



5.3

PrintList Pro

User Manual

Version 5.3



e-Node
30 rue de la République
33150 Cenon
France



www.e-node.net



Contents

About PrintList Pro	7
What is PrintList Pro, and what can I do with it?	7
Technical Details	7
Compatibility Information.	7
Technical Support	7
Installing the plugin	8
Using PrintList Pro in Demo mode	8
Licensing	9
Definitions	9
Free updates	9
License types	10
Registering your PrintList Pro License	11
Quick and easy way – End-user online instant activation	11
Quick and easy way – Developer online instant activation	11
The Demonstration mode dialog	12
Registering Server licenses	13
Using a text file	15
Combining methods	15
Online instant activation	16
Getting started with PrintList Pro	19
Creating your first PrintList Pro Area	19
Working with PrintList Pro Commands	20
Command parameters	20
When to use the PrintList Pro Commands	20
Upgrading from Previous Versions of PrintList Pro	21
Two major differences with previous versions	21
Compatibility Notes	21
New Configuration commands	22
New RGB commands	22
New Break Processing commands: Computed Breaks	22

Additions to existing commands23

New break features23

Printing an AreaList Pro area23

Configuring PrintList Pro 24

Using Defined Constants with PrintList Pro24

Specifying the Arrays to Print24

Printing Records25

Headers25

Sorting Arrays25

Formatting25

Styles26

 Constants26

 Column and Header Styles26

 Row-Specific Styles26

 Cell-Specific Styles26

 Styled text26

Colors28

 Defining colors28

 Column and Header Colors28

 Cell-Specific Colors29

Multiple Lines in each Row29

Variable Height Rows29

Column Widths30

Dividing Lines, Frame and Header Separator Lines30

 Hairline Line Width30

 Double lines30

Using Picture Arrays30

End of Page Callback Method31

Performance Issues with Formatting Commands31

Borders and Frames31

Header/Cell Icon Support31

 Picture Objects in Cells31

 Picture Objects in Headers31

Configuration Commands 32

PL_Register32

%PrintListPro34

PL_AddColumn35

PL_SetArraysNam35

PL_SetHeaders36

PL_SetFormat 37

PL_SetWidths 41

PL_SetHdrStyle 41

PL_SetHdrOpts 42

PL_SetMiscOptions 43

PL_SetStyle 43

PL_SetForeClr 44

PL_SetForeRGBColor 45

PL_SetBackClr 46

PL_SetBackRGBColor 47

PL_SetColBackColor 48

PL_SetColBackRGBColor 49

PL_SetRowStyle 50

PL_SetRowColor 51

PL_SetRowRGBColor 52

PL_SetDividers 53

PL_SetRGBDividers 54

PL_SetFrame 55

PL_SetRGBFrame 56

PL_SetHeight 57

PL_SetSort 58

PL_SetColOpts 58

PL_SetCellStyle 59

PL_SetCellColor 61

PL_SetCellRGBColor 62

PL_SetCellIcon 63

PL_SetCellBorder 65

PL_SetCellFrame 66

PL_SetPageProc 67

PL_GetVersion 67

PL_Load 68

PL_Save 68

Using the Callback Methods 69

Summary 69

Warnings 69

End of Page Callback 69

Custom Calculations in a Break 70

Custom Calculations in a Break Header 70

Calculated Column Callback 71

Computed Breaks 72

Field and Record Commands	73
Using the Field Printing Capability	73
Temporary Arrays	73
Arrays and Fields	73
Printing 4D Fields	73
Fields from Related One Tables	73
Sorting	73
Time Data	73
Maximum Number of Records Printed	74
Performance Issues When Printing Fields	74
Commands	74
PL_SetFile	74
PL_SetFields	75
Calculated Columns	76
Setting a Calculated Column (field mode)	76
Setting a Calculated Column (array mode)	77
Setting the Callback Method	77
Field mode example	78
Array mode example	78
Commands	79
PL_SetCalcCall	79
Break Level Processing	80
About PrintList Pro Break Level Processing	80
When Do Breaks Occur?	80
Using PrintList Pro Break Level Processing	82
Setting a Break Level	82
Text Overflow and Justification in Breaks	83
Built-in Calculations	83
Custom Calculations	83
Suppressing Repeated Values	83
Style and Color in Breaks	83
Multiple Lines in a Break	84
Lines Displayed in a Break	84
Hide the Detail Area	85
Page Breaks	85
Variable Height Breaks	85
Using Break Headers	85
Using Computed Breaks	86
Commands	87

PL_SetPageBreak	87
PL_SetBrkOpts	88
PL_SetBrkOrder	88
PL_SetRepeatVal	89
PL_SetBrkText	89
PL_SetBkHText	91
PL_SetBrkFunc	92
PL_SetBkHFunc	92
PL_SetBrkStyle	93
PL_SetBkHStyle	93
PL_SetBrkColor	94
PL_SetBrkRGBColor	95
PL_SetBkHColor	95
PL_SetBkHRGBColor	96
PL_SetBrkHeight	97
PL_SetBkHHeight	97
PL_SetBrkRowDiv	98
PL_SetBrkRowRGBDiv	98
PL_SetBrkColOpt	99
PL_SetBrkColRGBOpt	100
PL_SetBkHColOpt	100
PL_SetBkHColRGBOpt	101
PL_ProcessArrays	102
PL_GetBreakValue	103

Examples 104

Example 1 — One record current selection	104
Example 2 — Multiple record current selection	107
Example 3 — Adding a total line to the list	109
Example 4 — Break Level Processing	112
Example 5 — Computed Breaks	115

PrintList Pro Constants 119

Text Style Tags 121

Copyrights and Trademarks 123

Index 124



About PrintList Pro

What is PrintList Pro, and what can I do with it?

PrintList Pro is an easy-to-use tool for printing arrays and records on 4D layouts. It lets you print arrays or fields.

PrintList Pro is the perfect complement to [AreaList Pro](#), providing a full-featured plug-in, which can be used to print columns of data. You can use PrintList Pro for any standard columnar output (arrays or fields) and it be configured to easily print a PrintList Pro object, retaining all formatting features.

Because PrintList Pro is a plug-in, it is very fast, and provides capabilities not available to you using native 4D arrays or report printing tools, such as automatic column sizing, custom formatting, robust break level processing, calculated columns and more.

Data is passed to PrintList Pro using 4D arrays, or field references. If only two columns need to be printed, create two arrays or specify two fields and pass them as parameters to PrintList Pro. No string parsing or other contortions are needed.

PrintList Pro can be used with just one command — no special formatting is required. For those cases when more control is needed, several optional commands give you complete control over the appearance of the area.

Special tools are implemented if you wish to customize the appearance and configuration of PrintList Pro, allowing the customization to be implemented rapidly.

PrintList Pro's break level processing includes the ability to apply a variety of built-in calculations as well as the ability to perform custom calculations. Complete control over style, color, and formatting of all break level information is given.

PrintList Pro provides the ability to print up to **32767** columns (subject to memory limitations).

Technical Details

Compatibility Information

PrintList Pro version 5 is compatible with 4D v11, v12, v13, v14 and v15, for both MacOS and Windows (including 32-bit and 64-bit servers). It requires MacOS 10.7.5 or higher and Windows 7 or better.

Technical Support

Technical support for PrintList Pro is provided via the [online web forums](#).

Items that are new or modified in PrintList Pro version 5 are displayed in pink (magenta) characters.



Installation

Installing the plugin

PrintList Pro is provided as a bundle for both Windows and MacOS: there is just one version for both platforms. To install it, simply copy the file **PLP.bundle** into your Plugins folder.

Plugins folders can be located in one of two locations:

- In the 4D application folder (4D or 4D Server). When plugins are installed in this location, they will be available to every database that is opened with that application.
- Next to the database structure file for your project: in this case, the plugin will only be available to that database. On MacOS, this means that the Plugins folder must be placed within the database package or folder. To open a package, ctrl-click on the package and choose **Show Package Contents** from the contextual menu.

Using PrintList Pro in Demo mode

You can use PrintList Pro in Demo mode for 20 minutes, after which time it will cease to work. When this becomes annoying, it's time to buy a license, which you can do [on our website](#).

Licenses are either linked to the 4D product number, the workstation or the company name as described below.

Licensing

Like all e-Node plug-ins, PrintList Pro offers several license types. There are no such things as MacOS vs Windows or Development vs Deployment.

For current pricing, please [see the ordering page on our website](#).

Definitions

Regular and merged

- **Regular licenses** are used for applications that are opened with 4D Standalone or 4D SQL Desktop, or with 4D Server, either in interpreted or compiled mode (doesn't make a difference regarding plugin licensing).

These can be either single user or server databases and they are linked to the 4D or 4D Server license: you need to provide the number returned by the "Copy" or "eMail" buttons from the plugin demonstration mode alert (this number is actually the 4D command **GET SERIAL INFORMATION** first parameter). This number is a negative long integer such as -1234567.

- **Merged licenses** are used for double-clickable applications built with 4D Volume Desktop (single user) or with 4D Server by means of the 4D Compiler module.

These licenses are linked to the machine ID (single user workstation or server): you need to provide the number returned by the "Copy" or "eMail" buttons from the plugin demonstration mode alert (this number is calculated from the single user or server machine UUID). On 4D Server any remote client will return the server number. This number is a positive long integer such as 1234567.

In both cases the [demonstration mode dialog](#) will display the proper number according to the current setup (regular or merged) and the "Copy" and "eMail" buttons will use it as well.

License keys

- **Final licenses keys** are delivered by [e-Node](#) once you have provided the associated number as described above (4D serial information or machine ID). They activate the plugin either through [4D code](#) or the [Register button](#) from the [demonstration mode dialog](#).
- **Master keys** are delivered upon order if you opt for the [Online instant activation](#) system. The final license key is self-generated by the plugin and stored into the [license file](#), so you don't have to bother with 4D serial information or machine IDs.

Free updates

- **Regular licenses.** A new license will be supplied for free at any time (maximum once a year) if you change your 4D version or get a new 4D registration key for the same version, provided that your previous license match the current public version at exchange time. This rule applies whether you are already using the new version or not: just specify that you also want a key for the older version as well as the current one when you order an upgrade.
- **Merged licenses.** These licenses are independent from the 4D versions and product numbers. They will remain functional if you upgrade e.g. from 4D v14 to 4D v15 on the same machine (single user workstation or server).

You'll only need to update a merged license if your machine or motherboard is replaced (a new license will be supplied for free in this case, provided that your previous license match the current public version at the exchange time), or if you install a paid upgrade of the plugin.

Note: if you are using several concurrent versions of 4D you will need one plugin license for each version.

License types

- **Single-user.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode, including merged) of applications that are opened with 4D Standalone or 4D SQL Desktop or built with 4D Volume Desktop.
- **Server.** These licenses allow development (interpreted mode) or deployment (interpreted or compiled mode, including merged servers/remotes) on 4D Server with up to 10 users (“small server”), 11 to 20 users (“medium server”) or more (“large server”).
- **Unlimited Single User.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode, including merged) on any number of 4D Standalone (or single user merged applications built with 4D Volume Desktop) that run your 4D application(s).

It is a yearly license, which expires after the date when it is to be renewed. Expiration only affects interpreted mode. **Compiled applications using an obsolete license will never expire.**

A single license key will unlock all setups on all compatible 4D versions and all versions of the plugin. The license key is linked to the developer/company name.

This license allows deployment (selling new application licenses, updates or subscriptions) while the license is valid. **No new deployment may occur after expiry without a specific license** (merged or regular). End-users running deployments sold during the license validity period remain authorized without time limit, provided that they are no longer charged for the application using the plug-in (including maintenance or upgrades).

- **OEM.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode, including merged) on any number of 4D Servers (any number of users), 4D Standalone or single user/remote merged instances that run your 4D application(s).

It is a yearly license, under the exact same terms as the Unlimited Single User license described above, except that it also covers server deployments.

- **Unlimited OEM.** This license is a global OEM license, covering any combination of the plug-ins published by [e-Node](#), including [AreaList Pro](#), [SuperReport Pro](#), [PrintList Pro](#), [CalendarSet](#) and [Internet ToolKit](#) in all configurations.
- **Partner license.** This license matches 4D’s annual Partner subscription and covers all the plug-ins published by [e-Node](#), including [AreaList Pro](#), [SuperReport Pro](#), [PrintList Pro](#), [CalendarSet](#) and [Internet ToolKit](#).

For each product, a single registration key allows development (interpreted mode) or deployment (interpreted or compiled mode, except merged) on all 4D Standalones and 4D Servers (2 users) regardless of 4D product numbers, OS and versions. No merged applications.

This is a yearly license, expiring on February 1st (same date as 4D Partner licenses). Expiration only affects interpreted mode. **Compiled applications using an obsolete license will never expire.**

Note: you don’t have to be a 4D Partner subscriber to subscribe to the e-Node Partner license.

Registering your PrintList Pro License

Once you have purchased your license, you will receive a registration key. This code must be registered each time the database is started.

There are three ways to register your license:

- using the Demo mode dialog "[Register](#)" button,
- through a [text file](#),
- in your 4D code with a [command](#).

Both Register button and 4D code registrations can be performed in one single step through the [Online instant activation system](#).

Yearly licenses such as [Unlimited single user](#), [OEM](#) and [Partner](#) licenses do not require any serial information or online instant activation. The only way to register these licenses is through the [PL_Register](#) command.

Quick and easy way – End-user online instant activation

1. Make sure that the machine where the plugin will be used is connected to the Internet (single user workstation or in server mode the first remote client that will connect to the server).
2. Launch your application. Displaying any layout that uses the plugin will trigger the [demonstration mode dialog](#).
3. Enter the [Master key](#) that was delivered by [e-Node](#).
4. The plugin will display an alert indicating that it is now registered.

Note: this method does not require your source code to be modified or recompiled.

Quick and easy way – Developer online instant activation

1. Put the following lines of code into your **On Startup** database method, with the [Master key](#) that you received and your email address:

```
C_LONGINT ($result)
```

```
$result:=PL_Register ("yourMasterKey";0;"youremail@something.xxx") //0 if successful
```

2. Make sure that the machine where the plugin will be used is connected to the Internet (single user workstation or in server mode the first remote client that will connect to the server).
3. Install your application.
4. Launch your application. Displaying any layout that uses the plugin will silently (no dialog) register it.
5. You will receive an email with the [final key](#) that was issued and the IP address of the user site.

If the site has no Internet connection or if you want to use the plugin license system to help protect your own software copy, you can manage the final key registration yourself using one of the following methods.

The Demonstration mode dialog

The demonstration mode dialog is used for both [Online instant activation](#) and manual registration, unless the plugin is registered with a [final key](#) or [master key](#) through the 4D code.

When using manual registration, single user and server licenses require that you first send us the relevant information (serial or machine ID, see [Definitions](#)).

Note: sending the serial information or machine ID is not needed with the [Online instant activation](#) system.

This action is performed from the Demo mode dialog, which is displayed upon the first call to the plugin.

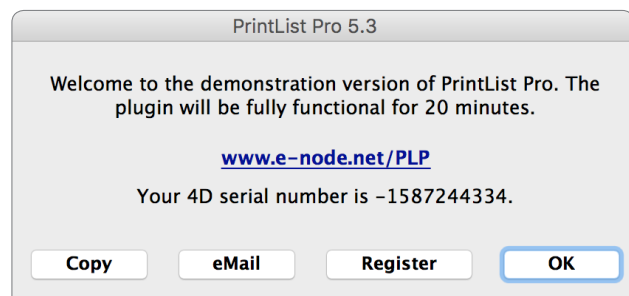
To trigger this display and enable your users to register without actually calling a command or setting up an area, you can also pass an **empty** string to `PL_Register` and the dialog will show:

```
C_LONGINT ($result)
$result:=PL_Register ("") //display the dialog
```

Note: calling `PL_Register` with any key (valid or invalid) will not display the dialog.

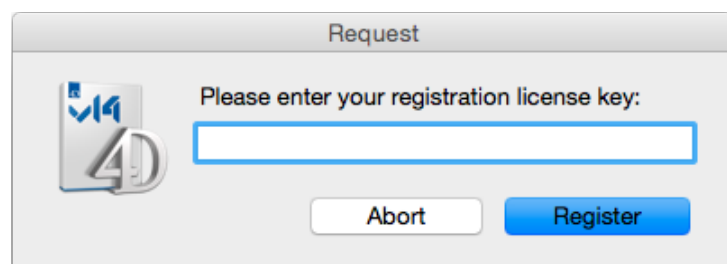
■ Retrieving the serial/machine information

The Demo mode dialog includes all relevant information (serial or machine ID, see [Definitions](#)) to obtain your license, as well as a “Copy” button to put this information into your clipboard or a text file, an “eMail” button to email the information to e-Node’s registration system and a “Register” button to enter your license key once received:

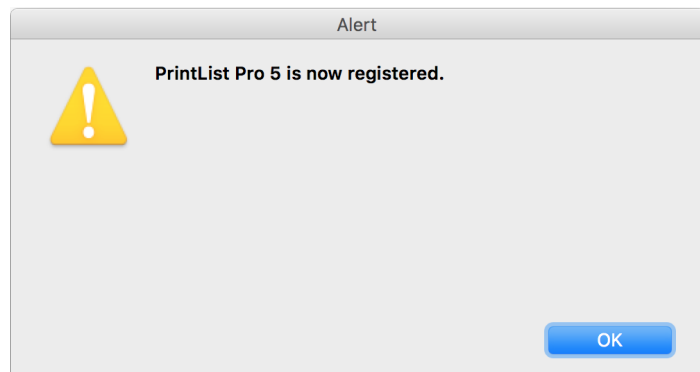


■ Using the “Register” button

Clicking on this button will display a standard 4D request to enter your registration key:



Paste or drag and drop your registration key and, if correct, the plug-in will be registered for all future uses on this workstation:



Note: if 4D does not activate the **Edit > Paste** menu item click **Abort** and **Register** again, or try drag and drop.

Note: you can directly paste the [Master key](#) that was delivered when using the [Online instant activation](#).

Registering Server licenses

Similarly, server licenses can be registered from the demonstration mode dialog without having to modify your code and use [PL_Register](#) (which of course you can do with any license type). In this case, the 4D Licenses folder, serial information or machine ID used will only be the 4D Server information, not the client workstation's.

Server licenses can be registered on any client workstation (remote mode), or on 4D Server itself.

■ Registering in Remote mode

The server and all workstations can be registered from any single client workstation connected to the server. As in Single user mode, the Demo mode dialog will be displayed on a client workstation when one of the following conditions are met:

- Calling a PrintList Pro command other than **PL_Register** with a non-empty parameter
- Calling **PL_Register** with an empty string

Use the **Copy**, **eMail** and **Register** buttons just as above and your server will be registered for all workstations.

Note: any other workstations previously connected (before registration occurred) will need to re-connect to the server to be functional.

■ Registering on 4D Server

To directly register the server and all workstations from the server machine itself, you need to display the Demo mode dialog on the server.

Call **PL_Register** with an empty string in the **On Server Startup** base method:

```
C_LONGINT ($result)
$result:=PL_Register ("") //display the dialog
```

Use the **Copy**, **eMail** and **Register** buttons just as above and your server will be registered for all workstations.

Note: the dialog will automatically be dismissed on the server after one minute in order not to block client connections (the server is only available to client workstations once the On Server Startup method has completed).

■ Merged licenses notes

Both methods can be either used with regular or merged servers and client workstations.

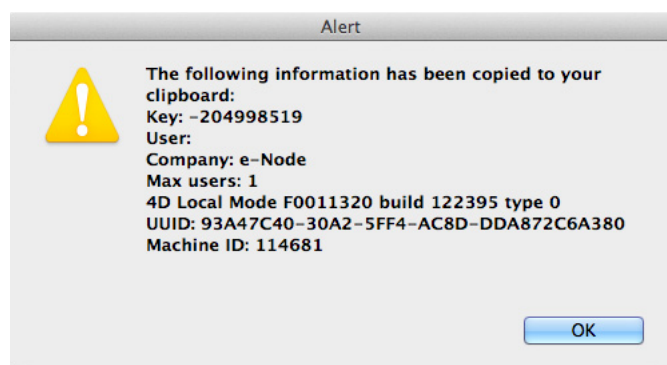
- Regular licenses are linked to the 4D Server serial information
- Merged licenses are linked to the 4D Server machine ID

Note: merged licenses will keep working if your 4D Server serial information is modified (upgrading or 4D Partner yearly updates), or if any client workstation hardware is changed.

It will only need to be updated if the 4D Server hardware is changed, or if the plugin itself requires a new key (paid upgrades upon major version changes).

You may want to register your merged server without having to turn off the database to modify the code. We have created a utility database to manage this - it's called Get Serial Info and you can download the appropriate version for your 4D version [from the e-Node server](#).

This is possible using any 4D setup on the server machine (such as a standard developer single user 4D). Keeping your production server alive, open the [Get Serial Info database](#) with 4D on the same server machine. Ignore the demonstration mode dialog (if your single user 4D is not registered for the plugin) and wait for the next Alert:



A text file is also saved with the same information.

The last line "Machine ID" is the number that you need to send in order to receive your merged server registration key.

You can also check the machine ID in standalone mode (or on any remote client with the built-client application or in interpreted mode as long as it is running on the same server machine) with [AreaList Pro](#) using the following call:

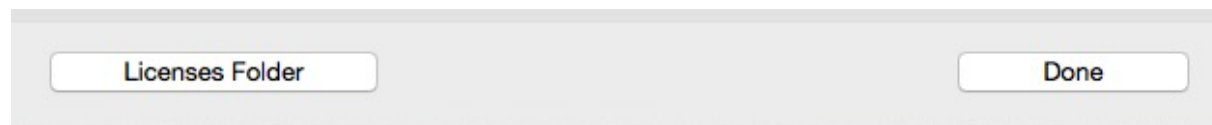
```
C_LONGINT($machineID)
$machineID:= AL_GetAreaLongProperty (0;"mach")
```

Note: you don't need an AreaList Pro license to do this.

Using a text file

Alternately, you can place a plain text file into your 4D Licenses folder.

To open this folder from 4D use the 4D Menu **Help > Update licenses**, then click the **Licenses Folder** button:



The text file **must** be called “PLP5.license4Dplugin” and be a plain text type file.

Just paste all your licenses for PrintList Pro v5.x, one per line, e.g.:

```
MyLicense1  
MyLicense2  
MyLicense3
```

Any license type can be included into this document, except unlimited single user, OEM and Partner licenses.

Note: the Demo mode dialog **Register** button actually does this: create the text file and include the license key, or add the license key to the existing document if any.

Note: when using the [Online instant activation system](#), the [Master key](#) is automatically converted to a [Final key](#) according to the current environment and this final key is stored into the license file.

Using PL_Register

1. Open the **On Startup** database method
2. Call the [PL_Register](#) function with your registration key - for example:

```
$result:=PL_Register ("YourRegistrationKey") //result = 0 means registration was successful
```

If you have several licenses for different 4D setups you can call **PL_Register** multiple times in a row without further testing. See the [Example with multiple calls](#).

Combining methods

When such a file exists in the Licenses folder PrintList Pro will check for valid licenses from this document as a first action before anything else (including checking any **PL_Register** command).

If a valid license is included into the “PLP5.license4Dplugin” document any calls to **PL_Register** will return zero (for “OK”). Therefore you can mix modes and use the text file (or **Register** button) as well as the command.

Unlimited single user, OEM, temporary and Partner licenses can only be entered through the **PL_Register** command.

Online instant activation

As of version 5.3, PrintList Pro provides an automated solution to register itself using an Internet connection.

This feature can be helpful whenever you don't want to bother your end user with plugin registration, or want to save the time to collect the serial/machine ID, or any other reason when you expect the process to be entirely and automatically managed from the client site.

It can also be used for your own development tools, removing the need to modify your 4D code to include or update registration licenses.

Note: the site must have an open outgoing HTTP Internet connection available.

“Master” keys

The basic principle is that we deliver a non-assigned license key, called “[master key](#)”, which you use in your call to [PL_Register](#) in your **On Startup** database method. This key will be used to generate valid keys for the plugin and environment, called “[final keys](#)”.

One single master key can generate as many final keys as you need, in case you order several licenses of the same kind (regular or merged, single user licenses or server licenses of the same size).

A master key looks like a final key, except that the second part is the plugin code name (same as the [license file](#) name) instead of the serial/machine ID, e.g. “123456-PLP5-xyz”.

Passing a master key as the first parameter to **PL_Register** when the plugin has not been previously registered by any of the methods above will result in a connection attempt to e-Node's license server as described below.

Master keys can also be entered by the user through the registration dialog. See [Quick and easy way – End-user online instant activation](#).

Process

If the plugin has not been previously registered (through Online instant activation, text file, register button or [PL_Register](#) with a final key), and if **PL_Register** receives a master key in its first parameter, it will recognize it as such, then:

1. Connect to e-Node's license server.
2. Ask the server if the master key has not been assigned yet (or if the master key is designed to generate several final keys, if there is any unassigned key up to that number).
3. Send the serial information (regular licenses) or the machine ID (merged licenses) to the license server.
4. If an error is detected (such as master key not matching the current setup) return an error to **PL_Register**.
5. If the master key is valid, receive its final key from the license server then register itself (writing into the license file).

Note: if a final key has already been issued for this serial/machine ID using this master key, it is simply resent.

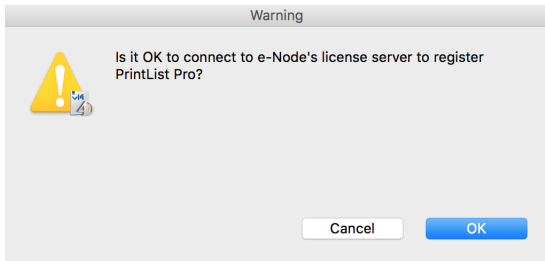
User interface

In addition, `PL_Register` second parameter allows optional settings regarding the user interface in the Online instant activation process.

`C_LONGINT ($result)`

`$result:=PL_Register ("Master key";0 ?+1 ?+2 ?+3;"youremail@something.com") //all dialogs`

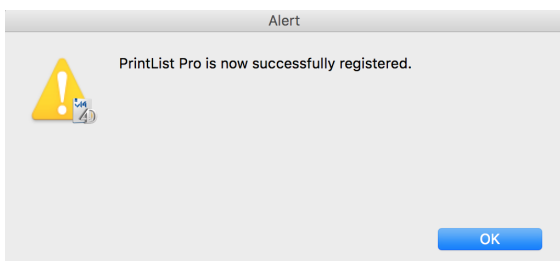
Display a confirmation dialog before step 1



Display an alert at step 4



Display an alert at step 5



eMail notification

The third parameter to [PL_Register](#) (optional) is the developer email to whom the information will be sent (if this parameter is used and non empty, of course).

