11.4

AreaList Pro™

User Manual

Version 11.4

Plugin Masters
177 cours de l'Argonne
33000 Bordeaux
France

PLUGIN MASTERS

pluginmasters.net

Manual revision: 30 October 2023

Contents

1 About AreaList Pro

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

AreaList Pro is a plugin for 4D which makes it possible for you to create and print dynamic, feature-rich scrolling list areas on 4D forms. You have a great deal of control over these areas – you can control many different options such as:

- Display either fields or arrays
- The appearance of the list: properties such as row coloring, text styling, row and column strokes
- What a user can or cannot do (for example, you may want to allow them to re-order the rows in one list, but not in another)
- Dragging and Dropping: specify where list items can be dragged and dropped to and from (e.g. from one area to another, or within a list, or from an external file)
- Display hierarchical lists and grids
- Specify whether data can or cannot be edited
- Print an AreaList Pro area, using the included PrintList Pro commands (previously available as a separate plugin)
- … and lots more!

Technical Details

Compatibility Information

AreaList Pro version 11 is compatible with 4D v18 and over for MacOS (Intel and Silicon, signed and notarized) and Windows (64-bit mode).It is not compatible with 32-bit machines (64-bit required).

It requires macOS Mojave (10.14) or higher and Windows 10 or Windows Server 2012 R2 or later.

You do not have to update all your AreaList Pro areas and code immediately. Previous versions commands are still here and will work with AreaList Pro version 11 with little or no change in your code. See the v9 manual for legacy commands.

Technical Support

Technical support for AreaList Pro is provided electronically via our online support reporting system.

You are encouraged to use the online web forums.

What's new in version 11

2 Installation

Installing the plugin

AreaList 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 **ALP.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 AreaList Pro in Demo mode

You can use AreaList 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 web site.

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

Licensing

Like all Plugin Masters plug-ins, AreaList 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 licenses** are used for applications that are opened with 4D Standalone or with 4D Server, either in interpreted or compiled mode, macOS or Windows (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). This mode also applies to structures run with 4D Runtime (unserialized 4D).

  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 Plugin Masters 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 v19 to 4D v20 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 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 Plugin Masters, including AreaList Pro with Print option, SuperReport 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 Plugin Masters, including AreaList Pro with Print option, SuperReport 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 or Runtimes.

  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 Partner license.

Note: All Print features need a specific license option.

Registering your AreaList 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 registration 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 AL_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 Plugin Masters.
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:=AL_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 registrered 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.

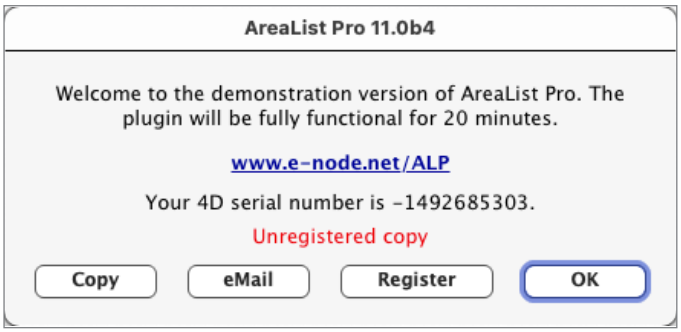
To trigger this display and enable your users to register without actually calling a command or setting up an area, you can also pass an **empty** string to AL_Register and the dialog will show:

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

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

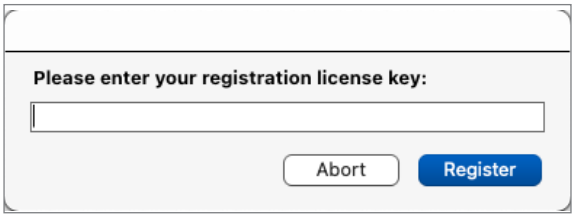
Retrieving the serial / machine information

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 Plugin Masters 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 AL_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 an AreaList Pro command other than ***AL_Register*** with a non-empty parameter
- Calling ***AL_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 ***AL_Register*** with an empty string in the **On Server Startup** base method:

```
C_LONGINT ($result)
$result:=AL_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 can 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 connected to the same server machine) using the following call:

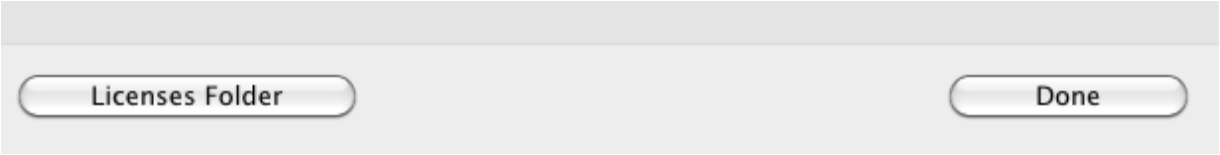
```
C_LONGINT($machineID)
$machineID:= AL_GetAreaLongProperty (0;"mach")
```

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 > License Manager,** then click the **Licenses Folder** button:



The text file must be called “ALP11.license4Dplugin” and be a plain text type file.

Just paste all your licenses for AreaList Pro v11.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 AL_Register

1. Open the **On Startup** database method

2. Call the AL_Register function with your registration key - for example:

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

If you have several licenses for different 4D setups you can call ***AL_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 AreaList Pro will check for valid licenses from this document as a first action before anything else (including checking any ***AL_Register*** command).

If a valid license is included into the “ALP11.license4Dplugin” document any calls to ***AL_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 ***AL_Register*** command.

Online registration

AreaList 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 want 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 AL_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-ALP9-xyz".

Passing a master key as the first parameter to ***AL_Register*** when the plugin has not been previously registered by any of the methods above will result in a connection attempt to the 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 registration, text file, register button or AL_Register with a final key), and if ***AL_Register*** receives a master key in its first parameter, it will recognize it a such, then:

1. Connect to the 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 ***AL_Register***

5. If the master key is valid, receive its final key from the license server then register itself (writing into the license file).

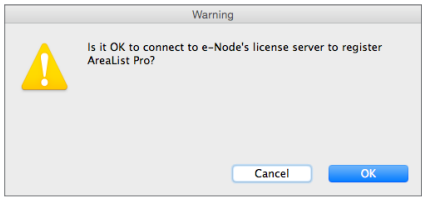
Note: if a final key has already been issued for this serial / machine ID using this master key, it is simply resent.

User interface

In addition, AL_Register second parameter allows optional settings regarding the user interface in the online registration process.

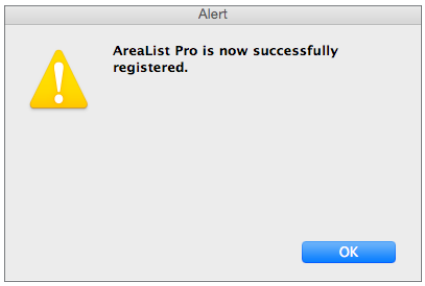
```
C_LONGINT ($result)
$result:=AL_Register ("Master key";0 ?+1 ?+2 ?+3;"youremail@something.com")  // all dialogs
```

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 AL_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: ALP11 license

  Body:

  License 123456-123456789-abcdefgh

  issued to 12.34.56.78

- When a key is resent:

  Title: ALP11 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 What's New in Versions 10 and 11

Version 10

Global Settings and Templates

Now you can globally apply settings to all your AreaList Pro areas! The settings can be applied to existing or new areas. See the detailed description in the Command Reference section.

Move a Column

Move a column's position within an area with AL_MoveColumn.

Duplicate a Column

Duplicate a column within an area with AL_DuplicateColumn.

Grouping columns and headers

- Group columns together
- Dragging a group of columns
- Column group header (above the column individual headers or replacing them): row and cell properties can be used with Row = -3
- Hierarchy row merged into one cell that goes across several columns

Area display features

- Zooming an entire area (smaller or larger display, factor 0.1 to 5)
- Display Transposed (rotate rows ↔ columns) – This allows "palette" style interface, with different data types among "rows"
- Multi-level break processing (ala PrintList Pro) including in hierarchical lists
- Show/hide vertical scrollbar

Column features

- Minimum column width for any column(s)
- New properties to enable/disable each column's sorting, dragging or resizing
- "Use ellipsis" settable for headers, specific columns, individual rows, footers, and even individual cells
- Popup entry type is now a column property: 0 = default (depends on data type), 1 = date, 2 = time
- Limiting entry string length (length limit is now a column property)
- Define a formula instead of a callback for a calculated column

Other cool features

- CSV format in Copy / Drag from an area
- Named selections as sources
- Entry placeholder support
- Formatting at cell level: ALP_Cell_Format and ALP_Cell_Flags
- Formatting at row level (font, color, alignment, spacing, etc.): ALP_Row_Flags
- Show a longint value as picture in Field or Array Display (e.g. "star rating" display)
- Access row and cell properties for header and footer (Header: row = 0, Footer: row = -1)

Miscellaneous

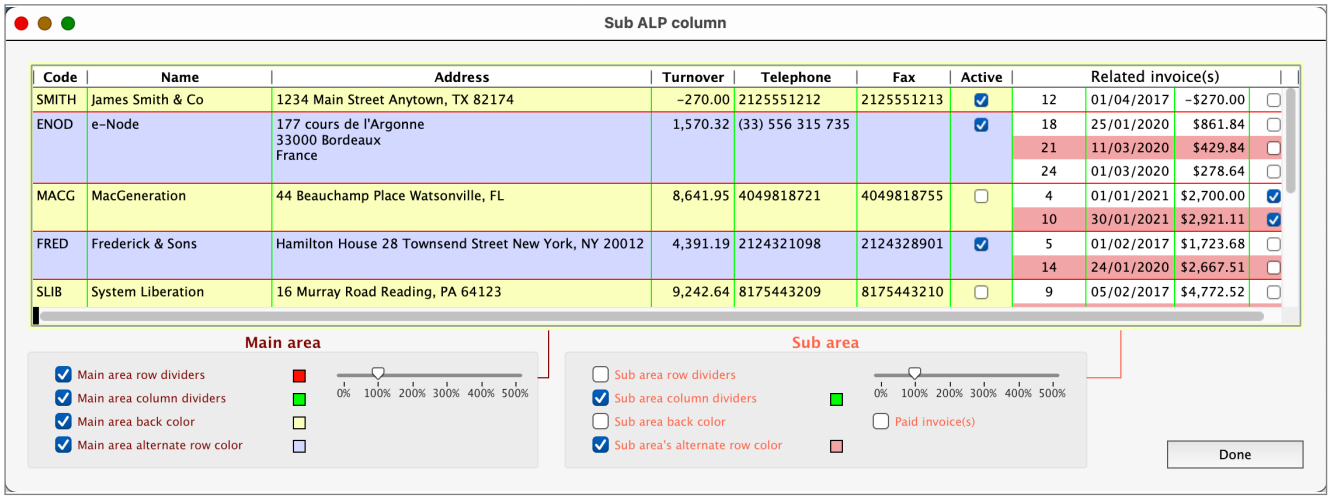
- Hide row dividers in empty areas (ALP_Area_ShowRowDividers is now 3-state)
- Custom picture above vertical scrollbar (old AL_SetHeaderOptions) including callback on mouse down or up
- Override row/cell bottom row divider color (set a specific color for the bottom divider of a given row or cell)
- ALP_Column_EntryAllowReturn: allow/disallow return in specific columns regardless of the area settings
- Single line headers in "grid" mode (i.e. several lines per row, ALP_Area_AltHdrRowsInGrid>1)
- Single line footers in "grid" mode (i.e. several lines per row, ALP_Area_FtrRowsInGrid>1)
- Drag and move a column (physically reorder the column) even when not running in compatibility mode)
- New API replacement of ***AL_SetInterface***: appearance + sort indicator properties

Version 11

Sub-ALP

Sub-ALP is a fantastic new feature in AreaList Pro which enables you to insert a whole AreaList Pro area within a cell, using the ORDA concept.

A Sub-ALP column displays cells that actually are AreaList Pro areas, with their own specific settings:



See the example in the v11 Tutorial chapter.

PrintList Pro

PrintList Pro is now a module of AreaList Pro. There will be no version 7 of PrintList Pro; now there is only one plugin: AreaList Pro v11.

Note: You must remove the PrintList Pro bundle from your Plugins folder, if it exists.

All AreaList Pro properties can be used for printing. Old PrintList Pro constants and commands are retained for compatibility with your existing code using any PrintList Pro API.

Regarding licensing, printing is now an option to AreaList Pro licenses.

Unlimited single users, OEM developers and Partners subscribers can request a license key to v11, which will include it.

PL_Register is still included for compatibility reasons, however it is now just a clone of ***AL_Register***. Calling ***AL_Register*** with a license key that includes the printing option will activate the whole PrintList Pro module.

A single call to ***AL_Register*** will be enough.

When upgrading to v11, your existing code will not require any other modification than removing the call to ***PL_Register*** and updating the ***AL_Register*** call.

Any developer who uses PrintList Pro and not AreaList Pro must upgrade to AreaList Pro v11 with a license key that will only provide the PrintList Pro functionalities, however they can use the whole (relevant to printing) AreaList Pro API without displaying any area.

Export to Excel

Exporting a displayed area to an Excel file takes two lines of code.

All formatting is preserved, including fonts, styles, headers, colors, alt rows, etc.

The export can be in either XLS or XLSX format (you are responsible for the proper extension to the document):

Use ALP_Area_ExportOptions then ALP_Area_ExportToFile to export the area to XLS or XLSX

The resulting document will include either the whole area or just the selected rows / specified columns:

Example

The following code will export the area to XLSX:

```
C_TEXT($Tpath)
C_LONGINT($area)
$Tpath:=Select folder("Select destination folder"; Get 4D folder(Documents folder))
If (Test path name($Tpath)=Is a folder)
   AL_SetAreaLongProperty($area; ALP_Area_ExportOptions; AL Export XLSX)  // XLSX
   AL_SetAreaTextProperty($area; ALP_Area_ExportToFile; $Tpath+"My exported Area.xlsx")  // XLSX export
   SHOW ON DISK($Tpath+"My exported Area.xlsx")
End if
```

The following code will export selected rows to XLS:

```
C_TEXT($Tpath)
C_LONGINT($area)
$Tpath:=Select folder("Select destination folder"; Get 4D folder(Documents folder))
   AL_SetAreaLongProperty($area; ALP_Area_ExportOptions; AL Export binary XLS + \
      AL Export Selected Rows Only)
   AL_SetAreaTextProperty($area; ALP_Area_ExportToFile; $Tpath+"My exported Area.xls")  // XLS export
   SHOW ON DISK($Tpath+"My exported Area.xlsx")
End if
```

Format and Options

The first step is to define the format and options with the ALP_Area_ExportOptions property.

Here are the possible settings.

Format (mandatory)

- AL Export binary XLS - create XLS file.
- AL Export XLSX - create XLSX file.

Options

The other settings are optional.

- AL Export No Header - do not write the header.
- AL Export Selected Rows Only
- AL Export Formatted Date Time - dates and time values will be exported as text according to the formatting used in the area.

If this option is not used, the values will be exported as numbers and displayed according to the Excel default format.

- AL Export Formatted As Picture - when using pictures for display (checkbox or custom picture list), the actual picture will be placed in the cell, no value.
- AL Export Boolean As Text - uses "✓" for True and "-" for False (unless AL Export No False is specified).
- AL Export No False - for False, nothing is written. Usable only when the Boolean value is not exported as Boolean (when the format uses pictures (checkbox or pictures) and AL Export Formatted As Picture or AL Export Boolean As Text are used.
- AL Export Mapped Values - when drawing through popup (ALP_Column_DisplayControl = 3), the mapped value is written.
- AL Export Include SubALP - also export sub-ALP columns.
- AL Export Using List - when this value is used and ALP_Area_ExportList is not empty, only columns defined in ALP_Area_ExportList will be exported (see below).

Selecting the Columns to export

As of version 11.2, the export can be restricted to some columns. A specific property is used for this purpose: ALP_Area_ExportList.

Use it to set the list of columns to export with ALP_Area_ExportToFile.

Set a comma separated list of column numbers, e.g. "11,3,4,9,1" to export only these columns in that order.

Once this is done, include the AL Export Using List option in ALP_Area_ExportOptions to use your list of columns.

Note: when moving columns by Drag and Drop (and ALP_Area_Compatibility = 1 or ALP_Area_DragMoveColumns = 1) or with ***AL_MoveColumn***, this list is automatically adjusted (column numbers are renumbered according to the move).

The list can also be set with ALP_Object_ExportList to be used with ***AL_GetObjects***/***AL_SetObject***

See Export to Excel in the v11 New Features Tutorial chapter.

Note: when ALP_Area_NumRowLines is not 1, Excel will be set to use its own wrap mode (even when AreaList Pro does not), so that it splits the text on "\r" or when it is a "long" text, i.e. that doesn't fit into a column, whose width is set by AreaList Pro.

Offscreen AreaList Pro areas

It may sound strange to have an offscreen AreaList Pro area, since the purpose of AreaList Pro is basically to display data.

However, the new v11 offscreen feature may be handy if you want to print or export an area without displaying it.

Here is an example of building an Excel document without showing anything to the user:

```
C_TEXT($Tpath)
C_LONGINT($LoffScreenAlp)
$LoffScreenAlp:=AL_GetAreaLongProperty(0; ALP_Area_NewOffscreen)  // New Offscreen Area
 //Do something to set up the area just as if it was onscreen
$Tpath:=Select folder("Select destination folder"; Get 4D folder(Documents folder))
If (Test path name($Tpath)=Is a folder)
   AL_SetAreaLongProperty($LoffScreenAlp; ALP_Area_ExportOptions; AL Export XLSX + AL Export No Header)
    // XLSX, no headers
   AL_SetAreaTextProperty($LoffScreenAlp; ALP_Area_ExportToFile; $Tpath+"My exported Area.xlsx")  // XLSX export
   SHOW ON DISK($Tpath+"My exported Area.xlsx")
End if
AL_SetAreaLongProperty(0; ALP_Area_DeleteOffscreen; $LoffScreenAlp)  //Delete Offscreen Area
```

Entity Selections

Use ORDA entities in your AreaList Pro areas. See Entity Selections in the v11 New Features Tutorial chapter.

Standalone Printing

In addition to the whole PrintList Pro module, now fully included in AreaList Pro v11, a simple AreaList Pro property triggers printing without using PrintList Pro: ALP_Area_PrintArea.

The area is printed as is, with its formatting and other settings.

This "standalone" printing requires a license key that includes the printing option, and one line of code:

```
$Lerror:=AL_GetAreaLongProperty(area; ALP_Area_PrintArea)
```

The print job uses current **PRINT SETTINGS.**

The ALP_Area_PrintOptions property can be used for specific settings.

Note: the "landscape" attribute, if used, will override the orientation defined in **PRINT SETTINGS**.

For more info, see the Examples database:

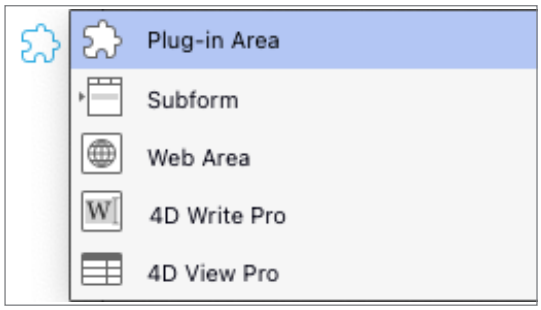
- Example 17 as described in the Legacy Features Tutorial chapter
- Standalone Printing as described in the v11 New Features Tutorial chapter

4 Getting Started with AreaList Pro

Creating your first AreaList Pro Area

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

1. Create a new form, or open an existing one that you want to add a list to.
2. Choose **Plug-in 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 **AreaList Pro** from the **Type** popup menu. (If the AreaList Pro option is not available, please refer to the installing the plugin instructions). Note that there is also an option under the Type popup for PrintList Pro. This is for backwards compatibility; PrintList Pro used to be a separate product, but it's now incorporated into AreaList Pro.
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 AreaList Pro version and copyright information.

Advanced Properties or Commands?

You now have a choice: You can configure your list by using the easy-to-use point-and-click interface offered by the Advanced Properties dialog, or you can use the AreaList Pro commands to control the list.

You can also use a combination of both methods.

The Advanced Properties dialog doesn't require you to write any code, and is suitable for many projects. However, if you want to have more control over your list, you can use the commands.

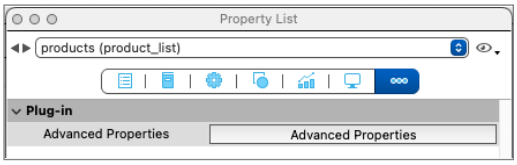
You can use both options together: you might use the Advanced Properties dialog to do most of the configuration for a list, and then apply some commands to add some additional programmable control.

When you do this, the settings specified in the Advanced Properties dialog will be applied when the form is loaded, and then the commands will be applied.

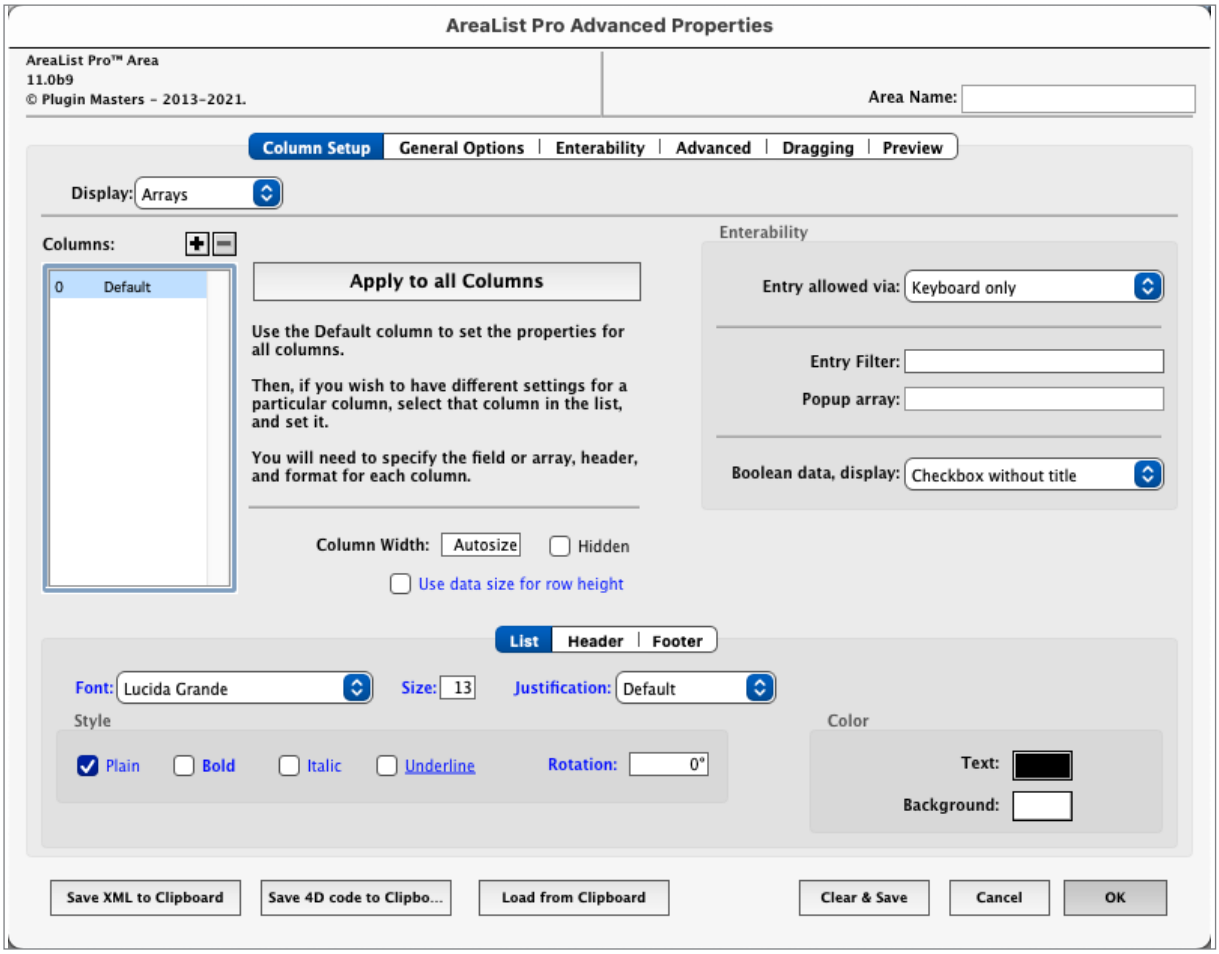
In the following sections we give a brief overview of how to work with both options; they are described in detail in other sections of this manual.

Using the Advanced Properties Dialog

To invoke the Advanced Properties Dialog, click on the **Advance Properties** button next to **Advanced Properties** in the area's Property List.



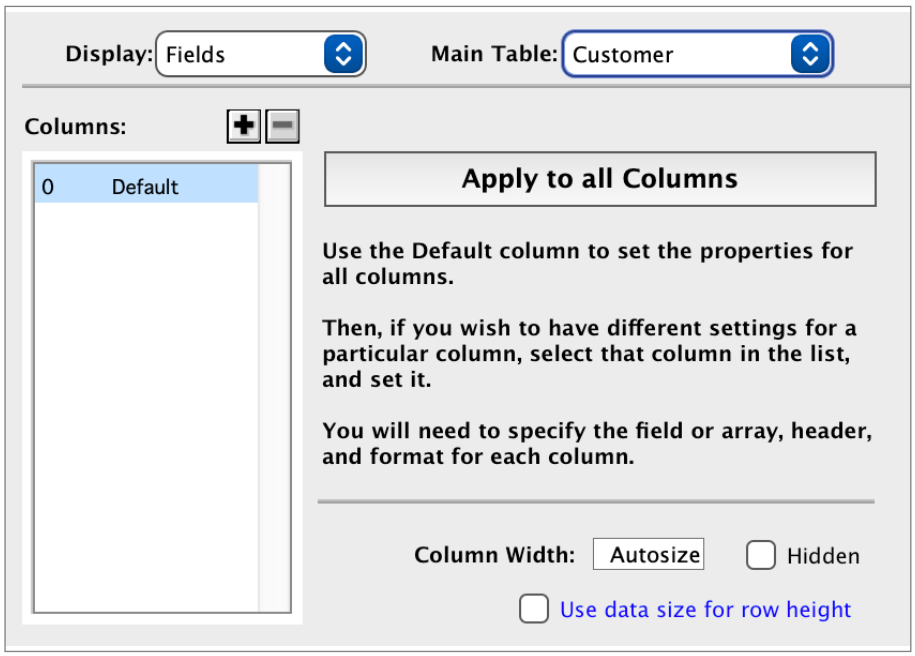
The Advanced Properties Dialog opens:



There are a few choices that you **must** make in order for your list to work, and there are lots of other choices that you **can** make to configure it.

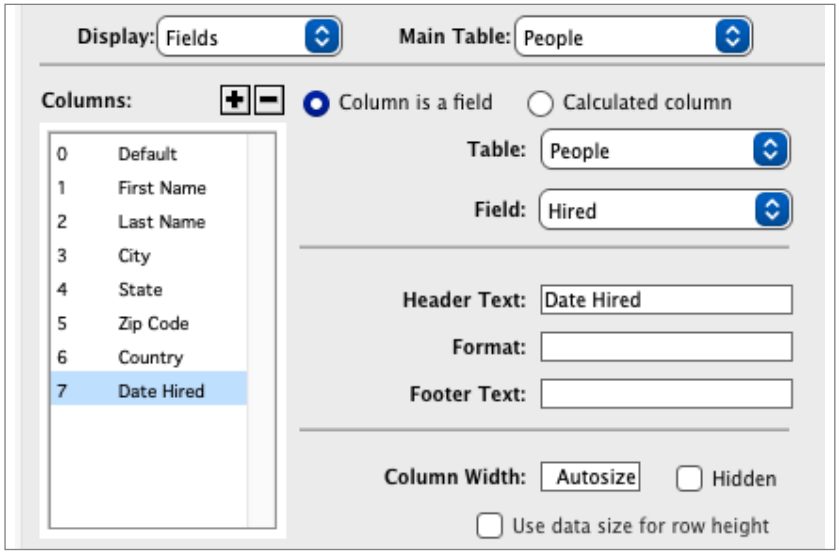
The first thing to choose is whether you want to display data from arrays or fields. Click on the **Display:** popup menu and choose either **Arrays** or **Fields**.

1. Next you need to specify how many columns your list will comprise, and which arrays or fields will populate them. Add a column by clicking on the big Plus sign in the **Columns**: area.
2. The area to the right of the Columns area changes. If you have chosen to display fields, it now looks like this:



4. Choose a table from the **Table**: popup menu
5. Choose a field from the **Field:** popup menu.
6. Enter the column header into the **Header Text:** field.

7. Add some more columns and select their properties. For example:



8. Click the **OK** button when you have added all the columns you need.

9. Your list is now ready to use!

You can find a detailed explanation of the Advanced Properties dialog here.

Working with AreaList Pro Commands and Functions

You can use the commands and functions to configure every aspect of an AreaList Pro area, and to get information about an area. The commands and functions are grouped into themes according to the aspect of the area that they affect, such as Rows, Columns, Sorting, Drag and Drop etc.

When to use the Commands and Functions

All AreaList Pro commands and functions need to be passed a reference to the area on which they will act. Since AreaList Pro areas are initialised in the On Load form event, the commands must be called during this phase or afterwards; if you try to call any AreaList Pro commands before the form has been loaded, you'll get an error message because 4D does not recognise the area reference.

If you do not issue any AreaList Pro commands in the On Load event, and you haven't chosen any fields or arrays in the Advanced Properties dialog, nothing will be displayed in the AreaList Pro area on the form.

You can modify the area by making calls to commands during any other event or from objects, such as buttons and menus, on the form or in menu bars.

The commands can be used completely independently of the Advanced Properties dialog, or they can work in conjunction with the options you set therein. For example, you might select the fields to display in the Advanced Properties dialog and then use some commands to specify different coloring for each row according to some criteria that you specify.

The maximum number of columns that can be added to an area is 32767 (subject to memory limitations).

Anatomy of an AreaList Pro Command

Each command you write must adhere to a specific syntax in order for it to be correctly understood by AreaList Pro. Some commands (the "getters" and the pointer variants) return a result code: these are **functions**. See the Command Reference section for the requirements for each command. You can check the result code to find out if a function executed OK or if there was a problem and, if so, get some information about what that problem was.

Every command consists of the command name followed by two or more parameters. The first parameter is always a reference to the AreaList Pro area.

For example, the ***AL_AddColumn*** function adds a field or array column to an area:

```
$err:=AL_AddColumn (area;->[table]fieldname;1)
```

This function adds a column to the area AreaList Pro area. The column displays data from the [table]fieldname field, and it will be the first column in the area. If the column was added successfully, $err will be 0; if not, $err will contain an error number. You can check the meanings of the error codes in the Result Codes list.

Commands that get or set properties for an area all include the property that you want to affect, and a value to use to specify an option (if it's a "setter") or to receive the result (if it's a "getter"). See the section on Getters and Setters, below.

All AreaList Pro commands are described in the Command Reference section along with examples of how to use them.

Debugger

In AreaList Pro default mode (and interpreted), errors will automatically display the 4D debugger window.

In compiled mode, an alert is not displayed by default. ALP_Area_TraceOnError default is 1: TRACE in interpreted, do nothing in compiled mode..

See Using the debugger.

Getters and Setters

Most of the commands are either “getters” or “setters”: they either **get** information about a specific property, or they **set** a specific property.

The Getters and Setters are each available in four variants, which allow for the different data types of the properties: Longinteger, Pointer, Real, and Text.

For example, if you want to set a property for a column, you can use one of the following commands:

- ***AL_SetColumnLongProperty***
- ***AL_SetColumnPtrProperty***
- ***AL_SetColumnRealProperty***
- ***AL_SetColumnTextProperty***

The pointer options (e.g. ***AL_SetColumnPtrProperty***) allow you to use just one version of the command for getting and setting all the relevant properties; you pass a pointer to the variable instead of the actual value.

Getters, some Setters (the Pointer variants), and some other commands use the following syntax:

$result:=***AL_Command*** (AreaRef;value1;value2;property;value3)

Most setters use the following syntax:

AL_Command (AreaRef;value1;value2;property;value3)

Command Part	What it is	Comments
$result	Result code	The code will contain 0 if the command executed successfully, or an error code if it didn’t.
AL_Command	Command name	Tells AreaList Pro what you want to do.
AreaRef	Name of the AreaList Pro area	The name you gave the AreaList Pro area in the Object Properties dialog.
value1	A value to pass to the command	Used with the Row, Column, and Cell commands, to pass the column or row number.
value2	A value to pass to the command	Used with the Cell commands to pass the row number.
property	Property	A constant representing the specific property you want to get or set.
value3	Value	Tells AreaList Pro exactly how you want to affect the property.

Example

Let's suppose that we want to set the width for the first column in an AreaList Pro area called ProductList to 200. You could use either ***AL_SetColumnLongProperty*** or ***AL_SetColumnPtrProperty***:

```
AL_SetColumnLongProperty (ProductList;1;ALP_Column_Width;200)
```

or

```
$W:=200
$err:=AL_SetColumnPtrProperty (ProductList;1;ALP_Column_Width;->$W)
```

Conversely, we can find out what the current setting is by using the associated GET commands:

```
$W:=0
$W:=AL_GetColumnLongProperty (ProductList;1;ALP_Column_Width)
```

or

```
$W:=0
$err:=AL_GetColumnPtrProperty (ProductList;1;ALP_Column_Width;->$W)
```

Properties

Each command theme has its own set of properties that can be used to get or set various aspects of the area, and for each property a 4D constant has been defined.

You'll find a complete reference in the Properties by Theme section.

See the Tutorial section below to learn more about getting started with AreaList Pro.

Command Descriptions and Syntax

AreaList Pro has its own collection of commands and functions that you use to control your AreaList Pro areas, to find out what actions the user has taken, and to do whatever processing is needed as a result of those actions.

For example, if the user drags a row from one AreaList Pro area to another, it's up to you to use the commands to do whatever is required with the dropped row.

Commands

The commands are organised into themes which relate to a particular part of the AreaList Pro area: Area, Cells, Columns, Objects, Rows, and some miscellaneous Utility commands. For each theme except Objects and Utility there is a group of four "Getter" functions and four "Setter" commands, each targeting a different property type. For example, the Area theme has the following Getters and Setters:

AL_GetAreaLongProperty	AL_SetAreaLongProperty
AL_GetAreaPtrProperty	AL_SetAreaPtrProperty
AL_GetAreaRealProperty	AL_SetAreaRealProperty
AL_GetAreaTextProperty	AL_SetAreaTextProperty

An AreaList Pro command syntax looks like this:

AL_SetAreaLongProperty (AreaRef:L;Property:T;Value:L)

AreaRef is the AreaList Pro area that the command refers to, and is always a longint.

Property is a constant representing the thing you want to set or get. Some commands accept one property, and some accept more.

Value is the value you want to use with the property.

In the example shown here, the value will always be a longint; there are other versions of the commands which require pointers, real numbers, or text.

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

:L - longint

:O - blob

:R - real

:T - text

:Y - array

:Z - pointer

Note: Boolean values are passed or returned as longints, where 1 = true and 0 = false.

For example, the ALP_Area_HideHeaders property has two options: – 1 to hide the headers and 0 to display them:

AL_SetAreaLongProperty (AreaRef;ALP_Area_HideHeaders;0) // Header will be displayed

You can find complete descriptions of the commands, along with examples, in the Command Reference section, and descriptions of all the properties in the Properties by Theme section.

This section includes details of how to use each property; the Type column tells you what type of data it requires, and this is matched to the command variant.

For example, the following snippet shows the details for the ALP_Column_FtrSize property, which you can use to get or set the font size for a column's footer row:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Column_FtrSize	✔	✔	✔	real	12 on Windows 13 on MacOS	4	128	Font size

Note: **Per** stands for **Persistent**. See Properties by Theme.

The **Type** is Real, so you would use the ***AL_SetColumnRealProperty*** command to set the footer font size for the second column to 10:

AL_SetColumnRealProperty (area;2;ALP_Column_FtrSize;10)

Note that Boolean properties are called as longints (1 = true, 0 = false).

Functions

Functions return a result code when they are called. Usually this will be the information you requested, such as the cell the user clicked in, or the row that was dragged, or the column header that was clicked.

Their syntax looks like this:

AL_GetAreaLongProperty (area:L; Property:T) ➞ result:L

AreaRef is the AreaList Pro area that the function refers to (always a longint).

Property is a constant representing the property you want to get information about. result is the result of the function (a longint, in this example).

For example, you can use ***AL_GetAreaLongProperty*** with the ALP_Area_SelRow option to find out which row the user selected:

```
C_LONGINT($row)
$row:=AL_GetAreaLongProperty (area;ALP_Area_SelRow)
```

Copying or dragging from an AreaList Pro Area

AreaList Pro supports **Edit > Copy** or Dragging from a selection of row or cells to any destination including to an external document such as an Excel type spreadsheet.

Note: make sure that ALP_Area_DragRowMultiple is set to true when dragging a multiple row selection.

Properties

The contents will be copied as delimited text, the “field” (column) default separator being TAB and the “record” (row) default separator being CR+LF. These settings can be modified using the ALP_Area_CopyFieldSep and ALP_Area_CopyRecordSep properties.

A field wrapper character can also be used, which will be placed both before and after each field (set with ALP_Area_CopyFieldWrapper, default is none).

In addition, the ALP_Area_CopyHiddenCols property (default to 0 = false) is used to define whether hidden columns must be included or not.

See AreaList Pro Area Copy & Drag Properties.

Headers

When the area is set as multiple row selection and headers are visible (ALP_Area_SelType set to 0 or not set, ALP_Area_SelMultiple set to 1, ALP_Area_HideHeaders set to 0 or not set), setting ALP_Area_CopyOptions to yes will make the header text to be copied into the pasteboard (or dragged to the destination) on top of the selected row values.

Upgrading from Previous versions of AreaList Pro

AreaList Pro version 11 is compatible with 4D version 18 and above.

To upgrade to AreaList Pro version 11, 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 v8.x (and earlier releases):

- ***AL_Register*** returns 0 if registration was successful
- The 4D project method ***Compiler_ALP*** is no longer needed

5 Tutorial: Legacy Features

The following examples illustrate how to use AreaList Pro.

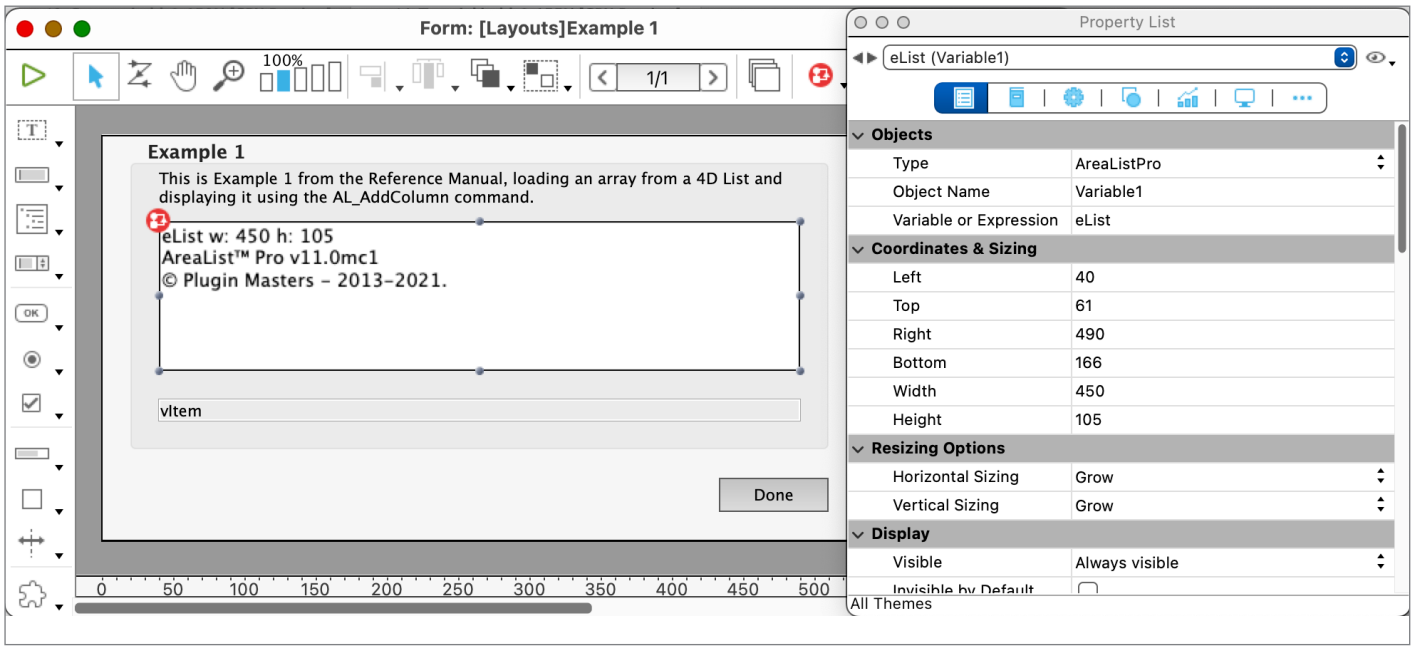
You can find them all in the Examples database.

Example 1: Loading an array from a 4D list

In this example, a 4D list is loaded into an array and displayed in an AreaList Pro area, with some formatting applied.

When the user clicks on a row, its contents are copied into a variable called vItem and displayed below the AreaList Pro area.

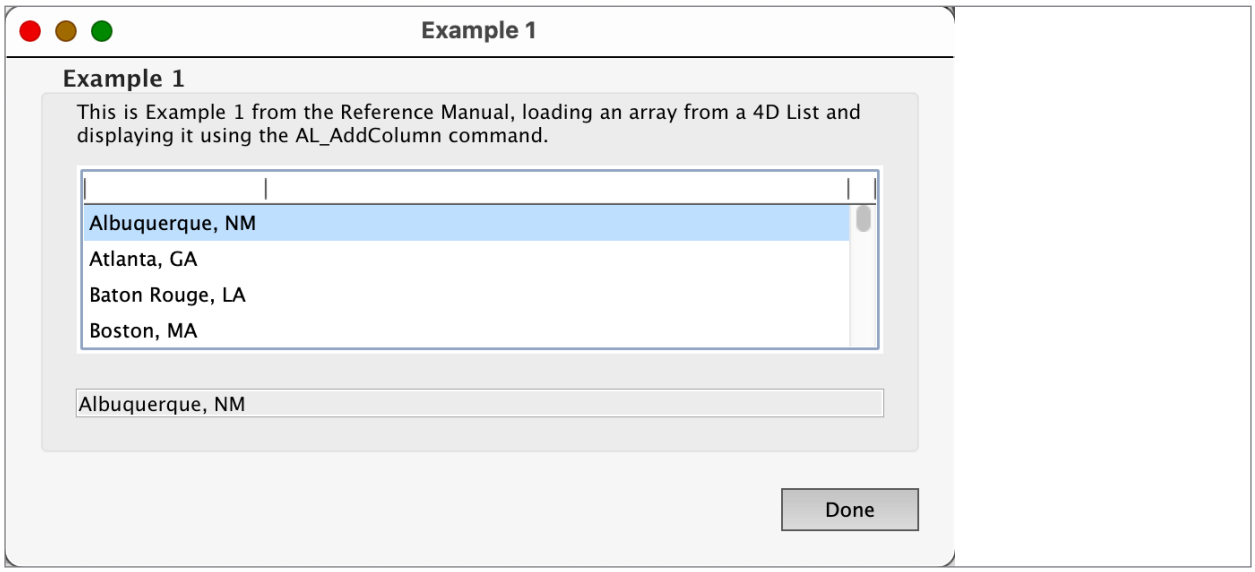
First we create a new AreaList Pro area on a form:



The AreaList Pro area Object Method handles all the formatting and events:

```
C_LONGINT($Lerror; $row)
Case of
  : (Form event code=On Load)  //initialize the AreaList Pro object
    LIST TO ARRAY("City, State"; aCityState)  //copy the list into an array
    $Lerror:=AL_AddColumn(eList; ->aCityState)  //display array in AreaList Pro object
    DEMO_Default(eList)  //general display settings
    AL_SetAreaLongProperty(eList; ALP_Area_SelRow; 1)  //row 1 selected
    vItem:=aCityState{1}
  : (Form event code=On Plug in Area)  //respond to user action
    If (AL_GetAreaLongProperty(eList; ALP_Area_AlpEvent)=AL Single click event)  //single-click on a row (or up / down arrow keys)
      $row:=AL_GetAreaLongProperty(eList; ALP_Area_SelRow)  //get the row the user selected
      //            OR
      //$row:=AL_GetAreaLongProperty (eList;ALP_Area_ClickedRow)  //get the row the user clicked
      vItem:=aCityState{$row}  //get the value in that element of the array
    End if   //ALP_Area_AlpEvent
End case
```

This is the result when you choose **Example 1** from the **Examples** menu:



Let's take a look at the commands that were used.

1. Loading the arrays. On the Object Method for the eList object, the following code loads the array and adds it to the AreaList Pro area:

```
Case of
  : (Form event Code=On Load)  // initialize the AreaList Pro object
  LIST TO ARRAY ("City, State";aCityState)  // copy the list into an array
  $error:=AL_AddColumn (eList;->aCityState)  // display array in AreaList Pro object
```

2. Apply **some formatting.** The ***DEMO_Default*** project method is called:

```
C_LONGINT($1; $AL_Area)
If (Count parameters>=1)
   $AL_Area:=$1
   AL_SetAreaLongProperty($AL_Area; ALP_Area_HideHeaders; 0)  //Header will be displayed
   AL_SetAreaLongProperty($AL_Area; ALP_Area_ShowFooters; 0)  // Hide footer
      If (Is Windows)  //Windows
      AL_SetColumnTextProperty($AL_Area; -2; ALP_Column_HdrFontName; "Tahoma")
      AL_SetColumnLongProperty($AL_Area; -2; ALP_Column_HdrSize; 11)
      AL_SetColumnLongProperty($AL_Area; -2; ALP_Column_HdrStyleB; 1)
      AL_SetColumnTextProperty($AL_Area; -2; ALP_Column_FontName; "Tahoma")
      AL_SetColumnLongProperty($AL_Area; -2; ALP_Column_Size; 11)
      AL_SetColumnLongProperty($AL_Area; -2; ALP_Column_StyleB; 0)
   Else   //MacOS
      AL_SetColumnTextProperty($AL_Area; -2; ALP_Column_FontName; "Lucida Grande")
      AL_SetColumnLongProperty($AL_Area; -2; ALP_Column_Size; 11)
      AL_SetColumnLongProperty($AL_Area; -2; ALP_Column_StyleB; 0)
      AL_SetColumnTextProperty($AL_Area; -2; ALP_Column_HdrFontName; "Lucida Grande")
      AL_SetColumnLongProperty($AL_Area; -2; ALP_Column_HdrSize; 11)
      AL_SetColumnLongProperty($AL_Area; -2; ALP_Column_HdrStyleB; 1)
   End if
   // Set the area properties
   AL_SetAreaLongProperty($AL_Area; ALP_Area_NumHdrLines; 1)  //Line(s) in header
   AL_SetAreaLongProperty($AL_Area; ALP_Area_NumRowLines; 1)  //Line(s) per row in list
   AL_SetAreaRealProperty($AL_Area; ALP_Area_RowIndentV; 3)  // Height padding 3 points
End if
```

When you click on a row in the area, its contents are displayed in the text area below. The On Plug in Area 4D event monitors clicks in the area (or up / down arrow keys), and we can then call ***AL_GetAreaLongProperty*** with either the ALP_Area_SelRow or ALP_AreaClickedRow property to get the selected row number:

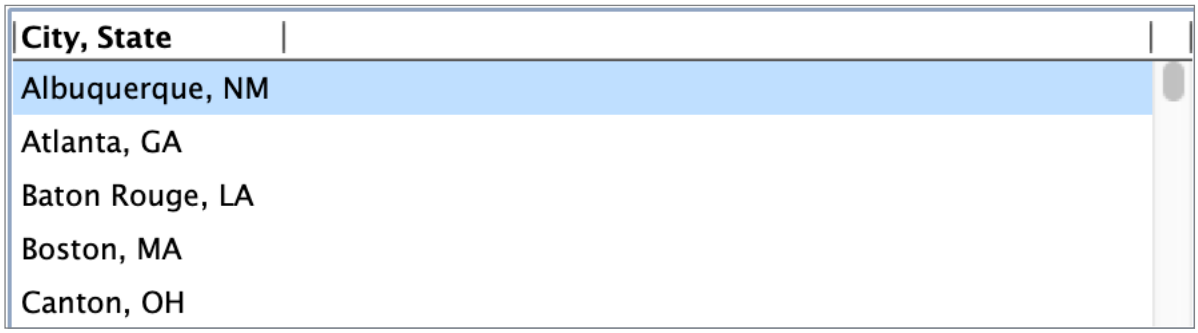
```
Case of
   : (Form event code=On Plug in Area)  //respond to user action
      If (AL_GetAreaLongProperty(eList; ALP_Area_AlpEvent)=AL Single click event)  //single-click on a row (or up / down arrow keys)
         $row:=AL_GetAreaLongProperty(eList; ALP_Area_SelRow)  //get the row the user selected
         //            OR
         //$row:=AL_GetAreaLongProperty (eList;ALP_Area_ClickedRow)  //get the row the user clicked

         vItem:=aCityState{$row}  //get the value in that element of the array
      End if   //ALP_Area_AlpEvent
```

Or we could call (but it would only react to click, not up / down arrow selection):

```
$row:=AL_GetAreaLongProperty (eList;ALP_Area_ClickedRow)  // get the clicked row
```

Example 2: Add header text

In Example 1 our AreaList Pro area looked OK, but it would look better if there was some text in the header row - for example:

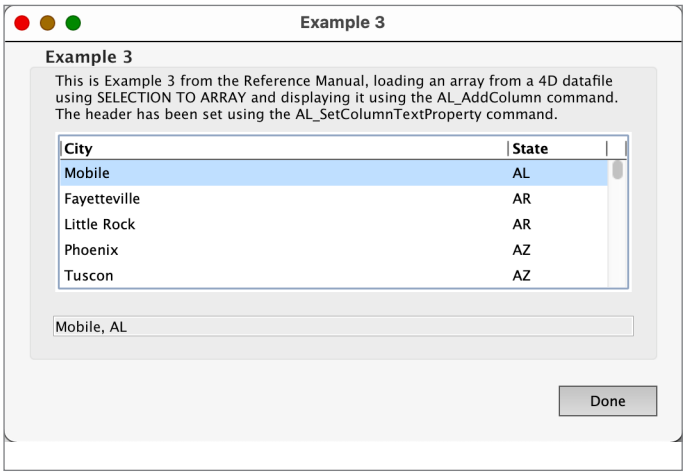


Header row text is added using ***AL_SetColumnTextProperty*** with the ALP_Column_HeaderText parameter in the area's Object Method:

AL_SetColumnTextProperty (eList;1;ALP_Column_HeaderText;"City, State")

Example 3: Creating arrays from a 4D table

In this example, two arrays are loaded from the database, added to the AreaList Pro area, and displayed with headers:



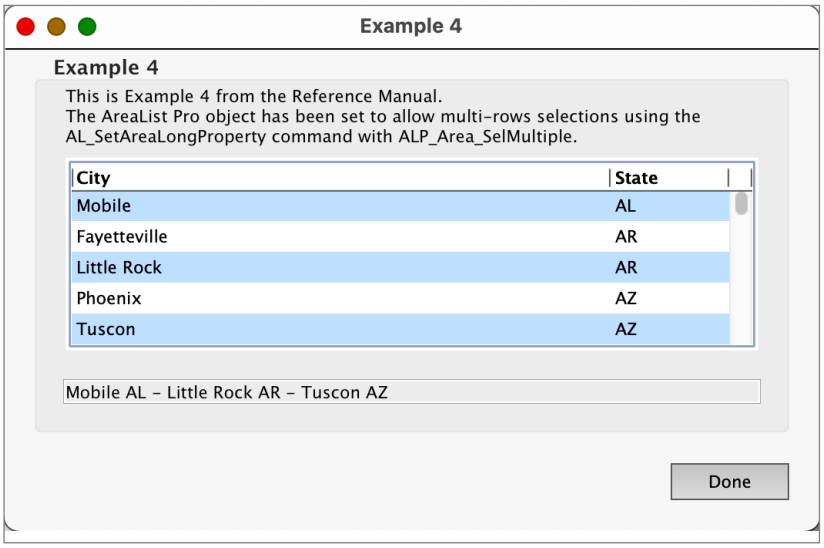

```
ALL RECORDS([Cities])  //load all records in the Cities table
SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState)  //copy field values into arrays
$error:=AL_AddColumn (eList;->aCity)  //display array in AreaList Pro object
$error:=AL_AddColumn (eList;->aState)  //display array in AreaList Pro object
AL_SetColumnTextProperty (eList;1;ALP_Column_HeaderText;"City")  //first column header
AL_SetColumnTextProperty (eList;2;ALP_Column_HeaderText;"State")  //second column header
DEMO_Default (eList)  //general display settings
AL_SetColumnRealProperty (eList;1;ALP_Column_Width;350)  //fixed width for column 1
AL_SetAreaLongProperty (eList;ALP_Area_AutoSnapLastColumn;1)  //calculate column 2 width to area edge
AL_SetAreaLongProperty (eList;ALP_Area_SelRow;1)  //row 1 selected
vItem:=aCity{1}+", "+aState{1}
```

We could also have used the ***AL_SetObject*** command to add the arrays:

```
ALL RECORDS([Cities])
SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState)
ARRAY POINTER(aPtrCols;2)
aPtrCols{1}:=->aCity
aPtrCols{2}:=->aState
$error:=AL_SetObjects (eList;ALP_Object_Columns;aPtrCols)
```

Example 4: Allow multi-row selection

The default row selection method is single rows. In this example we build on Example 3 and enable the selection of multiple rows:



In the Object Method for the AreaList Pro area we've added two commands

```
AL_SetAreaLongProperty (eList;ALP_Area_SelMultiple;1)  //set multi-row selection mode
```

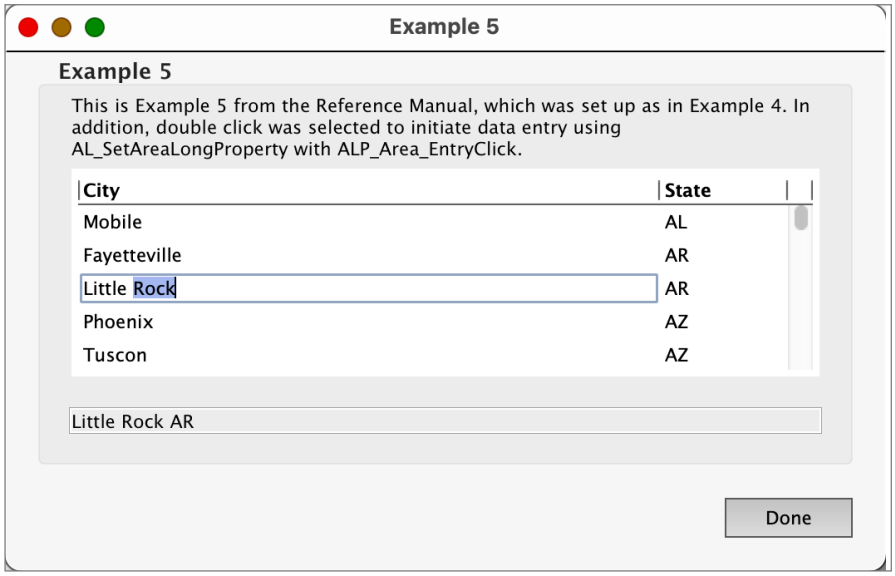
With this option selected, the user can Ctrl-click (Windows) or Cmd-click (Mac) to select multiple rows.

Note that the text area lists all the selected rows. We use the ***AL_GetObjects*** function to find out which rows were selected:

```
Case of
  : (Form event code=On Plug in Area)  //respond to user action
    If (AL_GetAreaLongProperty (eList;ALP_Area_AlpEvent)=AL Single click event)
        //single-click on a row (or up/down arrow keys)
      ARRAY LONGINT(aRows;0)
      $error:=AL_GetObjects (eList;ALP_Object_Selection;aRows)  //get the rows selected by user
      vItem:=""
      For ($i;1;Size of array(aRows))  //look at each row selected by user
        vItem:=vItem+aCity{aRows{$i}}+" "+aState{aRows{$i}}  //plug values in vItem
        If ($i<Size of array(aRows))  //not the last item
          vItem:=vItem+" - "  //separator
        End if
      End for
    End if  //ALP_Area_AlpEvent
End case
```

Example 5: Allow data entry via double-click

In many cases you simply want to display data without allowing it to be modified. Sometimes, however, you may want to allow the user to modify certain data.

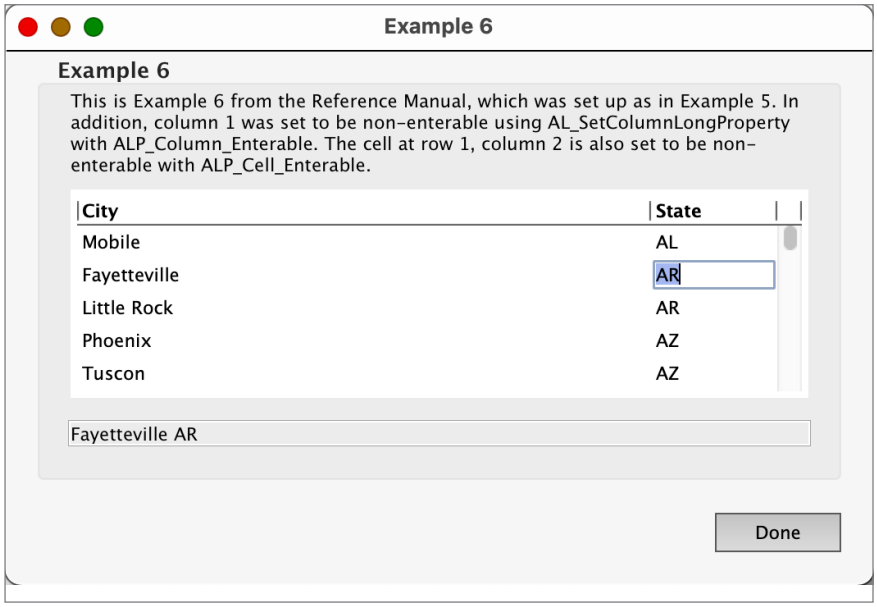


To enable that, you can use the ALP_Area_EntryClick property using the command ***AL_SetAreaLongProperty***:

```
AL_SetAreaLongProperty (eList;ALP_Area_EntryClick;2)  // set double click to enter data entry mode
```

Example 6: Specifying which columns are enterable

In Example 5 we applied the “entry by double-click” property to the entire area. But what if you only want certain columns to be enterable?



This can be controlled via the ALP_Column_Enterable property and the associated command ***AL_SetColumnLongProperty***:

```
AL_SetColumnLongProperty (eList;1;ALP_Column_Enterable;0)  // set column 1 to be non-enterable
```

You can similarly control enterability for individual cells with the ***AL_SetCellLongProperty*** command:

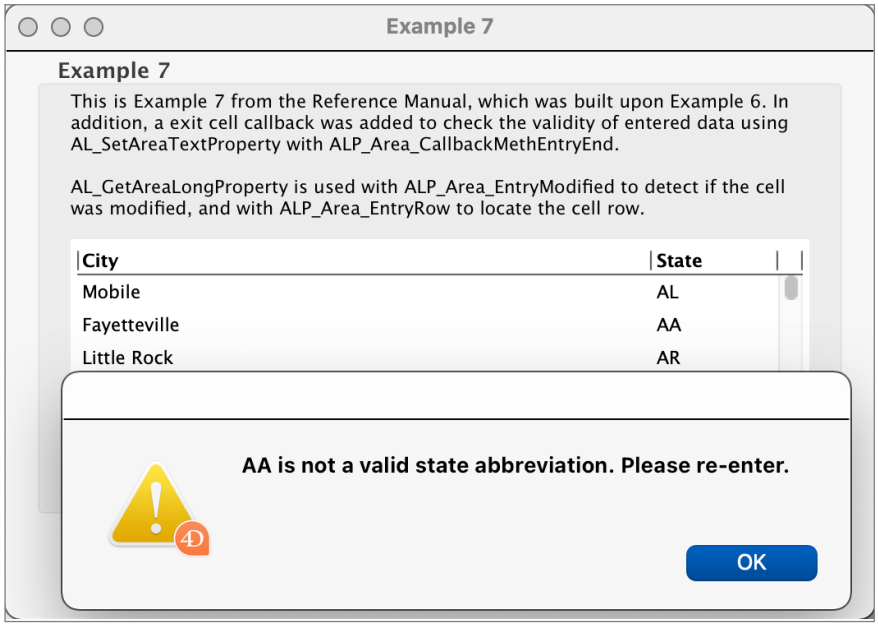
```
AL_SetCellLongProperty (eList;1;2;ALP_Cell_Enterable;0)
 // set the cell at row 1, column 2 to be non-enterable
```

Example 7: Using a callback method to check data entry validity

A “callback” is a 4D project method which is executed by a plug-in.

AreaList Pro lets you make use of callbacks when entering and exiting an AreaList Pro object. See the Callbacks chapter for more detailed information.

In this example we are using a callback when an entry in the State column is modified to make that a valid State abbreviation has been entered; if it hasn’t, the user will see an Alert:



There are two things you need to do to get a callback working: create the callback method, and tell AreaList Pro when to call it.

1. Create the callback method

Our callback method is called ***ExitCallback***:

```
C_BOOLEAN($0)  // "data valid" return value (True or False)
C_LONGINT($1)  // AreaList Pro object reference
C_LONGINT($2)  // action terminating data entry for this cell
If ($2#AL Esc key action)  // escape key will ignore (and reset) entered data
    If (AL_GetAreaLongProperty ($1;ALP_Area_EntryModified)>0)  // was a cell modified?
        vRow:=AL_GetAreaLongProperty ($1;ALP_Area_EntryRow)  // find out which row
         // only the state array col 2 will be checked, we don’t need to worry about the entered column
        LIST TO ARRAY("State Abbrev";aPossStates)  // create a new array of all possible States
        ARRAY POINTER($ArrayNames;0)
        $error:=AL_GetObjects ($1;ALP_Object_Columns;$ArrayNames)
            If (Find in array(aPossStates;$ArrayNames{2}->{vRow})=-1)  // is modified element not valid?
```

```
            $0:=False  // tell AreaList Pro it is invalid — this forces the user to re-enter it
            BEEP  // provide user feedback
            ALERT($ArrayNames{2}->{vRow}+" is not a valid state abbreviation. Please re-enter.")
        Else
            $0:=True  // tell AreaList Pro entry is valid
        End if
    Else
        $0:=True  // tell AreaList Pro entry is valid
    End if
End if
```

2. Tell AreaList Pro when to call the callback method

This uses the ALP_Area_CallbackMethEntryEnd property of ***AL_SetAreaTextProperty***:

AL_SetAreaTextProperty (eList;ALP_Area_CallbackMethEntryEnd;"ExitCallback")

For more information about callbacks, see the Callbacks chapter.

Example 8: Using both an Entry and Exit callback

Entry callbacks can be used to control what happens when a cell is entered. In this example, we want to skip (disallow) data entry in the first column if the corresponding State is CA.

1. The Entry Callback method

```
C_LONGINT($1)  // AreaList Pro object reference
C_LONGINT($2)  // entry cause
C_LONGINT($3)  // only useful when fields are being displayed
vRow:=AL_GetAreaLongProperty ($1;ALP_Area_EntryRow)  // find out which cell
vCol:=AL_GetAreaLongProperty ($1;ALP_Area_EntryColumn)
ARRAY POINTER($ArrayNames;0)
$error:=AL_GetObjects ($1;ALP_Object_Columns;$ArrayNames)
If (vCol=1)  // city
   If ($ArrayNames{2}->{vRow}="CA")  // pointer to second column array (state)
      AL_SetAreaLongProperty ($1;ALP_Area_EntrySkip;1)
   End if
End if
```

2. Tell AreaList Pro when to call the callback method

```
AL_SetAreaTextProperty (eList;ALP_Area_CallbackMethEntryStart;"EntryCallback")
```

For more information about callbacks, see the Callbacks chapter.

Example 9: Using an Event callback instead of the On Plug in Area event

This example shows how a generic event callback project method can be installed to replace the On Plug in Area event.

This is performed with the ***AL_SetAreaTextProperty*** command using the ALP_Area_CallbackMethOnEvent property, which instructs AreaList Pro to call the ***EventCallBack09*** project method instead of sending the On Plug in Area event to the object method and form method:

```
Case of
  : (Form event Code=On Load)
    AL_SetAreaTextProperty (eList;ALP_Area_CallbackMethOnEvent;"EventCallBack09")  // set event callback
End case
```

The ***EventCallBack09*** method checks various AreaList Pro Events to find out what triggered the callback and what to do about it:

```
C_LONGINT($0)  // object method and form method will not be executed if 0
C_LONGINT($1)  // AreaList Pro area
C_LONGINT($2)  // AreaList Pro event
C_LONGINT($3)  // 4D event
C_LONGINT($4)  // last clicked column (or column under the pointer for mouse moved event)
C_LONGINT($5)  // last clicked row (or row under the pointer for mouse moved event)
C_LONGINT($6)  // modifiers
ARRAY LONGINT(aRows;0)
$error:=AL_GetObjects ($1;ALP_Object_Selection;aRows)  // get the rows selected by user
evtUpdateText (->vItem;->aCity;->aState)  // event description
$0:=0  // event handled
```

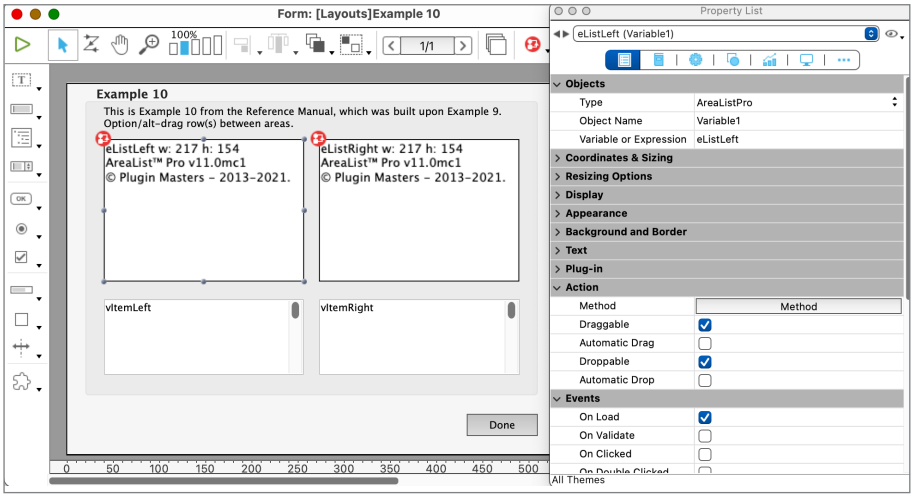
The ***evtUpdateText*** method updates the variable at the bottom of the list:

```
C_POINTER($1)  // -> variable
C_POINTER($2)  // -> city array
C_POINTER($3)  // -> state array
C_LONGINT($i)
$1->:=""
For ($i;1;Size of array(aRows))  // look at each row selected by user (aRows populated by event callback)
  $1->:=$1->+$2->{aRows{$i}}+" "+$3->{aRows{$i}}  // plug values in the text variable
  If ($i<Size of array(aRows))  // not the last item
    $1->:=$1->+" - "  // separator
  End if
End for
```

Example 10: Drag and drop between areas

This example demonstrates how dragging rows between AreaList Pro areas (and within the same area) can be allowed with the Alt/Option key.

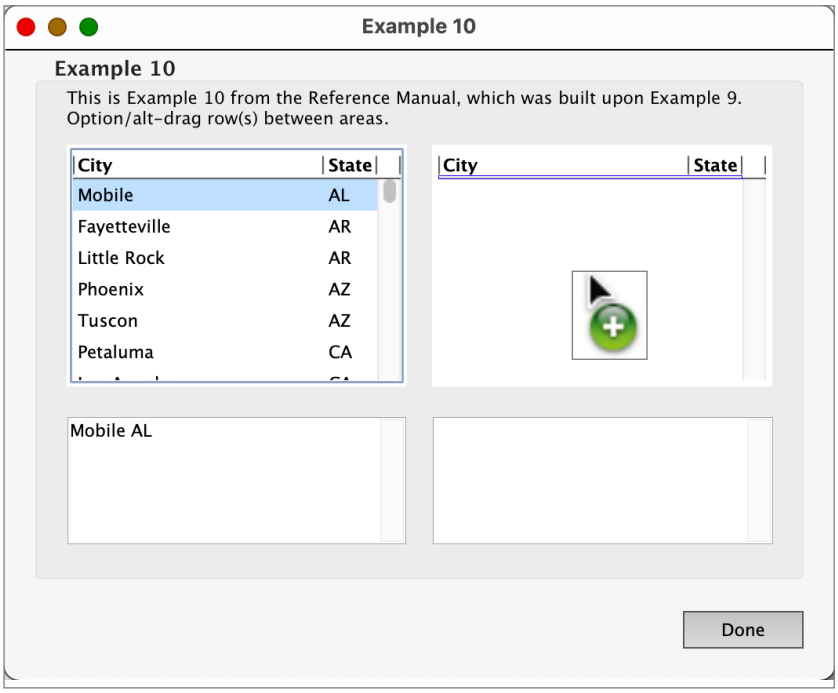
There are two AreaList Pro areas and two text areas on the form:



The Draggable and Droppable options have been selected for both of the AreaList Pro areas, so that rows can be dragged and dropped freely between the areas.

When it is initially opened, the area on the left contains a list of cities and states, and the area on the right is empty.

You can move a row (or multiple rows) from the left area to the right area or conversely, or within the same area by holding down the Alt or Option key as you drag it:



When a row is dragged, the text areas are updated to display what was dragged, and the two areas are updated: the dragged row is removed form the source area and added to the destination area. Within the same area, the rows are moved to the destination.

Note: Starting with 4D v11, drop is always handled in the destination process.

The area object methods react to On Load and On Drop events.

Left area:

```
Case of
  : (Form event Code=On Load)  // initialize the AreaList Pro object
    ALL RECORDS([Cities])  // load all records in the Cities table
    SELECTION TO ARRAY([Cities]City;aCityLeft;[Cities]State;aStateLeft)  // copy field values into arrays
    $error:=AL_AddColumn (Self->;->aCityLeft)  // display array in column 1
    $error:=AL_AddColumn (Self->;->aStateLeft)  // display array in column 2
    dragAreaSetup (Self->)
    AL_SetAreaTextProperty (Self->;ALP_Area_CallbackMethOnEvent;"EventCallBack10")
      // set event callback
    AL_SetAreaLongProperty (Self->;ALP_Area_SelRow;1)  // row 1 selected
    vItemLeft:=aCityLeft{1}+", "+aStateLeft{1}  // initialize text variable to row 1 values
  : (Form event Code=On Drop)
    AlpOnDrop
End case
```

The Event callback method ***EventCallBack10*** is a slight variation from ***EventCallBack09*** because we have two areas, with a text variable for each:

```
If ($1=eListLeft)  // left area
  evtUpdateText (->vItemLeft;->aCityLeft;->aStateLeft)
Else  // right area
  evtUpdateText (->vItemRight;->aCityRight;->aStateRight)
End if
```

Right area:

```
Case of
  : (Form event Code=On Load)  //initialize the AreaList Pro object
    ARRAY TEXT(aCityRight;0)  //empty arrays in this area
    ARRAY TEXT(aStateRight;0)
    $error:=AL_AddColumn (Self->;->aCityRight)  //display array in column 1
    $error:=AL_AddColumn (Self->;->aStateRight)  //display array in column 2
    dragAreaSetup (Self->)
    AL_SetAreaTextProperty (Self->;ALP_Area_CallbackMethOnEvent;"EventCallBack10")
        //set event callback
    vItemRight:=""
  : (Form event Code=On Drop)
    AlpOnDrop
End case
```

The ***dragAreaSetup*** project method sets headers, width, etc. as well as drag and drop properties:

```
AL_SetAreaTextProperty ($1;ALP_Area_DragSrcRowCodes;"drag")  //set source access code
AL_SetAreaTextProperty ($1;ALP_Area_DragDstRowCodes;"drag")  //set destination access code
AL_SetAreaLongProperty ($1;ALP_Area_DragRowMultiple;1)  //multiple rows dragging
AL_SetAreaLongProperty ($1;ALP_Area_DragRowOnto;0)  //Between rows (disable On row)
```

The ***AlpOnDrop*** project method takes care of the whole drag and drop action:

```
C_STRING(20;$selectedObject)
C_LONGINT($error)
C_LONGINT($dragDstRow;$dragDstCol;$dragSource;$dragSrcRow;$dragSrcCol;$dragDest;$dragType)
$dragDstRow:=AL_GetAreaLongProperty (Self->;ALP_Area_DragDstRow)
$dragDstCol:=AL_GetAreaLongProperty (Self->;ALP_Area_DragDstCol)
$dragSource:=AL_GetAreaLongProperty (Self->;ALP_Area_DragSrcArea)
    //0 if drag is not from AreaList Pro (or not from same 4D instance)
If ($dragSource#0)  //drop from $dragSource AreaList Pro area
  $dragSrcRow:=AL_GetAreaLongProperty (Self->;ALP_Area_DragSrcRow)
  $dragSrcCol:=AL_GetAreaLongProperty (Self->;ALP_Area_DragSrcCol)
  $dragType:=AL_GetAreaLongProperty ($dragSource;ALP_Area_DragDataType)
  $dragDest:=Self->
  ARRAY LONGINT(aRows;0)
  $error:=AL_GetObjects ($dragSource;ALP_Object_Selection;aRows)  //get the rows selected by user
  If ($dragType=1)
    $selectedObject:="Row"
  Else
    $selectedObject:="Column"
  End if
```

```
    Case of
        : ($dragSource=Self->)  // drag within the same area
        If (Self->=eListRight)
            evtDragWithin (->aRows;->aCityRight;->aStateRight;$dragDstRow;Self->)
        Else
            evtDragWithin (->aRows;->aCityLeft;->aStateLeft;$dragDstRow;Self->)
        End if
        : ($dragSource=eListLeft)  // source is left area
            evtRowsDragged ($dragSource;->aCityLeft;->aStateLeft;->aCityRight;->aStateRight;$dragDstRow)
            vItemLeft:=""
            evtUpdateText (->vItemRight;->aCityRight;->aStateRight)
    Else  // source is right area
        evtRowsDragged ($dragSource;->aCityRight;->aStateRight;->aCityLeft;->aStateLeft;$dragDstRow)
        vItemRight:=""
        evtUpdateText (->vItemLeft;->aCityLeft;->aStateLeft)
    End case
End if
```

The ***evtDragWithin*** project method updates the left or right area after a row(s) drag and drop from itself:

```
C_POINTER($1)  // the rows dragged by user
C_POINTER($2)  // -> source city array on itself
C_POINTER($3)  // -> source state array on itself
C_LONGINT($4)  // destination row
C_LONGINT($5)  // area reference
C_LONGINT($error;$i;$x)
ARRAY TEXT($tmpText_1;0)
ARRAY TEXT($tmpText_2;0)
ARRAY LONGINT(aRowsToSelect;Size of array($1->))  // select dragged rows once moved
For ($i;1;Size of array($2->)+1)  // all rows in the area + 1 in case drop is below the last row
    If ($i=$4)  // position where to insert the dragged row(s)
        For ($x;1;Size of array($1->))  // selected (dragged) rows
            APPEND TO ARRAY($tmpText_1;$2->{$1->{$x}})  // add city
            APPEND TO ARRAY($tmpText_2;$3->{$1->{$x}})  // and state
            aRowsToSelect{$x}:=Size of array($tmpText_1)  // this row will be selected
        End for
    End if
    If ($i<=Size of array($2->)) & (Find in array($1->;$i)=-1)  // current row is not part of the drag selection
        APPEND TO ARRAY($tmpText_1;$2->{$i})  // add city
        APPEND TO ARRAY($tmpText_2;$3->{$i})  // and state
    End if
End for
COPY ARRAY($tmpText_1;$2->)  // new city array
COPY ARRAY($tmpText_2;$3->)  // new state array
```

```
AL_SetAreaLongProperty ($5;ALP_Area_UpdateData;0)  // update destination area
$error:=AL_SetObjects ($5;ALP_Object_Selection;aRowsToSelect)  // select new rows
    // no need to update text - selection is unchanged
```

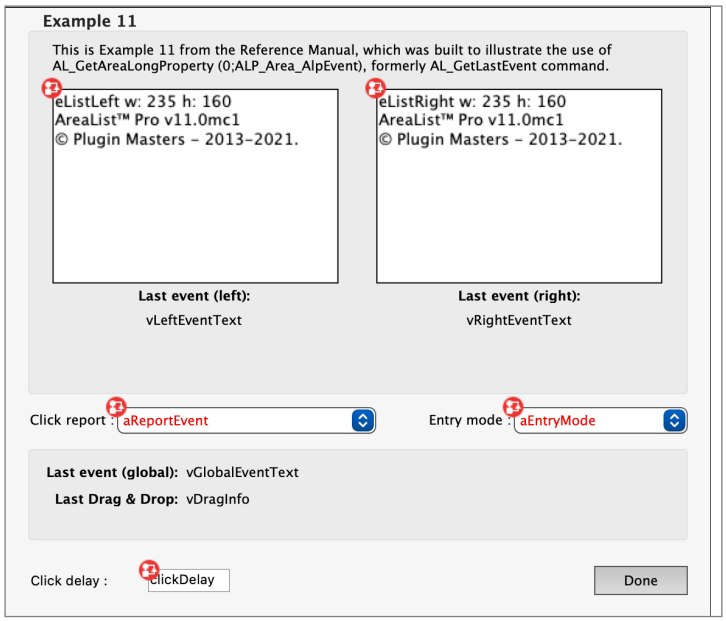
The ***evtRowsDragged*** project method both areas after a row(s) drag and drop between areas:

```
C_LONGINT($1)  // source AreaList Pro area
C_POINTER($2;$3)  // -> source city array ; -> source state array
C_POINTER($4;$5)  // -> destination city array ; -> destination state array
C_LONGINT($6)  // destination row
C_LONGINT($error)
C_LONGINT($i)
INSERT IN ARRAY($4->;$6;Size of array(aRows))  // insert rows at the drop destination row
INSERT IN ARRAY($5->;$6;Size of array(aRows))
ARRAY LONGINT(aRowsToSelect;Size of array(aRows))  // select dragged rows in destination area
For ($i;Size of array(aRows);1;-1)
    // look backwards at each row selected by user (aRows populated by event callback)
  $4->{$6-1+$i}:=$2->{aRows{$i}}  // city
  $5->{$6-1+$i}:=$3->{aRows{$i}}  // state
  DELETE FROM ARRAY($2->;aRows{$i})  // delete source city
  DELETE FROM ARRAY($3->;aRows{$i})  // delete source state
  aRowsToSelect{$i}:=$6-1+$i
End for
AL_SetAreaLongProperty ($1;ALP_Area_UpdateData;0)  // update source area (modified array size)
eDestination:=AL_GetAreaLongProperty ($1;ALP_Area_DragDstArea)  // destination area
AL_SetAreaLongProperty (eDestination;ALP_Area_UpdateData;0)  // update destination area (modified array size)
$error:=AL_SetObjects (eDestination;ALP_Object_Selection;aRowsToSelect)  // select new rows
$error:=AL_GetObjects (eDestination;ALP_Object_Selection;aRows)  // get the rows for evtUpdateText
```

Example 11: Determining a user's action

The ***AL_GetLastEvent*** function lets you find out exactly what actions the user has taken on an AreaList Pro area. This example demonstrates how to use this useful feature.

The example form contains two AreaList Pro areas and four text variables which will display information about the user's actions. It also includes two Popup Drop-down lists to select the Click event report type and the Click type to trigger entry mode:



The user's actions are monitored through the use of an On Timer event, which is set up in the form method. In the On Load phase the timer is set to execute every 10 ticks:

```
Case of
  : (Form event Code=On Load)
    vGlobalEventText:=""
    vDragInfo:=""
    SET TIMER(10)
```

Then, in the On Timer form event, the user's last action is captured by ALP_Area_AlpEvent with areaRef = 0 for all areas and passed to the ***AlpEventText*** method for parsing:

```
  : (Form event Code=On Timer)
    AlpEventText (AL_GetAreaLongProperty (0;ALP_Area_AlpEvent);->vGlobalEventText)
        // 0 = all areas
End case
```

The "Click report" Popup sets the click report type through the ALP_Area_SelClick property for both areas, the value being the selected menu line number minus one:

```
Case of
  : (Form event Code=On Clicked)
    AL_SetAreaLongProperty (eListLeft;ALP_Area_SelClick);ReportEvent_R-1)
    AL_SetAreaLongProperty (eListRight;ALP_Area_SelClick);ReportEvent_R-1)
End case
```

The "Entry mode" Popup sets the entry trigger click type through the ALP_Area_EntryClick property for each area, the value being the selected menu line number minus one:

```
Case of
  : (Form event Code=On Clicked)  // we need to set the two areas separately
    AL_SetAreaLongProperty (eListLeft;ALP_Area_EntryClick);EntryMode_R-1)
    AL_SetAreaLongProperty (eListRight;ALP_Area_EntryClick);EntryMode_R-1)
End case
```

The ***AlpEventText*** method receives the event code and a pointer to the event text variable, and returns the appropriate response:

```
C_LONGINT($1)  // event code
C_POINTER($2)  // to the event description (text)
Case of
  : ($1=AL Empty Area Double click)
    $2->:="Double-click in an empty part of the area (without displayed data)"
  : ($1=AL Double click event)
    $2->:="Double-click"
  : ($1=AL Empty Area Single click)
    $2->:="Single-click in an empty part of the area (without displayed data)"
   etc.
  : ($1=0)  // No event, $2-> unchanged
Else
    $2->:="Unknown event"
End case
If ($1#0)  // add the event code
  $2->:=$2->+" ("+String($1)+")"+Char(Carriage return)
End if
```

Every 10 ticks the user's last action will be sent to the ***AlpEventText*** method for analysis, and if he has done something, this will be reported in the variables displayed on the form.

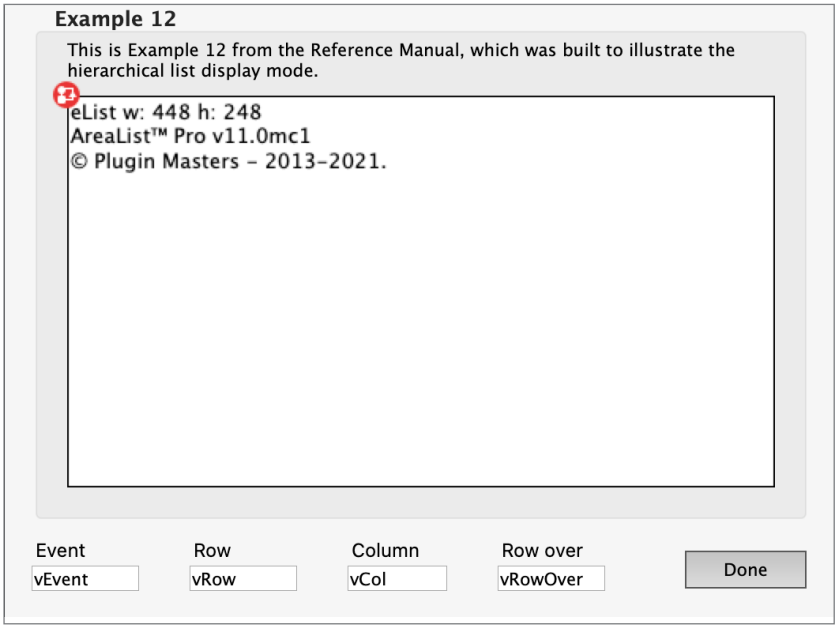
The Event callback method ***EventCallBack11*** is a slight variation from ***EventCallBack10***, including a text variable for each area event, using the received $2 parameter and the same ***AlpEventText*** project method as above:

```
If ($1=eListLeft)  // left area
  evtUpdateText (->vItemLeft;->aCityLeft;->aStateLeft)
  AlpEventText ($2;->vLeftEventText)
  vLeftEventText:=vLeftEventText+vItemLeft
Else  // right area
  evtUpdateText (->vItemRight;->aCityRight;->aStateRight)
  AlpEventText ($2;->vRightEventText)
  vRightEventText:=vRightEventText+vItemRight
End if
```

Example 12: Using Hierarchical Lists

This example illustrates how to display information in a hierarchical list with AreaList Pro.

The form contains an AreaList Pro area and fields to display information about the data:



In the On Load event on the object method for the AreaList Pro area, a set of arrays is created to display the data:

```
ALL RECORDS([Cities])  //load all records in the Cities table
ORDER BY([Cities];[Cities]State;>;[Cities]City;>)
SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState;[Cities]Id_letter;aLetterId)
    //copy field values into arrays
```

The whole presentation is based on hierarchical indentation initiated by the one-dimensional array (displayLevel) that contains the hierarchical level of each item displayed. Every father has an original level 0 that is incremented by 1 for each son, grandson, etc:

```
C_TEXT($oldstate;$oldId)
ARRAY LONGINT(displayLevel;0)
For ($i;1;Size of array(aState))
  Case of
    : ($oldstate#aState{$i})
      APPEND TO ARRAY($tmplistArray;aState{$i})
      APPEND TO ARRAY(displayLevel;0)
      APPEND TO ARRAY($tmplistArray;aLetterId{$i})
      APPEND TO ARRAY(displayLevel;1)
      APPEND TO ARRAY($tmplistArray;aCity{$i})
      APPEND TO ARRAY(displayLevel;2)
    : ($oldId#aLetterId{$i})
      APPEND TO ARRAY($tmplistArray;aLetterId{$i})
      APPEND TO ARRAY(displayLevel;1)
```

```
            APPEND TO ARRAY($tmplistArray;aCity{$i})
            APPEND TO ARRAY(displayLevel;2)
        : ($oldId=aLetterId{$i})
            APPEND TO ARRAY($tmplistArray;aCity{$i})
            APPEND TO ARRAY(displayLevel;2)
    End case
    $oldstate:=aState{$i}
    $oldId:=aLetterId{$i}
End for
COPY ARRAY($tmplistArray;aState)
```

Here, we expand the "CA" state content:

```
ARRAY LONGINT($expanded;Size of array($tmplistArray))
$expanded{17}:=1
$expanded{18}:=1
$expanded{20}:=1
$expanded{23}:=1
```

Display arrays in the AreaList Pro area:

```
$error:=AL_AddColumn (Self->;->aState;0)
$error:=AL_AddColumn (Self->;->displayLevel;0)
```

Some formatting:

```
AL_SetColumnTextProperty (Self->;1;ALP_Column_HeaderText;"State/City")
AL_SetColumnTextProperty (Self->;2;ALP_Column_HeaderText;"Level")
AL_SetAreaRealProperty (Self->;ALP_Area_HierIndent;20)  // set hierarchical indentation
DEMO_Default (Self->)  // general display settings
AL_SetColumnTextProperty (Self->;3;ALP_Column_Format;"0")  // specify Level column format
```

Set the event callback method:

```
AL_SetAreaTextProperty (Self->;ALP_Area_CallbackMethOnEvent;"EventCallBack")
```

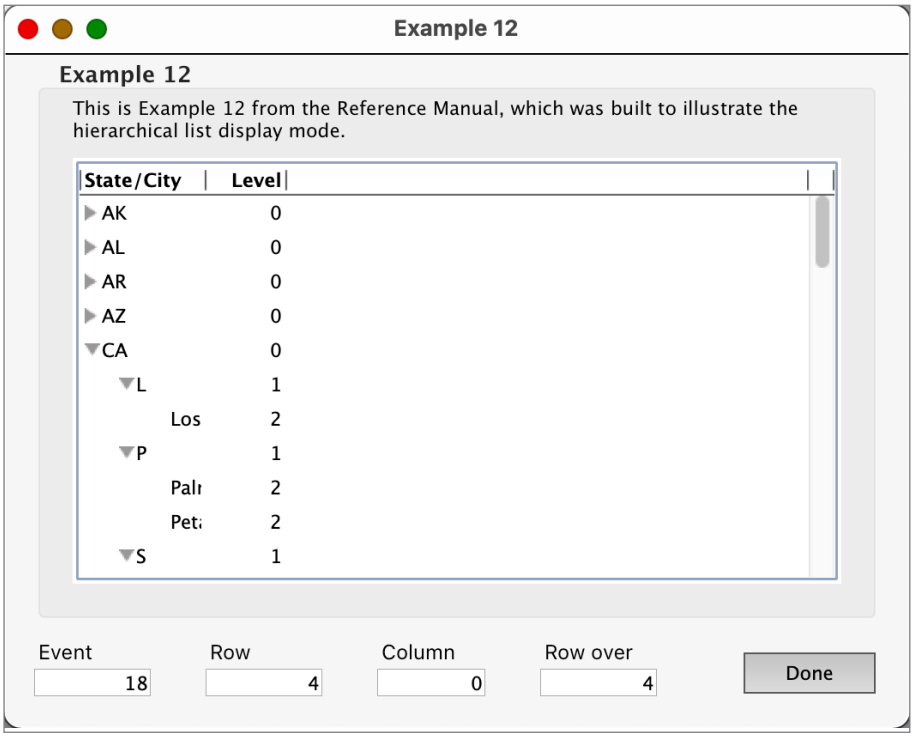
Add some variables to the ***EventCallBack*** method:

```
C_LONGINT($0)  // object method and form method will not be executed if 0
C_LONGINT($1)  // AreaList Pro area
C_LONGINT($2)  // AreaList Pro event
C_LONGINT($3)  // 4D event
C_LONGINT($4)  // last clicked column (or column under the pointer for mouse moved event)
C_LONGINT($5)  // last clicked row (or row under the pointer for mouse moved event)
C_LONGINT($6)  // modifiers
vEvent:=$2
vRow:=$5
vCol:=$4
vRowOver:=AL_GetAreaLongProperty ($1;ALP_Area_RollOverRow)
```

Now for the important part - tell AreaList Pro that we want to display these arrays in a hierarchical list:

```
$error:=AL_SetObjects2 (Self->;ALP_Object_Hierarchy;displayLevel;$expanded)
```

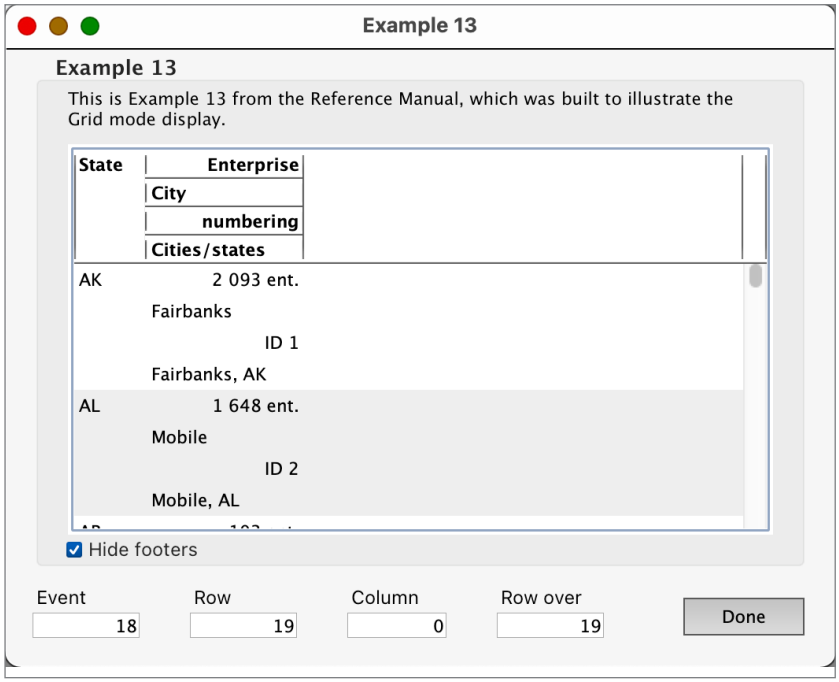
The end result looks something like this:



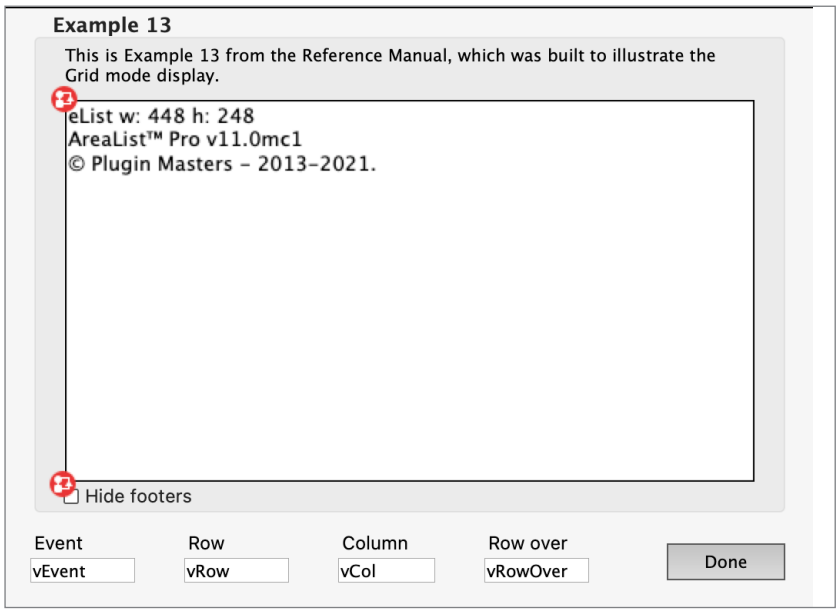
See the Hierarchical Lists topic for more information.

Example 13: Grids

Grids offer some interesting formatting possibilities tot he developer. Rather than displaying your data in simple rows and columns, the cells can be organised into groups - for example:



The form holds what looks like a perfectly ordinary AreaList Pro area:



The magic is in the On Load event of the area's object method…

First, create the arrays and add them to the AreaList Pro area:

```
Compiler  // reset the arrays
ALL RECORDS([Cities])  // load all records in the Cities table
ORDER BY([Cities];[Cities]State;>;[Cities]City;>)  // copy field values into arrays
SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState;[Cities]CityState;aCityState)
INSERT IN ARRAY(enterprise;1;Size of array(aState))
INSERT IN ARRAY(numRow;1;Size of array(aState))
For ($i;1;Size of array(aState))
   numRow{$i}:=$i
   enterprise{$i}:=((Random%(2100+$i))+100)
End for
 // Display arrays in the AreaList Pro area
$error:=AL_AddColumn (Self->;->aState)
$error:=AL_AddColumn (Self->;->aState)
$error:=AL_AddColumn (Self->;->aState)
$error:=AL_AddColumn (Self->;->aState)
$error:=AL_AddColumn (Self->;->aState)
 // Specify the values for the column headers and footers
AL_SetColumnTextProperty (Self->;1;ALP_Column_HeaderText;"State")
AL_SetColumnTextProperty (Self->;2;ALP_Column_HeaderText;"City")
AL_SetColumnTextProperty (Self->;3;ALP_Column_HeaderText;"Enterprise")
AL_SetColumnTextProperty (Self->;4;ALP_Column_HeaderText;"numbering")
AL_SetColumnTextProperty (Self->;5;ALP_Column_HeaderText;"Cities/states")
AL_SetColumnTextProperty (Self->;1;ALP_Column_FooterText;"States")
AL_SetColumnTextProperty (Self->;2;ALP_Column_FooterText;"Cities")
AL_SetColumnTextProperty (Self->;3;ALP_Column_FooterText;"Enterprise")
AL_SetColumnTextProperty (Self->;4;ALP_Column_FooterText;"ID")
AL_SetColumnTextProperty (Self->;5;ALP_Column_FooterText;"City & state")
```

Next, build 2-dimensional arrays for the grid layout:

```
ARRAY INTEGER($aiGrid;3;8)
For ($i;1;8)
   $aiGrid{2}{$i}:=1  // column span
   $aiGrid{3}{$i}:=1  // row span
End for
$aiGrid{1}{1}:=1  // State
$aiGrid{1}{2}:=3  // Enterprise
$aiGrid{1}{4}:=2  // City
$aiGrid{1}{6}:=4  // Numbering
$aiGrid{1}{8}:=5  // City - State
```

We want column 1 (the state) to span 4 lines:

```
$aiGrid{3}{1}:=4  // row span
```

Tell AreaList Pro how many columns to set in the grid, and add the grid to the area:

```
AL_SetAreaLongProperty (Self->;ALP_Area_ColsInGrid;2)  // 2 columns in a grid
$error:=AL_SetObjects (Self->;ALP_Object_Grid;$aiGrid)  // add the grid arrays to the area
```

Finally, some formatting is done and the ***EventCallBack*** method is set as the event callback projet method.

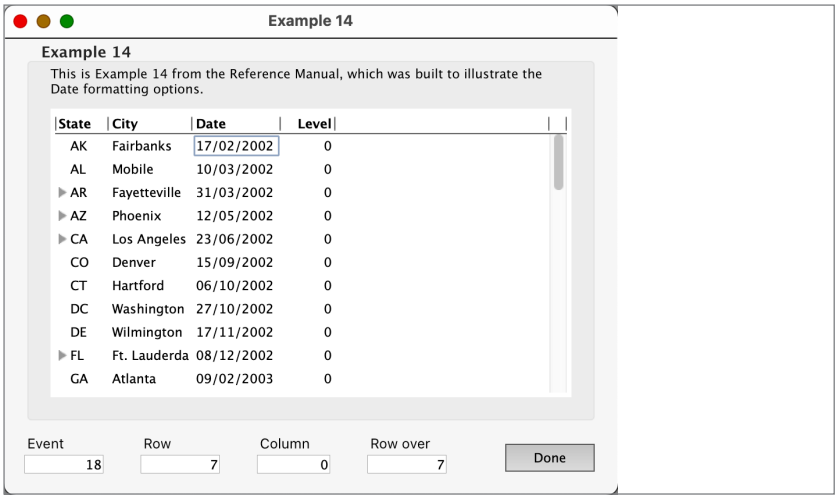
For a more detailed explanation of how to set up the grid arrays, please see the Grids topic.

Example 14: Date Formatting Options

This example demonstrates an example of how you can offer a simplified data entry option through the use of callbacks.It allows the user to speedily enter date information by typing in some numbers; they can enter just one digit, for example, and the method will complete the date using the current year and month.

For example, if the current date is June 1st 2011, and the user enters "5", it will complete the date as June 5, 2011.

The third column in the example is an enterable date field:



If you enter a complete date, this will be accepted as-is, as long as it is a valid date. But if you enter a number in one to 8 digits, it will be converted to a date as shown in the following table. In these examples, the current system date is July 21st, 2011:

No. of digits entered	Mapped to	E.G.	Result		
			French OS	US OS	UK OS
1	D	5	2011/07/05	07/05/2011	05/07/2011
2	DD	31	2011/07/31	07/31/2011	31/07/2011
3	DDM	131	2011/01/13	13/01/2011	01/13/2011
4	DDMM	1301	2011/01/13	13/01/2011	01/13/2011
5	DDMMY	13015	2005/01/13	01/13/2005	13/01/2005
6	DDMMYY	130112	2012/01/13	01/13/2012	13/01/2012
8	DDMMYYYY	13012012	2012/01/13	01/13/2012	13/01/2012

If you enter values directly (double-click in the 3rd column in a date cell), and if the value you enter is not in standard date format, the ***alpEdit_DateEntry*** method is called via the ***ExitEntry*** callback method.

The callback method is installed in the **On Load** phase of the AreaList Pro object on the form:

```
AL_SetAreaTextProperty (Self->;ALP_Area_CallbackMethEntryEnd;"ExitCallbackDate")
```

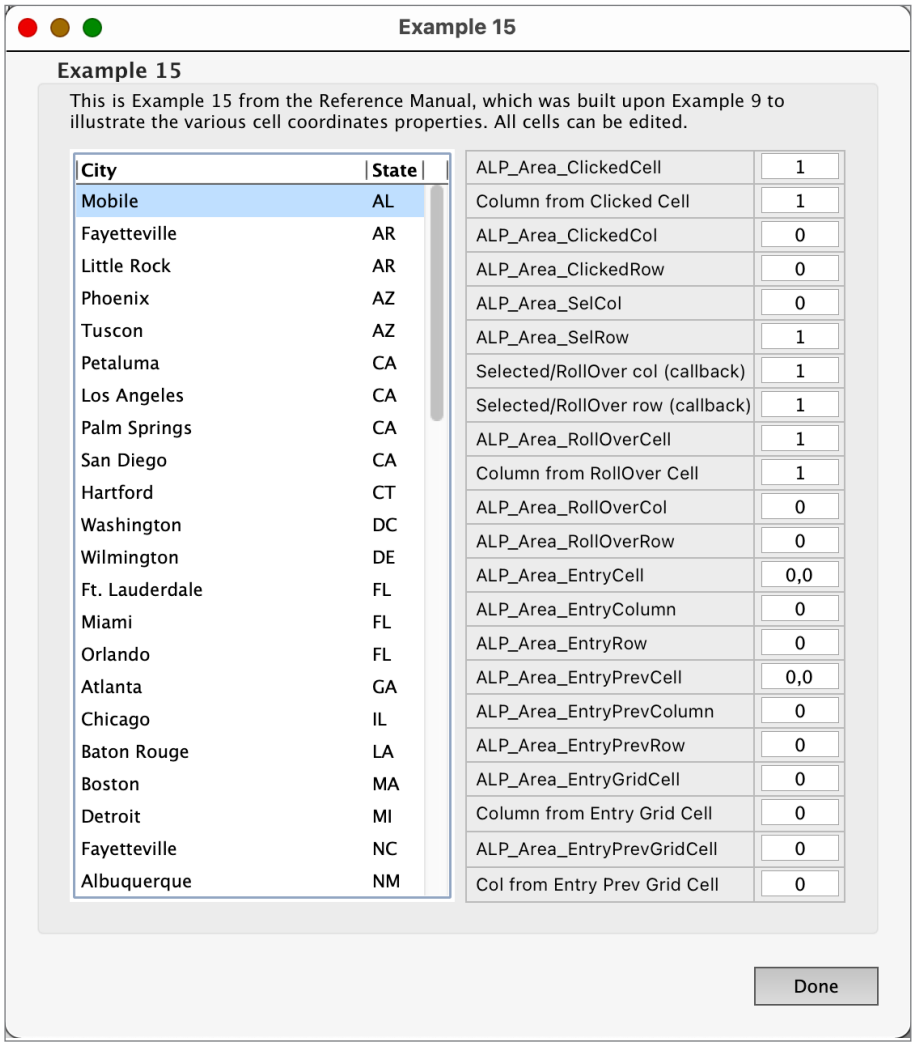
The ***ExitCallbackDate*** method will be invoked when data entry in a cell in the Date column ends.

After the callback method has done its calculations, the resulting date is poked into the same cell with the command:

```
$error:=AL_SetAreaPtrProperty ($1;ALP_Area_EntryValue;->$date)
```

Example 15: Cell coordinates properties

This example demonstrates the use of all cell coordinates properties:



An entry callback and event callback methods are set:

```
AL_SetAreaTextProperty (eList;ALP_Area_CallbackMethEntryStart;"EntryCallback15")
    // set entry callback to project method EntryCallback15
AL_SetAreaTextProperty (eList;ALP_Area_CallbackMethOnEvent;"EventCallBack15")
    // set event callback to project method EventCallBack15
```

The ***EntryCallback15*** method assigns the Entry coordinates properties values to the matching variables, then updates the layout:

```
C_LONGINT($0)  //object method and form method will not be executed if 0
C_LONGINT($1)  //AreaList Pro area
C_LONGINT($2)  //AreaList Pro event
C_LONGINT($3)  //4D event
C_LONGINT($4)  //column — last clicked column (or rollover for event 18)
C_LONGINT($5)  //row — last clicked row (or rollover for event 18)
C_LONGINT($6)  //modifiers

v15SelColCallback:=$4
v15SelRowCallback:=$5
Example15UpdateVariables
$0:=0  //event handled
```

Event callbacks receive the last clicked column and row (or rollover colum and row for event 18 "mouse moved") in parameters $4 and $5. The ***EntryCallback15*** method assigns them to the v15SelColCallback and v15SelRowCallback variables for display:

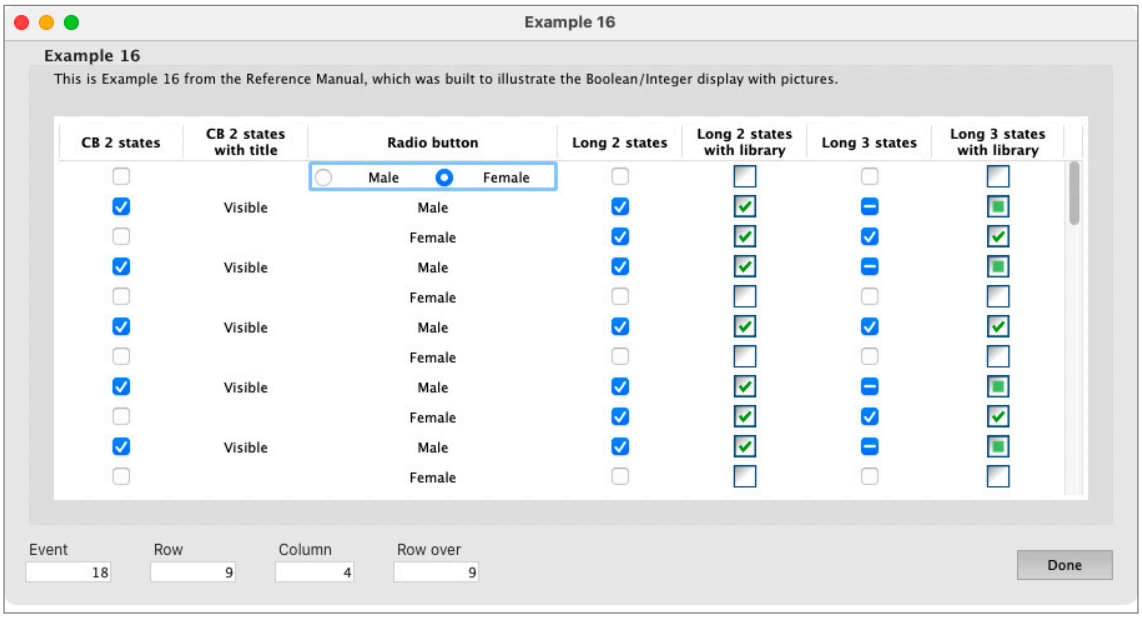
The ***Example15UpdateVariables*** method (also called during On Load) updates the other variables through properties:

```
v15ClickedCell:=AL_GetAreaLongProperty(eList; ALP_Area_ClickedCell)
v15ClickedCellColFromCell:=AL_GetColumnLongProperty(eList;v15ClickedCell;ALP_Column_FromCell)

v15ClickedCol:=AL_GetAreaLongProperty(eList; ALP_Area_ClickedCol)
v15ClickedRow:=AL_GetAreaLongProperty(eList; ALP_Area_ClickedRow)
v15SelCol:=AL_GetAreaLongProperty(eList; ALP_Area_SelCol)
v15SelRow:=AL_GetAreaLongProperty(eList; ALP_Area_SelRow)

v15RollOverCol:=AL_GetAreaLongProperty(eList; ALP_Area_RollOverCol)
v15RollOverRow:=AL_GetAreaLongProperty(eList; ALP_Area_RollOverRow)
v15RollOverCell:=AL_GetAreaLongProperty(eList; ALP_Area_RollOverCell)
v15RollOverCellColFromCell:=AL_GetColumnLongProperty(eList;v15RollOverCell;ALP_Column_FromCell)

POST OUTSIDE CALL(Frontmost process)  //update displayed variables
```

See also about emulating the user's typeahead in the Typeahead section.

Example 16: Boolean/Integer Display with Pictures

This example demonstrates the various ways of displaying Booleans and integers:.



The On Load event for the AreaList Pro area first sets up the variables for the columns, and fetches the checkbox images:

```
Case of
  : (Form event code=On Load)  //initialize the AreaList Pro object
    $Tpath:=Get 4D folder(Current resources folder)+"Images"+Folder separator+"Library"+Folder separator
    READ PICTURE FILE($Tpath+"CheckboxYes.png"; $pPictYes)
    READ PICTURE FILE($Tpath+"CheckboxNo.png"; $pPictNo)
    READ PICTURE FILE($Tpath+"CheckboxMaybe.png"; $pPictMaybe)

    ALL RECORDS([Cities])  //load all records in the Cities table
    SELECTION TO ARRAY([Cities]Bool_3;aBoolean_1;[Cities]Bool_2;aBoolean_2;[Cities]Bool_3;aBoolean_3;[Cities]Long_1;
aLongBoolean2states;[Cities]Long_1;aLongBoolean2statesLib;[Cities]Long_1;aLongBoolean3states;[Cities]Long_1;
aLongBoolean3statesLib)  //copy field values into arrays

    //LONGINT STANDARD 2 STATES
    For ($i; 1; Size of array(aLongBoolean2states))
      If (aLongBoolean2states{$i}>0)  //True or false, no 3rd state
        aLongBoolean2states{$i}:=1
        aLongBoolean2statesLib{$i}:=1
      End if
    End for

    $Lerror:=AL_SetIcon(eList; 2113; $pPictYes)  // add picture for the True value to the AreaList Pro picture library with ID = 2113
```

```
$Lerror:=AL_SetIcon(eList; 2114; $pPictNo)  // add picture for the False value to the AreaList Pro picture library with ID = 2114
$Lerror:=AL_SetIcon(eList; 2115; $pPictMaybe)  // add picture for the Mixed value to the AreaList Pro picture library with ID = 2115

AL_SetAreaLongProperty(eList; ALP_Area_EntryClick; 2)  //set the entry mode to 2 clicks

$Lerror:=AL_AddColumn(eList; ->aBoolean_1)  //display array in AreaList Pro object
$Lerror:=AL_AddColumn(eList; ->aBoolean_2)  //display array in AreaList Pro object
$Lerror:=AL_AddColumn(eList; ->aBoolean_3)  //display array in AreaList Pro object
$Lerror:=AL_AddColumn(eList; ->aLongBoolean2states)  //display array in AreaList Pro object
$Lerror:=AL_AddColumn(eList; ->aLongBoolean2statesLib)  //display array in AreaList Pro object
$Lerror:=AL_AddColumn(eList; ->aLongBoolean3states)  //display array in AreaList Pro object
$Lerror:=AL_AddColumn(eList; ->aLongBoolean3statesLib)  //display array in AreaList Pro object
```

Then it formats the columns.

Column 1- two-state Boolean:

```
AL_SetColumnTextProperty(eList; 1; ALP_Column_HeaderText; "CB 2 states")  //column header
AL_SetColumnLongProperty(eList; 1; ALP_Column_EntryControl; 0)  //checkBox without title
AL_SetColumnLongProperty(eList; 1; ALP_Column_DisplayControl; 0)  //display a Boolean as a normal-sized checkbox
/ 1 = small checkbox without title / 2 = mini checkbox without title
AL_SetColumnRealProperty(eList; 1; ALP_Column_Width; 100)
AL_SetColumnLongProperty(eList; 1; ALP_Column_Enterable; 1)
```

Column 2 - Two-state checkbox with text description:

```
AL_SetColumnTextProperty(eList; 2; ALP_Column_HeaderText; "CB 2 states\rwith title")  //column header
AL_SetColumnLongProperty(eList; 2; ALP_Column_EntryControl; 1)  //checkBox with title
AL_SetColumnTextProperty(eList; 2; ALP_Column_Format; "Visible;")  //specify the labels for the checkbox
AL_SetColumnLongProperty(eList; 2; ALP_Column_DisplayControl; -1)  //default formatted value
AL_SetColumnRealProperty(eList; 2; ALP_Column_Width; 100)
AL_SetColumnLongProperty(eList; 2; ALP_Column_Enterable; 1)
```

Column 3 - Radio button options:

```
AL_SetColumnTextProperty(eList; 3; ALP_Column_HeaderText; "Radio button")  //column header
AL_SetColumnLongProperty(eList; 3; ALP_Column_EntryControl; 2)  //radio buttons
AL_SetColumnTextProperty(eList; 3; ALP_Column_Format; "Male;Female")  //specify the labels for the radio buttons
AL_SetColumnLongProperty(eList; 3; ALP_Column_DisplayControl; -1)  //default formatted value
AL_SetColumnRealProperty(eList; 3; ALP_Column_Width; 200)
AL_SetColumnLongProperty(eList; 3; ALP_Column_Enterable; 1)
```

Column 4 - Two-state checkbox:

```
AL_SetColumnTextProperty(eList; 4; ALP_Column_HeaderText; "Long 2 states")  //column header
AL_SetColumnLongProperty(eList; 4; ALP_Column_EntryControl; 0)  //checkBox 2 states
AL_SetColumnLongProperty(eList; 4; ALP_Column_DisplayControl; 0)  //display a Boolean as a normal-sized checkbox
AL_SetColumnRealProperty(eList; 4; ALP_Column_Width; 100)
AL_SetColumnLongProperty(eList; 4; ALP_Column_Enterable; 1)
```

Column 5 - Longint 2 states with library:

```
AL_SetColumnTextProperty(eList; 5; ALP_Column_HeaderText; "Long 2 states\rwith library")  //column header
AL_SetColumnLongProperty(eList; 5; ALP_Column_EntryControl; 0)  //checkBox 2 states
AL_SetColumnTextProperty(eList; 5; ALP_Column_Format; "2113;2114")  //specify column format
AL_SetColumnLongProperty(eList; 5; ALP_Column_DisplayControl; 4)  //library pictures display
AL_SetColumnRealProperty(eList; 5; ALP_Column_Width; 100)
AL_SetColumnLongProperty(eList; 5; ALP_Column_Enterable; 1)
```

Column 6 - Longint displayed as a three-state checkbox:

```
AL_SetColumnTextProperty(eList; 6; ALP_Column_HeaderText; "Long 3 states")  //column header
AL_SetColumnLongProperty(eList; 6; ALP_Column_EntryControl; 1)  //checkBox 3 states
AL_SetColumnLongProperty(eList; 6; ALP_Column_DisplayControl; 0)  //display a Boolean as a normal-sized checkbox
AL_SetColumnRealProperty(eList; 6; ALP_Column_Width; 100)
AL_SetColumnLongProperty(eList; 6; ALP_Column_Enterable; 1)
```

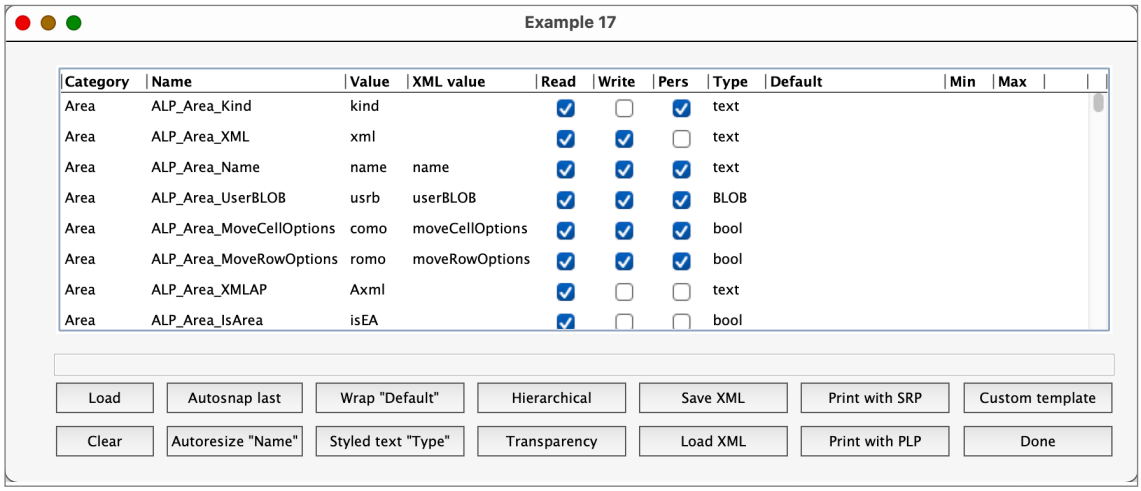
Column 7 - Longint 3 states with library:

```
AL_SetColumnTextProperty(eList; 7; ALP_Column_HeaderText; "Long 3 states\rwith library")  //column header
AL_SetColumnLongProperty(eList; 7; ALP_Column_EntryControl; 1)  //checkBox 3 states
AL_SetColumnTextProperty(eList; 7; ALP_Column_Format; "2116;2117;2118")  //specify column format
AL_SetColumnLongProperty(eList; 7; ALP_Column_DisplayControl; 4)  //library pictures display
AL_SetColumnRealProperty(eList; 7; ALP_Column_Width; 100)
AL_SetColumnLongProperty(eList; 7; ALP_Column_Enterable; 1)
```

Example 17: Importing, Printing, XML

In Example 17 we can see various examples of importing and exporting data to/from an AreaList Pro area, applying formatting, and printing.

When you first open Example 17, the AreaList Pro area is empty. To populate it, click the Load button. This opens an Open File dialog, with the database folder open. Double-click the file named **ALP Properties for 4D.csv** to populate the area with a list of oALP properties and details:



Autosnap last

Calculate the width of the last column to the area edge.

AL_SetAreaLongProperty(eList; ALP_Area_AutoSnapLastColumn; 1)

See the Column Widths section for more details.

Autoresize "Name"

Sets the Autoresize property for the **Name** column (Column 2):

AL_SetAreaLongProperty(eList; ALP_Area_AutoResizeColumn; 2)

Wrap "Default"

Sets the text wrap property for the **Default** column (Column 9)

AL_SetColumnLongProperty(eList; 9; ALP_Column_Wrap; 1) //wrap text in Default column (9)

AL_SetColumnLongProperty(eList; 9; ALP_Column_CalcHeight; 1) //calculate each row height using this column's data

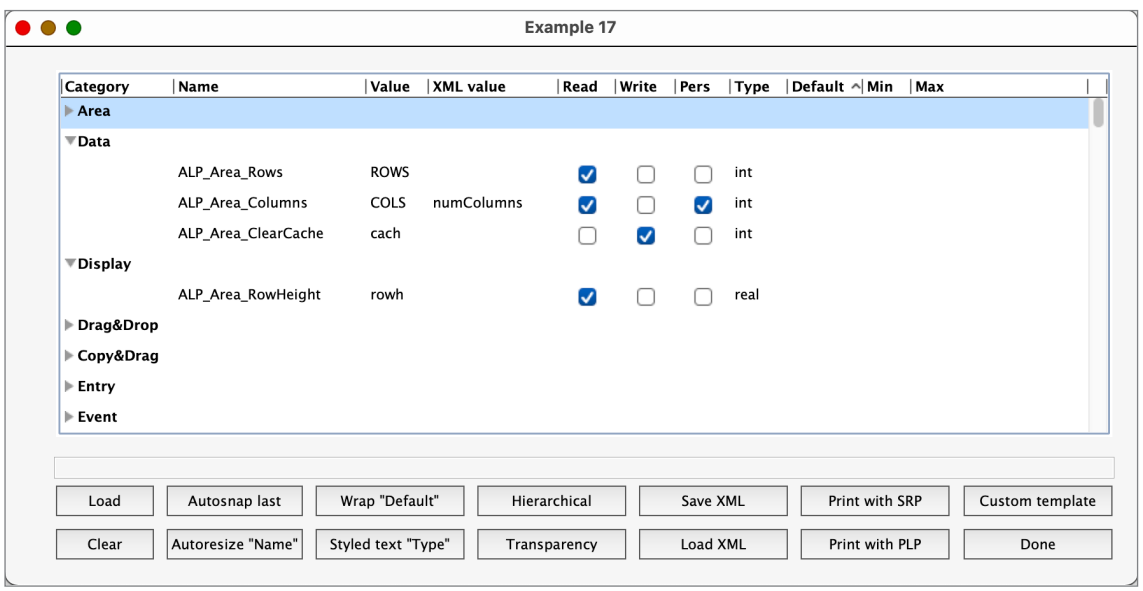
Styled Text "Type"

Allow styled text (rich text) in the Type column:

```
C_LONGINT($i)
AL_SetColumnLongProperty(eList; 8; ALP_Column_Attributed; 1)  //styled text in Type column (8)
//have some fun / amusons-nous un peu
For ($i; 1; Size of array(alp_TpropType_R))
   alp_TpropType_R{$i}:=Replace string(alp_TpropType_R{$i}; "a"; "<s -1><c red>a</c></s>")  //all A's smaller and red
   alp_TpropType_R{$i}:=Replace string(alp_TpropType_R{$i}; "i"; "<i><c green>i</c></i>")  //all I's italic and green
   alp_TpropType_R{$i}:=Replace string(alp_TpropType_R{$i}; "l"; "<s +1><c brown>l</c></s>")  //all L's larger and brown
   alp_TpropType_R{$i}:=Replace string(alp_TpropType_R{$i}; "t"; "<b><c blue>t</c></b>")  //all T's in bold and blue
End for
AL_UpdateArrays(eList; AL Recalculate arrays)  //update area / mise à jour de la zone
```

Hierarchical

Display the data as a hierarchical list:



Following is the button method. However, you could also replace the whole method with one command, using the ALP_Area_AutoHierarchy property, which was introduced in Version 10:

```
AL_SetAreaLongProperty(eList; ALP_Area_AutoHierarchy; 1)
```

This will use break processing to create collapsable break headers.

