

Conversion to 4D v16

Welcome to this "Conversion to 4D v16" manual, which describes various points to be checked before, during, and after conversion of a 4D v15 database to 4D v16.









Conversion of 4D v15 databases should go smoothly in 4D v16, and we provide a few recommendations to make sure everything goes well in the "**Principles for conversion**" chapter. However, once the conversion is done, there are a couple of things to check, such as "**New compatibility options**" and "**Changes in behavior**" both at the application level and that of the 4D commands, which you need to understand in order to make the best use of new features in 4Dv16 .

And finally, this manual summarizes **deprecated features in 4D v16** that developers needs to spot quickly in order to assess the time needed to set up new functions.

Note: Some modifications listed in this manual when introduced during the 4D v15 "R-release" program.

When converting earlier or even much older databases, it is often necessary to use intermediate versions. And for the various points to check, refer to the conversion documents for previous versions:

- **4D v15:** "[Conversion to 4D v15](#)" and "[Deprecated features in 4D v15 and higher](#)".
- **4D v14:** "[Conversion to 4D v14](#)" and "[Deprecated features 4D v14 and higher](#)".
- **4D v13:** "[Conversion to 4D v13](#)" and "[Deprecated features 4D v13 and higher](#)".
- **4D v12:** "[Deprecated features 4D v12 and higher](#)" (there was no "Conversion" document for this version).
- **4D v11:** "[Conversion to 4D v11 SQL](#)".

-  Principles for conversion
-  Compatibility dialog
-  Changes in behavior
-  Name or theme changes
-  Obsolete functions
-  Disabled functions
-  Changing from 32-bit versions to 64-bit versions
-  Converting 4D Write documents to 4D Write Pro

Principles for conversion

What to do before converting

- You must have an **"interpreted" version** of the database (xxxx.4DB file for the structure), as well as the Designer password to perform a conversion;
- **Make a copy of your database** before conversion;
- Perform a syntax check. Even if you do not want to compile your database, this check can help warn you about possible errors;
- Use the **Maintenance and Security Center** to check and repair the structure and data;
- Check whether you have any PICTs using the **GET PICTURE FORMATS** command (or the 4D Pack **_o_AP Is Picture Deprecated** command) and convert them using the **CONVERT PICTURE** command (in 4D v14, there is still the possibility of using QuickTime in 32-bit versions by means of a selector for the **SET DATABASE PARAMETER** command);
- (Optional) Possible to implement primary keys if data journaling is needed (starting with version 14) (see **Primary keys** in the *Design Reference* manual).
- Since version 13.5, it is mandatory for **Unique** fields to be **indexed**. You will no longer be allowed to create/modify any records for a non-indexed Unique field: attempting to save the record will generate an error (-9998 Unique record exists, 1088 Index is invalid or missing). To create missing indexes, or generate a disk file listing all the non-indexed fields, refer to the **Appendix: Useful methods for conversion** of the **"Conversion to 4D v15"** document.

How to convert

Databases created with version 15 of 4D or 4D Server (as well as ones created in v11, v12, v13 and v14) are compatible with 4D version 16 (structure and data files). You can convert any interpreted structure file. To do this, just launch 4D v16 and open your structure file (xxx.4DB file) in interpreted mode.

A dialog warns you that the structure and data files are going to be converted:

After your structure file is converted to 4D v16, it can no longer be opened using a former version.

Data files are not converted for databases in 4D v15 or 4D v15 Rx. However, they are converted for databases in 4D v14 and earlier versions. In this case, a second dialog box is displayed:

This data file is also converted to version 16 but it can still be opened and used with 4D v14.4 and higher or 4D v15 (4D v14 R5).

After conversion

Use the **Maintenance and Security Center** (MSC) again to check and repair the structure and data.

As a reminder, on the **structure**:

- orphan methods (**__Orphan__xxxx**) are indicated by **warnings** in the log file of the MSC and can be deleted using the Explorer (after checking that their code is no longer useful);
- forms cannot contain duplicate object names: they are signaled by **warnings** in the log file of the MSC. You can perform a repair operation on the database in order to modify these names (in this case, make sure you check the programming of object names).

New feature concerning the **structure**:

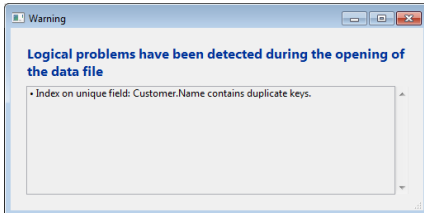
- Detection of pictures in the structure that contain a PICT format. See **Verifying the application** in the MSC chapter.

New features concerning the **data**: detection of duplicates in Unique fields. Additional information is provided:

- When using the MSC or a command such as **VERIFY DATA FILE**, the log files generated now include the names of the tables and fields concerned, as well as each duplicate value.

Note: When entering data, the "Duplicate key" error dialog box now includes the names of the tables and fields concerned, as well as the duplicate value, and the **GET LAST ERROR STACK** command also includes detailed information about any duplicates found.

When 4D opens a data file, if it is necessary to build (or rebuild) an index, duplicates are now detected automatically in any associated field(s) which are declared unique. In this case, a specific alert dialog box is displayed before opening the database, providing the user with the information needed to identify and remove the duplicates:



Rebuilding indexes

During the upgrade to 4D v16 and because of the update of the Unicode library (ICU - International Components for Unicode), all text and keyword indexes in 4D were rebuilt. This operation is performed automatically when the database is opened for the first time (warning: this operation may take a significant amount of time).

Similarly, when you re-open a v16 database with a 4D v15 R5 or earlier version, this automatically triggers the rebuilding of the text and keyword indexes.

Note: With 4D v16, we have significantly optimized the global reindexing algorithm for the data of the database. All its processes were reviewed and this operation can now be up to twice as fast. Global reindexing is required, for example, after repairing the database or when the .4dindx file has been deleted.

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.

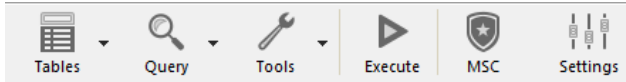
Example

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)
```

Compatibility dialog

To go to this dialog, you just need to click on the "Settings" icon in the main tool bar:



Then on the "Compatibility" tab.

Activating a new mechanism

A new option is added in 4D v16 on the **Compatibility** page: **Use new architecture for application deployments**.

- This option is **checked by default for databases created** with 4D v15 R4 and subsequent versions: the "new architecture" is automatically enabled; no additional setting is required in order to benefit from it.
- For compatibility reasons, this option is **unchecked by default in databases converted** from previous versions: you need to check it in order to benefit from the new features.

This option is available for all applications starting with 4D v16. It allows you to enable or disable new mechanisms related to the deployment of 4D applications (it must be defined on the machine generating the final application). For more details about the mechanisms controlled by this option, see the [Data file management in final applications](#) and [Management of connections by client applications](#) sections.

Former options

Other compatibility options can appear in this dialog box. They are added gradually over the successive versions, so the older the version in which your database was created, the more options you will have:

For more information about these options, see [Compatibility page](#).

Option removed

The "**Use 4DVAR Comments instead of Brackets**" option was removed from the **Compatibility** page of this dialog box in 4D v16.

Data file management in final applications

Opening the data file

When a user launches a merged application or an update (single-user or client-server applications), 4D tries to select a valid data file. Several locations are examined by the application successively.

The opening sequence for launching a merged application is:

1. 4D tries to open the **Last data file opened**, as described below (not applicable during initial launch).
2. If not found, 4D tries to open the data file in a **default data folder** next to the .4DC file in read-only mode (new in 4D v15, described below).
3. If not found, 4D tries to open the standard default data file (same name and same location as the .4DC file).
4. If not found, 4D displays a standard "Open data file" dialog box.

Last data file opened

Path of last data file

When the **Use new architecture for application deployments** compatibility option is checked (see [Compatibility page](#)), any standalone or server applications built with 4D stores the path of the last data file opened in the application's user preferences folder.

Compatibility note: In previous versions of the program, this information was stored in the structure file.

The location of the application's user preferences folder corresponds to the path returned by the following statement:

```
userPrefs:=Get 4D folder(Active 4D Folder)
```

The data file path is stored in a dedicated file, named *lastDataPath.xml*.

Thanks to this architecture, when you provide an update of your application, the local user data file (last data file used) is opened automatically at first launch.

This mechanism is usually suitable for standard deployments. However for specific needs, for example if you duplicate your merged applications, you might want to change the way that the data file is linked to the application. For more information, please refer to the next section "Configuring the data linking mode".

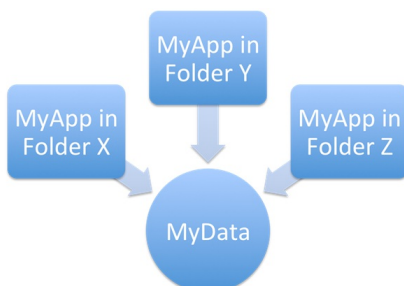
Configuring the data linking mode

With your compiled applications, 4D automatically uses the last data file opened. By default, when the new architecture is activated (starting with 4D v15 R4, see section above), the path of the data file is stored in the application's user preferences folder and is linked to the **application name**.

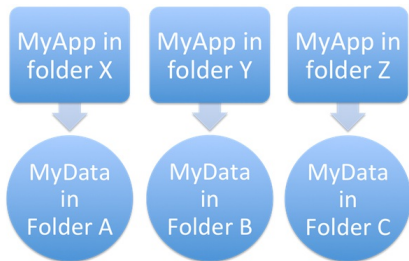
This may be unsuitable if you want to duplicate a merged application intended to use different data files. Duplicated applications actually share the application's user preferences folder and thus, always use the same data file -- even if the data file is renamed, because the last file used for the application is opened.

4D therefore lets you link the data file path to the **application path**. In this case, the data file will be linked using a specific path and will not just be the last file opened.

Duplication when data linked by application name:



Duplication when data linked by application path:



You can select the data linking mode during the build application process. You can either:

- Use the **Application page** or **Client/Server page** of the Build Application dialog box.
- Use the **LastDataPathLookup** (single-user application) or **LastDataPathLookup** (server application) XML key.

Defining a default data folder

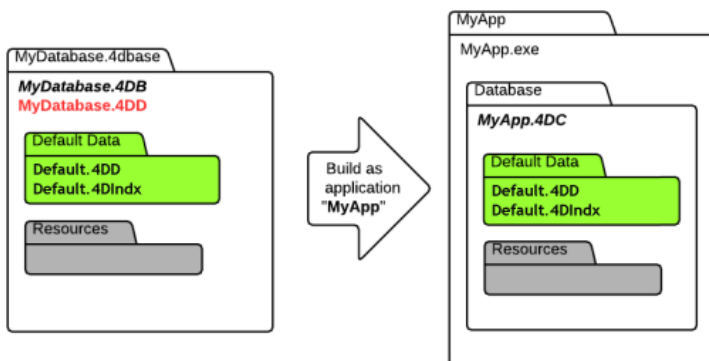
4D allows you to define a default data file file" at the application building stage. When the application is launched for the first time, if no local data file is found (see sequence described above), the default data file is automatically opened silently in read-only mode by 4D. This gives you better control over data file creation and/or opening when launching a merged application for the first time. More specifically, the following cases are covered:

- Avoiding the display of the 4D "Open Data File" dialog box when launching a new or updated merged application. You can detect, for example in the **On Startup database method**, that the default data file has been opened and thus execute your own code and/or dialogs to create or select a local data file.
- Allowing the distribution of merged applications with read-only data (for demo applications, for instance).

To define and use a default data file:

- You must provide a default data file (named "Default.4DD") and store it in a default folder (named "Default Data") inside the database package (4dbase). This file must be provided along with all other necessary files, depending on the database configuration: index (.4DIndx), external Blobs, journal, etc. It is your responsibility to provide a valid default data file. Note however that since a default data file is opened in read-only mode, it is recommended to uncheck the "Use Log File" option in the original structure file before creating the data file.
- When the application is built, the default data folder is integrated into the merged application. All files within this default data folder are also embedded.

The following graphic illustrates this feature:



When the default data file is detected at first launch, it is silently opened in read-only mode, thus allowing you to execute any custom operations that do not modify the data file itself.

The Compatibility page groups together parameters related to maintaining compatibility with previous versions of 4D. Keep in mind that the number of options displayed depends on the version of 4D with which the original database was created (2004.x, v11, v12, and so on), as well as the settings modified in this database.

Note: This page does not appear in databases created with the current version of 4D (non-converted databases).

- **Fields are enterable in dialog boxes:** In previous versions of 4D, it was not possible to enter values using fields in dialog boxes (displayed, for example, using the **DIALOG** command). This limitation has been removed since 4D 2004. You can still keep the previous behavior, especially if your database uses fields in dialog boxes to display data. By default, this option is checked for previous databases converted to version 2004 and is unchecked for databases created in version 2004.
- **Radio buttons grouped by name:** In previous versions of 4D, the coordinated behavior of a group of radio buttons was obtained by giving the same first letter to the variables associated with the buttons (for example, *m_button1*, *m_button2*, *m_button3*, etc.). Beginning with 4D 2004 this was changed as follows: to operate in a coordinated manner, a set of radio buttons must simply be grouped in the Form editor. For more information about this, refer to **Radio Buttons and Picture Radio Buttons**.
This new mode is valid for radio buttons, 3D radio buttons and picture radio buttons. For compatibility reasons, the former mode is kept by default in converted databases. However, you can force the use of the new mode by deselecting this option. Databases created in version 2004 use the new mode.
- **Reload form for each record during PRINT SELECTION:** In previous versions of 4D, the form used during a print using the **PRINT SELECTION** command was reloaded for each record. This allowed automatically reinitializing all object settings that the developer might have changed using language in the [On Printing Detail](#) form event.
In order to optimize performance, this mechanism was deleted beginning with 4D 2004. The 4D developer must now reinitialize the desired settings in the form method himself — this is identical to how list forms work with the [On Display Detail](#) form event. Nevertheless, you can keep the former mechanism using this option. Databases created in version 2004 use the new mode.
- **Do not use new context referencing mode:** When this option is not selected (default value), the 4D Web server places the context number in the basic URL of the HTML documents being sent.
With the former system (option checked), the 4D Web server sends the context number for each item of a page to the browser, which slows down processing. This option may nevertheless be checked for compatibility reasons. Keep in mind that you must restart the database after modifying this option in order for the new operation to become effective.
- **Remove "/" on unknown URLs:** In former versions of 4D, unknown URLs (URLs that do not correspond to an existing page nor to a 4D special URL) were returned in the On Web Authentication and On Web Connection (\$1) database methods and did not begin with the "/" character. This specificity was removed in 4D 2004. However, if you implemented algorithms based on this operation and wish to keep it, you can uncheck this option.
- **Prevent drop of data not coming from 4D:** Starting with v11, 4D allows drag and drop of selections, objects and/or files external to 4D, like picture files for example, in the Application mode.
This possibility must be supported by the database code. In databases converted from previous versions of 4D, this possibility may lead to malfunctioning if the existing code is not adapted accordingly.
This option can be used to anticipate this possible malfunctioning. When it is checked, the dropping of external objects is refused in 4D forms. Note that inserting external objects is still possible in objects having the **Automatic Drop** option, when the application can interpret the data being dropped (text or picture). For more information, refer to **Drag and Drop**.
- **Execute QUERY BY FORMULA On Server and Execute ORDER BY FORMULA OnServer:** Starting with 4D v11, for optimization purposes, the query and order "by formula" commands are executed on the server; only the result is returned to the client machine. This concerns the following commands: **QUERY BY FORMULA**, **QUERY SELECTION BY FORMULA** and **ORDER BY FORMULA**. When variables are called directly in the formula, the query is calculated with the value of the variable on the client machine. For example,