The emailed information includes both the final key issued and the IP address from where it was requested (and to where it was sent for registration).

- When a key is issued:

Title: PLP5 license

Body:

License 123456-123456789-abcdefgh

issued to 12.34.56.78

- When a key is resent:

Title: PLP5 license

Body:

License 123456-123456789-abcdefgh

resent to 12.34.56.78

The default mode (master key being passed as the only parameter) is silent: no confirmation, no alert, no email.

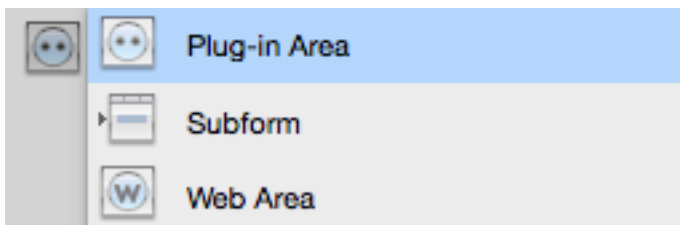
3

Getting started with PrintList Pro

Creating your first PrintList Pro Area

It's easy to create your first PrintList Pro list area.

1. Create a new form, or open an existing one that you want to add a list to.
2. Choose **Plugin Area** from the Plugin/Subform/Web Area button in the object bar:



3. Your cursor will turn into a crosshair. Draw a box on the form in the size that you want your list to be. This will create a rectangular box named **Plugin Area**.
4. In the Property List window, choose **PrntListPro** from the **Type** popup menu. (If the "PrntListPro" option is not available, please refer to the [installation](#) instructions).
5. Enter a name for your new area in the Variable Name field in the Property List window.
6. Your area will now show the PrintList Pro version and copyright information.

The variable name will be used as a parameter for the PrintList Pro commands.

Be careful to never have two PrintList Pro objects with the same variable name on a 4D form.

Working with PrintList Pro Commands

Each command you write must adhere to a specific syntax in order for it to be correctly understood by PrintList Pro. Some commands return a **result**: these are functions. You can use the commands and functions to configure every operation performed by PrintList Pro, and to get various informations.

Each PrintList Pro command has a syntax, or rules, that describe how to use the command in your 4D database. For each command, the name of the command is followed by the command's parameters, and result in case of functions.

A PrintList Pro command syntax looks like this:

PL_SetHdrStyle

(areaRef:L; columnNumber:L; fontName:T; size:L; styleNum:L)

The parameters are enclosed in parenthesis, and separated by semicolons.

Command parameters

Each parameter is followed by a colon and a letter indicating the type of data required for that parameter:

:L - longint

:O - blob

:P - picture

:R - real

:T - text

:Y - array

:Z - pointer

Each is preceded by one of three arrow signs, which indicate whether it is a value that you pass to the command or one that the command returns to you, or a value that is passed, then modified and returned by the command in the same parameter:

→ parameter A value that you pass to the command

← parameter A value that is returned by the command

Note: when calling a plugin command, all omitted parameters are initialized to the NULL of the respective types (0, 0.0, "", !00:00:00!, ...).

When to use the PrintList Pro Commands

The PrintList Pro commands must only be executed in the [On Printing Detail](#) phase of a form method or object method during the execution of the **PRINT SELECTION** or **PRINT RECORD** command.

The **PRINT SELECTION** command will execute a [On Printing Detail](#) phase for each record in the current selection (and requires at least one record in the current selection to be executed at all). PrintList Pro will print the array(s) in any PrintList Pro object once for every record in the current selection.

If you wish to use **PRINT SELECTION** to print an array only once, ensure that there is only one record in the current selection of the table used for printing (the one that holds the layout, which doesn't have to be related to the data that is actually printed).

If you wish to use **PRINT RECORD**, ensure that there a current record in the table used for printing (the one that holds the layout, which doesn't have to be related to the data that is actually printed).

Upgrading from Previous Versions of PrintList Pro

To upgrade to PrintList Pro version 5, simply install it as described in the [Installation](#) section of this manual, replacing your older version.

Two major differences with previous versions

As opposed to v4.x (and earlier releases):

- [PL_Register](#) returns 0 if registration was successful
- The 4D project method [Compiler_PLP](#) is no longer needed

Compatibility Notes

New features

- PrintList Pro version 5 supports Unicode for printing.
- Styled text is supported.
- PrintList Pro version 4.x commands are still here: your previous code should work fine, give or take the few changes and minor deprecated features below.
- The [PL_Register](#) command returns 0 for OK and an integer between 1 and 12 if not OK.
- The 4D project method [Compiler_PLP](#) is no longer needed.

In addition, PrintList Pro uses native drawing. Not all fonts contain italic (or bold) variations and those technologies do not synthesize styles. For example Geneva on most Macs has only the Regular typeface, Arial has Regular, Bold, Italic and Bold Italic typefaces.

Note: if bold or italic styles are set, but not installed for the font, PrintList Pro will print regular (plain) characters.

Removed features

- Picture escapes in text.
- Outline and shadow styles.
- Escape characters ("^1234" in cell data or headers) , including the first parameter to [PL_SetMiscOptions](#)
- Patterns. They are interpreted by PrintList Pro version 5 as transparency ratios:
 - “black”: 100 %
 - “darkgray”: 75 %
 - “gray”: 50 %
 - “lightgray”: 25 %
 - “white” or “none” or “” or anything else: 0 % = no drawing
- [PL_SetCellIcon](#) now only supports pictures from the Picture Library. ‘cicn’ and ‘PICT’ resources are no longer supported.

Removed commands

- [PL_GetAdvProps](#) (did nothing in PrintList Pro 4.7, anyway)
- [PL_SetArrays](#) (must use [PL_SetArraysNam](#))
- [PL_SetHeaderIcon](#) (use [PL_SetCellIcon](#) with `cellRow = 0`)
- [PL_SetSubSelect](#)
- [PL_SaveData](#) (was documented as obsolete – use the new [PL_Save](#) in XML)
- [PL_RestoreData](#) (was documented as obsolete – use the new [PL_Load](#) in XML)

New Configuration commands

[PL_AddColumn](#) (areaRef:L; dataPointer:Z; insertAt:L) → result:L

- add column at position `insertAt`; zero means append to the end
- when in Records mode, `dataPointer` should be a pointer to a field
- when in Arrays mode, `dataPointer` should be a pointer to an array (must not be a local array!)

Note: this command supports the component architecture (using arrays from the host database in a component and vice versa).

[PL_Load](#) (areaRef:L; XML:T) → result:L

- initialize an area from XML

[PL_Save](#) (areaRef:L; XML:T) → result:L

- save the area settings into XML

New RGB commands

[PL_SetBrkColRGBOpt](#) (areaRef:L; breakLevel:L; columnNum:L; showColDivider:L; lineWidth:F; dividerRed:L; dividerGreen:L; dividerBlue:L)

[PL_SetBkHColRGBOpt](#) (areaRef:L; breakLevel:L; columnNum:L; showColDivider:L; lineWidth:F; dividerRed:L; dividerGreen:L; dividerBlue:L)

[PL_SetBrkRowRGBDiv](#) (areaRef:L; lineWidth:F; dividerRed:L; dividerGreen:L; dividerBlue:L)

New Break Processing commands: Computed Breaks

[PL_ProcessArrays](#) (callbackMethodName:T; breakArrays:Y; dataArrays:Y; useDetail:L) ^a error:L

[PL_GetBreakValue](#) (handle:L; column:L; calculation:L) ^a value:F

See [Using Computed Breaks](#) for details about these powerful new commands, which can be used as array utilities without need to print anything or set up a plug-in area.

Additions to existing commands

- [PL_SetColOpts](#) has a new longint parameter at the end (ignored on Mac): 0 = use GDI plus for drawing (default), anything else: use GDI for drawing

PL_SetColOpts (areaRef:L; hideLastColumns:L; hideDetailArea:L; drawingEngine:L)

- [PL_SetFormat](#) has three new parameters at the end:

longint: 0 = plain text (default), anything else = attributed (multi-style) text

Note: the multi-style property is applied to all formatting in that column... breaks have to account for that!

real: line spacing to use, 0 is substituted by 1 (default)

longint: vertical alignment 0 - default, 1 - top, 2 - center, 3 - bottom

PL_SetFormat (areaRef:L; columnNumber:L; format:T; columnJust:L; headerJust:L; usePictureHeight:L; attributed:L; lineSpacing:F)

- Double lines (typical in accounting) are now supported in breaks: just use 2.0 as the **lineWidth** (5th parameter to [PL_SetBrkColOpt](#)/[PL_SetBrkColRGBOpt](#)/[PL_SetBkHColOpt](#)/[PL_SetBkHColRGBOpt](#)).

Two 0.25 point lines will be printed.

New break features

In break calculations, "\Var" will compute variance and "\Dev" will compute standard deviation.

Printing an AreaList Pro area

To print [AreaList Pro](#) areas, just call **AL_Save** and [PL_Load](#): they use the same XML UTF-8 format.

Then set row/cell options and you are done. Or add break processing options.

Using this feature will allow you to indirectly apply AreaList Pro v9 numerous formatting options (using the v9 property-based API) to a PrintList Pro area using an AreaList Pro area as the source.

Note that this feature is only available for persistent AreaList Pro properties. Refer to AreaList Pro manual for property persistency.



Configuring PrintList Pro

A PrintList Pro object is initialized in the [On Printing Detail](#) phase as the record is about to be printed.

This initialization will be contained in the PrintList Pro plug-in area object method or in the form method.

Using Defined Constants with PrintList Pro

There are defined [constants](#) that may be used as values for many parameters in the PrintList Pro commands. See the Constants tab of the Explorer in the 4D Design environment.

These constants are categorized according to the type of command that they are associated with, such as [PLP Break Levels](#), [PLP Colors](#), etc.

Specifying the Arrays to Print

4D arrays are passed to PrintList Pro via the [PL_AddColumn](#) or [PL_SetArraysNam](#) commands.

These commands must be called before any other PrintList Pro commands are executed.

These two functions return 0 if successful, or an error code indicating the problem:

Constant	Value	Action
PL SetArrays Passed	0	
PL Not an array	1	Check to make sure all arrays are correctly typed
PL Wrong type of array	2	Pointer and two-dimensional arrays are not allowed
PL Wrong number of rows	3	Make sure that all arrays have the same number of elements
PL Maximum number of arrays exc	4	32767 arrays is the maximum
PL Not enough memory	5	Unlikely these days

Up to **32767** arrays can be printed by PrintList Pro, with up to fifteen columns specified in each call to **PL_SetArraysNam**.

The position of the first array, **columnNumber**, and the number of arrays, **numArrays**, are also specified in this command. All array types except for pointer are allowed, and all arrays must have the same number of elements.

The maximum number of rows is only limited by available memory.

Aternately to standard single-dimension arrays, one dimension of a two-dimensional array may be passed to **PL_SetArraysNam** e.g. "My2DArray{1}" may be passed as **array1**.

PL_AddColumn can be used to add or insert an array or field column through a pointer.

Printing Records

PrintList Pro provides the capability to print 4D records directly, rather than using arrays. Please read the section [Field and Record Commands](#) for more information.

Headers

The column header labels are set using [PL_SetHeaders](#). The headers can be printed on all pages, the first page or not at all using [PL_SetHdrOpts](#).

The font, size, and style of each header may be set individually using [PL_SetHdrStyle](#). The justification may be set using [PL_SetFormat](#) and the color of the headers using [PL_SetForeClr](#) or [PL_SetForeRGBColor](#).

Multiple lines of text may be shown in the headers using [PL_SetHeight](#).

Sorting Arrays

PrintList Pro can perform multi-level sorting upon all the arrays using [PL_SetSort](#).

Up to 15 levels of sorting are available, and each column specified in the sort order can be sorted in either ascending or descending order.

While 15 columns can be used for the sorting criteria, all the arrays passed to PrintList will stay “in sync” and reflect the new sort order.

Some of the arrays can be hidden from printing using [PL_SetColOpts](#), which allows all the arrays to be kept in sync for sorting purposes, yet hides them during actual printing.

If the arrays passed to PrintList Pro are already sorted, use [PL_SetBrkOrder](#) to communicate the sort order to PrintList Pro without performing another sort.

Also, repeated values in a list can be suppressed using [PL_SetRepeatVal](#). Please read the section [Break Level Processing](#) for more information.

If a column containing a picture array is passed to [PL_SetSort](#), it will be ignored (skipped).

Formatting

Use [PL_SetFormat](#) to control the format and justification of all array information.

All valid 4D formats may be used including any custom formats created in the Design Environment.

If you are calling PrintList Pro from a component, make sure that the Design formats that you are using are defined in the component itself.

See [Break Level Processing](#) for information about formatting break headers and break footers.

Styles

Constants

Styles are set using 4D constants. The different values in the table below can be added together to produce combinations of styles. For example, bold italic has a value of 3.

Value	Style (constant)
0	<u>Plain</u>
1	<u>Bold</u>
2	<u>Italic</u>
4	<u>Underline</u>
8	Outline (obsolete)
16	Shadow (obsolete)
32	Condensed (obsolete)
64	Extended (obsolete)

Column and Header Styles

Styles for arrays can be set on a column by column basis using [PL_SetStyle](#) to set the style for the data, and [PL_SetHdrStyle](#) to set the header style.

If a 0 (zero) is used in the **columnNumber** parameter, the style will be applied to all columns.

Row-Specific Styles

[PL_SetRowStyle](#) is used to set the font and style of a specified row, and will override any column specification.

Do not use row specific commands to modify all rows if you want to set the whole area. Set the columns instead.

Cell-Specific Styles

Individual array elements, called cells, can be assigned a unique font, size and style.

This capability can be used to provide special formatting to design more attractive and useful reports.

You can use [PL_SetCellStyle](#) to set the font, size and style configuration for an individual cell, a range of cells, or a selection of discontinuous cells. You can choose to set all or just one of the style attributes of this command.

PrintList Pro will keep the cell and row-specific style settings with a row when the list is sorted.

If you do not want the cell and row style settings to move when the list is sorted, you should use the cell and row style routines after the call to [PL_SetSort](#).

Styled text

PrintList Pro supports the styled text feature of 4D v12 and above. When 4D passes styled text to PrintList Pro, it should be printed correctly if the attributed option has been set with [PL_SetFormat](#).

If this option is set, special tags can also be used in any text contained in a PrintList Pro area to print styled characters.

These tags work just like HTML tags: `<tag>styled text</tag>`. See [PrintList Pro Text Style Tags](#)

Colors

When using color or grayscale printers, many PrintList objects can be given color settings. Be sure to set the printer Color/Grayscale option to obtain the proper results.

Defining colors

There are three ways to define colors in PrintList Pro.

PrintList Pro's palette

PrintList Pro has its own palette, including the following colors:

Color	Constant
White	PL White
Black	PL Black
Magenta	PL Magenta
Red	PL Red
Cyan	PL Cyan
Green	PL Green
Blue	PL Blue
Yellow	PL Yellow
Gray	PL Gray
Light gray	PL Light gray

4D's palette

The 4D color palette is a 16 by 16 grid. To determine a color's value, you can locate the color's position on the color grid in the Design environment (the Color submenu which is available in the Form and Method editors), and count the number of rows down and columns across.

The equation is: **ColorValue = ((RowNumber - 1) x 16) + ColumnNumber.**

RGB colors

In addition, [PL_SetForeRGBColor](#) and [PL_SetBackRGBColor](#) can be used to perform similar settings with standard RGB values.

Column and Header Colors

Foreground and background colors can be specified for a PrintList Pro object using [PL_SetForeClr](#) and [PL_SetBackClr](#).

The foreground color can be specified for each column and column header, and the background color can be specified for the list and header areas.

Row-Specific Colors

[PL_SetRowColor](#) is used to set the foreground and background color of a specified row, and will override any column specification.

You can revert to the original column settings by setting the **plpRowForeColor** or **plpRowBackColor** parameter to the empty string (""), and the **4dRowForeColor** or **4dRowBackColor** parameter to -1.

[PL_SetRowRGBColor](#) can be used to perform similar settings with standard RGB values. Use this command to override all row-specific color settings by passing 0 for the **rowNumber** parameter.

Do not use row specific commands to modify all rows if you want to set the whole area. Set the columns instead.

Cell-Specific Colors

Individual array elements, called cells, can be assigned a unique foreground color and background color.

This capability can be used to set negative numbers in red, provide special formatting to show the current selected or enterable cell, and design more attractive and useful lists.

You can use [PL_SetCellColor](#) or [PL_SetCellRGBColor](#) to set the color configuration for an individual cell, a range of cells, or a selection of discontinuous cells.

PrintList Pro will keep the cell and row-specific color setting with a row when the list is sorted.

If you do not want the cell and row color settings to move when the list is sorted, be sure to call the cell and row color routines after the call to [PL_SetSort](#).

Multiple Lines in each Row

Multiple lines of text can be shown for each row using [PL_SetHeight](#). All rows will be printed with the number of lines specified, or with a variable height for each row.

PL_SetHeight can also be used to give each row additional space above and below the row's contents to give more spread out rows vertically.

Variable Height Rows

All rows can be printed with a varying height depending on the data that is to be printed. For rows, PrintList Pro will examine each row's text and picture element using the applied font and style settings to determine the tallest cell of each row.

Any given row can be of no height (i.e. no data) up to the height of an entire page. For any row that is larger than a page, PrintList Pro will attempt to show as much of it as possible by starting the row at the top of the page. The row will be truncated to a page — no one row can span two pages.

To set all rows to be variable height, use [PL_SetHeight](#) and set the **numRowLines** parameter to zero.

Setting an individual row or cell font size may cause PrintList Pro to override a fixed height row setting and print the row using a larger height.

In order to accommodate the larger font, PrintList Pro uses the variable height calculation to determine the height of the row based upon the font size setting.

When variable row height is used, Picture columns are used for row height calculation, too (even when **usePictHeight** in [PL_SetFormat](#) was set to zero).

Column Widths

Columns are automatically sized by default; however, a column size can be programmed using [PL_SetWidths](#)

All widths are given in pixels.

Dividing Lines, Frame and Header Separator Lines

Dividing lines can be added between rows and columns using [PL_SetDividers](#) or [PL_SetRGBDividers](#). The line width, pattern (transparency ratio), and color of the lines can be specified. The default is no dividing lines.

The PrintList Pro frame and header separator (the line between the headers and the list or detail area) lines can be set using [PL_SetFrame](#).

You (or the database end-user) must be sure to set the Color/Grayscale option in the print dialog when using colors.

Hairline Line Width

Lines can be printed a fraction of the line width seen on screen (1 pixel).

Typically, ¼ (.25) pixel produces the best results. All the lines that PrintList Pro prints may be given a fractional line width.

Double lines

Double lines (typical in accounting) are now supported in breaks: just use 2.0 as the **lineWidth** (5th parameter to [PL_SetBrkColOpt](#)/[PL_SetBrkColRGBOpt](#)/[PL_SetBkHColOpt](#)/[PL_SetBkHColRGBOpt](#)). Two 0.25 point lines will be printed.

Using Picture Arrays

PrintList Pro supports the printing of picture arrays. The **format** parameter of [PL_SetFormat](#) will cause the picture to be printed in one of five ways:

- truncated and justified to the upper left of the cell
- truncated and centered in the cell
- scaled to fit the cell
- scaled proportionally to fit the cell
- scaled proportionally to fit the cell and centered

The **usePictHeight** parameter of [PL_SetFormat](#) will tell PrintList Pro whether to use a picture's original height, which is stored with the picture, when calculating the row height for the PrintList Pro area.

If you choose not to use the picture's height in the row height calculation and additional space is needed to print the picture, the **numRowLines** parameter of [PL_SetHeight](#) should be used to increase the row height.

End of Page Callback Method

In 4D, a “callback” method is a project method called from an plug-in. PrintList Pro makes use of a callback method to inform you when the end of a printed page is reached.

This enables you to perform any necessary processing associated with the end of the page, for example, changing information printed in the footer area of that page or the header area of the next page.

Use [PL_SetPageProc](#) to specify the 4D method PrintList Pro is to call. PrintList Pro will pass the method specified by **callbackMethod** two parameters: the first indicates which PrintList Pro area is calling the method, and the second specifies the last row printed on that page.

Performance Issues with Formatting Commands

PrintList Pro uses an algorithm to automatically size the columns. Because of this, there is usually no need to use [PL_SetWidths](#) to manually size a column prior to printing a list.

However, if the number of items in the list is very large (several thousand items with many columns), then the list might take a few seconds longer to generate, due to the automatic sizing calculation.

If this is the case, using **PL_SetWidths** will improve the generation time of the list. Text arrays will take the longest to automatically size.

Since you can use **PL_SetWidths** on just some of the columns, if you are printing very large arrays, but only one is text, you could use **PL_SetWidths** on just the text array, and let PrintList Pro automatically calculate the other column widths.

[PL_SetFormat](#) does not affect the performance of PrintList Pro, regardless of the size of the arrays being printed.

Borders and Frames

[PL_SetCellBorder](#) provides the ability to set the border style for a cell.

[PL_SetCellFrame](#) prints a frame around a range of cells.

Both commands use RGB colors.

Header / Cell Icon Support

Picture Objects in Cells

[PL_SetCellIcon](#) uses **4D Picture Library items** to place icons into individual cells.

This routine includes an **iconRef** parameter, which is the reference number of a picture from the Design environment Picture Library. Pass zero (0) if you do not want any icon for the cell.

Picture Objects in Headers

In addition, **PL_SetCellIcon** provides the ability to procedurally place icons in column headers using 4D Picture Library objects.

Pass zero (0) in the **cellRow** parameter to set the header.



Configuration Commands

■ PL_Register

(registrationCode:T; options:L; email:T) → result

Parameter	Type	Description
→ registrationCode	text	Pass the registration key to register your copy of PrintList Pro. The key is either linked to the 4D or 4D Server serial number (individual licenses), to the machine ID (merged licenses), to the name of the company/developer (unlimited annual licenses) or to the product (master keys for Online instant activation).
→ options	longint	An optional longint combining up to 4 bits.
→ email	text	Online instant activation option: developer email to notify when a license is issued or resent.
← result	longint	0 or error code.

PL_Register is used to register the PrintList Pro plugin for standalone or server use.

Please see the [License Types](#) section for detailed information about the licensing options available for PrintList Pro.

Multiple calls to **PL_Register** are allowed. The plugin will be activated if at least one valid key is used, and all subsequent calls to **PL_Register** will return 0, unless the force check bit is set to true in the **options** parameter.

registrationCode — You must call **PL_Register** with a valid registration key, otherwise PrintList Pro will operate in demonstration mode - it will cease to function after 20 minutes. In case a [master key](#) is used the plugin will attempt a connection to e-Node's license server for [Online instant activation](#).

options — Optional. This parameter combines up to 4 bits as described below. The default mode (**registrationCode** being a passed as the only parameter) is silent: no force check, no confirmation, no alert, no email.

Bit number	Description
0	Force check: if this bit is is on (true), registrationCode is tested regardless of current registration state. If the plugin was not previously registered and the result is 0, it is registered the same way as if the bit was off (or the whole options parameter omitted) If the plugin was previously successfully registered, a registration error will be returned in result in case registrationCode is invalid, but the plugin will remain registered
1	Online instant activation option: confirm connection "Is it OK to connect to e-Node's license server to register PrintList Pro?"
2	Online instant activation option: display alert if registration error
3	Online instant activation option: display alert if registered

email — Optional. The developer [email address](#) where to send [Online instant activation](#) information.

result — 0 or error code:

Result code	Description
0	OK
1	Beta license has expired
2	Invalid license
3	The license has expired
4	The OEM license has expired
5	The maximum number of users has been exceeded
6	The license is for a different environment (e.g. the licence is for a single-user version, but you are using it with 4D Server)
7	The license is linked to a different 4D license
8	Invalid merged license
9	Only serial/ID licenses are allowed in text license files (includes Register button and Online instant activation)
10	Unauthorized master key (Online instant activation)
11	Can't connect to e-Node's license server to perform Online instant activation
12	No Online instant activation license available for this master key (unknown or all used)

When **PL_Register** is called with an empty string, the license dialog will be displayed if PrintList Pro is not registered and the dialog was not yet displayed. This allows you to show the registration dialog to your users without effectively calling a PrintList Pro command or displaying a PrintList Pro area.

Note: alternately to **PL_Register**, you can place a [plain text file](#) into your 4D Licenses folder or use the [Demo mode dialog "Register" button](#). This is only valid for non-unlimited licenses.

Basic example

```
C_LONGINT ($result)
```

```
$result:=PL_Register ("YourRegistrationKey")
```

Case of

```
:(($result=2)
```

```
  ALERT ("The PrintList Pro licence is invalid.")
```

```
:(($result=3)
```

```
  ALERT ("The PrintList Pro licence has expired.")
```

etc.

End case

Example with multiple calls

```
C_LONGINT ($result) //ignored in this case
```

```
$result:=PL_Register ("Registration key one")
```

```
$result:=PL_Register ("Registration key two")
```

```
$result:=PL_Register ("Registration key three")
```

etc.

```
if ($result#0) //registration failed on all keys
```

```
  ALERT ("PrintList Pro could not be registered.")
```

End if

Force check example

In this example we assume that only "Registration key two" is valid, but you want to check the other keys status.

```
C_LONGINT ($result)
```

```
$result:=PL_Register ("Registration key one";1) //invalid, will return an error, the plugin isn't registered
```

```
$result:=PL_Register ("Registration key two";1) //valid, will return 0, the plugin is registered
```

```
$result:=PL_Register ("Registration key three";1) //invalid, will return an error, the plugin is still registered
```

Online instant activation examples

Confirm connection, alert if successful, alert if failed, send email notification to developer@4dchampions.com:

```
C_LONGINT ($result)
```

```
$result:=PL_Register ("Master key";0 ?+1 ?+2 ?+3;"developer@4dchampions.com")
```

Silent connection, alert if successful, alert if failed, no email notification:

```
C_LONGINT ($result)
```

```
$result:=PL_Register ("Master key";0 ?+2 ?+3)
```

■ %PrintListPro

%PrintListPro is the command used to identify the PrintList Pro plugin area when you create a plugin area object on a layout.

This command is only used in the object definition for a PrintList Pro object, and should never be used as a command in a method.

■ PL_AddColumn

(areaRef:L; dataPointer:Z; insertAt:L) → result:L

Parameter	Type	Description
→ areaRef	longint	PrintList Pro area reference.
→ dataPointer	pointer	Pointer to an array or a field.
→ insertAt	longint	Position where to insert the column.
← result	longint	0 if successful.

PL_AddColumn adds a column at the specified position.

areaRef — PrintList Pro area reference.

dataPointer — This parameter specifies the data to print in the inserted column.

- when in Records mode, this parameter must be a pointer to a field
- when in Arrays mode, this parameter must be a pointer to an array (not a local array!)

insertAt — Position where to insert the column. Zero means “append to the end”.