```
C_TEXT($ToldCat)
C_LONGINT($i; $Lerror)
ARRAY LONGINT($Level_R; Size of array(alp_SpropCat_R))
If (Size of array(alp_SpropCat_R)>0)  //something to display
   $ToldCat:=""
   $i:=1
   Repeat   //we will add lines for level 0
      If (alp_SpropCat_R{$i}#$ToldCat)  //new category
         $ToldCat:=alp_SpropCat_R{$i}
         INSERT IN ARRAY(alp_SpropCat_R; $i; 1)
         alp_SpropCat_R{$i}:=$ToldCat
         INSERT IN ARRAY(alp_TpropName_R; $i; 1)
         INSERT IN ARRAY(alp_TpropValue_R; $i; 1)
         INSERT IN ARRAY(alp_TpropXml_R; $i; 1)
         INSERT IN ARRAY(alp_BpropRead_R; $i; 1)
         INSERT IN ARRAY(alp_BpropWrite_R; $i; 1)
         INSERT IN ARRAY(alp_BpropPersistent_R; $i; 1)
         INSERT IN ARRAY(alp_TpropType_R; $i; 1)
         INSERT IN ARRAY(alp_TpropDefault_R; $i; 1)
         INSERT IN ARRAY(alp_TpropMin_R; $i; 1)
         INSERT IN ARRAY(alp_TpropMax_R; $i; 1)
         INSERT IN ARRAY($Level_R; $i; 1)  //0
      Else   //same category
         alp_SpropCat_R{$i}:=""
         $Level_R{$i}:=1
      End if
      $i:=$i+1
   Until ($i>Size of array(alp_SpropCat_R))
   AL_UpdateArrays(eList; AL Recalculate arrays)  //inform AreaList Pro that the arrays have changed

   //some formatting
   For ($i; 1; Size of array(alp_SpropCat_R))
      If ($Level_R{$i}=0)  //root level
         AL_SetRowLongProperty(eList; $i; ALP_Row_StyleB; 1)  //root level rows in bold
         AL_SetCellLongProperty(eList; $i; 5; ALP_Cell_Invisible; 1)  //we don't want the checkboxes to be displayed at root level
         AL_SetCellLongProperty(eList; $i; 6; ALP_Cell_Invisible; 1)
         AL_SetCellLongProperty(eList; $i; 7; ALP_Cell_Invisible; 1)
      End if
   End for
   ARRAY LONGINT($Lexpanded_R; Size of array(alp_SpropCat_R))  // 0 = not expanded
   AL_SetAreaRealProperty(eList; ALP_Area_HierIndent; 20)  //set hierarchical indentation
   $Lerror:=AL_SetObjects2(eList; ALP_Object_Hierarchy; $Level_R; $Lexpanded_R)
End if
```

Transparency

Apply the Transparency attribute to the Category column:

Category	Name	Value	XML value
Area	ALP_Area_Kind	kind	
Area	ALP_Area_XML	xml	
Area	ALP_Area_Name	name	name
Area	ALP_Area_UserBLOB	usrb	userBLOB
Area	ALP_Area_MoveCellOptions	como	moveCellOptions
Area	ALP_Area_MoveRowOptions	romo	moveRowOptions
Area	ALP_Area_XMLAP	Axml	
Area	ALP_Area_IsArea	isEA	
Area	ALP_Area_Visible	visi	

```
AL_SetColumnLongProperty(eList; 1; ALP_Column_TextColor; 0x77FF0000)
```

Save XML

Save the contents and settings of the AreaList Pro area as an XML file:

```
C_LONGINT($Lerror)
C_TEXT($Tsettings)
C_TIME($HrefDoc)
$Lerror:=AL_Save(eList; $Tsettings)
$HrefDoc:=Create document(""; "TEXT")
If (Ok=1)
   SEND PACKET($HrefDoc; $Tsettings)
   CLOSE DOCUMENT($HrefDoc)
End if
```

Load XML

Load a previously created XML settings file:

```
C_LONGINT($Lerror)
C_TEXT($Tsettings)
C_TIME($HrefDoc)
$HrefDoc:=Open document(""; "*"; Read mode)
If (OK=1)
   RECEIVE PACKET($HrefDoc; $Tsettings; MAXTEXTLENBEFOREV11)  //to the end / jusqu'à la fin
   CLOSE DOCUMENT($HrefDoc)
   $Lerror:=AL_Load(eList; $Tsettings)
End if
```

See the section on XML for more details about saving and loading XML.

Print with SRP

Print the AreaList Pro area as a Preview using the SuperReport Pro plugin (requires a SuperReport Pro license):

ALP Properties

Category	Name	Value	XML value	Read	Write	Pers	Type
Area	ALP_Area_Kind	kind		True	False	True	text
Area	ALP_Area_XML	xml		True	True	False	text
Area	ALP_Area_Name	name	name	True	True	True	text
Area	ALP_Area_UserBLOB	usrb	userBLOB	True	True	True	BLOB
Area	ALP_Area_MoveCellOptions	como	moveCellOptions	True	True	True	bool
Area	ALP_Area_MoveRowOptions	romo	moveRowOptions	True	True	True	bool
Area	ALP_Area_XMLAP	Axml		True	False	False	text
Area	ALP_Area_IsArea	isEA		True	False	False	bool
Area	ALP_Area_Visible	visi		True	True	False	bool
Area	ALP_Area_Selected	sele		True	False	False	bool
Area	ALP_Area_ScrollLeft	scrl		True	True	False	real
Area	ALP_Area_ScrollTop	scrt		True	True	False	real
Area	ALP_Area_Compatibility	comp	compatibility	True	True	True	bool
Area	ALP_Area_CompHideCols	cohc	hideColumns	True	True	True	int
Area	ALP_Area_AutoSnapLastColu	snap	autoSnapLastCol	True	True	True	bool
Area	ALP_Area_ListWidth	Alwd		True	False	False	real
Area	ALP_Area_ListHeight	Alhi		True	False	False	real
Area	ALP_Area_DataWidth	Adwd		True	False	False	real
Area	ALP_Area_DataHeight	Adhi		True	False	False	real
Area	ALP_Area_Self	self		True	False	False	pointer
Area	ALP_Area_ScrollColumns	scco	scrollColumns	True	True	True	bool
Area	ALP_Area_Redraw	upds		False	True	False	n/a

```
C_TEXT($reportXml)
C_LONGINT($result)
vTitle:=Request("Report title"; ""; "OK"; "No title")
If (OK#1)
    vTitle:=""
End if
$reportXml:=AL_SuperReport(eList; ""; 0; 0; vTitle)
If ($reportXml#"")
    $result:=SR_Print($reportXml; 0; SRP_Print_DestinationPreview | SRP_Print_AskPageSetup; ""; 0; ""; 0)
End if
```

Print with PrintList Pro

Print the AreaList Pro area using the PrintList Pro plugin (now included with AreaList Pro).

PrintList Pro: ALP area printing

Category	Name	Value	XML value	Read	Write	Pers	Type	Default	Min	Max
Area	ALP_Area_Kind	kind		True	False	True	text			
Area	ALP_Area_XML	xml		True	True	False	text			
Area	ALP_Area_Name	name	name	True	True	True	text			
Area	ALP_Area_UserBLOB	usrb	userBLOB	True	True	True	BLOB			
Area	ALP_Area_MoveCellOptions	como	moveCellOptions	True	True	True	bool			
Area	ALP_Area_MoveRowOptions	romo	moveRowOptions	True	True	True	bool			
Area	ALP_Area_XMLAP	Axml		True	False	False	text			
Area	ALP_Area_IsArea	isEA		True	False	False	bool			
Area	ALP_Area_Visible	visi		True	True	False	bool			
Area	ALP_Area_Selected	sele		True	False	False	bool			
Area	ALP_Area_ScrollLeft	scrl		True	True	False	real			
Area	ALP_Area_ScrollTop	scrt		True	True	False	real			
Area	ALP_Area_Compatibility	comp	compatibility	True	True	True	bool	depends on initialization		
Area	ALP_Area_CompHideCols	cohc	hideColumns	True	True	True	int	0	0	
Area	ALP_Area_AutoSnapLastColumn	snap	autoSnapLastCol	True	True	True	bool	no		
Area	ALP_Area_ListWidth	Alwd		True	False	False	real			
Area	ALP_Area_ListHeight	Alhi		True	False	False	real			
Area	ALP_Area_DataWidth	Adwd		True	False	False	real			
Area	ALP_Area_DataHeight	Adhi		True	False	False	real			
Area	ALP_Area_Self	self		True	False	False	pointer			
Area	ALP_Area_ScrollColumns	scco	scrollColumns	True	True	True	bool			
Area	ALP_Area_Redraw	upds		False	True	False	n/a			
Area	ALP_Area_WindowsText	DIEN		True	True	False	bool	no		
Area	ALP_Area_WindowsClip	CLIP		True	True	False	bool	no		
Area	ALP_Area_ArrowsForHierarchy	arkh		True	True	False	bool	no		
Area	ALP_Area_AutoResizeColumn	SNAP	autoResizeColumn	True	True	True	int	0	-1	num. columns
Data	ALP_Area_TableID	tbid	mainTable	True	True	True	int	0		
Data	ALP_Area_Rows	ROWS		True	False	False	int			
Data	ALP_Area_Columns	COLS	numColumns	True	False	True	int			
Data	ALP_Area_CacheSize	cacs	cacheSize	True	True	True	int	1024	128	
Data	ALP_Area_ClearCache	cach		False	True	False	int			
Data	ALP_Area_FillCache	data		False	True	False	int	0	0	

```
If (Records in table([Layouts])=0)
    CREATE RECORD([Layouts])
    [Layouts]Field1:="PLP record"
    SAVE RECORD([Layouts])
End if
ALL RECORDS([Layouts])
FORM SET OUTPUT([Layouts]; "PL_List")
PRINT SETTINGS
If (OK=1)
    SET PRINT PREVIEW(True)
    PRINT SELECTION([Layouts]; >)
End if
```

For more information about PrintList Pro, see Chapter 17.

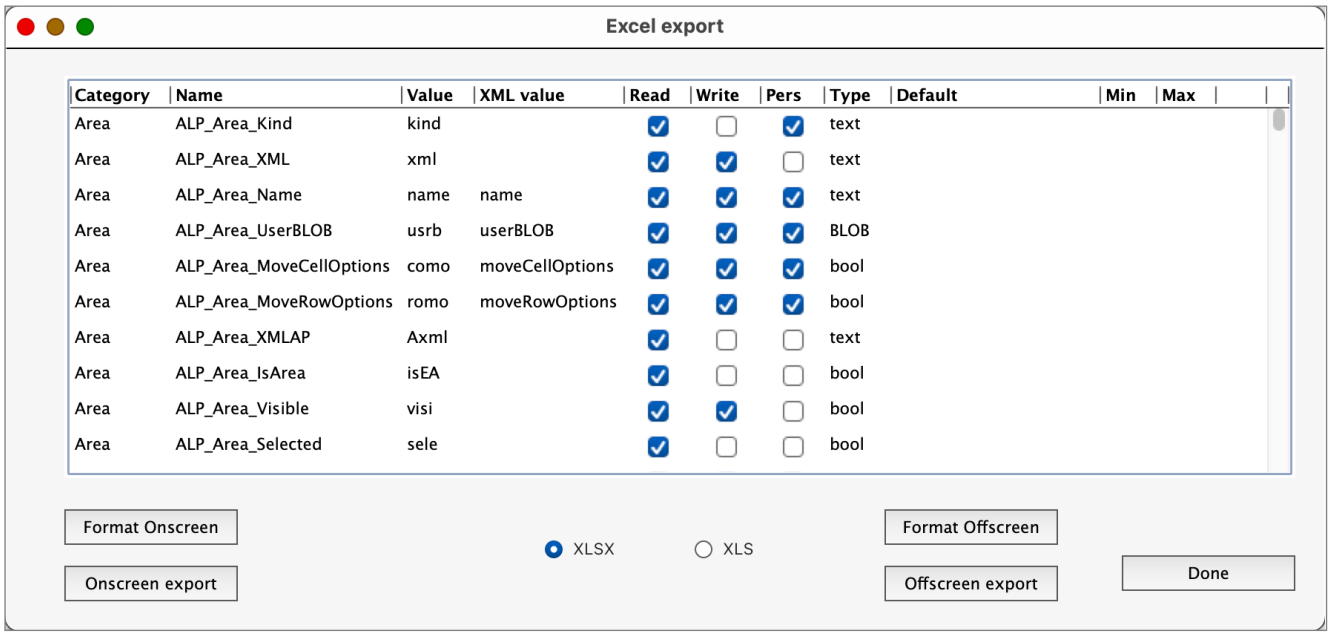
6 Tutorial: v11 New Features

The following examples illustrate how to use the new features in AreaList Pro v11.

You can find them all in the Examples database.

Export to Excel

This example demonstrates how to create an Excel spreadsheet (XLS or XLSX) from your AreaList Pro area:



Category	Name	Value	XML value	Read	Write	Pers	Type	Default	Min	Max
Area	ALP_Area_Kind	kind		☑	☐	☑	text			
Area	ALP_Area_XML	xml		☑	☑	☐	text			
Area	ALP_Area_Name	name	name	☑	☑	☑	text			
Area	ALP_Area_UserBLOB	usrb	userBLOB	☑	☑	☑	BLOB			
Area	ALP_Area_MoveCellOptions	como	moveCellOptions	☑	☑	☑	bool			
Area	ALP_Area_MoveRowOptions	romo	moveRowOptions	☑	☑	☑	bool			
Area	ALP_Area_XMLAP	Axml		☑	☐	☐	text			
Area	ALP_Area_IsArea	isEA		☑	☐	☐	bool			
Area	ALP_Area_Visible	visi		☑	☑	☐	bool			
Area	ALP_Area_Selected	sele		☑	☐	☐	bool			

Format Onscreen

Apply formatting to the data and display it onscreen:

Category	Name	Value	XML value	Read	Write	Pers	Type	Default
Area	ALP_Area_Kind	kind		☑	☐	☑	text	
Area	ALP_Area_XML	xml		☑	☑	☐	text	
Area	ALP_Area_Name	name	name	☑	☑	☑	text	
Area	ALP_Area_UserBLOB	usrb	userBLOB	☑	☑	☑	BLOB	
Area	ALP_Area_MoveCellOptions	como	moveCellOptions	☑	☑	☑	bool	
Area	ALP_Area_MoveRowOptions	romo	moveRowOptions	☑	☑	☑	bool	
Area	ALP_Area_XMLAF	Axml		☑	☐	☐	text	
Area	ALP_Area_IsArea	isEA		☑	☐	☐	bool	
Area	ALP_Area_Visible	visi		☑	☑	☐	bool	
Area	ALP_Area_Selected	sele		☑	☐	☐	bool	
Area	ALP_Area_ScrollLeft	scrl		☑	☑	☐	real	
Area	ALP_Area_ScrollTop	scrt		☑	☑	☐	real	

```
C_LONGINT($error)
If (onscreenFormatted)
    AL_RemoveColumn(eList; -2)  //remove all columns
    $error:=AL_Load(eList; aL_TareaDefXml)  //reload initial definition state
    AL_UpdateArrays(eList; AL Recalculate arrays)
    onscreenFormatted:=False
Else
    SetupAreaToExport(eList)
    onscreenFormatted:=True
End if
```

The ***SetupAreaToExport*** method applies various formatting:

```
C_LONGINT($1)
// no space for sort indicator in the header
AL_SetAreaLongProperty($1; ALP_Area_ShowSortIndicator; 0)

// Excel uses 1 point indent
AL_SetAreaTextProperty($1; ALP_Area_HdrIndent; "1;1")
AL_SetAreaTextProperty($1; ALP_Area_RowIndent; "1;1")
AL_SetAreaTextProperty($1; ALP_Area_FtrIndent; "1;1")
// name for Sheet1
AL_SetAreaTextProperty($1; ALP_Area_Name; "Properties")
// Basic formatting
AL_SetAreaLongProperty($1; ALP_Area_AltRowOptions; 1)  // alternate row background (“zebra” style)
AL_SetAreaLongProperty($1; ALP_Area_AltRowColor; 0x00DFF6FF | 0x80000000)
```

```
AL_SetColumnLongProperty($1; 0; ALP_Column_HdrHorAlign; AL Just center)  //centerered
AL_SetColumnLongProperty($1; 0; ALP_Column_HdrStyleB; 1)  //bold
AL_SetColumnTextProperty($1; 0; ALP_Column_HdrFontName; "Calibri")
AL_SetColumnLongProperty($1; 0; ALP_Column_HdrSize; 11)
AL_SetColumnTextProperty($1; 0; ALP_Column_FontName; "Calibri")
AL_SetColumnLongProperty($1; 0; ALP_Column_Size; 11)

AL_SetColumnTextProperty($1; 1; ALP_Column_HdrTextColor; "Purple")  //
AL_SetColumnRealProperty($1; 1; ALP_Column_HdrSize; 14)  //font size
AL_SetColumnLongProperty($1; 1; ALP_Column_HdrStyleB; 1)
AL_SetColumnLongProperty($1; 1; ALP_Column_HdrHorAlign; 2)  //centered
AL_SetColumnLongProperty($1; 1; ALP_Column_HdrVertAlign; 2)  //centered

AL_SetRowTextProperty($1; 3; ALP_Row_TextColor; "Red"; 1)
AL_SetRowTextProperty($1; 7; ALP_Row_TextColor; "Green"; 1)
AL_SetRowTextProperty($1; 7; ALP_Row_BackColor; "Brown")

AL_SetRowTextProperty($1; 13; ALP_Row_TextColor; "Yellow"; 1)
AL_SetRowRealProperty($1; 13; ALP_Row_Size; 18)
AL_SetRowTextProperty($1; 13; ALP_Row_BackColor; "Orange")

AL_SetRowTextProperty($1; 15; ALP_Row_TextColor; "Blue"; 1)
AL_SetRowLongProperty($1; 15; ALP_Row_StyleB; 1)
AL_SetRowLongProperty($1; 15; ALP_Row_StyleI; 1)

AL_SetCellTextProperty($1; 20; 2; ALP_Cell_BackColor; "Red")
AL_SetCellRealProperty($1; 20; 2; ALP_Cell_Size; 14)
AL_SetCellTextProperty($1; 20; 2; ALP_Cell_TextColor; "White")
AL_SetCellLongProperty($1; 20; 2; ALP_Cell_HorAlign; 2)  //centered

AL_SetCellTextProperty($1; 28; 8; ALP_Cell_BackColor; "Black")
AL_SetCellRealProperty($1; 28; 8; ALP_Cell_Size; 14)
AL_SetCellTextProperty($1; 28; 8; ALP_Cell_TextColor; "Yellow")
AL_SetCellLongProperty($1; 28; 8; ALP_Cell_HorAlign; 3)  //right
AL_SetCellRealProperty($1; 28; 8; ALP_Cell_Rotation; 90)  //rotation
AL_SetCellLongProperty($1; 28; 8; ALP_Cell_StyleB; 1)  //bold
AL_UpdateArrays($1; AL Recalculate arrays)  //update area
```

Onscreen export

Create an Excel file, formatting it as currently shown on-screen:

```
ExportArea(eList; bSelectedRows)
```

The ***ExportArea*** method:

```
C_LONGINT($1)  //area
C_LONGINT($2)  //selected only if 1
C_TEXT($Tpath)
$Tpath:=Select folder("Select a destination folder"; System folder(Desktop))
If (Test path name($Tpath)=Is a folder)
   If (bExportTypeXlsx=1)  //XSLX
      AL_SetAreaLongProperty($1; ALP_Area_ExportOptions; AL Export XLSX+($2*512))  // XLSX, selected only if $2=1
      AL_SetAreaTextProperty($1; ALP_Area_ExportToFile; $Tpath+"ALP Properties.xlsx")  //  export
      SHOW ON DISK($Tpath+"ALP Properties.xlsx")
   Else   //XLS
      AL_SetAreaLongProperty($1; ALP_Area_ExportOptions; AL Export binary XLS+($2*512))  // XLS, selected only if $2=1
      AL_SetAreaTextProperty($1; ALP_Area_ExportToFile; $Tpath+"ALP Properties.xls")  //  export
      SHOW ON DISK($Tpath+"ALP Properties.xls")
   End if
End if
```

Format offscreen

Create an offscreen area and apply the formatting to it:

```
C_LONGINT($error)
If (offscreenFormatted)
   AL_RemoveColumn(alp_Loffscreen; -2)  //remove all columns
   $error:=AL_Load(alp_Loffscreen; aL_TareaDefXml)  //reload initial definition state
   AL_UpdateArrays(alp_Loffscreen; AL Recalculate arrays)
   offscreenFormatted:=False
Else
   SetupAreaToExport(alp_Loffscreen)
   offscreenFormatted:=True
End if
```

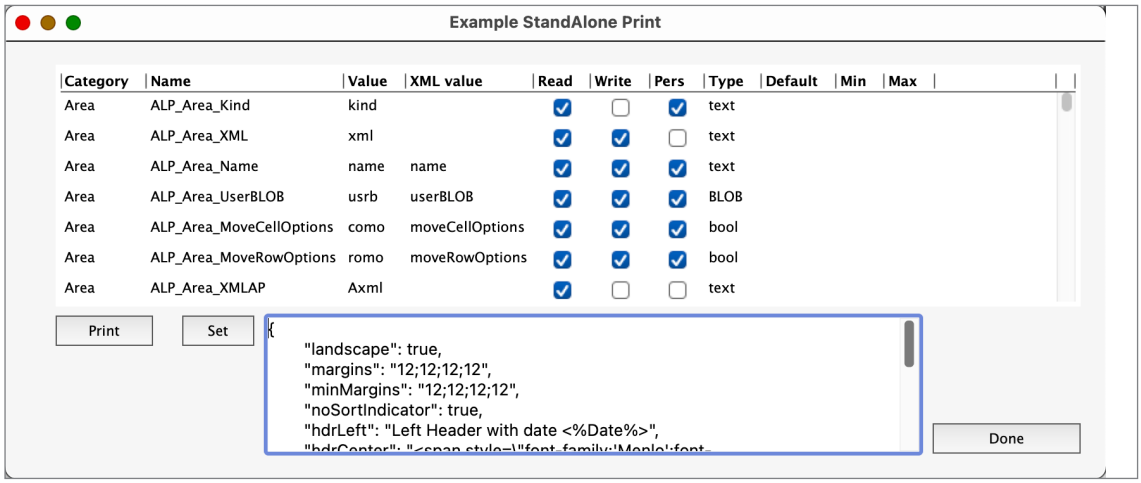
Offscreen export

Create an Excel file from the offscreen area.

```
ExportArea(alp_Loffscreen; 0)  //offscreen area export (whole area)
```

Standalone Printing

Print the AreaList Pro area using the built-in print function:



In the On Load event, the AreaList Pro area is set up by importing and formatting the **ALP Properties for 4D.csv** file:

```
C_LONGINT($Lerror)
Case of
  : (Form event code=On Load)  //initialize the AreaList Pro object
    aLInitArea(eList)
    ALparseDocument(Get 4D folder(Database folder)+"ALP properties for 4D.csv")
    AL_SetAreaLongProperty(eList; ALP_Area_MinRowHeight; 0)
    AL_SetAreaLongProperty(eList; ALP_Area_NumRowLines; 0)  //use variable row height
    AL_SetColumnLongProperty(eList; 5; ALP_Column_CalcHeight; 1)  //calculate each row height using this column's data
    $Lerror:=AL_GetAreaPtrProperty(eList; ALP_Area_PrintOptions; ->aLp_OCurrentValue)
    // you can assume $err is 0 when $area is valid, because you ask for an object and provide a pointer to an object
    If (aLp_OCurrentValue=Null)  //When ALP_Area_PrintOptions is not defined, create a new object and set the object to ALP
      aLp_OCurrentValue:=New object
      $Lerror:=AL_SetAreaPtrProperty(eList; ALP_Area_PrintOptions; ->aLp_OCurrentValue)
    End if
    SetDefaultPrintOptions
    $Lerror:=AL_GetAreaPtrProperty(eList; ALP_Area_PrintOptions; ->aLp_OCurrentValue)
```

Then stringify the settings:

```
    aLp_TCurrentValue:=JSON Stringify(aLp_OCurrentValue; *)
End case
```

The stringified settings (aLp_TCurrentValue) are shown in the form:

```
{
    "landscape": true,
    "margins": "12;12;12;12",
    "minMargins": "12;12;12;12",
    "noSortIndicator": true,
    "hdrLeft": "Left Header with date <%Date%>",
    "hdrCenter": "<span style=\"font-family:'Menlo';font-
```

Print

Print the area using the displayed settings:

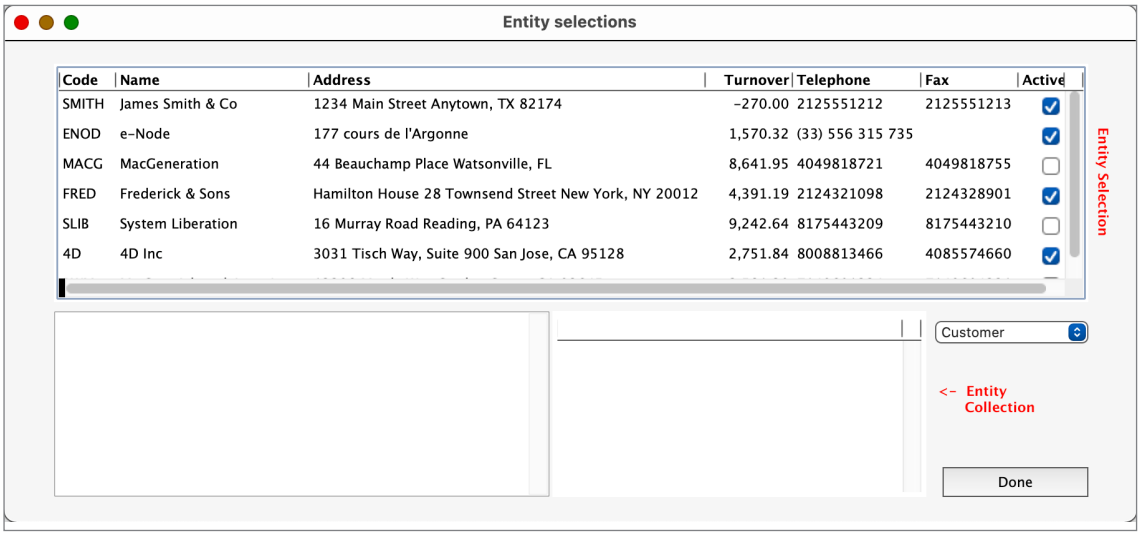
```
C_LONGINT($err)
PRINT SETTINGS(Page setup dialog+Print dialog)
If (OK=1)
    $err:=AL_GetAreaLongProperty(eList; ALP_Area_PrintArea)
    If ($err#0)
        ALERT("Printing error: "+String($err))
    End if
End if
```

We can enhance this through the use of additional properties:

```
C_LONGINT($err)
PRINT SETTINGS(Page setup dialog+Print dialog)
If (OK=1)
    AL_SetAreaLongProperty (eList;PLP_PrintHeaders;2)  //all pages
    AL_SetAreaRealProperty (eList;PLP_FrameWidth;0.25)  //frame width
    AL_SetAreaLongProperty (eList;ALP_Area_DrawFrame;1)  //draw frame
    $err:=AL_GetAreaLongProperty(eList; ALP_Area_PrintArea)
    If ($err#0)
        ALERT("Printing error: "+String($err))
    End if
End if
```

Entity Selections

This example demonstrates how to use ORDA entities with AreaList Pro:



In the On Load event of the area object method:

First we declare the variables we need:

```
C_LONGINT($i; $Lerror; $Lrow)
C_TEXT($Titem; $TcolName)
C_OBJECT($Oentity)
```

Then we retrieve the structure definition for the [Customer] table:

```
al_OcurrentTableDataClass:=ds.Customer  //table struct description from the DataStore
```

From the description we build a new object from the DataClass interface to this table (DataClass available as a ds DataStore property) but setting dataClassObject (or dataClassName) is not necessary, AreaList Pro will retrieve it when you set the entity selection object with ***AL_SetAreaPtrProperty(***eList***;*** ALP_Area_DataClassObject***; ->***al_OcurrentTableDataClass***)***.

Then we tell AreaList Pro that the display source is an Entity selection, but setting ALP_Area_DataSource is not necessary, AreaList Pro will set it to 4 when you set the entity selection object with ***AL_SetAreaLongProperty***(eList; ALP_Area_DataSource; 4).

Then we populate this new Entity selection object with the table contents (like **ALL RECORDS**):

```
al_OentitySelection:=al_OcurrentTableDataClass.all()  //select all the Customer's records
```

We feed AreaList Pro with the Entity selection that will be cached:

```
$Lerror:=AL_SetAreaPtrProperty(eList; ALP_Area_EntitySelection; ->al_OentitySelection)
$i:=1  //1st column number
```

Now we loop on each Entity selection item to tell AreaList Pro which columns it must display (all fields in this example)

```
For each ($TcolName; al_OcurrentTableDataClass)  //for each Customer dataclass element
```

Relations are also stored in the DataClass, but we only need fields:

```
If (al_OcurrentTableDataClass[$TcolName].kind="storage")  //it's a field
```

Then for each field we create the matching column, either indicating the field type...

```
$Lerror:=AL_AddEntityColumn(eList; $TcolName; al_OcurrentTableDataClass[$TcolName].fieldType)
//add the entity with the field name and its type
```

... or AreaList Pro will simply find out by itself from the first row data type

```
$Lerror:=AL_AddEntityColumn(eList; $TcolName; Is undefined)  //add the entity with the field name (AreaList Pro fetches field type from ORDA)
```

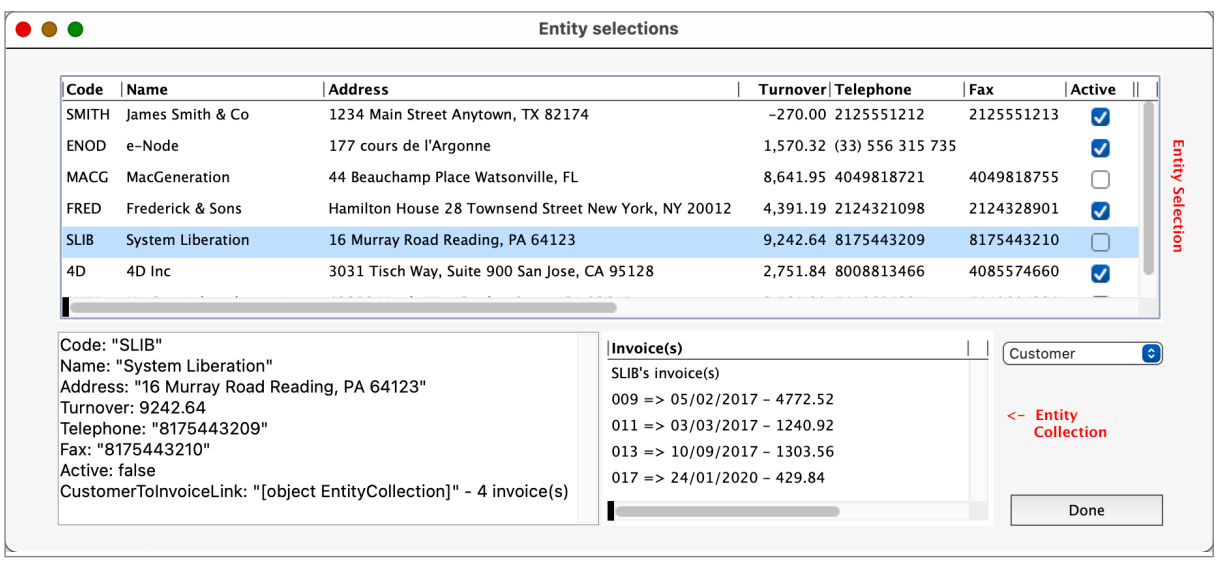
Lastly, we assign a header text for this column using the "name" attribute, which will be automatically set to $TcolName

```
        AL_SetColumnTextProperty(eList; $i; ALP_Column_HeaderText; $TcolName)  //column header title
        $i:=$i+1  //next column number
    End if
End for each
```

Each Entity contains a link to the customer's Invoices:

Expression	Value
al_OentitySelection	Selection of 7 Entities from Customer
__LockerID	Undefined
00	Entity(SMITH) from Customer
Active	True
Address	"1234 Main Street Anytown, TX 82174"
Code	"SMITH"
CustomerToInvoiceLink	Selection of 1 Entity from Invoice
Fax	"2125551213"
Name	"James Smith & Co"
Telephone	"2125551212"
Turnover	-270

To view the details and the Invoices Collection pertaining to one Customer, click on the customer's name in the list:



This is managed via the On Plugin Area event:

```
$Levent:=AL_GetAreaLongProperty(eList; ALP_Area_AlpEvent)
If ($Levent=AL Single click event)  //single-click on a row (or up / down arrow keys)
```

Initialise the Collection and other variables:

```
al_OcurrentRecsCollection:=New collection
$Tseparator:="-"*30
aLp_TCurrentValue:=""
```

Identify the selected row:

```
$Lrow:=AL_GetAreaLongProperty(eList; ALP_Area_ClickedRow)-1  //get the row the user clicked (AreaList Pro rows from 1, whereas object attributes begin at position 0)
If ($Lrow>=0)  //valid object position
```

We make a new object containing the selected row values through the Entity selection object:

```
$Oentity:=al_OentitySelection[$Lrow]  //new "element" object
```

Then we loop on each attribute to this object looking for "[object EntityCollection]" (with quotes):

```
For each ($Titem; $Oentity)  //for each item of this object element
```

Using "JSON Stringify" because the attribute is not always typed as text:

```
$Text:=JSON Stringify($Oentity[$Titem])
Case of
: ($Text="@[object EntityCollection]@")  //entity collection (N to 1)
aLp_TCurrentValue:=aLp_TCurrentValue+$Titem+": "+$Text
Case of
: (aL_Tables_R=1)  //customers
aLp_TCurrentValue:=aLp_TCurrentValue+" - "+String($Oentity[$Titem].length)+" invoice(s)"
If ($Oentity[$Titem].length>0)
al_OcurrentRecsCollection.push(New object("Related"; $Oentity[$Titem][0].CustCode+"'s invoice(s)"))
End if
: (aL_Tables_R=2)  //invoices
aLp_TCurrentValue:=aLp_TCurrentValue+" - "+String($Oentity[$Titem].length)+" item(s)"
If ($Oentity[$Titem].length>0)
al_OcurrentRecsCollection.push(New object("Related"; $Oentity[$Titem][0].itemToInvoiceLink.CustCode+"'s invoice item(s)"))
End if
End case
For ($i; 0; $Oentity[$Titem].length-1)
Case of
: (aL_Tables_R=1)  //customers
```

If we wanted to display the contents of the linked table in a collection, we would use al_OcurrentRecsCollection:=ds[Table name(->[Invoice])].all().toCollection() but here, only 3 fields in one column

```
al_OcurrentRecsCollection.push(New object("Related"; String($Oentity[$Titem][$i].Number; "000")+" => "+String($Oentity[$Titem][$i].CreateDate)+" - "+String($Oentity[$Titem][$i].Total)))
: (aL_Tables_R=2)  //invoices
```

If we wanted to display the contents of the linked table in a collection, we would use "al_OcurrentRecsCollection:=ds[Table name(->[Item])].all().toCollection()" but here, only 4 fields in one column

```
                al_OcurrentRecsCollection.push(New object("Related"; $Oentity[$Titem][$i].Description+" =>
"+String($Oentity[$Titem][$i].UnitPrice)+" * "+String($Oentity[$Titem][$i].Qty)+" = "+String($Oentity[$Titem][$i].LineTotal)))
            End case
        End for
        aLp_TCurrentValue:=aLp_TCurrentValue+"\r"
        If (aL_Tables_R<3)
```

Setting ALP_Area_DataSource is not necessary, AreaList Pro will set it to 3 when you set the entity selection object.

```
            AL_SetAreaPtrProperty(eListCol; ALP_Area_Collection; ->al_OcurrentRecsCollection)  //collection to be displayed
            If (AL_GetAreaLongProperty(eListCol; ALP_Area_Columns)<1)
                $Lerror:=AL_AddEntityColumn(eListCol; "Related"; Is text)
                AL_SetColumnRealProperty(eListCol; 1; ALP_Column_Width; 435)
                AL_SetColumnLongProperty(eListCol; 1; ALP_Column_Sortable; 0)  // disallow sort
                AL_SetAreaLongProperty(eListCol; ALP_Area_SelMultiple; 0)  //set multi-rows mode
                AL_SetAreaLongProperty(eListCol; ALP_Area_SelNone; 1)  //allow no selection
                AL_SetAreaLongProperty(eListCol; ALP_Area_AllowSortEditor; 0)  // disallow sort editor
                DEMO_Default(eListCol)  //general display settings
            Else
                AL_SetAreaLongProperty(eListCol; ALP_Area_UpdateData; 0)  //update destination area
            End if
            AL_SetColumnTextProperty(eListCol; 1; ALP_Column_HeaderText; Choose(aL_Tables_R=1; "Invoice(s)";
"Item(s)"))  //column title
        End if
    : ($Text="@[object Entity]@")  //object entity (1 to N)
    Else
        aLp_TCurrentValue:=aLp_TCurrentValue+$Titem+": "+$Text+"\r"
  End case
End for each
```

You can find additional details in the method ***_Docs_Entity_Selection*** from the Examples database.

For more information about ORDA, see the 4D online documentation.

Sub-ALP

It's all set up in the On Load event:

```
$i:=1  //1st column number
vRow:=Num(PrefsGet("vRow"; "1"))  //main area row dividers
vCol:=Num(PrefsGet("vCol"; "1"))  //main area column dividers
vRowSub:=Num(PrefsGet("vRowSub"; "0"))  //no sub area row dividers
vColSub:=Num(PrefsGet("vColSub"; "0"))  //no sub area column dividers
vRowAlternate:=Num(PrefsGet("vRowAlternate"; "0"))  //no alternate color for main area rows
vRowBack:=Num(PrefsGet("vRowBack"; "0"))  //no row back color
vRowAlternateSub:=Num(PrefsGet("vRowAlternateSub"; "0"))  //no alternate color for sub area rows
vRowBackSub:=Num(PrefsGet("vRowBackSub"; "0"))  //no row back color for sub area rows
vZoom:=Num(PrefsGet("vZoom"; "100"))  //main area default zoom
vZoomInvoices:=Num(PrefsGet("vZoomInvoices"; "100"))  //sub area default zoom
al_OcurrentTableDataClass:=ds.Customer
```

Setting dataClassObject (or dataClassName) is not necessary, AreaList Pro will retrieve it when you set the entity selection object.

Setting ALP_Area_DataSource is not necessary, AreaList Pro will set it to 4 when you set the entity selection object.

```
al_OentitySelection:=al_OcurrentTableDataClass.all()
$Lerror:=AL_SetAreaPtrProperty(Self->; ALP_Area_EntitySelection; ->al_OentitySelection)
AL_SetAreaLongProperty(Self->; ALP_Area_ShowRowDividers; 3*vRow)
AL_SetAreaLongProperty(Self->; ALP_Area_ShowColDividers; 9*vCol)
For each ($TcolName; al_OcurrentTableDataClass)
   $Lerror:=AL_AddEntityColumn(Self->; $TcolName; Is undefined)  //add the entity with the field name (AreaList Pro fetches field type from ORDA)
   Case of
      : ($Lerror#0)
         TRACE
      : (al_OcurrentTableDataClass[$TcolName].kind="storage")  //true field
         AL_SetColumnTextProperty(Self->; $i; ALP_Column_HeaderText; $TcolName)
         AL_SetColumnLongProperty(Self->; $i; ALP_Column_HdrHorAlign; 2)  //centered header

      : (al_OcurrentTableDataClass[$TcolName].kind="relatedEntities")
            //fieldType : 42
            //name : CustomerToInvoiceLink
            //inverseName : InvoiceToCustomerLink
            //relatedDataClass : Invoice
            //type : InvoiceSelection
         AL_SetColumnRealProperty(Self->; $i; ALP_Column_WidthUser; 0)
```

This is not needed, sub-ALP provides the headers, this one will be never used

```
//AL_SetColumnTextProperty(Self->; $i; ALP_Column_HeaderText; "Related entities")
eSubAlp:=AL_GetColumnLongProperty(Self->; $i; ALP_Column_SubALP)  //get handle to the sub-ALP
AL_SetAreaLongProperty(Self->; ALP_Area_NumHdrLines; 0)  // allow different sub-ALP header height
AL_SetAreaLongProperty(eSubAlp; ALP_Area_NumRowLines; 1)
```

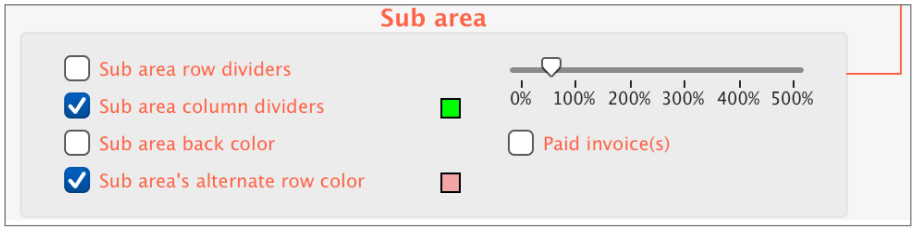
This is not needed, sub-ALP draws in a cell, this one will be never used - no frame in a cell, only in area

```
//AL_SetAreaLongProperty(eSubAlp; ALP_Area_DrawFrame; 1)
$Lerror:=AL_AddEntityColumn(eSubAlp; "This.Number"; Is undefined)  //data type will be fetched from the dataClass
$Lerror:=AL_AddEntityColumn(eSubAlp; "This.CreateDate"; Is undefined)
$Lerror:=AL_AddEntityColumn(eSubAlp; "This.Total"; Is undefined)
$Lerror:=AL_AddEntityColumn(eSubAlp; "This.Paid"; Is undefined)
AL_SetColumnTextProperty(eSubAlp; 1; ALP_Column_HeaderText; "Number")
AL_SetColumnTextProperty(eSubAlp; 2; ALP_Column_HeaderText; "Date")
AL_SetColumnTextProperty(eSubAlp; 3; ALP_Column_HeaderText; "Total")
AL_SetColumnTextProperty(eSubAlp; 4; ALP_Column_HeaderText; "Paid")
AL_SetColumnTextProperty(eSubAlp; 3; ALP_Column_Format; <>str10LanguageFormat_V)
AL_SetColumnLongProperty(eSubAlp; 4; ALP_Column_DisplayControl; 1)
  //0 = normal, 1 = 83% (size is divided by 1.2), 2 = 67% (size is divided by 1.5)
AL_SetColumnTextProperty(eSubAlp; 1; ALP_Column_GroupHdrText; "Related invoice(s)")
AL_SetCellLongProperty(eSubAlp; -3; 1; ALP_Cell_HorAlign; 2)
AL_SetColumnLongProperty(eSubAlp; -2; ALP_Column_Enterable; 0)
AL_SetColumnLongProperty(eSubAlp; -2; ALP_Column_GroupID; 1)
AL_SetAreaLongProperty(eSubAlp; ALP_Area_GroupHeader; 1)
AL_SetColumnLongProperty(eSubAlp; -2; ALP_Column_HdrHorAlign; 2)  //centered headers
AL_SetColumnLongProperty(eSubAlp; 1; ALP_Row_HorAlign; 2)  //centered invoice numbers
AL_SetColumnLongProperty(eSubAlp; 2; ALP_Row_HorAlign; 2)  //centered dates
AL_SetAreaLongProperty(eSubAlp; ALP_Area_ShowRowDividers; vRowSub)  //3
AL_SetAreaLongProperty(eSubAlp; ALP_Area_ShowColDividers; vColSub)  //9
AL_SetAreaLongProperty(eSubAlp; ALP_Area_AltRowOptions; vRowAlternateSub)
AL_SetAreaLongProperty(eSubAlp; ALP_Area_ResizeDuring; 1)  //to use variable column width, otherwise it does not work
DEMO_Default(eSubAlp)  //general display settings
End case
$i:=$i+1  //next column number
End for each
```

```
DEMO_Default(Self->)  //general display settings
AL_SetAreaLongProperty(Self->; ALP_Area_SelMultiple; 1)  //set multi-rows mode
AL_SetAreaLongProperty(Self->; ALP_Area_SelNone; 1)  //allow no selection
AL_SetAreaLongProperty(Self->; ALP_Area_EntryClick; 0)  //no way
AL_SetAreaLongProperty(Self->; ALP_Area_MinRowHeight; 0)
AL_SetAreaLongProperty(Self->; ALP_Area_NumRowLines; 0)  //use variable row height
AL_SetAreaLongProperty(Self->; 5; ALP_Column_CalcHeight; 1)  //automatic height
AL_SetAreaLongProperty(Self->; ALP_Area_AltRowOptions; vRowAlternate)
AL_SetAreaLongProperty(Self->; 7; ALP_Column_DisplayControl; 1)  //0 = normal, 1 = 83% (size is divided by 1.2), 2 = 67% (size is divided by 1.5)
AL_SetAreaLongProperty(Self->; AL_GetAreaLongProperty(Self->; ALP_Area_Columns); ALP_Column_CalcHeight; 1)  // calculate each row height using this sub-ALP column data
```

Zoom In

Change the display parameters of the Sub-ALP cells by changing the options in the Sub Area section:



You can find more details about this example in the method ***_Docs_SubALP***.

7 Programming the AreaList Pro User Interface

Managing the General Properties of an area

You can use the properties in the Area General Properties theme to manage many aspects of the appearance and behaviour of an area, such as whether columns will automatically be resized (ALP_Area_AutoResizeColumn), the behaviour of the arrow keys (ALP_Area_ArrowsForHierarchy), and many more options.

There are also many ways to manage cells, columns, rows, headers, and footers within an AreaList Pro area - see the Text Styling topic for info on how to set or get text attributes such as the font, size, style, and so on.

Entering Data

Initiating Data Entry

Data entry using typed characters may be initiated in an AreaList Pro cell by three methods:

- User click, single or double, with or without modifiers, or click and hold.
- Return or Tab keys, with or without shift, from the previous or next (shift) enterable cell in the AreaList Pro area.
- Programmatically.

Once typed data entry is initiated, standard editing functions can be performed on the selected cell, including the Edit menu commands Cut, Copy, Paste, Clear, Select All, and Undo. This is true for cells containing pictures, also (except Select All). However, due to system limitations, only Edit menu shortcuts will work while editing a cell, but not the menu itself (of which lines will be disabled) except for picture columns.

Text data being edited will always appear left-justified, regardless of the column's display justification. The I-Beam pointer can be dragged across the data in the cell to select a portion or all of the data.

In addition, the ALP_Area_CallbackMethMenu property provides the developer with a complete hook to working with the Edit menu.

Two user click modes

User clicks on a cell can be interpreted in two different ways to trigger entry into a cell:

Click or double-click with optional modifiers

Data entry on a given cell could be initiated upon a single click in that cell, a double-click, or a double-click along with the option/alt, ctrl/command, shift, or control key. This setting uses the ALP_Area_EntryClick property. If you do not wish to initiate data entry using this method, set this property to 0:

AL_SetAreaLongProperty ($area; ALP_Area_EntryClick; 0)

Setting the area reference to 0 will set the default value for all newly created areas.

Click-hold

AreaList Pro also provides the ability to initiate data entry by clicking and holding the mouse button down in the cell where you wish to perform data entry.

Using this interface, users are still able to select rows and/or enter cells via single-click and double-click.

If you do not wish to initiate data entry using this method, set the ALP_Area_ClickDelay property to 0:

AL_SetAreaLongProperty ($area; ALP_Area_ClickDelay; 0)

Otherwise, this property will set the delay (in ticks, i.e. 1/60 seconds) to hold the click before editing begins. Defaut is 30 (half a second).

Setting this property to -2 will use the system's double-click time.

Setting the area reference to 0 will set the default value for all newly created areas.

Note: if the user moves the mouse during the click and hold action, AreaList Pro may interpret that as a drag action when AreaList Pro dragging actions are active.

Summary

The table below summarizes all possible values for ALP_Area_EntryClick and the resulting behaviour for both modes (assuming that ALP_Area_ClickDelay is non-zero, otherwise click-hold will never trigger entry):

Value	Click or double-click with optional modifiers	Click-hold
0	no	no
1	single click	N/A (click will instantly trigger editing)
2	double click	yes
3	command-double click	yes
4	shift-double click	yes
5	option-double click	yes
6	control-double-click	yes
7	no	yes

Note: the ALP_Area_EntryFirstClickMode property determines how the first click is handled upon beginning entry (when using numeric, date, time or text entry). See Click action.

Editing 4D fields

Field mode areas allow direct editing of field values. The record will be saved automatically by AreaList Pro upon exiting the cell (if not disallowed by the "entry finished" callback method).

Editing alpha fields (with limited length in 4D field properties) includes a control that the entered value length does not exceed the defined field length. Extra characters won't be accepted. The maximum field length value can be retrieved through the ALP_Column_Length property.

Cell change properties

Entry into an enterable cell can be caused by using the ALP_Area_EntrySkip property from the previous enterable cell.Different properties can be used to move the cursor to a specific cell.

Either

```
AL_SetAreaTextProperty ($area; ALP_Area_EntryGotoCell; String ($row)+","+String($cell))
```

or

```
AL_SetAreaLongProperty ($area; ALP_Area_EntryGotoRow; $row)
AL_SetAreaLongProperty ($area; ALP_Area_EntryGotoColumn; $column)
```

or

```
AL_SetAreaLongProperty ($area; ALP_Area_EntryGotoRow; $row)
AL_SetAreaLongProperty ($area; ALP_Area_EntryGotoGridCell; $cell)
```

which is the same as using the first variant with ALP_Area_EntryGotoCell.

Note: ALP_Area_EntryGotoCell and ALP_Area_EntryGotoGridCell expect the cell number in the grid, not the column number.

If you use AreaList Pro in compatibility mode, the columns are physically reordered when moved using Drag & Drop. The cell number corresponds to the column number. But when you use native AreaList Pro 9 API mode, the columns are not reordered (or you defined the grid explicitly) and the cell number is not necessarily the same as the column number.

ALP_Area_EntryGotoColumn (like v8 ***AL_GotoCell***) searches the first cell displaying the requested column.

See the Grid section for more information about grids.

"Undo" value

When data entry is initiated on an AreaList Pro cell, the array contents for the element corresponding to that cell are copied to the zero element of the same array.

Since this element is usually never used, it makes a convenient storage place for the data in case you wish to revert to the old value; however, you should take care not to use this zero array element elsewhere in your code while data entry is in progress.

Saving field values

If you set the value with 4D for a field that is displayed in an AreaList Pro area, this new value won't be saved. You must set the value through AreaList Pro, not 4D. Instead of:

```
[MyTable]MyField:=$value
```

use:

```
AL_SetAreaTextProperty (area;ALP_Area_EntryValue;$value)  // set the value
```

or set the field value as above, but then:

```
AL_SetAreaLongProperty (area;ALP_Area_ClearCache;$row)  // refresh the row
```

Do not forget that from AreaList Pro's side, the area is just editing a record in the [MyTable] table, and when it ends editing, it saves the edited values into the database.

This means that you should not change or save any record while edited in AreaList Pro. On the other hand, if you change data in any non-AreaList Pro currently edited table (e.g. related), you obviously have to load, modify and save the record with 4D.

Checkboxes

Checkboxes can be used to enter and display boolean values.

In addition, the ALP_Column_EntryControl property set to 1 will display checkboxes with title.

There is one (and only one) condition for this setting to be used: when entry is started for a boolean column not displaying checkboxes (ALP_Column_DisplayControl is -1, formatted values) and ALP_Column_EntryControl is set to 1 (checkbox with title).

In this case, a checkbox control with title is displayed in the cell for entry.

The title is the **True** label of ALP_Column_Format (or defaults to ALP_Area_DefFmtBoolean if previously set).

Example:

```
AL_SetColumnTextProperty ($area; $column; ALP_Column_Format; "Yes;No")  // True ; False
AL_SetColumnLongProperty ($area; $column; ALP_Column_DisplayControl; -1)  // formatted
AL_SetColumnLongProperty ($area; $column; ALP_Column_EntryControl; 1)  // checkbox with title
```

You can also set up 3-states checkboxes as in 4D:

"Check box objects accept a third state. This third state is an intermediate status, which is generally used for display purposes. It allows, for example, indicating that a property is present in a selection of objects, but not in each object of the selection."

(4D Design Reference manual).

In order for a checkbox to take control of this third state in AreaList Pro, you must use a integer or long integer column type (values are 0, 1, or 2 for the intermediate state) and combine two settings with ***AL_SetColumnLongProperty***:

- ALP_Column_DisplayControl must be set to 0, 1, 2 or 4
- ALP_Column_EntryControl is set to 1

Note: you can use pictures to display your own checkboxes, including three-states.
See Displaying custom checkboxes using pictures. and Example 16 in the Examples Database.

Bullet “Password” characters

You may want to set a text column to a “password” bullet type display in order to mask the actual characters including while the user enters text in an edited cell (as with the old 4D "%Password" font).

Set the format of a text column to "•" to display bullets instead of the actual values and to use "•" for password type entry as well:

```
AL_SetColumnTextProperty ($area; $column; ALP_Column_Format; "•")  // as a string
AL_SetColumnTextProperty ($area; $column; ALP_Column_Format; Char(8226))  // as a decimal value
AL_SetColumnTextProperty ($area; $column; ALP_Column_Format; Char(0x2022))  // as hexa
```

Entering data in AreaList Pro with DisplayList

You can use DisplayList to display a custom selection list for data entry in AreaList Pro.

1. Make the area enterable:

```
AL_SetAreaLongProperty ($area;ALP_Area_EntryClick;3)  // enterable by cmd-double-click
```

2. Set a column to allow entry using popup:

```
AL_SetColumnLongProperty ($area;$column;ALP_Column_Enterable;3)  // keyboard & popup
```

3. Don't set a popup array/menu.

4. Install callback using ALP_Area_CallbackEntryPopup:

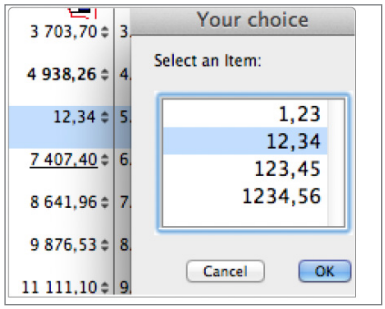
```
AL_SetAreaTextProperty($area;ALP_Area_CallbackMethPopup;"_Alp_PopupCallback")  // entry popup callback
```

5. Implement the callback:

```
// _Alp_PopupCallback
// this function is called when a cell has popup entry allowed, but no popup is defined
C_BOOLEAN($0)  // case handled? return False if not handled
C_LONGINT($1)  // AreaList Pro object reference
C_LONGINT($2)  // row
C_LONGINT($3)  // column
C_LONGINT($4)  // data kind
C_LONGINT($alpEditArea;$alpEditRow;$alpEditCol;$alpDataKind)
C_LONGINT($err)
$alpEditArea:=$1
$alpEditRow:=$2
$alpEditCol:=$3
$alpDataKind:=$4
```

```
Case of
  : ($alpDataKind=Is date)  // handle date popup - use custom date dialog
    C_DATE(vDate;vDate2)
     // vDate:=DatePicker Display Dialog - We use our own:
    $err:=AL_GetAreaPtrProperty ($alpEditArea;ALP_Area_EntryValue;->vDate2)
    C_LONGINT($mx;$my;$mb)
    GET MOUSE($mx;$my;$mb;*)
    $win:=Open form window("alpDatePicker";Pop up form window;$mx;$my)
    DIALOG("alpDatePicker")
    CLOSE WINDOW($win)
    If (OK=1)  // (vDate#!00/00/00!)
      $err:=AL_SetAreaPtrProperty ($alpEditArea;ALP_Area_EntryValue;->vDate)
    End if
    $0:=True
  : ($alpDataKind=Is time)  // handle time popup - not implemented, use default
  : ($alpDataKind=Is real)  // real column - demo how to use DisplayList
    C_REAL($v)
    ARRAY REAL($a1;4)
    $a1{1}:=1.23
    $a1{2}:=12.34
    $a1{3}:=123.45
    $a1{4}:=1234.56
    $v:=AL_GetAreaRealProperty ($alpEditArea;ALP_Area_EntryValue)
    $err:=Find in array($a1;$v)
    SetListLine ($err)
    SetListSize (0;0;3)  // SetListSize (200;200;1)
    $err:=DisplayList ($a1)
    If ($err#0)
      AL_SetAreaRealProperty ($alpEditArea;ALP_Area_EntryValue;$a1{$err})
      [Properties]Property:=String($err)
    End if
    $0:=True
Else
    // not handled - use default ("no values")
End case
```

Here is the result with the Is real case (using DisplayList):



Popup entry in specific cells

You can create an interface where some cells in a column have a popup icon while others do not.

1. Make the area enterable.