```
QUERY BY FORMULA([aTable];[aTable]aField=theValue)
```

will be executed on the server but with the contents of the *myvariable* variable of the client. On the other hand, this principle does not apply for formulas using methods that, themselves, call variables: in this case the

value of the variables is evaluated on the server.

In converted databases, this new functioning may affect existing algorithms. Consequently, by default in this context, these commands continue to be executed on the client machine. If you want to take advantage of the new algorithm in a converted database, you can simply check these options.

Note: This option can be set using the **SET DATABASE PARAMETER** command.

- **QUERY BY FORMULA Uses SQL Joins:** Starting with 4D v11, the **QUERY BY FORMULA** and **QUERY SELECTION BY FORMULA** commands carry out joins based on the SQL joins model. This means that it is not necessary for a structural automatic relation to exist between table A and table B in order to use a formula containing [Table_A]field_X=[Table_B]field_Y.

Since this mechanism could lead to malfunctioning in existing applications, it is deactivated by default in converted databases. It is recommended to activate it (after checking the code of the database) by checking this option in order to benefit from the optimization of the query by formula commands.

Notes:

- When the "SQL joins" mode is activated, the QUERY BY FORMULA and QUERY SELECTION BY FORMULA commands nevertheless use automatic relations set in the Structure editor in the following cases:
 - If the formula cannot be broken down into elements of the {field ;comparator ;value} form
 - If two fields of the same table are compared.
- This option can also be set per process using the **SET DATABASE PARAMETER** command.

- **Allow Nested Transactions:** Enables support of multi-level transactions. Beginning with v11, 4D accepts nested transactions on an unlimited number of levels. Since this new operation can lead to malfunctioning in databases developed with former versions of 4D, it is disabled by default in converted databases (transactions remain limited to a single level). If you want to take advantage of transactions on several levels in a converted database, you must check this option.

By default, this option is not checked. It is specific to each database.

Note: This option has no effect on transactions carried out in the SQL engine of 4D. SQL transactions are always multi-level.

- **Unicode mode:** Used to enable or disable the Unicode mode for the current database. In Unicode mode, the database engine, the language and the menus handle Unicode character strings natively. In non-Unicode mode (also called ASCII compatibility mode), the ASCII character set is used.

This option allows the compatibility of converted databases to be maintained. It is checked by default for databases created with 4D v11 and higher, and unchecked in converted databases.

Notes:

- This option is specific to each database. You can therefore have a Unicode database coexist with non-Unicode components (or vice versa) in interpreted mode.
- You can also configure the Unicode mode using the **SET DATABASE PARAMETER** command.

The specific characteristics of Unicode support in 4D are detailed in the *Language Reference* manual. For more information, refer to **EXPORT TEXT**.

- **Use period and comma as placeholders in numeric formats:** starting with v11, 4D uses regional system parameters for numeric display formats (see "Number formats" in **Display formats**). 4D automatically replaces the "," and "." characters in numeric display formats by, respectively, the thousand separator and the decimal separator defined in the operating system. The period and comma are thus considered as placeholder characters, following the example of 0 or #. In previous versions of 4D, numeric display formats do not take the regional parameters of the system into account. For example, the "###,##0.00" format is a valid format for an American system. However, when it is applied to a numeric value displayed on a French or Swiss system, the result is incorrect.

In converted database, for the sake of compatibility, this new mechanism is not activated. To take advantage of it, you must check this option.

- **Automatic variable assignment:** In previous versions of 4D, a standard mechanism of the Web server automatically recopied the value of variables sent by means of an HTTP form or a GET type URL into 4D process variables. In interpreted mode, the value of any variable received was copied directly into a 4D process variable with the same name; in compiled mode, the variables must have been pre-declared in a *COMPILER_WEB* project method.

Beginning with 4D v13.4, this mechanism is obsolete and no longer available in new databases. For compatibility, it is maintained in converted databases but you can disable it by unchecking this compatibility option. We now recommend using the dedicated **WEB GET VARIABLES** or **WEB GET BODY PART** commands.

- **Use legacy network layer (ignored on OS X 64-bit):** Starting with release v14 R5, 4D applications contain a new network layer, named *ServerNet*, to handle communications between 4D Server and remote 4D machines (clients). The former network layer has become obsolete, but it is kept to ensure compatibility with existing databases. Using this option, you can enable or disable the former network layer at any time in your converted 4D Server applications depending on your needs, for example, when migrating your client

applications (see the **Network and Client-Server options** section). *ServerNet* is used automatically for new databases, and disabled by default in converted databases (option checked).

Note that in case of a modification, you need to restart the application for the change to be taken into account. Any client applications that were logged must also be restarted to be able to connect with the new network layer (the minimum client version for using the *ServerNet* layer is 4D v14 R4, see the **Network and Client-Server options** section).

Notes:

- This option can also be managed by programming using the **SET DATABASE PARAMETER** command.
- As specified in its title, this option is ignored in the 64-bit version of 4D Server for OS X; only *ServerNet* can be used on this platform.

- **Save methods as Unicode:** allows you to save 4D method code strings in Unicode. In versions of 4D prior to v15, 4D method code strings (formulas, variable and method names, comments, etc.) were saved using the current local encoding. This encoding could cause issues, especially when 4D code was shared between developers from different countries: for example, if a French developer wrote some 4D code that included accents and then sent the database to an English developer, these accents would be lost. Serious issues could also occur with code written on Japanese versions. Saving methods as Unicode resolves all these types of issues and makes it possible to exchange 4D code containing specific local characters. We recommend that you enable the Unicode mode option for methods as soon as possible in your existing databases, especially if you work in an international environment.

Notes:

- This feature applies to the language itself and its interpretation. Some 4D editor windows, such as the Property list, still use current local encoding and therefore may display certain strings incorrectly. However, this does not affect code execution.
- If you modify this option, you need to restart the application in order for the change to be taken into account. You can check or uncheck this option at any time; only methods saved subsequently are affected.

- **Use new architecture for application deployments:** This option is available for all applications starting with 4D v15 R4. It enables or disables new mechanisms related to the deployment of 4D applications (it has to be set on the machine that generates the final application). The mechanisms controlled by this option are described in the **Last data file opened** and **Management of connections by client applications** sections. This option is unchecked by default in converted applications. In order to benefit from these new mechanisms, you will need to check it explicitly.

Changes in behavior

Licenses

License management for 4D products has been improved in 4D v16:

First activation simplified: Entering a new license number in the "License Manager" dialog box now automatically activates, in a single operation, 4D Server and all of its related expansions (additional clients, plug-ins, etc.)

New Refresh button: You can now activate your licenses by simply clicking on the Refresh button in the "License Manager" dialog box.

This new button connects you to our customer database and automatically activates all your new licenses or updates linked to the current license (the current license appears in bold in the Active Licenses list). You just need to enter your 4D identifiers (account and password). You can click on the **Refresh** button in the following contexts:

- when you have acquired an additional expansion and want to activate it,
- when you need to update an expired temporary number (Partners or evolutions).

New auto-activation feature: This feature is triggered when you launch a more recent 4D product for which you have not yet entered your license, or when the license detected on the machine where the product was launched is not valid. The auto-activation procedure starts:

- when you open/create a local interpreted database with 4D Developer Edition. In this case, a dialog box informs you that you will be connected to our customer database and your licenses will be activated (you will have to enter the password for your user account).
- when you launch a 4D Server application. In this case, in order to allow its use as a service or to permit automatic updates, the auto-activation process is transparent to the user and entirely automatic (no dialog box is displayed).

Language

OBJECT SET FORMAT / OBJECT Get format: These commands now support icons in list box headers.

METHOD GET CODE: This command returns code as indented text.

DELETE FOLDER: This command can now delete a folder that is not empty

Fonts

The **FONT LIST** command under Windows only returns vectorial fonts.

Printing

64-bit versions only: The new features detailed in this section are only available in 4D v16 64-bit versions (4D Developer Edition and 4D Volume Desktop, see the **Printing architecture (redesign)** section).

The printing architecture was entirely rewritten in 4D 64-bit versions to benefit from the most recent OS-based printing libraries and dialog boxes. Although this internal update is mostly transparent to 4D users, the following changes should be noted:

- The **Print job** dialog box (Windows and OS X) has been upgraded and is a standard system dialog on both platforms.
- On Windows, the **Page Setup** dialog box has been updated; it is now provided by the operating system.
- **PRINT SETTINGS** command: The **Page Setup** dialog box is not longer displayed automatically when a print command is called. To display it, you need to use the [Page_setup_dialog](#) constant in the *dialType* parameter. A second constant has been added to this command: [Print_dialog](#) lets you indicate whether or not to display the print dialog.
- The following print options (used with **GET PRINT OPTION** or **SET PRINT OPTION** commands) have been

modified:

Option (constant)	OS	Status in 4D v16	Comments
2 (Orientation option)	Windows and OS X	<i>Updated</i>	Can be called within a print job, which means that you can switch to portrait or landscape orientation in the same print job.
8 (Color option)	Windows only	<i>Removed</i>	Deprecated
13 (Mac spool file format option)	OS X only	<i>Removed</i>	Replaced by new option of the SET CURRENT PRINTER command.

Note: The **OPEN PRINTING JOB**, **CLOSE PRINTING JOB**, **SET PRINT OPTION** and **SET PRINT OPTION** commands are compatibles with the 4D Write Pro **WP PRINT** command: for more information, see **WP PRINT**. All options are supported for 4D Write Pro documents, except for the Paper option and Orientation option options, for which we recommend using the **WP USE PAGE SETUP** command instead in order to set the page size and orientation separately.

List boxes

Row Control Array

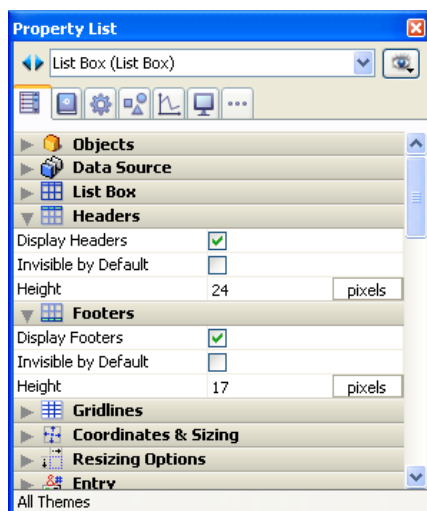
A new **Row Control Array** property gives you the ability to control new interface properties:

- hidden or visible (visible by default)
- enabled or disabled (enabled by default)
- selectable or not selectable (selectable by default)

The **Row Control Array** property can be set or get using the **LISTBOX SET ARRAY** and **LISTBOX Get array** commands. The array can also be returned by the **LISTBOX GET ARRAYS** command.

In previous versions of 4D, this property was named "**Hidden Rows Array**" and expected a Boolean array. For compatibility's sake, a Boolean array supported as a row control array. In this array, each element represents the hidden/displayed status of its corresponding row in the list box. **True** means that the row is hidden and **False** means that it is displayed.

Headers and footers



The minimum height for headers, in pixels, depends on the system. If you pass a value that is too small, it is replaced by the minimum size defined in the system for headers. There is no minimum size for rows and footers.

Under Windows 7, the minimum header height is 24 pixels. Any headers in your converted databases whose height is less than this are automatically resized to 24 pixels.

You can also set the height of headers and footers dynamically using the **LISTBOX SET HEADERS HEIGHT** and **LISTBOX SET FOOTERS HEIGHT** commands.

Since the rendering may not entirely match your expectations, this is a something you must remember to check on your forms.

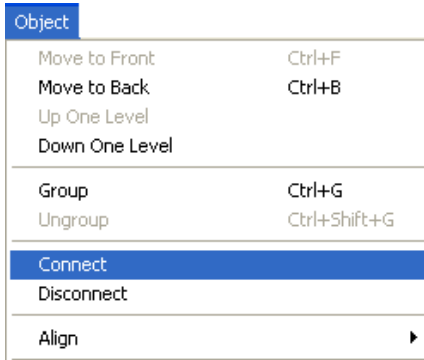
Converted list boxes

List boxes resulting from the conversion of former grouped scrollable areas are **connected**. Connected list boxes

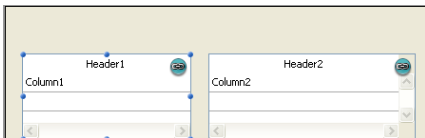
function in a coordinated manner: selecting a row in one list box leads to the same row being selected in any other list boxes belonging to the same connected group; scrolling a list box vertically triggers the same scrolling in all the list boxes belonging to the same connected group.

Note: Converted list boxes are also **grouped** in the form (standard 4D function).

List boxes can be connected and disconnected using the **Connect** and **Disconnect** commands found in the **Object** menu of the Form editor:



These commands are enabled when several list boxes are selected in a form. When a connected list box (i.e. a list box belonging to a connected group) is selected, a specific "badge" is displayed on all the list boxes that belong to this same connected group:

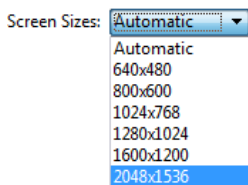


These principles allow you to reproduce the same operation of the former grouped scrollable areas; however, we recommend that you adapt your converted forms to use standard list box features.

Forms

The advanced options of the Form wizard have been updated based on the 4D product and hardware evolutions:

- The **Screen Sizes** list now includes "2048x1536" resolution:



- In the forms generated, the **Variable name** property is now left blank for navigation buttons.

Optimization of Replace string

Thanks to a new internal algorithm, the execution of the [#cmd id="233"/] command has been significantly accelerated in 4D v15 R3 when you replace a string by another of a different length. This is the case for example in the following replacements:

```
vResult:=Replace string(Source_Text;"a";"aa") //based on characters  
vResult2:=Replace string(Source_Text2;"à";"aa";*) //based on character codes
```

The new algorithm is optimized for both syntaxes: the larger the source text and the more replacements there are, the more significant the optimization will be.

Our benchmarks show the following results, compared with the previous algorithm:

Replacements based on character code (* passed) **Replacements based on character (* omitted)**

About 950 times faster

About 4400 times faster

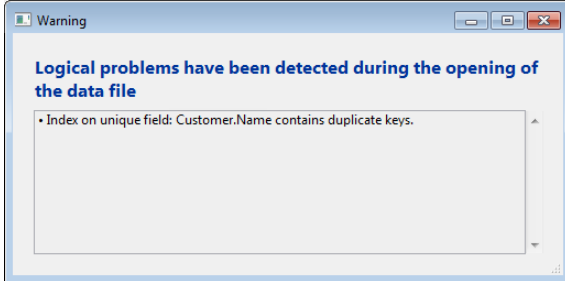
These tests were done by replacing "a" with "aa" in a file containing 32,000 occurrences to replace.

Note: Replacement of strings of the same length is just as fast as with the previous algorithm.

Duplicates in Unique fields

Additional information is provided when duplicates are detected in unique fields:

- When using the MSC or a command such as **VERIFY DATA FILE**, resulting logs now contain the names of the offending tables and fields, as well as each duplicated value
- During data entry, the "Duplicated key" error dialog box now contains the name of the offending table and field, as well as the duplicated value
- The **GET LAST ERROR STACK** command also contains detailed information on any duplicates.
- When 4D opens a data file, if an index needs to be built (or rebuilt), duplicates are now automatically detected in the associated field(s) declared as unique. In this case, a specific warning dialog box is displayed before database opening, providing the user with the information needed to identify and remove duplicates:



WEB: 4D tags and decimal separators

4D always uses the period character (.) as decimal separator when evaluating a numerical expression using the **4DTEXT**, **4DVAR**, **4DHTML**, **4DHTMLVAR** and **4DEVAL** tags. Regional settings are now ignored in this context.

For example, whatever the regional settings are:

```
value:=10/4
input:="<!--#4DTEXT value-->"
PROCESS 4D TAGS (input;output)
// always outputs 2.5 even if regional settings use the ',' as separator
```

So if your code evaluates numerical expressions using 4D tags with respect to the regional settings, you need to adapt it using the **String** command:

- To get *value* with a period as decimal point: `<!--#4DTEXT value-->`
- To get *value* with a decimal point based on the regional settings: `<!--#4DTEXT String(value)-->`

For more information, refer to **4D Transformation Tags**.

HTTP server

Disabling of the HTTP TRACE method, error 405. If you need to enable this method, you can use the [Web HTTP TRACE](#) option with the **WEB SET OPTION** command.

Timestamping of maintenance log files

The names of log files generated during maintenance operations through the MSC or the 4D Server administration window are now unique and therefore differ each time they are saved to disk. In previous versions, these files always used the same name so the previous log file (if any) was overwritten by the new one each time a new maintenance operation was performed, meaning that prior log files were automatically purged. It is now up to the database administrator to remove older log files as necessary, both for 4D and 4D Server.

4D Internet Commands

Modifications between versions v15.x/v15Rx and v16: concerning management of encoding and charsets, in particular for attachment filenames when sending emails. In databases where workarounds were implemented, **you need to verify that these modifications do not cause any malfunctions.**

Two commands were updated: SMTP_Charset and SMTP_SetPrefs.

- **SMTP_Charset**
Attachment filenames are encoded in base64
- Value 0 for parameters indicates using the default value (and not "Do not manage"), which means:

For *encodeHeaders*: UTF-8 charset for "Subject", ISO-8859-1 for other fields

For *bodyCharset*: UTF-8 charset encoded in base 64

- Value 1 for parameters indicates using values defined by the SMTP_SetPrefs command.

- **SMTP_SetPrefs**

The second parameter (renamed *charset&Encoding*) indicates the charset and encoding used in the message body, as well as the charset of headers and attachment filenames to be sent. The documentation was clarified and lists all the combinations accepted:

Value	Body charset and encoding	Headers and attachment filenames charset (encoding always base64)
-1	No change	No change
0	Application & binary; no encoding	ISO-8859-1
1	Default: UTF-8 & base64	Default: UTF-8 for subject, ISO-8859-1 for other fields
2	US-ASCII & 7bit	ISO-8859-1
3	US-ASCII & quotable-printable	ISO-8859-1
4	US-ASCII & base 64	ISO-8859-1
5	ISO-8859-1 & quotable-printable	ISO-8859-1
6	ISO-8859-1 & base64	ISO-8859-1
7	ISO-8859-1 & 8bit	ISO-8859-1
8	ISO-8859-1 & binary	ISO-8859-1
9	Reserved	Reserved
10	ISO-2022-JP (Japanese) & 7bit	ISO-2022-JP
11	ISO-2022-KR (Korean) & 7 bits	ISO-2022-KR
12	ISO-2022-CN (Traditional & Simplified Chinese) & 7 bit	ISO-2022-CN
13	HZ-GB-2312 (Simplified Chinese) & 7 bit	HZ-GB-2312
14	Shift-JIS (Japanese) & base64	Shift-JIS
15	UTF-8 & quoted-printable	UTF-8
16	UTF-8 & base64	UTF-8

OBJECT SET FORMAT

OBJECT SET FORMAT ({ * ; } object ; displayFormat)

Parameter	Type	Description
*	Operator	⇒ If specified, Object is an Object Name (String) If omitted, Object is a Field or a Variable
object	Form object	⇒ Object Name (if * is specified), or Field or Variable (if * is omitted)
displayFormat	String	⇒ New display format for the object

Description

OBJECT SET FORMAT sets the display format for the objects specified by *object* to the format you pass in *displayFormat*. The new format is only used for the current display; it is not stored with the form.

If you specify the optional * parameter, you indicate an object name (a string) in *object*. If you omit the optional * parameter, you indicate a field or a variable in *object*. In this case, you specify a field or variable reference (field or variable objects only) instead of a string. For more information about object names, see the [Object Properties](#) section.

OBJECT SET FORMAT can be used for both input forms and output forms (displayed or printed) and can be applied to fields or variables (enterable/non-enterable).

Naturally, you must use a display format compatible with the type of data found in the object or with the object itself.

Boolean

To format Boolean fields, there are two possibilities:

- You can pass a single value in *displayFormat*. In this case, the field will be displayed as a checkbox and its label will be the value specified.
- You can pass two values, separated by a semicolon (;), in *displayFormat*. In this case, the field will be displayed as two radio buttons.

Date

To format Date fields or variables, pass **Char(n)** in *displayFormat*, where *n* is one of the following predefined constants provided by 4D:

Constant	Type	Value	Comment
Blank if null date	Longint	100	"" instead of 0
Date RFC 1123	Longint	10	
Internal date abbreviated	Longint	6	Dec 29, 2006
Internal date long	Longint	5	December 29, 2006
Internal date short	Longint	7	12/29/2006
Internal date short special	Longint	4	12/29/06 (but 12/29/1896 or 12/29/2096)
ISO Date	Longint	8	2006-12-29T00:00:00 (deprecated)
ISO Date GMT	Longint	9	2010-09-13T16:11:53Z
System date abbreviated	Longint	2	Sun, Dec 29, 2006
System date long	Longint	3	Sunday, December 29, 2006
System date short	Longint	1	12/29/2006

Note: The [Blank if null date](#) constant must be added to the format; it indicates that in the case of a null value, 4D must display an empty area instead of zeros.

Time

To format Time fields or variables, pass **Char(n)** in *displayFormat*, where *n* is one of the following predefined constants provided by 4D:

Constant	Type	Value	Comment
Blank if null time	Longint	100	"" instead of 0
HH MM	Longint	2	01:02
HH MM AM PM	Longint	5	1:02 AM
HH MM SS	Longint	1	01:02:03
Hour min	Longint	4	1 hour 2 minutes
Hour min sec	Longint	3	1 hour 2 minutes 3 seconds
ISO time	Longint	8	0000-00-00T01:02:03
Min sec	Longint	7	62 minutes 3 seconds
MM SS	Longint	6	62:03
System time long	Longint	11	1:02:03 AM HNEC (Mac only)
System time long abbreviated	Longint	10	1•02•03 AM (Mac only)
System time short	Longint	9	01:02:03

Note: The Blank if null time constant must be added to the format; it indicates that in the case of a null value, 4D must display an empty area instead of zeros.

Picture

To format Picture fields or variables, pass **Char(n)** in *displayFormat*, where *n* is one of the following predefined constants provided by 4D:

Constant	Type	Value
On background	Longint	3
Replicated	Longint	7
Scaled to fit	Longint	2
Scaled to fit prop centered	Longint	6
Scaled to fit proportional	Longint	5
Truncated centered	Longint	1
Truncated non centered	Longint	4

Alpha and number

To format fields or variables of the Alpha or Number type, pass the label of the format directly in the *displayFormat* parameter.

For more information about display formats, see the **Number formats** and **Alpha formats** sections of the 4D *Design Reference* manual.

Note: In *displayFormat*, to use custom display formats that you may have created in the tool box, prefix the name of the format with a vertical bar (|).

Picture buttons

To format picture buttons, in the *displayFormat* parameter, pass a character string respecting the following syntax: *cols;lines;picture;flags{;ticks}*

- *cols* = number of columns in the picture.
- *lines* = number of lines in the picture.
- *picture* = picture used, coming from the picture library or a picture variable:
 - If the picture comes from the picture library, enter its number, preceded by a question mark (e.g.: "? 250").
 - If the picture comes from a picture variable, enter the variable name.
- *flags* = display mode and operation of a picture button. This parameter can take any of the following values: 0, 1, 2, 16, 32, 64 and 128. Each of these values represents a display mode or an operation mode. These values are cumulative; for instance, if you want to enable the modes 1 and 64, pass 65 in the *flags* parameter. Here are the details for each value:
 - *flags* = 0 (no option)
 - Displays the next picture in the series when the user clicks the picture. Displays the previous picture in the series when the user holds down the Shift key and clicks on the picture. When the user reaches the last picture in the series, the picture does not change when the user clicks it again. That is, it does not cycle back to the first picture in the series.
 - *flags* = 1 (Switch Continuously)

Similar to the previous option except that the user can hold down the mouse button to display the pictures continuously (i.e., as an animation). When the user reaches the last picture, the object does not cycle back to the first picture.

- *flags* = 2 (Loop Back to First Frame)

Similar to the previous option except that the pictures are displayed in a continuous loop. When the user reaches the last picture and clicks again, the first picture appears, and so forth.

- *flags* = 16 (Switch when Roll Over)

The contents of the picture button are modified when the mouse cursor passes over it. The initial picture is re-established when the cursor leaves the button's area. This mode is frequently used in multimedia applications or in HTML documents. The picture that is then displayed is the last picture of the thumbnail table, unless the Use Last Frame as Disabled option is selected (128). If that option is selected, it is the next-to-last thumbnail that is displayed.

- *flags* = 32 (Switch Back when Released)

This mode operates with two pictures. It displays the first picture all the time except when the user clicks the button. In that case, the second picture is displayed until the mouse button is released, whereupon it switches back to the first picture. This mode allows you to create an action button that displays its status (idle or clicked). You can use this mode to create a 3D effect or display any picture that depicts the action.

- *flags* = 64 (Transparent)

Used to make the background picture transparent.

- *flags* = 128 (Use Last Frame as Disabled)

This mode allows you to set the last thumbnail as the thumbnail to display when the button is disabled. When this mode is selected, 4D displays the last thumbnail when the button is disabled. When this mode is used in addition to the modes 0, 1 and 2, the last thumbnail is not taken into account in the sequence of the other modes. It will appear only when the button is disabled.

- *ticks* = activates the "Switch every n Ticks" mode and sets the time interval between the display of each picture. When this optional parameter is passed, it allows you to cycle through the contents of the picture button at the specified speed. For example, if you enter "2;3;?16807;0;10", the picture button will display a different picture every 10 ticks. When this mode is active, only the Transparent mode can be used (64).

Picture pop-up menus

To format picture pop-up menus, in the *displayFormat* parameter, pass a character string respecting the following syntax:

cols;lines;picture;hMargin;vMargin;flags

- *cols* = number of columns in the picture.
- *lines* = number of lines in the picture.
- *picture* = picture used, coming from the picture library or a picture variable:
 - If the picture comes from the picture library, enter its number, preceded by a question mark (e.g.: "?250").
 - If the picture comes from a picture variable, enter the variable name.
- *hMargin* = margin in pixels between the horizontal limits of the menu and the picture.
- *vMargin* = margin in pixels between the vertical limits of the menu and the picture.
- *flags* = transparency mode of picture pop-up menu. Accepts the values 0 and 64:
 - *flags* = 0: the picture pop-up menu is not transparent,
 - *flags* = 64: the picture pop-up menu is transparent.

Thermometers and rulers

To format objects of the thermometer or ruler type, in the *displayFormat* parameter, pass a character string respecting the following syntax:

min;max;unit;step;flags{;format{;display}}

- *min* = value of the first graduation of the indicator.
- *max* = value of the last graduation of the indicator.
- *unit* = interval between the indicator graduations.
- *step* = minimum interval of cursor movement in the indicator.
- *flags* = display mode and operation of indicators. This parameter accepts the values 0, 2, 3, 16, 32 and 128. These values can be accumulated in order to set several options (except for 128). Here are the details for each value:
 - *flags* = 0: does not display the units.
 - *flags* = 2: displays the units on the right or below the indicator.
 - *flags* = 3: displays the units on the left or above the indicator.

- *flags* = 16: displays graduations adjacent to the units.
- *flags* = 32: [On Data Change](#) is executed while the user is adjusting the indicator. If this value is not used, [On Data Change](#) occurs only after the user is finished adjusting the indicator.
- *flags* = 128: activates the "Barber shop" (continuous animation) mode. This value cannot be combined with others. In this mode, the other parameters are ignored (except for the *display* parameter if passed). For more information about this mode, please refer to the *Design Reference* manual.
- *format* = display format of the indicator graduations.
Keep in mind that the units and graduations are automatically hidden if the size of the indicator object does not permit them to be displayed correctly.
- *display* = specific display options. In the case of thermometers, this parameter is only taken into account when the *flags* subparameter is 128.
 - *display* = 0 (or is omitted): displays a standard ruler / displays a thermometer in continuous animation of the "barber shop" type.
 - *display* = 1 : activates "Stepper" mode for a ruler / activates the "Asynchronous progress" mode for a thermometer. For more information about these options, please refer to the *Design Reference* manual.

Dials

To format objects of the dial type, in the *displayFormat* parameter, pass a character string respecting the following syntax:

min;max;unit;step{;flags}

- *min* = value of the first graduation of the indicator.
- *max* = value of the last graduation of the indicator.
- *unit* = interval between the indicator graduations.
- *step* = minimum interval of cursor movement in the indicator.
- *flags* = operation mode of the dial (optional). This parameter only accepts the value 32: [On Data Change](#) is executed while the user is adjusting the indicator. If this value is not used, [On Data Change](#) occurs only after the user is finished adjusting the indicator.

Button grids

To format button grids, in the *displayFormat* parameter, pass a character string respecting the following syntax:

cols;lines

- *cols* = number of columns of the grid.
- *lines* = number of lines of the grid.

Note: For more information about the display formats for form objects, refer to the 4D Design Reference manual.

3D buttons

To format 3D buttons, in the *displayFormat* parameter, pass a character string respecting the following syntax:

title;picture;background;titlePos;titleVisible;iconVisible;style;horMargin;vertMargin;iconOffset;popupMenu;hyperlink;numStates

- *title* = Button title. This value can be expressed as text or a resource number (ex.: ":16800,1")
- *picture* = Picture linked to a button that comes from a picture library or a picture variable:
 - If the picture comes from a picture library, enter its number, preceded with a question mark (ex.: "?250").
 - If the picture comes from a picture variable, enter the variable name.
 - If the picture comes from a file stored in the Resources folder of the database, enter a URL of the type "# {folder}/picturename" or "file: {folder}/picturename".
- *background* = Background picture linked to a button (Custom style), that comes from a picture library, a picture variable, a PICT resource or a file stored in the Resources folder (see above).
- *titlePos* = position of the button title. Five values are possible:
 - *titlePos* = 1: Left
 - *titlePos* = 2: Top
 - *titlePos* = 3: Right
 - *titlePos* = 4: Bottom
 - *titlePos* = 5: Middle
- *titleVisible* = Defines whether or not the title is visible. Two values are possible:
 - *titleVisible* = 0: the title is hidden
 - *titleVisible* = 1: the title is displayed
- *iconVisible* = Defines whether or not the icon is visible. Two values are possible:
 - *iconVisible* = 0 : the icon is hidden

- *iconVisible* = 1 : the icon is displayed
- *style* = Button style. The value of this option determines whether various other options are taken into consideration (for example, background). The following values are possible:
 - *style* = 0: None
 - *style* = 1: Background offset
 - *style* = 2: Push button
 - *style* = 3: Toolbar button
 - *style* = 4: Custom
 - *style* = 5: Circle
 - *style* = 6: Small system square
 - *style* = 7: Office XP
 - *style* = 8: Bevel
 - *style* = 9: Rounded bevel
 - *style* = 10: Collapse/Expand
 - *style* = 11: Help
 - *style* = 12: OS X Textured
 - *style* = 13: OS X Gradient
- *horMargin* = Horizontal margin. Number of pixels delimiting the inside left and right margins of the button (areas that the icon and the text must not encroach upon).
- *vertMargin* = Vertical margin. Number of pixels delimiting the inside top and bottom margins of the button (areas that the icon and the text must not encroach upon).
- *iconOffset* = Shifting of the icon to the right and down. This value, expressed in pixels, indicates the shifting of the button icon to the right and down when the button is clicked (the same value is used for both directions).
- *popupMenu* = Association of a pop-up menu with the button. Three values are possible:
 - *popupMenu* = 0: No pop-up menu
 - *popupMenu* = 1: With linked pop-up menu
 - *popupMenu* = 2: With separate pop-up menu
- *hyperlink* = Title is underlined on mouseover to resemble a hyperlink (legacy mechanism). Two values are possible:
 - *hyperlink* = 0: title is not underlined on mouseover
 - *hyperlink* = 1: title is underlined on mouseover
- *numStates* = Number of states present in picture used as icon for the 3D button, and which will be used by 4D to represent the standard button states (from 0 to 4).

Certain options are not taken into account for all 3D button styles. Also, in certain cases, you may wish to not change all the options. To not pass an option, simply omit the corresponding value. For example, if you do not want to pass the *titleVisible*, *vertMargin* and *hyperlink* options, you can write:

