder	Longint	48	**Scope**: 4D remote machine **Kept between two sessions**: No **Possible values**: 0 (no synchronization), 1 (auto synchronization) or 2 (ask). **Description**: Dynamic synchronization mode for *Resources* folder of 4D client machine that executed the command with that of the server. When the contents of the *Resources* folder on the server has been modified or a user has requested synchronization (for example via the resources explorer or following the execution of the **SET DATABASE LOCALIZATION** command), the server notifies the connected client machines. Three synchronization modes are then possible on the client side. The Auto Synchro Resources Folder selector is used to specify the mode to be used by the client machine for the current session: • 0 (default value): no dynamic synchronization (synchronization request is ignored) • 1: automatic dynamic synchronization • 2: display of a dialog box on the client machines, with the possibility of allowing or refusing synchronization. The synchronization mode can also be set globally in the application Preferences.
Query by formula joins	Longint	49	**Scope**: Current process **Kept between two sessions**: No **Possible values**: 0 (use database configuration), 1 (always use automatic relations) or 2 (use SQL joins if possible). **Description**: Operating mode of the **QUERY BY FORMULA** and **QUERY SELECTION BY FORMULA** commands relating to the use of "SQL joins." In databases created starting with version 11.2 of 4D v11 SQL, these commands carry out joins based on the SQL joins model. This mechanism can be used to modify the selection of a table according to a query carried out on another table without these tables being connected by an automatic relation (necessary condition in previous versions of 4D). The QUERY BY FORMULA Joins selector lets you specify the operating mode of the query by formula commands for the current process: • 0: Uses the current settings of the database (default value). In databases created starting with version 11.2 of 4D v11 SQL, "SQL joins" are always activated for queries by formula. In converted databases, this mechanism is not activated by default for compatibility reasons but can be implemented via a preference. • 1: Always use automatic relations (= functioning of previous versions of 4D). In this mode, a relation is necessary in order to set the selection of a table according to queries carried out on another table. 4D does not do "SQL joins." • 2: Use SQL joins if possible (= default operation of databases created in version 11.2 and higher of 4D v11 SQL). In this mode, 4D establishes "SQL joins" for queries by formula when the formula is suited for it (with two notable exceptions, see the description of the **QUERY BY FORMULA** or **QUERY SELECTION BY FORMULA** command).

			Note: With 4D in remote mode, "SQL joins" can only be used if the formulas are executed on the server (they must have access to the records). To configure where formulas are to be executed, please refer to selectors 46 and 47.
HTTP compression level	Longint	50	**Scope**: 4D application **Kept between two sessions**: No **Description**: *Constant obsolete (kept for compatibility reasons only).* We now recommend using the **WEB SET OPTION** and **WEB GET OPTION** commands for configuring the HTTP server.
HTTP compression threshold	Longint	51	**Scope**:4D application **Kept between two sessions**: No **Description**: *Constant obsolete (kept for compatibility reasons only).* We now recommend using the **WEB SET OPTION** and **WEB GET OPTION** commands for configuring the HTTP server.
Server base process stack size	Longint	53	**Scope**: 4D Server **Kept between two sessions**: No **Possible values**: Positive longint. **Description**: Size of the stack allocated to each preemptive system process on the server, expressed in bytes. The default size is determined by the system. Preemptive system processes (processes of the 4D client base process type) are loaded to control the main 4D client processes. The size allocated by default to the stack of each preemptive process allows a good ease of execution but may prove to be consequential when very large numbers of processes (several hundred) are created. For optimization purposes, this size can be reduced considerably if the operations carried out by the database allow for it (for example if the database does not carry out sorts of large quantities of records). Values of 512 or even 256 KB are possible. Be careful, under-sizing the stack is critical and can be harmful to the operation of 4D Server. Setting this parameter should be done with caution and must take the database conditions of use into account (number of records, type of operations, etc.). In order to be taken into account, this parameter must be executed on the server machine (for example in the **On Server Startup Database Method**).
Idle connections timeout	Longint	54	**Scope**:4D application unless value is negative **Kept between two sessions**: No **Possible values**: Whole value expressing a duration in seconds. The value can be positive (new connections) or negative (existing connections). By default, the value is 20. **Description**: Maximum period of inactivity (timeout) for connections to both the 4D database engine and the SQL engine, as well as, in *ServerNet* mode (new network layer), to the 4D application server. When an idle connection reaches this limit, it is automatically put on standby, which freezes the client/server session and closes the network socket. In the server administration window, the state of the user process is indicated as "Postponed". This functioning is completely transparent for the user: as soon as there is new activity on the connection which is on standby, the socket is automatically reopened and the client/server session is restored. On the one hand, this setting lets you save resources on the server: connections on standby close the socket and free up a process on the server. On the other hand, it lets you avoid losing connections due to the closing of idle sockets by the firewall. For this, the timeout value for idle connections must be lower than that of the firewall in this case. If you pass a positive value in *value*, it applies to all new connections in all the processes. If you pass a negative value, it applies to connections that are open in the current process. If you pass 0, idle connections are not subjected to a timeout.