This command supports the component architecture (using arrays from the host database in a component and vice versa).

This command can be used as an alternative to [PL_SetArraysNam](#), but requires one line per array. [See the Calculated columns Array mode example.](#)

Examples

```
$error:= PL_AddColumn(eList;->aState;0) //add a column containing aState array at the end
```

```
$error:= PL_AddColumn(eList;->aCity;1) //insert a column containing aCity array at position 1
```

■ PL_SetArraysNam

(areaRef:L; columnNumber:L; numArrays:L; array1:T; ...; arrayN:T) → result:L

Parameter	Type	Description
→ areaRef	longint	PrintList Pro area reference.
→ columnNumber	longint	Column at which to set the first array.
→ numArrays	longint	Number of arrays to set (up to 15).
→ array1; ...; arrayN	text	Name(s) of 4D array(s).
← result	longint	0 if successful.

PL_SetArraysNam tells PrintList Pro what arrays to print. Up to fifteen arrays can be set at a time. Any 4D array type can be used except pointer and two-dimensional arrays.

Since PrintList Pro can print up to **32767** arrays, this command may have to be used more than once.

There are three very important points to note about this command:

- This command must be called first, before any of the other commands, in the [On Printing Detail](#) phase.
- The columns must be added in sequential order, unless the particular column has already been added. In other words, to set 30 arrays, you must set arrays 1 through 15 prior to setting arrays 16 through 30.

- All arrays set with this command must have the same number of elements as each other and as any other arrays previously set.

You can pass process arrays and interprocess arrays to PrintList Pro, but not local arrays (a local array has a name that starts with a "\$" character; an interprocess array has a name that starts with "<>" characters).

One dimension of a two-dimensional array may be passed in the array1; ...; arrayN parameters. For example: "my2DArray{1}" may be passed as array1.

areaRef — PrintList Pro area reference.

columnNumber — This parameter specifies the column number to set the first array being passed by this call of **PL_SetArraysNam**.

numArrays — This parameter specifies the number of columns being set with this call to **PL_SetArraysNam**.

Examples

Case of

```
:(Form event=On Printing Detail)
  SELECTION TO ARRAY([Contacts]FN;aFN:[Contacts]LN;aLN:[Contacts]City;aCity;\
    [Contacts]State;aState) //load the arrays
  $error:=PL_SetArraysNam (eNameList;1;4;"aFN";"aLN";"aCity";"aState") //starting at column 1,
    set 4 arrays to print through the plugin area eNameList
```

End case

```
// Set up the eList PrintList Pro object with 25 arrays
// two calls must be made since only 15 arrays can be passed each time
$error:=PL_SetArraysNam (eList;1;15;"array1";"array2";"array3";"array4";"array5";
  "array6";"array7";"array8";"array9";"array10";"array11";"array12";"array13";"array14";"array15")
$error:=PL_SetArraysNam (eList;16;10;"array16";"array17";"array18";"array19";"array20";
  "array21";"array22";"array23";"array24";"array25")
```

■ PL_SetHeaders

(areaRef:L; columnNumber:L; numHeaders:L; header1:T; ...; headerN:T)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column at which to set up the first header.
→ numHeader	longint	Number of headers to set (up to 15).
→ header1; ...; headerN	text	Value(s) to print in column header(s).

PL_SetHeaders is used to specify the value to print in the header for each column. Up to fifteen headers can be set at a time.

The size of the header value is used by the automatic column sizing algorithm. If you are printing a text array containing 2 character strings, the column will be very narrow, unless you specify a header which contains several characters.

For example, states are usually stored in a database as a two-character string. But if you specify a header of "State" the column will be sized about two and a half times wider.

If the header length is less than the values being printed in the column, then the header length will not affect the column width.

A, B, C, etc. will be printed in the headers if **PL_SetHeaders** is not used.

Examples

```
$error:=PL_SetArraysNam (eNameList;1;4;"aFN";"aLN";"aCity";"aState")
```

```
PL_SetHeaders(eNameList;1;4;"First Name";"Last Name";"City";"State")
```

```
$error:=PL_SetArraysNam (eNames;1;2;"aFN";"aLN")
```

```
PL_SetHeaders(eNames;1;2;Field name([People]FirstName);Field name([People]LastName))
```

■ PL_SetFormat

(areaRef:L; columnNumber:L; format:T; columnJust:L; headerJust:L; usePictureHeight:L; attributed:L; lineSpacing:F; vertAlignment:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column at which to set the format and justification.
→ format	text	Format to use.
→ columnJust	longint	Justification for column list items.
→ headerJust	longint	Justification for column header.
→ usePictHeight	longint	Use the picture height in the row height calculation.
→ attributed	longint	Use attributed (multi-style) text: 0 = no, 1 = yes.
→ lineSpacing	real	Line spacing to use.
→ vertAlignment	longint	Vertical alignment: 0 = default, 1 = top, 2 = center, 3 = bottom.

PL_SetFormat is used to control the **format** and justification of a column being printed. You can control the format of integer, real, date, boolean, and picture columns with the format parameter. Time values can be formatted also, since they use long integer arrays.

Any valid 4D format, including custom formats created in the Design environment, may be used with these column types.

Text columns can only be formatted using styled text with the **attributed** parameter set to 1.

Additionally, null time and date values can be set to print a blank by appending a dash character ("-") to the **format** text parameter.

The defaults for the different column types are:

Column Type	Format
(Long) Integer	"#,###,##0"
Real	"#,###,##0.00"
Boolean	"True;False"
Date	"0"
Picture	"0"

Note: since version 5.3, unused characters are stripped from the format instead of replacing "number sign" placeholders by non-breaking space (e.g. "1234" formatted with "###-###-###" will produce "1-234", not " - 1-234" as before).

format (for text arrays) — Not supported, **except for attributed text according to the value set by the attributed parameter.**

format (for numeric arrays) — See the 4D command **String** in the 4D Language Reference for the possible values. Any valid 4D numeric format may be used.

format (for boolean arrays) — The string contains two formats, one for the **True** value, the other for the **False** value, separated by a semicolon. Examples: "Male;Female" and "MacOS;Windows".

format (for date arrays) — See the 4D command **String** in the 4D Language Reference for the possible values. Any valid 4D date format may be used. Examples: "0" or "3" are valid formats.

Format	Example
0	09/21/16 (default)
1	9/21/16
2	Wed, Sep 21, 2016
3	Wednesday, September 21, 2016
4	09/21/16 or 09/21/1996
5	September 21, 2016
6	Sep 21, 2016

format (for "time" arrays) — See the 4D **String** command in the 4D Language Reference, and the 4D Design Reference discussion of formatting for the possible values. There are no time arrays in 4D as such, they are in reality long integer arrays. These arrays are printed as time PL_SetFormat values by using the proper format. The **format** is the two character sequence "&/" followed by the number given in the discussion of the **String** command. For example, one proper format for a time array would be "&/2".

Format	Example
0	01:02:03
1	01:02
2	1 hour 2 minutes 3 seconds
3	1 hour 2 minutes
4	1:02 AM

format (for picture arrays):

Format	Description
0	The picture will be truncated, if necessary, and justified to the upper left (default)
1	The picture will be truncated, if necessary, and centered in the cell
2	The picture will be scaled to fit the cell
3	The picture will be scaled to fit the cell, and remain proportional to its original size
4	The picture will be scaled to fit the cell, remain proportional to its original size, and centered in the cell

In addition:

- If **format** is not specified or out of range, the value will be interpreted as 0
- Formats 1 and 4 are always centered, format 2 fills the whole rectangle
- Only formats 0 and 3 will use the specified **columnJust** (default alignment is centered for these two formats)

columnJust and **headerJust** — The justification for a column and its header can be controlled independently.

The possible values are:

Value	Justification
0	Default
1	Left
2	Center
3	Right

By default, headers are left justified, unless the column elements are center justified. In that case, the header will default to center justification.

The default column justifications for the different column types are:

Column type	Default column justification
Long Integer (including Time)	Right
Real	Right
Boolean	Left
Date	Right
Text	Left
Picture	Depending on the format parameter

The **columnJust** parameter is **only used for picture columns where the **format** parameter is set to 0 or 3 (or not specified or out of range). Other values will use the **format** parameter to justify picture columns.**

usePictHeight:

Value	Mode
0	Ignore the picture height when calculating the row height (default)
1	Use height of the largest picture when calculating the row height

If the column **columnNumber** does not have a picture column, this parameter will be ignored.

If the list is configured to automatically calculate variable height rows, then picture array elements are always included in the automatic calculation, and this parameter is ignored. See [PL_SetHeight](#) and [Variable Height Rows](#) for more information.

attributed — **Styled text:**

Value	Mode
0	Plain text (default)
1	(or any non-zero value) — attributed (styled) text

Note: the styled text property is applied to all formatting in that column... breaks have to account for that!

lineSpacing — Line spacing to use (real value). 0 is substituted by 1.0 (default), which is also the default line spacing value used by [AreaList Pro](#) and [SuperReport Pro](#).

Line spacing is used for space between lines.

It is computed from the font height as (Ascent + Descent) * **lineSpacing**

In other words it is a percentage of the font height to use for advancing the text to the next line.

When it is 1.0 (default), next line starts right below the previous one.

For example, if you enter 2 as **lineSpacing** you will get (in the same cell):

```
Line 1
Line 2
Line 3
```

Instead of:

```
Line 1
Line 2
Line 3
```

Examples

//Format a real column (3rd column), default column justification, center header justification

```
PL_SetFormat (names;3;"###.###.00";0;2;0)
```

//Format a boolean column (4th column), right column justification and left header justification

```
PL_SetFormat (eList;4;"Male;Female";3;1;0)
```

//Format style 3 for a date column, default justification (5th column), default column and header justification, suppress null dates

```
PL_SetFormat (eNames;5;"3-")
```

//Format style 2 for a time column, right justification for header and column (7th column)

```
PL_SetFormat (eList;7;"&/2";3;3;0)
```

//Custom format style, default justification for column, center header (5thcolumn), attributed text, "compatible" line spacing

```
PL_SetFormat (eNames;5;"|Dollars";0;2;0;1;1)
```

//Scale picture column to fit proportionally (1st column), use default header justification, use picture size in row height calculation

```
PL_SetFormat (eList;1;"3";0;0;1)
```

vertAlignment — Vertical alignment:

Format	Description
0	Default
1	Top
2	Center
3	Bottom

■ PL_SetWidths

(areaRef:L; columnNumber:L; numWidths:L; width1:L; ...; widthN:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column at which to set the first width.
→ numWidths	longint	Number of widths to set (up to 15).
→ width1; ...; widthN	longint	Pixel width(s) of column(s).

PL_SetWidths is used to set the pixel width for one or more columns. Up to fifteen widths can be set at a time.

A width of zero forces a column to be sized automatically based on its data type.

A column cannot be less than 3 pixels wide. If you pass a value of less than 3 but greater than zero, PrintList Pro will ignore it and use 3.

PrintList Pro will not let a column be wider than the width of the list area minus 20.

If not called, the default width for all columns is determined based on the type of array or field printed in the column.

Example

```
$error:=PL_SetArraysNam (eNames;1;5;"aFN";"aLN";"aCity";"aState";"aZip")
PL_SetWidths(eNames;1;5;150;50;0;100;0) //0 forces autosizing for that column
```

■ PL_SetHdrStyle

(areaRef:L; columnNumber:L; fontName:T; size:L; styleNum:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column for which to set the header style.
→ fontName	text	Name of the font to use.
→ size	longint	Size of the font.
→ stylenum	longint	Style of the font.

PL_SetHdrStyle is used to control the appearance of the PrintList Pro column headers.

The columns can be controlled individually or as a group.

columnNumber — This parameter specifies what column header to apply the style to. Use a value of zero (0) to apply the parameters to all columns.

fontName — Use this parameter to specify the font for the specified **columnNumber**. If not called, or the specified font name is not found, the header(s) will be printed in the OS defined System Font. If the font is not installed, then the System Font will be used.

styleNum — This parameter is a font style code. Use the [Style constants](#), which can be combined.

Note: if bold or italic styles are set, but not installed for the font, PrintList Pro will print regular (plain) characters.

Examples

```
PL_SetHdrStyle(eList;1;"Geneva";12;1) // Geneva 12 point bold, column 1
PL_SetHdrStyle(Names;3;"New York";12;3) // New York 12 point bold italic, column 3
PL_SetHdrStyle(Names;0;"Palatino";10;3) // Palatino 10 point bold italic, all columns
```

■ PL_SetHdrOpts

(areaRef:L; printHeaders:L; printPixelWidth:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ printHeaders	longint	Print the headers above the list.
→ printPixelWidth	longint	Print column widths in the header.

PL_SetHdrOpts is used to control several PrintList Pro options pertaining to column headers.

printHeaders:

Value	Mode
0	No headers will be printed (default)
1	Headers will be printed only on the first page
2	Headers will be printed on all pages

printPixelWidth — Used during development to allow you to easily determine what pixel width looks best for each column:

Value	Mode
0	The normal header text will be printed (default)
1	The width of the column will be printed in each header

Examples

```
PL_SetHdrOpts(eList;2;0) // print headers on all pages, no pixel widths
PL_SetHdrOpts(eList;1;1) // print headers on first page, and pixel widths
```

■ PL_SetMiscOptions

(areaRef:L; escapeChar:T; useEllipsis:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ escapeChar	text	Escape character (obsolete).
→ useEllipsis	longint	Use ellipsis.

PL_SetMiscOptions is used to control miscellaneous PrintList Pro options.

escapeChar — **Obsolete, ignored.**

useEllipsis — Determines if auto-ellipsis is used when columns are smaller than the printed data:

Value	Mode
0	Use ellipsis in header and column data
1	Don't use ellipsis in header and column data (default)

■ PL_SetStyle

(areaRef:L; columnNumber:L; fontName:T; size:L; styleNum:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column for which to set the style.
→ fontName	text	Name of the font to use.
→ size	longint	Size of the font.
→ styleNum	longint	Style of the font.

PL_SetStyle is used to control the appearance of the PrintList Pro columns. The columns can be controlled individually or as a group.

columnNumber — This parameter specifies what column to apply the style to. Use a value of zero (0) to apply the parameters to all columns.

fontName — Use this parameter to specify the font for the specified **columnNumber**. If not called, or the specified font name is not found, the column(s) will be printed in the OS defined System Font. If the font is not installed, then the System Font will be used.

styleNum — This parameter is a font style code. Use the [Style constants](#), which can be combined.

Note: if bold or italic styles are set, but not installed for the font, PrintList Pro will print regular (plain) characters.

Examples

```
PL_SetStyle(eNames;0;"Geneva";9;0) // Geneva 9 plain, all columns
```

```
PL_SetStyle(eList;4;"Helvetica";12;32) // Helvetica 12 point condensed, 4th column
```

```
PL_SetStyle(eNames;1;"Times";9;1) // Times 9 point bold, 1st column
```

■ PL_SetForeClr

(areaRef:L; columnNumber:L; plpHdrForeColor:T; 4dHdrForeColor:L; plpListForeColor:T; 4dListForeColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column number.
→ plpHdrForeColor	text	Header foreground color from PrintList Pro's palette.
→ 4dHdrForeColor	longint	Header foreground color from 4D's palette.
→ plpListForeColor	text	List foreground color from PrintList Pro's palette.
→ 4dListForeColor	longint	List foreground color from 4D's palette.

PL_SetForeClr is used to specify the foreground colors for a column header and a list area column.

columnNumber — The column for which to set the foreground color. Use a value of zero (0) to apply the parameters to all columns.

plpHdrForeColor — Name of the color in [PrintList Pro's palette](#). This will be the foreground color for the column header. If the name is not in PrintList Pro's palette or it is a null string, then **4dHdrForeColor** will be used.

4dHdrForeColor — 1 to 256. The color at this position in [4D's palette](#) will be used for the foreground color for the column header.

plpListForeColor — Name of the color in PrintList Pro's palette. This will be the foreground color for the column. If the name is not in PrintList Pro's palette or it is a null string, then **4dListForeColor** will be used.

4dListForeColor — 1 to 256. The color at this position in 4D's palette will be used for the foreground color for the column.

If **PL_SetForeClr** is not called, the default is black for both the header and list foreground colors.

Examples

```
// Red for column header foreground, light gray for column foreground (all columns)
```

```
PL_SetForeClr(eNames;0;"Red";0;"Light Gray";0)
```

```
// Green for column header foreground, 13th color from 4D's palette for column foreground (4th column)
```

```
PL_SetForeClr (eNames;4;"Green";0;"";13)
```

■ PL_SetForeRGBColor

(areaRef:L; columnNumber:L; hdrForeRed:L; hdrForeGreen:L; hdrForeBlue:L; listForeRed:L; listForeGreen:L; listForeBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column number.
→ hdrForeRed	longint	Header fore red.
→ hdrForeGreen	longint	Header fore green.
→ hdrForeBlue	longint	Header fore blue.
→ listForeRed	longint	List fore red.
→ listForeGreen	longint	List fore green.
→ listForeBlue	longint	List fore blue.

PL_SetForeRGBColor is used to specify the foreground colors for a column header and a list area column, using the RGB values. This routine is similar to [PL_SetForeClr](#).

hdrForeRed — Header foreground RGB red value.

hdrForeGreen — Header foreground RGB green value.

hdrForeBlue — Header foreground RGB blue value.

listForeRed — List foreground RGB red value.

listForeGreen — List foreground RGB green value.

listForeBlue — List foreground RGB blue value.

Example

The following example will tell PrintList Pro to print the third column using a color scheme standard for MacOSX:

```
PL_SetForeRGBColor (xArea;3;237;254;243;237;254;243)
```

■ PL_SetBackClr

(areaRef:L; plpHdrBackColor:T; 4dHdrBackColor:L; plpListBackColor:T; 4dListBackColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ plpHdrBackColor	text	Header background color from PrintList Pro's palette.
→ 4dHdrBackColor	longint	Header background color from 4D's palette.
→ plpListBackColor	text	List background color from PrintList Pro's palette.
→ 4dListBackColor	longint	List background color from 4D's palette.

PL_SetBackClr is used to specify the background colors for the header and list area.

While the foreground color can be specified for each column, the background color for the header or the list area can only be specified for all columns using this command. You need to use [PL_SetColBackColor](#) or [PL_SetColBackRGBColor](#) to set the background colors of each column's header and each column itself.

plpHdrBackColor — Name of the color in [PrintList Pro's palette](#). This will be the background color for the column header. If the name is not in PrintList Pro's palette or it is a null string, then **4dHdrBackColor** will be used.

4dHdrBackColor — 1 to 256. The color at this position in [4D's palette](#) will be used for the background color for the column header.

plpListBackColor — Name of the color in PrintList Pro's palette. This will be the background color for the column. If the name is not in PrintList Pro's palette or it is a null string, then **4dListBackColor** will be used.

4dListBackColor — 1 to 256. The color at this position in 4D's palette will be used for the background color for the column.

If **PL_SetBackClr** is not called, the default is white for both the header and list background colors.

Examples

```
//Light gray for header background, white for list background, all columns
```

```
PL_SetBackClr(eNames;0;"Light Gray";0;"White";0)
```

```
//White for header background, 13th color from 4D's palette for list background, 1st column
```

```
PL_SetBackClr (eNames;1;"White";0;"";13)
```

■ PL_SetBackRGBColor

(areaRef:L; hdrBackRed:L; hdrBackGreen:L; hdrBackBlue:L; listBackRed:L; listBackGreen:L; listBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ hdrBackRed	longint	Header back red.
→ hdrBackGreen	longint	Header back green.
→ hdrBackBlue	longint	Header back blue.
→ listBackRed	longint	List back red.
→ listBackGreen	longint	List back green.
→ listBackBlue	longint	List back blue.

PL_SetBackRGBColor is used to specify the background colors for the header and list area, using the RGB values. This routine is similar to [PL_SetBackClr](#).

While the foreground color can be specified for each column, the background color for the header or the list area can only be specified for all columns using this command. You need to use [PL_SetColBackColor](#) or [PL_SetColBackRGBColor](#) to set the background colors of each column's header and each column itself.

hdrBackRed — Header background RGB red value.

hdrBackGreen — Header background RGB green value.

hdrBackBlue — Header background RGB blue value.

listBackRed — List background RGB red value.

listBackGreen — List background RGB green value.

listBackBlue — List background RGB blue value.

Example

The following example will tell PrintList Pro to print the list using a color scheme standard for MacOS X:

```
PL_SetBackRGBColor (xArea;237;254;243;237;254;243)
```

■ PL_SetColBackColor

(areaRef:L; columnNumber:L; plpHdrBackColor:T; 4dHdrBackColor:L; plpListBackColor:T; 4dListBackColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column number.
→ plpHdrBackColor	text	Header background color from PrintList Pro's palette.
→ 4dHdrBackColor	longint	Header background color from 4D's palette.
→ plpListBackColor	text	List background color from PrintList Pro's palette.
→ 4dListBackColor	longint	List background color from 4D's palette.

PL_SetColBackColor is used to specify the background colors for a column header and a list area column.

columnNumber — The column for which to set the background color. Use a value of zero (0) to apply the parameters to all columns.

plpHdrBackColor — Name of the color in [PrintList Pro's palette](#). This will be the background color for the column header. If the name is not in PrintList Pro's palette or it is a null string, then **4dHdrBackColor** will be used.

4dHdrBackColor — 1 to 256. The color at this position in [4D's palette](#) will be used for the background color for the column header.

plpListBackColor — Name of the color in PrintList Pro's palette. This will be the background color for the column. If the name is not in PrintList Pro's palette or it is a null string, then **4dListBackColor** will be used.

4dListBackColor — 1 to 256. The color at this position in 4D's palette will be used for the background color for the column.

If **PL_SetColBackColor** is not called, the default is white for both the header and list background colors.

Examples

```
//Light gray for header background, white for list background, all columns
```

```
PL_SetColBackColor(eNames;0;PL Light gray;0;PL White;0)
```

```
//White for header background, 13th color from 4D's palette for list background, 1st column
```

```
PL_SetColBackColor(eNames;1;PL White;0;"";13)
```


■ PL_SetColBackRGBColor

(areaRef:L; columnNumber:L; hdrBackRed:L; hdrBackGreen:L; hdrBackBlue:L; listBackRed:L; listBackGreen:L; listBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column number.
→ hdrBackRed	longint	Header back red.
→ hdrBackGreen	longint	Header back green.
→ hdrBackBlue	longint	Header back blue.
→ listBackRed	longint	List back red.
→ listBackGreen	longint	List back green.
→ listBackBlue	longint	List back blue.

PL_SetColBackRGBColor is used to specify the background colors for a column header and a list area column, using the RGB values. This routine is similar to [PL_SetColBackColor](#).

columnNumber — The column for which to set the background color. Use a value of zero (0) to apply the parameters to all columns.

hdrBackRed — Header background RGB red value.

hdrBackGreen — Header background RGB green value.

hdrBackBlue — Header background RGB blue value.

listBackRed — List background RGB red value.

listBackGreen — List background RGB green value.

listBackBlue — List background RGB blue value.

Example

The following example will tell PrintList Pro to print the third column using a color scheme standard for OSX:

```
PL_SetColBackRGBColor (xArea;3;237;254;243;237;254;243)
```

■ PL_SetRowStyle

(areaRef:L; rowNumber:L; styleNum:L; fontName:T; fontSize:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ rowNumber	longint	Number of row.
→ styleNum	longint	Style of the font.
→ fontName	text	Name of the font.
→ fontSize	longint	Size of the font.

PL_SetRowStyle is used to set the style and font for a particular row. It will override the style and font settings for all columns in that row. The size settings of each column will still apply.

Any subsequent sorting using [PL_SetSort](#) will cause the row style setting to be moved with the arrays. This will keep the style setting “in sync” with the original row.

Keep in mind that any settings applied to a row will be moved with that row’s data if the data is later sorted using **PL_SetSort**. If you do not want the row’s settings to move, call **PL_SetSort** before applying the row settings.

rowNumber — The row for which to set the style. Use a value of zero (0) to apply the parameters to all rows.

styleNum — This parameter is used to set the style for the row. Use the [Style constants](#), which can be combined.

Note: if bold or italic styles are set, but not installed for the font, PrintList Pro will print regular (plain) characters.

If a row style has been previously set, it may be removed by setting **styleNum** to -1. This may also be applied to all rows by passing a zero (0) for the **rowNumber**. This will have no effect on rows that have not been previously set.

The row style may be left unchanged by setting **styleNum** to 256.

fontName — This parameter specifies the font for a row. The row font may be left unchanged by setting **fontName** to the empty string (“”). If the font specified is not found, it will be treated as an empty string and ignored.

fontSize — This specifies the font size for a row. The row font size may be left unchanged by setting **fontSize** to 0.

Examples

```
PL_SetRowStyle(eNames;10;2;"";0) //set row 10 to be italic - no change in font size
```

```
PL_SetRowStyle(eNames;0;1;"Helvetica";14) //set all rows to be bold, Helvetica 14
```

```
PL_SetRowStyle(eList;12;3;"Times";0) //set the 12th row to print the Times font in bold italic style
```

■ PL_SetRowColor

(areaRef:L; rowNumber:L; plpRowForeColor:T; 4dRowForeColor:L; plpRowBackColor:T; 4dRowBackColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ rowNumber	longint	Number of row.
→ plpRowForeColor	text	Row foreground color from PrintList Pro's palette.
→ 4dRowForeColor	longint	Row foreground color from 4D's palette.
→ plpRowBackColor	text	Row background color from PrintList Pro's palette.
→ 4dRowBackColor	longint	Row background color from 4D's palette.

PL_SetRowColor is used to specify the foreground and background colors for a row. It will override the foreground and background color settings for all columns in that row.

Any subsequent sorting using [PL_SetSort](#) will cause the row color setting to be moved with the arrays. This will keep the color setting “in sync” with the original row.

Keep in mind that any settings applied to a row will be moved with that row's data if the data is later sorted using **PL_SetSort**. If you do not want the row's settings to move, call **PL_SetSort** before applying the row settings.

rowNumber — The row for which to set the foreground color. Use a value of zero (0) to apply the parameters to all rows.

plpRowForeColor — Name of the color in [PrintList Pro's palette](#). This will be the foreground color for the row. If the name is not in PrintList Pro's palette or it is a null string, then **4dRowForeColor** will be used.

4dRowForeColor — 1 to 256. Foreground color number for the row (from [4D's palette](#)). The row foreground color may be left unchanged by setting **plpRowForeColor** to the empty string (""), and **4dRowForeColor** to 0.

If a row color has been previously set, it may be removed by setting **plpRowForeColor** to an empty string (""), and **4dRowForeColor** to -1. This may also be applied to all rows by passing a zero (0) for the **rowNumber**. This will have no effect on rows that have not been previously set.

plpRowBackColor — Name of the color in PrintList Pro's palette. This will be the background color for the row. If the name is not in PrintList Pro's palette or it is the empty string (""), then **4dRowBackColor** will be used.