```
OBJECT SET FORMAT (myVar;"NiceButton;?256;:562;1;;1;4;5;;5;0;;2")
```

List box headers

To format the icon in a list box header, pass a character string in the *displayFormat* parameter, which respects the following syntax:

picture;iconPos

- *picture* = header picture, coming from the picture library, a picture variable, or a picture file:
 - If the picture comes from the picture library, enter its number, preceded by a question mark (e.g.: "?250").
 - If it comes from a picture variable, enter the variable name.
 - If it comes from a file stored in the Resources folder of the database, enter a URL of the type "#{folder/}picturename" or "file:{folder/}picturename".
- *iconPos* = position of icon in header. Two values are supported:
 - *iconPos* = 1: Left
 - *iconPos* = 2: Right

This feature is useful, for example, when you want to work with a customized sort icon.

Example 1

The following line of code formats the *[Employee]Date Hired* field to the fifth format (Internal date long).

```
OBJECT SET FORMAT ([Employee]Date Hired;Char (Internal date long))
```

Example 2

The following example changes the format for a `[Company]ZIP Code` field according to the length of the value stored in the field:

```
If (Length ([Company]ZIP Code)=9)
  OBJECT SET FORMAT ([Company]ZIP Code;"#####-####")
Else
  OBJECT SET FORMAT ([Company]ZIP Code;"#####")
End if
```

Example 3

The following example formats the value of the `[Stats]Results` field depending on whether it is a positive, negative, or null number:

```
OBJECT SET FORMAT ([Stats]Results;"### ##0.00; (### ##0.00);")
```

Example 4

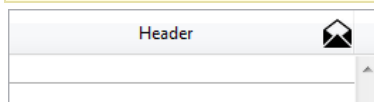
The following example sets the format of a Boolean field to display Married and Unmarried, instead of the default Yes and No:

```
OBJECT SET FORMAT ([Employee]Marital Status;"Married;Unmarried")
```

Example 5

Provided that you have stored a picture file named "envelope_open.png" in the Resources folder of the database, you can write:

```
vIcon:="#envelope_open.png"
vPos:="2" // Right
OBJECT SET FORMAT (*;"Header1";vIcon+";"+vPos)
```



Example 6

The following example sets the format of a Boolean field to display a checkbox labelled "Classified":

```
OBJECT SET FORMAT ([Folder]Classification;"Classified")
```

Example 7

You have a table of thumbnails containing 1 row and 4 columns, intended to display a picture button ("default", "clicked", "roll over" and "disabled"). You want to associate the Switch when Roll Over, Switch back when Released and Use Last Frame as Disabled options with it:

```
OBJECT SET FORMAT (*;"PictureButton";"4;1;?15000;176")
```

Example 8

Switching a thermometer to "Barber shop" mode:

```
OBJECT SET FORMAT ($Mythermo;";;;128")
```

```
$Mythermo:=1 `Start animation
```

⚙️ METHOD GET CODE

METHOD GET CODE (path ; code {; option} {; *})

Parameter	Type	Description
path	Text, Text array	⇒ Text or Text array containing one or more method path(s)
code	Text, Text array	⇐ Code of designated method(s)
option	Longint	⇒ 0 or omitted = simple export (without tokens), 1 = export with tokens
*	Operator	⇒ If passed = command applies to host database when executed from a component (parameter ignored outside of this context)

Description

The **METHOD GET CODE** command returns, in the *code* parameter, the contents of the method(s) designated by the *path* parameter. This command can return the code of all types of methods: database methods, triggers, project methods, form methods and object methods.

You can use two types of syntaxes, based either on text arrays, or text variables:

```
C_TEXT(tVpath) // text variables
C_TEXT(tVcode)
METHOD GET CODE (tVpath;tVcode) // code of a single method
```

```
ARRAY TEXT (arrPaths;0) // text arrays
ARRAY TEXT (arrCodes;0)
METHOD GET CODE (arrPaths;arrCodes) // code of several methods
```

You cannot mix the two syntaxes.

If you pass an invalid pathname, the *code* parameter is left empty and an error is generated.

In the text of the *code* generated by this command:

- Command names are written in English for all versions of 4D, except when you use a French version and check the "Use regional system settings" preference (see [Methods Page](#)). When you use the *option* parameter, the code can contain language tokens in order to make it independent from the 4D programming language and version (see below).
- To increase code readability, text is indented with tab characters based on programming structures, like in the Method editor.
- A line is added in the header of the code generated containing metadata used when importing code, for example:

```
// %attributes = {"lang":"en","invisible":true,"folder":"Web3"}
```

During an import, this line is not imported, it is only used to set the corresponding attributes (attributes that are not specified are reset to their default value). The "lang" attribute sets the export language and prevents an import into an application in a different language (in this case, an error is generated). The "folder" attribute contains the name of the method's parent folder; it is not shown when the method does not have a parent folder.

Additional attributes can be defined. For more information, refer to the description of the **METHOD SET ATTRIBUTES** command.

The *option* parameter allows you to select the code export mode with respect to the tokenized language elements of the method(s):

- If you pass 0 or omit the *option* parameter, the method code is exported without tokens, i.e. just like it is displayed in the Method editor.
- If you pass 1 or the [Code with tokens](#) constant, the method code is exported with tokens, i.e. tokenized elements are followed by their internal reference in the *code* exported contents. For example, the expression "**String**(a)" is exported "**String**:C10(a)", where "C10" is the internal reference of the **String** command.

Tokenized language elements include:

- 4D commands and constants,
- Table and field names,
- 4D plug-in commands.

Code exported with tokens is independent from any subsequent renaming of language elements. Thanks to tokens, code provided as text will always be interpreted correctly by 4D, whether by means of the **METHOD SET CODE** command or even by copy/paste. For more information about the syntax of 4D tokens, please refer to **Using tokens in formulas**.

If the command is executed from a component, it applies by default to the component methods. If you pass the * parameter, it accesses the methods of the host database.

Example 1

Refer to the example of the **METHOD SET CODE** command.

Example 2

This example illustrates the effect of the *option* parameter.

You want to export the following "simple_init" method:

```
Case of
  : (Form event=On_Load)
    ALL RECORDS([Customer])
End case
```

If you execute the following code:

```
C_TEXT($code)
C_TEXT($contents)
$code:=METHOD Get path(Path_project_method;"simple_init")
METHOD GET CODE($code;$contents;0) //no tokens
TEXT TO DOCUMENT("simple_init.txt";$contents)
```

The resulting document will contain:

```
//%attributes = {"lang":"en"} comment added and reserved by 4D
Case of
  : (Form event=On_Load)
    ALL RECORDS([Customer])
End case
```

If you execute the following code:

```
C_TEXT($code)
C_TEXT($contents)
$code:=METHOD Get path(Path_project_method;"simple_init")
METHOD GET CODE($code;$contents;Code_with_tokens) //use tokens
TEXT TO DOCUMENT("simple_init.txt";$contents)
```

The resulting document will contain:

```
//%attributes = {"lang":"en"} comment added and reserved by 4D
Case of
  : (Form event:C388=On_Load;K2:1)
    ALL RECORDS:C47([Customer:1])
End case
```


DELETE FOLDER

DELETE FOLDER (folder {; deleteOption})

Parameter	Type		Description
folder	String	→	Name or full path of the folder to be deleted
deleteOption	Longint	→	Folder deletion option

Description

The **DELETE FOLDER** command deletes the folder whose name or full path has been passed in *folder*.

By default, for security reasons, if you omit the *deleteOption* parameter, **DELETE FOLDER** only allows empty folders to be deleted. If you want the command to be able to delete non-empty folders, you must use the *deleteOption* parameter. In *deleteOption*, you can pass one of the following constants, found in the "**System Documents**" theme:

Constant	Type	Value	Comment
Delete only if empty	Longint	0	Deletes folder only when it is empty
Delete with contents	Longint	1	Deletes folder along with everything it contains

- When Delete only if empty (0) is passed or if you omit the *deleteOption* parameter:
 - The folder specified in the *folder* parameter is only deleted if it is empty; otherwise, the command does nothing and an error -47 (The file is already open, or the folder is not empty) is generated.
 - If the folder specified does not exist, the error -120 (Tried to access a file by using a pathname that specifies a non existing directory) is generated.
- When Delete with contents (1) is passed:
 - The folder along with all of its contents are deleted.
Warning: Even when this folder and/or its contents are locked or set to read-only, if the current user has suitable access rights, they are still deleted.
 - If this folder, or any of the files it contains, cannot be deleted, deletion is aborted as soon as the first inaccessible element is detected, and an error(*) is returned. In this case, the folder may be only partially deleted. When deletion is aborted, you can use **GET LAST ERROR STACK** command to retrieve the name and path of the offending file.
 - If the folder specified does not exist, the command does nothing and no error is returned.

(*) under Windows: -54 (Attempt to open locked file for writing)
under OS X: -45 (The file is locked or the pathname is not correct)

You can intercept these errors using a method installed by the **ON ERR CALL** command.

FONT LIST (fonts {; listType | *})

Parameter	Type	Description
fonts	Text array	← Array of font names
listType *	Longint, Operator	→ Font type list to return or * to return font names under OS X

Description

The **FONT LIST** command populates the *fonts* text array with the names of scalable fonts available on your system. The *listType* parameter lets you designate the type of font list you want to get. To do so, you can pass one of the following constants in the *listType* parameter, available in the "**Font Type List**" theme:

Constant	Type	Value	Comment
Favorite fonts	Longint	1	<i>fonts</i> contains the list of favorite fonts. - Under Windows: list of active font family names. - Under OS X: list of font family names found in the control panel, entitled "Favorites" in English, "Favoris" in French, "Favoriten" in German, and so on . This collection may be blank if the user has not added any favorite fonts.
Recent fonts	Longint	2	<i>fonts</i> contains the list of recent fonts (the ones used during the 4D session). This list is used in particular by multi-style text areas.
System fonts	Longint	0	<i>fonts</i> contains the list of all the system fonts. Default option when <i>listType</i> is omitted.

Under OS X, when you pass the optional * parameter, the command populates the *fonts* array with the names of the fonts themselves, and not with the names of the font families. The default operation simplifies programmed management of rich text areas, which use font families. If you pass the * parameter, font names, for example, "Arial bold", "Arial italic", "Arial narrow italic," are returned instead of families, such as "Arial", "Arial black" or "Arial narrow".

Under Windows, the * parameter has no effect. The command still returns the font families.

Note: Under OS X, if you use the result of this command with the **ST SET ATTRIBUTES**, you must not pass the * parameter.

About scalable fonts

This command returns only scalable fonts. Using non-scalable fonts (i.e. bitmap fonts) to design interfaces is not recommended since they are based on an outdated technology and suffer from limitations regarding size variations. They are not supported in cutting-edge features of 4D such as 4D Write Pro areas .

Under OS X, this principle has been in effect since OS X 10.4 (*QuickDraw* bitmap fonts are obsolete beginning with this version).

Under Windows, this principle is applied beginning with 4D v15 R4. In order to help developers select only modern fonts for their interfaces, only "trueType" or "openType" scalable fonts are listed. For example, "ASI_Mono", "MS Sans Serif" and "System" fonts are no longer available. In addition, GDI names are also ignored; only DirectWrite font family names are supported. For example, "Arial Black" or "Segoe UI Black" font families are not in the list; only "Arial" and "Segoe" are returned.

Compatibility notes for Windows:

- Bitmap fonts can still be used in your 4D forms (except in 4D Write Pro areas). They are just removed from the list returned by this command. However, to ensure compatibility with future versions of 4D and Windows, we recommend using only DirectWrite font families.
- Since bitmap fonts are filtered from the *fonts* parameter on Windows, the resulting list is different in 4D v15 R4 applications and higher, compared to previous releases. Please make sure to adapt your code if you were using this command to select a non-scalable font.

Example 1

In a form, you want a drop-down list that displays a list of the fonts available on your system. The method of the drop-down list is as follows:

```
Case of
  : (Form event=On_Load)
    ARRAY TEXT (asFont;0)
    FONT LIST (asFont)
  \ ...
End case
```

Example 2

You want to get a list of recent fonts:

```
FONT LIST ($arrFonts;Recent_fonts)
```

GET PRINT OPTION

```
GET PRINT OPTION ( option ; value1 {; value2} )
```

Parameter	Type		Description
option	Longint	→	Option number or PDF option code
value1	Longint, Text	←	Value 1 of the option
value2	Longint, Text	←	Value 2 of the option

Description

The **GET PRINT OPTION** command returns the current value(s) of a print option.

The *option* parameter enables you to specify the option to get. You can either get a standard option (longint), or a PDF option code (string). The command returns, in the *value1* and (optionally) *value2* parameters, the current value(s) of the specified *option*.

To specify a standard printing option, you can use of the following predefined constants, located in the “**Print Options**” theme:

Constant	Type	Value	Comment
Paper option	Longint	1	<p>If you use only <i>value1</i>, it contains the name of the paper. If you use both parameters, <i>value1</i> contains the paper width and <i>value2</i> contains the paper height. The width and height are expressed in screen pixels. Use the PRINT OPTION VALUES command to get the name, height and width of all the paper formats offered by the printer.</p> <p><i>value1</i> only: 1=Portrait, 2=Landscape. If a different orientation option is used, GET PRINT OPTION returns 0 in <i>value1</i>.</p>
Orientation option	Longint	2	<p>64-bit versions: This option can be called within a print job, which means that you can switch from portrait to landscape, or vice versa, during the same print job.</p> <p><i>value1</i> only: scale value in percentage. Be careful, some printers do not allow you to modify the scale. If you pass an invalid value, the property is reset to 100% at the time of printing.</p>
Scale option	Longint	3	
Number of copies option	Longint	4	<p><i>value1</i> only: number of copies to be printed.</p>
Paper source option	Longint	5	<p>(Windows only) <i>value1</i> only: number corresponding to the index, in the array of trays returned by the PRINT OPTION VALUES command, of the paper tray to be used. This option can only be used under Windows.</p>
Color option	Longint	8	<p>(Windows only) <i>value1</i> only: code specifying the mode for handling color: 1=Black and white (monochrome), 2=Color.</p> <p>64-bit versions: This option is not supported in 4D 64-bit versions (obsolete)</p> <p><i>value1</i>: code specifying the type of print destination: 1=Printer, 2=(PC)/PS File (Mac), 3=PDF file, 5=Screen (OS X driver option).</p> <p>If <i>value1</i> is different from 1 or 5, <i>value2</i> contains pathname for resulting document. This path will be used until another path is specified. If a file with the same name already exists at the destination location, it will be replaced. With GET PRINT OPTION, if the current value is not in the predefined list, <i>value1</i> contains -1 and the system variable OK is set to 1. If an error occurs, <i>value1</i> and the system variable OK are set to 0.</p>
Destination option	Longint	9	<p>Note: Under Windows, you can set the printing destination to 3 (PDF File) when the PDF Creator driver has been installed. When the (9;3;path) values are passed, 4D automatically starts a "silent" PDF printing which takes into account any option codes that are passed (note that if you pass an empty string in <i>value2</i> or omit this parameter, a file saving dialog appears at the time of printing.) After printing, the current settings are restored. This simplifies control of printing PDFs for 4D and lets you write multi-platform code. When the (9;3;path) values are not passed, printing is not controlled by 4D and any PDF Creator option codes that were passed are ignored.</p>
Double sided option	Longint	11	<p>(Windows only) <i>value1</i>: 0=Single-sided or standard, 1=Double-sided. If <i>value1</i>=1, <i>value2</i> contains the binding: 0=Left binding (default value), 1=Top binding.</p> <p>Note: This option can only be used under Windows.</p>
Spooler document name option	Longint	12	<p><i>value1</i> only: name of the current print document, which appears in the list of spooler documents. The name defined by this statement will be used for all the print documents of the session for as long as a new name or an empty string is not passed. To use or restore standard operation (using the method name in the case of a method, the table name for a record, etc.), pass an empty string in <i>value1</i>.</p> <p>(Mac only) <i>value1</i> only: 0=print job in PDF mode (default value) 1=print job in PostScript mode.</p> <p>Notes:</p> <ul style="list-style-type: none"> - This option has no effect under Windows. - Under OS X, printing is done as a PDF by default. However, the PDF print driver does not support PICT pictures with encapsulated PostScript information — these pictures are generated, more particularly, by vectorial drawing software. To avoid this problem, this option lets you modify the print mode to use under OS X for the current session. Keep in mind that printing in PostScript
Mac spool file format option	Longint	13	

			mode can lead to undesired side effects.
			64-bit versions: This option is not supported; it is replaced by the Generic PDF driver option of the SET CURRENT PRINTER command.
Hide printing progress option	Longint	14	<i>value1</i> only: 1=hide progress windows, 0=display progress windows (default). This option is particularly useful in the case of PDF printing under OS X. Note: There is already a Printing progress option found in the Database Settings dialog box (Interface page). However, it is applied globally to the application and does not hide all the windows under OS X.
Page range option	Longint	15	<i>value1</i> =first page to print (default value is 1) and (optional) <i>value2</i> =number of the last page to print (default value -1 = end of document). (4D 64-bit versions for Windows only) <i>value1</i> only: 1=select the GDI-based legacy printing layer for the subsequent printing jobs. 0=select the D2D printing layer (default).
Legacy printing layer option	Longint	16	64-bit versions: This selector is only supported on 4D 64-bit single-user applications on Windows; it is ignored on other platforms. It is mainly intended to allow legacy plug-ins to print inside 4D jobs in 4D 64-bit applications.

A PDF option code consists of two parts, *OptionType* and *OptionName*, combined together as "*OptionType:OptionName*". For more information on PDF option codes and possible values, refer to the description of the **SET PRINT OPTION** command.

Note: The **GET PRINT OPTION** command mainly supports PostScript printers. You can use this command with other types of printers, such as PCL or Ink, but in this case, it is possible that some options may not be available.

System variables and sets

The system variable OK is set to 1 if the command has been executed correctly; otherwise, it is set to 0.

LISTBOX SET HEADERS HEIGHT

LISTBOX SET HEADERS HEIGHT ({ * ; } object ; height { ; unit })

Parameter	Type		Description
*	Operator	⇒	If specified, object is an object name (string) If omitted, object is a variable
object	Form object	⇒	Object Name (if * is specified) or Variable (if * is omitted)
height	Longint	⇒	Row height
unit	Longint	⇒	Unit of height value: 0 or omitted = pixels, 1 = lines

Description

The **LISTBOX SET HEADERS HEIGHT** command modifies by programming the height of the header row in the list box designated by the *object* and * parameters.

If you pass the optional * parameter, this indicates that the *object* parameter is an object name (a string). If you do not pass this parameter, this indicates that the *object* is a variable. In this case, you pass a variable reference instead of a string.

You can designate either the list box or any header of the list box.

Pass the height to set in the *height* parameter. By default, if you omit the *unit* parameter, this height is expressed in pixels. To change the unit, you can pass one of the following constants (found in the **List Box** theme), in the *unit* parameter:

Constant	Type	Value	Comment
lk lines	Longint	1	Height designates a number of lines. 4D calculates the height of a line according to the font
lk pixels	Longint	0	Height is a number of pixels (default).

Headers must respect the minimum height set by the system. This height is 24 pixels under Windows and 17 pixels under Mac OS. If you pass a lower value in the *height* parameter, the minimum height is applied.

Note: For more information about calculation row heights, refer to the *Design Reference* manual.

String (expression {; format {; addTime}}) -> Function result

Parameter	Type	Description
expression	Expression	→ Expression for which to return the string form (can be Real, Integer, Long Integer, Date, Time String, Text or Boolean)
format	String, Longint	→ Display format
addTime	Time	→ Time to add on if expression is a date
Function result	String	→ String form of the expression

Description

The **String** command returns the string form of the numeric, Date, Time, string or Boolean expression you pass in *expression*.

If you do not pass the optional *format* parameter, the string is returned with the appropriate default format. If you pass *format*, you can force the result string to be of a specific format.

The optional *addTime* parameter adds a time to a date in a combined format. It can only be used when the *expression* parameter is a date (see below).

Numeric Expressions

If *expression* is a numeric expression (Real, Integer, Long Integer), you can pass an optional string format. Following are some examples:

Example	Result	Comments
String(2^15)	"32768"	Default format
String(2^15;"###,##0 Inhabitants")	"32,768 Inhabitants"	
String(1/3;"##0.00000")	"0.33333"	
String(1/3)	"0.333333333333333"	Default format(*)
String(Arctan(1)*4)	"3.14159265359"	Default format(*)
String(Arctan(1)*4;"##0.00")	"3.14"	
String(-1;"&x")	"0xFFFFFFFF"	
String(-1;"&\$")	"\$FFFFFFFF"	
String(0 ?+ 7;"&x")	"0x0080"	
String(0 ?+ 7;"&\$")	"\$80"	
String(0 ?+ 14;"&x")	"0x4000"	
String(0 ?+ 14;"&\$")	"\$4000"	
String(50.3;"&xml")	"50.3"	Always "." as decimal separator
String(Num(1=1);"True;;False")	"True"	
String(Num(1=2);"True;;False")	"False"	
String(Log(-1))	""	Undefined number
String(1/0)	"INF"	Positive infinite number
String(-1/0)	"-INF"	Negative infinite number

(*) Beginning with 4D v14 R3, the algorithm for converting real values into text is based on 13 significant digits (as opposed to 15 digits in previous versions of 4D).

The format is specified in the same way as it would be for a number field on a form. See the section **Display formats** in the 4D Design Reference manual for more information about formatting numbers. You can also pass the name of a custom style in *format*. The custom style name must be preceded by the "|" character.

Note: The **String** function is not compatible with "Integer 64 bits" type fields in compiled mode.

Date Expressions

If *expression* is a Date expression, the string is returned using the default format specified in the system.

In the *format* parameter, you can pass one of the constants described below (**Date Display Formats** theme).

In this case, you can also pass a time in the *addTime* parameter. This parameter lets you combine a date with a time so that you can generate time stamps in compliance with current standards ([ISO Date GMT](#) and [Date RFC](#)

1123 constants). These formats are particularly useful in the context of XML and Web processing. The *addTime* parameter can only be used when the *expression* parameter is a date.

Constant	Type	Value	Comment
Blank if null date	Longint	100	"" instead of 0
Date RFC 1123	Longint	10	
Internal date abbreviated	Longint	6	Dec 29, 2006
Internal date long	Longint	5	December 29, 2006
Internal date short	Longint	7	12/29/2006
Internal date short special	Longint	4	12/29/06 (but 12/29/1896 or 12/29/2096)
ISO Date	Longint	8	2006-12-29T00:00:00 (deprecated)
ISO Date GMT	Longint	9	2010-09-13T16:11:53Z
System date abbreviated	Longint	2	Sun, Dec 29, 2006
System date long	Longint	3	Sunday, December 29, 2006
System date short	Longint	1	12/29/2006

Note: Formats can vary depending on system settings.

Here are a few examples of simple formats (assuming that the current date is 12/29/2006):

```
$vsResult:=String(Current date) // $vsResult gets "12/29/06"  
$vsResult:=String(Current date;Internal date long) // $vsResult gets "December 29, 2006"  
$vsResult:=String(Current date;ISO Date GMT) // $vsResult gets "2009-03-04T23:00:00" in France
```

Notes for combined date/time formats:

- The ISO Date GMT format corresponds to the ISO8601 standard, containing a date and a time expressed with respect to the time zone (GMT).

```
$mydate:=String(Current date;ISO Date GMT;Current time) // returns, for instance, 2010-09-13T16:11:53Z
```

Note that the "Z" character at the end indicates the GMT format.

If you do not pass the *addTime* parameter, the command returns the date at midnight (local time) expressed in GMT time, which may cause the date to be moved forward or back depending on the local time zone:

```
$mydate:=String(!13/09/2010!;ISO Date GMT) // returns 2010-09-12T22:00:00Z in France
```

- The ISO Date format is similar to the ISO Date GMT, except that it expresses the date and time without respect to the time zone. Note that since this format does not comply with the ISO8601 standard, its use should be reserved for very specific purposes.

```
$mydate:=String(!13/09/2010!;ISO Date) // returns 2010-09-13T00:00:00 regardless of the time zone  
$mydate:=String(Current date;ISO Date;Current time) // returns 2010-09-13T18:11:53
```

- The Date RFC 1123 format formats a date/time combination according to the standard defined by RFC 822 and 1123. You need this format for example to set the expiration date for cookies in an HTTP header.

```
$mydate:=String(Current date;Date RFC 1123;Current time) // returns, for example Fri, 10 Sep 2010 13:07:20 GMT
```

The time expressed takes the time zone into account (GMT zone). If you only pass a date, the command returns the date at midnight (local time) expressed in GMT time which may cause the date to be moved forward or back depending on the local time zone:

```
$mydate:=String(Current date;Date RFC 1123) // returns Thu, 09 Sep 2010 22:00:00 GMT
```

Time Expressions

If *expression* is a Time expression, the string is returned using the default **HH:MM:SS** format. In the *format* parameter, you can pass one of the following constants (thème **Time Display Formats** theme):

Constant	Type	Value	Comment
Blank if null time	Longint	100	"" instead of 0
HH MM	Longint	2	01:02
HH MM AM PM	Longint	5	1:02 AM
HH MM SS	Longint	1	01:02:03
Hour min	Longint	4	1 hour 2 minutes
Hour min sec	Longint	3	1 hour 2 minutes 3 seconds
ISO time	Longint	8	0000-00-00T01:02:03
Min sec	Longint	7	62 minutes 3 seconds
MM SS	Longint	6	62:03
System time long	Longint	11	1:02:03 AM HNEC (Mac only)
System time long abbreviated	Longint	10	1•02•03 AM (Mac only)
System time short	Longint	9	01:02:03

Notes:

- The ISO Time format corresponds to the ISO8601 standard and contains, in theory, a date and a time. Since this format does not support combined dates/times; the date part is filled with 0s. This format expresses the local time.
- The Blank if null time constant must be added to the format; it indicates that in the case of a null value, 4D must return an empty string instead of zeros.

These examples assume that the current time is 5:30 PM and 45 seconds:

```
$vsResult:=String(Current time) ` $vsResult gets "17:30:45"
$vsResult:=String(Current time;Hour Min Sec) ` $vsResult gets "17 hours 30 minutes 45 seconds"
```

String Expressions

If *expression* is of the String or Text type, the command returns the same value as the one passed in the parameter. This can be useful more particularly in generic programming using pointers.

In this case, the *format* parameter, if passed, is ignored.

Boolean Expressions

If *expression* is of the Boolean type, the command returns the string "True" or "False" in the language of the application (for example, "Vrai" or "Faux" in a French version of 4D).

In this case, the *format* parameter, if passed, is ignored.

4D Transformation Tags

4D provides a set of transformation tags which allow you to insert references to 4D variables or expressions, or to perform different types of processing within a source text, referred to as a "template". These tags are interpreted when the source text is executed and generate an output text.

This principle is used in particular by the 4D Web server to build the **Semi-dynamic pages**.

These tags must generally be inserted as HTML type comments (`<!--#Tag Contents-->`) in the source text.

However, other comments such as `<!--Beginning of list-->` are also possible.

Note: An alternative `$`-based syntax is used in certain conditions for tags that return values, in order to make them XML-compliant. For more information, refer to the following paragraph **Alternative syntax for 4DTEXT, 4DHTML, 4DEVAL**.

The following 4D transformation tags are available:

- 4DTEXT, to insert 4D variables and expressions as text,
- 4DHTML, to insert HTML code,
- 4DEVAL, to evaluate any 4D expression
- 4DSCRIPT, to execute a 4D method,
- 4DINCLUDE, to include a page within another one,
- 4DBASE, to modify the default folder used by the 4DINCLUDE tag,
- 4DCODE, to insert blocks of 4D code,
- 4DIF, 4DELSE, 4ELSEIF and 4DENDIF, to insert conditions in the tagged code,
- 4DLOOP and 4DENDLOOP, to make loops in the tagged code.

It is possible to mix several types of tags. For example, the following HTML structure is entirely feasible:

```
<HTML> ... <BODY> <!--#4DSCRIPT/PRE_PROCESS-->           (Method call) <!--#4DIF (myvar=1)-->
>           (If condition) <!--#4DINCLUDE banner1.html--> (Subpage insertion) <!--
#4DENDIF-->           (End if) <!--#4DIF (mtvar=2)--> <!--#4DINCLUDE
banner2.html--> <!--#4DENDIF--> <!--#4DLOOP [TABLE]-->           (Loop on the current
selection) <!--#4DIF ([TABLE]ValNum>10)-->           (If [TABLE]ValNum>10) <!--#4DINCLUDE
subpage.html--> (Subpage insertion) <!--#4DELSE-->           (Else) <B>Value:
<!--#4DTEXT [TABLE]ValNum--></B><BR>           (Field display) <!--
#4DENDIF--> <!--#4DENDLOOP-->           (End for) </BODY> </HTML>
```

Executing templates

Parsing the contents of 'template' pages can be done in two ways:

- Using the **PROCESS 4D TAGS** command; this command accepts a 'template' as input, as well as (optional) parameters and returns a text resulting from the processing.
- Using 4D's integrated HTTP server: **Semi-dynamic pages** sent by means of the **WEB SEND FILE** (.htm, .html, .shtm, .shtml), **WEB SEND BLOB** (text/html type BLOB), or **WEB SEND TEXT** commands, or called using URLs. In this last case, for reasons of optimization, pages that are suffixed with ".htm" and ".html" are NOT parsed. In order to "force" the parsing of HTML pages in this case, you must add the suffix ".shtm" or ".shtml" (for example, `http://www.server.com/dir/page.shtm`). For more information on this point, refer to the **Semi-dynamic pages** section in the **Web Server** chapter.

4DTEXT

Syntax: `<!--#4DTEXT VarName-->` or `<!--#4DTEXT 4DExpression-->`

Alternative syntax: `$4DTEXT(VarName)` or `$4DTEXT(4DExpression)` (see **Alternative syntax for 4DTEXT, 4DHTML, 4DEVAL**)

The tag `<!--#4DTEXT VarName-->` allows you to insert a reference to the 4D variable or expression returning a value. For example, if you write (in an HTML page):

```
<P>Welcome to <!--#4DTEXT vtSiteName-->!</P>
```

The value of the 4D variable *vtSiteName* will be inserted in the HTML page when it is sent. This value is inserted as simple text, special HTML characters such as ">" are automatically escaped.

You can also insert 4D expressions using the *4DTEXT* tag. You can for example directly insert the contents of a field (`<!--#4DTEXT [tableName]fieldName-->`), an array element (`<!--#4DTEXT tabarr{1}-->`) or a method returning a value (`<!--#4DTEXT mymethod-->`). The expression conversion follows the same rules as the variable ones. Moreover, the expression must comply with 4D syntax rules.

Note: Executing a 4D method with *4DTEXT* from a Web request depends on the value of the "Available through 4D tags and URLs (4DACTION...)" attribute set in the Method properties. For more information about this, refer to the [Connection Security](#) section.

Although an expression can contain direct calls to 4D functions, this is not recommended for localization issues. For example, `<!--#4DTEXT Current date-->`, although correctly interpreted with a 4D in English will not be understood by a French version. The same applies to real numbers (the decimal separator can be different according to the language). In both cases, we strongly advise you to assign a variable through programming.

To ensure that expressions will be evaluated correctly regardless of the 4D language or version used, we recommend using the *token* syntax for elements whose name might vary between different versions (commands, tables, fields, constants). For example, to insert the **Current time** command, enter '**Current time:C178**'. For more information about this, refer to [Using tokens in formulas](#).

In case of an evaluation error, the inserted text will appear as "`<!--#4DTEXT myvar--> : ## error # error code`".

Notes:

- You work with process variables.
- For security reasons, it is recommended to use this tag when processing data introduced from outside the application, in order to prevent the insertion of malicious code (see the [Usage notes](#) section below).
- You can display a picture field content. However, it is not possible to display the content of a picture array item.
- It is possible to display the contents of an object field by means of a 4D formula. For example, you can write `<!--#4DTEXT OB Get:C1224([Rect]Desc;¥"color¥")-->`.
- You will usually work with Text variables. However, you can also use BLOB variables. You just need to generate the BLOB in Text without length mode.

4DHTML

Syntax: `<!--#4DHTML VarName-->` or `<!--#4DHTML 4DExpression-->`

Alternative syntax: `$4DHTML(VarName)` or `$4DHTML(4DExpression)` (see [Alternative syntax for 4DTEXT, 4DHTML, 4DEVAL](#))

Just like the *4DTEXT* tag, this tag lets you assess a variable or 4D expression that returns a value, and insert it as an HTML expression. Unlike the *4DTEXT* tag, this tag does not escape HTML special characters.

For example, here are the processing results of the 4D text variable *myvar* with the available tags:

<i>myvar</i> Value	Tags	Result
<code>myvar:=""</code>	<code><!--#4DTEXT myvar--></code>	<code>&lt;B&gt;</code>
<code>myvar:=""</code>	<code><!--#4DHTML myvar--></code>	<code></code>

To ensure that expressions will be evaluated correctly regardless of the 4D language or version used, we recommend using the *token* syntax for elements whose name might vary between different versions (commands, tables, fields, constants). For example, to insert the **Current time** command, enter '**Current time:C178**'. For more information about this, refer to [Using tokens in formulas](#).

In case of an interpretation error, the inserted text will be "`<!--#4DHTML myvar--> : ## error # error code`".

Notes:

- Executing a 4D method with *4DHTML* from a Web request depends on the value of the "Available through 4D tags and URLs (4DACTION...)" attribute set in the Method properties. For more information about this, refer to the [Connection Security](#) section.
- For security reasons, it is recommended to use the *4DTEXT* tag when processing data introduced from outside the application, in order to prevent the insertion of malicious code (see the [Usage notes](#) section below).

4DEVAL

Syntax: `<!--#4DEVAL VarName-->` or `<!--#4DEVAL 4DExpression-->`

Alternative syntax: `$4DEVAL(VarName)` or `$4DEVAL(4DExpression)` (see [Alternative syntax for 4DTEXT, 4DHTML, 4DEVAL](#))

4DHTML, 4DEVAL)

The 4DEVAL tag allows you to assess a variable or a 4D expression. Like the existing 4DHTML tag, 4DEVAL does not escape HTML characters when returning text. However, unlike 4DHTML or 4DTEXT, 4DEVAL allows you to execute any valid 4D statement, including assignments and expressions that do not return any value.

For example, you can execute:

```
$input:="<!--#4DEVAL a:=42-->" //assignment
$input:=$input+"<!--#4DEVAL a+1-->" //calculation
PROCESS 4D TAGS($input;$output)
//$output = "43"
```

To ensure that expressions will be evaluated correctly regardless of the 4D language or version used, we recommend using the *token* syntax for elements whose name might vary between different versions (commands, tables, fields, constants). For example, to insert the **Current time** command, enter '**Current time:C178**'. For more information about this, refer to **Using tokens in formulas**.

In case of an error during interpretation, the text inserted will be in the form: "**<!--#4DEVAL expr-->: ## error # error code**".

Notes:

- Executing a 4D method with *4DEVAL* from a Web request requires that the "Available through 4D tags and URLs (4DACTION...)" option is set in the Method properties. For more information, refer to the **Connection Security** section.
- For security reasons, it is recommended to use the 4DTEXT tag when processing data introduced from outside the application, in order to prevent the insertion of malicious code (see the **Usage notes** section below).

4DSCRIPT/

Syntax: `<!--#4DSCRIPT/MethodName/MyParam-->`

The *4DSCRIPT* tag allows you to execute 4D methods when processing the template.. The presence of the `<!--#4DSCRIPT/MyMethod/MyParam-->` tag as an HTML comment forces the execution of the **MyMethod** method with the *Param* parameter as a string in *\$1*.

Note: If the tag is called in the context of a Web process, when the page is loaded, 4D calls the **On Web Authentication Database Method** (if it exists). If it returns **True**, 4D executes the method.

The method must return text in *\$0*. If the string starts with the code character 1, it is considered as HTML (the same principle is true for the *4DHTML* tag).

Note: The execution of a method with *4DSCRIPT* depends on the value of the "Available through 4D tags and URLs (4DACTION...)" attribute defined in the Method properties. For more information about this, refer to the **Connection Security** section.

For example, let's say that you insert the following comment "Today is `<!--#4DSCRIPT/MYMETH/MYPARAM-->`" into a semi-dynamic Web page. When loading the page, 4D calls the **On Web Authentication Database Method** (if it exists), then calls the **MYMETH** method and passes the string `/MYPARAM` as the parameter *\$1*. The method returns text in *\$0* (for example "12/31/14"); the expression "Today is `<!--#4DSCRIPT/MYMETH/MYPARAM-->`" therefore becomes "Today is 12/31/14".

The MYMETH method is as follows:

```
C_TEXT($0)\\This parameter must always be declared
C_TEXT($1)\\This parameter must always be declared
$0:=String(Current date)
```

Warning: You must always declare the *\$0* and *\$1* parameters in the called method.

Note: A method called by *4DSCRIPT* must not call interface elements (**DIALOG**, **ALERT**...).

As 4D executes methods in their order of appearance, it is absolutely possible to call a method that sets the value of many variables that are referenced further in the document, whichever mode you are using. You can insert as many `<!--#4DSCRIPT...-->` comments as you want in a template.

4DINCLUDE

Syntax: `<!--#4DINCLUDE Path-->`

This tag is mainly designed to allow another HTML page (indicated by the *path* parameter) to be included in an HTML page. By default, only the body of the HTML page that is specified is included, in other words, the contents

found within the <body> and </body> tags (the tags themselves are not included). This lets you avoid conflicts related to meta tags present in the headers. However, if the HTML page specified does not contain <body></body> tags, the entire page is included. It is up to you to verify the consistency of the meta tags. The <!--#4DINCLUDE --> comment is very useful for tests (<!--#4DIF-->) or loops (<!--#4DLOOP-->). It is very convenient to include tags according to a criteria or randomly.

When including, regardless of the file name extension, 4D analyzes the called page and then inserts the contents (modified or not) in the page originating the 4DINCLUDE call.

An included page with the <!--#4DINCLUDE --> comment is loaded in the Web server cache the same way as pages called via a URL or sent with the **WEB SEND FILE** command.

In *path*, put the path leading to the document to include. **Warning:** In the case of a 4DINCLUDE call, the path is relative to the document being analyzed, that is, the "parent" document. Use the slash character (/) as a folder separator and the two dots (..) to go up one level (HTML syntax).

Notes:

- When you use the 4DINCLUDE tag with the **PROCESS 4D TAGS** command, the default folder is the folder containing the database structure file.
- You can modify the default folder used by the 4DINCLUDE tag in the current page, using the <!--#4DBASE --> tag (see below).

The number of <!--#4DINCLUDE path--> within a page is unlimited. However, the <!--#4DINCLUDE path--> calls can be made only at one level. This means that, for example, you cannot insert <!--#4DINCLUDE mydoc3.html--> in the mydoc2.html body page, which is called by <!--#4DINCLUDE mydoc2--> inserted in mydoc1.html.

Furthermore, 4D verifies that inclusions are not recursive.

In case of error, the inserted text is "<!--#4DINCLUDE path--> :The document cannot be opened".

Examples

```
<!--#4DINCLUDE subpage.html--> <!--#4DINCLUDE folder/subpage.html--> <!--#4DINCLUDE
../folder/subpage.html-->
```

4DBASE

Syntax: <!--#4DBASE folderPath-->

The <!--#4DBASE --> tag designates a working directory that is used by the <!--#4DINCLUDE--> tag. When it is called in a Web page, the <!--#4DBASE --> tag modifies all subsequent <!--#4DINCLUDE--> calls on this page, until the next <!--#4DBASE -->, if any. If the <!--#4DBASE --> folder is modified from within an included file, it retrieves its original value from the parent file.

The *folderPath* parameter must contain a pathname relative to the current page and it must end with a slash (/).

The designated folder must be located inside the Web folder.

Pass the **WEBFOLDER** keyword to restore the default path (relative to the page).

Thus the following code (4D v12), which must specify a relative path for each call:

```
<!--#4DINCLUDE subpage.html--> <!--#4DINCLUDE folder/subpage1.html--> <!--#4DINCLUDE folder/subpage2.html--> <!--#4DINCLUDE folder/subpage3.html--> <!--#4DINCLUDE ../folder/subpage.html-->
```

... can be rewritten using the <!--#4DBASE --> tag:

```
<!--#4DINCLUDE subpage.html--> <!--#4DBASE folder/--> <!--#4DINCLUDE subpage1.html--> <!--#4DINCLUDE subpage2.html--> <!--#4DINCLUDE subpage3.html--> <!--#4DBASE ../folder/--> <!--#4DINCLUDE subpage.html--> <!--#4DBASE WEBFOLDER-->
```

Example

Setting a directory for the home page using the <!--#4DBASE --> tag:

```
/* Index.html */ <!--#4DIF LangFR=True--> <!--#4DBASE FR/--> <!--#4DELSE--> <!--#4DBASE US/--> <!--#4DENDIF--> <!--#4DINCLUDE head.html--> <!--#4DINCLUDE body.html--> <!--#4DINCLUDE footer.html-->
```

In the head.html file, the current folder is modified through <!--#4DBASE -->, without this changing its value in Index.html:

```
/* Head.htm */ /* the working directory here is relative to the included file (FR/ or US/) */
```

```
<!--#4DBASE Styles/--> <!--#4DINCLUDE main.css--> <!--#4DINCLUDE product.css--> <!--#4DBASE
Scripts/--> <!--#4DINCLUDE main.js--> <!--#4DINCLUDE product.js-->
```

4DCODE

The 4DCODE tag allows you to insert a multi-line 4D code block in a template.

When a "<!--#4DCODE" sequence is detected that is followed by a space, a CR or a LF character, 4D interprets all the lines of code up to the next "-->" sequence. The code block itself can contain carriage returns, line feeds, or both; it will be interpreted sequentially by 4D.

For example, using the 4DCODE tag, you can write in a template:

```
<!--#4DCODE
//PARAMETERS initialization

$graphType:=1
If(OB Is defined:C1231($graphParameters;"graphType")) //US language only
  $graphType:=OB GET:C1224($graphParameters;"graphType")
  If($graphType=7)
    $nbSeries:=1
    If($nbValues>8)
      DELETE FROM ARRAY:C228 ($yValuesArrPtr{1}->;9;100000)
      $nbValues:=8
    End if
  End if
End if
-->
```

Note: In a 4DCODE tag, the 4D code must always be written using the English-US language. Therefore, 4DCODE ignores the "Use regional system settings" user preferences for the 4D language (see [Language for commands and constants](#)).

Here are the 4DCODE tag features:

- The **TRACE** command is supported and activates the 4D debugger, thus allowing you to debug your template code.
- Any error will display the standard error dialog that lets the user stop code execution or enter debugging mode.
- The text in between <!--#4DCODE and --> is split into lines accepting any line-ending convention (cr, lf, or crlf).
- The text is tokenized within the context of the database that called **PROCESS 4D TAGS**. This is important for recognition of project methods for example.
Note: The "Available through 4D tags and URLs 4DACTION" method property is not taken into account (see also the **Note about security** below).
- Even if the text always uses English-US, it is recommended to use the token syntax (:Cxxx) for command and constant names to protect against potential problems due to commands or constants being renamed from one version of 4D to another.

Note: For more information on the :Cxxx syntax, please refer to the [Using tokens in formulas](#) section.

Note about security: The fact that 4DCODE tags can call any of the 4D language commands or project methods could be seen as a security issue, especially when the database is available through HTTP. However, since it executes server-side code called from your own template files, the tag itself does not represent a security issue. In this context, as for any Web server, security is mainly handled at the level of remote accesses to server files.

4DIF, 4DELSE, 4DELSEIF and 4DENDIF

Syntax: <!--#4DIF expression--> {<!--#4DELSEIF expression2-->...<!--#4DELSEIF expressionN-->} {<!--#4DELSE-->} <!--#4DENDIF-->

Used with the <!--#4DELSEIF--> (optional), <!--#4DELSE--> (optional) and <!--#4DENDIF--> comments, the <!--#4DIF expression--> comment offers the possibility to execute portions of code conditionally.

The *expression* parameter can contain any valid 4D expression returning a Boolean value. It must be indicated within parenthesis and comply with the 4D syntax rules.

To ensure that expressions will be evaluated correctly regardless of the 4D language or version used, we

recommend using the *token* syntax for elements whose name might vary between different versions (commands, tables, fields, constants). For example, to insert the **Current time** command, enter '**Current time:C178**'. For more information about this, refer to **Using tokens in formulas**.

The `<!--#4DIF expression--> ... <!--#4DENDIF-->` blocks can be nested in several levels. Like in 4D, each `<!--#4DIF expression-->` should match a `<!--#4DENDIF-->`.

In case of an interpretation error, the text "`<!--#4DIF expression-->`: A Boolean expression was expected" is inserted instead of the contents located between `<!--#4DIF -->` and `<!--#4DENDIF-->`.

Likewise, if there are not as many `<!--#4DENDIF-->` as `<!--#4DIF -->`, the text "`<!--#4DIF expression-->`: 4DENDIF expected" is inserted instead of the contents located between `<!--#4DIF -->` and `<!--#4DENDIF-->`.

Using the `<!--#4DELSEIF-->` tag, you can test an unlimited number of conditions. Only the code that follows the first condition evaluated as **True** is executed. If no conditions are true, no statement is executed (if there is no final `<!--#4DELSE-->`).

You can use a `<!--#4DELSE-->` tag after the last `<!--#4DELSEIF-->`. If all the conditions are false, the statements following the `<!--#4DELSE-->` are executed.

The two following codes are equivalent.

- Code using 4DELSE only:

```
<!--#4DIF Condition1--> /* Condition1 is true*/ <!--#4DELSE--> <!--#4DIF Condition2--> /*
Condition2 is true*/ <!--#4DELSE--> <!--#4DIF Condition3--> /* Condition3 is true
*/ <!--#4DELSE--> /*None of the conditions are true*/ <!--#4DENDIF--> <!--
#4DENDIF--> <!--#4DENDIF-->
```

- Similar code using the 4DELSEIF tag:

```
<!--#4DIF Condition1--> /* Condition1 is true*/ <!--#4DELSEIF Condition2--> /* Condition2 is
true*/ <!--#4DELSEIF Condition3--> /* Condition3 is true */ <!--#4DELSE--> /* None of the
conditions are true*/ <!--#4DENDIF-->
```

Example 1

This example of code inserted in a static HTML page displays a different label according the `vname#" "` expression result:

```
<BODY> ... <!--#4DIF (vname#" ")--> Names starting with <!--#4DTEXT vname-->. <!--#4DELSE--> No
name has been found. <!--#4DENDIF--> ... </BODY>
```

Example 2

This example inserts different pages depending on which user is connected:

```
<!--#4DIF LoggedIn=False--> <!--#4DINCLUDE Login.htm --> <!--#4DELSEIF User="Admin" -->
<!--#4DINCLUDE AdminPanel.htm --> <!--#4DELSEIF User="Manager" --> <!--#4DINCLUDE
SalesDashboard.htm --> <!--#4DELSE--> <!--#4DINCLUDE ItemList.htm --> <!--#4DENDIF-->
```

4DLOOP and 4DENDLOOP

Syntax: `<!--#4DLOOP condition--> <!--#4DENDLOOP-->`

This comment allows repetition of a portion of code as long as the condition is fulfilled. The portion is delimited by `<!--#4DLOOP-->` and `<!--#4DENDLOOP-->`.

The `<!--#4DLOOP condition--> ... <!--#4DENDLOOP-->` blocks can be nested. Like in 4D, each `<!--#4DLOOP condition-->` should match a `<!--#4DENDLOOP-->`.

There are five kinds of conditions:

- `<!--#4DLOOP [table]-->`

This syntax makes a loop for each record from the *table* current selection in the current process. The code portion located between the two comments is repeated for each current selection record.

Note: When the 4DLOOP tag is used with a table, records are loaded in Read only mode.

The following code:

```
<!--#4DLOOP [People]--> <!--#4DTEXT [People]Name--> <!--#4DTEXT [People]Surname--><BR> <!--
#4DENDLOOP-->
```


... could be expressed in 4D language in the following way:

```
FIRST RECORD ([People])
While (Not (End selection ([People])))
  ...
  NEXT RECORD ([People])
End while
```

- **<!--#4DLOOP array-->**

This syntax makes a loop for each item array. The array current item is increased when each code portion is repeated.

Note: This syntax cannot be used with two dimension arrays. In this case, it is better to combine a method with nested loops.

The following code example:

```
<!--#4DLOOP arr_names--> <!--#4DTEXT arr_names{arr_names}--><BR> <!--#4DENDLOOP-->
```

... could be expressed in 4D language in the following way:

```
For ($Elem;1;Size of array (arr_names))
  arr_names:=$Elem
  ...
End for
```

- **<!--#4DLOOP method-->**

This syntax makes a loop as long as the method returns True. The method takes a Long Integer parameter type. First it is called with the value 0 to allow an initialization stage (if necessary); it is then called with the values 1, then 2, then 3 and so on, as long as it returns True.

For security reasons, within a Web process, the **On Web Authentication database method** can be called once just before the initialization stage (method execution with 0 as parameter). If the authentication is OK, the initialization stage will proceed.

Warning: **C_BOOLEAN(\$0)** and **C_LONGINT(\$1)** MUST be declared within the method for compilation purposes.

The following code example:

```
<!--#4DLOOP my_method--> <!--#4DTEXT var--> <BR> <!--#4DENDLOOP-->
```

... could be expressed in 4D language in the following way:

```
If (AuthenticationWebOK)
  If (my_method(0))
    $counter:=1
    While (my_method($counter))
      ...
      $counter:=$counter+1
    End while
  End if
End if
```

The *my_method* method can be as follow:

```
C_LONGINT ($1)
C_BOOLEAN ($0)
If ($1=0)
  `Initialisation
  $0:=True
Else
  If ($1<50)
    ...
    var:=...
    $0:=True
  Else
```

```

    $0:=False `Stops the loop
End if
End if

```

- **<!--#4DLOOP 4DExpression-->**

With this syntax, the 4DLOOP tag makes a loop as long as the 4D expression returns **True**. The expression can be any valid Boolean expression and must contain a variable part to be evaluated in each loop to avoid infinite loops.

For example, the following code:

```

<!--#4DEVAL $i:=0--> <!--#4DLOOP ($i<4)--> <!--#4DEVAL $i--> <!--#4DEVAL $i:=$i+1--> <!--#4DENDLOOP-->

```

produces the following result:

```

0
1
2
3

```

To ensure that expressions will be evaluated correctly regardless of the 4D language or version used, we recommend using the *token* syntax for elements whose name might vary between different versions (commands, tables, fields, constants). For example, to insert the **Current time** command, enter '**Current time:C178**'. For more information about this, refer to [Using tokens in formulas](#).

- **<!--#4DLOOP pointerArray-->**

In this case, the 4DLOOP tag works like it does with an array: it makes a loop for each element of the array referenced by the pointer. The current array element is increased each time the portion of code is repeated. This syntax is useful when you pass an array pointer as a parameter to the **PROCESS 4D TAGS** command. Example:

```

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 "

```

In case of an interpretation error, the text "**<!--#4DLOOP expression-->: description**" is inserted instead of the contents located between **<!--#4DLOOP -->** and **<!--#4DENDLOOP-->**.

The following messages can be displayed:

- Unexpected expression type (standard error);
- Incorrect table name (error on the table name);
- An array was expected (the variable is not an array or is a two dimension array);
- The method does not exist;
- Syntax error (when the method is executing);
- Access error (you do not have the appropriate access privileges to access the table or the method).
- 4DENDLOOP expected (the **<!--#4DENDLOOP-->** number does not match the **<!--#4DLOOP -->**).

Alternative syntax for 4DTEXT, 4DHTML, 4DEVAL

Several existing 4D transformation tags can be expressed using a \$-based syntax:

\$4dtag (expression) can be used instead of **<!--#4dtag expression-->**

This alternative syntax is available only for tags used to return processed values:

- 4DTEXT
- 4DHTML

- 4DEVAL

(Other tags, such as 4DIF or 4DSCRIPT, must be written with the regular syntax).

For example, you can write:

```
$4DEVAL(UserName)
```

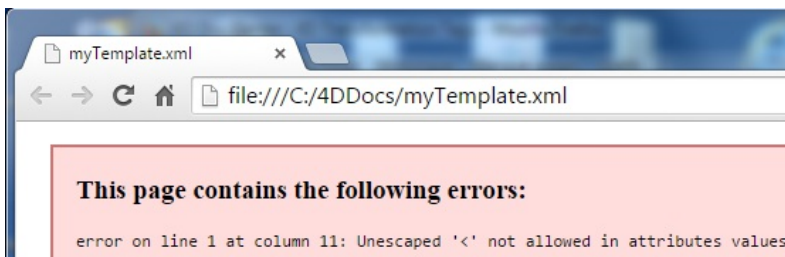
instead of:

```
<!--#4DEVAL(UserName)-->
```

The main advantage of this syntax is that it allows you to **write XML-compliant templates**. Some 4D developers need to create and validate XML-based templates using standard XML parser tools. Since the "<" character is invalid in an XML attribute value, it was not possible to use the "<!-- -->" syntax of 4D tags without breaking the document syntax. On the other hand, escaping the "<" character will prevent 4D from interpreting the tags correctly.

For example, the following code would cause an XML parsing error because of the first "<" character in the attribute value:

```
<line x1="<!--#4DEVAL $x-->" y1="<!--#4DEVAL $graphY1-->" />
```



Using the \$ syntax, the following code is validated by the parser:

```
<line x1="" y1="" />
```

Note that *\$4dtag* and *<!--#4dtag -->* are not strictly equivalent: unlike *<!--#4dtag -->*, *\$4dtag* processing does not interpret 4D tags recursively. \$ tags are always evaluated once and the result is considered as plain text.

Note: For more information on recursive processing, please refer to the [Recursive processing](#) paragraph.

The reason for this difference is to prevent malicious code injection. As explained below, it is strongly recommended to use 4DTEXT tags instead of 4DHTML tags when handling user text to protect against unwanted reinterpretation of tags: with 4DTEXT, special characters such as "<" are escaped, thus any 4D tags using the *<!--#4dtag expression -->* syntax will lose their particular meaning. However, since 4DTEXT does not escape the \$ symbol, we decided to break support for recursion in order to prevent malicious injection using the *\$4dtag (expression)* syntax.

The following examples show the result of processing depending on the syntax and tag used:

```
// example 1
myName:="<!--#4DHTML QUIT 4D-->" //malicious injection
input:="My name is: <!--#4DHTML myName-->"
PROCESS 4D TAGS(input;output)
//4D will quit!
```

```
// example 2
myName:="<!--#4DHTML QUIT 4D-->" //malicious injection
input:="My name is: <!--#4DTEXT myName-->"
PROCESS 4D TAGS(input;output)
//output is "My name is: & lt;!--#4DHTML QUIT 4D-->"
```

```
// example 3
myName:="" //malicious injection
input:="My name is: <!--#4DTEXT myName-->"
```

```
PROCESS 4D TAGS (input;output)
//output is "My name is: ERROR: missing ')"
```

Note that the *\$dtag* syntax supports matching pairs of enclosed quotes or parenthesis. For example, suppose that you need to evaluate the following complex (unrealistic) string:

```
String(1) + "\"(hello)\""
```

You can write:

```
input:="$4DEVAL( String(1)+\"\\\"(hello)\\\"") "
PROCESS 4D TAGS (input;output)
-->output is 1"(hello)"
```

Usage notes

Recursive processing

4D tags are interpreted recursively: 4D always attempts to reinterpret the result of a transformation and, if a new transformation has taken place, an additional interpretation is performed, and so on until the product obtained no longer requires any further transformation. For example, given the following statement:

```
<!--#4DHTML [Mail]Letter_type-->
```

If the [Mail]Letter_type text field itself contains a tag, for example `<!--#4DSCRIPT/m_Gender-->`, this tag will be evaluated recursively after the interpretation of the 4DHTML tag.

This powerful principle meets most needs related to text transformation. Note, however, that in some cases this can also allow malicious code to be inserted. For more information about this point, refer to the following section.

Prevention of malicious code insertion

4D transformation tags accept different types of data as parameters: text, variables, methods, command names, etc. When this data is provided by your own code, there is no risk of malicious code insertion since you control the input. However, your database code often works with data that was, at one time or another, introduced through an external source (user input, import, etc.).

In this case, it is advisable to not use transformation tags such as 4DEVAL or 4DSCRIPT, which evaluate parameters, directly with this sort of data.

In addition, according to the principle of recursion (see previous section), malicious code may itself include transformation tags. In this case, it is imperative to use the 4DTEXT tag.

Imagine, for example, a Web form field named "Name", where users must enter their name. This name is then displayed using a `<!--#4DHTML vName-->` tag in the page. If text of the `<!--#4DEVAL QUIT 4D-->` type is inserted instead of the name, interpreting this tag will cause the application to be exited.

To avoid this risk, you can just use the 4DTEXT tag systematically in this case. Since this tag escapes the special HTML characters, any malicious recursive code that may have been inserted will not be reinterpreted. To refer to the previous example, the "Name" field will contain, in this case, `<!--#4DEVAL QUIT 4D-->` which will not be transformed.

Using the "." as decimal separator

Starting from v15 R4, 4D always uses the period character (.) as decimal separator when evaluating a numerical expression using a 4D tag (4DTEXT, 4DVAR, 4DHTML, 4DHTMLVAR, and 4DEVAL). Regional settings are now ignored.

This feature facilitates code maintenance and compatibility between 4D languages and versions.

For example, whatever the regional settings:

```
value:=10/4
input:"<!--#4DTEXT value-->"
PROCESS 4D TAGS (input;output)
// always outputs 2.5 even if regional settings use the ',' as separator
```

Compatibility note: If your code evaluates numerical expressions using 4D tags with respect to the regional settings, you need to adapt it using the **String** command:

- To get *value* with a period as decimal point: `<!--#4DTEXT value-->`
- To get *value* with a decimal point based on the regional settings: `<!--#4DTEXT String(value)-->`

WEB SET OPTION

WEB SET OPTION (selector ; value)

Parameter	Type		Description
selector	Longint	→	Option code
value	Longint, Text	→	Option value

Description

The **WEB SET OPTION** command modifies the current value of various options concerning the functioning of the 4D Web server.

In the *selector* parameter, pass one of the constants from the **Web Server** theme and pass the new value of the option in *value*:

Constant	Type	Value	Comment
Web character set	Longint	17	<p>Scope: 4D local, 4D Server</p> <p>Kept between two sessions: Yes</p> <p>Description: Character set that the 4D Web Server (with 4D in local mode and 4D Server) should use to communicate with browsers connecting to the database. The default value actually depends on the language of the operating system. This parameter is set in the Database settings.</p> <p>Possible values: The possible values depend on the operating mode of the database relating to the character set.</p> <ul style="list-style-type: none"> • <i>Unicode Mode:</i> When the application is operating in Unicode mode, the values to pass for this parameter are character set identifiers (<i>MIBEnum</i> longint or <i>Name</i> string, identifiers defined by IANA, see the following address: http://www.iana.org/assignments/character-sets). Here is the list of identifiers corresponding to the character sets supported by the 4D Web server: <ul style="list-style-type: none"> 4=ISO-8859-1 12=ISO-8859-9 13=ISO-8859-10 17=Shift-JIS 2024=Windows-31J 2026=Big5 38=euc-kr 106=UTF-8 2250=Windows-1250 2251=Windows-1251 2253=Windows-1253 2255=Windows-1255 2256=Windows-1256 • <i>ASCII compatibility mode:</i> <ul style="list-style-type: none"> Western European 1: Japanese 2: Chinese 3: Korean 4: User-defined 5: Reserved 6: Central European 7: Cyrillic 8: Arabic 9: Greek 10: Hebrew 11: Turkish 12: Baltic
Web debug log	Longint	84	<p>Scope: Local Web server</p> <p>Kept between two sessions: No, but remains valid even if the HTTP server is restarted (a new log file is used in this case)</p> <p>Description: Allows you to get or set the status of the HTTP request log file of the 4D Web server. When enabled, this file, named "HTTPDebugLog_<i>nn.txt</i>", is stored in the "Logs" folder of the application (<i>nn</i> is the file number). It is useful for debugging issues related to the Web server. It records each request and each response in raw mode. Whole requests, including headers, are logged; optionally, body parts can be logged as well.</p> <p>Values: One of the constants prefixed with "wdl" (refer to the descriptions of these constants in this theme).</p> <p>Default value: 0 (not enabled)</p> <p>Scope: Local Web server</p> <p>Kept between two sessions: No</p> <p>Description: Compression level for all compressed HTTP exchanges for the 4D HTTP server (client requests or server replies, Web and Web Service). This</p>

Web HTTP compression level	Longint	50	<p>selector lets you optimize exchanges by either privileging speed of execution (less compression) or the amount of compression (less speed). The choice of a value depends on the size and type of data exchanged. Pass 1 to 9 in the <i>value</i> parameter where 1 is the fastest compression and 9 the highest. You can also pass -1 to get a compromise between speed and rate of compression. By default, the compression level is 1 (faster compression).</p> <p>Possible values: 1 to 9 (1 = faster, 9 = more compressed) or -1 = best compromise.</p> <p>Scope: Local HTTP server</p> <p>Kept between two sessions: No</p>
Web HTTP compression threshold	Longint	51	<p>Description: In the framework of optimized HTTP exchanges, size threshold for requests below which exchanges should not be compressed. This setting is useful in order to avoid losing machine time by compressing small exchanges.</p> <p>Possible values: Any Longint type value. Pass the size expressed in bytes in <i>value</i>. By default, the compression threshold is set to 1024 bytes</p> <p>Scope: Local Web server</p> <p>Kept between two sessions: No</p>
Web HTTP TRACE	Longint	85	<p>Description: Allows you to disable or enable the HTTP TRACE method in the 4D Web server. For security reasons, starting with 4D v15 R2, by default the 4D Web server rejects HTTP TRACE requests with an error 405 (see HTTP TRACE disabled). If necessary, you can enable the HTTP TRACE method for the session by passing this constant with value 1. When this option is enabled, the 4D Web server replies to HTTP TRACE requests with the request line, header, and body.</p> <p>Possible values: 0 (disabled) or 1 (enabled)</p> <p>Default value: 0 (disabled)</p> <p>Scope: 4D local, 4D Server</p> <p>Kept between two sessions: Yes</p>
Web HTTPS port ID	Longint	39	<p>Description: TCP port number used by the Web server of 4D in local mode and of 4D Server for secure connections via SSL (HTTPS protocol). The HTTPS port number is set on the "Web/Configuration" page of the Database settings dialog box.</p> <p>By default, the value is 443 (standard value). You can use the constants of the TCP Port Numbers theme for the <i>value</i> parameter.</p> <p>Possible values: 0 to 65535</p> <p>Scope: Local Web server</p> <p>Kept between two sessions: No, but remains valid even if the HTTP server is restarted</p>
Web inactive process timeout	Longint	78	<p>Description: Modifies the life duration of the inactive processes associated with sessions. At the end of the timeout, the process is killed on the server, the On Web Close Process database method is called then the session context is destroyed.</p> <p>Possible values: Longint (minutes)</p> <p>Default value: 480 minutes (pass 0 to restore the default value)</p> <p>Scope: Local Web server</p> <p>Kept between two sessions: No, but remains valid even if the HTTP server is restarted</p>
Web inactive session timeout	Longint	72	<p>Description: Modifies the life duration of inactive sessions (duration set in cookie). At the end of this period, the session cookie expires and is no longer sent by the HTTP client.</p> <p>Possible values: Longint (minutes)</p> <p>Default value: 480 minutes (pass 0 to restore the default value)</p> <p>Scope: 4D local, 4D Server</p> <p>Kept between two sessions: Yes</p>
Web IP address to listen	Longint	16	<p>Description: IP address on which the 4D Web server will receive HTTP requests with 4D in local mode and 4D Server. By default, no specific address is defined (<i>value</i> = 0). This parameter can be set in the Database settings. The <i>Web IP Address to listen</i> selector is useful for 4D Web Servers compiled and merged with 4D Desktop (in which there is no access to the Design mode). You pass a hexadecimal IP address in the <i>value</i> parameter. In other words, to</p>

designate a IP address such as "a.b.c.d", you should write:

```
C_LONGINT($addr)
$addr:=( $a<<24) | ($b<<16) | ($c<<8) | $d
WEB SET OPTION(Web IP address to listen;$addr)
```

Web keep session

Longint 70

Scope: Local Web server

Kept between two sessions: No, but remains valid even if the HTTP server is restarted

Description: Enables or disables the session management mode (described in the [Web Sessions Management](#) section)

Possible values: 1 (enable mode) or 0 (disable mode)

Default value: 1 for databases created in v13, 0 for converted databases.

Note that this mode also enables the mechanism for reusing temporary contexts in remote mode. For more information about this mechanism, refer to the description of this option in the [Web Server Settings](#) section.

Scope: 4D local, 4D Server

Kept between two sessions: Yes

Description: Starts or stops the recording of Web requests received by the Web server of 4D in local mode or 4D Server. By default, the value is 0 (requests not recorded).

Web log recording

Longint 29

The log of Web requests is stored as a text file named "logweb.txt" that is automatically placed in the Logs folder of the database, next to the structure file. The format of this file is determined by the value that you pass. For more information about Web log file formats, please refer to the [Information about the Web Site](#) section.

This file can also be activated on the "Web/Log" page of the Database settings.

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.

Warning: Formats 3 and 4 are custom formats whose contents must be set beforehand in the Database settings. If you use one of these formats without any of its fields having been selected on this page, the log file will not be generated.

Scope: 4D local, 4D Server

Kept between two sessions: Yes

Description: Strictly upper limit of concurrent Web processes of any type supported by the 4D Web Server with 4D in local mode and 4D Server. When this number (minus one) is reached, 4D will not create any other processes and returns the HTTP status 503 - Service Unavailable to all new requests. This parameter can prevent the 4D Web Server from saturation, which can occur when an exceedingly large number of concurrent requests are sent, or when too many context creations are requested. This parameter can also be set in the Database settings.

Web max concurrent processes

Longint 18

In theory, the maximum number of Web processes is the result of the following formula: Available memory/Web process stack size. Another solution is to view the information on Web processes displayed in the Runtime Explorer: the current number of Web processes and the maximum number reached since the Web server boot are indicated.

Possible values: Any value between 10 and 32 000. The default value is 100.

Scope: Local Web server

Kept between two sessions: No, but remains valid even if the HTTP server is restarted

Description: Limits the number of simultaneous sessions. When you reach the limit set, the oldest session is closed (and [On Web Close Process database method](#) is called) if the Web server needs to create a new one.

Web max sessions

Longint 71

Possible values: Longint. The number of simultaneous sessions cannot exceed the total number of Web processes ([Web Max Concurrent Processes](#) option, 100 by default)

Default value: 100 (pass 0 to restore the default value)

Scope: 4D local, 4D Server

Kept between two sessions: Yes

Web maximum requests size	Longint	27	<p>Description: Maximum size (in bytes) of incoming HTTP requests (POST) that the Web server is authorized to process. By default, the value is 2 000 000, i.e. a little less than 2 MB. Passing the maximum value (2 147 483 648) means that, in practice, no limit is set.</p> <p>This limit is used to avoid Web server saturation due to incoming requests that are too large. When a request reaches this limit, the 4D Web server refuses it.</p> <p>Possible values: 500 000 to 2 147 483 648.</p> <p>Scope: 4D in local mode and 4D Server.</p> <p>Kept between two sessions: No</p>
Web port ID	Longint	15	<p>Description: Sets or gets the number of the TCP port used by the 4D Web server with 4D in local mode and 4D Server. By default, the value is 80. The TCP port number is set on the "Web/Configuration" page of the Database Settings dialog box. You can use one of the constants in the TCP Port Numbers theme for the <i>value</i> parameter. This selector is useful within the framework of 4D Web servers that are compiled and merged using 4D Desktop (no access to the Design environment).</p> <p>Possible values: For more information about the TCP port number, refer to the Web Server Settings section.</p> <p>Default value: 80</p> <p>Scope: local Web server</p> <p>Kept between two sessions: No, but remains valid even if the HTTP server is restarted.</p>
Web session cookie domain	Longint	81	<p>Description: Sets or gets the value of the "domain" field of the session cookie. This selector (as well as selector 82) is useful for controlling the scope of the session cookies: If you set, for example, the value <code>"/*.4d.fr"</code> for this selector, the client will only send a cookie when the request is addressed to the domain <code>".4d.fr"</code>, which excludes servers hosting external static data.</p> <p>Possible values: Text</p> <p>Scope: Local Web server</p> <p>Kept between two sessions: No, but remains valid even if the HTTP server is restarted.</p>
Web session cookie name	Longint	73	<p>Description: Sets the name of the cookie used for saving the session ID.</p> <p>Possible values: Text</p> <p>Default value: "4DSID" (pass an empty string to restore the default value)</p> <p>Scope: local Web server</p> <p>Kept between two sessions: No, but remains valid even if the HTTP server is restarted.</p>
Web session cookie path	Longint	82	<p>Description: Sets or gets the value of the "path" field of the session cookie. This selector (as well as selector 81) is useful for controlling the scope of the session cookies: If you set, for example, the value <code>"/4DACTION"</code> for this selector, the client will only send a cookie for dynamic requests beginning with 4DACTION, and not for pictures, static pages, etc.</p> <p>Possible values: Text</p> <p>Scope: Local Web server</p> <p>Kept between two sessions: No</p>
Web session enable IP address validation	Longint	83	<p>Description: Enables or disables IP address validation for session cookies. For security reasons, by default the 4D Web server checks the IP address of each request containing a session cookie and rejects it if this address does not match the IP address used to create the cookie. In some specific applications, you may want to disable this validation and accept session cookies, even when their IP addresses do not match. For example when mobile devices switch between Wifi and 3G/4G networks, their IP address will change. In this case, you must pass 0 in this option to allow clients to be able to continue using their Web sessions even when the IP addresses change. Note that this setting lowers the security level of your application.</p> <p>When it is modified, this setting is effective immediately (you do not need to restart the HTTP server).</p> <p>Possible values: 0 (disabled) or 1 (enabled)</p> <p>Default value: 1 (IP addresses are checked)</p>

When you use the [Web debug log selector](#), you can pass one of the following constants in the *value* parameter:

Constant	Type	Value	Comment
wdl disable	Longint	0	Web HTTP debug log is disabled
wdl enable with all body parts	Longint	7	Web HTTP debug log is enabled with body parts in response and request
wdl enable with request body	Longint	5	Web HTTP debug log is enabled with body part in request only
wdl enable with response body	Longint	3	Web HTTP debug log is enabled with body part in response only
wdl enable without body	Longint	1	Web HTTP debug log is enabled without body parts (body size is provided in this case)

Example

Enabling the HTTP debug log without body parts:

```
WEB SET OPTION (Web_debug_log;wdl_enable_without_body)
```

A log entry looks like this:

```
# REQUEST
# SocketID: 1592
# PeerIP: 127.0.0.1
# PeerPort: 54912
# TimeStamp: 39089388
GET /4DWEBTEST HTTP/1.1
Connection: Close
Host: 127.0.0.1
User-Agent: 4D_HTTP_Client/0.0.0.0

# RESPONSE
# SocketID: 1592
# PeerIP: 127.0.0.1
# PeerPort: 54912
# TimeStamp: 39089389 (elapsed time: 1 ms)
HTTP/1.1 200 OK
Accept-Ranges: bytes
Connection: close
Content-Length: 3555
Content-Type: text/plain; charset=UTF-8
Date: Tue, 20 Jan 2015 10:51:29 GMT
Expires: Tue, 20 Jan 2015 10:51:29 GMT
Pragma: no-cache
Server: 4D/14.6.0