			This parameter can be set on both the server and client side. If you pass two different durations, the shorter one is taken into account. Usually, you do not need to change this value.
PHP interpreter IP address	Longint	55	**Scope**: 4D application **Kept between two sessions**: No **Values**: Formatted string of the type "nnn.nnn.nnn.nnn" (for example "127.0.0.1"). **Description**: IP address used locally by 4D to communicate with the PHP interpreter via FastCGI. By default, the value is "127.0.0.1". This address must correspond to the machine where 4D is located. This parameter can also be set globally for all the machines via the Database Settings. For more information about the PHP interpreter, please refer to the *Design Reference* manual.
PHP interpreter port	Longint	56	**Scope**: 4D application **Kept between two sessions**: No **Values**: Positive long integer type value. By default, the value is 8002. **Description**: Number of the TCP port used by the PHP interpreter of 4D. This parameter can also be modified globally for all the machines via the Database Settings. For more information about the PHP interpreter, please refer to the *Design Reference* manual.
PHP number of children	Longint	57	**Scope**: 4D application **Kept between two sessions**: No **Values**: Positive long integer type value. By default, the value is 5. **Description**: Number of child processes to be created and maintained locally by the PHP interpreter of 4D. For optimization reasons, the PHP interpreter creates and uses a set (pool) of system processes called "child processes" for processing script execution requests. You can vary the number of child processes according to the needs of your application. This parameter can also be modified globally for all the machines via the Database Settings. For more information about the PHP interpreter, please refer to the *Design Reference* manual. **Note:** Under Mac OS, all the child processes share the same port. Under Windows, each child process uses a specific port number. The first number is the one set for the PHP interpreter; the other child processes increment the first one. For example, if the default port is 8002 and you launch 5 child processes, they will use ports 8002 to 8006.
PHP max requests	Longint	58	**Scope**: 4D application **Kept between two sessions**: No **Values**: Positive long integer type value. By default, the value is 500. **Description**: Maximum number of requests accepted by the PHP interpreter. When this maximum number is reached, the interpreter returns errors of the "server busy" type. For security or performance reasons, you can modify this value. This parameter can also be modified globally for all the machines via the Database Settings. For more information about this parameter, please refer to the FastCGI-PHP documentation. **Note:** On the 4D side, these parameters are applied dynamically; it is not necessary to exit 4D in order for them to be taken into account. On the other hand, if the PHP interpreter is already launched, it will be necessary to exit and relaunch it in order for these modifications to be taken into account.
PHP use			**Scope**: 4D application **Kept between two sessions**: No **Values** : 0 = use internal interpreter, 1 = use external interpreter **Description**: Value indicating whether PHP requests in 4D are sent to the internal interpreter provided by 4D or to an external interpreter. By default the value is 0 (use of interpreter provided by 4D). If you want to use your own PHP interpreter, for example in order to use additional modules or a specific