4dRowBackColor — 1 to 256. Background color number for the row (from 4D's palette). The row background color may be left unchanged by setting **plpRowBackColor** to the empty string (""), and **4dRowBackColor** to 0.

If a row background color has been previously set, it may be removed by setting **plpRowBackColor** to the empty string (""), and **4dRowBackColor** to -1. This may also be applied to all rows by passing a zero (0) for the **rowNumber**. This will have no effect on rows that have not been previously set.

Examples

```
PL_SetRowColor(eNames;10;"Blue";0;"Light gray";0) //set row 10 to foreground blue, background light gray
```

```
PL_SetRowColor(eNames;0;"Blue";0;"Yellow";0) //set all rows to blue foreground, yellow background
```

```
PL_SetRowColor(eNames;0;"";-1;"";-1) //reset all row colors to use the column color settings
```

```
PL_SetRowColor(eList;10;"Blue";0;"Light Gray";0)
```

```
//set the 10th row to print a foreground color of blue and background color of light gray
```

```
PL_SetRowColor(eList;12;"Green";0;"";0)
```

```
//set the 12th row to print a foreground color of green and the current background color
```

■ PL_SetRowRGBColor

(areaRef:L; rowNumber:L; rowForeRed:L; rowForeGreen:L; rowForeBlue:L; rowBackRed:L; rowBackGreen:L; rowBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ rowNumber	longint	Row number.
→ rowForeRed	longint	Foreground red.
→ rowForeGreen	longint	Foreground green.
→ rowForeBlue	longint	Foreground blue.
→ rowBackRed	longint	Background red.
→ rowBackGreen	longint	Background green.
→ rowBackBlue	longint	Background blue.

PL_SetRowRGBColor provides the ability to set the foreground and background colors for an individual row using standard RGB colors.

This routine is similar to [PL_SetRowColor](#), except that it uses RGB color values.

rowForeRed — Foreground RGB red value.

rowForeGreen — Foreground RGB green value.

rowForeBlue — Foreground RGB blue value.

rowBackRed — Background RGB red value.

rowBackGreen — Background RGB green value.

rowBackBlue — Background RGB blue value.

Example

The following example will tell PrintList Pro to print the third row using a color scheme standard for MacOS X:

```
PL_SetRowRGBColor (xArea;3;237;0;243;0;254;0)
```

■ PL_SetDividers

(areaRef:L; colDividerWidth:F; colDividerPattern:T; plpColDividerColor:T; 4dColDividerColor:L; rowDividerWidth:F; rowDividerPattern:T; plpRowDividerColor:T; 4dRowDividerColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ colDividerWidth	real	Width of the column divider line.
→ colDividerPattern	text	Pattern of the column divider.
→ plpColDividerColor	text	Color from PrintList Pro's palette for the column divider.
→ 4dColDividerColor	longint	Color from 4D's palette for the column divider.
→ rowDividerWidth	real	Width of the row divider line.
→ rowDividerPattern	text	Pattern of the row divider.
→ plpRowDividerColor	text	Color from PrintList Pro's palette for the row divider.
→ 4dRowDividerColor	longint	Color from 4D's palette for the row divider.

PL_SetDividers is used to set the pattern ([transparency ratio](#)) and color of the column and row dividers.

See the [Patterns](#) item in the [Compatibility Notes](#).

colDividerWidth — 0 to 1. This option controls the line width of the column dividers. A value of 0.25 pixel should be used for hairlines. A value of 0 means that no dividers will be printed.

colDividerPattern — Name of the pattern ([transparency ratio](#)) for the column divider. If a null string is used then no column divider will be printed.

plpColDividerColor — Name of the color in [PrintList Pro's palette](#). This will be the color for the column divider. If the name is not in PrintList Pro's palette or it is a null string, then **4dColDividerColor** will be used.

4dColDividerColor — 1 to 256. The color at this position in [4D's palette](#) will be used for the column divider.

rowDividerWidth — 0 to 1. This option controls the line width of the row dividers. A value of 0.25 pixel should be used for hairlines. A value of 0 means that no dividers will be printed.

rowDividerPattern — Name of the pattern ([transparency ratio](#)) for the row divider. If a null string is used then no row divider will be printed.

plpRowDividerColor — Name of the color in PrintList Pro's palette for the row divider. If the name is not in PrintList Pro's palette or it is a null string, then **4dRowDividerColor** will be used.

4dRowDividerColor — 1 to 256. The color at this position in 4D's palette will be used for the row divider.

If neither **PL_SetDividers** nor [PL_SetRGBDividers](#) are called, then no column or row dividers will be printed.

Examples

```
//Print solid gray column dividers and no row dividers
```

```
PL_SetDividers (eNames;1;"Black";"Gray";0;0;"";"";0)
```

```
//Print column and row hairline dividers in a gray pattern
```

```
PL_SetDividers (eNames;0.25;"Gray";"Black";0;0.25;"Gray";"Black";0)
```

■ PL_SetRGBDividers

(areaRef:L; colDividerWidth:F; colDividerPattern:T; colDividerRed:L; colDividerGreen:L; colDividerBlue:L; rowDividerWidth:F; rowDividerPattern:T; rowDividerRed:L; rowDividerGreen:L; rowDividerBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ colDividerWidth	real	Width of the column divider line.
→ colDividerPattern	text	Column divider pattern string.
→ colDividerRed	longint	Column divider — Red.
→ colDividerGreen	longint	Column divider — Green.
→ colDividerBlue	longint	Column divider — Blue.
→ rowDividerWidth	real	Width of the row divider line.
→ rowDividerPattern	text	Row divider pattern string.
→ rowDividerRed	longint	Row divider — Red.
→ rowDividerGreen	longint	Row divider — Green.
→ rowDividerBlue	longint	Row divider — Blue.

PL_SetRGBDividers functions the same as the [PL_SetDividers](#) routine, except that the column and row divider colors use standard RGB values.

colDividerWidth — 0 to 1. This option controls the line width of the column dividers. A value of 0.25 pixel should be used for hairlines. A value of 0 means that no dividers will be printed.

colDividerPattern — Text, name of the pattern (**transparency ratio**) for the column divider. If a null string is used then no column divider will be printed.

colDividerRed — Column divider RGB red value.

colDividerGreen — Column divider RGB green value.

colDividerBlue — Column divider RGB blue value.

rowDividerWidth — 0 to 1. This option controls the line width of the row dividers. A value of 0.25 pixel should be used for hairlines. A value of 0 means that no dividers will be printed.

rowDividerPattern — Text, name of the pattern (**transparency ratio**) for the row divider. If a null string is used then no row divider will be printed.

rowDividerRed — Row divider RGB red value.

rowDividerGreen — Row divider RGB green value.

rowDividerBlue — Row divider RGB blue value.

If neither [PL_SetDividers](#) nor **PL_SetRGBDividers** are called, then no column or row dividers will be printed.

Example

The following example will set the column/row dividers using the **PL_SetRGBDividers** routine:

```
//Print column and row dividers in a hairline gray pattern
PL_SetRGBDividers(eNames;0.25;"Gray";209; 209; 209;0.25;"Gray"; 209; 209; 209)
```

■ PL_SetFrame

(areaRef:L; frameLineWidth:F; frameLinePattern:T; plpFrameLineColor:T; 4dFrameLineColor:L; headerLineWidth:F; headerLinePattern:T; plpHeaderLineColor:T; 4dHeaderLineColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ frameLineWidth	real	Width of the frame line.
→ frameLinePattern	text	Pattern of the frame line.
→ plpFrameLineColor	text	Color from PrintList Pro's palette for the frame line.
→ 4dFrameLineColor	longint	Color from 4D's palette for the frame line.
→ headerLineWidth	real	Width of the header separator.
→ headerLinePattern	text	Pattern of the header separator.
→ plpHeaderLineColor	text	Color from PrintList Pro's palette for the header separator.
→ 4dHeaderLineColor	longint	Color from 4D's palette for the header separator.

PL_SetFrame is used to set the pattern ([transparency ratio](#)) and color of the frame and header separator lines.

See the [Patterns](#) item in the [Compatibility Notes](#).

frameLineWidth — 0 to 1. This option controls the line width of the frame. A value of 0.25 pixel should be used for hairlines. A value of 0 means that no frame will be printed.

frameLinePattern — Name of the pattern ([transparency ratio](#)) for the frame. If a null string is used then no frame will be printed.

plpFrameLineColor — Name of the color in [PrintList Pro's palette](#). This will be the color for the frame. If the name is not in PrintList Pro's palette or it is a null string, then **4dFrameLineColor** will be used.

4dFrameLineColor — 1 to 256. The color at this position in [4D's palette](#) will be used for the frame.

headerLineWidth — 0 to 1. This option controls the line width of the header separator. A value of 0.25 pixel should be used for hairlines. A value of 0 means that no header separator will be printed.

headerLinePattern — Name of the pattern ([transparency ratio](#)) for the header separator. If a null string is used then no header separator will be printed.

plpHeaderLineColor — Name of the color in PrintList Pro's palette for the header separator. If the name is not in PrintList Pro's palette or it is a null string, then **4dHeaderLineColor** will be used.

4dHeaderLineColor — 1 to 256. The color at this position in 4D's palette will be used for the header separator.

If neither **PL_SetFrame** nor [PL_SetRGBFrame](#) are called, then no frame or header separator line will be printed.

Examples

```
//Print 1 pixel wide, solid gray header separator and no frame
PL_SetFrame (eNames;0;"";"";0;1;"Black";"Gray";0)
//Print hairline, solid black frame and header separator line
PL_SetFrame (eNames;0.25;"Black";"Black";0;0.25;"Black";"Black";0)
```

■ PL_SetRGBFrame

(areaRef:L; frameLineWidth:F; frameLinePattern:T; frameLineRed:L; frameLineGreen:L; frameLineBlue:L; headerLineWidth:F; headerLinePattern:T; headerLineRed:L; headerLineGreen:L; headerLineBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ frameLineWidth	real	Width of the frame line.
→ frameLinePattern	text	Pattern of the frame line.
→ frameLineRed	longint	Frame line — Red.
→ frameLineGreen	longint	Frame line — Green.
→ frameLineBlue	longint	Frame line — Blue.
→ headerLineWidth	real	Width of the header separator.
→ headerLinePattern	text	Pattern of the header separator.
→ headerLineRed	longint	Header separator — Red.
→ headerLineGreen	longint	Header separator — Green.
→ headerLineBlue	longint	Header separator — Blue.

PL_SetRGBFrame functions the same as the [PL_SetFrame](#) routine, except that the frame and header separator colors use standard RGB values.

frameLineWidth — 0 to 1. This option controls the line width of the frame. A value of 0.25 pixel should be used for hairlines. A value of 0 means that no frame will be printed.

frameLinePattern — Name of the pattern ([transparency ratio](#)) for the frame. If a null string is used then no frame will be printed.

frameLineRed — Frame line RGB red value. **frameLineGreen** — Frame line RGB green value. **frameLineBlue** — Frame line RGB blue value.

headerLineWidth — 0 to 1. This option controls the line width of the header separator. A value of 0.25 pixel should be used for hairlines. A value of 0 means that no header separator will be printed.

headerLinePattern — Name of the pattern ([transparency ratio](#)) for the header separator. If a null string is used then no header separator will be printed.

headerLineRed — Header separator RGB red value.

headerLineGreen — Header separator RGB green value.

headerLineBlue — Header separator RGB blue value.

If neither [PL_SetFrame](#) nor **PL_SetRGBFrame** are called, then no frame or header separator line will be printed.

Example

The following example sets the frame and header separator line using **PL_SetRGBFrame**:

```
//Print frame and header separator line in a hairline gray pattern
PL_SetRGBFrame(eNames;0.25;"Gray";209; 209; 209;0.25;"Gray"; 209; 209; 209)
```


■ PL_SetHeight

(areaRef:L; numHeaderLines:L; headerHeightPad:L; numRowsLines:L; rowHeightPad:L; minimumHeight:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ numHeaderLines	longint	Number of text lines in the header.
→ headerHeightPad	longint	Extra height for the header.
→ numRowsLines	longint	Number of text lines in each row.
→ rowHeightPad	longint	Extra height for each row.
→ minimumHeight	longint	Minimum pixel height remaining on the page.

PL_SetHeight is used to set the number of lines of text along with additional height padding in the header and in the rows. Only text columns can wrap to more than one line.

If **numRowLines** is set to 2 or more, text elements will be able to wrap into the number of lines specified for each row. Note that all rows will be given the same number of lines regardless of the actual number of lines used by a specific text element.

Additional padding may be set using **rowHeightPad** to allow more space between rows. Text will be centered vertically in the header or row. Note that the padding applies to the entire row and not on a line by line basis within the row.

numHeaderLines — The number of lines in the header. Default is 1.

headerHeightPad — The extra height, in pixels, to give to the header. Default is 2.

numRowLines — The number of lines to give to each row. A value greater than 0 means that the height of each row is the same. The fixed height will either be a function of the number of text lines specified or the height of the largest picture in a picture array if so configured (refer to [PL_SetFormat](#)). A value of zero means that the height of each row is to be calculated automatically based on the data that is to be printed. PrintList Pro examines the elements of all text and picture arrays to determine the height of each row. Default is 1.

rowHeightPad — The extra height, in pixels, to give to each row. Default is 0.

minimumHeight — The minimum remaining available height, in pixels, for the PrintList Pro area to print on the page. For example, if there are several PrintList Pro areas on one form, and you want to make sure that at least two rows are printed on one page for the area specified by **areaRef**, and the row height is 12 points, you can set this parameter to 24. PrintList Pro will test if it has 24 pixels (two rows) left available on the page before printing the area. If not, it will proceed onto the following page. You should specify at least the height of one row in this parameter.

Examples

```
PL_SetHeight(eList;1;4;1;2) //pad the header by 4 pixels and the rows by 2
```

```
PL_SetHeight(eList;2;5;2;0) //set header lines to 2, pad to 5 pixels, set row lines to 2, no padding
```

```
PL_SetHeight (eList;1;4;1;2;12) //check that 12 pixels (one row height here) are available before printing
```

■ PL_SetSort

(areaRef:L; column1:L; ...; columnN:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ column1; ...; columnN	longint	Column(s) to perform sort upon.

PL_SetSort is used to perform a multi-level sort.

column — These parameters specify the columns to use for the sort criteria.

A column greater than 0 causes an ascending sort to be performed upon that column, while a column less than 0 causes a descending sort to be performed upon that column.

If a column is 0, then all successive columns will be ignored.

If the arrays are already sorted, use [PL_SetBrkOrder](#) instead to communicate the sort order to PrintList Pro.

Examples

```
PL_SetSort(eNames;3;4;7) //sort on columns 3, 4, and 7 (all ascending)
```

```
PL_SetSort(eContacts;-1;3;-2) //sort on columns 1 (descending), 3 (ascending), and 2 (descending)
```

■ PL_SetColOpts

(areaRef:L; hideLastColumns:L; hideDetailArea:L; drawingEngine:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ hideLastColumns	longint	Number of columns from the right to hide.
→ hideDetailArea	longint	Hide the list and just show the breaks.
→ drawingEngine	longint	Drawing engine (ignored on Mac): GDI or GDI+.

PL_SetColOpts is used to hide columns from being printed and to hide the entire detail area to show just break level information.

hideLastColumns — This parameter specifies the number of arrays from the right to not print. This parameter is useful for keeping many arrays “in sync” when sorting, but only a subset are to be printed. Default is 0.

hideDetailArea — 0 or 1:

Value	Mode
0	Print the array values in the list (default)
1	Do not print the array values in the list This is useful for printing a summary of break level information without printing the actual list

drawingEngine (ignored on Mac):

- GDI: better rendering, no transparency, no horizontal scaling, limited text rotation features
- GDI+: allows the three features above, but may affect precise rendering on Windows

This parameter is only meaningful on Windows. The two possible values are:

Value	Mode
0	Use GDI+ (default)
1	Change the engine used for printing on Windows to GDI

Examples

```
PL_SetColOpts (eList;2;0) //hide the last two columns
```

```
PL_SetColOpts (eList;0;1;1) //hide the detail area, show only the breaks, use GDI on Windows
```

■ PL_SetCellStyle

(areaRef:L; firstCellCol:L; firstCellRow:L; lastCellCol:L; lastCellRow:L; cellArray:Y; styleNum:L; fontName:T; fontSize:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ firstCellCol	longint	First cell column.
→ firstCellRow	longint	First cell row.
→ lastCellCol	longint	Last cell column.
→ lastCellRow	longint	Last cell row.
→ cellArray	array	Discontiguous cells (two-dimensional longint array).
→ styleNum	longint	Style of the font.
→ fontName	text	Name of the font.
→ fontSize	longint	Size of the font.

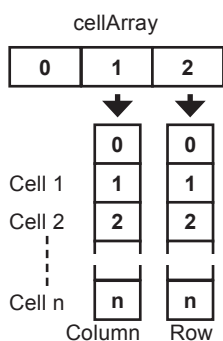
PL_SetCellStyle is used to set the font and/or style of a specific cell, range of cells, or list of cells.

To specify a single cell: if **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** or **lastCellRow** are less than or equal to 0 then only [**firstCellCol**, **firstCellRow**] will be set.

To specify a range of cells: if **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** and **lastCellRow** are greater than 0 then the range of cells from [**firstCellCol**, **firstCellRow**] to [**lastCellCol**, **lastCellRow**] will be set.

To specify discontiguous cells: if **firstCellCol** or **firstCellRow** are less than or equal to 0 then the cells in **cellArray** will be set.

cellArray — Two-dimensional long integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration:



styleNum — This parameter is used to set the style for the specified cells. Use the [Style constants](#), which can be combined.

Note: if bold or italic styles are set, but not installed for the font, PrintList Pro will print regular (plain) characters.

If a cell style has been previously set, the style may be removed by setting **styleNum** to -1. The cell style may be left unchanged by setting **styleNum** to 256.

fontName — This specifies the font for a cell. The cell font may be left unchanged by setting **fontName** to the empty string (""). If the specified font is not found, it will be treated as an empty string and ignored.

fontSize — This specifies the font size for a cell. The cell font size may be left unchanged by setting **fontSize** to 0.

Keep in mind that any settings applied to a cell will be moved with that cell's data if the data is later sorted using [PL_SetSort](#). If you do not want the cell's settings to move, call **PL_SetSort** before applying the cell settings.

Example

```

ARRAY LONGINT(aCellSet;2;4)
// Set cell at column 1, row 3 to bold Helvetica - no change in font size
PL_SetCellStyle (eArea;1;3;0;0;aCellSet;1;"Helvetica";0)
// Set cells from column 2, row 2 to column 5, row 5 to font size 14, no change in style and font
PL_SetCellStyle (eArea;2;2;5;5;aCellSet;256;"";14)
// Set the cells in aCellSet to Times
aCellSet{1}{1}:=1 // column 1, row 1
aCellSet{2}{1}:=1
aCellSet{1}{2}:=1 // column 1, row 2
aCellSet{2}{2}:=2
aCellSet{1}{3}:=2 // column 2, row 5
aCellSet{2}{3}:=5
aCellSet{1}{4}:=2 // column 2, row 6
aCellSet{2}{4}:=6
PL_SetCellStyle (eArea;0;0;0;0;aCellSet;256;"Times";0)

```

■ PL_SetCellColor

(areaRef:L; firstCellCol:L; firstCellRow:L; lastCellCol:L; lastCellRow:L; cellArray:Y; plpForeColor:T; 4dForeColor:L; plpBackColor:T; 4dBackColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ firstCellCol	longint	First cell column.
→ firstCellRow	longint	First cell row.
→ lastCellCol	longint	Last cell column.
→ lastCellRow	longint	Last cell row.
→ cellArray	array	Discontiguous cells (two-dimensional longint array).
→ plpForeColor	text	Foreground color from PrintList Pro's palette.
→ 4dForeColor	longint	Foreground color from 4D's palette.
→ plpBackColor	text	Background color from PrintList Pro's palette.
→ 4dBackColor	longint	Background color from 4D's palette.

PL_SetCellColor is used to set the foreground color and/or background color of a specific cell, range of cells, or list of cells.

To specify a single cell: if **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** or **lastCellRow** are less than or equal to 0 then only [**firstCellCol**, **firstCellRow**] will be set.

To specify a range of cells: if **firstCellCol** and **firstCellRow** are greater than 0 and **lastCellCol** and **lastCellRow** are greater than 0 then the range of cells from [**firstCellCol**, **firstCellRow**] to [**lastCellCol**, **lastCellRow**] will be set.

To specify discontiguous cells: if **firstCellCol** or **firstCellRow** are less than or equal to 0 then the cells in **cellArray** will be set.

cellArray — Two-dimensional long integer array. The first dimension must be two. The first array is for the column indices and the second array is for the row indices. The second dimension must be the same as the number of cells that are to be selected. See the following illustration.

plpForeColor — Name of the color in [PrintList Pro's palette](#). This will be the foreground color for the cell. If the name is not in PrintList Pro's palette or it is the empty string (""), then **4dForeColor** will be used.

4dForeColor — 1 to 256. Foreground color number for the cell (from [4D's palette](#)). If a cell foreground color has been previously set, it may be removed by setting **plpForeColor** to the empty string (""), and **4dForeColor** to 1. The cell foreground color may be left unchanged by setting **plpForeColor** to the empty string (""), and **4dForeColor** to 0.

plpBackColor — Name of the color in PrintList Pro's palette. This will be the background color for the cell. If the name is not in PrintList Pro's palette or it is the empty string (""), then **4dBackColor** will be used.

4dBackColor — 1 to 256. Background color number for the cell (from 4D's palette). If a cell background color has been previously set, it may be removed by setting **plpBackColor** to the empty string (""), and **4dBackColor** to 1. The cell background color may be left unchanged by setting **plpBackColor** to the empty string (""), and **4dBackColor** to 0.

Keep in mind that any settings applied to a cell will be moved with that cell's data if the data is later sorted using [PL_SetSort](#). If you do not want the cell's settings to move, call **PL_SetSort** before applying the cell settings.

Examples

```

ARRAY LONGINT(aCellArray;2;0) //MUST initialize a two-dimensional long integer array
//Set the foreground color of the cell at column 1, row 3 to blue
PL_SetCellColor (eList;1;3;0;0;aCellArray;"blue";0;"";0)
//Set background color of cells from column 2, row 2 to column 5, row 5 to green
PL_SetCellColor (eList;2;2;5;5;aCellArray;"";0;"Green";0)
//Set all negative values in the third column, a real array, to have a foreground color of red
For($i;1;Size of array(aRevenue)) //check each element in the array
  If(aRevenue{$i}<0) //is the value in this element negative?
    PL_SetCellColor(eList;3;$i;0;0;aCellArray;"Red";0;"";0) //if so, then print it in red
  End if
End for

```

■ PL_SetCellRGBColor

(areaRef:L; firstCellCol:L; firstCellRow:L; lastCellCol:L; lastCellRow:L; cellArray:Y; cellForeRed:L; cellForeGreen:L; cellForeBlue:L; cellBackRed:L; cellBackGreen:L; cellBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ firstCellCol	longint	First cell column.
→ firstCellRow	longint	First cell row.
→ lastCellCol	longint	Last cell column.
→ lastCellRow	longint	Last cell row.
→ cellArray	array	Discontiguous cells (two-dimensional longint array).
→ cellForeRed	longint	Foreground red.
→ cellForeGreen	longint	Foreground green.
→ cellForeBlue	longint	Foreground blue.
→ cellBackRed	longint	Background red.
→ cellBackGreen	longint	Background green.
→ cellBackBlue	longint	Background blue.

PL_SetCellRGBColor is used to set the foreground and/or background color of a specific cell, range of cells, or list of cells.

This routine works in the same manner as [PL_SetCellColor](#), except it allows you to specify the colors using standard RGB values.

cellForeRed — Foreground RGB red value.

cellForeGreen — Foreground RGB green value.

cellForeBlue — Foreground RGB blue value.

cellBackRed — Background RGB red value.

cellBackGreen — Background RGB green value.

cellBackBlue — Background RGB blue value.

■ PL_SetCellIcon

(areaRef:L; cellColumn:L; cellRow:L; pictRef:P; iconAlignment:L; horPosition:L; vertPosition:L; offsetOrWidth:L; scaling:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ cellColumn	longint	Column at which to set the icon.
→ cellRow	longint	Row at which to set the icon (0 to set the header).
→ iconRef	longint	Reference of the icon or picture to use.
→ iconAlignment	longint	Position of icon.
→ horPosition	longint	Horizontal position.
→ vertPosition	longint	Vertical position.
→ offsetOrWidth	longint	Pixel offset, or icon width depending on horPosition .
→ scaling	longint	Scaling.

PL_SetCellIcon provides the ability to procedurally print icons in individual cells.

One or two icons may be used (left and right). You can customize the icon(s) using items [from the 4D Picture Library](#) (see details below).

cellColumn — Cell column number.

cellRow — Cell row number (**or 0 to set the header**).

iconRef — Reference of the icon or picture to use from the Picture Library. To associate an icon to the cell, pass the reference number of a picture from the Design environment Picture Library. Pass zero (0) if you do not want any icon for the cell.

See [Header/Cell Icon Support](#) for examples.

iconAlignment — Position of icon (each cell can contain up to two icons):

Value	Mode
0	Places icon on left of cell
1	Places icon on right of cell

horPosition — One the following options:

Value	Mode
0	Default (left for left icon, right for right icon)
1	Align left
2	Align center
3	Align right

vertPosition — One the following options:

Value	Mode
0	Default (top)
1	Align top left
2	Align center
3	Align bottom

offsetOrWidth — when horizontal alignment in horPosition is zero (default position : left for left icon, right for right icon), the **offsetOrWidth** is the offset, i.e. the distance in pixels between the text and the icon (left or right):



horPosition = 0

Otherwise the **offsetOrWidth** is the pixel width that the icon will use - the icon will be aligned in this space:



horPosition = 2 (centered)

scaling — One the following options:

Value	Mode
0	Truncated
1	Scaled

The cell content (text) is printed into the space that is left once the icon is printed.

For example, if the column width is 100 pixels and you print a 15 pixel icon, the remaining width can be calculated as 100 minus the padding (default horizontal indent is 3 for header and data rows on both sides = 6 points), minus the column divider if shown (1 point): $100 - 6 - 1 - 15 = 78$ points where the text will be printed.