[Body Size: 3555]
```

Name or theme changes

On Web Session Suspend database method

Previous name	New name	Comments
On Web Session Suspend database method	On Web Close Process database method	Note: In the context of a 4D Mobile session (which can generate several processes), the On Web Close Process database method is called for each Web process that is closed, allowing you to save all types of data (variables, selection, etc.) generated by the 4D Mobile session process.

Listbox constants renamed

List box constants, previously prefixed with "**Listbox...**" are now prefixed with "**lk...**". See [List Box](#) and [Listbox Footer Calculation](#) themes.

Constant	Type	Value	Comment
lk add to selection	Longint	1	The row selected is added to the existing selection. If the row specified already belongs to the existing selection, the command does nothing.
lk all	Longint	0	The command affects all sub-levels (default value, used when parameter is omitted).
lk background color	Longint	1	
lk background color array	Longint	1	
lk break row	Longint	2	The command affects the sub-level to which the "cell" designated by the <i>row</i> and <i>column</i> parameters belongs. Note that these parameters represent the row and column numbers in the list box in standard mode and not in its hierarchical representation. If the <i>row</i> and <i>column</i> parameters are omitted, the command does nothing.
lk control array	Longint	3	
lk display footer	Longint	8	<p>Display Footers property Applies to: List box Possible values:</p> <ul style="list-style-type: none"> • lk_no (0): hidden • lk_yes (1): shown
lk display header	Longint	0	<p>Display Headers property Applies to: List box Possible values:</p> <ul style="list-style-type: none"> • lk_no (0): hidden • lk_yes (1): shown
lk display hor scrollbar	Longint	2	0=hidden, 1=shown
lk display ver scrollbar	Longint	4	0=hidden, 1=shown
lk font color	Longint	0	
lk font color array	Longint	0	
lk footer height	Longint	9	Height in pixels
lk header height	Longint	1	Height in pixels
lk hor scrollbar height	Longint	3	Height in pixels
lk hor scrollbar position	Longint	6	Position of the cursor in pixels
lk inherited	Longint	-255	
lk last printed row number	Longint	0	Returns in <i>info</i> the number of the last row printed. Lets you find out the number of the next row to be printed. The number returned may be greater than the number of rows actually printed if the list box contains invisible rows or if the OBJECT SET SCROLL POSITION command has been called. For example, if rows 1, 18 and 20 have been printed, <i>info</i> is 20.

lk level	Longint	3	The command affects all the break rows corresponding to the <i>level</i> column. This parameter designates a column number in the list box in standard mode and not in its hierarchical representation. If the <i>level</i> parameter is omitted, the command does nothing.
lk lines	Longint	1	Height designates a number of lines. 4D calculates the height of a line according to the font
lk pixels	Longint	0	Height is a number of pixels (default).
lk printed height	Longint	3	Returns in <i>info</i> the height in pixels of the object actually printed (including headers, lines, etc.). Remember that if the number of rows to print is less than the "capacity" of the list box, its height is automatically reduced.
lk printed rows	Longint	1	Returns in <i>info</i> the number of rows actually printed during the last call to the Print object command. This number includes any break rows added in the case of a hierarchical list box. For example, <i>info</i> is 10 if the list box contains 20 rows and the odd-numbered rows were hidden.
lk printing is over	Longint	2	Returns in <i>info</i> a Boolean indicating whether the last (visible) row of the list box has actually been printed. True = row has been printed; Otherwise, False.
lk remove from selection	Longint	2	The row selected is removed from the existing selection. If the row specified does not belong to the existing selection, the command does nothing.
lk replace selection	Longint	0	The row selected becomes the new selection and replaces the existing selection. The command has the same effect as a user click on a row (however, the On Clicked event is not generated). This is the default action (if the <i>action</i> parameter is omitted).
lk row height array	Longint	4	(4D View Pro license required)
lk row is disabled	Longint	2	The corresponding row is disabled. The text and controls such as check boxes are dimmed or grayed out. Enterable text input areas are no longer enterable. Default value: Enabled
lk row is hidden	Longint	1	The corresponding row is hidden. Hiding rows only affects the display of the list box. The hidden rows are still present in the arrays and can be managed by programming. The language commands, more particularly LISTBOX Get number of rows or LISTBOX GET CELL POSITION , do not take the displayed/hidden status of rows into account. For example, in a list box with 10 rows where the first 9 rows are hidden, LISTBOX Get number of rows returns 10. From the user's point of view, the presence of hidden rows in a list box is not visibly discernible. Only visible rows can be selected (for example using the Select All command). Default value: Visible
lk row is not selectable	Longint	4	The corresponding row is not selectable (highlighting is not possible). Enterable text input areas are no longer enterable unless the "Single-Click Edit" option is enabled. Controls such as check boxes and lists are still functional however. This setting is ignored if the list box selection mode is "None". Default value: Selectable
lk selection	Longint	1	The command affects selected sub-levels.
lk style array	Longint	2	
lk ver scrollbar position	Longint	7	Position of the cursor in pixels
lk ver scrollbar width	Longint	5	Width in pixels

Obsolete functions

Obsolete commands

To find all the obsolete commands in your databases, you can just search for "_o_" using **Find in Design** in the Edit menu or the **Find in design** area of the tool bar. We strongly recommend that you replace any obsolete commands with new commands or functions. However, keep in mind that commands which are not indicated as "disabled" will nevertheless continue to work *temporarily*.

To get a list of all the obsolete commands: [Language: deprecated and/or removed commands](#)

Obsolete commands in 4D v16

Previous name	New name	Comments
C_GRAPH	_o_C_GRAPH	Starting with 4D v15 R5 this command was renamed and is no longer provided in the list of 4D commands. Graph area variables are obsolete and no longer supported since 4D v14. You have to use picture variables (see GRAPH) The _o_INTEGRATE LOG FILE command is now obsolete. We recommend using INTEGRATE MIRROR LOG FILE . Unlike the _o_INTEGRATE LOG FILE command, the new INTEGRATE MIRROR LOG FILE command does not replace the current log file with the integrated one: the current log file of the database continues to be used. Accordingly, any changes made during integration are saved in the current log file.
INTEGRATE LOG FILE	_o_INTEGRATE LOG FILE	
Open external window	_o_Open external window	The _o_Open external window command does not work in 64-bit versions of 4D and 4D Server. This command will no longer be supported in future versions of the program.

Obsolete 4D Pack commands

4D Pack commands	Replaced with	Obsolete since	Current status
_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

_o_C_GRAPH

```
_o_C_GRAPH ( {method ;} variable {; variable2 ; ... ; variableN} )
```

Parameter	Type		Description
method	String	→	Name of method
variable	Variable	→	Name of variable(s) to declare

Compatibility note

Variables of the Graph area type are obsolete and no longer supported since 4D v14. You need to use picture variables (see [GRAPH](#)).

BLOB to print settings

BLOB to print settings (printSettings {; params}) -> Function result

Parameter	Type	Description
printSettings	BLOB	→ BLOB containing print settings
params	Longint	→ 0=Restore saved values for number of copies and page range, 1=Reset to default values
Function result	Longint	↻ Status code: 1=Operation successful, 0=No current printer, -1=Incorrect parameters, 2=Printer changed

Description

The **BLOB to print settings** command replaces the current 4D print settings with the parameters stored in the *printSettings* BLOB. This BLOB must have been generated by the **Print settings to BLOB** command or the **_o_AP Print settings to BLOB** 4D Pack command (see below).

The *params* parameter allows you to define how to handle the basic "number of copies" and "page range" settings:

- If you pass 0 or omit this parameter, the values stored in the BLOB are restored,
- If you pass 1, the values are reset to default: the number of copies is set to 1 and the page range is set to "all pages".

The print settings are applied to the current printer and for the whole session, as long as no command such as **PAGE SETUP**, **SET PRINT OPTION** or **PRINT SELECTION** without the > parameter modifies them. The parameters set are used more particularly by the **PRINT SELECTION**, **PRINT LABEL**, **PRINT RECORD**, **Print form** and **QR REPORT** commands, as well as by the menu commands of 4D, including those of the Design environment.

The **PRINT SELECTION**, **PRINT LABEL**, and **PRINT RECORD** commands must be called with the > parameter (if applicable) in order for the settings defined by **BLOB to print settings** to be kept.

The command returns one of the following status codes:

- -1: the BLOB is incorrect,
- 0: no current printer is selected (in this case the command does nothing),
- 1: the BLOB has been correctly loaded,
- 2: the BLOB has been correctly loaded but the current printer name has changed(*).

Note: Code (2) is always returned if the BLOB was created by the **_o_AP Print settings to BLOB** 4D Pack command, even if the printer name did not actually change, since this information was not included in the 4D Pack BLOBs.

(*) Settings depend on the currently selected printer at the moment the BLOB was saved. Applying these settings to a another printer is supported if both printers are of the same model. If the printers are different, only common parameters will be restored.

Windows / OS X

The *printSettings* BLOB can be saved and read on both platforms. However, even if some print settings are common, some others are platform-specific and depend on the drivers and system versions. If the same *printSettings* BLOB is shared between both platforms, you may lose parts of the information.

When used in a heterogeneous environment, in order to restore the maximum settings available for each platform (and not only the common part), it is recommended that you work with two *printSettings* BLOBs, one for each platform.

Compatibility with 4D Pack commands

The print settings BLOBs generated by the legacy **_o_AP Print settings to BLOB** command from 4D Pack can be loaded and used by the **BLOB to print settings** command. Note however that if they are saved using **Print settings to BLOB**, they are converted and can no longer be opened using **_o_AP BLOB to print settings**. The **BLOB to print settings** command stores much more printing information than **_o_AP Print settings to BLOB**.

Example

You want to apply print settings previously saved to disk to the current 4D printing context:

```
C_BLOB(curSettings)
DOCUMENT TO BLOB(Get 4D folder(Active 4D Folder)+"current4Dsettings.blob";curSettings)
//current4Dsettings has been created by Print settings to BLOB
$err:=BLOB to print settings(curSettings;0)
Case of
  :($err=1)
  //everything is OK
  :($err=2)
    CONFIRM("Printer has changed!\n\nCheck the print settings?")
    If(OK=1)
      PRINT SETTINGS
    End if
  :($err=0)
    ALERT("There is no current printer.")
  :($err=-1)
    ALERT("Invalid settings file.")
End case
```

Disabled functions

- **_o_AP Save BMP 8 bit**: command removed
- Certain functions are disabled in 64-bit versions of 4D, for more information refer to the:
 - Disabled 4D features and Unsupported 4D features paragraphs found in the [Using 4D Developer Edition 64-bit versions](#) section
 - Print options, [Printing](#) section

Using 4D Developer Edition 64-bit versions

With 4D v16, 4D provides **4D Developer Edition** and **4D Volume Desktop in 64-bit versions for OS X**.

Notes:

- Under Windows, 64-bit versions of 4D Developer and 4D Volume Desktop are provided **in preview versions**.
- Keep in mind that 4D also allows [Using 4D Server 64-bit version \(Windows\)](#) and [Using 4D Server 64-bit version \(OS X\)](#).

These versions allow your 4D stand-alone applications, as well as your 4D remote applications, to take full advantage of the power of 64-bit operating systems. The main advantage of the 64-bit architecture is that more RAM can be addressed. Moreover, implementing this architecture provided us with an opportunity to support powerful features, such as the ability to handle **Preemptive 4D processes**, to modernize printing as well as the Quick Report editor, or yet again, to allow your applications to take advantage of **Native object animations (4D 64-bit versions on OS X)**.

Although widely rewritten, 4D 64-bit applications are highly compatible with 4D databases developed in 32-bit versions. However, since they use the most recent technologies, we needed to update some features, as well as to stop supporting others. All these evolutions are detailed in the [Specific features of 64-bit versions](#) section below.

System requirements

4D Developer 64-bit versions require the following minimum configuration:

	Windows	OS X
OS	Windows 7 or higher (64-bit versions)	OS X version 10.10 (Yosemite) or higher
RAM	8 GB	8 GB

Please refer to the certification matrices [available on 4D's Web site](#) to find out which operating systems are compatible with your version of 4D.

Architecture

4D applications intended for 64-bit architectures are specific versions dedicated to this environment; (they will not run on a 32-bit OS).

In interpreted mode, the same 4D databases can be executed with either a 64-bit or a 32-bit 4D application (server or local). Development is identical regardless of which application is used (except for the limitations listed in this documentation).

In compiled mode, databases must have been compiled for the appropriate processor: 64-bit in order to be executed with a 64-bit 4D application, and 32-bit to be executed with a 32-bit 4D application. A database which has been compiled in 32-bit only and which does not contain interpreted code cannot be executed with a 64-bit 4D application, and vice versa. You can compile your database for just one specific architecture, or for both. For more information about compilation, please refer to the next section.

The following table summarizes compatibility between the various 4D execution environments and the database code:

	Available code	4D Dev 32-bit	4D Dev 64-bit
4D Server 32-bit	Interpreted	OK	OK(*)
	32-bit compiled only	OK	-
	32-bit and 64-bit compiled	OK	OK(*)
4D Server 64-bit	Interpreted	OK	OK(*)
	64-bit compiled only	-	OK(*)
	64-bit and 32-bit compiled	OK	OK(*)
Local database	Interpreted	OK	OK
	32-bit compiled only	OK	-
	64-bit compiled only	-	OK
	32-bit and 64-bit compiled	OK	OK

(*) With 32-bit versions of 4D Server (both platforms) and 64-bit versions of 4D Server for Windows, you need to make sure that the **ServerNet** network layer is activated on the server side, since the legacy network layer is not available in 64-bit versions of 4D. For more information, please refer to the [New ServerNet Network Layer \(compatibility\)](#) section.

Components and plug-ins

The following plug-ins and components can be loaded and executed interchangeably by 4D Server, 4D Developer Edition or 4D Volume Desktop in 32- or 64-bit versions:

- 4D for OCI
- 4D Internet Commands(*)
- 4D ODBC Pro
- 4D Pack
- 4D Progress
- 4D SVG
- 4D Widgets
- 4D Write Pro Interface

(*) Not available in 4D Developer Edition 64-bit version for Windows (preversion)

4D View and 4D Write

4D View and 4D Write are 32-bit plug-ins and normally can only be used with 32-bit versions of 4D. However, 4D provides the following arrangements:

- Non-executable 64-bit versions of these plug-ins are available as placeholders so that developers can load them and work in 64-bit versions (OS X or Windows) and then compile/deploy for 32-bit versions.
- 4D Server 64-bit for Windows can execute these plug-ins in faceless mode (no interface).

Specific features of 64-bit versions

This section covers particularities and evolutions concerning the implementation of the current 64-bit versions of 4D Developer Edition on Windows and on OS X.

Updated 4D features

Many 4D features and dialogs have been adapted or even rewritten to support the 64-bit architecture. Most of the changes are transparent and will work just like in 32-bit releases. However, some editors have been modified and now differ from their 32-bit versions, and some elementary features such as printing have been updated.

Feature	Impacted 4D version	Comment
Quick Report editor	OS X & Win	Completely rewritten. See Quick reports (64-bit) section.
Label editor	OS X & Win	Completely rewritten. See Label editor (64-bit) section.
Charts	OS X & Win	The GRAPH command accepts an Object type parameter that allows you to define graph settings.
Printing	OS X & Win	Update of "Printing" dialog boxes (use of standard system dialog boxes). The "Print settings" dialog box is no longer displayed automatically (see the PRINT SETTINGS command). Modification of the SET CURRENT PRINTER and SET PRINT OPTION commands.
Import/Export dialog boxes	OS X & Win	Work as in 32-bit version, except XSL support for XML exports (XSLT is no longer supported, see below) and via an ODBC source (disabled, see below)

Disabled 4D features

Some features are disabled in the 4D Developer Edition 64-bit version:

Feature/Technology	Impacted 4D version	Comment
Import/Export via ODBC Data source	OS X & Win	Disabled
Quick Report cross-table reports	OS X & Win	Disabled
Quick Report editor: borders	OS X & Win	Disabled
Label Editor standard codes	OS X & Win	Disabled
Using integrated Web Kit in Web areas	OS X & Win	Disabled. If option used, automatic switching to system Web engine. Under OS X, access to 4D \$4d methods is maintained
4D Internet Commands	Win	Currently not available

Unsupported 4D features

The following deprecated features or technologies are not supported in the 4D Developer Edition 64-bit version:

Feature/Technology	Impacted 4D version	Comment
XSLT with Xalan	OS X & Win	_o_XSLT APPLY TRANSFORMATION , _o_XSLT SET PARAMETER , and _o_XSLT GET ERROR do nothing. Use the PROCESS 4D TAGS command or the PHP <i>libxslt</i> module instead.
PICT format	OS X & Win	'Unsupported image format' picture + file extension is displayed instead. PICT format is globally deprecated in 4D, see also Pictures in PICT format .
QuickTime	OS X & Win	QuickTime for pictures is not supported. The QuickTime support database parameter is ignored.
cicn icons	OS X & Win	GET ICON RESOURCE command is not supported; it returns an error.
Database .RSR files	OS X & Win	Database .RSR files are not opened automatically. You need to use Open resource file .
Writable resource files	OS X & Win	_o_Create resource file is not supported; you can only open resource files in read-only.
_o_Font number	OS X & Win	This command is not supported; it returns an error.
_o_Open external window	OS X & Win	This command is not supported; it returns an error.
Legacy network layer	OS X & Win	Only <i>ServerNet</i> is supported.
ASCII compatibility mode	OS X & Win	Only Unicode mode is supported.
4D Write and 4D View plug ins	OS X & Win	Legacy plug-ins are not compatible with 64-bit versions of 4D; use 4D Write Pro Reference and 4D View Pro .
AP Print settings to BLOB / AP BLOB to print settings (4D Pack)	OS X & Win	Replaced by the Print settings to BLOB / BLOB to print settings commands.
OLE Tools	Win	Not supported.

Changing from 32-bit versions to 64-bit versions

Upgrading an existing 4D application on OS X from a 32-bit version of 4D to a 64-bit version requires some preparation work.

If your application runs on 4D Server 64-bit Windows or OS X, most of the work is already done. 64-bit versions of desktop applications may require a few additional steps. This section provides a step-by-step checklist to help you verify all the necessary points both before and after the upgrade.

Several features have been updated, disabled or even declared as deprecated for the 64-bit migration of our products. All details are listed in the [Specific features of 64-bit versions](#) section.

Note: Like with any upgrade process, it is good practice to use the MSC and launch a verification process before each major step to make sure both the data and structure are OK.

Check your plug-ins

The first requirement consists in upgrading your plug-ins (if any) to their 64-bit version:

- **4D plug-ins:**

All plug-ins already exist in 64-bit versions, except for 4D Write and 4D View.

- If your application uses 4D Write, you need to consider migrating your code to 4D Write Pro. Good practice is to keep your existing 32-bit code and start a new 64-bit based module with 4D Write Pro alongside it.
- If your application uses 4D View, you will have to use 4D View Pro features or other alternatives.

- **Third-party plug-ins:**

Contact your providers to get 64-bit versions.

Prepare for the upgrade to the 32-bit version

1. Upgrade your application to the latest 32-bit release, for example 4D v16 32-bit or higher.
2. Make sure Unicode mode is activated.
3. Convert any PICT/cicn/QuickTime pictures.
To detect deprecated pictures in your data, you can use the [GET PICTURE FORMATS](#) command.
You also need to replace all unsupported pictures in the structure of your database. A verification with the MSC will detect deprecated pictures in resources files for picture and 3D buttons as well as for static pictures.
4. Replace XSLT-based features ([_o_XSLT APPLY TRANSFORMATION](#), [_o_XSLT SET PARAMETER](#) or [_o_XSLT GET ERROR](#) commands), with the [PROCESS 4D TAGS](#) command for example.
5. Replace [_o_Font number](#) calls with font name calls.
6. Remove any code that creates or modifies resource files.

At this point, you are ready to open your database with a 64-bit version of 4D.

Open and check the database in a 64-bit version

1. Open your application with a 4D Developer Edition 64-bit version.
2. If you use the integrated WebKit for your Web areas, check them since they are automatically switched to the System engine (access to 4D methods through \$4d is still valid).
3. If your code uses the [Mac spool file format option](#) of the [SET PRINT OPTION](#) command, you need to replace it with a call to [SET CURRENT PRINTER](#) with the [Generic PDF driver](#) constant.
4. Check Label editor calls and usages (refer to [Label editor \(64-bit\)](#)).
5. Check Quick Report calls and usages (refer to [Quick reports \(64-bit\)](#)).

Your application is now fully 64-bit compatible and you can benefit from all the new 64-bit features in 4D.

Benefit from 64-bit features

In particular:

- The **64-bit architecture** pushes back database cache limits. Improve your database's performances simply by using a larger cache.

Adopt **powerful 64-bit features** such as preemptive processes, animated form objects, or new printing features.

Build your applications with 4D Runtime Volume License 64-bit..

Use final 64-bit versions of 4D Server on Win and Mac OS - refer to the [Using 4D Server 64-bit version \(OS X\)](#) and [Using 4D Server 64-bit version \(Windows\)](#) sections

- **New Quick report editor**, compatible with reports created using previous versions. See [Quick reports \(64-bit\)](#).
- **New Label editor**, compatible with label files created using previous versions. See [Label editor \(64-bit\)](#).
- Create **graphs** using an Object type parameter with the **GRAPH** command.

SET PRINT OPTION