external interpreter	Longint	60	PHP interpreter, for example in order to use additional modules or a specific configuration, pass 1 in *value*. In this case, 4D does not launch its internal interpreter in the case of PHP requests. The custom PHP interpreter must have been compiled in FastCGI and be located on the same machine as the 4D engine. Note that in this case, you must manage the interpreter entirely; it will not be started nor stopped by 4D. This parameter can also be modified globally for all the machines via the Database Settings.
Maximum temporary memory size	Longint	61	**Scope**: 4D application **Kept between two sessions**: No **Possible values**: Positive longint. **Description**: Maximum size of temporary memory that 4D can allocate to each process, expressed in MB. By default, the value is 0 (no maximum size). 4D uses a special temporary memory dedicated to indexing and sorting operations. This memory is intended to preserve the "standard" cache memory during massive operations. It is activated only when needed. By default, the size of the temporary memory is limited only by the resources available (according to the system memory configuration). This mechanism is suitable for most applications. However, in certain specific contexts, more particularly when a client-server application simultaneously carries out a large number of sequential sorts, the size of the temporary memory can increase critically, to the point where it can render the system unstable. In this context, setting a maximum size for the temporary memory allows you to preserve proper functioning of the application. In return, the running speed might be affected: when the maximum size is reached for a process, 4D uses disk files which may slow down processing. For specific needs such as those described above, a maximum size of around 50 MB is generally a good compromise. However, the ideal value will need to be determined according to the specificities of the application and will generally be the result of real-time volumetric testing.
SSL cipher list	String	64	**Scope**: 4D application **Kept between two sessions**: No **Possible values**: Sequence of strings separated by colons (for example "RC4-MD5:RC4-64-MD5:....") **Description:** Cipher list used by 4D for the secure protocol. This list modifies the priority of ciphering algorithms implemented by 4D. For example, you can pass the following string in the *value* parameter: "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH". For a complete description of the syntax for the ciphers list, refer to the ciphers page of the OpenSSL site. This setting applies to the entire 4D application (it concerns the HTTP server, SQL server, client/server connections, as well as the HTTP client and all the 4D commands that make use of the secure protocol) but it is temporary (it is not maintained between sessions). When the cipher list has been modified, you will need to restart the server concerned in order for the new settings to be taken into account. To reset the cipher list to its default value (stored permanently in the SLI file), call the **SET DATABASE PARAMETER** command and pass an empty string ("") in the *value* parameter. **Note:** With the **Get database parameter** command, the cipher list is returned in the optional *stringValue* parameter and the return parameter is always 0.
			Scope: 4D application **Kept between two sessions**: No **Possible values**: Positive longint > 1. **Description**: Minimum size of memory to release from the database cache

Cache unload minimum size	Longint	66	when the engine needs to make space in order to allocate an object to it (value in bytes). The purpose of this selector is to reduce the number of times that data is released from the cache in order to obtain better performance. You can vary this setting according to the size of the cache and that of the blocks of data being handled in your database. By default, if this selector is not used, 4D unloads at least 10% of the cache when space is needed.
Direct2D status	Longint	69	**Scope**: 4D application **Kept between two sessions**: No **Description**: Activation mode to implement Direct2D under Windows. **Possible values**: One of the following constants (mode 3 by default): Direct2D Disabled (0): Direct2D mode is not enabled and the database functions in the previous mode (GDI/GDIPlus). Direct2D Hardware (1): Use Direct2D as graphics hardware context for entire 4D application. If this context is not available, use Direct2D graphics software context (except under Vista, in which case GDI/GDIPlus mode is used for better performance). Direct2D Software (3) (Default mode): Beginning with Windows 7, use Direct2D graphics software context for entire 4D application. Under Vista, GDI/GDIPlus mode is used for better performance. **Compatibility note:** Starting with 4D v14, hybrid modes are disabled and redirected to available modes (the former mode 2 is equivalent to 1; former modes 4 and 5 are equivalent to mode 3).