Example

The following example will print an icon in r3c2, using an item (ID 1717) from the 4D Picture Library:

```

$col:=2
$row:=3
$iconRef:=1717
$iconPos:=1 //right
$horPos:=0 //default
$verPos:=2 //align center
$offset:=5
$scaling:=0
PL_SetCellIcon (ePLOutput;$col;$row;$iconRef;$iconPos;$horPos;$verPos;$offset;$scaling)

```


■ PL_SetCellBorder

(areaRef:L; cellColumn:L; cellRow:L; borderLeft:L; borderTop:L; borderRight:L; borderBottom:L; offset:L; width:F; redColor:L; greenColor:L; blueColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ cellColumn	longint	Column.
→ cellRow	longint	Row.
→ borderLeft	longint	Print left border.
→ borderTop	longint	Print top border.
→ borderRight	longint	Print right border.
→ borderBottom	longint	Print bottom border.
→ offset	longint	Offset from cell boundary in pixels.
→ width	real	Width of line.
→ redColor	longint	Red.
→ greenColor	longint	Green.
→ blueColor	longint	Blue.

PL_SetCellBorder provides the ability to set the border style and RGB color for a cell.

cellColumn — Column of cell where border will be applied.

cellRow — Row of cell where border will be applied.

borderLeft — Print left border.

borderTop — Print top border.

borderRight — Print right border.

borderBottom — Print bottom border.

offset — Offset from cell boundary in pixels. 0 if the border should be printed at cell boundary (default).

width — Width of line. This parameter is a real value, allowing fractional widths. See [Demo mode dialog](#).

redColor — RGB red value used for the border.

greenColor — RGB green value used for the border.

blueColor — RGB blue value used for the border.

■ PL_SetCellFrame

(areaRef:L; firstCellCol:L; firstCellRow:L; lastCellCol:L; lastCellRow:L; offset:L; width:F; redLightColor:L; greenLightColor:L; blueLightColor:L; redDarkColor:L; greenDarkColor:L; blueDarkColor:L; clearAllBorders:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ firstCellCol	longint	First cell column.
→ firstCellRow	longint	First cell row.
→ lastCellCol	longint	Last cell column.
→ lastCellRow	longint	Last cell row.
→ offset	longint	Offset from cell boundary in pixels.
→ width	real	Width of line.
→ redLightColor	longint	Red (light color).
→ greenLightColor	longint	Green (light color).
→ blueLightColor	longint	Blue (light color).
→ redDarkColor	longint	Red (dark color).
→ greenDarkColor	longint	Green (dark color).
→ blueDarkColor	longint	Blue (dark color).
→ clearAllBorders	longint	Clear all borders within the frame.

PL_SetCellFrame prints a frame around a range of cells. It uses RGB colors: light color for both left and top lines, dark color for both right and bottom line.

The range of cells from [**firstCellCol**, **firstCellRow**] to [**lastCellCol**, **lastCellRow**] will be set.

offset — Offset from cell boundaries in pixels. 0 if the frame should be printed at cell boundaries (default).

width — Width of line. This parameter is a real value, allowing fractional widths. See [Hairline Line Width](#).

redLightColor, **greenLightColor**, **blueLightColor** — RGB values used for both left and top lines colors.

redDarkColor, **greenDarkColor**, **blueDarkColor** — RGB values used for both right and bottom lines colors.

clearAllBorders — If this parameter value is 1, then all cells inside the frame will have their borders removed.

■ PL_SetPageProc

(areaRef:L; callbackMethod:T)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ callbackMethod	text	Name of the page callback project method.

PL_SetPageProc is used to specify a 4D project method to be called at the end of PrintList Pro's processing on every page. Keep in mind that a PrintList Pro page is not necessarily equivalent to a physical page.

It is possible to have several occurrences of a PrintList Pro object for a single page. Each occurrence will invoke the callback method at the end of its page. See [End of Page Callback](#).

callbackMethod — The name of the callback method that is called at the end of every PrintList Pro page.

You must use the following declaration in your callback method:

```
C_LONGINT ($1;$2)
```

PrintList Pro will pass the method specified by **callbackMethod** two parameters: the first indicates which PrintList Pro area is calling the method, and the second specifies the last row printed on that page.

Example

```
PL_SetPageProc (eList;"MyCallback")
```

■ PL_GetVersion

→ version:T

Parameter	Type	Description
← version	text	Version of the PrintList Pro plugin.

PL_GetVersion returns the version number of the currently used PrintList Pro plugin.

Note that getting this property will not trigger the registration dialog if PrintList Pro is not registered (allows to check version before registering)

Example

```
C_TEXT($version)
$version:=PL_GetVersion
```

■ PL_Load

(areaRef:L; XML:T) → result:L

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ XML	text	XML data that was saved using the PL_Save command.
← result	longint	0 if the XML was loaded OK; 1 if not.

PL_Load initializes an area from an XML (using UTF-8) text that was saved to a text field or variable using the [PL_Save](#) command or **AL_Save** (from [AreaList Pro](#)).

PL_Load can be used without any other command use (e.g. no defined columns - they will be read from the XML).

Do not call **PL_Load** more than once for the same area: it is not intended for that and many properties are not reinitialized to defaults.

Example

This example initializes a PrintList Pro area using settings that were saved into a field in the database.

```
$err:=PL_Load (area;[Settings]PLP_template)
```

■ PL_Save

(areaRef:L; XML:T) → result:L

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ XML	text	A variable or field to save an area's XML settings into.
← result	longint	0 if the XML was loaded OK; 2 if not.

PL_Save saves an area's settings as XML (using UTF-8) in a text variable or field.

Use **PL_Save** after configuring the area but before printing. PrintList Pro modifies properties during printing - you would not get the original settings.

Example

Save a PrintList Pro area's settings into a field in the database.

```
C_TEXT($Settings)
```

```
$err:=PL_Save (area;$Settings)
```

```
[Settings]PLP_template:=$Settings
```



Using the Callback Methods

A “callback” is a 4D project method which is executed by a plug-in. PrintList Pro lets you make use of callbacks when printing a PrintList Pro object.

Summary

PrintList Pro provides **five** different callback methods:

- when the end of a printed page is reached (**callbackMethod** parameter of [PL_SetPageProc](#))
- custom calculations in a break (**functionName** parameter of [PL_SetBrkFunc](#))
- custom calculations in a break header (**functionName** parameter of [PL_SetBkHFunc](#))
- calculated columns (**calcCallback** parameter of [PL_SetCalcCall](#)) in field display **or** array display mode
- **computed breaks, with or without printing** (**callbackMethodName** parameter of [PL_ProcessArrays](#))

Warnings

- Callback methods may use most 4D commands, but should not call any PrintList Pro commands or any 4D commands that affect the arrays.

Note: this limitation does not apply to computed break callbacks.

- Callback methods should preserve the current selection of the printing layout’s file by saving and restoring the selection if necessary.
- All callbacks receive parameters, which need to be declared as documented below.

End of Page Callback

PrintList Pro makes use of a callback method to inform you when the end of a printed page is reached. This enables you to perform any necessary processing associated with the end of the page, for example, changing information printed in the footer area of that page or the header area of the next page.

Use **PL_SetPageProc** to specify the 4D project method PrintList Pro is to call. PrintList Pro will pass the method specified by **callbackMethod** two parameters: the first indicates which PrintList Pro area is calling the method, and the second specifies the last row printed on that page.

You must use the following declaration in your callback method:

```
C_LONGINT ($1;$2)
```

Custom Calculations in a Break

[PL_SetBrkFunc](#) is used to specify the callback function for use with custom calculations. The callback function **functionName** is called whenever PrintList Pro encounters the string “\Function” within the text that is to be printed for a specific break level.

Refer to [PL_SetBrkText](#) for details on how to embed the custom calculation string.

PrintList Pro passes information needed for the custom calculation to the callback function.

You must use the following declarations in your callback method:

```
C_LONGINT ($1;$2) //break level, column  
C_TEXT ($3) //column format  
C_LONGINT($4;$5) //start row, end row  
C_TEXT ($0) //custom calculation result to print
```

Custom Calculations in a Break Header

A break header will print information just prior to the group of related values.

[PL_SetBkHFunc](#) is used to specify the name of the break header callback function. This function will be called for any break header that contains a break function.

Refer to [PL_SetBrkText](#) and [PL_SetBkHText](#) to determine how to set a break function for a break level.

The syntax of this command is identical to that of [PL_SetBrkFunc](#).

The callback function **functionName** is called whenever PrintList Pro encounters the string “\Function” within the text that is to be printed for a specific break level. PrintList Pro passes information needed for the custom calculation to the callback function.

You must use the following declarations in your callback method:

```
C_LONGINT ($1;$2) //break level, column  
C_TEXT ($3) //column format  
C_LONGINT($4;$5) //start row, end row  
C_TEXT ($0) //custom calculation result to print
```

Calculated Column Callback

A 4D callback may be attached to a specific column. When information is needed for this column, PrintList Pro will execute the callback to allow you to fill the column with data.

This allows the printing of data calculated from one or more fields **or arrays** as well as any ad hoc data that is desired.

Parameter	Type
\$1	Reference of PrintList Pro object on layout
\$2	Column number
\$3	Type of data in this column (field type or array type)
\$4	Pointer to temporary 4D array (field mode) or an existing sized array (array mode)
\$5	First row for which to calculate cell
\$6	Number of cells to calculate in column

The first three parameters are not absolutely necessary to determine how to fill the column. They are provided to give you more flexibility in the implementation of the callback method.

- The first parameter is the area long integer reference. This gives you the ability to use this callback method for more than one PrintList Pro object.
- The second parameter is the column number. This gives you the ability to use this callback method for many columns within a PrintList Pro object.
- The third parameter is the type of data in the column (field type **or array type**).

The last three parameters are absolutely necessary.

- In field mode, the fourth parameter is a pointer to one of the temporary 4D Arrays used internally by PrintList Pro. This is where you will load the data to be printed in the column. **In array mode, this is a declared, fully sized 4D array (by you as the developer), you have to fill the requested elements**
- The fifth parameter is the number of the first cell that needs to be filled in the column. This is the same as the selected number of the row that contains this cell.
- The sixth parameter is the number of cells (rows) to be filled in the column.

You must declare all six parameters (**\$1** to **\$6**) in the calculated column callback. If any of these parameters are not declared, you will get an error when compiling the database.

You must use the following declarations in your callback method:

```
C_LONGINT ($1;$2;$3;$5;$6)
```

```
C_POINTER ($4)
```

See [Calculated Columns](#) for details.

Computed Breaks

This powerful feature makes all break calculations available for subtotals or other calculated values in any break level, as well as any individual row sub-selection from the top.

These values are returned by PrintList Pro *without need for actual printing*.

[PL_ProcessArrays](#) is used to specify the name of the computed break callback function. This function will be called for the break levels and the columns specified by the **breakArrays** and **dataArrays** parameters.

In addition, the **useDetail** parameter allows calling the callback only on breaks, or for each individual row as well.

The Computed Break callback method receives three parameters: a **handle** needed to call [PL_GetBreakValue](#), the current row number and the current break level, or -1 if individual rows are set to call the callback with **PL_ProcessArrays**.

You must use the following declarations in your callback method:

```
C_LONGINT ($1) //handle to pass over to PL_GetBreakValue  
C_LONGINT($2) //current row number  
C_LONGINT ($3) //break level (or -1 for an individual row)
```

PL_GetBreakValue is called from the callback method to perform usual break level processing calculations such as sum, minimum, etc. for the current break level (or individual row) and the specified column.

See [Using Computed Breaks](#) for details.



Field and Record Commands

PrintList Pro uses the **SELECTION RANGE TO ARRAY** command in 4D to get the records for printing.

Up to **32767** fields (columns) can be printed in a PrintList Pro object.

Using the Field Printing Capability

Temporary Arrays

PrintList Pro internally uses interprocess 4D arrays to get the record data from 4D. **These arrays do not have to be declared in 4D.**

Arrays and Fields

Arrays and fields may not be printed together in the same PrintList Pro object. If arrays are printed in an object, then the field commands will be ignored. Conversely, if fields are printed in an object, then the array commands will be ignored.

Printing 4D Fields

Fields from Related One Tables

Fields from a main table and from related one tables may be printed in the same PrintList Pro object. See the commands [PL_SetFile](#) and [PL_SetFields](#) for further information about printing fields from related one tables.

Sorting

PrintList Pro uses 4D's sorting routines when sorting fields.

When printing records, fields from a related one table can be included in a sort.

Time Data

Time data will be converted to a longint since this is how it is stored internally by 4D.

Maximum Number of Records Printed

The maximum number of PrintList Pro records printed in a PrintList Pro object is **only limited by 4D's own limitations and available memory**.

Performance Issues When Printing Fields

When PrintList Pro prints fields, the automatic column sizing algorithm uses only the first 20 records (or less, if the selection contains less than 20 records) in the selection. These records are always read regardless of whether the columns are automatically or manually sized.

Therefore there is no performance penalty using the automatic column sizing algorithm when printing fields.

See [Performance Issues with Formatting Commands](#) for more information.

Commands

■ PL_SetFile

(areaRef:L; tableNum:L) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ tableNum	longint	Number of 4D table.
← resultCode	longint	Result code.

PL_SetFile tells PrintList Pro what table is the main table from which to print records.

This command is only necessary if the field to be printed in column one is not from the main table, but from a related one table.

PL_SetFile must be called before any fields have been set, otherwise it will be ignored. If this command is not called, then PrintList Pro will use the table of the field printed in column one as the main table.

resultCode — The possible values are:

Constant	Value	Action
PL SetFile Passed	0	Reference of PrintList Pro object on layout
PL Not enough memory	5	Increase 4D's RAM partition
PL Not a file	6	Check to make sure that the table represented by tableNum does exist
PL Wrong 4D version	10	(obsolete)
PL Arrays have been set	11	You've attempted to set fields or a table when arrays have already been set
PL Fields have been set	12	You've attempted to set arrays when fields have already been set

Example

```
$result:=PL_SetFile (eList;Table (->[People]))
```

■ PL_SetFields

(areaRef:L; tableNum:L; columnNumber:L; numFields:L; field1; ...; fieldN:L) → resultCode:L

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ tableNum	longint	Number of 4D table.
→ columnNumber	longint	Column at which to set the first field.
→ numFields	longint	Number of fields to set (up to 15).
→ field1; ...; fieldN	longint	Number(s) of 4D field(s).
← resultCode	longint	Result code.

PL_SetFields tells PrintList Pro what fields to print. Up to fifteen fields can be set at a time. Any 4D field type can be used.

Fields from related one tables may also be printed (see [PL_SetFile](#)). A separate call to **PL_SetFields** must be made to set these fields. To print a related one field, pass the table number of the related one table in the **tableNum** parameter.

This command is also used to print calculated columns. See [Calculated Columns](#).

resultCode — The possible values are:

Constant	Value	Action
PL SetFile Passed	0	Reference of PrintList Pro object on layout
PL Not enough memory	5	Increase 4D's RAM partition
PL Not a file	6	Check to make sure that the table represented by tableNum does exist
PL Not a field	7	The fieldNum passed is not a valid 4D field number
PL Wrong field type	8	The field passed cannot be used by PrintList because the field's type is not supported
PL Maximum fields exceeded	9	32767 fields is the maximum
PL Wrong 4D version	10	(obsolete)
PL Arrays have been set	11	You've attempted to set fields or a table when arrays have already been set

Examples

```
// Set up the eList PrintList Pro object with 5 Fields, all from the same Table
$error:=PL_SetFields(eList;Table(->[People]);1;5;Field(->[People]First Name);Field (->[People]Last Name);
Field (->[People]Salary);Field (->[People]Arrival);Field (->[People]Male))

// Set up the eList PrintList Pro object with 4 Fields, the third one from a related Table
$error:=PL_SetFields(eList;Table(->[People]);1;2;Field(->[People]First Name);Field(->[People]Last Name))
$error:=PL_SetFields(eList;Table(->[Companies]);3;1;Field(->[Companies]Company Name))
$error: PL_SetFields(eList;Table(->[People]);4;1;Field(->[People]Salary))

// Set up the eList PrintList Pro object with 4 Fields, the first one from a related Table
$error:=PL_SetFile(eList;Table(->[People])) // set the main Table since the Field to be set in column one
// is not from the main Table, but from a related one Table
$error:=PL_SetFields(eList;Table(->[Companies]);1;1;Field(->[Companies]Company Name))
$error:=PL_SetFields(eList;Table(->[People]);2;3;Field(->[People]First Name);\
Field(->[People]Last Name); Field(->[People]Salary))
```



Calculated Columns

PrintList Pro columns can be calculated “on the fly” to print the results of calculations performed in a callback method.

This feature is available for both field and **array** printing modes.

Setting a Calculated Column (field mode)

The [PL_SetFields](#) command is used both to set fields to be printed and to set up calculated columns.

- If the **fieldNum** parameter contains an integer greater than or equal to 1, the column will print the field represented by that number.
- If the **fieldNum** parameter contains an integer less than or equal to 0, the column will print calculated data. The absolute value of **fieldNum** will determine the type of data to be printed in the column.

The following table shows the data types that may be printed in a calculated column in field mode:

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

For example, to print a calculated column of type Real, pass [Is Real](#) (-1) in the **fieldNum** parameter.

Setting a Calculated Column (array mode)

The [PL_SetCalcCall](#) command is used to set up calculated columns in **array mode**.

To make a column calculated, create a regular array-based column and then use:

```
PL_SetCalcCall (area; column; methodName)
```

The callback parameters are expected to be declared as (area:L; column:L; type:L; ptr:W; first:L;count:L).

This callback method has the same parameters as a column callback in fields mode, but the array is fully sized (by you as developer), you have to fill the requested elements.

The type is the actual array type, not a field type (e.g. [LongInt array](#) instead of [Is LongInt](#))

The following table shows the data types that may be printed in a calculated column in **array mode**:

Constant	Value
Real array	14
Integer array	15
LongInt array	16
Date array	17
Text array	18
Picture array	19
String array	21
Boolean array	22

Setting the Callback Method

In both field mode and **array mode**, use the [PL_SetCalcCall](#) command to set the [Calculated Column Callback](#) for a column.

In field mode, PrintList Pro will dimension the temporary array before invoking the calculated column callback. There is no need to do it in the callback itself.

In array mode, the arrays used to place the calculated values must be declared and sized just as the other displayed arrays.

Field mode example

The following is an example of a calculated callback method in field mode. It merely calculates an employee's one year anniversary by adding one year to their hire date (using the 4D **Add to date** function).

```
//CalcColCallback
//$1: Area reference (PrintList Pro longint reference) //$2: Column number
//$3: Type of data in this column
//$4: Pointer to temporary 4D array
//$5: First record for which to calculate cell
//$6: Number of cells to calculate in column
//Declare the parameters
C_LONGINT($1;$2;$3;$5;$6) //these must be declared
C_POINTER($4) //this must be declared
C_LONGINT ($i)
ARRAY DATE($aHireDate;0) //local array can be used since we only need it here for calculation
SELECTION RANGE TO ARRAY($5;$5+$6-1;[Employee]Hire Date;$aHireDate)
For ($i;1;$6)
    $4->{$i}:= Add to date($aHireDate{$i};1;0;0)
End for
```

Array mode example

The following is an example of a calculated callback method in **array mode**, using the same simple calculation as above, but with 4D arrays being printed.

These arrays have been initially declared and included in the PrintList Pro area with the same command (either [PL_SetArraysNam](#) or [PL_AddColumn](#)) for both non-calculated and calculated arrays:

```
//Declare the arrays
ARRAY TEXT(aName;0)
ARRAY DATE(aHireDate;0) //not printed, but needed for calculation
SELECTION TO ARRAY([Employee]Name;aName;[Employee]Hire Date;aHireDate)
ARRAY DATE (aAnniversary;Size of array(aName)) //this is our calculated array - must be of same size!
//Arrays to print
$error:= PL_AddColumn(eList;->aName;0) //no need to specify colum number
$error:= PL_AddColumn(eList;->aAnniversary;0)
//Set calculated callback method for column 2
PL_SetCalcCall (eList;2;"CalcColCallbackArray")
```

Now we use the callback as previously to populate the array on the fly:

```
//CalcColCallbackArray
//$1: Area reference (PrintList Pro longint reference) //$2: Column number
//$3: Type of array in this column
//$4: Pointer to the printed array
//$5: First row for which to calculate cell
//$6: Number of cells to calculate in column
//Declare the parameters
C_LONGINT($1;$2;$3;$5;$6) //these must be declared
C_POINTER($4) //this must be declared
C_LONGINT ($i)
For ($i;$5;$5+$6-1)
    $4->{$i}:= Add to date(aHireDate{$i};1;0;0)
End for
```

Commands

■ PL_SetCalcCall

(areaRef:L; columnNumber:L; calcCallback:T)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNumber	longint	Column number.
→ calcCallback	text	4D method called to fill row(s) of a calculated column.

PL_SetCalcCall is used to set a callback method for a calculated column.

columnNumber — This parameter specifies the column on which to attach the **calcCallback** method.

calcCallback — This method will be called whenever row(s) need to be filled in a calculated column. If this is an empty string then no method will be called.

The first two parameters (\$1 and \$2) passed to this callback method are **areaRef** and **columnNumber**. Therefore, if desired, the same callback can be used for more than one PrintList Pro object and for many columns in an object.

For information on how to write a calculated column callback, see the section [Calculated Column Callback](#).

Example

```
//Set calculated callback method for column 3
PL_SetCalcCall (eArea;3;"CalcColCallback")
```



Break Level Processing

About PrintList Pro Break Level Processing

The PrintList Pro break level processing routines provide the functionality of 4D's Quick Report capabilities, and much more.

You can choose to display a variety of information for any break level and any column within that break including: static text, Quick Report calculations, custom calculations or break data insertion for each break.

Each piece of information can be shown in different fonts, sizes and styles as well as different colors. You can also control the display of repeated values.

When Do Breaks Occur?

When a list is sorted, often some of the sorted array elements will have groups of identical values.

For example, if the membership of a club that consisted of members from many different countries was sorted by country, the membership list would likely consist of many members that were from the same country. Because the list is sorted, all the members from the same country would be grouped together in the list. A "break" would occur in the list anytime the group changes (the country is different).

Multiple break levels occur when the list is sorted on multiple criteria. The number of any break level is determined by its position in the sort order.

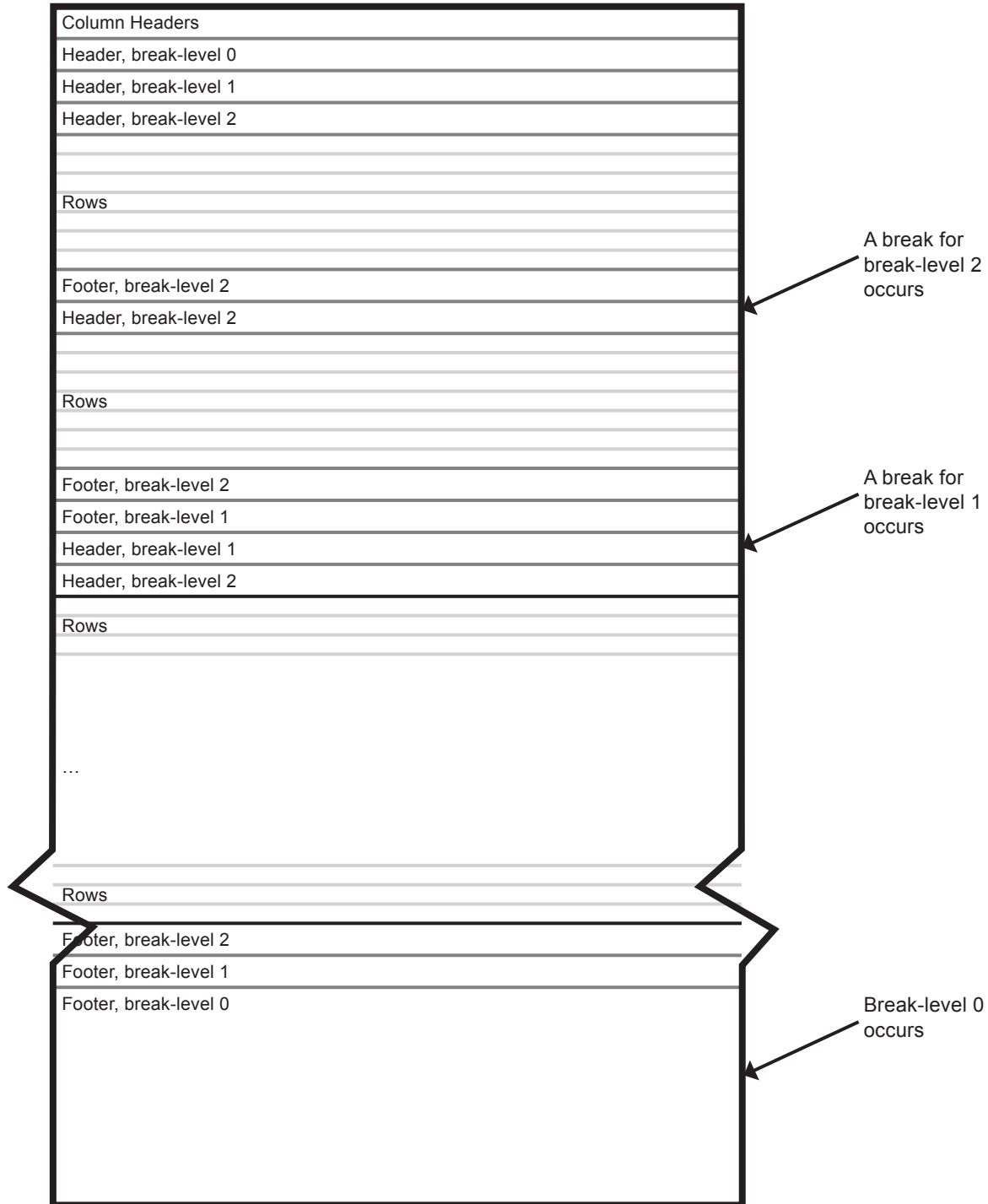
If the same membership list is sorted by country first and then by city, a break for break level one (country) would occur whenever the country changes in the list, and a break for break level two (city) would occur whenever the city changes in the list.

When a break occurs for a certain level, every break level higher than it will occur as well. In the club example, break level 2 (city) will always break whenever a break for break level 1 (country) occurs.

You may want to show leading or summary information for the groups of information within the list. In PrintList Pro, these areas are called break headers and break footers respectively.

A break header will print information just prior to the group of related values while a break footer will print information immediately after the group.

The following is an illustration of the location of break headers and footers within a list:



Using PrintList Pro Break Level Processing

Any break level routine that accepts a column number requires that the column/array be set using [PL_SetArraysNam](#) or [PL_AddColumn](#) prior to calling that routine.

PrintList Pro provides up to 15 break levels and a total line at the end of list. For any given list, the number of break levels can be no greater than the number of sorted arrays.

PrintList Pro will only print a break for a break level that has been configured using [PL_SetBrkText](#) for break footers or [PL_SetBkHText](#) for break headers.