```
SET PRINT OPTION ( option ; value1 {; value2} )
```

Parameter	Type		Description
option	Longint	→	Option number or PDF option code
value1	Longint, Text	→	Value 1 of the option
value2	Longint, Text	→	Value 2 of the option

Description

The **SET PRINT OPTION** command is used to modify, by programming, the value of a print option. Each option defined using this command is applied to the entire database and for the duration of the session as long as no other command that modifies print parameters (**PRINT SETTINGS**, **PRINT SELECTION** without the > parameter, etc.) is called. If a print job has been opened, the option is set for the job and cannot be modified as long as the job has not terminated.

The *option* parameter allows you to indicate the option to be modified. You can pass either one of the predefined constants of the “**Print Options**” theme, or a PDF option code (usable with the PDFCreator driver under Windows only).

Pass the new value(s) of the specified *option* in the *value1* and (optionally) *value2* parameters. The number and nature of the values to be passed depend on the type of option specified.

Using an option number (constant)

The following table lists the options and their possible values:

Constant	Type	Value	Comment
Paper option	Longint	1	<p>If you use only <i>value1</i>, it contains the name of the paper. If you use both parameters, <i>value1</i> contains the paper width and <i>value2</i> contains the paper height. The width and height are expressed in screen pixels. Use the PRINT OPTION VALUES command to get the name, height and width of all the paper formats offered by the printer.</p> <p><i>value1</i> only: 1=Portrait, 2=Landscape. If a different orientation option is used, GET PRINT OPTION returns 0 in <i>value1</i>.</p>
Orientation option	Longint	2	<p>64-bit versions: This option can be called within a print job, which means that you can switch from portrait to landscape, or vice versa, during the same print job.</p> <p><i>value1</i> only: scale value in percentage. Be careful, some printers do not allow you to modify the scale. If you pass an invalid value, the property is reset to 100% at the time of printing.</p>
Scale option	Longint	3	
Number of copies option	Longint	4	<i>value1</i> only: number of copies to be printed.
Paper source option	Longint	5	(Windows only) <i>value1</i> only: number corresponding to the index, in the array of trays returned by the PRINT OPTION VALUES command, of the paper tray to be used. This option can only be used under Windows.
Color option	Longint	8	<p>(Windows only) <i>value1</i> only: code specifying the mode for handling color: 1=Black and white (monochrome), 2=Color.</p> <p>64-bit versions: This option is not supported in 4D 64-bit versions (obsolete)</p> <p><i>value1</i>: code specifying the type of print destination: 1=Printer, 2=(PC)/PS File (Mac), 3=PDF file, 5=Screen (OS X driver option).</p> <p>If <i>value1</i> is different from 1 or 5, <i>value2</i> contains pathname for resulting document. This path will be used until another path is specified. If a file with the same name already exists at the destination location, it will be replaced. With GET PRINT OPTION, if the current value is not in the predefined list, <i>value1</i> contains -1 and the system variable OK is set to 1. If an error occurs, <i>value1</i> and the system variable OK are set to 0.</p>
Destination option	Longint	9	<p>Note: Under Windows, you can set the printing destination to 3 (PDF File) when the PDF Creator driver has been installed. When the (9;3;path) values are passed, 4D automatically starts a "silent" PDF printing which takes into account any option codes that are passed (note that if you pass an empty string in <i>value2</i> or omit this parameter, a file saving dialog appears at the time of printing.) After printing, the current settings are restored. This simplifies control of printing PDFs for 4D and lets you write multi-platform code. When the (9;3;path) values are not passed, printing is not controlled by 4D and any PDF Creator option codes that were passed are ignored.</p>
Double sided option	Longint	11	<p>(Windows only) <i>value1</i>: 0=Single-sided or standard, 1=Double-sided. If <i>value1</i>=1, <i>value2</i> contains the binding: 0=Left binding (default value), 1=Top binding.</p> <p>Note: This option can only be used under Windows.</p>
Spooler document name option	Longint	12	<p><i>value1</i> only: name of the current print document, which appears in the list of spooler documents. The name defined by this statement will be used for all the print documents of the session for as long as a new name or an empty string is not passed. To use or restore standard operation (using the method name in the case of a method, the table name for a record, etc.), pass an empty string in <i>value1</i>.</p> <p>(Mac only) <i>value1</i> only: 0=print job in PDF mode (default value) 1=print job in PostScript mode.</p> <p>Notes:</p> <ul style="list-style-type: none"> - This option has no effect under Windows. - Under OS X, printing is done as a PDF by default. However, the PDF print driver does not support PICT pictures with encapsulated PostScript information — these pictures are generated, more particularly, by vectorial drawing software. To avoid this problem, this option lets you modify the print mode to use under OS X for the current session. Keep in mind that printing in PostScript
Mac spool file format option	Longint	13	

			mode can lead to undesired side effects.
			64-bit versions: This option is not supported; it is replaced by the Generic PDF driver option of the SET CURRENT PRINTER command.
Hide printing progress option	Longint	14	<i>value1</i> only: 1=hide progress windows, 0=display progress windows (default). This option is particularly useful in the case of PDF printing under OS X. Note: There is already a Printing progress option found in the Database Settings dialog box (Interface page). However, it is applied globally to the application and does not hide all the windows under OS X.
Page range option	Longint	15	<i>value1</i> =first page to print (default value is 1) and (optional) <i>value2</i> =number of the last page to print (default value -1 = end of document). (4D 64-bit versions for Windows only) <i>value1</i> only: 1=select the GDI-based legacy printing layer for the subsequent printing jobs. 0=select the D2D printing layer (default).
Legacy printing layer option	Longint	16	64-bit versions: This selector is only supported on 4D 64-bit single-user applications on Windows; it is ignored on other platforms. It is mainly intended to allow legacy plug-ins to print inside 4D jobs in 4D 64-bit applications.

Once set using this command, a print option is kept throughout the duration of the session for the entire 4D application. It will be used by the **PRINT SELECTION**, **PRINT RECORD**, **Print form**, **QR REPORT** and **WP PRINT** commands, as well as for all 4D printing, including that in Design mode.

Notes:

- It is indispensable to use the optional > parameter with the **PRINT SELECTION**, **PRINT RECORD** and **PAGE BREAK** commands in order to avoid resetting the print options that were set using the **SET PRINT OPTION** command.
- The **SET PRINT OPTION** command mainly supports PostScript printers. You can use this command with other types of printers, such as PCL or Ink, but in this case, it is possible that some options may not be available.

Using a PDF option code (Windows)

In order to be able to use a PDF option code in the *option* parameter, you must have installed the PDFCreator driver in your 4D environment (for more information, refer to the [Integration of PDFCreator driver under Windows](#) section). Moreover, in order for option codes to be taken into account, you need to have enabled control of PDF printing for 4D using the following statement:

```
SET PRINT OPTION(Destination_option;3;fileName)
```

Otherwise, option codes are ignored.

A PDFoption code is a Text type value consisting of two parts, *OptionType* and *OptionName*, combined together as "*OptionType:OptionName*". Here is the description of this code:

- *OptionType* Indicates whether you are specifying a native PDFCreator option or a 4D PDF administration option. Two values are accepted:
 - **PDFOptions** = native option
 - **PDFInfo** = internal option.
- *OptionName* Specifies the option to be set (depending on the *OptionType* value).
 - If *OptionType* = **PDFOptions**, you can pass one of several PDFCreator native options in *OptionName*. For example, the UseAutosave option affects the automatic backup. In order to be able to modify this option, pass "PDFOptions:UseAutosave" in the *option* parameter and the value to be used in the *value1* parameter. For a complete description of the PDFCreator native options, please refer to the documentation provided with the PDFCreator driver.
 - If *OptionType* = **PDFInfo**, you can pass one of the following specific selectors in *OptionName*:
 - **Reset print:** used to reset the internal waiting status in order, more particularly, to exit from an infinite loop. In this case, *value1* is not used.
 - **Reset standard options:** used to reset all the PDFCreator options to their default values. If printing is in progress, the default settings are applied after its completion. In this case, *value1* is not used.
 - **Start:** used to start or stop the PDFCreator spooler. Pass 0 in *value1* to stop it and 1 to start it.
 - **Reset options:** used to reset all the options modified since the beginning of the session using the **SET PRINT OPTION** command and the **PDFOptions** selector.
 - **Version:** used to read the current version number of the PDFCreator driver. This selector can only

- be used with the **GET PRINT OPTION** command. The number is returned in the *value1* parameter.
- **Last error**: used to read the last error returned by the PDFCreator driver. This selector can only be used with the **GET PRINT OPTION** command. The error number is returned in the *value1* parameter.
 - **Print in progress**: used to find out if 4D is waiting for printing by PDFCreator. This selector can only be used with the **GET PRINT OPTION** command. The *value1* parameter returns 1 if 4D is waiting for PDFCreator and 0 otherwise.
 - **Job count**: used to find out the number of jobs waiting in the printing queue. This selector can only be used with the **GET PRINT OPTION** command. The number of jobs is returned in the *value1* parameter.
 - **Synchronous Mode**: used to set the synchronization mode between printing requests sent by 4D and the PDFCreator driver. Since 4D cannot get information about the current status of a print job that is in the printing queue, this option can be used to better control the execution of the jobs by only sending them when the status of the PDFCreator driver is "free". In this case, 4D is synchronized with the driver. Pass 0 in *value1* for 4D to send print requests immediately (default value) and 1 in order for 4D to be synchronized and to wait for the driver to have finished the job in progress before sending another one.

Note: After each printing, 4D automatically re-establishes the previous settings of the PDFCreator driver in order to avoid any interference with other programs using PDFCreator.

Example 1

The following method enables the PDF driver so as to print all the records of the table at the C:¥Test¥Test_PDF_X location where X is the sequence number of the record:

```

SET CURRENT PRINTER(PDFCreator Printer Name)
// Under Windows, select the virtual printer installed by PDFCreator
If(OK=1) // If PDFCreator is actually installed

    ALL RECORDS([Table_1])
    For($i;1;Records in selection([Table_1]))
        SET PRINT OPTION(Destination_option;3;"C:\\Test\\Test_PDF_"+String($i))
// Destination option 3 launches a PDFCreator print job
        PRINT RECORD([Table_1];*)
        NEXT RECORD([Table_1])
    End for
// Resetting of the PDFCreator driver options
    SET PRINT OPTION("PDFInfo:Reset standard options";0)
End if

```

Example 2

In 64-bit versions, the value of Orientation option can be modified within the same print job (special case). Note that the option must have been set before the **PAGE BREAK** command:

```

ALL RECORDS([People])
PRINT SETTINGS
If(OK=1)
    OPEN PRINTING JOB
    SET PRINT OPTION(Orientation_option;1) //portrait
    Print form([People];"Vertical_Form")

    SET PRINT OPTION(Orientation_option;2) //landscape
    PAGE BREAK //must be called imperatively AFTER the option
    Print form([People];"Horiz_Form")
    CLOSE PRINTING JOB
End if

```

System variables and sets

The system variable OK is set to 1 if the command has been executed correctly; otherwise, it is set to 0. If you pass an invalid option code (option not recognized by PDFCreator for example), OK is set to 0.

Error management

If the value passed for an *option* is invalid or if it is not available on the printer, the command returns an error (that you can intercept using an error-handling method installed by the **ON ERR CALL** command) and the current value of the option remains unchanged.

Converting 4D Write documents to 4D Write Pro

Converting a 4D Write document

4D Write Pro can open and convert legacy 4D Write documents while supporting most of their specific properties:

In the picture above, we have a 4D Write area on the left and a 4D Write Pro area on the right (created using the new library object - see below). The contents of the 4D Write area were recovered simply using the **WP New** command:

```
// we retrieve the contents of the 4D Write area in the 4D Write Pro area  
[WRITEAREAS]AreaNTWP:=WP New ([WRITEAREAS]AreaNT_)
```

But since 4D Write can only be used with 32-bit versions of 4D v16, you must convert your 4D Write documents before changing to the 64-bit version.

Unlike 4D Write, 4D Write Pro is not a plug-in but is fully integrated into 4D itself. Note that 4D Write Pro uses the same license as 4D Write. You need to have this license installed in your application in order to enable the feature.

4D Write Pro objects permit 4D Write documents to be imported in two ways:

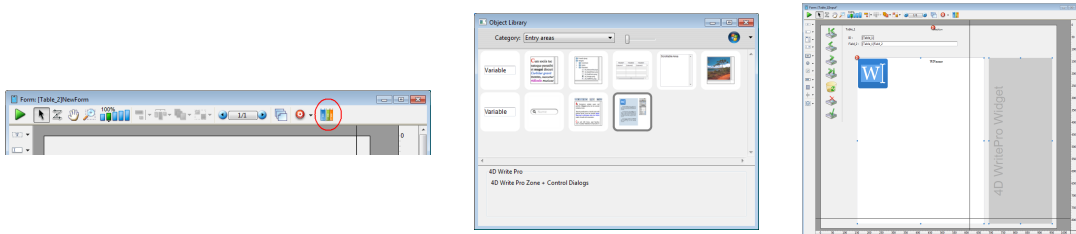
- For 4D Write files stored on disk, you can use the **WP Import document** command,
- For 4D Write files stored in BLOB fields, you can use the **WP New** command.

Compatibility notes:

- Only 4D Write documents of the last generation ("4D Write v7") are supported.
- Check which features and objects can be imported by consulting: **Which properties will be recovered from 4D Write?**
- Copying-pasting from a 4D Write document to a 4D Write Pro area is not supported for the moment. A 4D Write document can only be imported using 4D Write Pro language commands.
- Under Windows, 4D Write Pro features rely on Direct2D. With machines under Windows 7 or Windows Server 2008, make sure that the *Platform Update for Windows* component has been installed so that you can benefit from the required Direct2D version.

New 4D Write Pro object in the object library

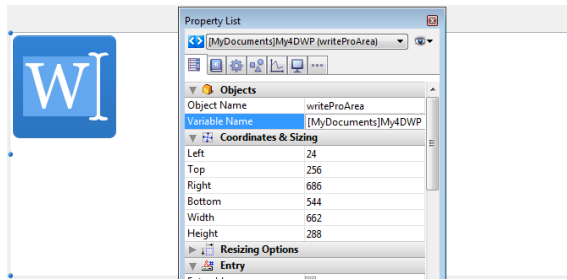
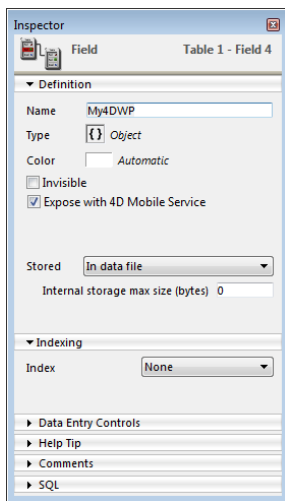
In 4D v16, the library of preconfigured objects in the Form editor includes the new **4D Write Pro** form object. Dragging and dropping this object onto a form automatically inserts a preconfigured 4D Write Pro area with an associated 4D Write Pro Widget subform containing control panels to manage the area's contents:



For more information, see [4D Write Pro area](#).

4D Write Pro: Associating an Object field

In your database structure, any 4D Object field can be used to store 4D Write Pro documents. Once the Object field intended to store your 4D Write Pro documents is defined, you can just reference it from the form containing the area. In the Form editor, enter the field name using the standard "[Table]Field" notation in the **Variable Name** area of the Property List for the 4D Write Pro area:



Your 4D Write Pro area is then associated with the Object type field.

Filtering 4D expressions

Filtering was not enabled for 4D Write Pro documents in previous versions. If your 4D Write Pro documents reference 4D methods, they will no longer be evaluated correctly once they are converted into 4D v16 or higher. "## Error # 48" messages will be displayed instead.

In this case, you must add the methods to the list of allowed methods using the **SET ALLOWED METHODS** command.

Modified commands

New commands have been added and existing ones have evolved to work with 4D Write Pro:

- **OBJECT SET HORIZONTAL ALIGNMENT**: This command supports 4D Write Pro objects. For 4D Write Pro areas only, a new `wk justify` constant is now available for the `alignment` parameter for 4D Write Pro objects, allowing you to set a justified alignment.
- **OB SET**: This command supports the definition of attributes in 4D Write Pro objects, the same way as **WP SET ATTRIBUTES**. The following syntax is supported:
OB SET (objSel | wpDoc; attribName ; attribValue {; attribName2 ; valeurAttrib2 ; ... ; attribNameN ; attribValueN})
 Limitation: you cannot pass a picture field or variable directly as an attribute value.
- **OB Get**: This command supports the definition of attributes in 4D Write Pro objects, the same way as **WP GET ATTRIBUTES**. The following syntax is supported:
OB Get (objSel | wpDoc; attribName) -> Function result
 This command has the same limitation as **OB SET**: you cannot use a picture field or variable directly as an attribute value.
- "Stringifying" 4D Write Pro attributes: If you convert a 4D Write Pro object into JSON using **JSON Stringify**, only the "title" attribute will be available in the output string. Custom attributes, if any, will be "stringified" (see "Using custom attributes" in the **Storing 4D Write Pro documents in 4D Object fields** section).
- **QUERY BY ATTRIBUTE**: As specified in the **Storing 4D Write Pro documents in 4D Object fields** section, the **QUERY BY ATTRIBUTE** command supports 4D Write Pro attributes (internal and custom) when the documents are stored in Object fields.

.4wp document format

Starting with 4D v16, you can save and reopen 4D Write Pro documents to and from disk without any loss using the native **.4wp** format.

The **.4wp** format consists of a zip folder whose name is the document title and whose contents are HTML text and images:

- HTML text combines regular HTML with 4D expressions (which are not computed) as well as 4D-specific tags,
- images are stored in a folder with the same name as the document title, next to the HTML file.

Since .4wp documents are based on HTML, they can be imported or opened in any external application supporting

HTML.

Note: The 4D Write Pro internal document format is a proprietary HTML extension, compatible with HTML5/XHTML5, but which supports its own subset of HTML/CSS attributes and tags. As a result, only HTML documents exported by 4D Write Pro can be opened by 4D Write Pro without any risk of data loss.

WP New {{ source }} -> Function result

Parameter	Type	Description
source	String, BLOB, Object	→ String: 4D HTML source, BLOB: 4D Write Blob document (.4w7/.4wt) or 4D Write Pro document (.4wp) Object: a 4D Write Pro object range
Function result	Object	↪ 4D Write Pro object

Description

The **WP New** command creates and returns a 4D Write Pro object.

By default, if you omit the *source* parameter, the command returns an empty 4D Write Pro object.

You can also use the *source* parameter, in which case the new 4D Write Pro object will be filled with the contents of the *source*. You can pass:

- a *string* parameter: In this case, you pass a 4D HTML source, i.e. a text exported by **WP EXPORT VARIABLE** with the [wk web page html 4D](#) option. This text can contain references (4D tags and expressions) and embedded images.
- a *blob* parameter: In this case, you pass either:
 - a 4D Write Pro (.4wp) format document stored in a BLOB. For more information about the 4D Write Pro document format, please refer to [.4wp document format](#).
 - a legacy 4D Write area loaded in a BLOB (BLOBs containing .4w7 or .4wt documents are supported). For a detailed list of 4D Write features that are currently supported in 4D Write Pro objects, please refer to the [Importing 4D Write documents](#) section.
If you want to import a 4D Write document (.4w7 or .4wt) stored on disk, you can also consider using the [WP Import document](#) command.
- an *object* parameter: In this case, you pass a 4D Write Pro range object. **WP New** will return a new document created from the specified range. Note that, if the range is not equal to the full document range, only the first section is exported and bookmarks are not exported, if any.

The returned object is a complete document that can be passed to the [MissingRef](#) command, for example.

Example 1

You want to create an empty 4D Write Pro object:

```
myWPObject:=WP New
```

Example 2

You want to create a 4D Write Pro object containing a simple 4D expression reference:

```
C_TEXT(myText)
myText:="Today is "
ST INSERT EXPRESSION(myText;"string(current date;System date long)";ST_End_text)
myWPA:=WP New(myText)
```

Example 3

You want to initialize your Write Pro area with a previously-created template:

```
//Export template from an existing area
C_TEXT(wpTemplate)
WP EXPORT VARIABLE(myWPArea;wpTemplate;wk_web_page_html_4D)

// use the template for a new area
```

```
C_OBJECT(myNewWPA)
myNewWPA:=WP New(wpTemplate)
```

Example 4

You want to import a 4D Write document stored in a 4D field of the current record into a new 4D Write Pro area:

```
C_OBJECT(wpArea)
wpArea=WP New([Templates]Reference_)
```

Example 5


You have defined a template document with different preformatted parts, each of them being stored as a bookmark. When producing a final document from the template, you can extract any bookmark as a new document and insert it in the final document.

```
ARRAY TEXT($_BookmarkNames;0)
WP GET BOOKMARKS([TEMPLATES]WP;$_BookmarkNames) //get the bookmarks from the template
$targetRange:=WP New //create an empty document (will be the final document)

$P:=Find in array($_BookmarkNames;"Main_Header") //handle the main header part
If($P>0)
    $Range:=WP Get bookmark range(WParea;$_BookmarkNames{$P}) //select the range
    $RangeDoc:=WP New($Range) //create a new document from the range
    WP INSERT DOCUMENT($targetRange;$RangeDoc;wk_append+wk_freeze_expressions) //wk append=after
replacement, $targetRange is equal to end of replaced text
End if
```

SET ALLOWED METHODS

SET ALLOWED METHODS (methodsArray)

Parameter	Type	Description
methodsArray	String array 	Array of method names

Description

The **SET ALLOWED METHODS** command sets the methods that are displayed in the Formula editor for the current session. The designated methods will appear at the end of the list of commands and can be used in formulas. By default (if this command is not used), no methods are visible in the Formula editor. If a formula uses an unauthorized method name, a syntax error is generated and the formula cannot be validated.

Pass the name of an array containing the list of methods to offer in the Formula editor in the *methodsArray* parameter. The array must have been set previously.

You can use the wildcard character (@) in method names to define one or more authorized method groups.

If you would like the user to be able to call 4D commands that are unauthorized by default or plug-in commands, you must use specific methods that handle these commands.

Note: The mechanism for restricting access to commands and methods in the Formula editor can be disabled for all users or for the Designer and Administrator by means of an option on the "Security" page of the Database Settings. If the "Disabled for all" option is checked, the **SET ALLOWED METHODS** command will have no effect.

Example

This example authorizes all methods starting with "formula" and the "Total_general" method in the Formula editor:

```
ARRAY STRING(15;methodsArray;2)
methodsArray{1}:="formula@"
methodsArray{2}:="Total_general"
SET ALLOWED METHODS(methodsArray)
```