Direct2D get active status	Longint	74	**Note**: You can only use this selector with the **Get database parameter** command and its value cannot be set.
			Description: Returns active implementation of Direct2D under Windows.
			Possible values: 0, 1, 2, 3, 4 or 5 (see values of selector 69). The value returned depends on the availability of Direct2D, the hardware and the quality of Direct2D support by the operating system.
			For example, if you execute:
			`SET DATABASE PARAMETER(Direct2D status;Direct2D Hardware)`
			`$mode:=Get database parameter(Direct2D get active status)`
			
			- On Windows 7 and higher, *$mode* is set to 1 when the system detects hardware compatible with Direct2D; otherwise, *$mode* is set to 3 (software context).
			- On Windows Vista, *$mode* is set to 1 when the system detects hardware compatible with Direct2D; otherwise, *$mode* is set to 0 (disabling of Direct2D).
			- On Windows XP, *$mode* is always set to 0 (not compatible with Direct2D).

Diagnostic log recording	Longint	79	**Scope**: 4D application **Kept between two sessions**: No **Possible values**: 0 or 1 (0 = do not record, 1 = record) **Description**: Starts or stops recording of the 4D diagnostic file. By default, the value is 0 (do not record). 4D can continuously record a set of events related to the internal application operation into a diagnostic file. Information contained in this file is intended for the development of 4D applications and can be analyzed with the help of the 4D tech support. When you pass 1 in this selector, a diagnostic file, named *DatabaseName_X*.txt, is automatically created (or opened) in the database **Logs** folder. Once this file reaches a size of 10 MB, it is closed and a new file named *DatabaseName_X*.txt is generated, with an incremented sequence number X. Note that you can include custom information in this file using the **LOG EVENT**

Log command list	String	80	command. **Scope**: 4D application **Kept between two sessions**: No **Possible values**: String containing a list of 4D command numbers to record (separated by semi-colons) or "all" to record all the commands or "" (empty string) to record none of them. **Description**: List of 4D commands to record in the debugging file (see selector 34, Debug Log Recording). By default, all 4D commands are recorded. This selector restricts the quantity of information saved in the debugging file by limiting the 4D commands whose execution you want to record. For example, you can write: `SET DATABASE PARAMETER(Log command list;"277;341") //Record only the QUERY and QUERY SELECTION commands`
Spellchecker	Longint	81	**Scope**: 4D application **Kept between two sessions**: No **Possible values**: 0 (default) = native OS X spellchecker (Hunspell disabled), 1 = Hunspell spellcheck enabled. **Description**: Enables the Hunspell spellcheck under OS X. By default, the native spellchecker is enabled on this platform. You may prefer to use the Hunspell spellcheck, for example, in order to unify the interface for your cross-platform applications (under Windows, only the Hunspell spellcheck is available). For more information, refer to **Support of Hunspell dictionaries**.
QuickTime support	Longint	82	**Scope**: 4D application **Kept between two sessions**: Yes **Possible values**: 0 (default) = QuickTime disabled, 1 = QuickTime enabled. **Description**: In 4D starting with v14, by default QuickTime codecs are no longer supported. For compatibility, you can use this selector to re-enable them in your database. Modification of this option requires that the database be restarted. Nevertheless, you should note that in future versions of 4D, QuickTime support is permanently removed.
JSON use local time	Longint	85	**Scope**: Current process **Kept between two sessions**: No **Possible values**: 0 = ignore local time zone, 1 (default) = take time zone into account. **Description**: By default, 4D dates converted to JSON format take the local time zone into account. For example, converting the date !23/08/2013! gives you "2013-08-22T22:00:00Z" in JSON format when the operation is performed in France during Daylight Savings Time (GMT+2). This principle conforms to the standard operation of JavaScript. This can be a source of errors when you want to send JSON date values to someone in a different time zone. This is the case, for example, when you export a table using **Selection to JSON** in France that is meant to be reimported in the US using **JSON TO SELECTION**. By default, since dates are re-interpreted in each time zone, the values stored in the database will be different. In this case, you can modify the conversion mode for dates so that they do not take the time zone into account by passing 0 in this selector. Converting the date !23/08/2013! will then give you "2013-08-23T00:00:00Z" in all cases.
			Scope: 4D in local mode, 4D Server **Kept between two sessions**: Yes **Description:** Sets or gets the current status of the legacy network layer for client/server connections. The legacy network layer is obsolete beginning with 4D v14 R5 and should be replaced progressively in your applications with the