A break level can be configured regardless if lower breaks have been configured. This is different than 4D's Quick Report Editor that requires the users to "stack" break levels upon lower (closer to 0) break levels and hide the break levels that are not desired.

With PrintList Pro, a break level can be configured regardless if lower breaks have been configured; therefore, there is no need to "hide" a break. In the people example, information can be shown for a group of people in the same city without necessarily showing any information for the state.

In conjunction with [PL_SetArraysNam](#), PrintList Pro uses the information in [PL_SetSort](#) to determine where the breaks occur within the arrays.

If the arrays passed to PrintList Pro are pre-sorted, [PL_SetBrkOrder](#) should be used to notify PrintList Pro of the sort order without forcing an unnecessary sort.

The break level parameter used in many break level routines is the position of a particular column within the sort order given.

For example, if a list of people were to be sorted by state, county, and city respectively, the calls to set and sort the arrays would be:

```
PL_SetArraysNam (eList;1;4;aPeople;aCity;aCounty;aState)
```

```
PL_SetSort (eList;4;3;2)
```

Break level 1 is state (column 4), break level 2 is county (column 3), and break level 3 is city (column 2).

The following commands are used to configure breaks: [PL_SetBrkColOpt](#), [PL_SetBrkColRGBOpt](#), [PL_SetBrkColor](#), [PL_SetBrkRGBColor](#), [PL_SetBrkFunc](#), [PL_SetBrkHeight](#), [PL_SetBrkStyle](#), [PL_SetBrkText](#).

The following commands are used to configure break headers: [PL_SetBkHColOpt](#), [PL_SetBkHColRGBOpt](#), [PL_SetBkHColor](#), [PL_SetBkHRGBColor](#), [PL_SetBkHFunc](#), [PL_SetBkHHeight](#), [PL_SetBkHStyle](#), [PL_SetBkHText](#).

[PL_SetBrkRowDiv](#) and [PL_SetBrkRowRGBDiv](#) are not specifically break footer or header routines. These commands specify the line that are drawn between a break footer and the following break header or group of rows.

Setting a Break Level

To show information for a break level, you will need to use [PL_SetBrkText](#) for break footers and [PL_SetBkHText](#) for break headers.

A text variable containing the information to be printed is passed in for a particular cell within the break. Because a break can consist of more than one text line, the supplied text may wrap into several lines. See [Multiple Lines in a Break](#) and [Variable Height Breaks](#) for more information.

Carriage returns may be embedded into the text to force wrapping.

Text Overflow and Justification in Breaks

Unlike a Quick Report, information to be printed in a cell can overflow into adjacent columns depending on the justification.

As specified in the call to [PL_SetBrkText](#) or [PL_SetBkHText](#), the area used to print the text is taken from the column specified and adjacent columns to the right for left justification, columns to the left for right justification, and columns on both sides for center justification.

Built-in Calculations

Calculations may be embedded into the text passed to [PL_SetBrkText](#) or [PL_SetBkHText](#).

Built-in functions — sum, minimum, average, maximum, count, **variance**, **standard deviation** and break value — can be inserted into the text at any desired location. These are identical to 4D's QuickReport calculations except for break value which inserts the value from the array that caused the break.

Custom Calculations

You may create custom calculations to be used with or instead of the built-in calculations.

When PrintList Pro sees the custom calculation delimiter, it will execute a callback method, specified using [PL_SetBrkFunc](#) for break footers and [PL_SetBkHFunc](#) for break headers, that performs the custom calculation.

The callback method may use most 4D commands, but should not call any PrintList Pro commands or any 4D commands that affect the arrays.

Also, the callback method should preserve the current selection of the printing layout's file by saving and restoring the selection if necessary. The value returned by the callback method will then be printed at the embedded position within the break text.

Suppressing Repeated Values

Repeated values in a sorted list can be suppressed using [PL_SetRepeatVal](#). The repeated value is shown on the first occurrence and at the top of each page thereafter.

[PL_SetRepeatVal](#) works on any sorted list regardless of whether any break level information is shown or not.

Style and Color in Breaks

Style and color settings can be provided for each column within each break level using:

- [PL_SetBrkColor](#) for break footers and [PL_SetBkHColor](#) for break headers to set foreground and background colors using [PrintList Pro's palette](#) or [4D's palette](#).
- [PL_SetBrkRGBColor](#) for break footers and [PL_SetBkHRGBColor](#) for break headers to set foreground and background colors using standard RGB values.
- [PL_SetBrkStyle](#) for break footers and [PL_SetBkHStyle](#) for break headers to set text style settings

If no style or color information is given, PrintList Pro will use the corresponding column settings from the list.

Multiple Lines in a Break

Both break headers and footers can be configured to be a fixed number of lines per break or a variable number of lines.

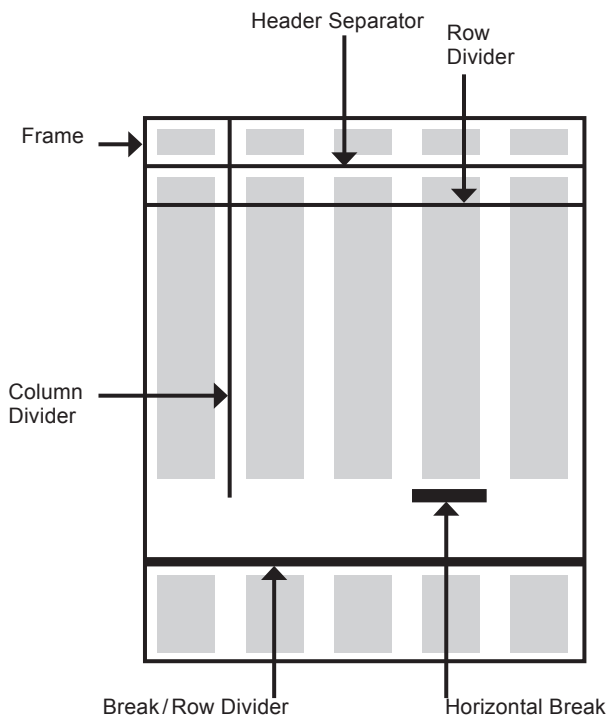
To set the number of lines to be printed in a break level, use [PL_SetBrkHeight](#) for break footers and [PL_SetBkHHeight](#) for break headers. Please read the section [Variable Height Breaks](#) for more information.

If no calls to [PL_SetBrkText](#) are made for a specific break level, nothing will be displayed for any break occurring for that level regardless of the number of lines or height pad specified in [PL_SetBrkHeight](#).

Lines Displayed in a Break

PrintList Pro provides complete control over all the lines printed in a break as shown in the following illustration:

Whenever a break or group of breaks is printed, the line following the last break, referred to as the Break/Row Divider, can be configured using [PL_SetBrkRowDiv](#) or [PL_SetBrkRowRGBDiv](#). If the Break/row divider is not set, the row divider (if set) will be printed by default.



If column dividers have been set using [PL_SetDividers](#) or, they can be printed in the break using [PL_SetBrkColOpt/PL_SetBrkColRGBOpt](#) for break footers and [PL_SetBkHColOpt/PL_SetBkHColRGBOpt](#) for break headers.

If set, the column divider will be printed in the break area to the right of each column within the break level.

A horizontal line, referred to in the illustration on the previous page as the Horizontal Break line, may be printed within the break areas as well.

- for break footers, use [PL_SetBrkColOpt/PL_SetBrkColRGBOpt](#) to print a line at the top of the cell within the break footer area
- for break headers, use [PL_SetBkHColOpt/PL_SetBkHColRGBOpt](#) to print a line at the bottom of the cell within the break header area

Hide the Detail Area

The detail area (the list of array data) can be hidden to show only the break level information on the page using [PL_SetColOpts](#). This is ideal for giving a summary of the array information.

Page Breaks

[PL_SetPageBreak](#) tells PrintList Pro whether or not to force a page break on any given break level.

The page break will occur immediately after the break footer is printed for that break level. However, a page break can be set for a break level regardless of whether a break level is configured to print a break footer.

[PL_SetBrkOpts](#) can be called with the parameter **printLastPageBreak** to print or suppress a a page break if it occurs on the last page. This option is used to avoid the printing of an unneeded blank page at the end of a PrintList Pro report.

Variable Height Breaks

Any given break level can be set to be a variable height. The same rules apply to breaks as to the rows in that a break can be of no height or up to the height of an entire page.

When a break level is set to be variable height, PrintList Pro will perform the necessary break level calculation(s) to determine the height of the text that is to be printed in the break.

To set a break level to be variable height use [PL_SetBrkHeight](#) for break footers and [PL_SetBkHHeight](#) for break headers.

Using Break Headers

The ability to configure break headers is identical to that of break footers including:

- all calculations: sum, min, average, max, count, **variance**, **standard deviation** and break value
- a callback for the break function (one callback for all break headers)
- full style (font, size, style) control for each cell within the break
- horizontal and vertical line/divider control
- foreground and background colors for each cell within the break
- variable height breaks

Break headers can be configured using six commands: [PL_SetBkHText](#), [PL_SetBkHFunc](#), [PL_SetBkHStyle](#), [PL_SetBkHColor](#)/[PL_SetBkHRGBColor](#), [PL_SetBkHHeight](#), and [PL_SetBkHColOpt](#)/[PL_SetBkHColRGBOpt](#). These commands are identical in syntax to the break footer commands.

[PL_SetBrkColOpt](#)/[PL_SetBrkColRGBOpt](#) can be called to print horizontal lines at the top of a cell with a break footer. With break headers, the equivalent commands **[PL_SetBkHColOpt](#)**/**[PL_SetBkHColRGBOpt](#)** will print lines at the bottom of a cell within the break header.

Just as with break footers, break headers will only be printed if the command to set the break text, **[PL_SetBkHText](#)**, has been called for a particular break level.

Break headers can be configured to be fixed or variable height and have full background color control as is now available with break footers.

Using Computed Breaks

These powerful commands **can be used as array utilities without need to print anything and don't even require to set up a plug-in area.**

[PL_ProcessArrays](#) is used to specify the name of the computed break callback function. This function will be called for the break levels and the columns specified by the **breakArrays** and **dataArrays** parameters (pointers to arrays).

In addition, the **useDetail** parameter allows calling the callback only on breaks (value 0), or for each individual row as well (value 1).

PL_ProcessArrays operates on a copy of the arrays, you can freely modify them in the callback.

The callback method is called by PrintList Pro as:

callbackMethodName (handle:L; row:L; breakLevel:L)

[PL_GetBreakValue](#) can only be called from the callback method with the specified **handle** that was received.

This command performs any break level processing calculation (sum, minimum, average, maximum, count, variance, standard deviation) for the current break level (or individual row) and the specified **column** (from the list previously defined by the **dataArrays** parameter).

[See Example 5 — Computed Breaks.](#)

Commands

■ PL_SetPageBreak

(areaRef:L; breakLevel:L; insertPageBreak:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ insertPageBreak	longint	Insert a page break after the break level.

PL_SetPageBreak is used to set a page break after each break at the specified break level.

A page break can be inserted provided the rows are sorted. However, it is not necessary to configure a break level using [PL_SetBrkText](#) or [PL_SetBkHText](#) in order to insert a page break.

For the specified break level, the break header, the rows, and the break footer will print (if so configured) prior to the page break. Any subsequent break footers or rows will be printed at the top of the following page.

breakLevel — The position in the sort order specified in [PL_SetSort](#) or [PL_SetBrkOrder](#) i.e. 1 through n, where n is the number of levels of sort.

insertPageBreak — 0 or 1:

Value	Mode
0	No page break will be inserted (default)
1	A page break will be inserted after each break at the specified break level

Refer to [PL_SetBrkOpts](#) to see how to print or suppress a page break on the last page of a PrintList report.

Example

```
PL_SetPageBreak(eList;1;1) //force a page break after printing each break of break level 1
```

■ PL_SetBrkOpts

(areaRef:L;printLastPageBreak:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ printLastPageBreak	longint	Print a page break on the last page.

PL_SetBrkOpts is used to set options pertaining to break levels.

printLastPageBreak — 0 or 1:

Value	Mode
0	The page break will be suppressed (default)
1	If there is a page break on the last page, it will be printed

Refer to [PL_SetPageBreak](#) for configuring page breaks.

Example

```
PL_SetBrkOpts(eList;0) // don't print the last page break
```

■ PL_SetBrkOrder

(areaRef:L; colNum1:L; ...; colNumN:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ colNum1; ...; colNumN	longint	Column(s) that reflects the sort order.

PL_SetBrkOrder is used to communicate the sort order of a previously sorted list to PrintList Pro. The sort information is used by PrintList Pro to determine where breaks occur within the sorted list.

The syntax of this call is identical to that of [PL_SetSort](#).

colNum — A value greater than 0 indicates that an ascending sort was performed upon that column, while a value less than 0 indicates a descending sort. If a **columnNum** is 0 then all successive columns will be ignored.

Note: *PL_SetBrkOrder* does not perform a sort.

Examples

```
PL_SetBrkOrder(eContacts;3;4;7) // was sorted on columns 3, 4, and 7 (all ascending)
```

```
PL_SetBrkOrder(eContacts;-1;3;-2) // was sorted on columns 1 descending, 3 ascending, 2 descending
```


■ PL_SetRepeatVal

(areaRef:L; columnNum:L; repeatValues:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ columnNum	longint	Column number.
→ repeatValues	longint	Hide/print repeated values for this column.

PL_SetRepeatVal is used to control the printing of repeated values within a sorted column.

Use [PL_SetSort](#) to sort the arrays or if already sorted, use [PL_SetBrkOrder](#) to communicate the sorted order to PrintList Pro.

columnNum — The column number to apply this command to. Use a value of zero (0) to apply the repeat settings to all columns in the list.

repeatValues — 0 or 1:

Value	Mode
0	Print all repeated values (default)
1	Only print the first occurrence of a repeated value after reaching a break and at the top of each page thereafter

Examples

```
PL_SetRepeatVal(eList;3;0) // show repeat values for column 3
```

```
PL_SetRepeatVal(eList;0;1) // hide repeat values for all columns
```

■ PL_SetBrkText

(areaRef:L; breakLevel:L; columnNum:L; breakText:T; numColsToOverflow:L; justification:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Column number.
→ breakText	longint	Text to be printed when a break occurs.
→ numColsToOverflow	longint	Number of adjacent columns to overflow into.
→ justification	longint	Justification of the break text.

PL_SetBrkText is used to specify the text, passed in as **breakText**, to be printed in the **breakLevel** and **columnNum** specified.

Calculations can be performed upon the corresponding values in the list. A calculation is performed by embedding a special calculation string(s) into **breakText** as described below.

Text can overflow to adjacent columns using the **numColsToOverflow** parameter. The text is justified within a column(s) using the justification parameter.

If no call to *PL_SetBrkText* is made for **breakLevel**, no break information will be printed for that break level.

breakLevel — The position in the sort order specified in [PL_SetSort](#) or [PL_SetBrkOrder](#) — i.e., 1 through n, where n is the number of levels of sort. Use a value of 0 to specify the total line to be printed at the end of the list.

columnNum — The column number at which the specified text will be placed for the specified **breakLevel**.

breakText — The text to be shown in **columnNum** whenever a break occurs for **breakLevel**. The text may automatically wrap to multiple lines or carriage returns can be embedded to force text wrapping. Use [PL_SetBrkHeight](#) or [PL_SetBkHHeight](#) to set the number of text lines for a break level if more than 1 line is anticipated.

A calculation may be performed on all the values in **columnNum** since the last break for **breakLevel**.

breakText can include embedded calculation strings which inform PrintList Pro to perform the desired calculation using the array values associated with **columnNum**. The result of the calculation is formatted into a string which replaces the calculation string at its location within **breakText**.

The following table shows a list of the calculations and the associated strings (not case-sensitive) that can be included in **breakText**:

Calculation	Calculation String
Sum	"\Sum"
Minimum	"\Minimum"
Average	"\Average"
Maximum	"\Maximum"
Count	"\Count"
Variance	"\Var"
Standard deviation	"\Dev"
Break Value Insertion	"\BreakValue"

Sum, Minimum, Average, Maximum, Count and **Standard deviation** are identical to that of 4D's QuickReport editor.

Break Value Insertion will use the array value from the sorted column that is associated with **breakLevel** for insertion into **breakText**.

Custom Calculation will execute the callback function specified in [PL_SetBrkFunc](#) or [PL_SetBkHFunc](#) to retrieve a string for insertion into **breakText**.

See [PL_SetBrkFunc](#) and [PL_SetBkHFunc](#) for details of performing custom calculations using the callback function.

Not all calculations are available on all array types. The following table lists the possible calculations for the various array data types and the resulting type of the calculation:

Column Data Type	Calculation	Data Type of Result
numeric	Sum	same as column
numeric	Minimum	same as column
numeric	Average	real
numeric	Maximum	same as column
all	Count	longint
numeric	Variance	same as column
numeric	Standard deviation	same as column
all	Break Value Insertion	same as break column
all	Custom Calculation	formatted text

When the resulting value's data type is the same as the **columnNum** data type, the value is formatted using the column's format.

Otherwise, the calculations' results will use PrintList Pro's real and integer default formats where appropriate.

The result of the custom calculation is formatted by the callback function which returns text.

numColsToOverflow — Number of adjacent columns to use when overflowing to the right (when left justified), or left (when right justified), or both (when center justified) of **columnNum**. A value of zero, which is the default, indicates that **breakText** will only be printed within the column.

justification — The justification of **breakText** within the column (or columns if **numColsToOverflow** is greater than 0):

Value	Justification
0	Default (justification of columnNum in the list detail area will be used)
1	Left
2	Center
3	Right

Examples

```
//Break level 1, column 3, overflow 2 columns to the right, left-justified
```

```
PL_SetBrkText(eList;1;3;"Company Subtotals";2;1)
```

```
//Break level 3, column 6, no column overflow, use default justification
```

```
PL_SetBrkText (eList;3;6;"\Sum";0;0)
```

```
//Break level 4, column 3, overflow into 1 column on both sides of this column, center-justified
```

```
PL_SetBrkText(eList;4;3;"There are \Count people in this department.";1;2)
```

■ PL_SetBkHText

(areaRef:L; breakLevel:L; columnNum:L; breakText:T; numColsToOverflow:L; justification:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Column number.
→ breakText	longint	Text to be printed when a break occurs.
→ numColsToOverflow	longint	Number of adjacent columns to overflow into.
→ justification	longint	Justification of the break text.

PL_SetBkHText is used to specify the text that is to be printed in the break header. The syntax of this command is identical to that of [PL_SetBrkText](#).

■ PL_SetBrkFunc

(areaRef:L; functionName:T)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ functionName	text	Function name to be called back.

PL_SetBrkFunc is used to specify the callback function for use with custom calculations. The callback function **functionName** is called whenever PrintList Pro encounters the string "\Function" within the text that is to be printed for a specific break level.

Refer to [PL_SetBrkText](#) for details on how to embed the custom calculation string.

PrintList Pro passes information needed for the custom calculation to the callback function.

The callback function should be created using the following interface:

```
//Function: MyBreakFunction (breakLevel; column; colFormat; startRow; endRow)
C_LONGINT ($1;$2) //break level, column
C_TEXT ($3) //column format
C_LONGINT($4;$5) //start row, end row
C_TEXT ($0) //custom calculation result to print
//... perform the custom calculation
$0:=String(customCalc;$3) //format the string using the column format
```

The callback function is passed the break level number for which the break has occurred, the column number, the format of that column, and the starting and ending indexes of the corresponding array.

The function should return, as formatted text, the result of the calculation. The return variable, **\$0**, should be of type text and should be 255 characters or less.

Example

```
PL_SetBrkFunc(eList;"Break Function") //custom calculation callback
```

See [Example 4 - Break Level Processing](#) for an example of a custom calculation callback function.

■ PL_SetBkHFunc

(areaRef:L; functionName:T)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ functionName	text	Function name to be called back.

PL_SetBkHFunc is used to specify the name of the break header callback function. This function will be called for any break header that contains a break function.

Refer to [PL_SetBrkText](#) and [PL_SetBkHText](#) to determine how to set a break function for a break level.

The syntax of this command is identical to that of [PL_SetBrkFunc](#).

■ PL_SetBrkStyle

(areaRef:L; breakLevel:L; columnNum:L; fontName:T; size:L; styleNum:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout
→ breakLevel	longint	Break level number
→ columnNum	longint	Column number
→ fontName	text	Name of the font
→ size	longint	Size of the font
→ styleNum	longint	Style of the font

PL_SetBrkStyle is used to control the appearance of the break level text for the **breakLevel** and **columnNum** specified. The columns can be controlled individually or as a group.

breakLevel — This parameter specifies the break level to apply this command to.

columnNum — The column to apply this command to. Use a value of zero (0) to apply the parameters to all columns within that break level.

fontName — This specifies the font for the break. The break font may be left unchanged by setting **fontName** to the empty string (""). If the font specified is not found, it will be treated as an empty string and ignored.

fontSize — This specifies the font size for the break. The break font size may be left unchanged by setting **fontSize** to 0.

styleNum — This parameter is a font style code. Use the [Style constants](#), which can be combined.

Note: if bold or italic styles are set, but not installed for the font, PrintList Pro will print regular (plain) characters.

■ PL_SetBkHStyle

(areaRef:L; breakLevel:L; columnNum:L; fontName:T; size:L; styleNum:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Column number.
→ fontName	text	Name of the font.
→ size	longint	Size of the font.
→ styleNum	longint	Style of the font.

PL_SetBkHStyle is used to set the style of the text that is to be printed in the break header.

The syntax of this command is identical to that of [PL_SetBrkStyle](#).

■ PL_SetBrkColor

(areaRef:L; breakLevel:L; columnNum:L; plpForeColor:T; 4dForeColor:L; plpBackColor:T; 4dBackColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Column number.
→ plpForeColor	text	Foreground color from PrintList Pro's palette.
→ 4dForeColor	longint	Foreground color from 4D's palette.
→ plpBackColor	text	Background color from PrintList Pro's palette.
→ 4dBackColor	longint	Background color from 4D's palette.

PL_SetBrkColor is used to specify the color of the text to be printed in the specified **columnNum** and **breakLevel**.

breakLevel — This parameter specifies the break level to apply this command to.

columnNum — The column to apply this command to. Use a value of zero (0) to apply the colors to all columns within that break level.

plpForeColor — Name of the color in [PrintList Pro's palette](#). This will be the foreground color for the break. If the name is not in PrintList Pro's palette or it is the empty string (""), then **4dForeColor** will be used.

4dForeColor — 1 to 256. Foreground color number for the break (from [4D's palette](#)). If a break foreground color has been previously set, it may be removed by setting **plpForeColor** to the empty string (""), and **4dForeColor** to 1. The break foreground color may be left unchanged by setting **plpForeColor** to the empty string (""), and **4dForeColor** to 0.

plpBackColor — Name of the color in PrintList Pro's palette. This will be the background color for the break. If the name is not in PrintList Pro's palette or it is the empty string (""), then **4dBackColor** will be used.

4dBackColor — 1 to 256. Background color number for the break (from 4D's palette). If a break background color has been previously set, it may be removed by setting **plpBackColor** to the empty string (""), and **4dBackColor** to 1. The break background color may be left unchanged by setting **plpBackColor** to the empty string (""), and **4dBackColor** to 0.

Examples

```
//Set the foreground color for the break in column 3, break level 1, to red - no background color
```

```
PL_SetBrkColor (eList;1;3;"red";0;"";0)
```

```
//Set the background color for the break in column 3, break level 1, to blue - no foreground color
```

```
PL_SetBrkColor (eList;1;3;"";0;"Blue";0)
```

■ PL_SetBrkRGBColor

(areaRef:L; breakLevel:L; columnNum:L; breakForeRed:L; breakForeGreen:L; breakForeBlue:L; breakBackRed:L; breakBackGreen:L; breakBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Column number.
→ breakForeRed	longint	Foreground red.
→ breakForeGreen	longint	Foreground green.
→ breakForeBlue	longint	Foreground blue.
→ breakBackRed	longint	Background red.
→ breakBackGreen	longint	Background green.
→ breakBackBlue	longint	Background blue.

PL_SetBrkRGBColor is used to specify the color of the text to be printed in the specified **columnNum** and **breakLevel**.

This routine works in the same manner as [PL_SetBrkColor](#), except it allows you to specify the colors using standard RGB values.

■ PL_SetBkHColor

(areaRef:L; breakLevel:L; columnNum:L; plpForeColor:T; 4dForeColor:L; plpBackColor:T; 4dBackColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Column number.
→ plpForeColor	text	Foreground color from PrintList Pro's palette.
→ 4dForeColor	longint	Foreground color from 4D's palette.
→ plpBackColor	text	Background color from PrintList Pro's palette.
→ 4dBackColor	longint	Background color from 4D's palette.

PL_SetBkHColor is used to specify the color of the text to be printed in the break header for the specified **columnNum** and **breakLevel**.

breakLevel — This parameter specifies the break level to apply this command to.

columnNum — The column to apply this command to. Use a value of zero (0) to apply the colors to all columns within that break level.

plpForeColor — Name of the color in [PrintList Pro's palette](#). This will be the foreground color for the break. If the name is not in PrintList Pro's palette or it is the empty string (""), then **4dForeColor** will be used.

4dForeColor — 1 to 256. Foreground color number for the break header (from [4D's palette](#)). If a break header foreground color has been previously set, it may be removed by setting **plpForeColor** to the empty string (""), and **4dForeColor** to 1. The break header foreground color may be left unchanged by setting **plpForeColor** to the empty string (""), and **4dForeColor** to 0.

plpBackColor — Name of the color in PrintList Pro's palette. This will be the background color for the break header. If the name is not in PrintList Pro's palette or it is the empty string (""), then **4dBackColor** will be used.

4dBackColor — 1 to 256. Background color number for the break header (from 4D's palette). If a break header background color has been previously set, it may be removed by setting **plpBackColor** to the empty string (""), and **4dBackColor** to 1. The break header background color may be left unchanged by setting **plpBackColor** to the empty string (""), and **4dBackColor** to 0.

Examples

```
//Set the foreground color for the break header in column 3, break level 1, to red - no background color
```

```
PL_SetBkHColor (eList;1;3;"red";0;"";0)
```

```
//Set the background color for the break header in column 3, break level 1, to blue - no foreground color
```

```
PL_SetBrkColor (eList;1;3;"";0;"Blue";0)
```

■ PL_SetBkHRGBColor

(areaRef:L; breakLevel:L; columnNum:L; brkHdrForeRed:L; brkHdrForeGreen:L; brkHdrForeBlue:L; brkHdrBackRed:L; brkHdrBackGreen:L; brkHdrBackBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Column number.
→ brkHdrForeRed	longint	Foreground red.
→ brkHdrForeGreen	longint	Foreground green.
→ brkHdrForeBlue	longint	Foreground blue.
→ brkHdrBackRed	longint	Background red.
→ brkHdrBackGreen	longint	Background green.
→ brkHdrBackBlue	longint	Background blue.

PL_SetBkHRGBColor is used to specify the color of the text to be printed in the break header for the specified **columnNum** and **breakLevel**.

This routine works in the same manner as [PL_SetBkHColor](#), except it allows you to specify the colors using standard RGB values.

■ PL_SetBrkHeight

(areaRef:L; breakLevel:L; numBreakLines:L; breakHeightPad:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ numBreakLines	longint	Number of text lines in the break.
→ breakHeightPad	longint	Extra height for the break.

PL_SetBrkHeight is used to set the number of lines of text along with additional height pad for **breakLevel**. If no calls to [PL_SetBrkText](#) are made for **breakLevel**, nothing will be displayed for any break occurring for that level regardless of the number of lines or height pad specified in **PL_SetBrkHeight**.

numBreakLines — The number of lines to give to each break of the specified break level. A value greater than 0 means that the height of each break is the same. The fixed height will be a function of the number of text lines specified. A value of zero means that the height of each break is to be calculated automatically based on the data that is to be printed. Default is 1.

breakHeightPad — The extra height to give to the break level. **breakHeightPad** sets an additional padding to allow more space around the break. Text will be centered vertically within the break. Default is 0.

The padding applies to the entire break and not on a line by line basis within the break.

Examples

```
//Allocate 5 lines and no pad for break level 3
PL_SetBrkHeight (eList;3;5;0)
//Break level 2, Pad by 4 pixels, only 1 line
PL_SetBrkColor (eList;2;1;4)
```

■ PL_SetBkHHeight

(areaRef:L; breakLevel:L; numBreakLines:L; breakHeightPad:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ numBreakLines	longint	Number of text lines in the break.
→ breakHeightPad	longint	Extra height for the break.

PL_SetBkHHeight is used to set the number of lines of text along with additional height pad for the specified break header level. The syntax of this command is identical to that of [PL_SetBrkHeight](#).

numBreakLines — The number of lines to give to each break of the specified break level. A value greater than 0 means that the height of each break is the same. The fixed height will be a function of the number of text lines specified. A value of zero means that the height of each break is to be calculated automatically based on the data that is to be printed. Default is 1.

breakHeightPad — The extra height to give to the break level. **breakHeightPad** sets an additional padding to allow more space around the break. Text will be centered vertically within the break. Default is 0.

The padding applies to the entire break and not on a line by line basis within the break.

■ PL_SetBrkRowDiv

(areaRef:L; lineWidth:F; pattern:T; plpColor:T; 4dColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ lineWidth	real	Width of the break/row divider line.
→ pattern	text	Pattern of the line.
→ plpColor	text	Color from PrintList Pro's palette for the line.
→ 4dColor	longint	Color from 4D's palette for the line.

PL_SetBrkRowDiv is used to set the line width, pattern ([transparency ratio](#)) and color of the break/row divider line which is drawn between any/all break level information and the rows of list data (detail area) that immediately follow.

lineWidth — 0 to 1. This option controls the line width of the break/row divider line. A value of 0.25 pixels should be used for hairlines. A value of 0 means that no line will be printed.

pattern — Name of the pattern ([transparency ratio](#)) for the break/row divider line. If a null string is used then no line will be printed. See the [Patterns](#) item in the [Compatibility Notes](#).

plpColor — Name of the color in [PrintList Pro's palette](#). This will be the color for the break/row divider line. If the name is not in PrintList Pro's palette or it is a null string, then **4dColor** will be used.

4dColor — 1 to 256. The color at this position in [4D's palette](#) will be used for the break/row divider line.

If **PL_SetBrkRowDiv** is not called, then the settings for the detail area row dividers, if any, will be used.

Examples

```
//Print 1 pixel wide, solid Red break/row divider line
```

```
PL_SetBrkRowDiv (eList;1;"Black";"Red";0)
```

```
//Print hairline width, solid gray break/row divider line
```

```
PL_SetBrkRowDiv (eList;0.25;"Black";"Gray";0)
```

■ PL_SetBrkRowRGBDiv

(areaRef:L; lineWidth:F; dividerRed:L; dividerGreen:L; dividerBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ lineWidth	real	Width of the break/row divider line.
→ dividerRed	longint	Divider line — Red.
→ dividerGreen	longint	Divider line — Green.
→ dividerBlue	longint	Divider line — Blue.

PL_SetBrkRowRGBDiv is used to set the line width and color of the break/row divider line which is drawn between any/all break level information and the rows of list data (detail area) that immediately follow.

This routine works in the same manner as [PL_SetBrkRowDiv](#), except it allows you to specify the colors using standard RGB values.

■ PL_SetBrkColOpt

(areaRef:L; breakLevel:L; columnNum:L; showColDivider:L; lineWidth:F; pattern:T; plpColor:T; 4dColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Number of column.
→ showColDivider	longint	Show column divider, if any, in the break .
→ lineWidth	real	Width of the horizontal break line.
→ pattern	text	Pattern of the horizontal break line.
→ plpColor	text	Color from PrintList Pro's palette for the horizontal break line.
→ 4dColor	longint	Color from 4D's palette for the horizontal break line.

PL_SetBrkColOpt is used to control the printing of column dividers and horizontal lines within the **breakLevel** for each **columnNum** and to print a horizontal line within a column for this break level.

If **showColDivider** is 1, a column divider will be printed along the right side of the column specified. The line characteristics are identical to the column divider shown in the list (detail area).

PL_SetBrkColOpt may be called to show a horizontal line at the top of the break specified in **breakLevel** in the column specified by **columnNum**. This horizontal line could be used as a subtotal line to separate a column of values from a sum or average that is calculated in the break.

If **PL_SetBrkColOpt** is not called for any columns in a given break level, then no column dividers or horizontal break lines will be printed for that break level.

showColDivider — 0 or 1:

Value	Mode
0	Don't show a column divider (default)
1	Show the column divider along the right side of the column specified in the columnNum parameter whenever a break for the break level specified in breakLevel occurs

lineWidth — 0 to 1. This option controls the line width of the horizontal break line. A value of 0.25 pixels should be used for hairlines. A value of 0 means that no line will be printed. **Double lines (typical in accounting) are supported in breaks: just use 2.0 as the lineWidth: two 0.25 point lines will be printed.**

pattern — Name of the pattern (**transparency ratio**) for the horizontal break line. If a null string is used then no line will be printed. **See the [Patterns](#) item in the [Compatibility Notes](#).**

plpColor — Name of the color in [PrintList Pro's palette](#). This will be the color for the horizontal break line. If the name is not in PrintList Pro's palette or it is a null string, then **4dColor** will be used.

4dColor — 1 to 256. The color at this position in [4D's palette](#) will be used for the horizontal break line.

Examples

```
//Break level 2, column 3, print column divider and a hairline wide, solid Blue horizontal line in column
```

```
PL_SetBrkColOpt (eList;2;3;1;0.25;"Black";"Blue";0)
```

```
//Break level 4, print the column dividers in all columns, no horizontal break lines
```

```
PL_SetBrkColOpt (eList;4;0;1;0;"";"";0)
```

■ PL_SetBrkColRGBOpt

(areaRef:L; breakLevel:L; columnNum:L; showColDivider:L; lineWidth:F; dividerRed:L; dividerGreen:L; dividerBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Number of column.
→ showColDivider	longint	Show column divider, if any, in the break.
→ lineWidth	real	Width of the horizontal break line.
→ dividerRed	longint	Horizontal break line — Red.
→ dividerGreen	longint	Horizontal break line — Green.
→ dividerBlue	longint	Horizontal break line — Blue.

PL_SetBrkColRGBOpt is used to control the printing of column dividers and horizontal lines within the **breakLevel** for each **columnNum** and to print a horizontal line within a column for this break level.

This routine works in the same manner as [PL_SetBrkColOpt](#), except it allows you to specify the colors using standard RGB values.

■ PL_SetBkHColOpt

(areaRef:L; breakLevel:L; columnNum:L; showColDivider:L; lineWidth:F; pattern:T; plpColor:T; 4dColor:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Number of column.
→ showColDivider	longint	Show column divider, if any, in the break .
→ lineWidth	real	Width of the horizontal break line.
→ pattern	text	Pattern of the horizontal break line.
→ plpColor	text	Color from PrintList Pro's palette for the horizontal break line.
→ 4dColor	longint	Color from 4D's palette for the horizontal break line.

PL_SetBkHColOpt is used to control the printing of column dividers and horizontal lines within a break header cell.

This command is identical to [PL_SetBrkColOpt](#), except **PL_SetBkHColOpt** will print the horizontal lines at the bottom of the cell instead of at the top.

If **showColDivider** is 1, a column divider will be printed along the right side of the column specified. The line characteristics are identical to the column divider shown in the list (detail area).

PL_SetBkHColOpt may be called to show a horizontal line at the bottom of the break specified in **breakLevel** in the column specified by **columnNum**.

If **PL_SetBkHColOpt** is not called for any columns in a given break level, then no column dividers or horizontal break lines will be shown for that break level.

showColDivider — 0 or 1:

Value	Mode
0	Don't show a column divider (default)
1	Show the column divider along the right side of the column specified in the columnNum parameter whenever a break for the break level specified in breakLevel occurs

lineWidth — 0 to 1. This option controls the line width of the horizontal break line. A value of 0.25 pixels should be used for hairlines. A value of 0 means that no line will be printed. **Double lines (typical in accounting) are supported in breaks: just use 2.0 as the lineWidth: two 0.25 point lines will be printed.**

pattern — Name of the pattern (**transparency ratio**) for the horizontal break line. If a null string is used then no line will be printed. See the [Patterns](#) item in the [Compatibility Notes](#).

plpColor — Name of the color in [PrintList Pro's palette](#). This will be the color for the horizontal break line. If the name is not in PrintList Pro's palette or it is a null string, then **4dColor** will be used.

4dColor — 1 to 256. The color at this position in [4D's palette](#) will be used for the horizontal break line.

■ **PL_SetBkHCoIRGBOpt**

(areaRef:L; breakLevel:L; columnNum:L; showColDivider:L; lineWidth:F; dividerRed:L; dividerGreen:L; dividerBlue:L)

Parameter	Type	Description
→ areaRef	longint	Reference of PrintList Pro object on layout.
→ breakLevel	longint	Break level number.
→ columnNum	longint	Number of column.
→ showColDivider	longint	Show column divider, if any, in the break.
→ lineWidth	real	Width of the horizontal break line.
→ dividerRed	longint	Horizontal break line — Red.
→ dividerGreen	longint	Horizontal break line — Green.
→ dividerBlue	longint	Horizontal break line — Blue.

PL_SetBkHCoIRGBOpt is used to control the printing of column dividers and horizontal lines within a break header cell.

This routine works in the same manner as [PL_SetBkHCoIOpt](#), except it allows you to specify the colors using standard RGB values.

■ PL_ProcessArrays

(callbackMethodName:T; breakArrays:Y; dataArrays:Y; useDetail:L) → error:L

Parameter	Type	Description
→ callbackMethodName	text	Name of the Computed Break callback project method.
→ breakArrays	array	Array of pointers to arrays where breaks must occur (one array for each break level).
→ dataArrays	array	Array of pointers to arrays containing the data to be processed in the callback method.
→ useDetail	longint	0 = callback method is called only on breaks 1 = callback method is called on breaks and on each row.

PL_ProcessArrays is used to set [Computed Breaks](#). No plugin area is needed, this feature is pure computing on previously sorted arrays (e.g. with [MULTI SORT ARRAY](#))

callbackMethodName — 4D project method to be called on breaks as defined by **breakArrays** and also on each row according to **useDetail**.

The callback method is called as: **callbackMethodName** (handle:L; row:L; breakLevel:L)

- **handle** is to be used as parameter 1 when calling [PL_GetBreakValue](#) for the callback
- **row** is the current row number
- **breakLevel** is the current break level, -1 for data row (if **useDetail** = 1)

breakArrays — **ARRAY POINTER** containing pointers to previously sorted arrays where breaks should occur and **callbackMethodName** be called. Local arrays are allowed. First element determines break level 1, second 2, etc.

When **breakArrays** is empty, the only break generated will be 0.

dataArrays — **ARRAY POINTER** containing pointers to arrays that will be passed by **callbackMethodName** to [PL_GetBreakValue](#) in order to retrieve the computed break values. Local arrays are allowed. First element is column 1 for [PL_GetBreakValue](#), second is 2, etc.

When **dataArrays** is empty, no calculations are performed.

useDetail — set to 0 for **callbackMethodName** to be called only with break information, set it to 1 to call the callback method with every data row and with breaks.

PL_ProcessArrays operates on a copy of the arrays, you can freely modify them in the callback.

■ PL_GetBreakValue

(handle:L; column:L; calculation:L) → value:F

Parameter	Type	Description
→ handle	text	Identification of the current <i>PL_ProcessArrays</i> call.
→ column	array	Index of the column to be processed in the <i>dataArrays</i> array set by <i>PL_ProcessArrays</i> array.
→ calculation	array	Calculation type.

PL_GetBreakValue is called from the callback method set by [PL_ProcessArrays](#) to perform usual break level processing **calculations** such as sum, minimum, etc. for the current break level (or individual row) and the specified **column**.

The **calculation** result is returned in **value** by *PL_GetBreakValue*.

handle is an identification of the current *PL_ProcessArrays* call. This is actually the process ID. Make sure not to call *PL_ProcessArrays* in the callback!

column is the index from the *dataArrays* pointer array defined by *PL_ProcessArrays*. When **column** is out of range, zero is returned.

calculation can be one of the following values:

Value	Calculation
1	Sum
2	Minimum
3	Average
4	Maximum
5	Count
6	Variance
7	Standard deviation

callbackMethodName is the only place where you can call *PL_GetBreakValue*, using the handle provided as the first parameter received by the callback method.

Calculations are performed “on the fly” after fetching each data row. So when you call *PL_GetBreakValue* on a data row (not on a break), you will get the current values for that row.

For example, using empty **breakArrays** and **useDetail=1**, calling *PL_GetBreakValue* to get the SUM (1), the command will return the sum of rows 1 through current row.

See also [Using Computed Breaks](#) and [Example 5 — Computed Breaks](#).



Examples

The examples in this section are designed to provide an overview of the use of PrintList Pro and the basic commands.

You may also wish to examine the PrintList Pro demo, for more examples on the various PrintList Pro capabilities.

Example 1 — One record current selection

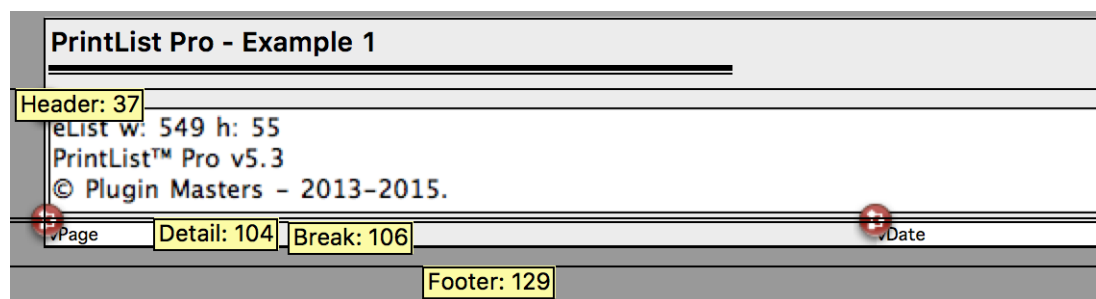
Print a list of information (First name, Last name, Salary, City, State, Zip, Country) contained in a series of seven arrays. Show column headers and print dividing lines between the columns. Do not print dividing lines between the rows.

PRINT SELECTION is used to print PrintList Pro plug-in objects. It is important therefore, that you carefully control the current selection, since a PrintList Pro plug-in object will print once for each record in the current selection.

This example illustrates a situation where the PrintList Pro object is to be printed once, so we can use the **PRINT RECORD** command regardless of the current selection.

First we need to create the layout and draw the PrintList Pro plug-in object. We'll name the object `eList`.

Our layout now looks something like this:



The project method which controls the printing is:

```
ALL RECORDS([Layouts])
```

```
OUTPUT FORM([Layouts];"Example 1")
```

```
PRINT RECORD([Layouts])
```

We'll use the [On Printing Detail](#) event to configure our PrintList Pro area.

Here is the PrintList Pro area's object method:

```
If (Form event=On Printing Detail)
  ALL RECORDS([People])
  // Create the arrays from the data
  SELECTION TO ARRAY([People]First Name;aFname:[People]Last Name;aLname:[People]Salary;\
    aSalary:[People]City;aCity:[People]State;aState:[People]Zip;aZip:[People]Country;aCountry)
  $plErr:=PL_SetArraysNam (eList;1;7;"aFname";"aLname";"aSalary";"aCity";"aState";"aZip";\
    "aCountry") // set the arrays
  If ($plErr=0)
    PL_SetHdrOpts(eList;2;0) // print headers on all pages
    PL_SetHeaders(eList;1;7;"First Name";"Last Name";"Salary";"City";"State";"Zip";"Country")
    // apply to all headers: Lucida Grande 10 point bold
    PL_SetHdrStyle(eList;0;"Lucida Grande";10;1)
    PL_SetStyle(eList;0;"Lucida Grande";9;0) // apply to all columns: Lucida Grande 9 point plain
    // print only column dividers: solid gray hairlines
    PL_SetDividers (eList;25;"Black";"Gray";0;0;"";"";0)
    // sort column 2 (Last name) in descending order
    PL_SetSort (eList;-2)
  End if
End if
```

The printed layout will appear as shown below:

PrintList Pro – Example 1

First Name	Last Name	Salary	City	State	Zip	Country
Todd	Zipnick	52,230.08	Phoenix	AZ	60090	USA
Bob	Yuderman	22,295.00	Paris		94538	France
Jeffrey	Young	49,687.96	Los Angeles	CA	94404	USA
Del	Yocam	63,118.86	San Jose	CA	95014	USA
Curtis	Wright	84,651.42	Rome		95113	Italy
William	Woodward	26,602.10	Bangkok		94107	Thailand
Ron	Wong	24,500.00	San Jose	CA	95014	USA
Ron	Wolf	25,432.96	Minneapolis	MN	95190	USA
Amy	Wohl	62,771.94	London		19004	England
Robert	Wiggins	75,296.34	Jersulaem		94306	Israel
Clair	Whitmer	27,975.08	Tapei	Ta	94105	ROC
Joel	Weiss	86,803.50	Redmond	WA	94544	USA
Peter	Watkins	92,377.74	Portland	OR	95014	USA
John	Warnock	95,805.78	Milan		94039-7900	Italy
George	Voltz	60,843.30	San Jose	CA	02144	USA
Steve	Vollum	63,119.84	Munich		97005	Germany
Larry	Tesler	39,933.04	Santa Fe	NM	95014	USA
Michael	Tchong	24,963.54	Cupertino	CA	94105	USA
Jeffrey	Tarter	65,115.12	Denver	CO	02138	USA
Harry	Sweere	60,111.24	Boston	MA	55121	USA
Karen	Sullivan	61,707.66	Portland	OR	10018	USA
Michael	Stern	28,716.94	Brooklyn	NY	19131	USA
Mitch	Stein	26,078.78	Portland	OR	94039-7900	USA
Sherwin	Steffin	70,962.78	Cupertino	CA	91302	USA
Martha	Steffen	50,354.36	Tapei	Ta	95014	ROC
Steven	Stansel	24,130.54	San Jose	CA	01867	USA
David	Smith	23,551.36	Los Angeles	CA	92670	USA
Doug	Sleeter	65,797.20	Santa Fe	NM	95014	USA
Mike	Slade	47,057.64	Boston	MA	98072	USA
Pradeep	Singh	49,758.52	Munich		98072	Germany
Rich	Shapero	61,049.10	Tapei	Ta	94501	ROC
Richard	Shaffer	26,659.92	Quincy	MA	10016	USA
Dan	Shafer	21,181.72	Ft. Worth	TX	94062	USA
Jonathan	Seybold	22,741.88	Dallas	TX	90265	USA
Patricia	Seybold	44,395.96	Milan		02109	Italy
John	Sculley	25,392.78	Redmond	WA	95014	USA
Scott	Schwartz	24,334.38	Los Angeles	CA	85203	USA
James	Sanford	61,424.44	London		95014	England
Stephen	Saltzman	62,937.56	Los Angeles	CA	97207	USA
Jonathan	Rotenberg	66,305.82	Denver	CO	02108	USA
Steve	Rosenthal	56,808.64	Boston	MA	94797	USA
Mort	Rosenthal	61,939.92	Telluride	CO	02090	USA
Brad	Romney	40,489.68	Moscow	Ru	84144	Soviet Union
Hugh	Rogovy	79,862.16	New York	NY	98121	USA
Christopher	Robert	51,063.88	San Francisco	CA	02090	USA
Thomas	Rielly	83,801.76	Denver	CO	94704	USA
Jim	Rickard	68,025.72	Baghdad		94303	Iraq
Rick	Richardson	25,649.54	New York	NY	10172	USA
Cheryl	Rhodes	89,026.14	Milan		94062	Italy
Jackie	Rae	50,960.00	Munich		94538	Germany
Michelle	Preston	31,782.38	Phoenix	AZ	10004	USA
Pete	Peterson	70,460.04	Santa Fe	NM	84057	USA
Carol	Person	28,636.58	Ft. Worth	TX	94107	USA
Rachel	Parker	53,586.40	Dallas	TX	94025	USA
Ray	Palkovic	60,667.88	San Antonio	TX	06904	USA
Kazue	Osugi	88,573.38	Detroit	MI	94705	USA
Bobby	Orbach	23,286.76	Portland	OR	10003	USA
David	O'Connor	65,682.54	Madison	WI	02142	USA

Example 2 — Multiple record current selection

Print a list of company names, showing all the people who work for each company. This example will illustrate the use of PrintList Pro with multiple records in the current selection.

For purposes of illustration, the company names will be printed directly from the `[Companies]` table, and the `[People]` information will be printed from a series of arrays using PrintList Pro.

Create the layout and draw the PrintList Pro plug-in object. This process is substantially the same as that explained in the first example, with one exception — we are going to include a field directly on the layout which will print information alongside the PrintList Pro object.

Our layout is illustrated below:

PrintList Pro - Example 2			
Company	Employees/Salaries		
Header: 50	eList w: 291 h: 55 PrintList™ Pro v5.3 © Plugin Masters - 2013-2015.		
[Companies]Company Name			
Page	Detail: 120	Break: 120	Date
Footer: 145			

The only difference between this example and [Example 1](#) is that we are printing out multiple records.

Since the PrintList Pro commands are executed in the `On Printing Detail` event of the **PRINT SELECTION** command we can change the People arrays “on the fly”.

The `[Companies]` file is in a One-to-Many relationship with the `[People]` file, and the links are automatic.

As each `[Companies]` record is printed a new selection of `[People]` records is created.

This `[People]` selection is stored in arrays and printed.

The controlling project method is:

```
ALL RECORDS([Companies])
OUTPUT FORM([Companies];"Example 2")
PRINT SELECTION([Companies])
```

The PrintList Pro area's object method is:

```
If (Form event=On Printing Detail)
  // Create the arrays - the current selection of [People] changes with each new record
  SELECTION TO ARRAY([People]First Name;aFname;[People]Last Name;aLname;[People]Salary; aSalary)
  $plErr:=PL_SetArraysNam(eList;1;3;"aFname";"aLname";"aSalary") //set the arrays
  If ($plErr=0)
    PL_SetHdrOpts(eList;1;0) //print headers at the top of the area only
    PL_SetHeaders(eList;1;3;"First Name";"Last Name";"Salary") //set headers
    // apply to all headers: Lucida Grande 10 point bold
```

```
PL_SetHdrStyle(eList;0;"Lucida Grande";10;1)
PL_SetStyle(eList;0;"Lucida Grande";9;0) //apply to all columns: Lucida Grande 9 point plain
//format column 3, right justified header and column
PL_SetFormat (eList;3;"$###,###,###.00";3;3)
//solid black hairline frame/hdr line
PL_SetFrame (eList;0.25;"Black";"Black";0;0.25;"Black";"Black";0)
End if
End if
```

A portion of our resulting printout appears below:

PrintList Pro – Example 2

Company	Employees/Salaries		
Addison-Wesley Publishing Co.,	First Name	Last Name	Salary
	Mike	Erickson	\$1,000.00
	Steven	Stansel	\$24,130.54
Apple Computer, Inc.	First Name	Last Name	Salary
	Barbara	Anderson	\$68,484.36
	Samir	Arora	\$63,847.98
	Randy	Battat	\$26,898.06
	Bill	Coldrick	\$42,369.32
	Debi	Coleman	\$71,092.14
	Moira	Cullen	\$69,613.32
	David	Eyes	\$28,964.88
	Jonathan	Fader	\$30,048.76
	Jim	Floyd	\$25,055.66
	Linda	Glish	\$63,990.08
	Russ	Havard	\$20,953.38
	Mike	Homer	\$46,056.08
	Barbara	Krause	\$41,832.28
	Tim	Kreps	\$80,500.14
	Jon	Magill	\$59,917.20
	John	Sculley	\$25,392.78
	Doug	Sleeter	\$65,797.20
	Martha	Steffen	\$50,354.36
	Larry	Tesler	\$39,933.04
Peter	Watkins	\$92,377.74	
Ron	Wong	\$24,500.00	
National Apple Professional In	First Name	Last Name	Salary
	Mike	Bailey	\$83,410.74
CompuServe	First Name	Last Name	Salary
	Sharon	Jones	\$90,019.86
Affinity Microsystems, Ltd.	First Name	Last Name	Salary
	Rick	Barron	\$59,908.38
San Jose Mercury News	First Name	Last Name	Salary
	Jim	Bartimo	\$21,418.88
	Rory	O'Connor	\$63,409.92
	Ron	Wolf	\$25,432.96
Think Educational Software, In	First Name	Last Name	Salary
	Gregory	Berkin	\$62,239.80
Raw Fish Systems	First Name	Last Name	Salary
	Steve	Bobker	\$81,937.80
Bogas Productions	First Name	Last Name	Salary
	Ed	Bogas	\$51,108.96
Radius, Inc.	First Name	Last Name	Salary

Example 3 — Adding a total line to the list

Print the same list of company names and people as [Example 2](#), and add a total line after the end of each list of people. The total line will contain a sum of the salaries for all the people working for that company.

A total line requires us to use PrintList Pro's Break Level commands.

While most break levels require the list to be sorted, a total line is the exception. The total line is configured by passing 0 for the

break level parameter.