Use legacy network layer	Longint	87	*ServerNet* network layer. *ServerNet* will be required in upcoming 4D releases in order to benefit from future network evolutions. For compatibility reasons, the legacy network layer is still supported to allow a smooth transition for existing applications; (it is used by default in applications converted from a release prior to v14 R5). Pass 1 in this parameter to use the legacy network layer (and disable *ServerNet*) for your client/server connections, and pass 0 to disable the legacy network (and use the *ServerNet*). This property can also be set by means of the "Use legacy network layer" option found on the **Compatibility page** of the Database Settings (see **Network and Client-Server options**). In this section, you will also find a discussion about migration strategy. We recommend that you activate the *ServerNet* as soon as possible. You will need to restart the application in order for this parameter to be taken into account. It is not available in 4D Server v14 R5 64-bit version for OS X, which only supports the *ServetNet*; (it always returns 0). **Possible values:** 0 or 1 (0 = do not use legacy layer, 1 = use legacy layer) **Default value:** 0 in databases created with 4D v14 R5 or higher, 1 in databases converted from 4D v14 R4 or earlier.
SQL Server Port ID	Longint	88	**Scope**: 4D local, 4D Server. **Kept between two sessions**: Yes **Description**: Gets or sets the TCP port number used by the integrated SQL server of 4D in local mode or 4D Server. By default, the value is 19812. When this selector is set, the database setting is updated. You can also set the TCP port number on the "SQL" page of the Database Settings dialog box. **Possible values**: 0 to 65535. **Default value**: 19812
Circular log limitation	Longint	90	**Scope**: 4D local, 4D Server. **Kept between two sessions**: No **Possible values**: Any integer value, 0 = keep all logs **Description**: Maximum number of files to keep in rotation for each type of log. By default, all files are kept. If you pass a value *X*, only the *X* most recent files are kept, with the oldest being erased automatically when a new one is created. This setting applies to each of the following log files: request logs (selectors 28 and 45), debug log (selector 34), events log (selector 79), as well as Web request logs and Web debug logs (selectors 29 and 84 of the **WEB SET OPTION** command).
Number of formulas in cache	Longint	92	**Scope**: 4D application **Kept between two sessions**: No **Possible values**: Positive longints **Default value**: 0 (no cache) **Description**: Sets or gets the maximum number of formulas to be kept in the cache of formulas, which is used by the **EXECUTE FORMULA** command. This limit is applied to all processes, but each process has its own formula cache. Caching formulas accelerates the **EXECUTE FORMULA** command execution in compiled mode since each cached formula is tokenized only once in this case. When you change the cache value, existing contents are reset even if the new size is larger than the previous one. Once the maximum number of formulas in the cache is reached, a new executed formula will erase the oldest one in the cache (FIFO mode). This parameter is only taken into account in compiled databases or compiled components.
Cache flush periodicity	Longint	95	**Scope**: 4D local, 4D Server **Kept between two sessions**: No **Possible values**: longint > 1 (seconds) **Description**: Gets or sets the current cache flush periodicity, expressed in seconds. Modifying this value overrides the **Flush Cache every X Seconds** option in the **Database/Memory page** of the Database settings for the

session (it is not stored in the Database settings).

Note: The *table* parameter is only used by selectors 31, 32, 46 and 47. In all other cases, it is ignored if it is passed.

Example 1

The following statement will avoid any unexpected timeout:

```
 `Increasing the timeout to 3 hours for the current process
SET DATABASE PARAMETER(4D Server Timeout;-60*3)
 `Executing a time-consuming operation with no control from 4D
...
WR PRINT MERGE(Area;3;0)
...
```

Example 2

This example temporarily forces the execution of a query by formula command on the client machine:

```
 curVal:=Get database parameter([table1];Query By Formula On Server) `Store the current setting
 SET DATABASE PARAMETER([table1];Query By Formula On Server;1) `Force execution on the client machine
 QUERY BY FORMULA([table1];myformula)
 SET DATABASE PARAMETER([table1];Query By Formula On Server;curVal) `Re-establish current setting
```

Example 3

You want to export data in JSON that contains a converted 4D date. Note that conversion occurs when the date is saved in the object, so you must call the **SET DATABASE PARAMETER** command before calling **OB SET**:

```
C_OBJECT($o)
SET DATABASE PARAMETER(JSON use local time;0)
OB SET($o ;"myDate";Current date) // JSON conversion
$json:=JSON Stringify($o)
SET DATABASE PARAMETER(JSON use local time;1)
```

WEB GET VARIABLES

WEB GET VARIABLES (nameArray ; valueArray)

Parameter	Type		Description
nameArray	Text array	⇦	Web form variable names
valueArray	Text array	⇦	Web form variable values

Description

The **WEB GET VARIABLES** command fills the text arrays *nameArray* and *valueArray* with the variable names and values contained in the Web form "submitted" (i.e. sent to the Web server).

This command gets the value for all the variables which can be included in HTML pages: text area, button, check box, radio button, pop up menu, choice list...

Note: Regarding check boxes, the variable name and value are returned only if the check box has been actually checked.

This command is valid regardless of the type of URL or form (POST or GET method) sent to the Web server.

This command can be called, if necessary, in the **On Web Connection Database Method** or any other 4D method resulting from a form submission.

About Web forms and their associated actions

Each form contains named data entry area (text area, buttons, checkboxes).
When a form is submitted (a request is sent to the Web server), the request contains (within others) the list of the data entry areas and their associated values.
A form can be submitted through two methods (both can be used with 4D):

- POST, usually used to add data into the Web server - to a database,
- GET, usually used to request the Web server - data coming from a database.

Example

A form contains two fields, vName and vCity with "ROBERT" and "DALLAS" values. The action associated to the form is "/4DACTION/WEBFORM".

- If the form method is POST (most frequently used), the data entered will not be visible in the URL (http://127.0.0.1/4DACTION/WEBFORM).
- If the form method is GET, the data entered will be visible in the URL (http://127.0.0.1/4DACTION/WEBFORM?vNAME=ROBERT&vCITY=DALLAS).

The WEBFORM method can be as follows:

```
ARRAY TEXT($anames;0)
ARRAY TEXT($avalues;0)
WEB GET VARIABLES($anames;$avalues)
```

The result will be:

```
$anames{1}="vNAME"
$anames{2}="vCITY"
$avalues{1}="ROBERT"
$avalues{2}="DALLAS"
```

The vNAME variable contains ROBERT and the vCITY variable contains DALLAS.

_o_Font number

_o_Font number (fontName) -> Function result

Parameter	Type		Description
fontName	String	⇨	Font name for which to return the font number
Function result	Longint	⇦	Font number

Description

This command is obsolete and must no longer be used beginning with 4D v14. It is kept for compatibility reasons but it will not be supported in future versions of the program.