The object method for the PrintList Pro area in Example 2 has been modified to include the Break Level calls needed as shown below:

```

If (Form event=On Printing Detail)
  //Create the arrays - The current selection of [People] changes with each new record
  SELECTION TO ARRAY([People]First Name;aFName:[People]Last Name;aLName:[People]Salary; aSalary)
  $plErr:=PL_SetArraysNam(eList;1;3;"aFName";"aLName";"aSalary") //set the arrays
  If ($plErr=0)
    PL_SetHdrOpts(eList;1;0) //print headers at the top of the area only
    PL_SetHeaders(eList;1;3;"First Name";"Last Name";"Salary") //set headers
    //apply to all headers: Lucida Grande 10 point bold
    PL_SetHdrStyle(eList;0;"Lucida Grande";10;1)
    PL_SetStyle(eList;0;"Lucida Grande";9;0) //apply to all columns: Lucida Grande 9 point plain
    //format column 3, right justified header and column
    PL_SetFormat(eList;3;"###,###,###.00";3;3) //solid black hairline frame/hdr line
    PL_SetFrame (eList;0.25;"Black";"Black";0;0.25;"Black";"Black";0)
    PL_SetWidths(eList;1;3;80;80;100) //set the column widths
    //Configure the total line
    PL_SetBrkText(eList;0;3;"Sum";0;0) //calculate the sum in the total line
    PL_SetBrkHeight(eList;0;1;4) //add some padding to the total line
    PL_SetBrkColOpt(eList;0;3;0;0.25;"Black";"Black";0) //draw a line above the total
  End if
End if

```

The total line now appears in the list as is shown below:

PrintList Pro – Example 3

Company	Employees/Salaries		
Addison-Wesley Publishing Co.,	First Name	Last Name	Salary
	Mike	Erickson	\$1,000.00
	Steven	Stansel	\$24,130.54
			\$25,130.54
Apple Computer, Inc.	First Name	Last Name	Salary
	Barbara	Anderson	\$68,484.36
	Samir	Arora	\$63,847.98
	Randy	Battat	\$26,898.06
	Bill	Coldrick	\$42,369.32
	Debi	Coleman	\$71,092.14
	Moira	Cullen	\$69,613.32
	David	Eyes	\$28,964.88
	Jonathan	Fader	\$30,048.76
	Jim	Floyd	\$25,055.66
	Linda	Glish	\$63,990.08
	Russ	Havard	\$20,953.38
	Mike	Homer	\$46,056.08
	Barbara	Krause	\$41,832.28
	Tim	Kreps	\$80,500.14
	Jon	Magill	\$59,917.20
	John	Sculley	\$25,392.78
	Doug	Sleeter	\$65,797.20
	Martha	Steffen	\$50,354.36
	Larry	Tesler	\$39,933.04
Peter	Watkins	\$92,377.74	
Ron	Wong	\$24,500.00	
		\$1,037,978.76	
National Apple Professional In	First Name	Last Name	Salary
	Mike	Bailey	\$83,410.74
			\$83,410.74
CompuServe	First Name	Last Name	Salary
	Sharon	Jones	\$90,019.86
			\$90,019.86
Affinity Microsystems, Ltd.	First Name	Last Name	Salary
	Rick	Barron	\$59,908.38
			\$59,908.38
San Jose Mercury News	First Name	Last Name	Salary
	Jim	Bartimo	\$21,418.88
	Rory	O'Connor	\$63,409.92
	Ron	Wolf	\$25,432.96
		\$110,261.76	
Think Educational Software, In	First Name	Last Name	Salary
	Gregory	Berkin	\$62,239.80
			\$62,239.80

Example 4 — Break Level Processing

Print the same list of information shown in [Example 1](#) and add some break level information for all the people in the same city. The break will show a sum, minimum, average, maximum and **standard deviation** of the people's salaries in each city.

Labels for each of the calculations will be printed in the adjacent column to salary. The break will also show the number of people in the city and the city name.

A custom calculation is included along with the other calculations to calculate an end of year bonus based on 5% of the average salary.

The list is sorted by country, state, city and last name. This allows the suppression of repeated values for each of these columns. In order to set break information for the city, we must configure break level 3 because the city array is 3rd in the sort order.

Each of the calculations mentioned will be printed on a separate line, so the height of the break is set to 6 lines. Carriage returns are inserted between the labels and calculations so that each will start on a new line.

The code for the callback function follows the object method to the PrintList Pro object. In addition, two header lines are shown to demonstrate the ability for multiple lines in a header.

The object method for the PrintList Pro plug-in area in [Example 1](#) has been modified to include the [Break Level](#) calls needed as shown below:

```

If (Form event=On Printing Detail)
  // Create the arrays from the data
  ALL RECORDS([People])
  SELECTION TO ARRAY([People]First Name;aFName:[People]Last Name;aLName:[People]Salary;\
aSalary:[People]City;aCity:[People]State;:[People]Country;aCountry)
  // Set the arrays
  $plErr:=PL_SetArraysNam (eList;1;6;"aFName";"aLName";"aSalary";"aCity";"aState"; "aCountry")
  If ($plErr=0)
    PL_SetHdrOpts(eList;2;0) // print headers on all pages
    PL_SetHeight(eList;2;4;1;0) // 2 hdr lines, 4 hdr pad, 1 row line, 2 row pad
    PL_SetHeaders(eList;1;6;"First Name";"Last Name";"Salary";"City";"State";"Country")
    PL_SetHdrStyle(eList;0;"Lucida Grande";10;1) // all headers: Lucida Grande 10 point bold
    PL_SetStyle(eList;0;"Lucida Grande";9;0) // all columns: Lucida Grande 9 point plain
    PL_SetFormat(eList;3;"$###,###,###.00";3;3) // format column 3, right justified header and column
    PL_SetFrame(eList;0.25;"Black";"Black";0;0.25;"Black";"Black";0) // print solid black hairline frame
    PL_SetWidths(eList;1;6;76;80;89;79;80;48) // set the column widths
    PL_SetBackClr(eList;"Light Gray";0;"White";0)
    // Sort by Country (descending), State, City, and Last Name (descending)
    PL_SetSort (eList;-6;5;4;-2)
    // Break level Configuration
    PL_SetRepeatVal(eList;0;1) // suppress repeating values in all columns
    PL_SetBrkFunc(eList;"Break Function") // set the callback function
    PL_SetBrkRowDiv(eList;0.25;"Black";"Black";0) // print a Break/Row divider
    // Configure break level 3, city
    PL_SetBrkHeight(eList;3;6;4) // print 6 lines for break level 3

```



```

//Print the calculation labels in the column to the left of the salaries
PL_SetBrkText (eList;3;2;"Sum\rAverage\rMinimum\rMaximum\rStandard Dev\rBonus";0;3)
PL_SetBrkStyle(eList;3;2;"Lucida Grande";9;1) //make the labels bold
//Print the Sum, Minimum, Average, Maximum, Standard Deviation and Bonus for salaries
PL_SetBrkText (eList;3;3;"\Sum\r\Minimum\r\Average\r\Maximum\r\Dev\r\Function";0;0)
// Show the number of people in this city
PL_SetBrkText(eList;3;5;"\r\Count people in \BreakValue";1;2)
PL_SetBrkStyle(eList;3;5;"Lucida Grande";10;3) // make the city count info 10 points bold
PL_SetBrkColOpt(eList;3;3;0;0.25;"Black";"Black";0) //print a subtotal line in the salary column
End if
End if

```

The **Break Function** callback method for the “Bonus” custom calculation is as follows:

```

C_LONGINT ($1;$2) //break level, column
C_TEXT ($2;$3) //column format
C_LONGINT($4;$5) //start row, end row
C_TEXT($0) // custom calculation result to print
C_LONGINT($i;$count)
C_REAL($result;$average)
// Calculate the average
For ($i;$4;$5)
    $result:=$result+aSalary{$i}
End for
$count:=$5-$4+1
$average:=$result/$count
$result:=Int($average*0,05) //5% rounded
$0:=String($result;$3)

```

Here is a sample of the list containing the breaks:

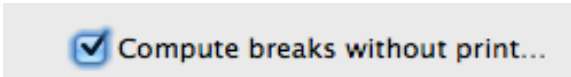
PrintList™ Pro Demo: Break-Level Processing.

First Name	Last Name	Salary	City	State	Country
Bill	Goodhew	\$23 275,98	Podunk	AR	USA
Stan	Getz	\$60 956,00	Podunk	AR	USA
Yogen	Dalal	\$41 491,24	Podunk	AR	USA
Bill	Coldrick	\$42 369,32	Podunk	AR	USA
	Sum	\$168 092,54			
	Minimum	\$23 275,98			
	Average	\$42 023,13			4 people in Podunk
	Maximum	\$60 956,00			
	Standard Dev	\$13 325,83			
	Bonus	\$2 101,00			
Todd	Zipnick	\$52 230,08	Phoenix	AZ	USA
Michelle	Preston	\$31 782,38	Phoenix	AZ	USA
John	Markoff	\$20 416,34	Phoenix	AZ	USA
	Sum	\$104 428,80			
	Minimum	\$20 416,34			
	Average	\$34 809,60			3 people in Phoenix
	Maximum	\$52 230,08			
	Standard Dev	\$13 163,11			
	Bonus	\$1 740,00			
Michael	Tchong	\$24 963,54	Cupertino	CA	USA
Sherwin	Steffin	\$70 962,78	Cupertino	CA	USA
Jonathan	Fader	\$30 048,76	Cupertino	CA	USA
Mary	Evslin	\$46 685,24	Cupertino	CA	USA
Moira	Cullen	\$69 613,32	Cupertino	CA	USA
Ed	Cheffetz	\$53 588,36	Cupertino	CA	USA
	Sum	\$295 862,00			
	Minimum	\$24 963,54			
	Average	\$49 310,33			6 people in Cupertino
	Maximum	\$70 962,78			
	Standard Dev	\$17 654,11			
	Bonus	\$2 465,00			
Jeffrey	Young	\$49 687,96	Los Angeles	CA	USA
David	Smith	\$23 551,36	Los Angeles	CA	USA
Scott	Schwartz	\$24 334,38	Los Angeles	CA	USA
Stephen	Saltzman	\$62 937,56	Los Angeles	CA	USA
Bill	Gates	\$26 287,52	Los Angeles	CA	USA
Bruce	Davis	\$57 186,92	Los Angeles	CA	USA
Raines	Cohen	\$29 784,16	Los Angeles	CA	USA
Paul	Brainerd	\$59 952,48	Los Angeles	CA	USA
Lofty	Becker	\$85 627,50	Los Angeles	CA	USA
	Sum	\$419 349,84			
	Minimum	\$23 551,36			
	Average	\$46 594,42			9 people in Los Angeles
	Maximum	\$85 627,50			
	Standard Dev	\$20 581,37			
	Bonus	\$2 329,00			
Christopher	Robert	\$51 063,88	San Francisco	CA	USA
Regis	McKenna	\$57 916,04	San Francisco	CA	USA
Bob	Leff	\$87 341,52	San Francisco	CA	USA
Mike	Kramer	\$21 337,54	San Francisco	CA	USA
Ed	Bogas	\$51 108,96	San Francisco	CA	USA

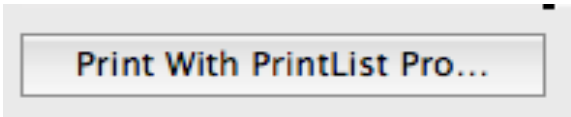
Example 5 — Computed Breaks

Computed breaks are a powerful array utility module, which does not require printing.

The demonstration database includes an example under PrintList Pro > Configuration options... Check the “Compute breaks without print” box:



then click the “Print With PrintList Pro” button:



We will perform a similar action, using the same arrays as in [Example 4](#), then call [PL_ProcessArrays](#) and [PL_GetBreakValue](#) to retrieve the results and build a break summary.

We will create a text variable containing all break results without printing anything, then copy it into the pasteboard. We could also display the text variable like in the demonstration database, or use it otherwise.

```

C_TEXT(vBrkComputeResult)
ARRAY TEXT(TbreakText_R;0)
C_LONGINT($i)
// Create the arrays from the data
ALL RECORDS([People])
SELECTION TO ARRAY([People]First Name;aFname:[People]Last Name;aLname:[People]Salary;\
    aSalary:[People]City;aCity:[People]State;aState:[People]Country;aCountry)
// Sort by Country (descending), State, City, and Last Name (descending)
MULTI SORT ARRAY(aCountry;<;aState;>;aCity;>;aLname;<;aFname;aSalary)
// Declare the "data" array pointing on the actual arrays on which to perform calculations in our callback
// (we include all arrays in case we'll need them someday, even though we will only use aSalary here)
ARRAY POINTER($dataArray;0)
APPEND TO ARRAY($dataArray;->aCountry)
APPEND TO ARRAY($dataArray;->aState)
APPEND TO ARRAY($dataArray;->aCity)
APPEND TO ARRAY($dataArray;->aLname)
APPEND TO ARRAY($dataArray;->aFname)
APPEND TO ARRAY($dataArray;->aSalary)
// Declare the "break" array pointing on the arrays where to catch breaks in our callback
// according to our MULTI SORT ARRAY above (sorted arrays)
ARRAY POINTER($breakArrays;0)
APPEND TO ARRAY($breakArrays;->aCountry)
APPEND TO ARRAY($breakArrays;->aState)
APPEND TO ARRAY($breakArrays;->aCity)

```

```

APPEND TO ARRAY($breakArrays;->aLName)
//Now we process the arrays using the "PlpComputeBreak" project method for computing
if (PL_ProcessArrays ("PlpComputeBreak";$breakArrays;$dataArray;0)=0)
    //4th parameter is 0 for each break or 1 for each row
    //Now TbreakText_R has been populated by PlpComputeBreak: concatenate breaks into a text variable
    vBrkComputeResult:=""
    For ($i;1;Size of array(TbreakText_R))
        vBrkComputeResult:=vBrkComputeResult+TbreakText_R{$i}+Char(Carriage return)
    End for
    SET TEXT TO PASTEBBOARD(vBrkComputeResult) //or do something else with vBrkComputeResult
End if

```

Here is our *PlpComputeBreak* callback method.

```

C_LONGINT($1;$2;$3) //handle, row, break level ($breakArrays)
C_LONGINT($dataArray) //position of the array to feed computed break in $dataArray
    //from the calling method (third parameter to PL_ProcessArrays)
C_TEXT($format;$breakText)
if ($3>=0) //is it a break
    $dataArray:=6 //aSalary
    $format:="$###,###,###.00"
    Case of
        : ($3=1) //break on aCountry
        : ($3=2) //break on aState
        : ($3=3) //break on aCity
        // $breakText:="Hello, I am the break at level "+String($3)+" after row "+String($2) \
        // +(Char(Carriage return)*2) //we could do this to use $2
        // Concatenate the text for the current break
        $breakText:=""
        $breakText:=$breakText+"There are "+String(PL_GetBreakValue ($1;$dataArray;5)) +" people in"\
            +aCity{$2-1}+(Char(Carriage return)*2)
        $breakText:=$breakText+"Sum: "+String(PL_GetBreakValue ($1;$dataArray;1);$format)\
            +Char(Carriage return)
        $breakText:=$breakText+"Minimum: "+String(PL_GetBreakValue ($1;$dataArray;2);$format)\
            +Char(Carriage return)
        $breakText:=$breakText+"Average: "+String(PL_GetBreakValue ($1;$dataArray;3);$format)\
            +Char(Carriage return)
        $breakText:=$breakText+"Maximum: "+String(PL_GetBreakValue ($1;$dataArray;4); $format)\
            +Char(Carriage return)
        $breakText:=$breakText+"Standard Dev: "+String(PL_GetBreakValue ($1;$dataArray;7);$format)\
            +Char(Carriage return)

```

```
// We perform the 5% calculation right here:
$breakText:=$breakText+"Bonus:"+String(Round(PL_GetBreakValue($1;$dataArray;3)*0,05;0);\
    $format)+Char(Carriage return)
$breakText:=$breakText+"-----"+Char(Carriage return)
APPEND TO ARRAY(TbreakText_R;$breakText)
: ($3=4) // break on last name
End case
Else // this is a row because $3 < 0
    // do something
End if
```

Here is the resulting text:

There are 4 people in Podunk

Sum: \$168 092,54

Minimum: \$23 275,98

Average: \$42 023,13

Maximum: \$60 956,00

Standard Dev: \$13 325,83

Bonus: \$2 101,00

There are 3 people in Phoenix

Sum: \$104 428,80

Minimum: \$20 416,34

Average: \$34 809,60

Maximum: \$52 230,08

Standard Dev: \$13 163,11

Bonus: \$1 740,00

There are 6 people in Cupertino

Sum: \$295 862,00

Minimum: \$24 963,54

Average: \$49 310,33

Maximum: \$70 962,78

Standard Dev: \$17 654,11

Bonus: \$2 466,00

(etc.)



PrintList Pro Constants

PLP Colors		PLP Justification	
PL White	White	PL Just Default	0
PL Black	Black	PL Just Left	1
PL Magenta	Magenta	PL Just Center	2
PL Red	Red	PL Just Right	3
PL Cyan	Cyan	PLP Font Style	
PL Green	Green	PL Plain	0
PL Blue	Blue	PL Bold	1
PL Yellow	Yellow	PL Italic	2
PL Gray	Gray	PL Underline	4
PL Light gray	Light gray	PL Outline	8
PL Use 4D palette color		PL Shadow	16
PLP Patterns		PL Condensed	32
PL White Pattern	White	PL Extended	64
PL Black pattern	Black	PLP Break Levels	
PL Gray pattern	Gray	PL Break Level 1	1
PL Light gray pattern	Light gray	PL Break Level 2	2
PL Dark gray pattern	Dark gray	PL Break Level 3	3
PLP Command Results		PL Break Level 4	4
PL Registration Failed	1	PL Break Level 5	5
PL Registration Passed	0	PL Break Level 6	6
PL SetArrays Passed	0	PL Break Level 7	7
PL Not an array	1	PL Break Level 8	8
PL Wrong type of array	2	PL Break Level 9	9
PL Wrong number of rows	3	PL Sum	\Sum
PL Maximum number of arrays exc	4	PL Average	\Average
PL Not enough memory	5	PL Minimum	\Minimum
PL SetFile Passed	0	PL Maximum	\Maximum
PL Not a file	6	PL Count	\Count
PL Wrong 4D version	10	PL Variance	\Var
PL Arrays have been set	11	PL Deviation	\Dev
PL Fields have been set	12	PL Break Value Insertion	\BreakValue
PL SetFields Passed	0	PL Custom Calculation	\Function
PL Not a field	7		
PL Wrong field type	8		

<u>PL Maximum fields exceeded</u>	9
<u>PL Save Data Passed</u>	0
<u>PL Save Data Failed</u>	1
<u>PL Restore Data Passed</u>	0
<u>PL Restore Data Failed</u>	1

PLP Break Values

<u>PL Break Value Sum</u>	1
<u>PL Break Value Minimum</u>	2
<u>PL Break Value Average</u>	3
<u>PL Break Value Maximum</u>	4
<u>PL Break Value Count</u>	5
<u>PL Break Value Variance</u>	6
<u>PL Break Value Deviation</u>	7

PLP Options

<u>PL No Headers</u>	0
<u>PL First Page Headers</u>	1
<u>PL Print All Headers</u>	2
<u>PL Suppress Pixel Width</u>	0
<u>PL Print Pixel Width</u>	1
<u>PL Print Array</u>	0
<u>PL Hide Array</u>	1
<u>PL Print All Records</u>	-1
<u>PL Print Page Breaks</u>	1
<u>PL Suppress Page Breaks</u>	0
<u>PL Print Last Page Break</u>	1
<u>PL Suppress Last Page Break</u>	0
<u>PL Print Repeated Values</u>	0
<u>PL Suppress Repeated Values</u>	1
<u>PL Print Column Divider</u>	0
<u>PL Suppress Column Divider</u>	1
<u>PL Suppress Detail Area</u>	1



Text Style Tags

If the **attributed** option has been set in [PL_SetFormat](#), special tags can be used in any text contained in a PrintList Pro area to print multi-styled characters.

These tags work just like HTML tags: `<tag>styled text</tag>`.

Style	Tag
Bold	<code></code>
Italic	<code><i></code>
Underline	<code><u></code> or <code><ins></code>
Strike-through	<code></code>
Set font size to # points	<code><s #></code>
Increase font size by # quarters (1/4) of current size	<code><s +#></code>
Decrease font size by # quarters (1/4) of current size	<code><s -#></code>
Set font by name	<code><f "font name"></code> (needs to be quoted if the name contains more than one word)
Set color (any format can be used, e.g. <code><c 0xFFFF0000></code> <code><c 1.0,0,0></code> <code><c P123></code> <code><c dark orange></code>)	<code><c color name></code>

4D internal format for styled text is stored as `` where the style attributes used by PrintList Pro are:

- font-family
- font-size
- font-weight (bold/normal)
- font-style (italic/normal)
- text-decoration (underline/line-through/none)
- color (#RRGGBB)
- background-color (#RRGGBB)

It is also possible to set the format as attributed, and specify the style attributes using the **PL_SetFormat** command.

Example for an longint column:

```
C_TEXT($format)
```

```
$format:="<c blue>+#####</c>;<i><c red>-.#####</c></i>;<s+1><c green><b>ZERO</b></c></s>"
```

```
PL_SetFormat ($area;1;$column;0;0;1;1;1;0)
```

```
//Right-aligned (it is a number; default of zero will use 2), auto-sized height (because ZERO is bigger),
```

```
//attributed, with "compatible" line spacing and default vertical alignment
```




Copyrights and Trademarks

All trade names referenced in this document are the trademark or registered trademark of their respective holders.

PrintList Pro is copyright Plugin Masters SAS and exclusively published worldwide by [e-Node](#).

4th Dimension, 4D and 4D Server are trademarks of [4D SAS](#).

Windows, Excel and Vista are trademarks of [Microsoft Corporation](#).

Macintosh, MacOS and MacOS X are trademarks of [Apple, Inc.](#)

PrintList Pro manual originally written by [Pat Bensky](#).

14

Index

4D Server	13	Colors	28, 46, 48, 51
64-bit	7	Column dividers	100
%PrintListPro	34	Commands	20
A		Command syntax	20
AreaList Pro	23	Compatibility	7
Array mode	77	Compiler_PLP	21
Attributed	23, 121	Component	22, 35
Average	90	Computed Break	102
B		Computed Breaks	22, 72, 86, 115
Border	65	Constants	24, 119
Borders	31	Count	90
Break	70, 80	Custom Calculations	70, 83
Break header	91, 92	D	
Break Header	70	Demo mode	8
Break level	58	Demonstration mode	12
Break Level	111, 112	Divider	54, 98
Break Level Processing	80	Dividers	53, 99
C		Dividing Lines	30
Calculated column	71, 79	Double lines	23
Calculated columns	75, 76	E	
Calculated values	72	Ellipsis	43
Calculations	83	EEmail notification	18
Callback	31, 67, 69, 92, 102, 112	End of Page	69
Callback method	79, 103, 116	F	
Callback Method	77	Field mode	76
Color	44, 61, 83, 94, 95, 98, 99, 101	Field Printing	73

Fields	75
Final keys	16
Font	43, 50, 59, 93
Format	37
Formatting	25
Forums	7
Frame	30, 55, 56, 66
Frames	31

G

GDI	23, 58
-----------	--------

H

Header	36, 57
Headers	25, 39, 41
Header separator	55, 56
Height	97
Hide	85
Hide columns	58
Horizontal lines	99, 100

I

Icon	31
Icons	63
Installation	8
Instant activation	16

J

Justification	37
---------------------	----

L

License server	16
License types	9, 10
Line width	99, 101

M

Machine ID	14
Master key	16
Maximum	90
Merged	14
Merged licenses	9
Minimum	90
Multiple Lines	29, 84
Multiple record	107
Multi-style	23
Multi-styled	121

N

Number format	122
Number of lines	57, 97

O

OEM	10
Omitted parameters	20
One record	104
Online instant activation	11, 13, 15, 16, 34

P

Padding	57, 97
Page break	87
Page Breaks	85
Palette	28
Parameters	20
Partner	10
Pattern	55, 98, 99, 101
Patterns	21
Picture Arrays	30
Picture Library	21
PL_AddColumn	22, 35
PL_GetBreakValue	103
PL_GetVersion	67
PL_Load	68
PL_Register	15, 32

PL_Save	68	PL_SetHdrStyle	41
PL_SetArraysNam	35	PL_SetHeaders	36
PL_SetBackClr	46	PL_SetHeight	57
PL_SetBackRGBColor	47	PL_SetMiscOptions	43
PL_SetBkHColOpt	100	PL_SetPageBreak	87
PL_SetBkHColor	95	PL_SetPageProc	67
PL_SetBkHColRGBOpt	101	PL_SetRepeatVal	89
PL_SetBkHFunc	92	PL_SetRGBDividers	54
PL_SetBkHHeight	97	PL_SetRGBFrame	56
PL_SetBkHRGBColor	96	PL_SetRowColor	51
PL_SetBkHStyle	93	PL_SetRowRGBColor	52
PL_SetBkHText	91	PL_SetRowStyle	50
PL_SetBrkColOpt	99	PL_SetSort	58
PL_SetBrkColor	94	PL_SetStyle	43
PL_SetBrkColRGBOpt	100	PL_SetWidths	41
PL_SetBrkFunc	92	Plugin Area	19
PL_SetBrkHeight	97	Printing Records	25
PL_SetBrkOpts	88	PrintList Pro Area	19
PL_SetBrkOrder	88	Print records	74
PL_SetBrkRGBColor	95		
PL_SetBrkRowDiv	98	R	
PL_SetBrkRowRGBDiv	98	Register	12, 32
PL_SetBrkStyle	93	Registering	11
PL_SetCalcCall	79	Registering Server licenses	13
PL_SetCellBorder	65	Regular licenses	9
PL_SetCellColor	61	Remote mode	13
PL_SetCellFrame	66	Repeated values	89
PL_SetCellIcon	63	Repeated Values	83
PL_SetCellRGBColor	62	RGB 22, 28, 45, 47, 49, 52, 54, 56, 62, 65, 66, 95, 96, 98, 100, 101	
PL_SetCellStyle	59		
PL_SetColBackColor	48	S	
PL_SetColOpts	58	Separator Lines	30
PL_SetDividers	53	Single-user license	10
PL_SetFields	75	Sort	58, 88
PL_SetFile	74	Sorting	73
PL_SetForeClr	44	Sorting Arrays	25
PL_SetForeRGBColor	45	Standard deviation	23, 90
PL_SetFormat	37	Style	43, 50, 59, 60, 83, 93
PL_SetFrame	55		
PL_SetHdrOpts	42		

Styled text	21, 26
Styles	26
Style Tags	121
Sum	90

T

Text Overflow	83
Time	73
Total line	109
Two-dimensional array	24

U

Updates	9
Upgrading	21

V

Variable Height	29, 85
Variance	23, 90
Version	67

W

Width	41
Widths	30
Wrap	57

X

XML	23, 68
-----------	--------