SEND RECORD

SEND RECORD {(aTable)}

Parameter	Type		Description
aTable	Table	➡	Table from which to send the current record, or Default table, if omitted

Description

SEND RECORD sends the current record of *aTable* to the serial port or document opened by the **SET CHANNEL** command. The record is sent with a special internal format that can be read only by **RECEIVE RECORD**. If no current record exists, **SEND RECORD** has no effect.

The complete record is sent. This means that pictures and BLOBs stored in or with the record are also sent.

Important: When records are being sent and received using **SEND RECORD** and **RECEIVE RECORD**, the source table structure and the destination table structure must be compatible. If they are not, 4D will convert values according to the table definitions when **RECEIVE RECORD** is executed.

Note: If you send a record to a document using this command, the document must have been opened using the **SET CHANNEL** command. You cannot use **SEND RECORD** with a document opened with **Open document**, **Create document** or **Append document**.

Compatibility note: Beginning with version 11 of 4D, this command no longer supports subtables.

Example

See example for the **RECEIVE RECORD** command.

_o_CREATE SUBRECORD

_o_CREATE SUBRECORD (subtable)

Parameter	Type		Description
subtable	Subtable	⇨	Subtable for which to create a new subrecord

Compatibility note

Subtables are no longer supported starting with version 11 of 4D. A compatibility mechanism ensures the functioning of this command in converted databases; however, it is strongly recommended to replace any subtables with standard related tables.

_o_ADD DATA SEGMENT

_o_ADD DATA SEGMENT

Does not require any parameters

Description

Compatibility note: Starting with version 11 of 4D, data segments are no longer supported (the size of the data file is now unlimited). When it is called, this command does nothing.

Quick table

Explanation of values for the “Status” column:

- **Removed**: No longer available in the current version (or the version indicated).
- **Deprecated**: Should no longer be used and will be removed in a future major version
- **OS**: Depends on officially deprecated OS technologies (e.g.: PICT format). Status is the same as Deprecated, but an OS could remove the support before we do

Feature	Replacement	Status	Status in 4D 64-bit versions
Mac OS QuickDraw fonts	Font names	Deprecated	Removed
Altura Mac2Win	Plug-in developers: use native Windows code	Deprecated	Removed
Dynamic assignment of variables received via HTTP (compatibility option for databases created prior to v13.4)	**WEB GET VARIABLES** command (to recover variables). **WEB GET BODY PART**/**WEB Get body part count** commands (to recover posted files)	Deprecated	Deprecated
Non-Unicode mode (converted pre-v11 database)	Move to Unicode	Deprecated	Removed
QuickTime support (compatibility option available using **SET DATABASE PARAMETER**(QuickTimeSupport;1))	Use native formats	Deprecated	Removed
API QuickDraw	New SDK plug-in for third-party plug-ins	Deprecated	Removed
Converted subtables	Use N->1 tables	Deprecated	Deprecated
XSLT	Use PHP *libxslt* module or the **PROCESS 4D TAGS** command	Deprecated	Removed
4D Pack	Various integrated 4D commands or other technologies	Deprecated	Deprecated
Mac Resources	Use "Resources" folder. For compatibility, you can still use it in converted databases. We no longer support write access commands.	OS	OS (cicn icons: removed)
PICT	Use modern formats; help is provided by **GET PICTURE FORMATS**	OS	Removed

Previous documents

This document concerns the **4D v16 product range**. For reference, you can consult previous documents (PDF) describing deprecated features in prior product ranges, available here:

- Deprecated and Removed Features in 4D v15 - **(Rev. June 2015)**
- Deprecated and Removed Features in 4D v14 - **(Rev. Oct 2014)**
- Deprecated and Removed Features in 4D v13 - **(Rev. 20 Feb 2012)**
- Deprecated and Removed Features in 4D v12 - **(Rev. 03 Jun 2010)**