```
AL_SetAreaLongProperty ($area;ALP_Area_EntryClick;3)  // enterable by cmd-double-click
```

2. Make the columns enterable. Those which should display popups must be “enterable by popup” (could be with keyboard, too).

```
// all columns enterable with keyboard only:
AL_SetColumnLongProperty ($area; -2; ALP_Column_Enterable; AL Column entry typed only)
// $column1 enterable by popup only:
AL_SetColumnLongProperty ($area; $column1; ALP_Column_Enterable; AL Column entry popup only)
// $column2 enterable by popup only:
AL_SetColumnLongProperty ($area; $column2; ALP_Column_Enterable; AL Column entry popup only)
```

3. Make some cells not enterable based on the value.

```
For ($row; 1; Size of array (myArray))
   If (myArray{$row} # "@chair@")
      AL_SetCellLongProperty ($area; $row; $column1; ALP_Cell_Enterable; 0)
   End if
   If (myArray{$row} # "@table@")
      AL_SetCellLongProperty ($area; $row; $column2; ALP_Cell_Enterable; 0)
   End if
End for
```

Default for enterability of a cell is -1, which means “inherit the column's enterability”.

The cell enterability property (ALP_Cell_Enterable) can be set to -1 or to any value allowed for column enterability (ALP_Column_Enterable).

- When the column is set to be enterable by popup, but a cell is **not** enterable by popup (0 = no entry, 1 = keyboard only), the popup icon is not displayed in the cell, but space is reserved for it.
- Conversely, when the column is **not** set to be enterable by popup, the space for the icon is not reserved (and the icon is not drawn) even if the cell is set to be enterable by popup.

Leaving a Cell

Leaving a cell can be triggered by three methods:

- User click on another part of the AreaList Pro area (not on a non focusable 4D object or another window from any application since version 9.9).
- Return or Tab keys, with or without shift, to the next or previous (shift) enterable cell in the AreaList Pro area (note that the Enter key can be mapped to Return or Tab according to the ALP_Area_EntryMapEnter property).
- Programmatically.

Events

In many situations you will want to know what the user did: which cell they edited; which row they dragged; and so on. You can get this information by calling the ***AL_GetAreaLongProperty*** command with the ALP_Area_AlpEvent option.

For example, to find out how many columns the user sorted on after opening the AreaList Pro Sort Editor, you can use the following code in the area's object method:

```
$event:=AL_GetAreaLongProperty (Self->;ALP_Area_AlpEvent)
Case of
  : ($event=AL Sort editor event)
    $sorted:=AL_GetAreaTextProperty (Self->;ALP_Area_SortList)  // gets list of sorted column numbers
    $sortcount:=AL_GetAreaLongProperty (Self->;ALP_Area_Sort)  // how many columns were sorted?
End case
```

You can find a complete list of event codes and their meanings in the AreaList Pro Event codes section.

Sorting

Areas can be sorted either by clicking in a column header or by invoking the AreaList Pro sort editor. If the user clicks in a column header, the area is sorted in ascending order on that column; if he clicks again, the column will be sorted in descending order.

If he wants to sort on more than one column, he can use the sort editor.

The developer can control many aspects of sorting, including disabling the sort option entirely and "hijacking" the user sort if he wants to handle it in a certain way.

When a column has been sorted, a triangle (sort indicator) appears in the header:

When both ALP_Area_HeaderMode and ALP_Area_ShowSortIndicator properties are not zero, the v8 sort order button is displayed above the vertical scrollbar:

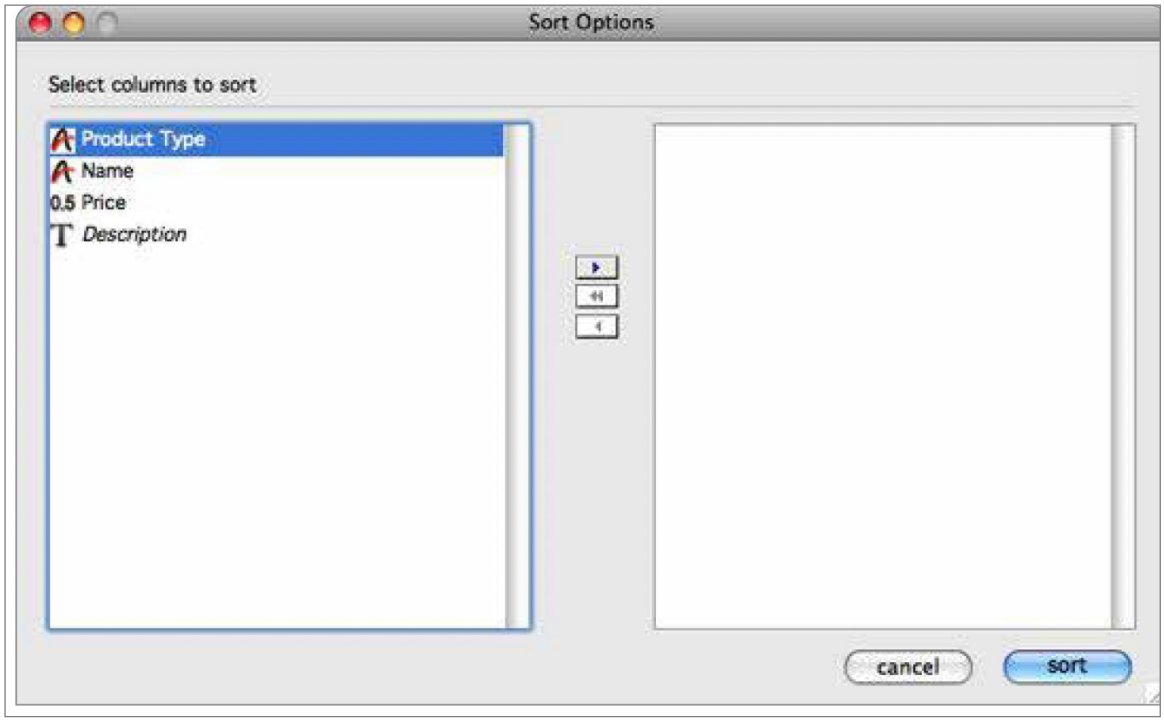
On Windows Vista, 7 and above, value 2 to ALP_Area_ShowSortIndicator draws the sort (non native) triangle to the right, not on top.

The Sort Editor

If you want your users to be able to use the sort editor, you must first enable it by calling ***AL_SetAreaLongProperty*** with the ALP_Area_AllowSortEditor option - for example:

```
AL_SetAreaLongProperty (area;ALP_Area_AllowSortEditor;1)
```

To activate the Sort Editor, the user cmd-clicks in the column header. The Sort Editor window then opens:



To add a column to the sort list, double-click it, drag and drop it into the right-hand area, or select it and click the right arrow.

After the user has completed a sort, you can find out which columns were sorted by calling the ALP_Area_SortList option of ***AL_GetAreaTextProperty*** - for example:

```
$event:=AL_GetAreaLongProperty (Self->;ALP_Area_AlpEvent)
Case of
  : ($event=AL Sort editor event)
    $sorted:=AL_GetAreaTextProperty (Self->;ALP_Area_SortList)
End case
```

$sorted now contains a comma-separated list of the columns that the user sorted. If a negative number is shown, that column was sorted in descending order.

Button labels

The ALP_Area_SortOK and ALP_Area_SortCancel properties can be used to override the default values for the “Sort” and “Cancel” buttons. It either one of these is empty (not set) then AreaList Pro will load the button labels from the “ALP.xlf” file located in the localized subfolder of the Resources folder in the AreaList Pro bundle (depending on the languages available as XLF).

See AreaList Pro Area Sort Properties in the Properties by Theme section.

Taking control of the Sort

It is possible for the developer to “hijack” a user sort and take control of it. Why would you want to do that? Here’s an example:

In your AreaList Pro area you have **First Name** and **Last Name** columns. If a user clicks on the **Last Name** column header, you want to sort the area by Last name **and** First Name. We place the following code in the AreaList Pro object’s method:

```
$event:=AL_GetAreaLongProperty (Self->;ALP_Area_AlpEvent)
Case of
  : ($event=AL Sort button event)  //user clicked a column header
    $selected:=AL_GetAreaLongProperty (Self->;ALP_Area_SortColumn)  //which column?
    Case of
      : ($selected=2)  //last name
        $sort:="2,1"  //sort on Last name, First name
        AL_SetAreaTextProperty (Self->;ALP_Area_SortList;$sort)
    Else
      $sort:=String($selected)
      AL_SetAreaTextProperty (Self->;ALP_Area_SortList;$sort)  //sort on the user’s column
    End case
End case
```

Note: ALP_Area_Sort_Column is always the column number (does not contain the sort direction). Use ALP_Area_Sort_List if you want to know the direction.

Setting the sort indicator and sorted column list

You can indicate yourself the sorted column(s) by using either ALP_Area_SortListNS or ALP_Object_SortListNS (asking AreaList Pro to set the sorted columns without doing the actual sort):

```
AL_SetAreaTextProperty($area;ALP_Area_SortListNS;"-1,2")
AL_SetObjects($area;ALP_Object_SortListNS;$sortArray)
```

Note: NS in the constant name stands for “No Sort”.

Bypassing the Sort editor

Using 4D code

You may want to sort the arrays on multiple criteria via code and not using AreaList Pro sort features.

You can sort the arrays manually and then tell AreaList Pro how it is sorted using **MULTI SORT ARRAY** and:

```
$err:=AL_SetObjects ($area;ALP_Object_SortListNS;$arraysSortOrder).
```

This is the same as using v8 ***AL_SetSortedCols***.

Or you can use ***AL_SetAreaTextProperty***:

```
AL_SetAreaTextProperty ($area;ALP_Area_SortListNS;$sortList).
```

Using AreaList Pro

You can directly tell AreaList Pro how to sort using:

```
$err:=AL_SetObjects ($area;ALP_Object_SortList;$arraysSortOrder).
```

This is the same as using v8 ***AL_SetSort***.

Alternatively you can use ***AL_SetAreaTextProperty***:

```
ARRAY INTEGER ($arraysSortOrder;4)
$arraysSortOrder{1}:=-7  // descending on column 7
$arraysSortOrder{2}:=5  // ascending on column 5
$arraysSortOrder{3}:=1
$arraysSortOrder{4}:=-14
$sortList:="-7,5,1,-14"
AL_SetAreaTextProperty ($area;ALP_Area_SortList;$sortList)
```

Here is an example where a click on column 7 (visible) will actually trigger a sort on column 8 (invisible) yet show the indicator in the clicked column (including the sort direction).

When you set the sort list, AreaList Pro simply uses what you provided (as long as the column numbers are in the range). If the first column to be sorted on is hidden, no visible column will be shown as the sort column.

When you sort the arrays yourself, you just need to tell AreaList Pro that it must reload them.

But when you sort using AreaList Pro, you have to specify which column to mark as the sort column:

```
// handle a click into a column header when sorting is set to be bypassed
If ((AL_GetAreaLongProperty ($area;ALP_Area_AlpEvent)=AL Sort button event)\
      & (AL_GetAreaLongProperty ($area;ALP_Area_UserSort)=AL User sort bypass))
   $sort:=AL_GetAreaLongProperty ($area;ALP_Area_SortList)
   Case of
      : ($sort=7)  // 7th column ascending
         AL_SetAreaLongProperty ($area;ALP_Area_SortList;8)  // sort on column 8 ascending
         AL_SetAreaLongProperty ($area;ALP_Area_SortListNS;7)  // but highlight column 7 header
      : ($sort=-7)  // 7th column descending
         AL_SetAreaLongProperty ($area;ALP_Area_SortList;-8)  // sort on column 8 descending
         AL_SetAreaLongProperty ($area;ALP_Area_SortListNS;-7)  // but highlight column 7 header
   Else
      AL_SetAreaLongProperty ($area;ALP_Area_SortList;$sort)  // sort the clicked column
   End case
End if
```

Internal Sorting

When you are displaying arrays, the default behaviour is for arrays to be physically sorted. However, you can turn physical sorting off using the ALP_Area_DontSortArrays property - for example:

AL_SetAreaLongProperty (area; ALP_Area_DontSortArrays;1)

This is required (it is set internally) when displaying a hierarchy (the hierarchy can be sorted).

When ALP_Area_DontSortArrays is on:

When the user clicks a row, the real row number in your arrays is reported - independently from the position on-screen:

C — 1

B — 2

A — 3

D — 4

A click on “A” will report 3 as the clicked row

If the data is sorted on the first column, the user will see:

A — 3

B — 2

C — 1

D — 4

A click on “A” will report 3 as the clicked row (and because the arrays are untouched, the third element is “A”).

You can request the internal sort order as follows:

```
ARRAY LONGINT($aIndex;0)
$err:=AL_GetObjects (area;ALP_Area_Sort;$aIndex)
```

You will get 3, 2, 1, 4 in $aIndex.

In other words, if internal sorting is on, arrays are not sorted - the row order is sorted internally by AreaList Pro. If internal sorting is off, the arrays are all sorted.

Why might you want to turn internal sorting on?

Because it will give improved performance.

For example, let’s suppose you have 40 parallel arrays, and the user is presented with just 3 of those arrays.

But because it can be sorted, you have to add the rest of the arrays as invisible, and AreaList Pro has to sort (physically move all data in) all 40 arrays.

You don’t have to add all the arrays; you can add just one index array and access all the other arrays indirectly.

When you use this feature, you only have to add the 3 displayed arrays - the order of the arrays will not change.

AreaList Pro will sort only the internal index array.

Note: when a hierarchical list is shown, internal sorting is always turned on when a sort is performed.

Calculated columns

Calculated columns are not sortable (a click in header is ignored) if you don't bypass the internal sorting.

In this case, you can get the clicked column, sort the selection and then set the actual sort order. And yes, setting a calculated column as a sorted column is allowed.

Use ALP_Area_SortListNS to mark the column as sorted and display the sort indicator in the header.

Your code might look like this:

```
: (Form event=On Plug in Area)
   $event:=AL_GetAreaLongProperty (Self->;ALP_Area_AlpEvent)
If ($event=AL Sort button event)
      If (AL_GetAreaLongProperty (Self->;ALP_Area_ClickedCol)=2)  // Column to sort is 2
         AL_SetAreaLongProperty (Self->;ALP_Area_ClearCache;-2)  // Update all rows
         $colToSort:=AL_GetAreaLongProperty (Self->;ALP_Area_SortList)
         If (Abs($colToSort)=$clickedCol)
            $colToSort:=-$colToSort
         Else
            $colToSort:=2
         End if
         AL_SetAreaLongProperty (Self->;ALP_Area_SortListNS;$colToSort)
         If ($colToSort>0)
            ORDER BY([Order];[Order]FieldToSortOn;>)
         Else
            ORDER BY([Order];[Order]FieldToSortOn;<)
         End if
   End if
End if
```

Here is another way: make AreaList Pro to bypass the internal sorting by setting ALP_Area_UserSort to AL User sort bypass.

```
AL_SetAreaLongProperty (Self->;ALP_Area_UserSort;AL User sort bypass)
```

In this case, calculated columns are allowed (the click in header is not ignored), but you must handle all the sorting yourself.

When you get the AL Sort button event, ask for the clicked column including sort direction:

```
$colToSort:=AL_GetAreaLongProperty (Self->;ALP_Area_SortList)
```

then handle the sorting - but for all columns, not just for the calculated column(s).

Comma-separated list vs array

If you don't like the idea of a comma-separated list of sort columns, you can use an array of sort columns. In some cases parsing a sort to comma separated values is more complex than parsing to an array.

```
ARRAY INTEGER($aiSortList;0)
$err:=AL_GetObjects($area;ALP_Object_SortList;$aiSortList)  // get order
ARRAY LONGINT($alSortList;2)
$aiSortList{1}:=-4
$aiSortList{2}:=7
$err:=AL_SetObjects($area;ALP_Object_SortList;$aiSortList)  // set order, sort data
```

If your columns data are already sorted, you can use:

```
$err:=AL_SetObjects($area;ALP_Object_SortListNS;$aiSortList)  // set order, don't sort data
```

Restoring highlighted selection

If you want AreaList Pro (or a given area) to always restore the highlighted selection after a sort in field mode, set the ALP_Area_SelPreserve property to true. Otherwise, the ALP_Object_RowSelection property is the way to perform an action in AreaList Pro similar to 4D's **GET HIGHLIGHTED RECORDS**, then **HIGHLIGHT RECORDS**.

AreaList Pro uses 4D record IDs though, not UserSet or any other set as 4D does.

```
// Get the currently selected record numbers from the user's row selection
ARRAY LONGINT($records;0)
$err:=AL_GetObjects ($area;ALP_Object_RowSelection;$records)
// Sort your 4D selection
ORDER BY([Table];[Table]Field1;>;[Table]Field2;<)
// Inform user how the selection is ordered
AL_SetAreaTextProperty ($area;ALP_Area_SortListNS;"1;-3")  // ordered by first and third columns
// (alternate method) the line above is the same as using an array with sort order information:
ARRAY LONGINT($order;2)
$order{1}:=1  // first column, ascending
$order{2}:=-3  // third column, descending
AL_SetObjects ($area;ALP_Object_SortListNS;$order)
// (end of alternate method)
// Inform AreaList Pro of 4D's selection change (clear the cache, fetch data)
AL_SetAreaLongProperty ($area;ALP_Area_UpdateData;0)
If (Size of array($records)>0)  // will also work if this test is omitted
   // Restore the user's row selection using record numbers
   $err:=AL_SetObjects ($area;ALP_Object_RowSelection;$records)
   // Make the selected record visible
   AL_SetRowLongProperty ($area;AL_GetAreaLongProperty ($area;ALP_Area_SelRow);ALP_Row_Reveal;0)
End if
```

Typeahead

When at least one column is sorted the user can type one or several characters to scroll the list to the desired value in the sorted column. This “typeahead” is available both in array and field modes (set the ALP_Area_TypeAheadFieldMode property to true in order to enable typeahead in field mode).

The behavior depends on the ALP_Area_TypeAheadEffect property:

Value	Behavior
-2	report AL Typeahead event (no search)
-1	ignore typeahead (do nothing)
0	select first matching row (value >= "search")
1	select first matching row if selection is empty and scroll view to show first matching row otherwise (legacy behavior, no event reported)
2	change the selection to matching rows (value = "search@") – selection will be empty if no matching value is found

Note: at least one column must be sorted (only the first sorted column will be considered).

Set ALP_Area_TypeAheadEffect to 1 to get the old AreaList Pro v8 behavior: on typeahead, the selection is not changed when selection mode is multiple rows selection and the current selection is not empty – only the view is scrolled to show the first matching row.

AL Typeahead event is available with ALP_Area_TypeAheadEffect = -2. Nothing else happens with this setting, besides updating ALP_Area_TypeAheadString. It is also available with 0 or 2.

No event is reported when ALP_Area_TypeAheadEffect = -1 (ignore) or 1 (legacy mode for backward compatibility).

ALP_Area_TypeAheadString contains the string that was typed within twice the OS-set double-click time (this value can be modified using the ALP_Area_TypeAheadTime property). This is true in both array and field modes, unless ALP_Area_TypeAheadEffect.= -1.

Values 0 (default) and above perform a search:

- In field mode, the query is executed after the timeout (ALP_Area_TypeAheadTime).
- In array mode, the query is executed immediately after each keystroke.
- The query is performed within the sorted column adding a “@" to the string.
- Attributed "styled" text is supported (the query uses AL_GetPlainText in this case).

Note: when typehead is activated in field mode, the query used is **QUERY SELECTION** when possible (plain text/alpha field), **QUERY SELECTION BY FORMULA** otherwise.

As of version 11.1, the Setter for ALP_Area_TypeAheadString executes the type-ahead in focused target when possible - similarly to user typing the specified text: you can scroll/select row in AreaList Pro with code, e.g. assuming Example 15 is sorted by city, you can select a row:

AL_SetAreaLongProperty(eList; ALP_Area_SortList; 1)

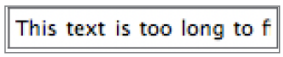
AL_SetAreaTextProperty(eList; ALP_Area_TypeAheadString; "Orlando")

Text wrapping

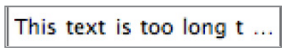
Wrapping text in a cell means that long lines will be split (on word boundary if possible, in a middle of a word otherwise) and will continue on the next line if possible.

Note: when word wrapping is enabled and a word does not fit into a column width, the word is split (e.g. “TEXT” can be split into “TE” + “XT” on next line).

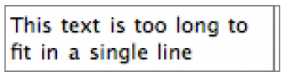
This text is too long to fit into the cell:



We could set the ALP_Area_UseEllipsis property on to indicate the overflow:



Or we can set wrapping on so that the text fits on several lines:



Note: regardless of the wrapping mode, text is always split on CR or LF and will continue on next line if possible. In other words explicit line breaks are always honored, even with wrapping off.

The ALP_Area_NumXXXLines properties must be used if you want the wrapped text to be displayed on several lines:

- ALP_Area_NumHdrLines (column header)
- ALP_Area_NumRowLines (column rows)
- ALP_Area_NumFtrLines (column footer)

Use of ALP_Area_MaxRowLines (new in v11.3)

When ALP_Area_NumRowLines = 0 (variable row height) and ALP_Area_MaxRowLines is set to a value greater than 0, at most this value will be displayed.

In that case when ellipsis is to be used (e.g. ALP_Column_UseEllipsis), last line will display it.

Note: that ellipsis addition to the last line is implemented only for left aligned text (default alignment for text is mapped to left), does not work for any other alignment (where ellipsis is at the beginning or in the middle of a long line).

ALP_Area_MaxRowLines differs from ALP_Area_MaxCellHeight in:

- number of lines of text instead of points
- ellipsis for left aligned text can be drawn at end of last visible line

Example

Having left aligned text "111\r222\r333" and ellipsis active (e.g. ALP_Column_UseEllipsis)

- ALP_Area_NumRowLines = 1 (old AreaList Pro v7 behavior: ignore \r, draw single line)

  111222333

- ALP_Area_NumRowLines = 0, ALP_Area_MaxRowLines = 0:

  111
  222
  333

- ALP_Area_NumRowLines = 0, ALP_Area_MaxRowLines = 1:

  111…

- ALP_Area_NumRowLines = 0, ALP_Area_MaxRowLines = 2:

  111
  222…

- ALP_Area_NumRowLines = 0, ALP_Area_MaxRowLines >= 3:

  111
  222
  333

Compatibility mode on

In compatibility mode (ALP_Area_Compatibility set to 1) wrapping will occur whenever the ALP_Area_NumXXXLines value is different than 1 (= 0 or > 1).

Note: calling either old ***AL_SetRowOpts*** or ***AL_SetColOpts*** will set the compatibility mode on.

Compatibility mode off

The relevant wrapping properties must also be set when ALP_Area_Compatibility is set to 0:

- ALP_Column_HdrWrap (column header)
- ALP_Column_Wrap (column rows)

  AL_SetColumnLongProperty ($eList;4;ALP_Column_Wrap;1) // wrap long lines in column 4

- ALP_Column_FtrWrap (column footer)
- ALP_Row_Wrap (individual row, supersedes column wrapping setting if any)
- ALP_Cell_Wrap (individual cell, supersedes column and row wrapping setting if any)

For example, if you are setting the ALP_Column_Wrap property and if you want to use variable row height:

```
AL_SetAreaLongProperty ($eList;ALP_Area_NumRowLines;0)  // variable row height
AL_SetColumnLongProperty ($eList;4;ALP_Column_CalcHeight;1)
// the above line is meant to calculate each row / header / footer height using this column's data
```

Note: ALP_XXX_Wrap properties are ignored in compatibility mode.

Text Styling

AreaList Pro gives you many options for styling text within the following objects:

- cells
- columns
- column footers
- column headers
- rows

You can set the following attributes:

- font
- font size
- font style (bold, italic, underline)
- uppercase
- rotation
- color
- wrapping (see above)
- alignment
- horizontal scaling
- line spacing
- baseline shift
- dynamic row height
- automatic text truncation (ellipsis)

The following tables describe which properties you can use to style text in the various objects.

Column Properties

Use these properties with commands in the Columns theme.

All the available column properties are listed in the Columns section of the Properties chapter.

Row Properties

Use these properties with commands in the Rows theme:

All the available row properties are listed in the Rows section of the Properties chapter.

Cell Properties

Use these properties with commands in the Cells theme.

All the available cell properties are listed in the Cells section of the Properties chapter.

Object Properties

Use these properties with commands in the Objects theme. Note: We're talking about plugin objects here, not programming objects!

All the available object properties are listed in the Objects section of the Properties chapter.

Formatting

Column property

The format for a given column is set by the ALP_Column_Format column property.

In addition you can use the Advanced properties dialog, and of course the area's XML description.

Custom styles

4D's custom styles are identified by the "OR" sign used as a prefix "|".

"|format name" type parameters will be interpreted by AreaLlist Pro for the column display, whether set from advanced properties, XML or the ALP_Column_Format property:

AL_SetColumnTextProperty ($area; $column; ALP_Column_Format; "|format")

Note: text type formats will only access the styles available in the host database, not components.

Empty string for null dates

Since 4D formats for dates are numbers and the AreaList Pro format a column text property, the date format number must be passed as as string:

AL_SetColumnTextProperty ($area; $column; ALP_Column_Format; "107")

The 4D layout editor has a display property for dates “Empty if null”. This avoids the display of empty dates as "00/00/00". This property can be set with 4D code using the **String** command, by adding 100 (Blank if null date) to the date format number, e.g. "107" = **String** (Blank if null date | Internal date short). AreaList Pro will honor these settings.

In addition, there is a special AreaList Pro format: "xxx-" (xxx is the format, e.g. "7" becomes "7-").

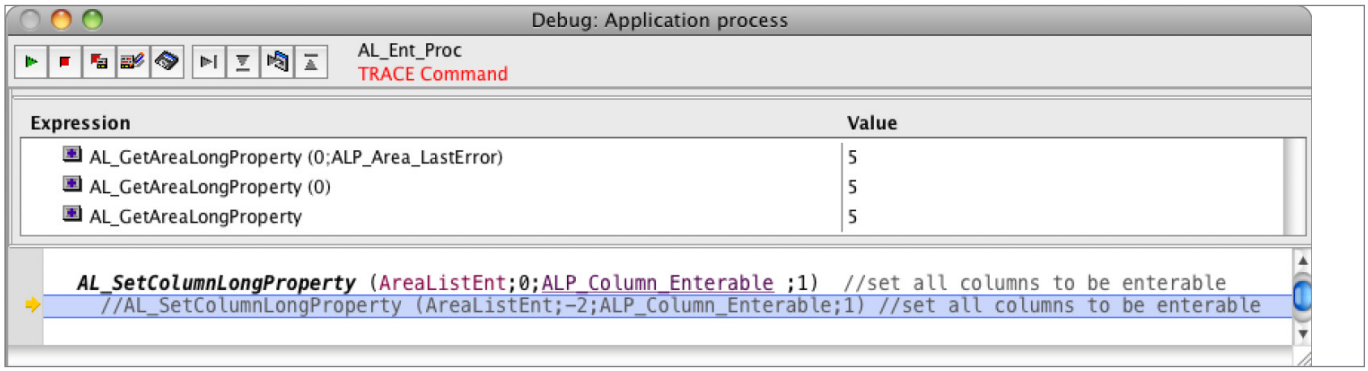
The trailing minus sign "-" means “use empty string for null date”. The following line will do exactly the same as the above example:

AL_SetColumnTextProperty ($area; $column; ALP_Column_Format; "7-")

Using the debugger

Trace mode

When the ALP_Area_TraceOnError property’s bit 0 is set to true in interpreted mode (the default), if there is an error in a command that does not return an error code, and you are using 4D in interpreted mode, the 4D debugger window will automatically open with the line immediately following the problem line highlighted:



In this example, the command is invalid (5 = ALP_Err_InvalidRequest, see Error codes).

Zero as column number is used to set default style properties for newly created columns, but enterability is not a property of a style. The correct call (commented out next line) is:

AL_SetColumnLongProperty (AreaListEnt;-2;ALP_Column_Enterable;1)

//-2 = update all existing columns

Getting the last error

The error is retrieved through the ALP_Area_LastError property, which is global to all AreaList Pro areas.

This last error can be displayed in the debugger window using either one of the following:

AL_GetAreaLongProperty (0;ALP_Area_LastError) // full syntax

AL_GetAreaLongProperty (0) // shortened syntax

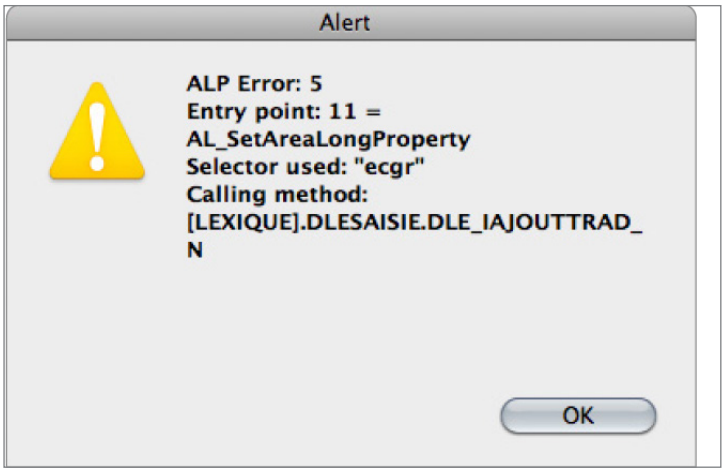
AL_GetAreaLongProperty // short syntax

Always use the full syntax in your code if you want to test the error programmatically.

The short syntax provides a convenient and quick way to type into the debugger window and get the error code.

Compiled mode

In compiled mode, if ALP_Area_TraceOnError property's bit 1 is set to true an alert is displayed with the error code, the AreaList Pro command, the calling 4D method and the property used (selector, see Property Values, Constants and XML Names):



Read-only mode

With AreaList Pro v9.9.1 and above, an area can be made partially or completely read-only. This is achieved using the ALP_Area_ReadOnly property, which is a combination of bits:

Bit	Effect
0	Make area not enterable
1	Make area not droppable (ignore drag)
2	Make area not draggable

For example, we want to prohibit drag and drop from and to this area, regardless of any drag and drop settings:

AL_SetAreaLongProperty ($eList;ALP_Area_ReadOnly;6) // bits 1 and 2 on

8 Using the Callback Methods

A “callback” is a 4D project method which is executed by a plug-in. AreaList Pro lets you make use of callbacks when displaying or printing an AreaList Pro area.

Callbacks are method types that allow you to react according to a user’s actions - for example, to carry out data validation, or to enable/disable buttons depending on which options are available.

There are eight actions that can trigger a callback:

- An AreaList Pro event
- AreaList Pro area selected
- AreaList Pro area deselected
- Cell entered
- Cell exited
- Popup entry
- Edit menu action
- End of a page (during printing)

In addition, Calculated Columns use a callback method to perform their calculations in field display or array display mode.

Callback Parameters

All callbacks receive the area long integer reference as their first parameter ($1) You must use the following declaration in your callback method:

```
C_LONGINT ($1)
```

Since the long integer $1 parameter contains 4D's representation of the AreaList Pro object, it can be used as the first parameter of any AreaList Pro method called.

Most callback methods receive additional parameters, which need to be declared also, as documented below.

Some callback methods are actually functions, and they return a value.

- The callback set by ALP_Area_CallbackMethOnEvent returns a longint, so $0 must be declared as a longint. If the returned value is 0, no further code will be executed on event (neither object method nor form method).
- The callback set by ALP_Area_CallbackMethEntryEnd returns either True or False; if it returns False (rejected), the user will not be allowed to leave the cell. This enables you to do all kinds of data validation.
- The callback set by ALP_Area_CallbackMethPopup returns either True or False; if it returns True (click handled) AreaList Pro won't display its own popup.
- The callback set by ALP_Area_CallbackMethMenu returns a longint. If the returned value is 0, AreaList Pro will process the Edit menu action, otherwise the callback has overridden this processing (AL Edit Menu Handled Mask).

Event

Property: ALP_Area_CallbackMethOnEvent

Parameters:

```
C_LONGINT($1)  // AreaList Pro object reference
C_LONGINT($2)  // AreaList Pro event
C_LONGINT($3)  // 4D event
C_LONGINT($4)  // last clicked column (or column under the pointer for mouse moved event)
C_LONGINT($5)  // last clicked row (or row under the pointer for mouse moved event)
C_LONGINT($6)  // modifiers
C_LONGINT($0)
```

The $0 result is a combination of two bits:

- bit 0: call the object method and form method
- bit 1: don't update variables

The combination gives one of the following values:

- 0 = update variables
- 1 = update variables, call the object method and form method
- 2 = do nothing
- 3 = call the object method and form method

Note: the above results are not relevant to Drag and Drop events where $0 is only used in the context of an external object being dragged over the area (AL Allow drop event). It is used to allow or reject the drop (see Using the Event callback method).

Area selected

Property: ALP_Area_CallbackMethSelect

Parameter:

```
C_LONGINT($1) // AreaList Pro object reference
```

Area deselected

Property: ALP_Area_CallbackMethDeselect

Parameter:

```
C_LONGINT($1) // AreaList Pro object reference
```

Cell entered

Property: ALP_Area_CallbackMethEntryStart

Parameters:

```
C_LONGINT($1) // AreaList Pro object reference
C_LONGINT($2) // entry cause
C_LONGINT($3) // record loaded: will only exist when fields are being displayed
```

Cell exited

Property: ALP_Area_CallbackMethEntryEnd

Parameters:

```
C_LONGINT($1) // AreaList Pro object reference
C_LONGINT($2) // exit cause
C_BOOLEAN($0) // allow cell exit
```

Popup entry

Property: ALP_Area_CallbackMethPopup

Parameters:

```
C_LONGINT($1) // AreaList Pro object reference
C_LONGINT($2) / row
C_LONGINT($3) // column
C_LONGINT($4) // data type
C_BOOLEAN($0) // True if handled; False if not handled
```

Edit menu action

Property: ALP_Area_CallbackMethMenu

Parameters:

```
C_LONGINT($1)  / AreaList Pro object reference
C_LONGINT($2)  // edit event
C_LONGINT($0)
```

Compatibility note: previous versions used a text $3 parameter. This third parameter was used as the return value for Undo string, but 4D no longer supports this feature.

Calculated column

Property: ALP_Column_Callback

Parameters:

```
C_LONGINT($1)  // AreaList Pro object reference
C_LONGINT($2)  // column number
C_LONGINT($3)  // type of data in this column
C_POINTER($4)  // pointer to temporary 4D array (field mode) or sized 4D array (array mode)
C_LONGINT($5)  // first record for which to calculate cell
C_LONGINT($6)  // number of cells to calculate in column
```

Printing Page End

Property: PLP_PageEndCallback

See PL_SetPageProc in the PrintList Pro chapter.

Properties to use with Callbacks

The following Callback properties can be used with the Area and Column command themes

Area properties

Use these properties with commands in the Area theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Area_CallbackMethEntryEnd	✔	✔	✔	text				End entry callback function. The return value can be used for validation; the default value is False
ALP_Area_CallbackMethEntryStart	✔	✔	✔	text				Start entry callback method (area; action; {recLoaded])
ALP_Area_CallbackMethPopup	✔	✔	✔	text				Popup entry callback method (area; row; column; dataType) -> bool:Handled For popup handling: used when a popup is clicked but no popup array/menu is defined The callback is called as function: return False to invoke internal implementation, otherwise use ***AL_SetAreaXXXProperty*** ($1;ALP_Area_EntryValue;$value) to actually set the new value and return True
ALP_Area_AlpEvent	✔			int				Last AreaList Pro event: see AreaList Pro Event codes May be used with AreaRef set to zero (last event in any area)
ALP_Area_CallbackMethDeselect	✔	✔	✔	text				Area deselected callback method (area)
ALP_Area_CallbackMethMenu	✔	✔	✔	text				Edit menu callback function (area; event) -> long:flags See the list of the Edit menu constants
ALP_Area_CallbackMethOnEvent	✔	✔	✔	text				Event callback function (area; alpEvt; 4Devent; column; row; modifiers)
ALP_Area_CallbackMethSelect	✔	✔	✔	text				Area selected callback method (area)
ALP_Area_ToolTip	✔	✔		text				Tool Tip text To be set from the event callback function

Column Property

Use this property with commands in the Columns theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Column_Callback	✔	✔	✔	text				Callback method for a calculated column (area; column; type; ptr; first; count) If the AreaList Pro area displays several calculated columns, the callback methods will be called in the column number order

Setting up a Callback

There are two things you need to do in order to use a callback method:

1. Create a method to execute the callback
2. Set the callback method(s) for the area. This is done by calling ***AL_SetAreaTextProperty*** with one of the callback properties and the name of the callback method. For example, to set up a callback that will execute when data entry is initiated, use the ALP_Area_CallbackMethEntryStart property:

   AL_SetAreaTextProperty (myALPArea;ALP_Area_CallbackMethEntryStart;"EntryCallback")

If you are using the Advanced Properties dialog to set up your area, you can specify the callbacks to use on the **Enterability** and **Advanced** pages.

Warnings

- Callback methods called during cell editing must not modify underlying data (arrays or records) – i.e. they must not resize or rebuild the arrays (array display) or change the current 4D selection (field display).
- You should not call any AreaList Pro commands which change the number of displayed columns, their position in the area, or their sorted order in a callback method.

Calculated Column Callback

A 4D callback may be attached to a specific column. When information is needed for this column, AreaList Pro will execute the callback to allow you to fill the column with data. This allows the displaying of data calculated from one or more fields or arrays as well as any ad hoc data that is desired.

Parameter	Description
$1	Reference of AreaList 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 AreaList Pro object.The last three parameters are absolutely necessary.
- The second parameter is the column number. This gives you the ability to use this callback method for many columns within a AreaList Pro object.
- The third parameter is the type of data in the column (field type or array type).
- In field mode, the fourth parameter is a pointer to one of the temporary 4D Arrays used internally by AreaList Pro. This is where you will load the data to be displayed 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.

Using Callback Methods During Data Entry

Two main callbacks are available to monitor data entry into a cell, when entering and exiting the cell.

In addition, the popup entry callback will run if there is a popup entry mode allowed but no popup array is defined.

In such case: entry callback is called, popup entry callback is called, entry exit is called.

If the popup array / map is defined, entry callback is called, the menu is shown using a dynamic popup menu (or the internal Date/Time "popup"), entry exit is called.

See Example 3: Using a Popup Callback to create dynamic popups and Data Entry Controls for popup entry details.

In addition to altering the array content, you can change color and style, reject or accept entered data, and change the current data entry cell using the AreaList Pro commands listed above.

You should not call any command which changes the number of displayed arrays, their position in the area, or their sorted order.

Executing a Callback Upon Entering a Cell

As described above, an "entry started" callback method is a 4D method called when data entry begins for a cell or an AreaList Pro popup menu is clicked, and is specified by passing the method name in the ALP_Area_CallbackMethEntryStart property.

If this property is not set then no method will be called.

Parameters

AreaList Pro will pass the callback method two parameters if arrays are being displayed, or three parameters if fields are displayed.

- the first parameter is a long integer that corresponds to the AreaList Pro object on the layout
- the second parameter is a long integer that reports what action (mode) caused data entry to begin in the cell
- the third parameter is a long integer that reports whether the record was loaded or not (this parameter only exists when fields are being displayed)

You must use the following declaration in your callback method:

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

Click action

The ALP_Area_EntryFirstClickMode property determines how the first click is handled upon beginning entry (when using numeric, date, time or text entry)

- 0 = the click is routed to the entry widget and the cursor is placed wherever the click occurs (default behavior)
- 1 = ignore click, select all when the value is NULL or the formatted value is empty string (numeric, date, time)
- 2 = ignore click, select all when the value is NOT NULL (numeric, date, time, text)
- 3 = ignore click, always select all (same behavior as when tabbing between the fields)

Note: explicit setting of the highlighted text in the Cell entered callback is always honored.

Entry mode

As stated above, the second parameter passed to the callback routine, the long integer $2, contains the mode by which data entry began, according to the following table:

Constant	Value	Entry mode
AL Click action	1	Click in Cell
AL Tab key action	2	Tab
AL Shift_Tab key action	3	Shift-Tab
AL Return key action	4	Return
AL Shift_Return key action	5	Shift-Return
AL GotoCell action	6	ALP_Area_EntryGotoCell and variants See Cell change properties
AL SkipCell action	9	ALP_Area_EntrySkip
AL Other cell popup action	10	Click on cell popup when cursor not already in cell
AL Active cell popup action	11	Click on cell popup when cursor already in cell

Popup menu entry

The entry callback is also executed whenever a popup menu is clicked, but before the menu is actually displayed.

When this occurs, the $2 parameter provided by AreaList Pro will be 10 if the popup was clicked on a cell other than the one actively in data entry. Mode 11 will be reported if data entry was already established in the cell for which the popup was clicked.

Field mode parameter

The third parameter only exists when fields are displayed, not arrays. If the value is 1, then the record was loaded properly and the field contents can be edited. If the third parameter is 0, then the record is locked by another process or user.

If typed data entry is underway and the record can not be loaded, then ALP_Area_EntryGotoCell or ALP_Area_EntrySkip may be used to continue data entry in another cell.

If neither of these properties is used then data entry will end. If popup data entry is underway and the record can not be loaded then data entry will end.

Executing a Callback Upon Leaving a Cell

As described above, an “entry finished” callback method is a 4D project method called when data entry ends for a cell, or when an AreaList Pro popup menu is released for a cell not in typed data entry.

The entry finished callback method is specified by passing the method name in the ALP_Area_CallbackMethEntryEnd property.

If this property is not set then no method will be called.

AreaList Pro will pass the callback method two parameters. The first parameter is a long integer that corresponds to the AreaList Pro object on the layout. The second parameter is a long integer that reports what action (mode) caused data entry to end in the cell.

You must use the following declarations in your entry finished callback method:

```
C_LONGINT($1)  // AreaList Pro object reference
C_LONGINT($2)  // exit cause
C_BOOLEAN($0)  // allow cell exit
```

The second parameter passed to the callback routine, the long integer $2, contains the mode by which data entry ended, according to the following table:

Constant	Value	Entry mode
AL Click action	1	Click in Cell
AL Tab key action	2	Tab
AL Shift_Tab key action	3	Shift-Tab
AL Return key action	4	Return
AL Shift_Return key action	5	Shift-Return
AL GotoCell action	6	ALP_Area_EntryGotoCell and variants See Cell change properties
AL ExitCell action	7	ALP_Area_EntryExit or “hard deselect”
AL Cell validate action	8	Deselect the cell (“soft deselect”)
AL Other cell popup action	10	Click on cell popup when cursor not already in cell
AL Active cell popup action	11	Click on cell popup when cursor already in cell

The callback method is actually a function. It must return **True** for the value entered into the cell to be accepted, and False for the value to be rejected. If the value is rejected the user will not be allowed to leave the cell.

When displaying arrays and data entry is initiated in a cell, the contents of the array element will be copied into the zero element of the array being displayed in the column. Please read the section “Undo” value for more information.

When fields are displayed, the contents of the field are not copied. Thus it is up to you to save the field contents in the entry started callback method if they will be needed for comparison in the entry finished callback method.

When displaying arrays and the entry finished callback method is executed, the array element corresponding to the cell has already been updated with the new value that was entered by the user. Thus, the zero element which contains the old data and the element representing the current cell can both be used to determine data validity.

Among the possible situations and responses that may occur are the following:

- The data is valid. Set $0:=**True** to complete data entry for the cell.

  The data is invalid. Copy the old data from the zero element to the array element corresponding to the cell. Set $0:=**True** to complete data entry for the cell.

For example:

```
$row:=AL_GetAreaLongProperty ($1;ALP_Area_EntryRow)  // edited cell row
aFname{$row}:=aFname{0}  // reset the cell contents to their original state
$0:=True
```

- The data is invalid. Inform the user that the data is invalid. Set $0:=**False** to force the user to remain in the cell and enter another value.
- The data is invalid. Inform the user that the data is invalid. Modify the cell contents, call ALP_Area_EntryGotoCell to go to the current cell, and set $0:=**True**. This achieves the same effect as rejecting the entry, but allows the cell contents to be modified.

For example:

```
$row:=AL_GetAreaLongProperty ($1;ALP_Area_EntryRow)  // edited cell row
$column:=AL_GetAreaLongProperty ($1;ALP_Area_EntryColumn)  // edited cell column
aFname{$row}:=aFname{0}  // reset the cell contents to their original state
AL_SetAreaLongProperty ($1; ALP_Area_EntryGotoRow; $row)  // go to the same cell
AL_SetAreaLongProperty ($1; ALP_Area_EntryGotoColumn; $column)
$0:=True
```

The AL ExitCell action (7) and AL Cell validate action (8) events depend upon the way the area is deselected.

- Soft deselect happens when the user selects a menu or clicks on a non-focusable object: the AreaList Pro area temporarily looses the focus.
- Hard deselect: clicking on a focusable object moves the focus to that object, the AreaList Pro area looses the focus.

Resizing the window (even using a splitter) will not end the entry and the exit callback method will receive $2=8.

Assuming that the entry finished callback allows the value, hard deselect will cause the callback (if any) to run with $2=7, in field mode the record is stored, whereas soft deselect will cause the callback (if any) to run with $2=8, in field mode the record is not stored.

In other words soft deselect only makes the area deselected temporarily, e.g. a click on a non-focusable checkbox will make it “active” (focus not drawn), its object method is executed, then the focus is returned to the AreaList Pro area. From the user’s point of view, soft deselect does not deselect the area.

Examples

Example 1

Let’s say that we do not want to allow the State to be modified if it’s “CA”. We would create an “entry start” callback method and initialise it in the On Load event for the AreaList Pro area:

```
AL_SetAreaTextProperty (myALPArea;ALP_Area_CallbackMethEntryStart;”EntryCallback”)
```

The ***EntryCallback*** method will handle the event when the user clicks into a cell:

```
C_LONGINT($1)  //AreaList Pro object reference
C_LONGINT($2)  //entry cause
C_LONGINT($3)  //only useful when fields are being displayed
C_LONGINT(vCurrCol;vCurrRow)
vCurrRow:=AL_GetAreaLongProperty ($1;ALP_Area_EntryRow)  //edited cell row
vCurrCol:=AL_GetAreaLongProperty ($1;ALP_Area_EntryColumn)  //edited cell column
ARRAY POINTER($ArrayNames;0)
$errorcode:=AL_GetObjects ($1;ALP_Object_Columns;$ArrayNames)
If (vCurrCol=1)  //city
   If ($ArrayNames{2}->{vCurrRow}=”CA”)  //pointer to second col array (state)
      AL_SetAreaLongProperty ($1;ALP_Area_EntrySkip;1)  //disallow data entry
   End if
End if
```

Example 2: Display a Tooltip

We want to display a Tooltip telling the user whether an area is draggable or droppable. First create a callback method:

```
//Event Callback method
C_LONGINT($1;$2;$3;$4;$5;$6)
$event:=$2
Case of
   : ($event=AL Mouse moved event)
      Case of
         : ($1=ProductList)
            AL_SetAreaTextProperty ($1;ALP_Area_ToolTip;"Drag from this list")
         : ($1=Selected)
            AL_SetAreaTextProperty ($1;ALP_Area_ToolTip;"Drop onto this list")
      End case
End case
```

Set this callback method in the On Load phase of the form method for both areas:

```
Case of
   : (Form event=On Load)
      AL_SetAreaTextProperty (ProductList;ALP_Area_CallbackMethOnEvent;"EventCallback")
      AL_SetAreaTextProperty (Selected;ALP_Area_CallbackMethOnEvent;"EventCallback")
End case
```

Note: you could also display a tooltip when the mouse is over a column header using $5 = 0 (row 0 is the header).

Example 3: Using a Popup Callback to create dynamic popups

You can use a Popup callback to dynamically change the contents of a column's popup menu. Suppose that we have various product types on offer, and each type of product comes in different pack sizes. We want to present to the user a "Pack size" popup whose contents depend upon the type of the selected product.

When the form is loaded we create some arrays and matching popup menu texts - one for each product type:

```
// pack sizes for chocolates
ARRAY TEXT(atChocsizes;3)
atChocsizes{1}:="4oz"
atChocsizes{2}:="8oz"
atChocsizes{3}:="16oz"
tPopChoc:="4oz;8oz;16oz"
// pack sizes for nuts
ARRAY TEXT(atNutsizes;3)
atNutsizes{1}:="6oz"
atNutsizes{2}:="12oz"
atNutsizes{3}:="18oz"
tPopNuts:="6oz;12oz;18oz"
```

Next we tell AreaList Pro that the Pack Size column (column no. 3 in this example) is enterable only by popup:

```
AL_SetColumnLongProperty (area;3;ALP_Column_Enterable;2)
```

Note: we do not attach a popup to that column!

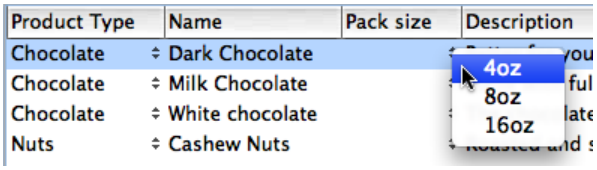
Then we assign a popup callback method to the area:

```
AL_SetAreaTextProperty (area;ALP_Area_CallbackMethPopup;"Alp_PopupCallback")  // empty popup callback
```

The callback method ***Alp_PopupCallback*** contains the following code:

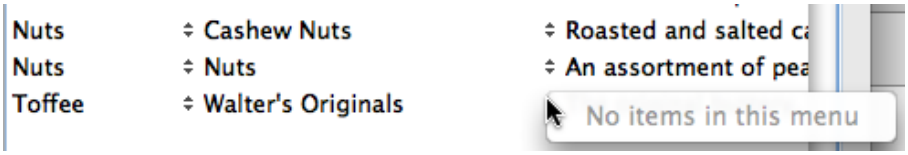
```
C_BOOLEAN($0)  // Return True if handled; False if not handled
C_LONGINT($1)  // AreaList Pro object reference
C_LONGINT($2)  // row
C_LONGINT($3)  // column
C_LONGINT($4)  // data type
$go:=True
Case of
   : ([product]product_type="chocolate")
      $pop:=tPopChoc
      $array:=->atChocsizes
   : ([product]product_type="nuts")
      $pop:=tPopNuts
      $array:=->atNutsizes
   Else
      $go:=False
End case
If($go)
   $choice:=Pop up menu($pop)
      If ($choice>0)
         AL_SetAreaTextProperty ($1;ALP_Area_EntryValue;$array->{$choice})
         $0:=True
      End if
End if
```

When the user clicks on the Pack Size popup icon he will see this popup if the product type is chocolate:



This one if it's Nuts:

… and this one if it's Toffee, because we forgot to set up the array for Toffee:



See also [Entering data in AreaList Pro with DisplayList](#).

9 Columns

This chapter presents various column-related topics, such as numbering / order, moving, widths, columns hiding and setting calculated columns.

Compatibility mode

Compatible mode on

When ALP_Area_Compatibility is set to 1:

- the visibility of columns (ALP_Column_Visible) is always reset before drawing and modified according to the number of hidden columns (ALP_Area_CompHideCols)
- the area is made visible on update event (ALP_Area_Visible)
- if only one column is to be displayed, it will have the width of the area
- columns are physically reordered on Drag
- in single row selection mode, the first row is selected during initialization, after a sort or on update

Compatible mode off

If you turn the compatibility mode off (setting ALP_Area_Compatibility to zero):

- any columns to be hidden have to be maintained
- the columns are not reordered - the order is defined by the grid setup
- you can hide any column, not only the columns at the end

Column numbers in compatible mode off

The columns are physically reordered (and renumbered) only in compatibility mode. When compatibility mode is off, columns are never moved - they remain in their creation order even if dragged by the user.

Modifying column display

For example, if you drag column 4 and drop it on the first column:

- the displayed column order will be 4, 1, 2, 3
- the internal order is not changed: 1, 2, 3, 4

When you apply any command to a column or a cell in this mode, use the original column number.

Therefore if you change the column order, either by dragging the columns in the interface or setting it using $err:=***AL_SetObjects*** (area;ALP_Object_Grid;$colOrderArray), then either hiding or showing any column, that AreaList Pro area will revert to the column order used to initialize the area.

If you want to hide/unhide a column and preserve the column order, manipulate the grid, not the visibility of a column.

ALP_Column_Visibility and ALP_Object_Grid are interdependent.

If you add a column, it has to be added to the grid. That's why the grid is cleared (and later re-created from visible columns).

Similarly, if you remove a column, it has to be removed from the grid.

If you make a column visible, it has to be added to the grid…

On the other hand, if you explicitly set the grid (using AL_SetObjects with ALP_Object_Grid), the visibility of columns is changed according to the new grid (only columns in the grid will be set to visible, all others will be set to invisible).

Further to, to restore the previous user state you can use ***AL_Save*** to save the settings and ***AL_Load*** to restore them.

Using Object Property commands

AreaList Pro Object Property commands always return information based upon the original definition, as opposed to the "grid order" arrangement.

The following properties always use the original (developer defined) column order:

ALP_Object_Source

ALP_Object_ColumnWidth

ALP_Object_HeaderText

ALP_Object_FooterText

For example, if you want to determine the data source of all columns (based upon how the user has rearranged the columns), the ALP_Object_Source property will always return the "original" order.

You need to use the results from ALP_Object_Grid to determine the user rearrangement.

The same applies to the column widths, for example (see below).

Bottom line: when the user drags a column to a different place, only the grid is changed, not the "physical" column order.

Procedurally moving columns

You can procedurally move columns, similar to the way the user does it via the interface (i.e. drag columns to reorder them).

For a simple grid (just one row of columns - as in AreaList Pro previous versions):

```
ARRAY INTEGER($columns;0)
$err:=AL_GetObjects ($area;ALP_Object_Grid;$columns)  // get current column order
// reorder $columns as you want
$err:=AL_SetObjects ($area;ALP_Object_Grid;$columns)  // set new column order
```

Note: since AreaList Pro v9, compatibility mode off, columns are never moved physically, only the display order is changed.

For example after column number 2 is moved before column 1, when the first displayed column is clicked, column number 2 is returned.

Column widths

User auto-size

When resize is allowed for an area (ALP_Area_ColumnResize = 1) the user can auto-size a column that is not already programmatically auto-sized (ALP_Area_AutoResizeColumn / ALP_Area_AutoSnapLastColumn): a double-click on the right column edge will calculate its width from the data displayed in this column and update the display. This action sets the column's ALP_Object_ColumnWidthUser property to zero.

Option (alt)-double-click on any (non programmatically auto-sized) right column edge resizes all the area's columns at once.

Note: if there is less than 10 points available for an auto-sized column (area width minus sum of all other column widths), it is **not** auto-sized, the user width is used (and the area will be horizontally scrollable).

Properties

The user will be able to resize columns if ALP_Area_ColumnResize is set to yes.

Column widths are accessed through the ALP_Object_ColumnWidth / ALP_Column_Width and ALP_Object_ColumnWidthUser / ALP_Column_WidthUser properties.

ALP_Object_XXX is a "batch" accessor to a property or a multi-valued property.

ALP_Object_ColumnWidth and ALP_Object_ColumnWidthUser are accessors to ALP_Column_Width and ALP_Object_ColumnWidthUser.

For example, instead of looping through all columns and asking for ALP_Column_Width using ***AL_GetColumnLongProperty***, you declare an array and call ***AL_GetObject*** with ALP_Object_ColumnWidth to fill it.

ALP_Object_ColumnWidthUser is the width of a column, zero means calculate it from the data displayed by this column (auto-size).

ALP_Object_ColumnWidth is the current width of a column. It is either equal to ALP_Object_ColumnWidthUser or calculated from the data if ALP_Object_ColumnWidthUser is zero (auto-size).

ALP_Object_ColumnWidthUser originally has the value specified by the developer. If the user resizes a column, it becomes the user-specified value, which can be zero (the user double-clicked the separator in the column header).

In other words, these two properties always have the same value, except for auto-size, where ALP_Column_WidthUser is zero and ALP_Column_Width is the actual width computed from the data.

You can use both ALP_Column_WidthUser and ALP_Column_Width in the setter, which will set both properties and zero will trigger column width recalculation: then ALP_Column_Width will be set and ALP_Column_WidthUser will be zero.

Saving original settings

You may want to revert back to the original column widths in case the user does not like their “adjustements”.

However, is it not possible to get the original width values, as were passed to AreaList Pro, after the user has adjusted the columns widths.

You have to get the widths after you initialize the AreaList Pro area and before the user is able to make changes (i.e. in the On Load phase):

```
ARRAY REAL (alSavedWidths; 0) // note that we are using a real type array
$err:=AL_GetObjects (area; ALP_Object_ColumnWidthUser; alSavedWidths)
```

Then you can eventually reset all widths to the original settings:

```
$err:=AL_SetObjects (area; ALP_Object_ColumnWidthUser; alSavedWidths)
```

If the advanced properties are used, you can also access their original settings:

```
$xmlAP:=AL_GetAreaTextProperty (area; ALP_Area_XMLAP)
```

and reset the whole AreaList Pro area:

```
$err:=AL_Load (area; $xmlAP)
```

This will, of course, reset everything, not only column widths!

Column wider than the visible area

If you leave the width setting to 0, the column that holds the text may be wider than the visible area. This is a feature

You can limit it to the visible width when you switch horizontal scrolling to “columns” mode:

```
AL_SetAreaLongProperty ($eList;ALP_Area_ScrollColumns;1)
```

Displaying column widths

Previous versions of AreaList Pro (prior to Version 9) used to display column widths in the headers when clicking on a X button located at the area’s lower right corner.

AreaList Pro now does more than this. It provides the column width, its number and its data source. The information is displayed in a tooltip whenever the mouse is over a header or any cell and the three main modifier keys are pressed (command-option-shift on MacOS, ctrl-alt-shift on Windows).

This behavior is triggered by the ALP_Area_ShowWidths area property, which you can set for example according to the user name:

```
AL_SetAreaLongProperty ($eList;ALP_Area_ShowWidths;Num(<>userName=”Administrator”))
```

Value 1 means “display in interpreted and compiled modes”; value 2 means “display in interpreted mode only”; value 3 means “always (compiled & uncompiled)”.

Hiding columns

Hidden columns

ALP_Area_CompHideCols is used to hide the last x columns (x being the property value) or know how many columns are currently hidden in the area, only in compatibility mode (ALP_Area_Compatibility=1).

If your last x columns are hidden, the value returned by ***AL_GetAreaLongProperty*** ($area; ALP_Area_CompHideCols) will be zero in compatibility mode off (ALP_Area_Compatibility = 0).

This property is simply unused in such case, but it can be nevertheless set to any positive integer value (or zero) and it will preserve the value so that you can get it later.

In this case ALP_Area_CompHideCols will return a value even though compatibility mode is off. When you turn compatibility on, it will be used to modify the visibility of columns (the grid is not cleared when you modify the value of ALP_Area_Compatibility) and will return their count.

Note: the grid is not cleared either when you modify ALP_Area_CompHideCols in compatible mode on (ALP_Area_Compatibility = 1).

Number of hidden columns

There is no single accessor to set or get the number of hidden columns in compatibility mode off (ALP_Area_Compatibility = 0).

The simplest way is to combine ALP_Area_Columns and ALP_Object_HeaderTextNH:

```
$count:=AL_GetAreaLongProperty ($area; ALP_Area_Columns)  // number of columns
ARRAY TEXT($headers;0)
$err:=AL_GetObjects ($area; ALP_Object_HeaderTextNH; $headers)
// header text for visible columns (NH stands for Not Hidden)
$count:= $count - Size of array ($headers)  // number of hidden columns (total minus not hidden)
```

Note: since AreaList Pro v9 (only in compatibility mode off), you can hide any column, not only the columns at the end.

Grid clearing

In both compatibility modes (on or off) the grid is lost (cleared) when:

- a column is added
- a column is removed
- a column's visibility is changed
- ALP_Area_RowsInGrid is set (does not have to be changed)
- ALP_Area_ColsInGrid is set (does not have to be changed)
- ***AL_SetColOpts*** is called with a different 5th argument (columns to hide) - note that this call of a v8.x command will turn compatibility mode on

Once the grid has been cleared or if it has not been not defined, it is (re-)created from visible columns.

Calculated columns

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

This feature is available for both field and array modes.

Setting a Calculated Column (field mode)

The AL_AddCalculatedColumn command is used to set up calculated columns in field mode.

The following table shows the data types that may be displayed 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 display a calculated column of type Real, pass Is Real (1) in the **dataType** parameter and the Calculated Column Callback in the **callbackMethodName** parameter.

Setting a Calculated Column (array mode)

The ALP_Column_Calculated property is used to set up calculated columns in array mode.

This property can only be set in this mode.

To make an column calculated, use:

AL_SetColumnLongProperty (area; column; ALP_Column_Calculated; 1)

AL_SetColumnTextProperty (area; column; ALP_Column_Callback; 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. Integer Array instead of Is Integer)

The following table shows the data types that may be displayed 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
Time array (v14)	32

Setting the Callback Method

In field mode, use the **callbackMethodName** parameter in AL_AddCalculatedColumn to set the Calculated Column Callback for a column. The ALP_Column_Callback property can later be used to modify the callback method name on the fly.

In array mode, directly use the ALP_Column_Callback property.

In field mode, AreaList 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 (AreaList 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 displayed.

These arrays have been initially declared and included in the AreaList Pro area with either AL_AddColumn or AL_SetObjects with the ALP_Object_Columns property for both non-calculated arrays and calculated arrays:

```
// Declare the arrays
ARRAY TEXT (aName;0)
ARRAY DATE (aHireDate;0)  // not displayed, 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 display
$error:= AL_AddColumn(eList;->aName;0)  // no need to specify colum number
$error:= AL_AddColumn(eList;->aAnniversary;0)
// Set calculated status and callback method for column 2
AL_SetColumnLongProperty (eList;2; ALP_Column_Calculated; 1)
AL_SetColumnTextProperty (eList;2; ALP_Column_Callback; "CalcColCallbackArray")
```

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

```
// CalcColCallbackArray
// $1: Area reference (AreaList Pro longint reference)
// $2: Column number
// $3: Type of array in this column
// $4: Pointer to the displayed 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
```

Sorting a Calculated Column (field mode)

The ALP_Column_Sortable property sets a method to be called in order to perform a sort.

Set this property to a method name which will be called by **ORDER BY FORMULA** as *Method* ($alp:int; $colNum:int) -> value to use for order

Note that $colNum is negative for descending order - **ORDER BY FORMULA** explicitly uses descending order, you should not revert it

To be usable by end user (click on header or Sort Editor), you must explicitly set ALP_Column_Sortable to 1 (calculated columns are not sortable by default).

Example:

```
$err:=AL_AddCalculatedColumn($area; $colType; “MyColumnCallback”)
AL_SetColumnTextProperty($area; $col; ALP_Column_SortFormula; “MyColumnSortCallback”)
AL_SetColumnLongProperty($area; $col; ALP_Column_Sortable; 1)

 // MyColumnSortCallback
C_LONGINT($1)  // ALP
C_LONGINT($2)  // column number, negative when descending order will be used (using explicit “; <”)
 // C_xxxxx($0)
$0:=MyCalculatedValueForCurrentRow.call($1; Abs ($2))
```

Column dividers

Column dividers display can be fine-tuned with the ALP_Area_ShowColDividers bit-field property:

Bit number	Description
0	Draw over data and footer
1	Hide footer dividers (bit 0 ignored if bits 0 and 2 off)
2	Draw last column divider

Note: value 2 should logically be “no divider” as value 0 but AreaList Pro ignores bit 0 for compatibility in this case.

Possible values

Bit 0 draw dividers, not last col	Bit 1 hide all footer dividers	Bit 2 draw last col divider	Value	Behaviour
0	0	0	0	No divider
1	0	0	1	Draw dividers, including footers, not the last one to the right
0	1	0	2	Draw dividers except footers, not the last one to the right (bit 0 ignored)
1	1	0	3	Draw dividers except footers, not the last one to the right
0	0	1	4	Draw only the last divider to the right including footer
1	0	1	5	Draw dividers, including footers, including the last one to the right
0	1	1	6	Draw only the last divider to the right except footer
1	1	1	7	Draw dividers except footers, including the last one to the right

Examples

AL_SetAreaLongProperty ($area; ALP_Area_ShowColDividers;0) // No divider
AL_SetAreaLongProperty ($area; ALP_Area_ShowColDividers;1) // Draw dividers, including footers, not the last one to the right
AL_SetAreaLongProperty ($area; ALP_Area_ShowColDividers;2) // Draw dividers except footers, not the last one to the right (bit 0 ignored)
AL_SetAreaLongProperty ($area; ALP_Area_ShowColDividers;3) // Draw dividers except footers, not the last one to the right
AL_SetAreaLongProperty ($area; ALP_Area_ShowColDividers;4) // Draw only the last divider to the right including footer
AL_SetAreaLongProperty ($area; ALP_Area_ShowColDividers;5) // Draw dividers, including footers, including the last one to the right
AL_SetAreaLongProperty ($area; ALP_Area_ShowColDividers;6) // Draw only the last divider to the right except footer
AL_SetAreaLongProperty ($area; ALP_Area_ShowColDividers;7) // Draw dividers except footers, including the last one to the right

Colors

Column divider colors are set with ALP_Area_ColDivColor.

Hiding the last colum divider separator

In addition, the ALP_Area_HideLastHdrDivider property allows hiding the last colum divider separator (on the right).

10 Working with Colors

You can use colors in your AreaList Pro areas in various ways: to color text, backgrounds, calendar elements, and so on.

Specifying Colors

Internally, all colors since AreaList Pro v9 use ARGB (alpha-red-green-blue, each channel using 8 bits:0-255/0x00-0xFF).

You can use the alpha channel to specify transparency. The value should be between 0 - 255 (0x00 - 0xFF).Transparency of 0 means fully transparent (invisible) color; transparency of 255 (0xFF) means fully opaque color.

However, there are seven ways that you can specify colors in AreaList Pro. Where necessary, they will be converted to the ARGB model. The seven methods can be split into two groups: color values passed as string values, and color values passed as longint values.

Color values passed as string values

1. Using one of the standard color names (red, green, blue, dark red, dark blue, white, gray, light gray, cyan, magenta, yellow, brown, orange, dark orange, purple, black). In this case, you pass the color name using one of the text commands (e.g. ***AL_SetCellTextProperty***).

   For all above values, the alpha is always 100%. You can also use “transparent”, which will set the alpha channel to 0%.

2. Using standard hexadecimal notion with one of the text commands.

   e.g. “0xFFFF0000” is 100% red

3. Using hexadecimal ARGB (alpha-red-green-blue) notation. In this format, a leading # is used, followed by two hexa numbers per channel; if less than four channels are specified, full alpha (0xFF) is assumed. Note that this is the format used internally by AreaList Pro.

   e.g. “#FF0000” is the same as “#FFFF0000” = 100% red

4. 3- or 4-part RGBA comma-separated real type channel values can be used with one of the text commands. Channel values have to be in range 0.0 - 1.0; if three values are specified, alpha is assumed to be 1.0. This is simply the percentage for each color (and alpha for transparency). Note that in this case alpha is at the end. This format conversion is triggered by any “.” in the value.

   e.g. “1.0,0,0” is the same as “1.0,0,0,1.0” = 100% red

5. 3- or 4-part RGBA comma-separated long integer type channel values can also be used with one of the text commands. Channel values have to be in the range 0 - 65535; if three values are specified, alpha is assumed to be 65535. Note that in this case alpha is at the end.

   e.g. “65535,0,0” is the same as “65535,0,0,65535” = 100% red

6. Using the “good old” 4D 256 color palette. Any 4D 256 color palette can be specified as “Pxxx” where xxx is the palette index in range 1 – 256. For example:

   AL_SetCellTextProperty ($area;$row;$col;ALP_Cell_FillColor;”P2”) //set the fill color to yellow

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**.

Color passed in longint values

7. Using a long integer with a longint command (e.g. ***AL_SetCellLongProperty***). In this case, nothing is assumed about the alpha channel and the alpha value needs to be specified. The color can be conveniently written in hexa notation like 0xAARRGGBB; for example 0xFF00FF00 is 100% green. However, this number in decimal notation is -16711936.

Note that the color picker and 4D RGB commands use longint values for color without the alpha channel. This means that the developer must add alpha channel information to the color if he is going to pass a color to AreaList Pro by code - for example:

```
$ALPColor:=$Color | 0xFF000000
```

Color Options

You can specify colors for the following elements in your AreaList Pro areas:

Area properties

Use these properties with commands in the Area theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Area_AltRowColor	✔	✔	✔	color	#FFEEEEEE			Alternate row color (default is light gray)
ALP_Area_AltRowOptions	✔	✔	✔	long int	0	0	15	Alternate row coloring options: bit 0: 1 = enable, 0 = disable bit 1: 1 = apply ALP_Area_AltRowColor to even rows, 0 = apply to odd rows bit 2: 1 = alt color applies to empty space below the last row (if any) bit 3: 0 (default) = use the existing color when defined at cell or row level, instead of alternate color for alternate rows (column color is ignored) 1 = mix the alternate color with the existing color set for the cell/row/column (in this order)
ALP_Area_ColDivColor	✔	✔	✔	color	#FF808080			Column divider color (default is gray)
ALP_Area_MiscColor2	✔	✔	✔	color	#FFEEEEEE			Area color below the vertical scrollbar MODIFIED: this area is not customizable since AreaList Pro v9 (the scrollbar is drawn, and it is bigger than in 8.x) Since AreaList Pro v9, this color is used as the background color: before drawing anything, the whole AreaList Pro area is erased using this color Default is light gray
ALP_Area_MiscColor3	✔	✔	✔	color	#FFEEEEEE			Area color left of the horizontal scrollbar Default is light gray
ALP_Area_MiscColor4	✔	✔	✔	color	#FFEEEEEE			Area color right of the horizontal scrollbar Default is light gray
ALP_Area_RowDivColor	✔	✔	✔	color	#FF808080			Row divider color (default is gray)

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Area_CalendarColors	✔	✔		text				8 colors separated with "|" to be used by the date «calendar» popup First 5 colors define object backgrounds: active month, inactive month, selected date, current date, current selected date Next 2 colors define foreground: numbers in active month, numbers in inactive month 8th parameter is the popup background color; it needs a non-zero alpha channel to be set, e.g. #FFE9F1FF instead of #E9F1FF When not set explicitly, default colors depend on ALP_Area_CalendarLook To restore the default colors (as if ALP_Area_CalendarColors was not set), pass an empty text value Default values are: "#00FFFFDD|#00EEEEEE|#00EEAAAA|#00FF8888|#00FF5555|#00000000|#00444444|#00CCCCCC" for the regular (default) calendar look, and: "#FFFFFFFE|#FFFFFFFE|#00EEEEEE|#00FF8888|#008F8F8F|#00000000|#00444444|#FFFFFFFC" for the alternate Date popup (according to ALP_Area_CalendarLook)

Column Properties

Use these properties with commands in the Columns theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Column_HdrTextColor	✔	✔	✔	color	#FF000000			Font color Default is black
ALP_Column_FtrBackColor	✔	✔	✔	color	#00FFFFFF			Background color Default is transparent (no color)
ALP_Column_FtrTextColor	✔	✔	✔	color	#FF000000			Font color Default is black
ALP_Column_BackColor	✔	✔	✔	color	#00FFFFFF			Background color Default is transparent (no color)
ALP_Column_TextColor	✔	✔	✔	color	#FF000000			Font color Default is black

Row Properties

Use these properties with commands in the Rows theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Row_BackColor	✔	✔	✔	color				Background color
ALP_Row_TextColor	✔	✔	✔	color				Font color

Cell Properties

Use these properties with commands in the Cells theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Cell_BottomBorderColor	✔	✔	✔	color				Bottom border color
ALP_Cell_FillColor	✔	✔	✔	color				Color used to fill the border rectangle
ALP_Cell_LeftBorderColor	✔	✔	✔	color				Left border color
ALP_Cell_RightBorderColor	✔	✔	✔	color				Right border color
ALP_Cell_TopBorderColor	✔	✔	✔	color				Top border color
ALP_Cell_BackColor	✔	✔	✔	color				Background color
ALP_Cell_TextColor	✔	✔	✔	color				Font color

Converting RGB values

AreaList Pro colors are very close to the format used by 4D.

In 4D, RGB colors are long integers interpreted as 0x00RRGGBB, so there are 3 channels each in range 0 - 255.

AreaList Pro uses ARGB - 0xAARRGGBB - 4 channels each in range 0 - 255.

For example, let's examine the following AreaList Pro v8.5 command:

```
AL_SetRowRGBColor ($area;$i;-1;-1;-1;<>greenbar_red;<>greenbar_green;<>greenbar_blue)
```

Where:

- <>greenbar_red:=244
- <>greenbar_green:=248
- <>greenbar_blue:=255

Let's combine them using simple math:

```
$argb:=0xFF000000 | (<>greenbar_red << 16) | (<>greenbar_green << 8) | <>greenbar_blue
```

We get 0xFFF4F8FF.

Generally, to create ARGB color for use with AreaList Pro, use

```
$argb:=($alpha << 24) | ($red << 16) | ($green << 8) | $blue
```

which is the same as

```
$argb:=($alpha * 256 * 256 * 256) + ($red * 256 * 256) + ($green * 256) + $blue
```

To create RGB color for use with 4D, use

```
$rgb:=($red << 16) | ($green << 8) | $blue
```

which is the same as

```
$rgb:=($red * 256 * 256) + ($green * 256) + $blue
```

Row Coloring Options

Combining bits in the Row Options property

Bits 0, 1 and 2 are used in combination to manage all possible alternate color row settings.

The resulting long integer sets the alternate row coloring ("zebra" style) options. Here are the possible values (this is for bit 3 = 0, add 8 to the values below to set bit 3 to true and combine alternate color with existing colors, see below, and also ALP_Area_AltRowOptions):

- 0, 2, 4, 6 - don't use alternate row coloring (bit 0 = 0)
- 1 - use alternate coloring for even rows (bit 0 = 1)
- 3 - use alternate coloring for odd rows (bits 0 and 1 = 1)
- 5 - use alternate coloring for even rows including empty space below last data row (bits 0 and 2 = 1)
- 7 - use alternate coloring for odd rows including empty space below last data row (bits 0, 1 and 2 = 1)

The "empty space below last data row" refers to the area between the last row and the footer / horizontal scrollbar / bottom of the AreaList Pro area, where a click or a rollover reports -2 using ALP_Area_ClickedRow or ALP_Area_RollOverRow.

See Row Numbering.

Combining Alt Row color with Background color

When bit 3 is set to yes in ALP_Area_AltRowOptions, the cell's background color is first set as defined (from cell, row or column settings in that order), then combined with the alternate row color in case this other color is defined. See ALP_Area_AltRowOptions.

The result for alternate rows will be a blend of both specific and alternate colors.

1st Quarter	347 966 €	-3 543 €	197 898 €	-45 334 €	496 987 €
2nd Quarter	350 976 €	23 690 €	-1 554 €	-19 772 €	353 340 €
3rd Quarter	429 750 €	76 449 €	98 620 €	34 875 €	639 694 €
4th Quarter	510 990 €	96 877 €	165 890 €	45 350 €	819 107 €

Empty rows

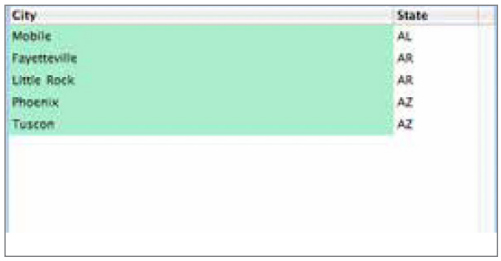
The background column color will always apply to data rows as well as any visible empty rows at the bottom of the area.

This code sets a color for column 1:

AL_SetColumnLongProperty (eList;1;ALP_Column_BackColor;0xFFAAEECC)

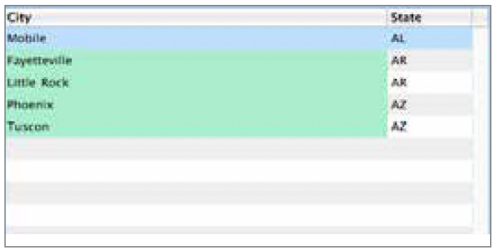
To apply the color only to data rows and leave the bottom part blank, we can use the ALP_Cell_BackColor cell property, and -2 as the value for the **row** parameter of AL_SetCellLongProperty, meaning “all rows”:

AL_SetCellLongProperty (eList;-2;1;ALP_Cell_BackColor;0xFFAAEECC)



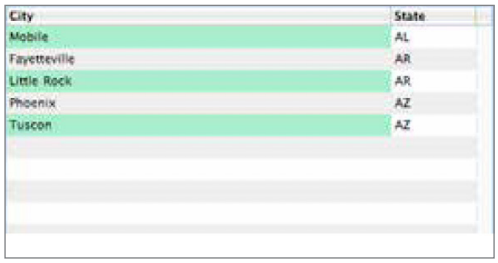
Alternate row coloring will apply to the “empty” bottom section when bit 2 of ALP_Area_AltRowOptions is set to yes:

AL_SetAreaLongProperty (eList;ALP_Area_AltRowOptions;5)



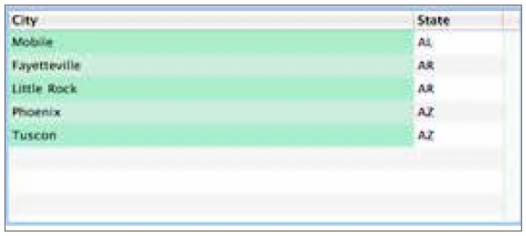
To apply alternate row coloring to the data rows as well, use 13 instead of 5 for ALP_Area_AltRowOptions (bit 3 means “blend the cell color with alternate row color”):

AL_SetAreaLongProperty (eList;ALP_Area_AltRowOptions;13)



However, but for that option to produce any visual effect, the alternate row color must be partially transparent, e.g. 0x80EEEEEE instead of 0xFFEEEEEE:

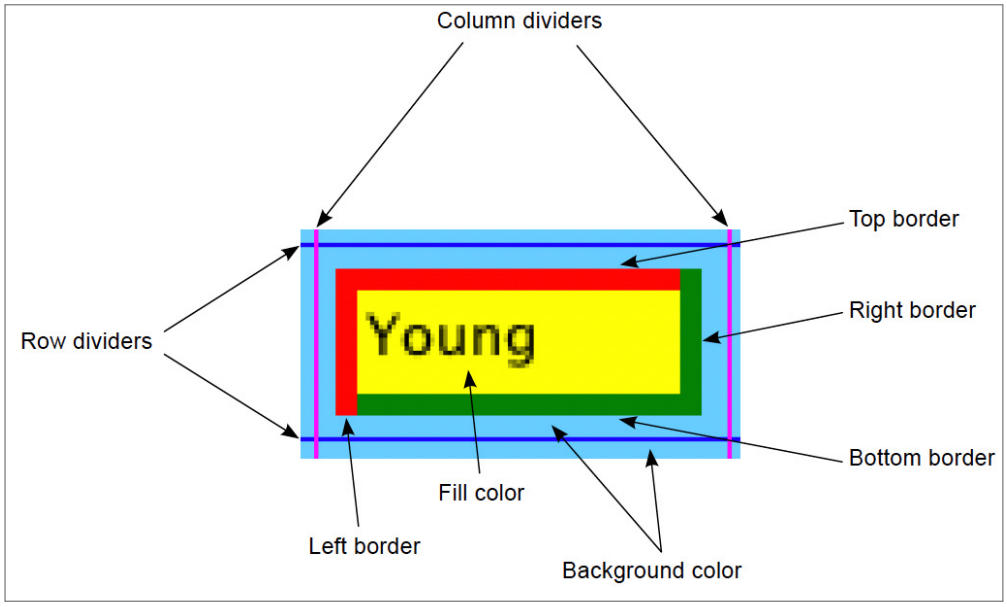
AL_SetAreaLongProperty (eList;ALP_Area_AltRowColor;0x80EEEEEE)



Coloring Cell Sections

Summary

Here are the various parts of the cell, which can be individually set / colored:

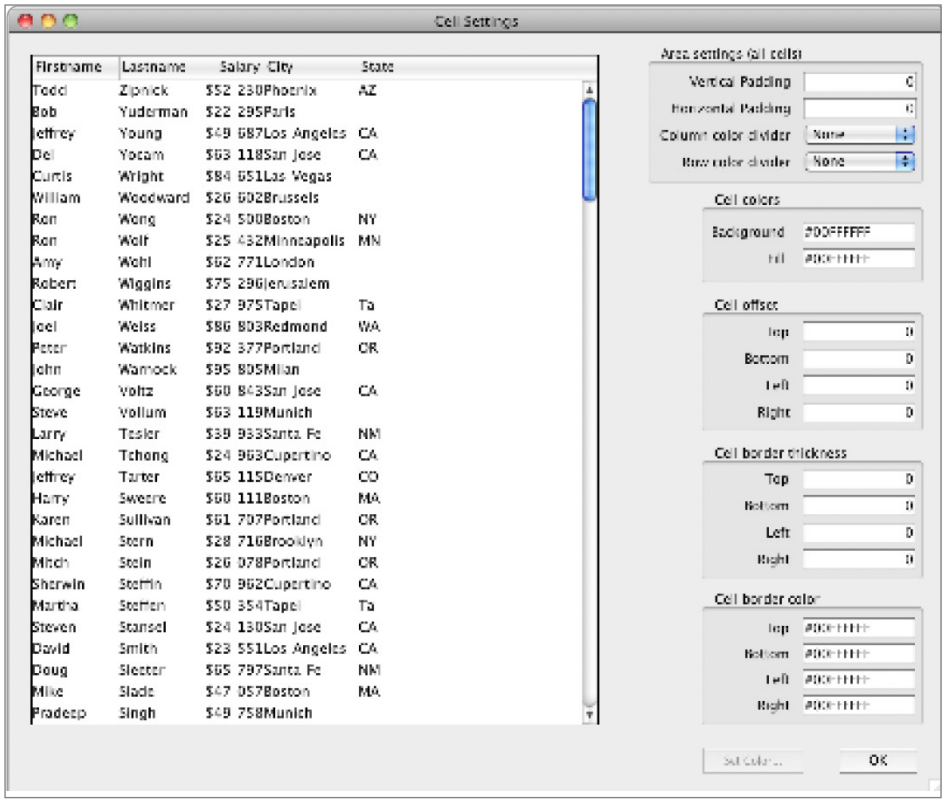


Example

To illustrate the various cell sections that can be set we'll use the AreaList Pro demonstration database (AreaList > Configuration Options then Format > Cell Settings).

Getting started

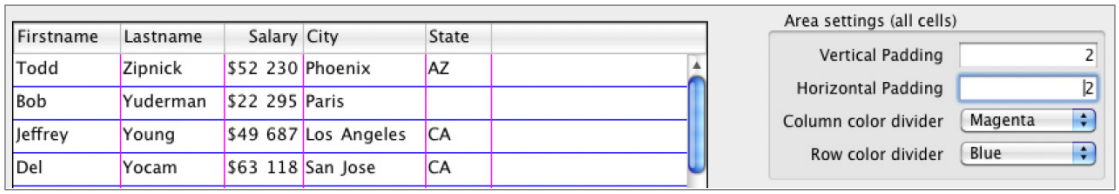
Let's start with no padding, no offset, no borders, no colors:



Padding and Dividers

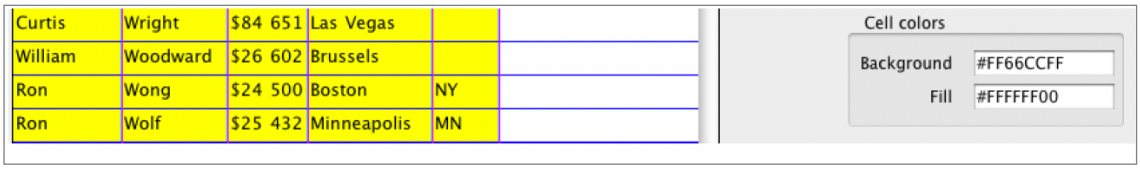
The area will look more legible with some padding to move the text away from the cell borders.

We also set Column Dividers to magenta and Row Dividers to blue:



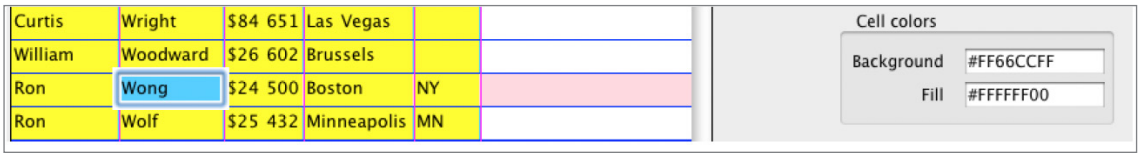
Background and Fill

Now we set a light blue Background and yellow Fill, but we only see the Fill since there are not Cell Offsets yet:



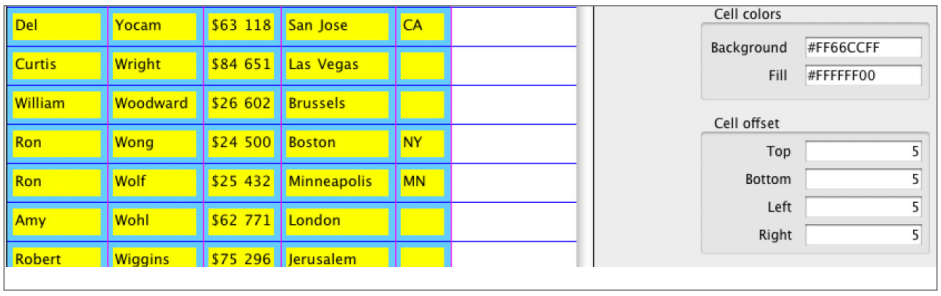
Text Editing

We will see our blue Background if we enter a cell for text editing:



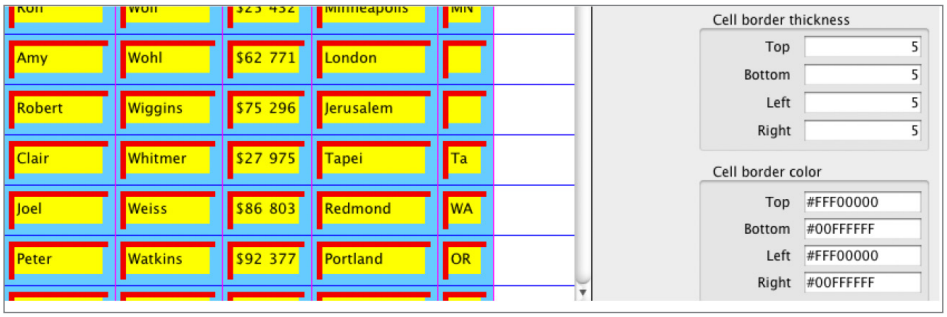
Cell Offsets

Setting Cell Offsets will reveal our Background color:



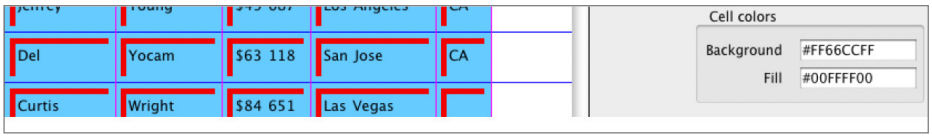
Borders

Let's set our top and left Borders to red, 5 points thick. Note that right and bottom borders are transparent here, since we set their thickness but no color:



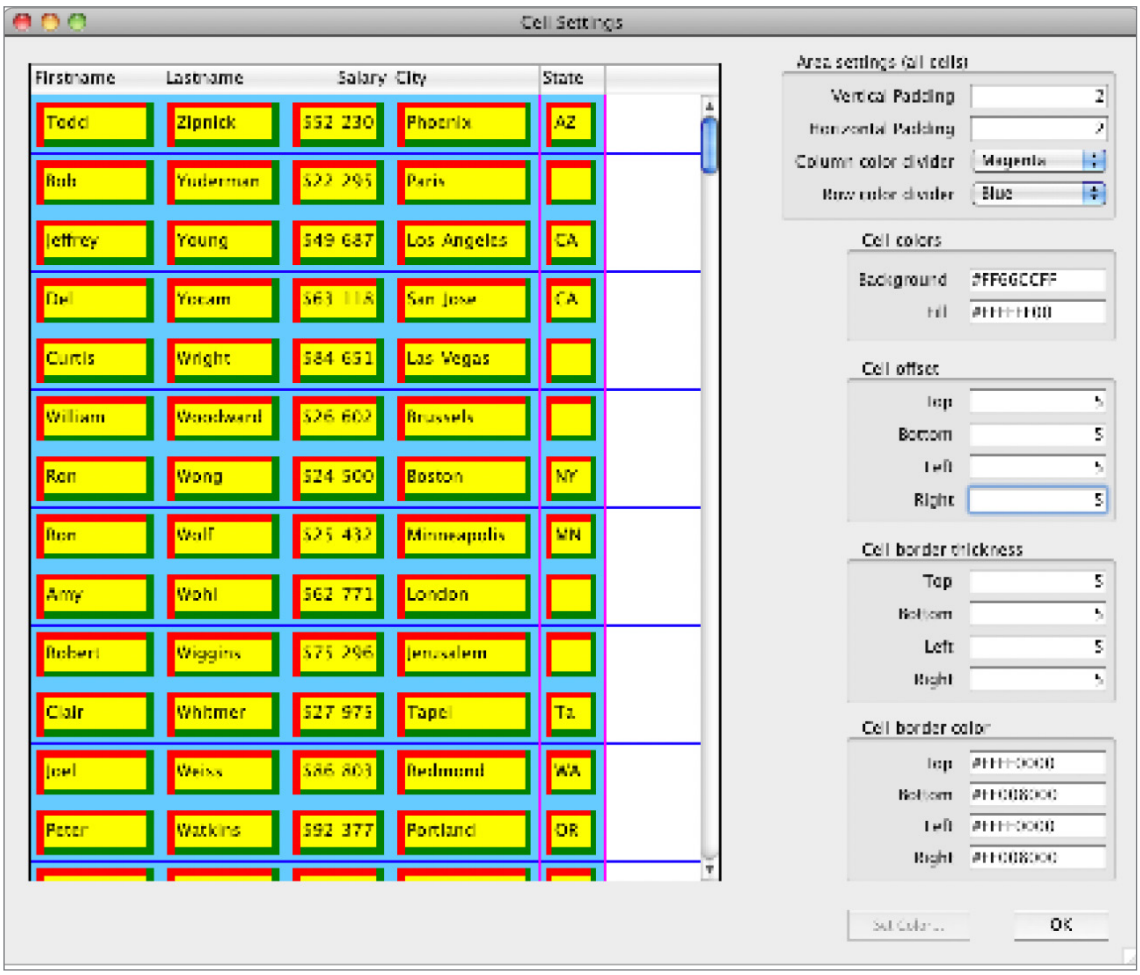
Transparent Fill

AreaList Pro v9 provides full transparency (alpha channel) support. Setting the Fill color to totally transparent (beginning with #00 instead of #FF = 100 % opaque) would reveal our Background color behind the whole cell:



Final Result

Let's revert to our opaque yellow fill and add green right and bottom borders:



Attractive, isn't it?

Custom row highlight

You can manage the highlight of the selected rows from AreaList Pro yourself, beyond what the system offers.

For example, modify the foreground, style, font and background colors of a row when you click on an row in AreaList Pro:

```
// disable row hightlighting on form/object load event
AL_SetAreaLongProperty ($area;ALP_Area_SelNoHighlight;1)
```

On click event, loop on each selected row:

```
AL_GetObjects ($area;ALP_Object_Selection;$selectedRows):
```

Backup selected rows numbers, style, font and colors, then apply the following:

```
AL_SetRowLongProperty($area;$rowNum;ALP_Row_StyleF;$styleNum)  // style
AL_SetRowTextProperty ($area;$rowNum;ALP_Row_FontName;$fontName)  // font
AL_SetRowTextProperty ($area;$rowNum;ALP_Row_TextColor;$foreGroundColor)  // foreground color
AL_SetRowTextProperty ($area;$rowNum;ALP_Row_BackColor;$backGroundColor)  // background color
```

Since v11.3, the system highlight color can be read or overriden with ALP_Area_HighlightColor.

When set, this color will be used in place of the system highlight color. To reset it to use the system color again, set the value to 0.

Note: this is a global property, it will affect all areas.

Empty column background color

When the total width of all columns is lesser than the area's display width, and neither ALP_Area_AutoResizeColumn nor ALP_Area_AutoSnapLastColumn properties are used, an empty "column" will fill the remaining space on the right. Its background color will be inherited from the last visible column's property.

Setting the entire area to a single color

Here's how to set the entire AreaList Pro area to a single color (including empty rows):

- Set ALP_Area_MiscColor2 (background), ALP_Area_MiscColor3 (space to the left of horizontal scrollbar when columns are locked) and ALP_Area_MiscColor4 (space under vertical scrollbar when both scrollbars are shown) to the color.
- Leave the column list background (ALP_Column_BackColor) at default (#00FFFFFF = transparent), otherwise the whole column (including empty rows) will use that column color (if not overridden by row/cell color or alternate row coloring).

Patterns

Patterns are no longer supported. Since version 9 they are interpreted by AreaList Pro as transparency ratios (alpha channel value):

- "black" or 1: 100% (0xFF)
- "darkgray" or 4: 75% (0xC0)
- "gray" or 2: 50% (0x80)

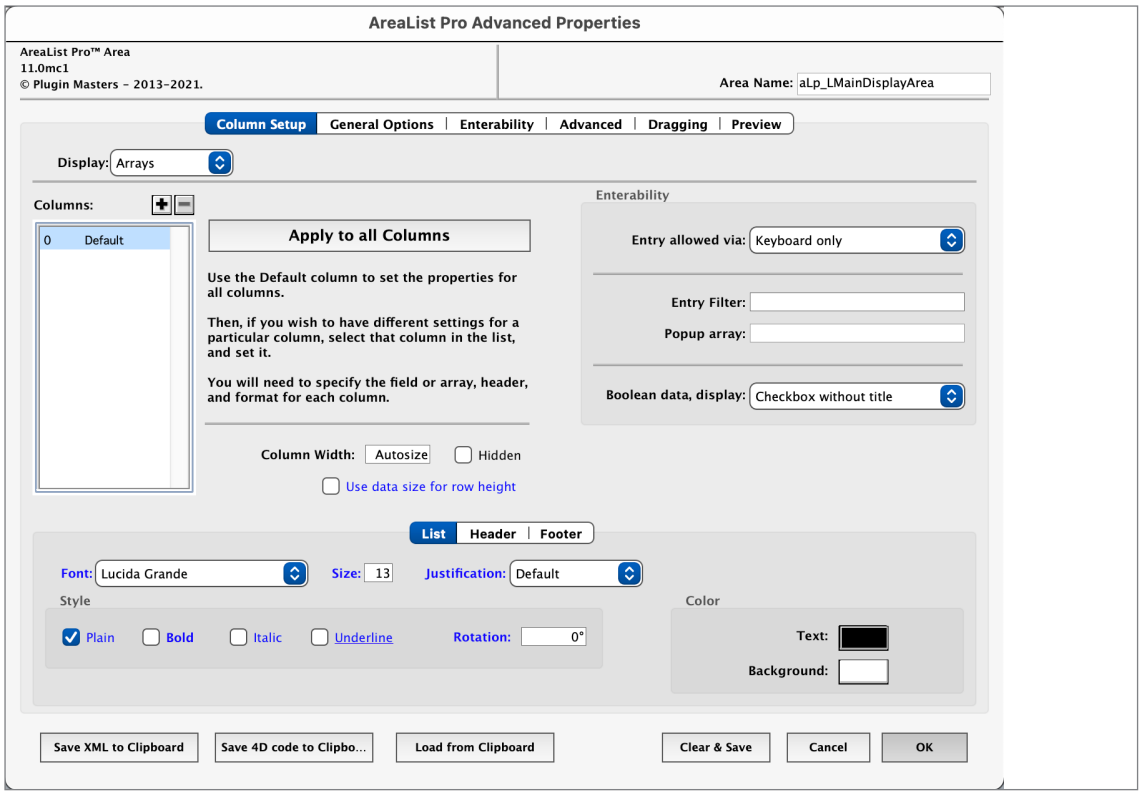
- “lightgray” or 3: 25% (0x40)
- “white” or 0 or “none” or “” or anything else: 0% = no drawing (0x00)

11 The Advanced Properties Dialog

The Advanced Properties Dialog

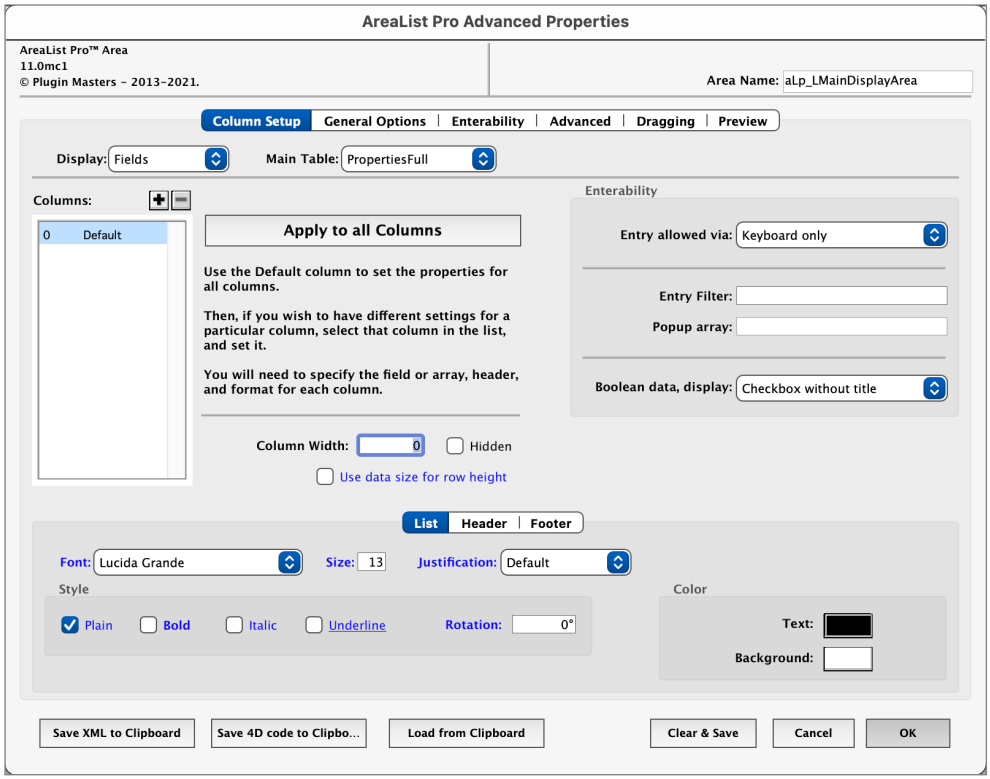
The Advanced Properties Dialog allows you to configure most aspects of an AreaList Pro area without having to do any programming. To use this option:

1. Create a new AreaList Pro area on your form
2. Click on the **Advanced Properties** button next to **Advanced Properties** in the object's Property List window.
3. The Advanced Properties window opens:



:

Column Setup Tab



Default Column

You can use the default column to set up the attributes for new columns you include by clicking the **Add** button. New columns that are added are assigned the settings in the default column.

This behaviour is true at any time, not just the first time that the Advanced Properties dialog is configured. If you change the settings for the default column, any new columns you add will get the new default settings, but existing columns will not be changed. To apply the changes to existing columns, click the **Apply to all Columns** button.

Apply to all Columns

Note that most of the objects on this page have their labels shown in **blue** when the defaut column is selected.

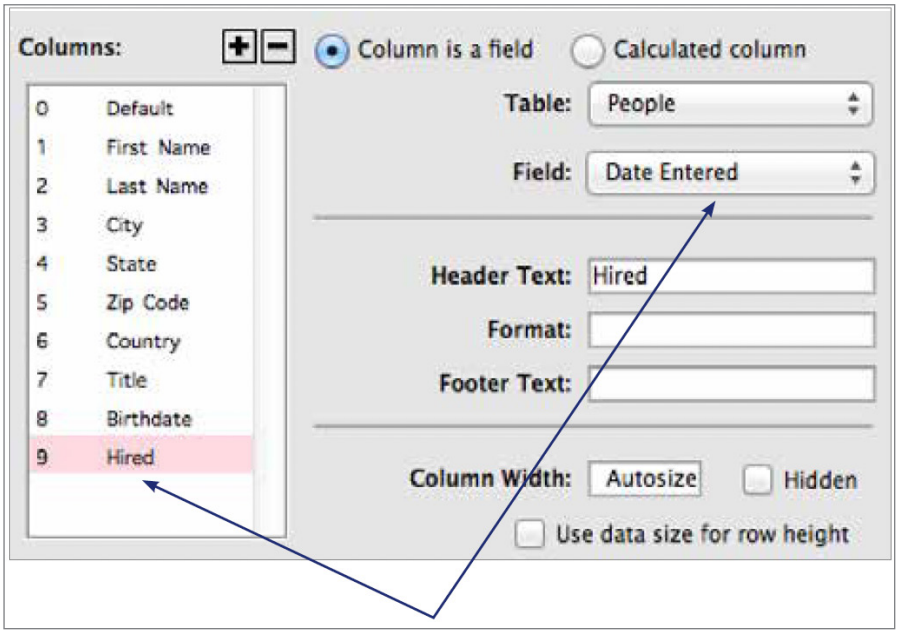
If you have made a change to your Default column and you want to apply that change to all the existing columns, click this button. The current Default column settings will be applied to the properties with blue labels for all columns.

Column Settings

Display: Choose whether you want to show fields or arrays in the area.

Main Table: If displaying fields, select the basic table that the fields will be drawn from.

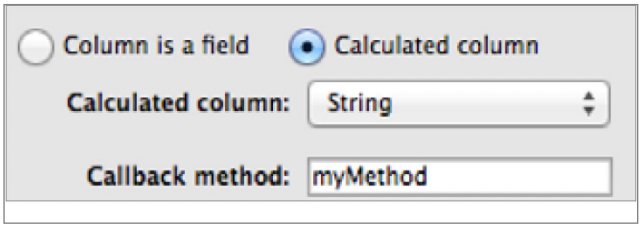
Columns: This is where you specify the actual columns that will appear in the area. To add a new column, click on the big **+** sign. The display will then change:



Column is a field/calculated: if the data for this column will be drawn directly from a field, choose the table and field (you need to add at least one column with the plus "+" button to display the Table & Field popup menus).

If you want to add a calculated column, choose the **Calculated column radio button**.

The display changes again:



To use a calculated column, you will need to create a callback method to handle the actual calculation.

Choose the calculated column type and enter the name of your callback method. For an example of using a callback method to perform a calculation, see the example for the AL_AddCalculatedcolumn command.

If you are displaying arrays, enter the name of the array you want to use in this column.

Note: arrays must be declared before the area is displayed.

Header Text: Enter the title for this column.

Format: You can enter a standard 4D formatting mask here. For example, to display a price with a dollar sign and two decimal places, enter the format "$###0.00".

Footer Text: Enter some text for the footer row, if desired.

Column Width: The default columns width will be as specified in the **Default** column setting (Autosize, unless you change it).

Hidden: Select if you want this column to be hidden.

Use data size for row height: Row height will be calculated according to the font size. If you are displaying pictures in any column, this setting will read “Use picture size for row height” when that column is selected.

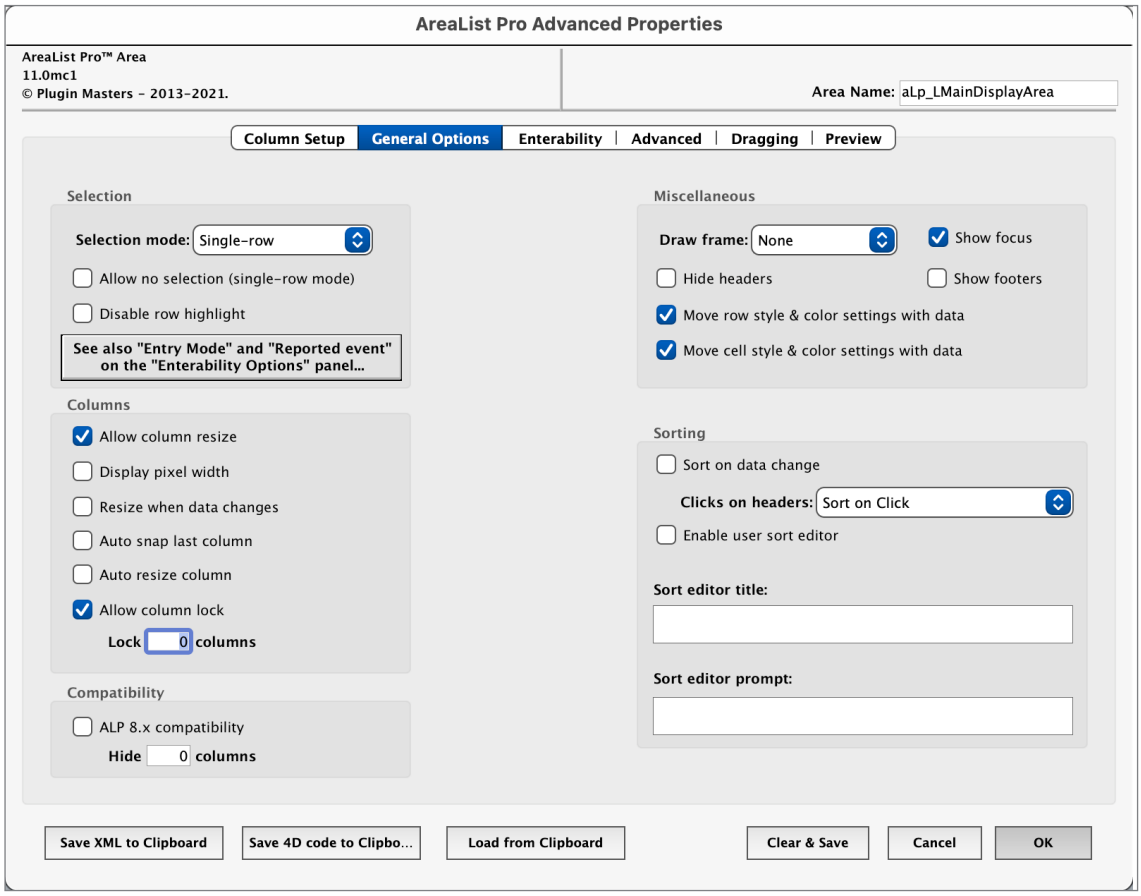
Style options (font, size, color, etc.): You can select any styling for each column.

Enterability: For each column you can specify whether it will be enterable, and by what means - e.g. by keyboard and/or popup. If you select By Popup, you’ll need to enter the name of the array with which to populate the popup in the Popup array field.

Boolean data, display: Choose how you want Boolean data to be displayed (check box with title, check box without title, or radio button).

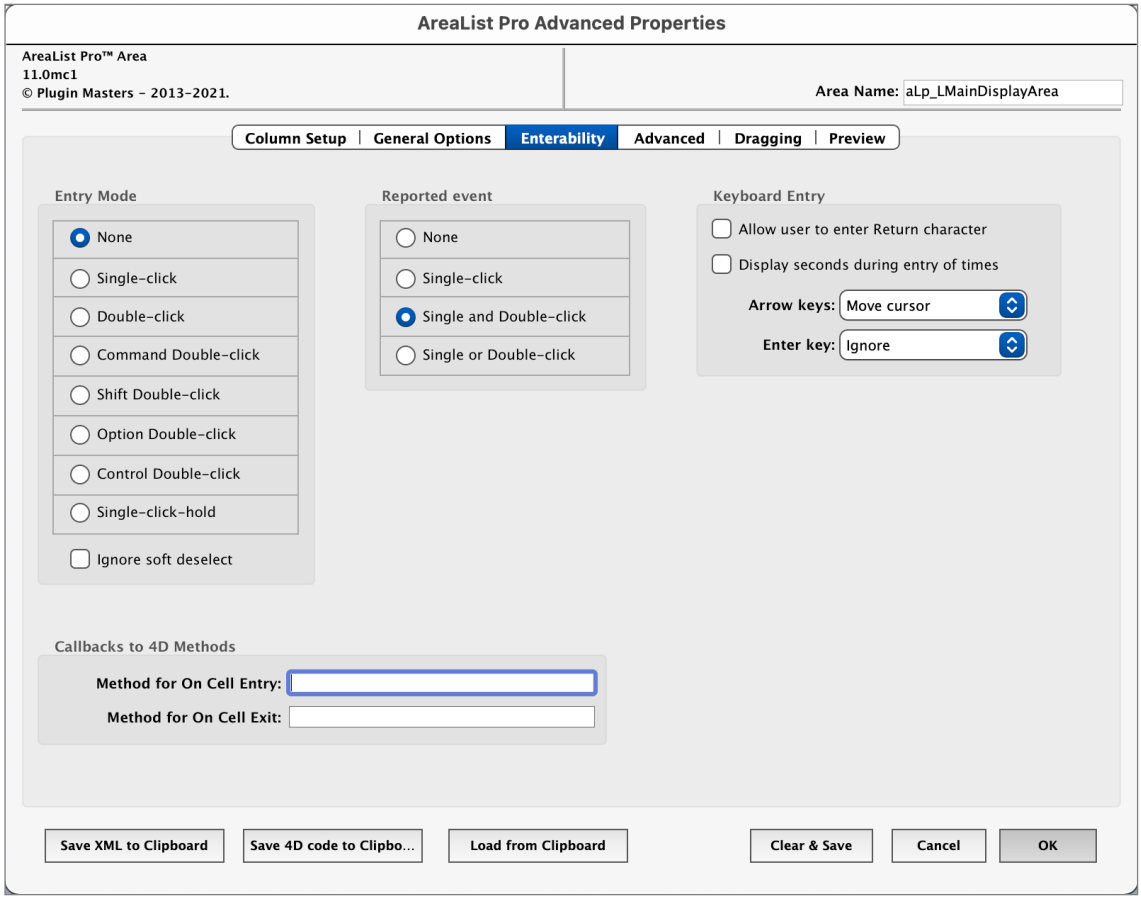
General Options Tab

Here you can set various options that will apply to the entire area, such as whether column resizing is allowed, if the Sort Editor should be available, whether headers and footers will be shown, and so on:



Enterability Tab

On the Enterability tab you can specify the entry and selection mode, various keyboard entry options, and callbacks to use when entering or leaving a cell:



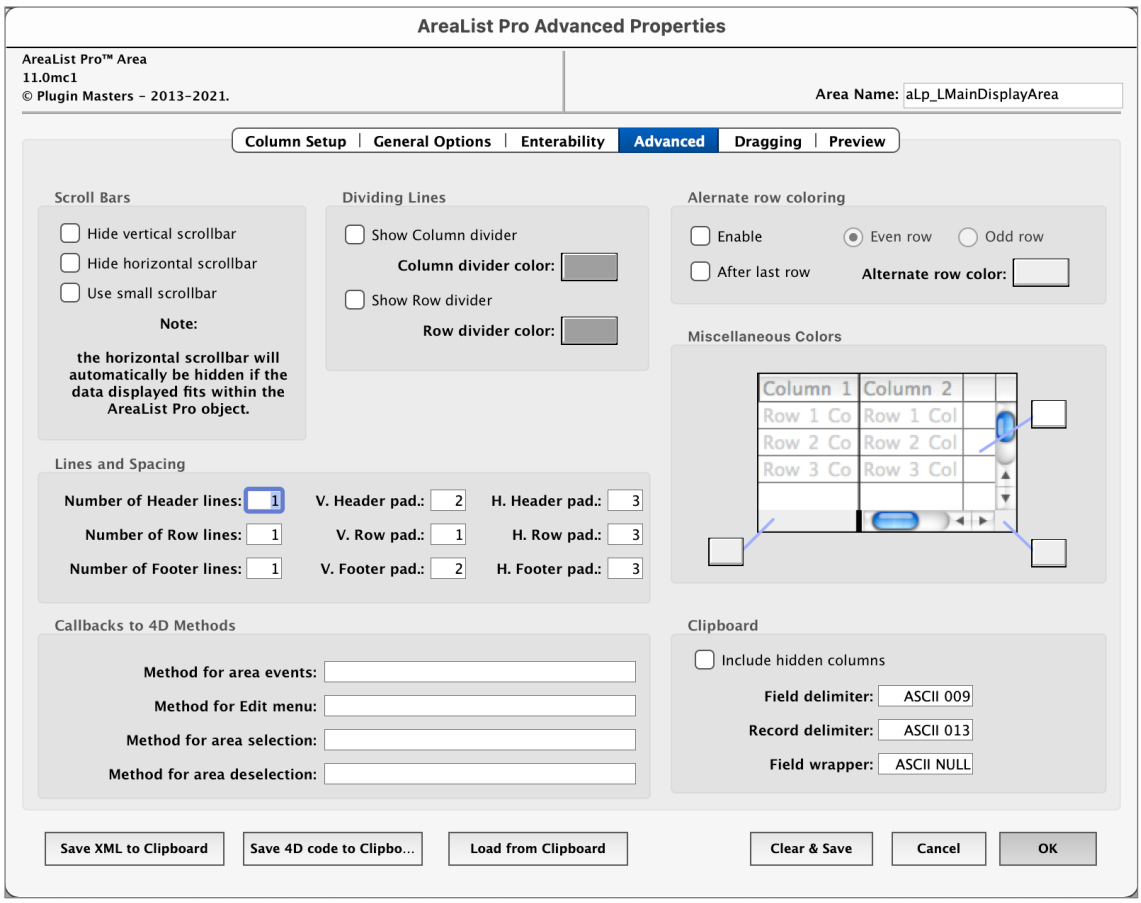
Callbacks: You can specify a callback method that will execute when an enterable cell is entered or exited.

Advanced Tab

The Advanced tab enables you to customise the look of your AreaList Pro area by choosing various options such as colors, whether to hide or show scroll bars, and how to format data when it is copied to the clipboard.

You can also designate callback methods to run when:

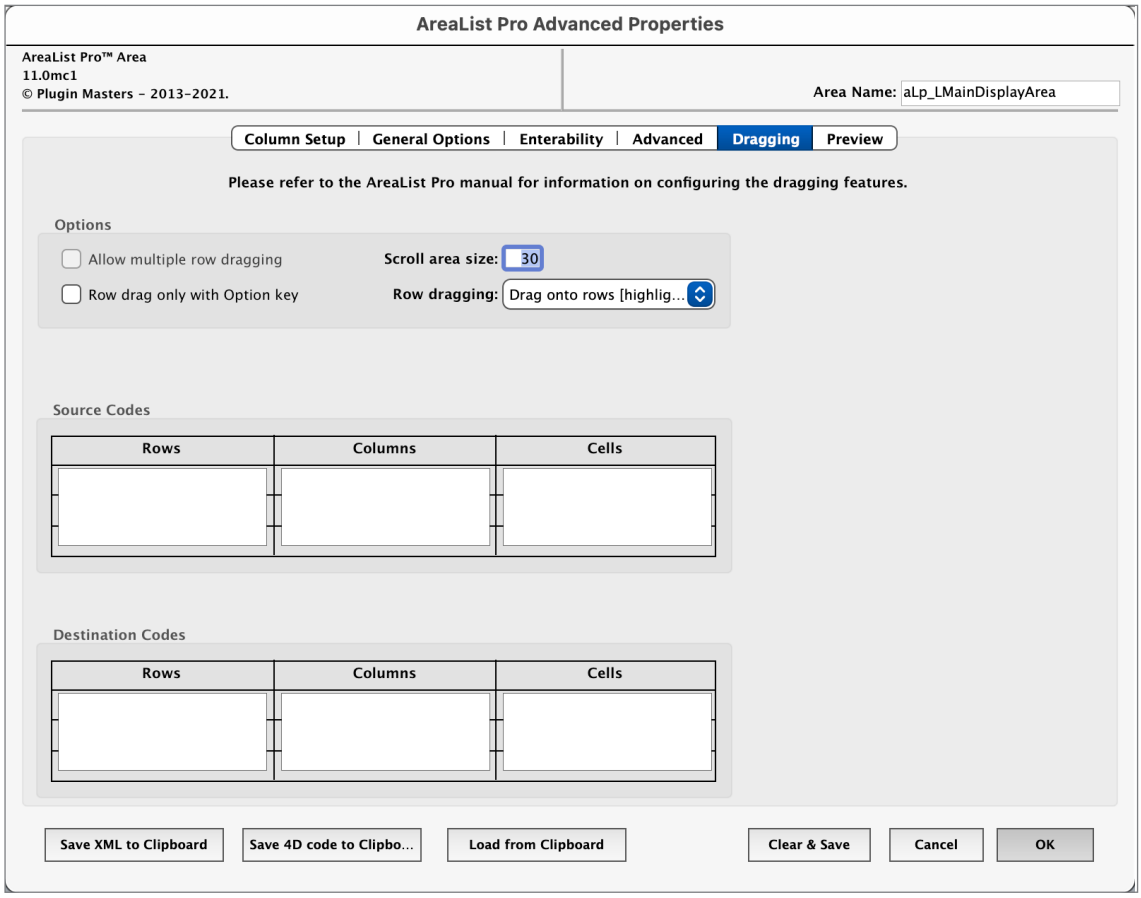
- The area is selected or deselected
- The **Edit** menu is used
- An area event occurs



Dragging Tab

Before you can configure any dragging and dropping with AreaList Pro, you must select the **Draggable** and/or **Droppable** properties in the **Action** topic of the area's Property List.

On this tab you describe what kind of drag and drop actions you want to allow:



Allow multiple row dragging: If this option is not selected, only one row can be dragged and dropped. If it is selected, the user can selected multiple rows to drag and drop in one action.

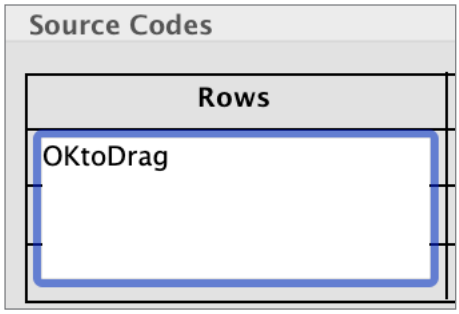
Scroll area size: The size of the frame around the AreaList Pro area border where dragging will start area scrolling. When the user drags something near the AreaList Pro area border, the contents will be scrolled.

Row drag only with Option key: If this is selected, a row drag will only be allowed if the Option key is used.

Row dragging: Choose whether drags go between rows or on top of rows in the destination area.

Source and Destination Codes

When you want to enable dragging between two AreaList Pro areas, you pair them up by specifying Source and Destination codes. For example, suppose you want to allow rows to dragged **from** this area (the Source). You could create a code “OKtodrag” and add it to the Rows area under Source Codes:



You would then add the same code to the Destination Codes area in the AreaList Pro area that you want to allow dragging **TO** from this area.

You can add any number of codes to each option - one code per line.

Drags can take place between AreaList Pro areas on the same form, on a different form in the same process, or on a form in another process. Drags can also take place from non-AreaList Pro objects such as 4D fields and external files.

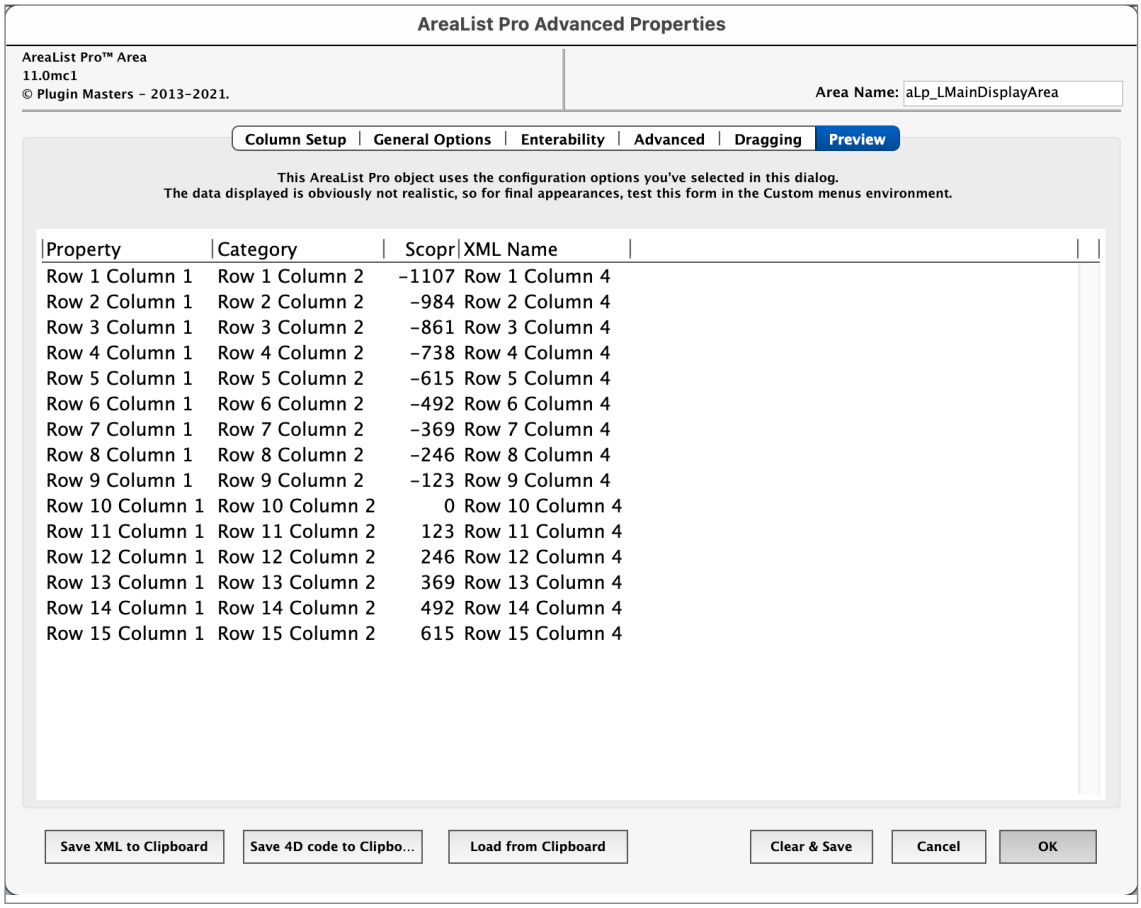
Once an object has been dragged, you will need to handle the Drop event programmatically; AreaList Pro doesn’t know what you want to do with the data that has been dropped, so you must tell it.

Please refer to the Drag and Drop topic for more detailed information and some examples.

Preview Tab

In the Preview tab you can - guess what - see a preview of how your area will look. For example, this preview shows an area in which various options have been selected:

- the **Price** column is in blue italics and displays a $ sign
- a popup menu has been associated with the **Type** column
- the **New** column is displayed as a checkbox



Property	Category	Scopr	XML Name
Row 1 Column 1	Row 1 Column 2	-1107	Row 1 Column 4
Row 2 Column 1	Row 2 Column 2	-984	Row 2 Column 4
Row 3 Column 1	Row 3 Column 2	-861	Row 3 Column 4
Row 4 Column 1	Row 4 Column 2	-738	Row 4 Column 4
Row 5 Column 1	Row 5 Column 2	-615	Row 5 Column 4
Row 6 Column 1	Row 6 Column 2	-492	Row 6 Column 4
Row 7 Column 1	Row 7 Column 2	-369	Row 7 Column 4
Row 8 Column 1	Row 8 Column 2	-246	Row 8 Column 4
Row 9 Column 1	Row 9 Column 2	-123	Row 9 Column 4
Row 10 Column 1	Row 10 Column 2	0	Row 10 Column 4
Row 11 Column 1	Row 11 Column 2	123	Row 11 Column 4
Row 12 Column 1	Row 12 Column 2	246	Row 12 Column 4
Row 13 Column 1	Row 13 Column 2	369	Row 13 Column 4
Row 14 Column 1	Row 14 Column 2	492	Row 14 Column 4
Row 15 Column 1	Row 15 Column 2	615	Row 15 Column 4

Click **OK** when you are happy with your settings, and your area is ready to use.

12 Drag and Drop

AreaList Pro enables rows, columns and cells to be dragged and dropped from and to AreaList Pro areas.

Overview

You can control which areas can be dragged from or to, what options are available (e.g. whether multiple rows can be dragged or not), and what happens after a drop. Row dragging can be initiated either by alt/option-clicking on an item (cell, row, or column) and dragging it or by simply dragging it, depending on how it has been configured.

When an item is clicked and dragged, the pointer will change.

If the drop is allowed, the pointer will have a plus symbol attached to it when it hovers over the “drop” area:

If the drop is not allowed by the destination object, you’ll see a “no entry” sign instead:

Dragging

You can allow rows, columns and cells to be dragged from an AreaList Pro area and dropped to various destinations:

- a row, a column or a cell in the same AreaList Pro area
- a row, a column or a cell in another AreaList Pro area
- a day, an event or a banner in a CalendarSet area
- any 4D droppable object
- another application that accepts text

Dropping

You can also allow dropping onto rows, columns and cells in an AreaList Pro area from various sources:

- row(s), a column or cell(s) from the same AreaList Pro area
- row(s), a column or a cell(s) from another AreaList Pro area

- events and banners from a CalendarSet area
- any 4D draggable object
- text or other contents displayed in another application window
- a document in a MacOS Finder or Windows Explorer window

Item types

When dragging and dropping between AreaList Pro areas, or within the same area, the destination item will match the source item:

- row(s) will be dropped onto a row
- column will be dropped onto a column (use the header to select the source column that you want to drag)
- cell(s) will be dropped onto a cell

Controlling the Drag and Drop

You can control each AreaList Pro area's draggability and droppability:

- if the area can be dragged from
- if the area can be dropped to
- which item types (rows, columns and / or cells) can be dropped
- which AreaList Pro or CalendarSet areas can be the source or destination
- if external sources (non-AreaList Pro / CalendarSet objects) are allowed
- whether an attempted a drop on the area is accepted or rejected
- what happens after a drop

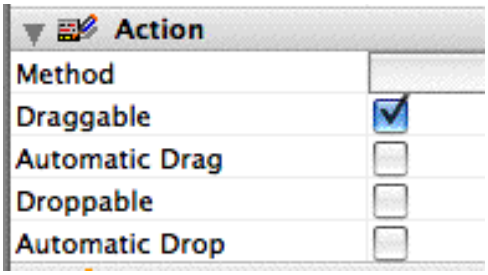
Configuring Drag and Drop

You must configure AreaList Pro to allow dragging out of and into an AreaList Pro area.

Setting the 4D Object Properties

The first thing you must do is select the Drag and Drop properties for the AreaList Pro areas as 4D objects.

1. Select the source object (the AreaList Pro area that you want to enable dragging **from**)
2. In the **Action** topic of the **Property List** dialog, select the **Draggable** checkbox:



3. Select the destination object (the AreaList Pro area that want to drop **to**)
- 4. In the **Action** topic of the **Property List** dialog, select the **Droppable** checkbox.

An area can be both Draggable and Droppable, which will amongst others enable drag and drop within the area.

Note: in compatibility mode, the AreaList Pro area is draggable and droppable even if it is not set as draggable or droppable in the object properties.

Drag and drop Properties

AreaList Pro properties provide the control necessary to allow or disallow dragging within an area, between two or more areas or from non-plugin objects. You can allow dropping from and onto rows, columns and/or cells.

In order to facilitate dragging and dropping, you need to tell AreaList Pro which area(s) you want to allow the dragging between, and specify various options such as which objects can be dragged (rows, columns, and/or cells), which areas those objects can be dragged to and from, and whether certain keys - such as the Alt/Option key - will have any particular effect.

Access “codes” overview

To allow dragging out of AreaList Pro, you must pass an “access code” for each type of item (rows, columns and/or cells) that can be dragged from this area. You must specify at least one code to enable dragging, using ALP_Area_DragSrcXXXCodes properties (where XXX is Row, Col or Cell).

Any number of codes can be passed. Allowing many codes provides for more flexibility in enabling and disabling dragging between various areas.

In order to allow dropping into AreaList Pro, you must pass an “access code” for each type of item (rows, columns and/or cells) that can be the destination of a drop. You must specify at least one code to enable dropping, using ALP_Area_DragDstXXXCodes properties (where XXX is Row, Col or Cell). As with source code properties, any number of codes can be passed for flexibility reasons.

Property list

AreaList Pro provides a number of properties that you can use to set and find the required details. Use these properties with the ***AL_SetArea…*** and ***AL_GetArea…*** commands:

Property	Type	Description
ALP_Area_DragDataType	longint	Dragged data: 1 = row(s) 2 = column 3 = cell(s)
ALP_Area_DragDstArea	longint	Destination area
ALP_Area_DragDstCell	longint	Destination area cell
ALP_Area_DragDstCellCodes	text	Drag destination cell codes
ALP_Area_DragDstCol	longint	Destination area column
ALP_Area_DragDstColCodes	text	Drag destination column codes
ALP_Area_DragDstProcessID	longint	4D’s process ID of the destination area
ALP_Area_DragDstRow	longint	Destination area row
ALP_Area_DragDstRowCodes	text	Drag destination row codes
ALP_Area_DragProcessID	longint	4D’s process ID of the source area
ALP_Area_DragSrcArea	longint	The dragged AreaList Pro area
ALP_Area_DragSrcCell	longint	Source area cell
ALP_Area_DragSrcCellCodes	text	Drag source cell codes
ALP_Area_DragSrcCol	longint	The source area column
ALP_Area_DragSrcColCodes	text	Drag source column codes

Property	Type	Description
ALP_Area_DragSrcRow	longint	Source area row
ALP_Area_DragSrcRowCodes	text	Drag source row codes

Alt/Option key

A default setting that you may want to change is the “drag with Alt key” option.

The default setting for this is that the user must hold down the Alt or Option key to effect a drag. You can turn this off by setting the ALP_Area_DragOptionKey property for the source area to False - for example:

AL_SetAreaLongProperty (ProductList;ALP_Area_DragOptionKey;0) // don’t need alt key to drag

What are access “codes”?

The access codes that are passed in ALP_Area_DragSrcXXXCodes and ALP_Area_DragDstXXXCodes (where XXX is Row, Col or Cell) are used to enable dragging between specific drag partners.

These drag partners can be the same AreaList Pro area, different AreaList Pro areas, CalendarSet plugin areas, text selections from 4D or other applications and external documents.

- Setting at least one source access code to an area will make it draggable (from the specified object type).
- Setting at least one destination access code to an area will make it droppable (on the specified object type).

Drag and drop between plugin areas

When dragging from and to AreaList Pro or CalendarSet areas (or within the same area) the source of the drag and the target of the drop are designated as drag and drop partners.

You need to tell AreaList Pro (and CalendarSet if used) what these partnerships are, and to do this you create **access codes**.

An access code is simply a text code that you create. Let’s say you have a layout that contains two AreaList Pro areas: **ProductList** and **SelectedItems**, and you want to enable items to be dragged from ProductList and dropped onto SelectedItems. You might decide on the access code “select”.

To enable row dragging you will need two lines of code:

AL_SetAreaTextProperty (ProductList;ALP_Area_DragSrcRowCodes;"select")

AL_SetAreaTextProperty (SelectedItems;ALP_Area_DragDstRowCodes;"select")

You can list any number of access codes.

For example, suppose you have four AreaList Pro areas on a form - AreaA, AreaB, AreaC and AreaD. You want to allow drag and drop from AreaA to AreaB or AreaC, and from AreaD to AreaC but not AreaB:

1. Create two access codes: “dropB” and “dropC”.

2. Set the access code properties for the four areas as follows:

AL_SetAreaTextProperty (AreaA;ALP_Area_DragSrcRowCodes;"dropB|dropC")

AL_SetAreaTextProperty (AreaD;ALP_Area_DragSrcRowCodes;"dropC")

AL_SetAreaTextProperty (AreaB;ALP_Area_DragDstRowCodes;"dropB")

AL_SetAreaTextProperty (AreaC;ALP_Area_DragDstRowCodes;"dropC")

Note that the items in the list of codes are separated by a pipe character: “dropB|dropC”

You can specify different codes for cells, rows, and columns.

Note: as opposed to CalendarSet commands where each access code is a parameter by itself: "dropB";"dropC", with AreaList Pro properties all access codes are passed in the same string (list of codes separated by '|').

That's all you need to do to enable basic drag and drop functionality between two areas with default settings.

When a drag and drop takes place, the drag sender's plugin code communicates its access codes to the drop receiver's plugin code. The drop receiver will compare the access codes of the sender to its own codes. If any of the codes match, the drop is allowed.

This mechanism allows a number of combinations between several drag and drop partners.

Note: access codes are strictly compared, taking into account case and diacritics.

Example One (one area)

The following is an example of enabling the dragging and dropping of events within the same AreaList Pro area by setting a unique identifier that only enables dragging within this area.

```
 //enable drag rows to rows within this area
vSelfStr:="ALParea"+String(eList)  //only allows dragging and dropping within this area
AL_SetAreaTextProperty (eList;ALP_Area_DragSrcRowCodes;vSelfStr)
AL_SetAreaTextProperty (eList;ALP_Area_DragDstRowCodes;vSelfStr)
```

Example Two (two areas)

Suppose we have a form on which there are two AreaList Pro areas: **ProductList** contains a list of products, and **SelectedItems** contains a list of products that have been selected from the list. We want to allow users to add products to **SelectedItems** by dragging rows from **ProductList**.

We've set up the two areas as described above (under Set the access code properties). Three arrays have been initialised and added to **SelectedItems** (atProductPurch, aiQty, and arTotal).

When a product is dropped onto **SelectedItems** we need to add a row to each of the **SelectedItems** arrays and fill them with the appropriate data if the product hasn't already been selected, or update the arrays if it has already been selected.

In the object method for **SelectedItems** we call a method called ***AddProductToOrder***:

```
Case of
   : (Form event=On Drop)
   AddProductToOrder (Self)
End case
 //AddProductToOrder project method
C_LONGINT($DropArea;$SourceRow;$ProductRow)
$DropArea:=$1->
$SourceRow:=AL_GetAreaLongProperty ($DropArea;ALP_Area_DragSrcRow)
GOTO SELECTED RECORD([product];$SourceRow)
$ProductRow:=Find in array(atProductPurch;[product]product_name)
If ($ProductRow<1)
   APPEND TO ARRAY(atProductPurch;[product]product_name)
   APPEND TO ARRAY(aiQty;1)
   APPEND TO ARRAY(arTotal;[product]retail_price)
```

```
Else
   aiQty{$ProductRow}:=aiQty{$ProductRow}+1
   arTotal{$ProductRow}:=aiQty{$ProductRow}*[product]retail_price)
End if
AL_SetAreaLongProperty ($DropArea;ALP_Area_CheckData;0)  // tell AreaList Pro we expanded the arrays
```

Drag and drop with external objects

External objects are defined in this context as non AreaList Pro (or CalendarSet) plugin areas:

- Dragging from and dropping to 4D fields, variables or any droppable / draggable object
- Dragging from and dropping to 4D text selections (like a text variable content)
- Dragging from and dropping to text selections in other applications (open windows from e.g. a text editor)
- Dragging from external files (no dropping to)

Since external objects obviously don't support access codes they will be potential recipients of any drag from a draggable AreaList Pro area, or source of any drop to a droppable AreaList Pro area.

Any source access code will make a AreaList Pro area draggable to external objects.

The "_external" special access code must be used as a destination access code in order to make an AreaList Pro area droppable from external objects:

- "_external" can only be used as a destination access code and is the only way to make an AreaList Pro area accept a drop from objects other that AreaList Pro or CalendarSet areas
- "_external" as a destination access code means "drop onto this AreaList Pro area from any external object"

We can modify the example above to allow dragging from an external object to area B:

```
AL_SetAreaTextProperty (AreaA;ALP_Area_DragSrcRowCodes;"dropB|dropC")  // drag to B, C and external objects
AL_SetAreaTextProperty (AreaD;ALP_Area_DragSrcRowCodes;"dropC")  // drag to C and external objects
AL_SetAreaTextProperty (AreaB;ALP_Area_DragDstRowCodes;"dropB|_external")  // accept drop from A & external objects
AL_SetAreaTextProperty (AreaC;ALP_Area_DragDstRowCodes;"dropC")  // accept drop from A & D
```

When dragging from an external object to an AreaList Pro area, the destination type will either be a row or a cell according to the area's current selection mode (ALP_Area_SelType).

Using the Event callback method

The processing of the drop event can be handled either in the event callback method or in the On Drop event on the AreaList Pro area method.

Generally it is best to handle the processing in the On Drop form event.

AL_GetAreaLongProperty with ALP_Area_AlpEvent is used to find out that a drop occured: in this case, the AL Row drop event, AL Column drop event, AL Cell drop event or AL Object drop event value tells us that something was dropped onto the area, and what was dropped (row(s), column, cell(s) or non-AreaList Pro object).

You can also fine-tune your control over the user's drag and drop from the event callback method set by the ALP_Area_CallbackMethOnEvent property (callback methods are explained in detail elsewhere):

```
AL_SetAreaTextProperty (ProductList;ALP_Area_CallbackMethOnEvent;"AlpEventCallback")
```

During a drag and drop, this method will be called under two circumstances:

- When a drop is attempted from an external object, while the pointer is still hovering over the AreaList Pro destination area: the $0 value returned by the callback will allow the subsequent drop action or not (as with setting access codes to control drag and drop between plugin areas): AL Allow drop event.
- After a drop has been performed, whatever the source was (plugin area or external object): AL Row drop event, AL Column drop event, AL Cell drop event or AL Object drop event.

Note: the callback method will **not** be called with AL Allow drop event when the source of the drag and drop is a plugin area, where access codes are used instead to specifically allow drag and drop between areas.

When the drop is completed, the form method/AreaList Pro area object (or callback method if previously set by ALP_Area_CallbackMethOnEvent) is executed. You can then determine what the last user action was using ALP_Area_AlpEvent (form/object method) or $2 (callback method).

Note: the event reported by ALP_Area_AlpEvent will never be AL Allow drop event for an external object source, but the callback will receive this event in $2. See “Allow drop” below.

The source area's last event is AL Row drag event, AL Column drag event or AL Cell drag event (drag events). The destination area's last event is AL Row drop event, AL Column drop event, AL Cell drop event or AL Object drop event.

The drag events are not reported if the drag ended as a drop into the same AreaList Pro area.

Note: AL Row drag event, AL Column drag event and AL Cell drag event drag event should no longer be used anyway. Use the AL Row drop event, AL Column drop event, AL Cell drop event and AL Object drop event callback events instead, or the form/object method On Drop 4D event using ***AL_GetAreaLongProperty*** with ALP_Area_AlpEvent.

Allow drop

The callback method is called with the AL Allow drop event event when an external object is dragged over the area. It is used to allow or reject the drop.

For example:

```
Case of
  : ($2=AL Allow drop event)
    $0:=1  // allow
End case
```

Note: never display a window in the callback while processing this AL Allow drop event, including **TRACE** or **ALERT**: 4D would freeze (ACI0087433).

This event callback method (actually a function in this case) receives six parameters, and must return a result:

- $1 is the destination AreaList Pro area reference.
- $2 is the event, in this case AL Allow drop event
- $0 is expected by AreaList Pro, with a special meaning for this event: 0 (disallow drop) or 1 (allow drop).

Note: if no callback is set the drop from an external object will **not** be accepted (even when "_external" is set as a destination access code for the area). Therefore the callback is the only way to allow a drop from an external object on a droppable area.

The pointer shape will depend upon this result:

- plus symbol if the drop is allowed:

- “no entry” sign if the drop is not allowed:

Note: once an allowed destination has been rolled over, the pointer will stay as “+” even when you subsequently move to a place where the drop is not allowed: the drop will nevertheless be allowed and the On Drop form event will be triggered.

This is due to a limitation of 4D, which does not call the plugin again to ask if the drop is allowed unless you leave the area and re-enter it again.

In this case ALP_Area_DragSrcArea, ALP_Area_DragSrcRow, ALP_Area_DragSrcCol and ALP_Area_DragSrcCell will return zeros.

After the drop

When an item is dropped onto an AreaList Pro area, the following information is available to you:

- Notification that a drop occurred
- Which item was dragged and which type (row, column or cell)
- Where the item was dragged from (this area or another area, or another kind of object)
- The type of data that was the recipient of the drop (row, column or cell)

The processing of the drop event can either be handled in the callback with the AL Object drop event event or the form method / AreaList Pro area object method with the On Drop event, which is always executed in the context of the destination area / process.The call sequence is:

1. Callback method with AL Row drop event, AL Column drop event, AL Cell drop event or AL Object drop event in $2

2. Area object method with On Drop form event

3. Form method with On Drop form event

Note: when displaying AreaList Pro in an external window there is no form / object method to execute, therefore the callback is the only way to control drag and drop in this context.

The callback method receives six parameters as usual, the first two being:

- $1 the destination AreaList Pro area reference.
- $2 the event code, here AL Row drop event, AL Column drop event, AL Cell drop event or AL Object drop event

You can also determine which AreaList Pro object type was dropped using ALP_Area_DragDataType.

To determine which area was the source of the drag, use ALP_Area_DragSrcArea. This property returns the area reference (a long integer) and ALP_Area_DragProcessID is the process ID of the source area, whether it is the same AreaList Pro area or another AreaList Pro area, or a CalendarSet area (the area reference will be negative in this case).

To find out which row, column or cell was dropped, use either ALP_Area_DragSrcRow, ALP_Area_DragSrcCol or ALP_Area_DragSrcCell.

The above properties will return a single source row, column or cell. For example ALP_Area_DragSrcRow is the selected row (ALP_Area_SelRow) at the time the drag was initiated.

If you want to handle multiple rows (ALP_Area_DragRowMultiple + ALP_Area_SelMultiple) or cells as sources, use the Object Properties with the Objects command theme to retrieve the source area's selection:

```
ARRAY LONGINT(aRows;0)
$error:=AL_GetObjects (eList;ALP_Object_Selection;aRows)  // get the rows selected by user
```

When dragging to / from another area or a 4D object, that object can either reside in the same window or on another window, which may require use of 4D's process communication commands to take action on the drop.

Note: when dragging and dropping to or from other objects, AreaList Pro is only providing a user interface to the drag, and notifying you, the developer, that the drop has occurred. You are responsible for manipulating any arrays or other data structures.

See also Reordering after dragging within one area and Dragging to a 4D object.

Receiving a drop from a non-AreaList Pro Object

As well as dragging between two AreaList Pro areas, you can also drag between non-AreaList Pro objects and AreaList Pro areas — for example, an event or a banner from CalendarSet, another 4D object, a text selection in any (drag and drop savvy) application window or a drop from an external document.

Receiving a drop from a CalendarSet area is identified by ALP_Area_DragSrcArea returning a negative value (opposite of the CalendarSet area reference) and its process in ALP_Area_DragProcessID.

If the source of the drag is an external object, the callback (if set for the area) will be triggered twice:

- when dragging over the area ($2=AL Allow drop event)
- after the drop ($2=AL Object drop event)

In both cases (and in the 4D object / form method after the drop) ALP_Area_DragSrcArea will be zero and ALP_Area_DragProcessID will be:

- the 4D originating object's process if the source is a 4D object
- -1 if the source is a document on disk or another application

Also in both callback calls the content of the dragged and dropped object is the pasteboard data (text, picture and / or other).

Use **GET PASTEBOARD DATA** to analyze the dragged data and allow the drop (AL Allow drop event) or process it (AL Object drop event).

Allowing the drop from external objects in the callback

When dragging from non-plugin objects, you can use an event callback method to “catch” the user action and allow it or not (event callback methods are set with the ALP_Area_CallbackMethOnEvent property). If no callback is set, the drop will be allowed.

In order for external object drag and drop to be disallowed once an area has been made droppable with the "_external" destination access code, the callback method must be set and handle the AL Allow drop event case.

See the External documents example below.

CalendarSet

The source CalendarSet area is referenced as a negative value in the ALP_Area_DragSrcArea property (opposite of the CalendarSet area reference).

Plugin area references use sequential numbering: 1 can be AreaList Pro and CalendarSet area references at the same time, therefore CalendarSet area #1 will be referenced from AreaList Pro's point of view as -1, to differentiate from AreaList Pro area #1.

```
C_LONGINT($srcArea;$srcCSArea)
If (Form event=On Drop)  // here we use the object method, no callback set
   $srcArea:=AL_GetAreaLongProperty (eList; ALP_Area_DragSrcArea)
   $srcCSArea:=-$srcArea  // CalendarSet's area reference (note the minus sign)
   If ($srcCSArea=CSappointments)  // is this CalendarSet area our appointment calendar?
    // Do something with the appointements
   End if
End if
```

Refer to the CalendarSet manual regarding accepting a drop in a CalendarSet area.

4D

You can use the following code in the On Drop event when accepting a drop from a 4D object:

```
DRAG AND DROP PROPERTIES($srcObject;$srcElement;$srcProcess)
```

Note: $srcObject is **Nil** if the source 4D object has Automatic Drag enabled. $srcObject is also **Nil** if it comes from a different application (or 4D instance).

Then you can use the following code to check what has been dropped:

```
ARRAY TEXT($4Dsignatures;0)
ARRAY TEXT($nativeTypes;0)
ARRAY TEXT($formatNames;0)
GET PASTEBOARD DATA TYPE($4Dsignatures;$nativeTypes;$formatNames)
```

Note: the On Drop event code will work correctly after the drop when used in an area's object method but in an event callback the form event is zero and drag & drop properties from 4D will not function. However the pasteboard can still be analyzed in the callback with both events AL Allow drop event and AL Object drop event.

External documents

After Allowing the drop from external objects in the callback, the AL Object drop event AreaList Pro event or the On Drop 4D event are used to open the document according to the pathname retrieved from the pasteboard, then process it.

Let's suppose we want to import some data into a series of arrays by dropping a text file onto an AreaList Pro area. We've saved a spreadsheet that contains information on some new Nuts products as a tab-delimited text file:

"Macadamia nuts, 50g"	MAC-001	Nuts	Snack-sized bag of nuts	2.5
"Macadamia nuts, 100g"	MAC-002	Nuts	Family-sized bag of nuts	4.5
"Pecans, 50g"	PEC-001	Nuts	Snack-sized bag of pecan nuts	3.5
"Macadamia nuts, 100g"	PEC-002	Nuts	Family-sized bag of pecan nuts	5.5
"Dry roasted Peanuts 50g"	PEA-001	Nuts	Snack-sized bag of salted, roasted peanuts	2.5

When this text file is dropped onto a list of products, we want to create a new row for each new product and populate the appropriate arrays with the product's details.

Setting up the Area

1. Create a new AreaList Pro area on your form
2. In the Property List, select the **Droppable** option under the **Action** topic, and the **On Drop** event.
3. Create a callback method:

```
//AlpEventCallback
C_LONGINT($1)  //AreaList Pro object reference
C_LONGINT($2)  //AreaList Pro event
C_LONGINT($3)  //4D event
C_LONGINT($4)  //last clicked column (or column under the pointer for mouse moved event)
C_LONGINT($5)  //last clicked row (or row under the pointer for mouse moved event)
C_LONGINT($6)  //modifiers
C_LONGINT($0)
Case of
   : ($2=AL Allow drop event)
      $0:=1  //allow
End case
```

4. Assign that callback method to the area:

```
Case of
   : (Form event=On Load)
      AL_SetAreaTextProperty (ProductList;ALP_Area_CallbackMethOnEvent;"AlpEventCallback")
End case
```

This code can go either on the AreaList Pro destination area object method or in the form method.

Handling the Drop

Add some code to the On Drop event section of the AreaList Pro destination area object method:

```
Case of
  : (Form event=On drop)
    DRAG AND DROP PROPERTIES($srcObject;$srcElement;$srcProcess)
    $dragSource:=AL_GetAreaLongProperty (Self->;ALP_Area_DragSrcArea)
    If ($dragSource=0)  // not an AreaList Pro area
      If (Nil($srcObject))  // external source
        GET PASTEBOARD DATA("com.4d.private.file.url";$data)  // gets file pathname
        If (OK=1)
          PLATFORM PROPERTIES($platform)
            If ($platform=Windows)
              $LineDelimit:=Char(10)
            Else
              $LineDelimit:=Char(13)
            End if
          $path:=Get file from pasteboard(1)  // first file
          $FileType:=Document type($path)
          If (($FileType="txt") | ($FileType="text"))
            SET CHANNEL(10;$path)  // open the file
            While (OK=1)  // file opened OK & more data to receive
            RECEIVE PACKET($tdata;$LineDelimit)  // get one row
              ARRAY TEXT(atVals;0)
              explode ($tdata;9;->atVals)  // parse the text into the array (see below)
              If (Size of array(atVals)=5)
                APPEND TO ARRAY(atName;Replace string(atVals{1};Char(34);""))
                APPEND TO ARRAY(atCode;atVals{2})
                APPEND TO ARRAY(atType;atVals{3})
                APPEND TO ARRAY(atDesc;Replace string(atVals{4};Char(34);""))
                APPEND TO ARRAY(arWprice;Num(atVals{5}))
              End if
            End while
            SET CHANNEL(11)  // close the file
            SORT ARRAY(atName;atCode;atType;atDesc;arWprice)
            AL_SetAreaLongProperty (Self->;ALP_Area_CheckData;0)
             // tell AreaList Pro we expanded the arrays
          End if
        End if
      End if
    End if
End case
```

5. To test it, load the form and then drop the text file onto the area. The On Drop event will execute and the five new products will be added to the area.

Utility

The ***explode*** project method splits a delimited line of text into an array:

```
// explode
// explodes a text string into parts using designated separator character
// and returns them in an array
// parameters: $1 = the text string
// $2 = the separation character(ASCII value)
// $3 = pointer to the array to put the values in
// supports only TEXT arrays
// array must be declared and zero'd first
// example: explode (tText;9;->atVals))
C_TEXT($text;$char)
$text:=$1
$char:=Char($2)
$Elements:=0
While (Length($text)>0)
    $pos:=Position($char;$text)
    If ($pos>0)
        $value:=Substring($text;1;$pos-1)
        $text:=Substring($text;$pos+1)
    Else
        $value:=$text
        $text:=""
    End if
    APPEND TO ARRAY($3->;$value)
End while
```

Hints and Tips

Here is some feedback from our support regarding Drag & Drop in AreaList Pro, which you may find useful in addition to the above explanations.

Feel free to ask for more using the AreaList Pro / PrintList Pro forum.

Row dragging in cell selection mode

Row dragging isn't enabled when an AreaList Pro object is in cell selection mode.

To set the selection mode, use the ALP_Area_SelType property of ***AL_SetAreaLongProperty*** - for example:

```
AL_SetAreaLongProperty (area; ALP_Area_SelType;0)  // selection mode = rows
```

Dragging a row to the bottom of the list

If you drag a row from another AreaList Pro area and release below the bottom existing row, ***AL_GetAreaLongProperty*** with ALP_Area_DragDstRow will return the last existing row. This behaviour is compatible with versions 8.x.

Dragging "onto" a row means just that: drag onto an existing row.

You can drop a row onto the empty area below the last row (you can also drop data into an empty AreaList Pro area – without any rows), but the last row is reported as the destination.

You can set the area to "insert" mode:

```
AL_SetAreaLongProperty ($area; ALP_Area_DragRowOnto; 0)
```

In this case, it will append the row to the list rather than inserting above the last existing row.

Drag Line property

ALP_Area_DragLine is used when the source (referenced area) does not have the ALP_Area_DragSrcRowCodes property set. This is also true for ALP_Drop_DragAcceptLine (with the destination area).

It is useless if you call ALP_Area_DragSrcRowCodes to set the matching codes between source and destination. Drag will only be allowed to a matching destination.

The option key values to ALP_Area_DragLine (1-3 = with option, 4-6 = without option) can be set using ALP_Area_DragOptionKey.

Using ALP_Area_DragSrcRowCodes makes it consistent with ALP_Area_DragSrcColCodes (which in turn makes ALP_Drop_DragAcceptColumn obsolete) and ALP_Area_DragSrcCellCodes.

Drag and drop and compatibility mode

If drag within the area does not work unless the compatibility is turned on, check the AreaList Pro object properties on the 4D form - it has to be draggable / droppable.

In compatibility mode, the area is draggable / droppable even if it is not marked as such on the 4D form (previous AreaList Pro versions ignored this setting).

Be aware that Drop has to be handled On Drop, not in the drag context as opposed to previous versions (the drop can end anywhere, not necessarily in an AreaList Pro area).

Reordering after dragging within one area

Rows

AreaList Pro will automatically reorder the rows (i.e. array elements) if all the following conditions are met.

- drag and drop occurs in the same area
- the area is in arrays mode
- ALP_Area_SelType is 0 (row selection)
- ALP_Area_DragRowOnto is 0 (the feedback to the user is “insert” - highlight between rows) - when using the old v8.x API, you have to use 0 as the fifth parameter in your call to ***AL_SetDrgOpts***
- ALP_Area_DragRowMultiple or ALP_Area_SelMultiple is 0

Otherwise your 4D code must process the drag as in the evtDragWithin method from Example 10.

Column

Only one colum at a time can be dropped within the area or to another destination.

There is a property to force a column move instead of reordering the grid. See ALP_Area_DragMoveColumns.

If dropped within the area the dropped colum is moved (inserted) to the target column's position and the columns are automatically reordered.

If the area is not in compatibility mode the grid order is updated (ALP_Object_Grid).

Note: the above behavior does not apply to Grids where rows include several lines.

Selection mode effects

Keep in mind that the selection mode (as defined by ALP_Area_SelType) may limit the ability to drag and drop depending on the object types (specified by their access codes).

- specifying column source access codes makes the area draggable: a column can be dragged
- specifying column destination access codes makes the area droppable: a column can be dropped
- specifying row source access codes in cell selection mode does not make the area draggable
- specifying row source access codes in row selection mode makes the area draggable: row(s) can be dragged
- specifying row destination access codes makes the area droppable: row(s) can be dropped; in cell selection mode, only from a different area (which must be in row selection mode)
- specifying cell source access codes does not make the area draggable when you are in rows mode
- specifying cell source access codes in cell selection mode makes the area draggable: cells(s) can be dragged
- specifying cell destination access codes makes the area droppable: cells(s) can be dropped; if in row selection mode, only from a different area (which must be in cell selection mode)

Dragging to a 4D object

Suppose you want to drag from AreaList Pro to another 4D object (non-AreaList Pro) and you need to collect information about the dragged row(s).

Since AreaList Pro simply starts a drag (when allowed), you must handle On Drag Over & On Drop in the 4D object/form method.

You can retrieve the source variable name from **DRAG AND DROP PROPERTIES** (to know that drag is from an AreaList Pro area) or you can directly access "net.e-node.alp.object" (which is a longint in native byte order) to get the AreaList Pro area reference.

Then you can get the dragged row number using ALP_Area_DragSrcRow (or selection if multiple-row drag is allowed).

For that to work, the 4D object must be droppable, must have enabled On Drop event and must not handle the drop automatically (“Automatic Drop” must not be checked).

```
C_LONGINT ($srcALP)
C_BLOB ($blob)
GET PASTEBOARD DATA ("net.e-node.alp.object";$blob)
If (OK=1)
   $srcALP:=BLOB to longint($blob;Native byte ordering)
   $row:=AL_GetAreaLongProperty ($srcALP;ALP_Area_DragSrcRow)
   // do something
End if
```

Disabling Drag and/or Drop with Read-only mode

It it possible to completely prevent dragging from or dropping to an area using the Read-only mode (ALP_Area_ReadOnly property). The respective bits in this property, if set, will supersede any relevant settings described in this chapter.

See Read-only mode.

13 Advanced Topics

Data Entry Controls

For certain types of data you may want to provide special controls for display and/or data entry - for example, for Boolean values you might want to display check boxes; for date data entry, a popup calendar could be useful for the user, and popup menus can be very useful for entering data from a defined set of values.

AreaList Pro provides the following controls:

Booleans Data Entry

You can choose to display Booleans as check boxes with or without a title, or as radio buttons with the ALP_Column_EntryControl property:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Column_EntryControl	✔	✔	✔	int	0	0	2	Entry control, depending upon column type (boolean or integer/long integer) For boolean columns: 0 = checkbox without title 1 = checkbox with title 2 = radio buttons For integer/long integer columns: 0 = 2-states checkbox (values 0, 1) 1 = 3-states checkbox (values 0, 1, 2) (ALP_Column_DisplayControl must be set to 0, 1, 2 or 4 in order to use checkboxes in integer/long integer columns)

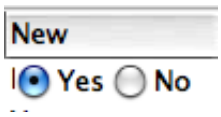
Example 1

To enter the Boolean value in Column 5 via check boxes:

AL_SetColumnLongProperty (area;5;ALP_Column_EntryControl;0)

Example 2

To enter the Boolean value in Column 5 via radio buttons:



First make sure that the column is wide enough to display the radio button labels:

AL_SetColumnRealProperty (area;5;ALP_Column_Width;100)

Then tell AreaList Pro to use radio buttons:

AL_SetColumnLongProperty (area;5;ALP_Column_EntryControl;2)

And finally specify the labels for the radio buttons:

AL_SetColumnTextProperty (area;5;ALP_Column_Format;"Yes;No")

Display

Booleans can be displayed as check boxes (in three sizes) or as custom pictures or text with the ALP_Column_DisplayControl property of ***AL_SetColumnLongProperty:***

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Column_DisplayControl	✔	✔	✔	int	-1	-1	4	Display control type: -1 = default (formatted value) 0 = checkbox without title 1 = small checkbox without title 2 = mini checkbox without title (0, 1 and 2 are identical on Windows) 3 = mapped through ALP_Column_PopupArray +ALP_Column_PopupMap or ALP_Column_PopupMenu (these 3 properties have to be defined) 4 = use pictures (see Displaying custom checkboxes using pictures)

Example 1

To display a Boolean as a normal-sized checkbox in column 5:

AL_SetColumnLongProperty (area;5;ALP_Column_DisplayControl;0)

Example 2

To display a Boolean as "Yes" or "No" in column 5:

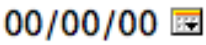
AL_SetColumnLongProperty (area;5;ALP_Column_DisplayControl;-1)

AL_SetColumnTextProperty (area;5;ALP_Column_Format;"Yes;No")

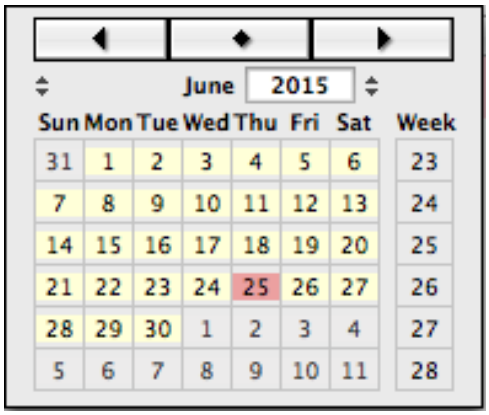
Dates

Popup Date Control

A popup date control appears as a little calendar when the data is being edited:



Click on the calendar icon and a calendar opens:



To jump to the current date, click on the middle button.

Choose the date you want to insert by clicking or double-clicking on it.

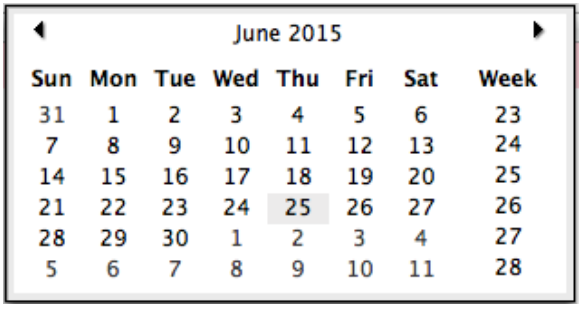
To specify a popup date control, use the "by popup" option of the ALP_Column_Enterable property - for example. to specify a date popup for column 6:

AL_SetColumnLongProperty (area;6;ALP_Column_Enterable;2)

Background and foreground colors can be set for all calendar items using the ALP_Area_CalendarColors property.

Note: on Windows, when the window is zoomed, the date & time popups will not resize the window to its normal state.
The popup windows will be modal and a click outside them will be ignored - use ESC to cancel the popup.

An alternate "Windows" look popup date control is available on both platforms when the ALP_Area_CalendarLook property is set to yes:



In both cases the entry is ended by a double click on a date, or by the “Esc” key or a click on another object (cancels the entry), or by any of the keys used to trigger leaving a cell.

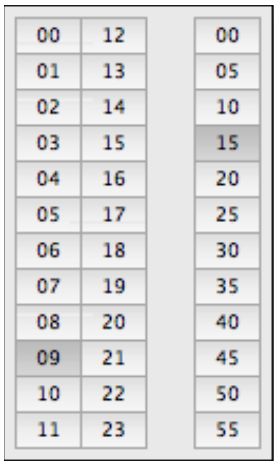
The “Del” key will set the date to the Null value (!00/00/00!) and dismiss the popup.

Time

Popup Time Control

When you specify a popup time control, a little alarm clock icon appears in each cell in the column:

Click on the clock to open the popup time selector:



Choose the hour from the first two columns, and the minutes from the third column. You can also enter the time manually.

To specify a popup time control, use the "by popup" option for the ALP_Column_Enterable property - for example, to specify a popup time control for column 7:

AL_SetColumnLongProperty (area;7;ALP_Column_Enterable;2)

The entry is ended by a double click on a hour or minute button, by the “Esc” key or a click on another object (cancels the entry), or by any of the keys used to trigger leaving a cell.

The “Del” key will set the time to the Null value (?00:00:00?) and dismiss the popup.

Popup Menus

To create a popup menu and associate it with a column in your AreaList Pro area, you create an array containing the required values and then associate that array with the appropriate column. For example, let's say we want to provide four options to choose from in a "Types" column.

First, create the array of values:

```
ARRAY TEXT(atTypes;4)
atTypes{1}:="Chips"
atTypes{2}:="Chocolate"
atTypes{3}:="Nuts"
atTypes{4}:="Toffees"
```

Next, tell AreaList Pro that we want to allow data entry by popup only:

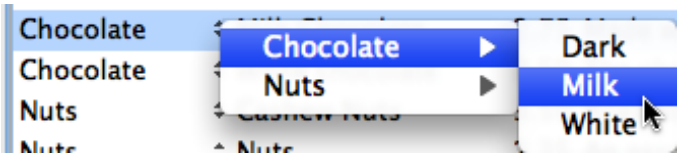
```
AL_SetColumnLongProperty (area;1;ALP_Column_Enterable;2)
```

Finally, assign the array to the popup:

```
AL_SetColumnPtrProperty (area;1;ALP_Column_PopupArray;->atTypes)
```

Hierarchical Popup Menus

You can also use hierarchical popup menus - for example:



Hierarchical popup menus use 4D menus which you create using the **Create menu** command. As an example we will look at how the example shown above was created.

Each menu item must have a menu item parameter defined: this value will be returned to AreaList Pro and stored in the data when the user selects an item from menu.

First we create three menus: the main ("parent" menu) and two submenus (Chocolate and Nuts):

```
$hpopup:=Create menu
$subChoc:=Create menu
$subNuts:=Create menu
```

Next, we populate the two submenus:

```
APPEND MENU ITEM($subChoc;"Dark")
SET MENU ITEM PARAMETER($subChoc;-1;"Dark")
APPEND MENU ITEM($subChoc;"Milk")
SET MENU ITEM PARAMETER($subChoc;-1;"Milk")
APPEND MENU ITEM($subChoc;"White")
SET MENU ITEM PARAMETER($subChoc;-1;"White")
APPEND MENU ITEM($subNuts;"Brazil")
SET MENU ITEM PARAMETER($subNuts;-1;"Brazil")
APPEND MENU ITEM($subNuts;"Macadamia")
SET MENU ITEM PARAMETER($subNuts;-1;"Macadamia")
APPEND MENU ITEM($subNuts;"Mixed")
SET MENU ITEM PARAMETER($subNuts;-1;"Mixed")
```

Add the two submenus to the parent menu:

```
APPEND MENU ITEM($hpopup;"Chocolate";$subChoc)
APPEND MENU ITEM($hpopup;"Nuts";$subNuts)
```

Tell AreaList Pro to apply this menu to column 1 and make it enterable by popup only:

```
AL_SetColumnLongProperty (area;1;ALP_Column_Enterable;2)
AL_SetColumnTextProperty (area;1;ALP_Column_PopupMenu;$hpopup)
```

and, finally, tell AreaList Pro to map the data to the menu titles:

```
AL_SetColumnLongProperty (area;1;ALP_Column_DisplayControl;3)
```

Grids

In addition to displaying fields and arrays in a spreadsheet-style “row and column” layout, AreaList Pro also enables you to display your data in **grids** since Version 9.

Think of a grid as a ***table*** within a ***row*** (that’s “table” in the sense of tabular data, not a database table). This gives you many more ways to present your data.

As an example, compare these two AreaList Pro areas:

List Style

Product Type	Name	Price	Description
Chocolate	Dark Chocolate	2.50	Better for you: dark chocolate has been sh
Chocolate	Milk Chocolate	2.75	Made with full-fat, organic milk.
Chocolate	White chocolate	2.50	The chocolate purist might argue that it's
Nuts	Nuts	2.25	An assortment of peanuts, cashew nuts, et

Grid Style

Type Name Price	Description
Chocolate Dark Chocolate 2.5	Better for you: dark chocolate has been shown to have healthy qualities. How many more reasons do you need?
Chocolate Milk Chocolate 2.75	Made with full-fat, organic milk.
Chocolate White chocolate 2.5	The chocolate purist might argue that it's not really chocolate – but who cares?
Nuts Nuts 2.25	An assortment of peanuts, cashew nuts, etc. Supplied in a decorative blue and red tin.

They both display the same data, but in very different ways.

Terminology

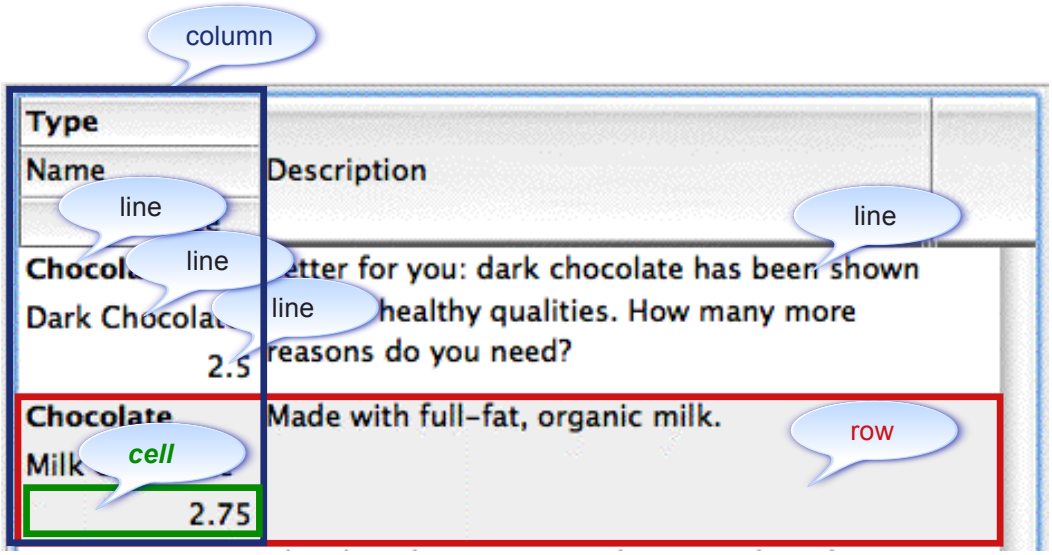
We refer to the parts of a grid as:

line: A field or array that has been added to the area; equivalent to a column in a list view

row: The group of **lines** that are displayed for each record.

column: A column, which may contain any number of **lines**.

cell: The intersection of a **column** and a **line**.



The grid allows you to:

- Specify how lines are grouped into columns.
- Span data across two or more lines, either vertically or horizontally. In this example, the Description field spans the Type, Name, and Price lines, allowing the text to wrap within its cell.
- Hide any line.
- Allow lines to be dragged and dropped.
- Allow the user to sort on any line.

When a column is added or removed, the grid is destroyed.

Any column can be made invisible. Then,

- When in compatibility mode, the grid is destroyed, all columns are made visible, and all columns to be hidden are made invisible
- When no grid is defined by the user (after last destruction), it is created automatically from all visible columns
- When a grid is defined by the developer, it is used and visibility of the columns is modified accordingly.

If you are familiar with how HTML tables are created, a grid works in much the same way (in fact grids are stored as HTML in the area's XML data).

Creating a Grid

To create a grid you first add the fields or arrays to the area in the same way as for a list view, and then you tell AreaList Pro how to organise those columns into the grid.

Building the Grid Array

The grid array is a two-dimensional array that describes how the lines are organised into cells and columns. The first dimension of the array must be 3, and the second dimension will be the number of lines x the number of columns.

The First Dimension

The three elements of the first dimension of the array represent:

1. Line: which data column will be displayed in the line
2. Horizontal span: how many horizontally adjacent cells the data can span
3. Vertical span: how many vertically adjacent cells the data can span

The Second Dimension

The second dimension of the array contains the relevant values: one element for each line.

In our example there will be 6 elements in the second dimension - 2 columns x 3 lines:

Type	Description
Name	
Price	

The array declaration, therefore, will be:

```
ARRAY LONGINT($aiGrid;3;6)
```

Filling the Array

The first dimension of the array contains the column numbers that will comprise our lines. They are filled from left to right and top to bottom:

```
$aiGrid{1}{1}:=1  // Type: the 1st column that was added to the area
$aiGrid{1}{2}:=4  // Description: the 4th column that was added to the area
$aiGrid{1}{3}:=2  // Name: the 2nd column that was added to the area
$aiGrid{1}{5}:=3  // Price: the 3rd column that was added to the area
```

This could be visually represented in this way:

$aiGrid{1}{1}	$aiGrid{1}{2}
$aiGrid{1}{3}	
$aiGrid{1}{5}	

The second and third dimensions specify the row and column spans. Most of these values will be 1, so we can populate the arrays easily:

```
For ($i;1;6)
   $aiGrid{2}{$i}:=1  // column span
   $aiGrid{3}{$i}:=1  // row span
End for
```

We want column 2 (the description) to span 3 lines:

```
$aiGrid{3}{2}:=3  // row span
```

Creating the Grid

Finally, we instruct AreaList Pro to create a grid from our two-dimensional array:

```
AL_SetAreaLongProperty (area;ALP_Area_ColsInGrid;2)  //2 columns in the grid
$err:=AL_SetObjects (area;ALP_Object_Grid;$aiGrid)  //Assign the array to the grid
```

Example

To illustrate how this works we'll look at the code that creates the example shown above.

1. Find the records we want to display, and sort them:

```
ALL RECORDS([product])
ORDER BY([product];[product]product_type;[product]product_name)
```

2. Add the four columns to the area:

```
ARRAY LONGINT($aPtr;4)
$aPtr{1}:=->[product]product_type
$aPtr{2}:=->[product]product_name
$aPtr{3}:=->[product]retail_price
$aPtr{4}:=->[product]description
$err:=AL_SetObjects (area;ALP_Object_Columns;$aPtr)
```

3. Set the attributes for the columns (area, column, width, header text, footer text, format, attributed, calculate height, enterability, bold):

```
AL_SetColumn (area;1;0;"Type";"";"";False;False;0;True)
AL_SetColumn (area;2;0;"Name";"";"";False;False;0;False)
AL_SetColumn (area;3;0;"Price";"";"";False;False;0;False)
AL_SetColumn (area;4;300;"Description";"";"";True;True;1;False)
AL_SetColumn method:
C_LONGINT($1;$2)  //AreaList Pro, column
C_REAL($3)  //width
C_TEXT($t;$4;$5;$6)  //header, footer, format
C_BOOLEAN($7;$8;$10)  //attributed, height, bold
C_LONGINT($9)  //enterability
AL_SetColumnRealProperty ($1;$2;ALP_Column_Width;$3)
AL_SetColumnTextProperty ($1;$2;ALP_Column_HeaderText;$4)
AL_SetColumnTextProperty ($1;$2;ALP_Column_Format;$6)
AL_SetColumnLongProperty ($1;$2;ALP_Column_Attributed;Num($7))
AL_SetColumnLongProperty ($1;$2;ALP_Column_CalcHeight;Num($8))
AL_SetColumnLongProperty ($1;$2;ALP_Column_Enterable;$9)
AL_SetColumnLongProperty ($1;$2;ALP_Column_StyleB;Num($10))
AL_SetColumnLongProperty ($1;$2;ALP_Column_HdrStyleB;Num($10))
```

4. Create a two-dimensional array that describes how the lines are organised into cells and columns:

```
ARRAY LONGINT($aiGrid;3;6)
For ($i;1;6)
   $aiGrid{2}{$i}:=1  // column span
   $aiGrid{3}{$i}:=1  // row span
End for
$aiGrid{1}{1}:=1  // column number
$aiGrid{1}{2}:=4  // column number
$aiGrid{1}{3}:=2  // column number
$aiGrid{1}{5}:=3  // column number
$aiGrid{3}{2}:=3  // line span: Description spans 3 lines
```

5. Create the grid:

```
AL_SetAreaLongProperty (area;ALP_Area_ColsInGrid;2)
$err:=AL_SetObjects (area;ALP_Object_Grid;$aiGrid)
```

6. Do a bit of formatting:

```
AL_SetAreaLongProperty (area;ALP_Area_AltRowOptions;1)  // alternate row background ("zebra" style)
AL_SetAreaLongProperty (area;ALP_Area_ShowFooters;0)  // don't show footers
AL_SetAreaLongProperty (area;ALP_Area_SelMultiple;1)  // select multiple rows
AL_SetAreaLongProperty (area;ALP_Area_AllowSortEditor;1)  // allow sort editor
AL_SetColumnLongProperty (area;4;ALP_Column_Wrap;1)  // 4th column wraps long text
AL_SetAreaLongProperty (area;ALP_Area_ShowRowDividers;1)  // show row dividers
AL_SetAreaLongProperty (area;ALP_Area_NumRowLines;0)  // variable row height
AL_SetAreaLongProperty (area;ALP_Area_EntryClick;2)  // enterable by double-click
```

Re-ordering Rows in a Grid

You can allow users to re-order the rows in a grid using drag and drop - but you need to manage the re-displaying of the grid after a row has been moved.

Grid Properties

Following is a list of the properties specifically for use with grids.

Area Properties

These properties are used with the commands in the Area theme. Grid cell numbers start at 1 and go from left to right and from top to bottom in the grid - for example:

1	2
3	4
5	6

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Area_ColsInGrid	✔	✔	✔	int	-1	-1		Number of columns in grid
ALP_Area_ColsLocked	✔	✔	✔	int	0	0		Number of locked columns in grid
ALP_Area_RowsInGrid	✔	✔	✔	int	1	-1	20	Number of rows in grid
ALP_Area_EntryGotoGridCell	✔	✔		int				Grid cell number to start entry in (cell in grid, not column number)
ALP_Area_EntryGridCell	✔			int				Grid cell number of edited cell
ALP_Area_EntryPrevGridCell	✔			int				Previously edited grid cell number

Column Properties

These properties are used with commands in the Columns theme.

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Column_FindCell	✔	✔	✔	int				Find the first grid cell number showing data from the column
ALP_Column_FromCell	✔			int				Get the column number from the grid cell number

Object Properties

These properties are used with commands in the Objects theme.

Constant	Get	Set	Array Type	Comments
ALP_Object_FooterTextNH	✔	✔	text	Footer text of visible columns in grid order
ALP_Object_Grid	✔		int	Column numbers Use a 2D array to access colSpan & rowSpan, too

Hierarchical Lists

Hierarchical lists are an excellent way to display data that is organised into groups and sub-groups. Consider these two examples:

List Style

Product Type	Name	Price	Description
Chocolate	Dark Chocolate	2.50	Better for you: dark chocolate has been s
Chocolate	Milk Chocolate	2.75	Made with full-fat, organic milk.
Chocolate	White chocolate	2.50	The chocolate purist might argue that it'
Nuts	Nuts	2.25	An assortment of peanuts, cashew nuts,

Hierarchical List Style

Type	Name	Description
Chocolate		
Nuts		
Toffee		

In the second example, the data has been organised into hierarchical lists. You can click on a triangle to expand a list - for example:

Type	Name	Description
Chocolate		
	Dark Chocolate	Better for you: dark chocolate has b
	Milk Chocolate	Made with full-fat, organic milk.
	White chocolate	The chocolate purist might argue th
Nuts		
	Cashew Nuts	Roasted and salted cashew nuts in a
	Nuts	An assortment of peanuts, cashew r
Toffee		

How to create a Hierarchical List

In Version 10 we added a very simple way to display an area as a hierarchical list. It requires just one line of code:

```
AL_SetAreaLongProperty(eList; ALP_Area_AutoHierarchy; 1)
```

This will use break processing to create collapsable break headers.

A slightly more complex version provides a way to turn the hieararchical lsiplay on and off:

```
If (AL_GetAreaLongProperty(eList; ALP_Area_AutoHierarchy)=0)
   AL_SetAreaLongProperty(eList; ALP_Area_SortList; 1)  // sort on first column
   AL_SetAreaLongProperty(eList; ALP_Area_AutoHierarchy; 1)  // turn on break processing & hierarchy on first column
   AL_SetAreaLongProperty(eList; ALP_Area_FillCache; 0)  // force ALP to create the breaks & hierarchy
   AL_SetRowLongProperty(eList; 0; ALP_Row_CollapseAll; 0)  // collapse all rows
   $rowID:=AL_GetBreakLongProperty(eList; 1; 1; 0; ALP_Break_RowID)  // get break header row number
   AL_SetRowLongProperty(eList; $rowID; ALP_Row_StyleB; 1)  //root level rows in bold
Else
   AL_SetAreaLongProperty(eList; ALP_Area_AutoHierarchy; 0)  // turn off break processing & hierarchy
End if
```

See the example in the Examples database, and the ALP_Area_AutoHierarchy property description.

Hierarchical List Properties

The following properties are specifically for use with hierarchical lists.

Area Properties

These properties are used with the commands in the Area theme.

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Area_ArrowsForHierarchy	✔	✔		bool	0			When hierarchy is displayed, left/right arrow keys (without command key) are used to collapse/ expand nodes, not for horizontal scrolling
ALP_Area_HierIndent	✔	✔	✔	real	16	0	64	Indent increment for every hierarchy level (for use with hierarchical lists)

Row Hierarchy Properties

These properties are used with commands in the Rows theme.

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Row_Collapse	✔	✔		bool				Collapse this row (all children will be invisible)
ALP_Row_CollapseAll		✔		bool				“Deep collapse”: collapse this row and all its children (all children will be invisible and collapsed)
ALP_Row_Expand	✔	✔		bool				Show children of this row If any child was not collapsed, more levels will be visible
ALP_Row_ExpandAll		✔		bool				«Deep expand»: show children of this row and all children (all children will be visible and fully expanded)
ALP_Row_Level	✔			int				Returns the level associated with this row (set using ALP_Object_Hierarchy)
ALP_Row_Parent	✔			int				Returns the immediate parent of this row (zero if this row is at the top level)
ALP_Row_Visible	✔			bool				Returns whether this row is visible (all parents of this row are expanded) This has nothing to do with real visibility on screen, but with the expanded state of all parents

Object Properties

This property is used with commands in the Objects theme.

Constant	Get	Set	Array Type	Comments
ALP_Object_Hierarchy	✔	✔	int	Hierarchy: level, expanded 2D or two arrays: you can call ***AL_SetObjects*** with a 2-dimensional array or ***AL_SetObjects2*** with two arrays

Pictures

Images can be displayed as icons in cells and in headers and footers, and they can be set either on the left or the right of the cell.

The pictures can be stored in variables, fields, or the 4D picture library.

Various settings can be applied to an icon through the use of flags - for example, scaling, offset from the border, etc.

Formatting picture columns

AreaList Pro uses constants for picture alignment. They are identical to previous AreaList Pro versions:

Constant	Value
AL Truncated upper left	0
AL Truncated centered	1
AL Scaled to fit	2
AL Scaled proportional	3
AL Scaled prop centered	4

For instance, to set the format of a picture column to be “scaled to fit prop centered”, use

AL_SetColumnTextProperty (area;$columnIndex;ALP_Column_Format;"4")

- format 0 is "Truncated" — the picture is always aligned, e.g. if the picture is bigger, you should see the right side of the picture for a right-aligned picture
- format 1 is "Truncated (centered)" — the picture is always centered, which is the same as format 0 with horizontal and vertical alignment set to centered (2)
- format 2 is "Scaled to fit" — it always covers the full cell, there is nothing to align
- format 3 is "Scaled proportionally" — if the picture is smaller than the cell (after scaling the picture), it will be aligned
- format 4 is "Scaled proportionally (centered)" — the picture is always scaled and centered, which is the same as format 3 with horizontal and vertical alignments set to centered (2)

In addition:

- if the value set for ALP_Column_Format is not specified or out of range, it 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 justification if any was set with one of the alignment properties (ALP_Column_HorAlign, ALP_Row_HorAlign, ALP_Cell_HorAlign, ALP_Column_VertAlign), default alignment is top left for these two formats

These alignement properties should not be confused with the icon flags (see below).

AL_SetCellLongProperty (…;ALP_Cell_LeftIconFlags;AL Icon Flags Scaled)

is related to pictures (icons) embedded into a cell (in any kind of column), not for column data as such.

Using a picture from a field or variable

To display an image stored in a variable or field, you call AL_SetIcon. ***AL_SetIcon*** accepts the area reference (or zero for global workstation settings), an image ID (which you create), and the image. The ID must be a positive number between 1 and 16,777,215.

You can then use the image ID to set the value of the ALP_Cell_LeftIconID / ALP_Cell_RightIconID properties.

Using a picture from the 4D Picture Library

To use a picture from the 4D Picture Library, you pass the appropriate 4D Library picture reference to ALP_Cell_LeftIconID/ ALP_Cell_RightIconID.

The picture will be automatically added to AreaList Pro's picture library (cache) if not already there. See the note below regarding project mode.

Displaying custom checkboxes using pictures

The ALP_Column_DisplayControl property can be set to 4 meaning "draw pictures".

Boolean, Integer and Long integer arrays/fields can be shown as pictures.

Set the format (ALP_Column_Format) to "TrueID;FalseID" or "TrueID;FalseID;MixedID" for 3-state entry, as controlled by ALP_Column_EntryControl: 1 means use 3-state (same as when drawing native checkboxes).

If the picture is not present in AreaList Pro's cache (has not been set by AL_SetIcon), AreaList Pro tries to get it from the 4D's picture library. If the respective picture ID in the format is not present (or is zero), nothing will be drawn.

For example, "12345" will draw picture 12345 for True/1 and nothing otherwise and ";4321" will draw picture 4321 for False/0 and nothing otherwise.

Flags

Flags can be used to set various properties for the icons, such as offset, scaling, and alignment.

Icons are loaded from AreaList Pro's picture library (a.k.a. picture cache, populated with ***AL_SetIcon***) or from 4D's Picture library (with ALP_Cell_LeftIconID/ ALP_Cell_RightIconID)

The following flags can be used with the ALP_Cell_LeftIconFlags or ALP_Cell_RightIconFlags properties:

Flag	Description
offset/width	A number between 0 - 255 Offset from the cell border in points, or icon width (depending on the horizontal position) See Alignment and offset
horizontal position	0 - default position (left for left icon, right for right icon) 256 - AL Icon Flags Horizontal Left - Align on the left 512 - AL Icon Flags Horizontal Center - Horizontally centered 768 - AL Icon Flags Horizontal Right - Align on the right
vertical position	0 - default position (top) 1024 - AL Icon Flags Vertical Top - Align top 2048 - AL Icon Flags Vertical Center - Vertically centered 3072 - AL Icon Flags Vertical Bottom - Align bottom
scaling	0 - default (trimmed) - Picture is displayed trimmed to row height. 4096 - AL Icon Flags Scaled - Picture is displayed scaled to fit to the height of the row (height of the icon is defined by height of the row and picture is proportionally scaled)
mask	AL Icon Flags Horizontal Mask AL Icon Flags Vertical Mask AL Icon Flags Offset Mask Masks are used to extract a partial value from a combined value. For example, if you want to extract the offset from the value (received with a getter) you will do: $flags:=***AL_GetCellLongValue***(area; ALP_Cell_LeftIconFlags) offset:=$flags & AL Icon Flags Offset Mask

Examples

Example 1

Get a picture of a dollar sign from a field and display it in the header row for the Price column:

```
C_PICTURE($pPict)
QUERY([pictures];[pictures]picturename="dollar")
$pPict:=[pictures]pic
$err:=AL_SetIcon (area;20;$pPict)  // add picture to the AreaList Pro picture library / cache with ID = 20
AL_SetCellLongProperty (area;0;3;ALP_Cell_LeftIconID;20)  // add icon to column 3 in the header
```

Example 2

Using a picture from the 4D picture library:

To display a picture from the 4D Picture Library you simply need to pass the Picture Library ID number:

```
AL_SetCellLongProperty (area;0;3;ALP_Cell_LeftIconID;2072)  // use picture no. 2072
```

Example 3

Display an icon in the third column's header; center the icon horizontally and position it at the bottom of the cell:

```
C_PICTURE($pPict)
QUERY([pictures];[pictures]picturename="dollar")
$pPict:=[pictures]pic
$err:=AL_SetIcon (area;20;$pPict)
AL_SetCellLongProperty (area;0;3;ALP_Cell_LeftIconID;20)
$flags:=AL Icon Flags Horizontal Center+AL Icon Flags Vertical Bottom
AL_SetCellLongProperty (area;0;3;ALP_Cell_LeftIconFlags;$flags)
```

Example 4

Set an offset of 5 points from the cell border for column 3's header:

```
AL_SetCellLongProperty (area;0;3;ALP_Cell_LeftIconFlags;5)
```

Example 5

Display 3-state custom checkboxes from pictures in a long integer column:

```
$err:=AL_SetIcon (area;33;$pPictYes)  // set picture for the True value with ID = 33
$err:=AL_SetIcon (area;34;$pPictNo)  // set picture for the False value with ID = 34
$err:=AL_SetIcon (area;35;$pPictMaybe)  // set picture for the Mixed value with ID = 35
AL_SetColumnLongProperty (area;3;ALP_Column_EntryControl;1)  // use three-state entry
AL_SetColumnLongProperty (area;3;ALP_Column_DisplayControl;4)  // display pictures
AL_SetColumnTextProperty (area;3;ALP_Column_Format;"33;34;35")  // specify column format
```

Note: if ***AL_SetIcon*** is not called, AreaList Pro will look for pictures ID 33, 34 and 35 from 4D's picture library. **However** be aware that 4D Project mode does not support the Picture Library: using this mode, you need to store the pictures on disk (generally in the Resources folder) and explicitly call ***AL_SetIcon***. See Example 16.

Alignment and offset

AreaList Pro uses a specific offset/width implementation for icon drawing.

Offset

When horizontal alignment in ALP_Cell_XXXIconFlags is zero (default position : left for left icon, right for right icon), the low 8 bits (offset/width) of ALP_Cell_XXXIconFlags are interpreted as the offset, i.e. the distance in points between the text and the icon (left or right):

0 - default position (left for left icon, right for right icon)

Width

Otherwise the low 8 bits (offset/width) of ALP_Cell_XXXIconFlags are interpreted as the point width that the icon will use - the icon will be aligned in this space:

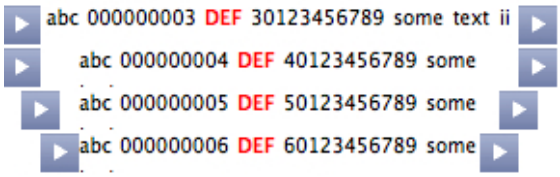
512 - AL Icon Flags Horizontal Center - Horizontally centered

Example

```
C_LONGINT(vIconOffset;vIconHPos;vIconVPos;vIconFmt;vIconID;vIconWidth)
vIconID:=11
vIconOffset:=3
vIconFmt:=0  //AL Icon Flags Scaled
vIconHPos:=0  //default alignment
vIconVPos:=AL Icon Flags Vertical Center
AL_SetCellLongProperty ($area;3;7;ALP_Cell_LeftIconID;vIconID)
AL_SetCellLongProperty ($area;3;7;ALP_Cell_LeftIconFlags;vIconOffset | vIconFmt | vIconHPos | vIconVPos)
AL_SetCellLongProperty ($area;3;7;ALP_Cell_RightIconID;vIconID)
AL_SetCellLongProperty ($area;3;7;ALP_Cell_RightIconFlags;vIconOffset | vIconFmt | vIconHPos | vIconVPos)
vIconWidth:=40
vIconHPos:=AL Icon Flags Horizontal Left
AL_SetCellLongProperty ($area;4;7;ALP_Cell_LeftIconID;vIconID)
AL_SetCellLongProperty ($area;4;7;ALP_Cell_LeftIconFlags;vIconWidth | vIconFmt | vIconHPos | vIconVPos)
vIconHPos:=AL Icon Flags Horizontal Right
AL_SetCellLongProperty ($area;4;7;ALP_Cell_RightIconID;vIconID)
```

AL_SetCellLongProperty ($area;4;7;ALP_Cell_RightIconFlags;vIconWidth | vIconFmt | vIconHPos | vIconVPos)
vIconHPos:=AL Icon Flags Horizontal Center
AL_SetCellLongProperty ($area;5;7;ALP_Cell_LeftIconID;vIconID)
AL_SetCellLongProperty ($area;5;7;ALP_Cell_LeftIconFlags;vIconWidth | vIconFmt | vIconHPos | vIconVPos)
AL_SetCellLongProperty ($area;5;7;ALP_Cell_RightIconID;vIconID)
AL_SetCellLongProperty ($area;5;7;ALP_Cell_RightIconFlags;vIconWidth | vIconFmt | vIconHPos | vIconVPos)
vIconHPos:=AL Icon Flags Horizontal Right
AL_SetCellLongProperty ($area;6;7;ALP_Cell_LeftIconID;vIconID)
AL_SetCellLongProperty ($area;6;7;ALP_Cell_LeftIconFlags;vIconWidth | vIconFmt | vIconHPos | vIconVPos)
vIconHPos:=AL Icon Flags Horizontal Left
AL_SetCellLongProperty ($area;6;7;ALP_Cell_RightIconID;vIconID)
AL_SetCellLongProperty ($area;6;7;ALP_Cell_RightIconFlags;vIconWidth | vIconFmt | vIconHPos | vIconVPos)

Here is the resulting display for rows 3-6:



Displaying custom pictures instead of AreaList Pro's native icons

AreaList Pro's native icons for popups and hierarchical lists can be replaced with your own custom pictures.

Setting custom icons

This is performed through the AL_SetIcon command: if the referenced icon is present in AreaList Pro's global picture library / cache (set by ***AL_SetIcon*** with area reference = 0), it will be used, superseding internal icons (popups, hierarchical list triangles) and per-area picture library / cache (set by ***AL_SetIcon*** with specified area reference).

Internal icon IDs and widths

Use the following values in the **iconID** parameter of ***AL_SetIcon*** in order to use your own pictures.

- Generic popup on MacOS : ID = 1, width = 5
- Generic popup On Windows: ID = 1, width = 7
- Date popup : ID = 2, width = 15
- Time popup : ID = 3, width = 16
- HL node right: ID = 4, width = 9
- HL node down: ID = 5, width = 9

Value Mapping

Value Mapping is a way to map one set of values to another for display purposes. Suppose that each record in your database has a numeric value stored in a field.

This numeric value is meaningful to your program, but meaningless to a human: it would be much more useful for the user to know what those numeric values actually mean.

One way to handle the problem would be to create a new text array, loop through all the records (or array elements), and populate the text array according to the values found in the numeric array. Value mapping is a much more efficient way to handle the situation.

AreaList Pro needs two sets of values: one set of values that are compared to values stored in a field or array, and another set of values that are displayed in the column.

In the case of two arrays, stored values are looked up in the array passed in ALP_Column_PopupArray property and displayed values are from the ALP_Column_PopupMap array.

In the case of ALP_Column_PopupMenu, stored values are looked up in menu item parameters and displayed values are menu titles.

The column that uses mapping does not need to be enterable by popup.

The best way to create a value map is using a combination of the ALP_Column_PopupArray and ALP_Column_PopupMap properties. Both properties are persistent, and the mapping is done directly in AreaList Pro; AreaList Pro creates a 4D menu for the popup, which is accessible through ALP_Column_PopupMenu (which is **not** persistent)

Another way is to use a developer-supplied 4D Menu. In this case, AreaList Pro has to call 4D to get the elements of the menu to find the value/title pairs. Using this method, you can make it hierarchical.

Example 1: Mapping using 4D's menu

```
<>mMenu:=Create menu
APPEND MENU ITEM(<>mMenu;"Area")
SET MENU ITEM PARAMETER(<>mMenu;1;"1")
APPEND MENU ITEM(<>mMenu;"Column")
SET MENU ITEM PARAMETER(<>mMenu;2;"2")
APPEND MENU ITEM(<>mMenu;"Row")
SET MENU ITEM PARAMETER(<>mMenu;3;"3")
APPEND MENU ITEM(<>mMenu;"Cell")
SET MENU ITEM PARAMETER(<>mMenu;4;"4")
APPEND MENU ITEM(<>mMenu;"-")
APPEND MENU ITEM(<>mMenu;"AreaObject")
SET MENU ITEM PARAMETER(<>mMenu;6;"5")
APPEND MENU ITEM(<>mMenu;"DropArea")
SET MENU ITEM PARAMETER(<>mMenu;7;"6")
AL_SetColumnTextProperty (exALP;3;ALP_Column_PopupMenu;<>mMenu)
```

Note: <>mMenu belongs to 4D; AreaList Pro does not release it.

Example 2: Mapping using PopupArray/PopupMap

First, create an array of values:

```
ARRAY LONGINT($alpopup;7)
$alpopup{1}:=1
$alpopup{2}:=2
$alpopup{3}:=3
$alpopup{4}:=4
$alpopup{5}:=0
$alpopup{6}:=5
$alpopup{7}:=6
$err:=AL_SetColumnPtrProperty (exALP;3;ALP_Column_PopupArray;->$alpopup)
```

Next, either create the map from an array:

```
ARRAY TEXT($menu;7)
$menu{1}:="Area"
$menu{2}:="Column"
$menu{3}:="Row"
$menu{4}:="Cell"
$menu{5}:="-"
$menu{6}:="AreaObject"
$menu{7}:="DropArea"
$err:=AL_SetColumnPtrProperty (exALP;3;ALP_Column_PopupMap;->$menu)
```

… or create it from text:

```
AL_SetColumnTextProperty (exALP;3;ALP_Column_PopupMap;"Area"+Char(3)+ Column"+Char(3)+"Row"+Char(3)+"Cell"+
Char(3)+"-"+Char(3)+"AreaObject"+Char(3)+"DropArea")
```

Note: a 4D menu created by AreaList Pro belongs to AreaList Pro; AreaList Pro will release it.

It is accessible using:

```
AL_GetColumnTextProperty (exALP;3;ALP_Column_PopupMenu)
```

And then just use that menu:

```
AL_SetColumnLongProperty (exALP;3;ALP_Column_DisplayControl;3)
// column shows popup values instead of direct values
AL_SetColumnLongProperty (exALP;3;ALP_Column_Enterable;AL Column entry popup only)
// allow entry using popup
```

XML

Every AreaList Pro setting has an XML name, or tag, and you can save an area's settings into a field or variable which can then be loaded and applied to any AreaList Pro area.

In this way you can create AreaList Pro "templates" which can be used on various layouts or even transferred to other databases.

See the AL_Load and AL_Save commands for more information about saving and loading XML.

This feature can be very useful if you want to distribute AreaList Pro area settings without having to recompile your application: simply set up the area the way you want it, save the settings to XML, and send that XML to another user.

Note: not all defined XML tags are saved – only properties that were set to values other than the default values will be included.

You can find a complete list of XML tags in the Property Values, Constants and XML Names section.

It is also possible to set XML values through properties such as ALP_Area_XML, ALP_Drop_XML, ALP_Column_XML, ALP_Row_XML, ALP_Row_StyleXML, or ALP_Cell_XML.

Note that the main difference between ***AL_Load*** and setting the XML directly is in columns: setting XML does not clear/add columns.

14 Commands by Theme

Using the Command Reference

Each AreaList Pro command is described in detail in this section. Each description contains the following elements:

Name of the command

AL_AddColumn

Parameters

Result

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

Descriptions of the parameters

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout or global setting.
➞ dataPointer	pointer	When in Records mode, DataPointer should be a pointer to field. When in Arrays mode, DataPointer should be a pointer to an array (must not be a local array!).
➞ insertAt	longint	Position at which to insert a column; 0 means add to the end.
← result	longint	

A description of this command

Add a column at the specified position (**insertAt**).

The column can be either a field - in which case you pass a pointer to the field in the **dataPointer** parameter - or an array - in which case you pass a pointer to the array in the **dataPointer** parameter.

In addition to passing one-dimensional arrays, you can also pass the first element of a two-dimensional array. In this case, the first dimension relates to columns and the second dimension relates to rows (see the example below).

Example

One or more examples showing how to use the command

This example adds three columns to an AreaList Pro area r

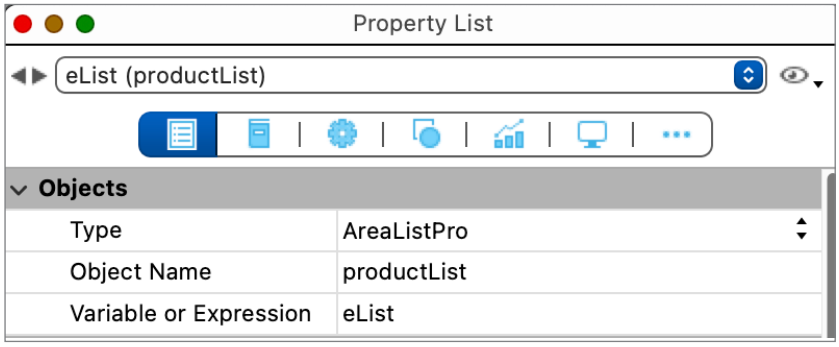
```
$err:=AL_AddColumn (area;->[product]product_code;0)
$err:=AL_AddColumn (area;->[product]product_name;0)
$err:=AL_AddColumn (area;->[product]product_type;0)
```

Name of the command

This is what tells AreaList Pro what you want to do; the command name must always be entered exactly as shown.

Parameters

Every command requires at least one parameter. Most require the first parameter to be the area reference: this is the **Variable Name** that you assigned to the AreaList Pro area that you want the command to affect:



Global Settings and Templates for Areas, Columns, Rows and Cells

Since AreaList Pro Version 10, you can use the area reference to apply global style changes to an area, and you can use these options to create area templates.

The options available are:

Global Settings for Areas

Area Reference	Description
0	Apply to all areas to be created (defining an area “template”)
-2	Apply to all existing areas, but not to areas to be created
-3	Apply to all existing areas in current process only
-4	Apply to all existing areas, and to areas to be created

Global Settings for Columns

Column Number	Description
0	Apply to all columns to be created (defining a column “template”) in either a specific area, or in area 0, -2, -3 or -4 as above (changing Column 0 of Area 0 does not affect newly created columns in existing areas!)
-2	Apply to all existing columns in either a specific area, or in area -2 or -3 as above for existing areas

Global Settings for Rows

Row Number	Description
-2	Apply to all rows in either a specific area, or in area -2 or -3 as above for existing areas

Example

You want to enable users to set a preference as to the font size in all existing AreaList Pro areas.

After the user chooses their preferred font size, you can use:

$fontSize:= <font size selected by the user>

$result:=***AL_SetAreaRealProperty*** (-2;-2;ALP_Row_Size; ->$fontSize)

Global Settings for Cells

Number	Description
Row -2	Apply to all rows displaying data for the specified Column Number
Column -2	Apply to all columns in the area for the specified Row Number
Row -2 Column -2	Apply to all cells in the area

Example

Clear all cell options in all areas created in this process (including offscreen areas)

$result:=***AL_SetCellLongProperty*** (-3; -2; -2;ALP_Cell_Clear)

Result

Functions return a result after they have been called. Unless otherwise specified in the parameter description table, the result codes are long integers and have the following meanings:

Result Code	Description
-1	Generic error
0	No error - the command executed successfully
1	Can’t load XML
2	Can’t save XML
3	Invalid area reference
4	Invalid object reference
5	Invalid request
6	Invalid array type
7	Invalid nil pointer
8	Invalid pointer type
9	Invalid array size
10	Can’t load record
11	Can’t save record

Parameter Descriptions

Each command has its own set of parameters, and they are each described in the parameter descriptions table. The tables comprise three columns: Parameter, Type, and Description.

Parameter: The name of the parameter, as shown in the Parameter list. Each is preceded by one of two arrows which indicate whether it is a value that you pass to the command or one that the command returns to you:

➞ Area	A value that you pass to the command
← Array	A value that is returned by the command

Type: The type of the parameter.

Note: if your database is running in non-Unicode mode, text objects are limited to 32k characters.

Description: Information about the parameter.

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

Command Description

An explanation of what the command does and how to use it.

Examples

One or more examples demonstrating how the command might be used.

Command Themes

The commands are organised into seven themes:

Area: Commands that affect the entire AreaList Pro area

Columns: Commands that affect columns

Rows: Commands that affect rows

Cells: Commands that affect individual cells

Objects: Commands that affect AreaList Pro objects

Break Processing: Commands to manage Break processing

Utility: Miscellaneous commands such as AreaList Pro licence registration

For each theme there is a set of properties that can be used with that theme's commands. You will find a complete list of properties in the Properties by Theme section.

Area

The commands in this theme affect the overall AreaList Pro area.

The properties that can be used with these commands can be found in the AreaList Pro Area Properties and the AreaList Pro Drop Area themes.

See Global Area settings.

If you access workstation-only properties (properties not specific to areas, called Plugin properties, such as ALP_Area_TraceOnError or ALP_Area_Version), AreaRef is ignored.

AL_GetAreaLongProperty

(areaRef:L; property:T) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to get.
← result	longint	Value of the “got” property.

Get details of an area’s **property**. The properties that can be used with this command are the ones of type “longint” and “boolean” (1 or 0) listed in the AreaList Pro Area Properties section.

Example

After a drag-and drop operation, you need to find out which row from the source area was dragged:

```
$SourceRow:=AL_GetAreaLongProperty ($DropArea;ALP_Area_DragSrcRow)
```

AL_GetAreaPtrProperty

(areaRef:L; property:T; pointer:Z) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to get.
← pointer	pointer	Pointer to variable to hold the result; the variable must be initialized to the correct type before calling this function.
← result	longint	

Get details of an area’s **property**.

The properties that can be used with this command are listed in the AreaList Pro Area Properties section.

Example 1

To find out the number of columns in an area:

```
C_LONGINT($numColumns)
$err:=AL_GetAreaPtrProperty(area; ALP_Area_Columns;->$numColumns)
```

Example 2

To get the current entity being displayed:

```
C_OBJECT($entity)
$err:=AL_GetAreaPtrProperty ($area;ALP_Area_CurrentEntity;->$entity)
```

Example 3

To get the current entity being edited:

```
C_OBJECT($entity)
$err:=AL_GetAreaPtrProperty ($area;ALP_Area_EntryEditedEntity;->$entity)
```

Related entities for entitySelection are supported ("relCustomer.Name" while displaying Projects related to Customers) and are enterable.

AL_GetAreaRealProperty

(areaRef:L; property:T) ➞ result:R

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to get.
⬅ result	real	Value of the “got” property.

Get details of an area’s **property**. The properties that can be used with this command are the ones of type “real” listed in the AreaList Pro Area Properties section.

Example

To find out the current position of the horizontal scroll bar:

```
$ScrollPos:=AL_GetAreaRealProperty (area;ALP_Area_ScrollLeft)
```

AL_GetAreaTextProperty

(areaRef:L; property:T) ➞ result:T

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to get.
⬅ result	text	Value of the “got” property.

Get details of an area’s **property**. The properties that can be used with this command are the ones of type “text” listed in the AreaList Pro Area Properties section.

Example

To get a list of the column numbers in their currently sorted order:

$SortList:=***AL_GetAreaTextProperty*** (area;ALP_Area_SortList)

AL_SetAreaLongProperty

(areaRef:L; property:T; value:L)

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to set.
➞ value	longint	The value to set the property with (a long integer).

Set a specific **property** for an area. The properties that can be set with this command are the ones of type “longint” and “boolean” (1 or 0) listed in the AreaList Pro Area Properties section.

Example

To start data entry in the first row of the second column:

AL_SetAreaLongProperty (area;ALP_Area_EntryGotoColumn;2)

AL_SetAreaLongProperty (area;ALP_Area_EntryGotoRow;1)

AL_SetAreaPtrProperty

(areaRef:L; property:T; pointer:Z) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to set.
➞ pointer	pointer	Pointer to a variable that holds a value to pass to the function.
⬅ result	longint	

Set a specific **property** for an area. The properties that can be set with this command are listed in the AreaList Pro Area Properties section.

This version of ***AL_SetAreaProperty*** allows you to write generic code that uses a **pointer** to any type of variable.

Example 1

To show the sort editor:

```
$showSortEditor:=1
$error:=AL_SetAreaPtrProperty (area;ALP_Area_ShowSortEditor;->$showSortEditor)
```

Example 2

Setting area to display a Collection:

```
$error:=AL_SetAreaPtrProperty (area;ALP_Area_Collection;->$co) // ALP_Area_DataSource is set to 3 if successful
```

Example 3

Setting area to display an Entity Selection:

```
$error:=AL_SetAreaPtrProperty (area;ALP_Area_EntitySelection;->$co) // ALP_Area_DataSource is set to 4 if successful
```

AL_SetAreaRealProperty

(areaRef:L; property:T; value:R)

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to set.
➞ value	real	The value to set the property with a real number.

Set a specific **property** for an area. The properties that can be set with this command are the ones of type “real” listed in the AreaList Pro Area Properties section.

Example

To set the horizontal indent (padding) for the header row to 4 points:

AL_SetAreaRealProperty (area;ALP_Area_HdrIndentH;4)

AL_SetAreaTextProperty

(areaRef:L; property:T; value:T)

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to set.
➞ value	text	The value to set the property with (text).

Set a specific **property** for an area. The properties that can be set with this command are the ones of type “text” listed in the AreaList Pro Area Properties section.

Example

You can create your own text prompt for the AreaList Pro sort editor:

AL_SetAreaTextProperty (area;ALP_Area_SortPrompt;"My custom prompt message")

AL_SuperReport

(areaRef:L; template:T; options:L; styleOptions:L; title:T) ➞ result:T

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ template	text	XML SuperReport template or full path to a XML template or empty to use AreaList Pro’s built-in template.
➞ options	longint	0 = use current columns widths; 1 = use developer or user-defined width.
➞ styleOptions	longint	Style properties that should **not** be overtaken by AreaList Pro - see constants in SuperReport Pro manual, Style Features.
➞ title	text	Optional text centered in the header.
← result	text	

Fills a SuperReport Pro report with the area information for printing. See the Printing with SuperReport Pro section.

Columns

The commands in this theme affect columns within the AreaList Pro area.

The properties that can be used with these commands can be found in the AreaList Pro Column Properties theme.

AL_AddCalculatedColumn

(areaRef:L; dataType:L; callbackMethodName:T; insertAt:L) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ dataType	longint	The array type to use:
➞ callbackMethodName	text	Name of the method to execute to fill the array. The following parameters will automatically be passed to the callback method: $1: AreaList Pro area - longint $2: column - Longint $3: type - Longint $4: pointer to temporary 4D array $5: first - Longint: first record for which to calculate cell $6: count - number of cells to calculate in the column
➞ insertAt	longint	Position at which to insert a column; 0 means add to the end.
← result	longint	

Array Type	Constant	Value
Alpha	Is Alpha Field	0
Boolean	Is Boolean	6
Date	Is Date	4
Integer	Is Integer	8
Longint	Is Longint	9
Picture	Is Picture	3 or 10
Real	Is Real	1
Text	Is Text	2
Time	Is Time	11
Universal date/time		Mapped to 2 (Text)

Add a calculated column after the last column or at the specified position.

Note: this command is only useful in field mode.

Example

In our database we have retail prices for our products. However, account holders in different levels receive a discount of between 5 and 15%. We want to display the prices with the appropriate discount for the account holder's level. So instead of adding the Price column, we can add a calculated column.

The actual calculation is done in a callback method whose name you pass when you add the column.

First, we create our callback method:

```
// CalculateDiscountPrice
// Callback method to calculate discount prices
C_LONGINT($1;$2;$3;$5;$6)  // must be declared
C_POINTER($4)  // this must be declared
SELECTION RANGE TO ARRAY($5;$5+$6-1;[product]retail_price;$price)
For ($i;1;$6)
   Case of
      : (<>DiscLevel=1)  // <>DiscLevel was set when the user logged in
         $4->{$i}:=Round($price{$i}*0.95;2)  // 5% discount
      : (<>DiscLevel=2)
         $4->{$i}:=Round($price{$i}*0.9;2)  // 10% discount
      : (<>DiscLevel=3)
         $4->{$i}:=Round($price{$i}*0.85;2)  // 15% discount
   End case
End for
```

Now all we need to do is add the calculated column to our AreaList Pro area:

```
$err:=AL_AddCalculatedColumn (area;Is real;"CalculateDiscountPrice")
```

AL_AddColumn

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

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ dataPointer	pointer	When in records / fields mode, should be a pointer to field. When in arrays mode, should be a pointer to an array (must not be a local array!).
➞ insertAt	longint	Position at which to insert a column; 0 means add to the end.
← result	longint	

Add a column at the specified position (**insertAt**).

The column can be either a field - in which case you pass a pointer to the field in the **dataPointer** parameter - or an array - in which case you pass a pointer to the array in the **dataPointer** parameter.

In field mode, when all columns are from a related table, the area display is based upon the current selection from the master table, not the related table's selection.

Versions 9.9.2 and above allow any kind of one-dimension array to be added as a column (pointer, blob, object): it is not usable for display, but this feature can be used to maintain the arrays in sync (using sort or AL_ModifyArrays).

In addition to passing one-dimensional arrays, you can also pass the first element of a two-dimensional array. In this case, the first dimension relates to columns and the second dimension relates to rows (see the example below).

Example 1

This example adds three columns to an AreaList Pro area referenced as area:

```
$err:=AL_AddColumn (area;->[product]product_code;0)
$err:=AL_AddColumn (area;->[product]product_name;0)
$err:=AL_AddColumn (area;->[product]product_type;0)
```

Example 2: Using Two-Dimensional Arrays

In this example we're going to use a 2-D array to create the columns and rows in the area. The first dimension of the array will create the columns and the second dimension creates the rows.

In other words, if the array has 2 in the first dimension and 4 in the second dimension (e.g. anArray;2;4) there will be two columns and four rows.

The disadvantage of using two-dimensional arrays is that every column must be the same data type.

This example will create an area of n rows (n being the number of products) and two columns:

```
ALL RECORDS([product])
$Records:=Records in selection([product])
ARRAY TEXT(ArrayValues;2;$Records)
For ($i;1;$Records)
   ArrayValues{1}{$i}:=[product]product_name
   ArrayValues{2}{$i}:=[product]product_code
   NEXT RECORD([product])
End for
For ($i;1; Size of array(ArrayValues))
   $err:=AL_AddColumn ($1;->ArrayValues{$i};0)
End for
```

AL_AddEntityColumn

(areaRef:L; $entityName:S; $dataType:L; $insertAt:L) -> error:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ $entityName	text	Name of collection or entity selection.
➞ $dataType	longint	EG number, date
➞ insertAt	longint	Position at which to insert a column; 0 means add to the end.
← result	longint	

Add a column at the specified position (**insertAt**).

$dataType is useful for collections, e.g. to set the column to display time; collections have only "number" - integer, longint, time, real are all the same.

You should set the data source first - collection or entity selection.

Example

$name:="some.entity.name"

$dataType:=Is undefined // data type will be fetched from the dataClass for entitySelection or from first data row for collection

$error:=***AL_AddEntityColumn*** ($area;$name;$dataType)

This is equivalent to when ALP_Area_DataSource is already set to 3 or 4:

$name:="some.entity.name"

$error:=***AL_AddColumn*** ($area;->$name)

AL_DuplicateColumn

(area:L; columnID:L; sourcePtr:Z; insertAt:L;) -> error:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ columnID	longint	Column no. to duplicate..
➞ sourcePtr	pointer	When in Records mode, sourcePtr should be a pointer to field. When in Arrays mode, sourcePtr should be a pointer to an array (must not be a local array!).
➞ insertAt	longint	Position at which to insert the duplicated column; 0 or missing means add to the end.
← result	longint	

Duplicate a column at the specified position (insertAt).

Example:

```
$err:=AL_DuplicateColumn ($area; 2; ->[SomeTable]SomeField; 1)
```

AL_GetColumnLongProperty

(areaRef:L; column:L; property:T) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ column	longint	The column number for which to get the property.
➞ property	text	The property to get.
⬅ result	longint	Value of the “got” property.

Get details of a **column**’s longint **property**. The properties that you can get with this command are the ones of type “longint” and “boolean” (1 or 0) listed in the AreaList Pro Column Properties theme.

Example

To find out how many columns in an area are invisible:

```
$columnCount:=AL_GetAreaLongProperty (area;ALP_Area_Columns)
$Invisible:=0
For($i;1; $columnCount)
   If (AL_GetColumnLongProperty (area;$i;ALP_Column_Visible)=0)  // not visible
      $Invisible:=$Invisible+1
   End If
End For
```

AL_GetColumnPtrProperty

(areaRef:L; column:L; property:T; pointer:Z) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ column	longint	The column number for which to get the property.
➞ property	text	The property to get.
⬅ pointer	pointer	Pointer to variable to hold the result.
⬅ result	longint	

Get details of a **column**'s **property** using a **pointer**. The properties that you can get with this command are listed in the AreaList Pro Column Properties theme.

Example

To find out the enterability of column 1:

```
C_LONGINT($enterable)
$err:=AL_GetColumnPtrProperty (area;1;ALP_Column_Enterable;->$enterable)
```

AL_GetColumnRealProperty

(areaRef:L; column:L; property:T) ➞ result:R

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ column	longint	The column number for which to get the property.
➞ property	text	The property to get.
⬅ result	real	Value of the "got" property.

Get details of a **column**'s real **property**. The properties that you can get with this command are the ones of type "real", listed in the AreaList Pro Column Properties theme.

Example

To find the current width of column number 2:

```
C_REAL($colWidth)
$colWidth:=AL_GetColumnRealProperty (area;2;ALP_Column_Width)
```

AL_GetColumnTextProperty

(areaRef:L; column:L; property:T) ➞ result:T

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ column	longint	The column number for which to get the property.
➞ property	text	The property to get.
⬅ result	text	Value of the “got” property.

Get details of a **column**’s text **property**. The properties that you can get with this command are the ones of type “text”, listed in the AreaList Pro Column Properties theme.

Example

To get column 3’s header text:

```
C_TEXT($headerText)
$headerText:=AL_GetColumnTextProperty (area;3;ALP_Column_HeaderText)
```

AL_MoveColumn

(areaRef:L; from:L; insertAt:L) → error:L

Parameter	Type	Description
areaRef	longint	Reference of AreaList Pro object
from	longint	Column no. to move from
insertAt	longint	Column no. to move to.

Example

Make column 18 the first column:

```
AL_MoveColumn ($area; 18; 1)//
```

AL_SetColumnLongProperty

(areaRef:L; column:L; property:T; value:L {; count:L})

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout.
→ column	longint	The number of the column for which to set the property.
→ property	text	The property to set.
→ value	longint	Value to pass to the property (long integer).
→ count	longint	Number of columns to set, starting at Column (optional).

Set a specific longint **property** for a **column** or several columns. The properties that you can set with this command are the ones of type “longint” and “boolean” (1 or 0), listed in the AreaList Pro Column Properties theme.

If the **count** parameter is omitted, only column number **column** will be set.

If you want to set all the columns in one go, enter -2 for the **column** number (**count** is ignored).

Example

To set the horizontal alignment for column 1 to “center”:

AL_SetColumnLongProperty (area;1;ALP_Column_HorAlign;2)

AL_RemoveColumn

(areaRef:L; column:L; {count:L}) → result:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout.
→ column	longint	Number of the (first) column to remove.
→ count	longint	Number of columns to remove, starting at column (optional).
← result	longint	

Remove one column or several columns from an area.

If the **count** parameter is omitted, only column number **column** will be removed. Otherwise **count** column(s) will be removed starting at **column**.

If you want to remove all the columns in one go, enter -2 for the column number.

Example

To remove columns 3 and 4 from area:

$err:=***AL_RemoveColumn*** (area;3;2)

AL_SetColumnPtrProperty

(areaRef:L; column:L; property:T; pointer:Z {; count:L}) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ column	longint	The number of the column for which to set the property.
➞ property	text	The property to set.
➞ pointer	pointer	Pointer to a variable that holds a value to pass to the function.
➞ count	longint	Number of columns to set, starting at Column (optional).
← result	longint	

Set a specific **property** for a **column** or several columns using a **pointer** to the value you want to set. The properties that you can set with this command are listed in the AreaList Pro Column Properties theme.

If the **count** parameter is omitted, only column number **column** will be set. Otherwise **count** column(s) will be set starting at **column**.

If you want to set all the columns in one go, enter -2 for the **column** number (**count** is ignored).

Example

To rotate the header text for columns 2 and 3 by 90º:

```
$rotation:=90
$err:=AL_SetColumnPtrProperty (area;2;ALP_Column_HdrRotation;->$rotation;2)
```

AL_SetColumnRealProperty

(areaRef:L; column:L; property:T; value:R {; count:L})

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ column	longint	The number of the column for which to set the property.
➞ property	text	The property to set.
➞ value	pointer	Value to pass to the function (real number).
➞ count	longint	Number of columns to set, starting at Column (optional).

Set a specific **property** for a **column** or several columns. The properties that you can set with this command are the ones of type “real”, listed in the AreaList Pro Column Properties theme.

If the **count** parameter is omitted, only column number **column** will be set. Otherwise **count** column(s) will be set starting at **column**.

If you want to set all the columns in one go, enter -2 for the **column** number (**count** is ignored).

Example

To set the width of columns 2, 3 and 4 to 30 points:

```
AL_SetColumnRealProperty (area;2;ALP_Column_Width;30;3)
```

AL_SetColumnTextProperty

(areaRef:L; column:L; property:T; value:T {; count:L})

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ column	longint	The number of the column for which to set the property.
➞ property	text	The property to set.
➞ value	pointer	Value to pass to the function (text).
➞ count	longint	Number of columns to set, starting at Column (optional).

Set a specific text **property** for a **column** or several columns. The properties that you can set with this command are the ones of type “text”, listed in the AreaList Pro Column Properties theme.

If the **count** parameter is omitted, only column number **column** will be set. Otherwise **count** column(s) will be set starting at **column**.

If you want to set all the columns in one go, enter -2 for the **column** number (**count** is ignored).

Example

To set the header text for column 1 to “Product Name”:

```
AL_SetColumnTextProperty (area;1;ALP_Column_HeaderText;"Product Name")
```

Rows

The commands in this theme affect the rows in an AreaList Pro area.

The properties that can be used with these commands can be found in the AreaList Pro Row Properties theme.

See Global Row settings and Row Numbering.

AL_GetRowLongProperty

(areaRef:L; row:L; property:T) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Number of row for which to get the details.
➞ property	text	The property to get.
⬅ result	longint	Value of the “got” property.

Get details of a **row**’s longint **property**.

Example

To get the parent of row 3 in a hierarchical list:

```
C_LONGINT($parent)
$parent:=AL_GetRowLongProperty (area;3;ALP_Row_Parent)
```

AL_GetRowPtrProperty

(areaRef:L; row:L; property:T; pointer:Z) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Number of row for which to get the details.
➞ property	text	The property to get.
➞ pointer	pointer	Pointer to a variable to hold the result.
⬅ result	longint	

Get details of a **row**'s **property** using a **pointer**.

Example

To get the height of row 3:

```
C_REAL($height)
$err:=AL_GetRowPtrProperty (area;3;ALP_Row_Height;->$height)
```

AL_GetRowRealProperty

(areaRef:L; row:L; property:T) ➞ result:R

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Number of row for which to get the details.
➞ property	text	The property to get.
⬅ result	real	Value of the "got" property.

Get details of a **row**'s real **property**.

Example

To get row 2's horizontal scale:

```
C_REAL($scale)
$scale:=AL_GetRowRealProperty (area;2;ALP_Row_HorizontalScale)
```

AL_GetRowTextProperty

(areaRef:L; row:L; property:T) ➞ result:T

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Number of row for which to get the details.
➞ property	text	The property to get.
⬅ result	text	Value of the “got” property.

Get details of a **row**’s text **property**.

Example

To get the name of the font for row 2:

```
C_TEXT($font)
$font:=AL_GetRowTextProperty (area;2;ALP_Row_FontName)
```

AL_ModifyArrays

(areaRef:L; selector:L; row:L; count:L) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ selector	longint	Action to perform.
➞ row	longint	Position where insert or delete occurred or should occur.
➞ count	longint	Number of rows to insert or delete, starting at Row.

Insert or delete a number of rows (array elements) at the specified position, or inform AreaList Pro that a number of elements have been inserted or deleted.

selector:

0 = **count** elements were inserted at **row**

1 = insert **count** elements at **row** using **INSERT IN ARRAY**

2 = **count** elements were deleted at **row**

3 = delete **count** elements at **row** using **DELETE FROM ARRAY**

This value informs AreaList Pro that the arrays have been modified (**selector** = 0 or 2) or asks AreaList Pro to modify the arrays (**selector** = 1 or 3). All arrays contained in the area will be processed, including hidden columns if any.

AreaList Pro will adjust the cache and move the row and cell options (depending on the current values for ALP_Area_MoveRowOptions and ALP_Area_MoveCellOptions). Thus ***AL_ModifyArrays*** is especially useful if you want to insert or delete rows while keeping any options (e.g. formatting) that you may have set for specific rows or cells.

Constants are available for all actions performed by ***AL_ModifyArrays***.

Example

The following two examples produce the exact same result in an area displaying two arrays.

Insert with 4D and inform AreaList Pro:

```
INSERT IN ARRAY (myArray1;5;3)  // insert 3 elements at position 5
INSERT IN ARRAY (myArray2;5;3)  // same with the other array
AL_ModifyArrays (area;AL Modify Insert info;5;3)  // inform AreaList Pro
```

Ask AreaList Pro to insert:

```
AL_ModifyArrays (area;AL Modify Insert action;5;3)  // insert 3 elements at position 5 in both arrays
```

AL_SetRowLongProperty

(areaRef:L; row:L; property:T; value:L {; count:L})

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Number of the row for which to set the property.
➞ property	text	The property to set.
➞ value	longint	Value to pass to the function (long integer).
➞ count	longint	Number of rows to set, starting at Row (optional).

Set a specific longint **property** for a **row** or several rows. If the **count** parameter is omitted, only row number **row** will be set. Otherwise **count** row(s) will be set starting at **row**.

If you want to set all the rows in one go, enter -2 for the **row** number (**count** is ignored).

Example

To set the text in rows 2 to 10 to bold + underline:

```
AL_SetRowLongProperty (area;2;ALP_Row_StyleF;5;9)
```

AL_SetRowPtrProperty

(areaRef:L; row:L; property:T; pointer:Z {; count:L}) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Number of the row for which to set the property.
➞ property	text	The property to set.
➞ pointer	pointer	Pointer to a variable that holds a value to pass to the function.
➞ count	longint	Number of rows to set, starting at Row (optional).
⬅ result	longint	

Set a specific **property** for a **row** or several rows using a **pointer** to the value you want to set.

If the **count** parameter is omitted, only row number **row** will be set. Otherwise **count** row(s) will be set starting at **row**.

If you want to set all the rows in one go, enter -2 for the **row** number (**count** is ignored).

Example

To set the font size for row 2 to 20:

```
$fontSize:=20
AL_SetRowPtrProperty (area; 2; ALP_Row_Size; ->$fontSize;19)
```

AL_SetRowRealProperty

(areaRef:L; row:L; property:T; value:R {; count:L})

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Number of the row for which to set the property.
➞ property	text	The property to set.
➞ value	real	Value to pass to the function (real number).
➞ count	longint	Number of rows to set, starting at Row (optional).

Set a specific **property** for a **row** or several rows.

If the **count** parameter is omitted, only row number **row** will be set. Otherwise **count** row(s) will be set starting at **row**.

If you want to set all the rows in one go, enter -2 for the **row** number (**count** is ignored).

Example

To rotate 90° the text displayed in row 33:

```
AL_SetRowRealProperty (area;33;ALP_Row_Rotation;90)
```

AL_SetRowTextProperty

(areaRef:L; row:L; property:T; value:T {; count:L})

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Number of the row for which to set the property.
➞ property	text	The property to set.
➞ value	text	Value to pass to the function (text).
➞ count	longint	Number of rows to set, starting at **row** (optional).

Set a specific **property** for a **row** or several rows. If the **count** parameter is omitted, only row number **row** will be set. Otherwise **count** row(s) will be set starting at **row**.

If you want to set all the rows in one go, enter -2 for the **row** number (**count** is ignored).

Example

Set the text color for rows 3 to 7 to black:

```
AL_SetRowTextProperty (area;3;ALP_Row_TextColor;"Black";5)
```

Cells

The commands in this theme affect individual cells within the AreaList Pro area.

The properties that can be used with these commands can be found in the AreaList Pro Cell Properties theme.

See examples in the AreaList Pro Cell Properties section.

See also Global Cell settings.

AL_GetCellLongProperty

(areaRef:L; row:L; column:L; property:T) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Row number for which to get the property.
➞ column	longint	Column number for which to get the property.
➞ property	text	The property to get.
⬅ result	longint	Value of the “got” property.

Get details of a cell’s **property**. The properties that you can get with this command are the ones of type “longint” and “boolean” (1 or 0) listed in the AreaList Pro Cell Properties theme.

Example

To find out the enterability of the cell at coordinates 1,1:

```
$enterable:=AL_GetCellLongProperty (area;1;1;ALP_Cell_Enterable)
```

AL_GetCellPtrProperty

(areaRef:L; row:L; column:L; property:T; pointer:Z) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Row number for which to get the property.
➞ column	longint	Column number for which to get the property.
➞ property	text	The property to get.
← pointer	pointer	Pointer to variable to hold the result.
← result	longint	

Get details of a cell's **property** using a **pointer**. The properties that you can get with this command are listed in the AreaList Pro Cell Properties theme.

Example

To find out which font the cell at coordinates 1,4 is currently set in:

```
Selectedfont:=""
$error:=AL_GetCellPtrProperty (area;1;4;ALP_Cell_FontName;->Selectedfont)
```

AL_GetCellRealProperty

(areaRef:L; row:L; column:L; property:T) ➞ result:R

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Row number for which to get the property.
➞ column	longint	Column number for which to get the property.
➞ property	text	The property to get.
← result	real	Value of the "got" property.

Get details of a cell's real **property**. The properties that you can get with this command are the ones of type "real" listed in the AreaList Pro Cell Properties theme.

Example

To get the horizontal scale of the cell at coordinates 4,2:

```
C_REAL($scale)
$scale:=AL_GetCellRealProperty (area;4;2;ALP_Cell_HorizontalScale)
```

AL_GetCellTextProperty

(areaRef:L; row:L; column:L; property:T) ➞ result:T

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	Row number for which to get the property.
➞ column	longint	Column number for which to get the property.
➞ property	text	The property to get.
⬅ result	text	Value of the “got” property.

Get details of a cell’s text **property**. The properties that you can get with this command are the ones of type “text” listed in the AreaList Pro Cell Properties theme.

Example

To get a description of the options, in XML, of the cell at coordinates 4,2:

```
C_TEXT($xml)
$xml:=AL_GetCellTextProperty (area;4;2;ALP_Cell_XML)
```

AL_SetCellLongProperty

(areaRef:L; row:L; column:L; property:T; value:L {; rowCount:L} {; columnCount:L})

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	The number of the row for which to set the property.
➞ column	longint	The number of the column for which to set the property.
➞ property	text	The property to set.
➞ value	longint	Value to pass to the function (long integer).
➞ rowCount	longint	Number of rows to set, starting at Row (optional).
➞ columnCount	longint	Number of columns to set, starting at Column (optional).

Set a specific longint **property** for a cell or several cells. The properties that you can set with this command are the ones of type “longint” and “boolean” (1 or 0) listed in the AreaList Pro Cell Properties theme.

If the **count** parameters are omitted, only cell at coordinates **row**, **column** will be set.

Otherwise **rowCount** x **columnCount** cells will be set starting at the cell at coordinates **row**, **column**.

Example

To set the vertical alignment to “bottom” for the cell at coordinates 4,2:

```
AL_SetCellLongProperty (area;4;2;ALP_Cell_VertAlign;3)
```

AL_SetCellPtrProperty

(areaRef:L; row:L; column:L; property:T; pointer:Z {; rowCount:L} {; columnCount:L}) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ row	longint	The number of the row for which to set the property.
➞ column	longint	The number of the column for which to set the property.
➞ property	text	The property to set.
➞ pointer	pointer	Pointer to a variable that holds a value to pass to the function.
➞ rowCount	longint	Number of rows to set, starting at Row (optional).
➞ columnCount	longint	Number of columns to set, starting at Column (optional).
← result	longint	

Set a specific **property** for a cell or several cells using a **pointer** to the value you want to set.

If the **count** parameters are omitted, only cell at coordinates **row**, **column** will be set.

Otherwise **rowCount** x **columnCount** cells will be set starting at the cell at coordinates **row**, **column**.

Example

To set the color of the text in cell 1,3 to blue:

```
C_TEXT($color)
$color:="Blue"
$err:=AL_SetCellPtrProperty (area;1;3;ALP_Cell_TextColor;->$color)
```

Note: for more information on how to specify colors, see the Working with Colors section.

AL_SetCellRealProperty

(areaRef:L; row:L; column:L; property:T; value:R {; rowCount:L} {; columnCount:L}))

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout.
→ row	longint	The number of the row for which to set the property.
→ column	longint	The number of the column for which to set the property.
→ property	text	The property to set.
→ value	real	Value to pass to the function (real number).
→ rowCount	longint	Number of rows to set, starting at Row (optional).
→ columnCount	longint	Number of columns to set, starting at Column (optional).

Set a specific **property** for a cell or several cells. The properties that you can set with this command are the ones of type “real”, listed in the AreaList Pro Cell Properties theme.

If the **count** parameters are omitted, only cell at coordinates **row**, **column** will be set.

Otherwise **rowCount** x **columnCount** cells will be set starting at the cell at coordinates **row**, **column**.

Example

To rotate the text in the cells from coordinates 2,1 to coordinates 8,4 by 180º (turn them upside down):

AL_SetCellRealProperty (area;2;1;ALP_Cell_Rotation;180;7;4)

AL_SetCellTextProperty

(areaRef:L; row:L; column:L; property:T; value:T {; rowCount:L} {; columnCount:L}))

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout.
→ row	longint	The number of the row for which to set the property.
→ column	longint	The number of the column for which to set the property.
→ property	text	The property to set.
→ value	text	Value to pass to the function (text).
→ rowCount	longint	Number of rows to set, starting at Row (optional).
→ columnCount	longint	Number of columns to set, starting at Column (optional).

Set a specific text **property** for a cell or several cells. The properties that you can set with this command are the ones of type “text”, listed in the AreaList Pro Cell Properties theme.

If the **count** parameters are omitted, only cell at coordinates **row**, **column** will be set.

Otherwise **rowCount** x **columnCount** cells will be set starting at the cell at coordinates **row**, **column**.

Example

To set the font for the cell at row 3, column 2 to Arial Black:

AL_SetCellTextProperty (area;3;2;ALP_Cell_FontName;"Arial Black")

Objects

AL_SetObjects/***AL_GetObjects*** are used for getting or setting an area's property where there is an array of values rather than an individual value.

For example, if you wanted to know **how many** rows were selected in an AreaList Pro area with row selection, you would use the ALP_Area_Select property:

```
numberOfRows:=AL_GetAreaLongProperty(area; ALP_Area_Select)
```

numberOfRows tells you the number of rows that are selected.

However, if you want to know **which** rows are selected, you cannot use ***AL_GetAreaLongProperty***. The selected rows are not one number - they are a set of numbers; one for each selected row. You can use ***AL_GetObjects*** with the ALP_Object_Selection property - for example:

```
ARRAY LONGINT(selectedRows;0)
$error:=AL_GetObjects(area; ALP_Object_Selection; selectedRows)
```

After the call the selectedRows array will contain the row numbers that are selected. Note that the array is passed directly, not as a pointer.

The properties that can be used with these commands can be found in the AreaList Pro Objects theme.

AL_GetObjects

(areaRef:L; property:T; array:Y) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to get.
⬅ array	array	An array to hold the result. Depending on the property, this can be an array of row or column numbers or a two-dimensional array of cell coordinates.
⬅ result	longint	

Get details of an area's object **property**.

Example 1: Using a One-Dimensional Array

To find out which rows have been selected (when in multi-row selection mode), place this code in the AreaList Pro area's Object Method:

```
Case of
: (Form event=On Plug in Area)
   Case of
      : (AL_GetAreaLongProperty (area;ALP_Area_AlpEvent)=1)  // did user click on a row?
      ARRAY LONGINT(aRows;0)
      $err:=AL_GetObjects (area;ALP_Object_Selection;aRows)  // get selected rows
   End case
End case
```

aRows now contains an element for each selected row containing the row number.

Example 2: Using a Two-Dimensional Array

To get the cell selection when cell selection mode has been turned on with ALP_Area_SelType set to 1 (single cell) or 2 (mutiple cells):

```
ARRAY LONGINT(aCells;0;0)
$err:=AL_GetObjects (area;ALP_Object_Selection;aCells)  //get selected cells
```

aCells{1} now contains the row numbers and aCells{2} the column numbers.

AL_GetObjects2

(areaRef:L; property:T; array1:Y; array2:Y) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to get.
← array1	array	An array to hold the first array result.
← array2	array	An array to hold the second array result. Depending on the property, this can be an array of row or column numbers.
← result	longint	

Get details of an area's object **property** using two arrays. These arrays must be one-dimensional. The properties that can be used with this command can be found in the AreaList Pro Objects theme.

Example 1

To get two arrays listing the table and field numbers that have been assigned to the columns in an area:

```
ARRAY LONGINT($arrayTableNos;0)
ARRAY LONGINT($arrayFieldNos;0)
$err:=AL_GetObjects2 (area;ALP_Object_Fields;$arrayTableNos;$arrayFieldNos)
```

Example 2

To get the cell selection when cell selection mode has been turned on with ALP_Area_SelType set to 1 (single cell) or 2 (mutiple cells):

```
ARRAY LONGINT(aRows;0)
ARRAY LONGINT(aCols;0)
$err:=AL_GetObjects2 (area;ALP_Object_Selection;aRows;aCols)  //get selected cells
```

aRows now contains the row numbers and aCols the column numbers.

AL_SetObjects

(areaRef:L; property:T; array:Y) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to set.
➞ array	array	An array containing the values to pass to the function. Depending on the property, this can be an array of row or column numbers or a two-dimensional array of cell coordinates.
result	longint	

Set a **property** for an area. You can use either a one - or two- dimensional array.

Example

Add a number of arrays to an AreaList Pro area:

```
ARRAY POINTER(aPtr;4)
aPtr{1}:=->[product]product_type
aPtr{2}:=->[product]product_name
aPtr{3}:=->[product]retail_price
aPtr{4}:=->[product]description
$err:=AL_SetObjects (area;ALP_Object_Columns;aPtr)
    // DELETE all columns and then add specified pointers (effectively replacing all columns)
```

AL_SetObjects2

(areaRef:L; property:T; array1:Y; array2:Y) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ property	text	The property to set.
➞ array	array	An array containing the values to pass to the function. Depending on the property, this can be an array of row or column numbers or a two-dimensional array of cell coordinates.
➞ array2	array	An array containing the values to pass to the function Depending on the property, this can be an array of row or column numbers.
⬅ result	longint	

Set properties for area objects using two arrays.

Example

A good example of the use of this command is for displaying data in a hierarchical list.

```
$err:=AL_SetObjects2 (area;ALP_Object_Hierarchy;$aiLevel;$aiExpanded)
```

Break Processing

Get/Set Break Processing Properties

If cellNumber is zero, usable properties are:

ALP_Break_RowID

ALP_Break_NumRowLines

ALP_Break_MinRowHeight

ALP_Break_RowIndent

ALP_Break_RowIndentH

ALP_Break_RowIndentV

Use ALP_Area_RowsInGrid to set the number of lines (cells) in each column break.

otherwise:

ALP_Cell_Value: the break value (string which can use the “\\SUM” and other functions evaluated in the break processing)

AL_SetBreakPtrProperty

(areaRef:L; header:L; level:L; cellNumber:L; property:T; value:Z) → error:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout.
→ header	longint	0 = footer, 1 = header
→ level	longint	Break level: 0 - 64
→ cellNumber	longint	0 to apply to the break, > 0 to apply to the individual cell
→ property	text	
→ value		
← result	longint	

Set properties for break processing.

AL_SetBreakLongProperty

(areaRef:L; header:L; level:L; cellNumber:L; property:T; value:Z) → error:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout.
→ header	longint	0 = footer, 1 = header
→ level	longint	Break level: 0 - 64
→ cellNumber	longint	0 to apply to the break, > 0 to apply to the individual cell
→ property	text	
→ value		
← result	longint	

Set properties for break processing.

AL_SetBreakRealProperty

(areaRef:L; header:L; level:L; cellNumber:L; property:T; value:8) → error:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout.
→ header	longint	0 = footer, 1 = header
→ level	longint	Break level: 0 - 64
→ cellNumber	longint	0 to apply to the break, > 0 to apply to the individual cell
→ property	text	
→ value	real	
← result	longint	

Set properties for break processing.

AL_SetBreakTextProperty

(areaRef:L; header:L; level:L; cellNumber:L; property:T; value:T) → error:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ header	longint	0 = footer, 1 = header
➞ level	longint	Break level: 0 - 64
➞ cellNumber	longint	0 to apply to the break, > 0 to apply to the individual cell
➞ property	text	
➞ value	text	
⬅ result	longint	

Set properties for break processing.

AL_GetBreakPtrProperty

(areaRef:L; header:L; level:L; cellNumber:L; property:T; value:Z) → error:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ header	longint	0 = footer, 1 = header
➞ level	longint	Break level: 0 - 64
➞ cellNumber	longint	0 to apply to the break, > 0 to apply to the individual cell
➞ property	text	
➞ value		
⬅ result	longint	

Get properties for break processing.

AL_GetBreakLongProperty

(areaRef:L; header:L; level:L; cellNumber:L; property:T) → value:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ header	longint	0 = footer, 1 = header
➞ level	longint	Break level: 0 - 64
➞ cellNumber	longint	0 to apply to the break, > 0 to apply to the individual cell
➞ property	text	
⬅ result	longint	

Get break processing properties.

AL_GetBreakTextProperty

(areaRef:L; header:L; level:L; cellNumber:L; property:T) → value:T

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ header	longint	0 = footer, 1 = header
➞ level	longint	Break level: 0 - 64
➞ cellNumber	longint	0 to apply to the break, > 0 to apply to the individual cell
➞ property	text	
← result	text	

Get break processing properties.

AL_GetBreakRealProperty

(areaRef:L; header:L; level:L; cellNumber:L; property:T) → value:8

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ header	longint	0 = footer, 1 = header
➞ level	longint	Break level: 0 - 64
➞ cellNumber	longint	0 to apply to the break, > 0 to apply to the individual cell
➞ property	text	
← result	real	

Get break processing properties.

Get/Set Break Processing Objects

The only property is ALP_Object_Grid → use 2D array

AL_GetBreakObjects

(areaRef:L; header:L; level:L; property:T; array:Y) → error:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ header	longint	0 = footer, 1 = header
➞ level	longint	Break level: 0 - 64
➞ property	text	
➞ array	text array	

Parameter	Type	Description
← result	text	

Get break processing object. The only supported property is grid (ALP_Object_Grid), thus the array must be of type **ARRAY INTEGER**, **ARRAY LONGINT** or 2D array (int/long), not text.

See accessing grid for area with ***AL_SetObjects***/***AL_GetObjects***. This is the same, but for one break.

AL_SetBreakObjects

(areaRef:L; header:L; level:L; property:T; array:Y) → error:L

Parameter	Type	Description
→ areaRef	longint	Reference of AreaList Pro object on layout.
→ header	longint	0 = footer, 1 = header
→ level	longint	Break level: 0 - 64
→ property	text	
→ array	text array	
← result	text	

Set break processing object. The only supported property is grid (ALP_Object_Grid), thus the array must be of type **ARRAY INTEGER**, **ARRAY LONGINT** or 2D array (int/long), not text.

See accessing grid for area with ***AL_SetObjects***/***AL_GetObjects***. This is the same, but for one break.

Utility

%AL_DropArea

%AL_DropArea is the command used to identify the plug-in area to which an AreaList Pro row or column can be dragged, but which does not display anything.

This command will appear in the 4D Object Types popup on a layout Property List.

It is only used in the object definition for an ***%AL_DropArea*** object, and should never be used as a command in a 4D method.

%AreaListPro

%AreaListPro is the command used to identify the AreaList Pro plug-in area when you create a plug-in area object on a layout.

This command is only used in the object definition for an AreaList Pro object, and should never be used as a command in a 4D method.

AL_ColorPicker

(ARGBColor:L) ➞ result:L

Parameter	Type	Description
↔ ARGBColor	longint	The number of the color that was selected.
← result	longint	1 if a color was selected; 0 if not.

Invokes the color picker to select a color using the system color palette.

Note: AreaList Pro uses the alpha channel A for transparency. See the Working with Colors section.

The function returns a longint in **ARGBColor** which you can then use as a parameter to set colors for rows, text, etc.

Note that the 4D function **Select RGB Color** performs a similar function, but without the alpha channel.

AL_GetIcon

(areaRef:L; iconID:L) ➞ result:P

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout or 0 for global workstation settings.
➞ iconID	longint	ID number of a picture used internally by AreaList Pro.
← result	picture	The specified picture.

Use this command to get a picture as used internally in AreaList Pro.

Current appearance of the area is used to resolve special icon IDs (areaRef can be zero).

Can be used e.g. to get native appearance pictures (ALP_Area_Appearance = -1) and use them with pictures appearance (ALP_Area_Appearance = 11).

Constants are available for miscellaneous internal icons (popups, node disclosure triangle, checkbox, radio button, sort indicator):

- AL Icon Generic Popup PictID
- AL Icon Date PictID
- AL Icon Time PictID
- AL Icon Node Right PictID
- AL Icon Node Down PictID
- AL Icon CB Unchecked PictID
- AL Icon CB Checked PictID
- AL Icon CB Mixed PictID
- AL Icon RB Unchecked PictID
- AL Icon RB Checked PictID
- AL Icon Sort Up PictID
- AL Icon Sort Down PictID
- AL Icon Appearance Offset

When using ***AL_GetIcon***, current ALP_Area_Appearance is used to resolve the real picture ID, but when setting the icon, nothing special is done, you must use AL Icon Appearance Offset to modify icons for ALP_Area_Appearance = 11 (AL Use Pictures Interface).

AL_Load

(areaRef:L; XML:T) ➞ result:L

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

Initialize an area from an **XML** (using UTF-8) text that was saved to a text field or variable using the ***AL_Save*** command.

Please see the section on XML for more details about saving and loading XML.

Example

This example initialises an AreaList Pro area using settings that were saved into a field in the database.

```
$err:=AL_Load (area;[Settings]ALP_template)
```

AL_Register

(registrationCode; options; email) ➞ result

Parameter	Type	Description
➞ registrationCode	text	Pass the registration key to register your copy of AreaList 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 registration).
➞ options	longint	An optional longint combining up to 4 bits.
➞ email	text	Online registration option: developer email to notify when a license is issued or resent.
← result	longint	0 or error code.

AL_Register is used to register the AreaList Pro plugin for standalone or server use.

Please see the License Types section for detailed information about the licensing options available for AreaList Pro.

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

registrationCode — You must call ***AL_Register*** with a valid registration key, otherwise AreaList 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 the license server for Online registration.

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 registration option: confirm connection “Is it OK to connect to the license server to register AreaList Pro?”
2	Online registration option: display alert if registration error
3	Online registration option: display alert if registered

email — Optional. The developer email address where to send Online registration 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 registration)
10	Unauthorized master key (Online registration)
11	Can't connect to the license server to perform Online registration
12	No Online registration license available for this master key (unknown or all used)

When ***AL_Register*** is called with an empty string, the license dialog will be displayed if AreaList 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 AreaList Pro command or displaying a AreaList Pro area.

Note: alternately to ***AL_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)
$result:=AL_Register ("YourRegistrationKey")
Case of
   :($result=2)
      ALERT ("The AreaList Pro licence is invalid.")
   :($result=3)
      ALERT ("The AreaList Pro licence has expired.")
   etc.
End case
```

Example with multiple calls

```
C_LONGINT ($result)  //ignored in this case
$result:=AL_Register ("Registration key one")
$result:=AL_Register ("Registration key two")
$result:=AL_Register ("Registration key three")
```

etc.

```
If ($result#0)  //registration failed on all keys
   ALERT ("AreaList 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:=AL_Register ("Registration key one";1)  //invalid, will return an error, the plugin isn't registered
$result:=AL_Register ("Registration key two";1)  //valid, will return 0, the plugin is registered
$result:=AL_Register ("Registration key three";1)  //invalid, will return an error, the plugin is still registered
```

Online registration examples

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

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

AL_GetPlainText

(styledText:T) ➞ plainText:T

Parameter	Type	Description
➞ styledText	text	Styled (attributed) text.
⬅ plainText	text	Plain text.

Converts an attributed (styled) text to plain text. See AreaList Pro Text Style Tags

Example

```
C_TEXT($text)
$text:=AL_GetPlainText ("<c blue><b>test</b></c>")  //returns "test"
```

AL_Save

(areaRef:L; XML:T) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList 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 saved OK; 2 if not.

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

Please see the section on XML for more details about saving and loading XML.

Example

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

```
C_TEXT($Settings)
$err:=AL_Save (area;$Settings)
[Settings]ALP_template:=$Settings
```

AL_SetIcon

(areaRef:L; iconID:L; iconValue:P) ➞ result:L

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout or 0 for global workstation settings
➞ iconID	longint	ID number (assigned by the developer) of a picture to add to the AreaList Pro picture library. The number must be between 1 and 16777215.
➞ iconValue	picture	A picture variable that contains the specified picture. If this is empty, the picture is removed from the AreaList Pro picture library. The picture can be a variable or a 4D field.
⬅ result	longint	

Use this command to add a picture to AreaList Pro's picture library, which is basically a picture cache.

You can use 0 as the **areaRef** to set an icon in the global workstation cache, or specify a given area.

You assign a reference number, which can then be used in the icon placement and formatting commands.

Each picture is stored only once but can be used any number of times, thus being very efficient with memory usage.

Note: icon IDs 1 to 5 can be used to replace AreaList Pro's native icons for popups and hierarchical lists. See Displaying custom pictures instead of AreaList Pro's native icons.

Example 1

To add a picture from a 4D field to the AreaList Pro picture library and assign it the reference number 100:

```
$err:=AL_SetIcon (area;100;[pictures]clock_icon)
```

For detailed examples of using pictures in your AreaList Pro area, see the Pictures topic.

Example 2

This example sets custom pictures for checkboxes and sort indicators.

```
C_PICTURE($Pimage)
C_TEXT($Tpath)
C_LONGINT($i)

//Location of the custom pictures
$Tpath:=Get 4D folder (Current resources folder; *)+"Images"+Folder separator

//Checkboxes
READ PICTURE FILE($Tpath+"Cbx_Yes.png"; $Pimage)
$i:=AL_SetIcon(0; AL Icon CB Checked PictID+AL Icon Appearance Offset; $Pimage) //0 for all areas
READ PICTURE FILE($Tpath+"Cbx_No.png"; $Pimage)
$i:=AL_SetIcon(0; AL Icon CB Unchecked PictID+AL Icon Appearance Offset; $Pimage)

//Sort indicators
READ PICTURE FILE($Tpath+"Sort_Up.png"; $Pimage)
$i:=AL_SetIcon(0; AL Icon Sort Up PictID+AL Icon Appearance Offset; $Pimage)
READ PICTURE FILE($Tpath+"Sort_Down.png"; $Pimage)
$i:=AL_SetIcon(0; AL Icon Sort Down PictID+AL Icon Appearance Offset; $Pimage)

//Use custom pictures
AL_SetAreaLongProperty(0; ALP_Area_Appearance; 11)  //11 = pictures set by developer
```

15 Properties by Theme

In this section you'll find complete details about each property that can be used with the AreaList Pro commands. They are organised into themes according to which group of commands they relate to:

AreaList Pro Area Properties affect the whole AreaList Pro area

AreaList Pro Column Properties affect columns

AreaList Pro Row Properties affect rows

AreaList Pro Cell Properties affect cells

AreaList Pro Object Properties affect various objects used by AreaList Pro

Breaks manage break processing

Printing incorporating PrintList Pro

The following details are included for each property:

Constant: the name of the property that you type into the command.

Get: whether the constant can be used in Getter commands

Set: whether it can be used in Setter commands

Per: persistent. If a property is persistent, it means that the property is saved with the area definition and will be applied when the area is displayed again.

Type: the type of the value:

Bool: boolean value (True/yes=1 or False/no=0)

Int: a long integer (**ARRAY INTEGER** is supported with small data, like ALP_Area_SortList, since the number of columns is limited to 32767)

Real: a real number

Text: an alphanumeric

Color: the "Color" type will accept seven methods, whether as string values or longint values. See Working with colors.

Cell: a string containing both row number and column number separated by a comma («row,column») = e.g. '5,3' is the cell located at the fifth row, third column.

Default: the default value that will be used for this property unless you specify otherwise

Min: the minimum acceptable value, where appropriate

Max: the maximum acceptable value, where appropriate

Comments: a description of the constant and, where appropriate, a list of allowable options

Properties shown in green were added or modified in Version 10.

Properties shown in magenta were added or modified in Version 11.

AreaList Pro Area Properties

Use these properties with commands in the Area command theme:

AL_AddCalculatedColumn

AL_AddColumn

AL_AddEntityColumn

AL_GetAreaLongProperty

AL_GetAreaPtrProperty

AL_GetAreaRealProperty

AL_GetAreaTextProperty

AL_SetAreaLongProperty

AL_SetAreaPtrProperty

AL_SetAreaRealProperty

AL_SetAreaTextProperty

Global Area settings:

- Area = 0 : apply to all areas to be created (defining an area “template”).
- Area = -2 : apply to all existing areas, but not to areas to be created.
- Area = -3 : apply to all existing areas in current process only.
- Area = -4 : apply to all existing areas, and to areas to be created.
- Other values mean “access this area’s settings”.

For some Area properties pertaining to areas (e.g. ALP_Area_UseDateControls or ALP_Area_ClickDelay), not global settings, you can use 0 as the Area Reference to accessing the default values for all newly initialized (or re-initialized) areas.

If you access workstation-only properties (properties not specific to areas, called Plugin properties, such as ALP_Area_TraceOnError or ALP_Area_Version), AreaRef is ignored.

AreaList Pro Area General Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area General Properties				
ALP_Area_AccessFocusedTarget	✔			int				Sub-ALP: AreaList Pro reference to parent (owning) AreaList Pro, but sets the ALP_Area_APITarget, too
ALP_Area_AltHdrClear		✔						Clear the alternate header grid & cells
ALP_Area_AltHdrRowsInGrid	✔	✔	✔	int	1	-1	20	Number of rows in alternate header grid
ALP_Area_APITarget	✔	✔		int				Sub-ALP: row in parent for access
ALP_Area_ArrowsForHierarchy	✔	✔		bool	no			When hierarchy is displayed, left/right arrow keys (without command key) are used to collapse/expand nodes, not for horizontal scrolling
ALP_Area_AutoHierarchy	✔	✔	✔	text or int				Int: one level (column number to use for break processing) Text: comma-separated list of columns to use for break processing Negative values allowed (processed as positive, but consistent with ALP_Area_SortList/ALP_Area_SortListNS) Implies break processing (automatic break headers) and hierarchy Automatic break headers, automatic hierarchy created from the breaks You can add break footers
ALP_Area_AutoResizeAllColumns	✔	✔	✔	int	0	0	2	0 = none 1 = listbox like (shrink to fit) 2 = HTML like (proportional resizing of marked columns) to work properly: - set the minimum column width, otherwise the columns will be 0 points wide - to have any effect, at least one visible column must have ALP_Column_Proportional set to 1! - only visible columns with ALP_Column_Proportional property set to 1 are resized - ALP_Column_Proportional means that the difference (shrink/expand) is distributed proportionally across the columns (if possible - honoring min/max width) using current column width, e.g. if you have one column with width 100, another one with width 200, the first one will get 1/3 of the resize, the second one will get 2/3 of the resize - if only one visible column has this property set, it behaves similarly to ALP_Area_AutoResizeColumn
ALP_Area_AutoResizeColumn	✔	✔	✔	int	0	-1	number of columns	0 = do nothing -1 = ALP_Area_AutoSnapLastColumn otherwise autoresize the column to match the area size if there is enough space left

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area General Properties								
ALP_Area_AutoSnapLastColumn	✔	✔	✔	bool	no			If set, the last visible column will be resized (ALP_Column_Width) to match the area size if there is enough space left Note: this property is the same as ALP_Area_AutoResizeColumn (which will be superseded and conversely, whichever comes last) with the last visible column This property is simply a wrapper to ALP_Area_AutoResizeColumn with values 0/-1)
ALP_Area_Compatibility	✔	✔	✔	bool	depends			AreaList Pro 8.x compatibility mode (defaut value depends on initialization)
ALP_Area_CompHideCols	✔	✔	✔	int	0	0		Number of columns to hide (compatibility mode only) See Hiding columns
ALP_Area_DataClassName	✔	✔	✔	text				An ORDA data class name
ALP_Area_DataClassObject	✔	✔		object				An ORDA data class object
ALP_Area_DataHeight	✔			real				Total height of all columns If the value is greater than ALP_Area_ListHeight, the vertical scrollbar will be active
ALP_Area_DataWidth	✔			real				Total width of all columns If the value is greater than ALP_Area_ListWidth, the horizontal scrollbar will be active
ALP_Area_DeleteOffscreen		✔						Setter deletes specified offscreen area
ALP_Area_DontSetCursor	✔	✔		bool	no			When set, AreaList Pro will not set the cursor Note: the cell entry widget is not affected, it maintains the cursor on its own
ALP_Area_ExportList	✔	✔	✔	text	“”			Set the list of columns to export with ALP_Area_ExportToFile See Selecting the Columns to export
ALP_Area_ExportOptions		✔		int	0	0	131071	Set options for exporting data to a file using ALP_Area_ExportToFile These constants are documented in the Format and Options section See also the Tutorial for an example
ALP_Area_ExportToFile		✔		text				Export the area to a file Value must be the full path to a document to write
ALP_Area_IsArea	✔			bool				Is this an AreaList Pro area?
ALP_Area_Kind	✔		✔	text				Object kind = “ALP”
ALP_Area_ListHeight	✔			real				Height of the list area: the data area not including the frame, scrollbars, header and footer
ALP_Area_ListLeft	✔			real				Left coordinate of list area
ALP_Area_ListTop	✔			real				Top coordinate of list area
ALP_Area_ListWidth	✔			real				Width of the list area, not including the frame and scrollbars
ALP_Area_MoveCellOptions	✔	✔	✔	bool	yes			Move cell options with cells (on sort)
ALP_Area_MoveInFieldsMode	✔	✔	✔	bool	no			Honor move of cell/row options even in Fields mode
ALP_Area_MoveRowOptions	✔	✔	✔	bool	yes			Move row options with rows (on sort and drag)

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area General Properties				
ALP_Area_Name	✔	✔	✔	text				Name of the area (if empty, the variable name is initialized from the form variable name in design mode)
ALP_Area_NamedSelection	✔	✔		text				Name of the named selection to use – Similar to fields mode (setup the fields as usual), but using named selection
ALP_Area_NewOffscreen	✔							Getter creates new offscreen area
ALP_Area_ParentALP	✔			int				Sub-ALP: AreaList Pro reference to parent (owning) ALP
ALP_Area_ParentColumn	✔			int				Sub-ALP: Parent column of this data set
ALP_Area_ReadOnly	✔	✔		int	0	0	15	Make the area read-only for the specified feature(s) bit 0 (value 1): make area not enterable bit 1 (value 2): make area not droppable (ignore drag) bit 2 (value 4): make area not draggable bit 3 (value 8): make subALP not focusable
ALP_Area_Redraw		✔		n/a				Redraw the area object (redraw is done when area is called by 4D)
ALP_Area_ScrollColumns	✔	✔	✔	bool				If set to true, horizontal scrolling is done in number of columns, not in points Automatically set to true when compatibility is turned on When set to True, no visible column will ever be larger that the area width.
ALP_Area_ScrollLeft	✔	✔		real				Horizontal scroll position in points
ALP_Area_ScrollTop	✔	✔		real				Vertical scroll position in points
ALP_Area_Selected	✔			bool				Is selected (has focus in 4D)
ALP_Area_Self	✔			pointer				Pointer to the area object **C_POINTER**($ptr) $err:=***AL_GetAreaPtrProperty*** ($area;ALP_Area_Self;->$ptr)
ALP_Area_TopRow	✔			int				The row number of the first (possibly partially) visible row on screen
ALP_Area_UserBLOB	✔	✔	✔	BLOB				BLOB for free use by developer
ALP_Area_Visible	✔	✔		bool				Area is displayed in current page (or on page 0) (not necessarily visible, because scrolled or a DIALOG on top of it)
ALP_Area_WindowsClip	✔	✔		bool				Deprecated since v10 Does nothing
ALP_Area_WindowsText	✔	✔		int	0	0	2	0 (default) = use GDI+ 1 = change the engine used for drawing on Windows to GDI drawing 2 = DirectDraw + DirectWrite 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 Can be used with existing areas to dynamically switch the drawing engine used on Windows

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area General Properties								
ALP_Area_XML	✔	✔		text				Full description of the area in XML
ALP_Area_XMLAP	✔			text				Advanced properties from design
ALP_Break_FooterCalcMethod								Method for handling break footer's "\\ FUNCTION" See PL_SetBrkFunc
ALP_Break_HeaderCalcMethod								Method for handling break header's "\\ FUNCTION" See PL_SetBrkFunc
ALP_Break_HideDetails								Don't show data rows (old PrintList Pro feature, $3 in call to ***PL_SetColOpts***) Show only breaks

AreaList Pro Area Data Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Data Properties								
ALP_Area_CacheSize	✔	✔	✔	int	1024	128	32767	Cache size (in number of rows to cache)
ALP_Area_CallbackFormulaOnSave	✔	✔		object	null			When set, it is executed as Formula with no parameters after call to entity.save() (thus in entity selection mode only) "This" inside the Formula contains: "status" - object - returned by entity.save() call -> check for This.status.success "alp" - number - ALP reference (can be sub-ALP; in such case the ALP_Area_APITarget is already set) "row" - number - edited row number "column" - number - edited column number "entity" - object - the entity which was actually saved "attribute" - text - name of the edited property of the entity
ALP_Area_CheckData		✔		int	0			Check number of rows (in arrays/selection), then ALP_Area_FillCache: fill the cache Usable when you append rows to arrays, no need to clear the cache, just change the number of rows, possibly adding them to the cache
ALP_Area_ClearCache		✔		int				Clear cells cache Set value to a row number to clear only that row (see Row Numbering) Set value to -2 to refresh all rows
ALP_Area_Collection	✔	✔		collection				Collection to be displayed
ALP_Area_Columns	✔		✔	int				Current number of columns.
ALP_Area_CurrentEntity	✔			object				Entity being edited or fetched (usable by callbacks, like calculated columns), "current row" in both collection and entity selection

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Data Properties								
ALP_Area_DataSource	✔	✔	✔	int	0	0	4	0 - unspecified 1 - fields 2 - arrays 3 - collection 4 - entity selection
ALP_Area_EntitySelection	✔	✔		object				Displayed entity selection
ALP_Area_FillCache		✔		int	0			Fill cells cache Argument is the number of rows to fetch into the cache (from first visible row) Default is rows in visible area Maximum is limited by the number of rows and ALP_Area_CacheSize value
ALP_Area_Rows	✔			int				Current number of rows
ALP_Area_TableID	✔	✔	✔	int	0			Main table number Zero when showing arrays >0 when showing fields
ALP_Area_UpdateData		✔		int	0			Clear cells cache (ALP_Area_ClearCache), Check for selection size change (ALP_Area_CheckData), Fill cells cache (ALP_Area_FillCache) Can use integer parameter - number of rows to fetch into the cache 0 means fetch rows visible on screen
ALP_Area_UserCollection	✔	✔	✔	collection				Collection for free use by developer Use with ***AL_Get/SetAreaPtrProperty*** and a pointer to a collection

AreaList Pro Area Display Properties

Note: many of these Display properties also apply to Printing with the PrintList Pro module, and conversely.

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Display Properties				
ALP_Area_AltRowColor	✔	✔	✔	color	#FFEEEEEE			Alternate row color (default is light gray)
ALP_Area_AltRowOptions	✔	✔	✔	int	0	0	15	Alternate row coloring options: bit 0: 1 = enable, 0 = disable bit 1: 1 = apply ALP_Area_AltRowColor to even rows, 0 = apply to odd rows bit 2: 1 = alt color applies to empty space below the last row (if any) bit 3: 0 (default) = use the existing color when defined at cell or row level, instead of alternate color for alternate rows (column color is ignored) 1 = mix the alternate color with the existing color set for the cell/row/column (in this order)
ALP_Area_Appearance	✔	✔	✔	int	0	0	15	Appearance to use for native headers and checkboxes -1 - system (get the pictures from current system) 0 - automatic (use pictures best matching current system appearance) 1 - OS 9 Platinum 2 - OSX 8 Aqua 3 - Win XP 4 - Win Vista 5 - Win 7 6 - OSX 8 Graphite 7 - OSX 10 Aqua 8 - OSX Graphite 9 - Win 8 10 - Win 10 11 - pictures set by developer 12 - Win 11 (AL Force Win11 Interface) How to set the pictures for value 11: $err:=***AL_SetIcon*** (area; AL Appearance Normal PictID;$pictNormal) $err:=***AL_SetIcon*** (area; AL Appearance Hilite PictID;$pictHilite) $err:=***AL_SetIcon*** (area; AL Appearance Clicked PictID;$pictClicked) ***AL_SetAreaLongProperty*** (area; ALP_Area_Appearance;11)
ALP_Area_BottomRow	✔			int				The row number of the last (possibly partially) visible row on screen

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Display Properties				
ALP_Area_CalcAllRows	✔	✔		int	0	0	2	Calculate column width from all rows: 0 = no 1 = yes (AreaList Pro 8.x mode: only text (longest text, no support for attributed text) & pictures (widest picture), all other columns have default by type and format used) 2 = fully (can be slow on large arrays)
ALP_Area_ColDivColor	✔	✔	✔	color	#FF808080			Column divider color (default is gray)
ALP_Area_ColsInGrid	✔	✔	✔	int	-1	-1		Number of columns in grid
ALP_Area_ColsLocked	✔	✔	✔	int	0	0		Number of locked columns in grid
ALP_Area_ColumnLock	✔	✔	✔	bool	yes			Allow column lock by user
ALP_Area_ColumnResize	✔	✔	✔	bool	yes			Allow column resize by user If set to true, the column width will also automatically be adjusted when the user double-clicks on the column separator in the header
ALP_Area_ColumnResizeDistance	✔	✔	✔	real	3	1	16	Number of points (to left/right) from column divider to still handle as resize of header
ALP_Area_DrawFrame	✔	✔	✔	int	0	0	2	Draw frame: 0 = none 1 = black rectangle 2 = modern look (sunken)
ALP_Area_EmptyRowColor	✔	✔	✔	color	#00000000			Row color for empty rows below data Alternating row colors can be still used
ALP_Area_FrameColor	✔	✔	✔	color				
ALP_Area_FtrIndent	✔	✔	✔	point	3;2			Horizontal and vertical indents for the footer rows in points The first value represents the horizontal indent (left and right) and the second value is the vertical indent (top and bottom)
ALP_Area_FtrIndentH	✔	✔	✔	real	3	0	64	Horizontal indent for the footer rows, in points
ALP_Area_FtrIndentV	✔	✔	✔	real	2	0	64	Vertical indent for the footer rows, in points
ALP_Area_GroupHeader	✔	✔	✔	bool	0	1	2	Add a group header above (1) or replacing (2) column headers – Setup ALP_Column_GroupID and ALP_Column_GroupHdrText
ALP_Area_HdrIndent	✔	✔	✔	point	3;2			Horizontal and vertical indents for the header rows in points The first value represents the horizontal indent (left and right) and the second value is the vertical indent (top and bottom)
ALP_Area_HdrIndentH	✔	✔	✔	real	3	0	64	Horizontal indent for the header rows, in points
ALP_Area_HdrIndentV	✔	✔	✔	real	2	0	64	Vertical indent for the header rows, in points
ALP_Area_HdrPictID	✔	✔	✔	int	0			IconID to be used above the vertical scrollbar
ALP_Area_HeaderDivColor	✔	✔	✔	color				

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Display Properties				
ALP_Area_HeaderMode	✔	✔	✔	int	0	0	2	0 = native headers 1 = plain color rectangles 2 = sunken/raised color rectangles When both ALP_Area_HeaderMode and ALP_Area_ShowSortIndicator properties are not zero, the v8 sort order button is displayed above the vertical scrollbar
ALP_Area_HideHeaders	✔	✔	✔	bool	no			Hide headers
ALP_Area_HideLastHdrDivider	✔	✔	✔	bool	no			Do not draw the divider separator for last column (on the right)
ALP_Area_HierIndent	✔	✔	✔	real	16	0	64	Indent increment for every hierarchy level (for use with hierarchical lists)
ALP_Area_HighlightColor	✔	✔		int				The system highlight color can be read or overriden with ALP_Area_HighlightColor When set, this color will be used in place of the system highlight color To reset it to use the system color again, set the value to 0
ALP_Area_LimitRows	✔	✔		int	-1	-2		-1 (default) = display all rows >1 = limit the number of rows to display ("shrink" selection) Reset to -2 when a column is added or removed
ALP_Area_MaxCellHeight	✔	✔	✔	real	0	0	65536	Maximum cell height (to be used with pictures, long text or sub-ALP) When single-row grid is used, it is the maximum row height
ALP_Area_MaxRowLines	✔	✔	✔	int	0	0	64	When ALP_Area_NumRowLines = 0 (variable row height) and this property is set to a value greater than 0, at most ALP_Area_MaxRowLines will be displayed
ALP_Area_MinFtrHeight	✔	✔	✔	real	0	0	256	Minimum allowable height for the footer row in points
ALP_Area_MinHdrHeight	✔	✔	✔	real	0	0	256	Minimum allowable height for the header row in points
ALP_Area_MinRowHeight	✔	✔	✔	real	0	0	256	Minimum row height in points
ALP_Area_MiscColor1	✔	✔	✔	color	#FFEEEEEE			Area color above the vertical scrollbar
ALP_Area_MiscColor2	✔	✔	✔	color	#FFEEEEEE			This color is used as the background color: before drawing anything, the whole AreaList Pro area is erased using this color Default is light gray
ALP_Area_MiscColor3	✔	✔	✔	color	#FFEEEEEE			Area color left of the horizontal scrollbar Default is light gray
ALP_Area_MiscColor4	✔	✔	✔	color	#FFEEEEEE			Area color right of the horizontal scrollbar Default is light gray
ALP_Area_MiscColor5	✔	✔	✔	color	#00FFFFFF			Headers background. Transparent.
ALP_Area_NumFtrLines	✔	✔	✔	int	1	0	64	Number of lines within the footer 0 = variable height
ALP_Area_NumHdrLines	✔	✔	✔	int	1	0	64	Number of lines within the header 0 = variable height
ALP_Area_NumRowLines	✔	✔	✔	int	1	0	64	Number of lines within the row 0 = variable height

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Display Properties				
ALP_Area_OneLineFtr	✔	✔	✔	bool				One cell for each column footer regardless of ALP_Area_RowsInGrid in “grid” mode (ALP_Area_RowsInGrid >1) – Footer text is set by ALP_Column_FooterText of the array/ field defined for the first line of the grid
ALP_Area_OneLineHdr	✔	✔	✔	bool				One cell for each column header regardless of ALP_Area_RowsInGrid in “grid” mode (ALP_Area_RowsInGrid >1) – Header text is set by ALP_Column_HeaderText of the array/ field defined for the first line of the grid
ALP_Area_ResizeDuring	✔	✔	✔	bool	no			Calculate automatic column widths during row fetching, e.g. on scroll (when data are not yet in the cache)
ALP_Area_RowDivColor	✔	✔	✔	color	#FF808080			Row divider color (default is gray)
ALP_Area_RowHeight	✔			real				Initial height of every row
ALP_Area_RowHeightFixed	✔	✔	✔	bool	yes			Deprecated, superseded by ALP_Area_NumRowLines/ALP_Area_NumHdrLines/ ALP_Area_NumFtrLines with value(s) 0
ALP_Area_RowIndent	✔	✔	✔	text	3;1			Horizontal and vertical indents in points The first value represents the horizontal indent (left and right) and the second value is the vertical indent (top and bottom)
ALP_Area_RowIndentH	✔	✔	✔	real	3	0	64	Horizontal indent for the rows, in points
ALP_Area_RowIndentV	✔	✔	✔	real	1	0	64	Vertical indent for the rows, in points
ALP_Area_RowsInGrid	✔	✔	✔	int	1	-1	20	Number of rows in grid
ALP_Area_ScrollbarArrowsPos	✔		✔	int	-1	-1	3	-1 = use default (depends on appearance: none for OSX, normal otherwise) 0 = none 1 = normal (at top/left + at bottom/right) 2 = at min. end (both at top/left) 3 = at max. end (both at bottom/right)
ALP_Area_ScrollbarStyle	✔	✔	✔	bool	0	0	15	1 means “use overlay mode” - visible only while scrolling (with mouse wheel), scrollbar does not consume space in the area, it is drawn temporarily over the data
ALP_Area_ShowColDividers	✔	✔	✔	int	0	0	15	Show column dividers (using ALP_Area_ColDivColor): bit 0: draw bit 1: not in footer bit 2: draw in last columns (if there is empty space after right-most column, divider is drawn) bit 3: not in empty rows
ALP_Area_ShowFocus	✔	✔	✔	bool	yes			Show focus ring
ALP_Area_ShowFooters	✔	✔	✔	bool	no			Show footers
ALP_Area_ShowHScroll	✔	✔	✔	int	0 or 1	0	3	Show horizontal scrollbar: 0 = automatic, hidden 1 = automatic, shown 2 = manual, always hidden 3 = manual, always shown Default value is 0 or 1 depending on data and area width

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Display Properties				
ALP_Area_ShowRowDividers	✔	✔	✔	int	0	0	3	Show row dividers (using ALP_Area_RowDivColor) 0 - no 1 - yes 3 - yes, but not in empty area below last data row IOW set bit 0 to 1 to draw row dividers and bit 1 to 1 to hide dividers in empty area
ALP_Area_ShowSortIndicator	✔	✔	✔	int	1	0	2	If 0, no triangle (in header mode = 0) or no underline (header mode > 0) is drawn When both ALP_Area_HeaderMode and ALP_Area_ShowSortIndicator properties are not zero, the v8 sort order button is displayed above the vertical scrollbar On Windows Vista, 7 and 8, value 2 draws the sort (non native) triangle to the right, not on top.
ALP_Area_ShowVScroll	✔	✔	✔	int	0	0	3	Show vertical scrollbar 0 - manual not shown 1 - manual shown 2 - automatic, not shown 3 - automatic, shown Note that “auto” vs “manual” differs from ALP_Area_ShowHScroll usage for compatibility reasons
ALP_Area_ShowWidths	✔	✔	✔	int	0	0	3	0 - no action 1 = when modifiers are held down, display tooltip for current cell 2 - same as 1, but not in compiled mode 3 - always (compiled & uncompiled)” Changed the modifiers from command-option-shift to command-shift (ctrl-shift on Windows; better is to press shift, then ctrl; pressing ctrl and then shift changes the keyboard layout on some machines) When ALP_Area_ShowWidths is 3, the widths are displayed in headers (and printed with standalone printing) When ALP_Area_ShowWidths is not 0 and you option-click the resize area between columns (and possibly drag to resize it), the widths are displayed in header (temporarily while the mouse button is pressed) position mouse over a header and press all modifier keys to show a tool-tip
ALP_Area_SmallScrollbar	✔	✔	✔	bool	no			Use small scrollbars
ALP_Area_Transposed	✔	✔	✔	bool	no			Exchange (visually) the rows and columns
ALP_Area_TopRow	✔			int				The row number of the first (possibly partially) visible row on screen
ALP_Area_UseEllipsis	✔	✔	✔	int	0	0	2	AreaList Pro will automatically truncate data and display the standard ellipsis (…) when columns are resized smaller than the displayed data: 0 = none 1 = trailing for left aligned text, center otherwise 2 = trailing for left aligned text, leading for right aligned text, center otherwise
ALP_Area_Zoom	✔	✔		real	1	0.1	5	Zoom factor for every text in the area

AreaList Pro Area Drag & Drop Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Drag & Drop Properties								
ALP_Area_DragAcceptColumn	✔	✔	✔	bool	no			OBSOLETE: use Drag access codes instead
ALP_Area_DragAcceptLine	✔	✔	✔	bool	no			OBSOLETE: use Drag access codes instead
ALP_Area_DragColumn	✔	✔	✔	int	0	0	3	OBSOLETE: use Drag access codes instead
ALP_Area_DragDataType	✔			int				Dragged data: 1 = row(s) 2 = column 3 = cell(s)
ALP_Area_DragDstArea	✔			int				Destination area
ALP_Area_DragDstCell	✔			int				Destination area grid cell
ALP_Area_DragDstCellCodes	✔	✔	✔	text				Drag destination cell codes The format is a list of codes separated by '|'
ALP_Area_DragDstCol	✔			int				Destination area column
ALP_Area_DragDstColCodes	✔	✔	✔	text				Drag destination column codes The format is a list of codes separated by '|'
ALP_Area_DragDstProcessID	✔			int				4D's process ID of destination area
ALP_Area_DragDstRow	✔			int				Destination area row
ALP_Area_DragDstRowCodes	✔	✔	✔	text				Drag destination row codes The format is a list of codes separated by '|'
ALP_Area_DragLine	✔	✔	✔	int	0	0	6	OBSOLETE: use Drag access codes instead
ALP_Area_DragMoveColumns	✔	✔	✔	bool	No			Physically reorder columns on column drag (instead of modifying the grid)
ALP_Area_DragOptionKey	✔	✔	✔	bool	no			Drag rows using option key
ALP_Area_DragProcessID	✔			int				4D's process ID of this area
ALP_Area_DragRowMultiple	✔	✔	✔	bool	no			Allow multiple rows to be dragged
ALP_Area_DragRowOnto	✔	✔	✔	bool	yes			Show drag feedback: True: onto a row False: between rows
ALP_Area_DragScroll	✔	✔	✔	int	30	-1	30	Size of frame around the AreaList Pro area border where dragging will start area scrolling When the user drags something near the AreaList Pro area border, the contents will be scrolled -1 means "no scroll" (both directions!) when dragging row/cell
ALP_Area_DragSrcArea	✔			int				The dropped AreaList Pro area Zero if a different object was dropped (or AreaList Pro area from another application)
ALP_Area_DragSrcCell	✔			int				Source area grid cell
ALP_Area_DragSrcCellCodes	✔	✔	✔	text				Drag source cell codes The format is a list of codes separated by '|'
ALP_Area_DragSrcCol	✔			int				Source area column
ALP_Area_DragSrcColCodes	✔	✔	✔	text				Drag source column codes The format is a list of codes separated by '|'
ALP_Area_DragSrcRow	✔			int				Source area row (only if AreaList Pro is dropped on AreaList Pro)
ALP_Area_DragSrcRowCodes	✔	✔	✔	text				Drag source row codes The format is a list of codes separated by '|'

AreaList Pro Area Copy & Drag Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Copy & Drag Properties								
ALP_Area_CopyFieldSep	✔	✔	✔	text	TAB			Field separator for copy/drag operation See Copying or dragging from an AreaList Pro Area
ALP_Area_CopyFieldWrapper	✔	✔	✔	text				One character string The character used to “wrap” fields when the user copies selected rows to the clipboard This character will be placed both before and after each field If the value is the null (empty) string, then no character will wrap the fields
ALP_Area_CopyHiddenCols	✔	✔	✔	bool	no			Using Edit->Copy or a row drag, clipboard will contain all columns if set to true **Note**: Kept for compatibility, use ALP_Area_CopyOptions
ALP_Area_CopyOptions	✔	✔	✔	bool	no			Include the headers in the copied/dragged data (but only when the headers are not hidden with ALP_Area_HideHeaders)
ALP_Area_CopyRecordSep	✔	✔	✔	text	CR LF			Record separator for copy/drag operation

AreaList Pro Area Entry Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Entry Properties				
ALP_Area_AllowTabIntoPopup	✔	✔		bool	no			Hitting TAB while entering text in a cell moves to the next enterable cell (enterable by popup, but not by keyboard) and displays the cell popup Without this property, cells not enterable by keyboard are simply skipped (you must click on the popup icon)
ALP_Area_CallbackMethEntryEnd	✔	✔	✔	text				End entry callback function Called as: Handler (area; action) -> bool:Allow
ALP_Area_CallbackMethEntryStart	✔	✔	✔	text				Start entry callback method Called as: Handler (area; action; recLoaded)
ALP_Area_CallbackMethPopup	✔	✔	✔	text				Popup entry callback method (area; row; column; dataType) -> bool:Handled For popup handling: used when a popup is clicked but no popup array/menu is defined The callback is called as function: return False to invoke internal implementation, otherwise use ***AL_SetAreaXXXProperty*** ($1;ALP_Area_EntryValue;$value) to actually set the new value and return True See the example in the Callbacks section
ALP_Area_ClickDelay	✔	✔	✔	int	30	-2	300	Delay to start editing (in 1/60 sec.): -2 = use current double-click time 0 = disable
ALP_Area_EntryAllowArrows	✔	✔	✔	int	0	0	4	Arrow keys are used to move to next entry cell 0 - not allowed 1 - old behavior (modifier keys ignored) 2 - cmd+arrow 3 - option+arrow 4 - ctrl+arrow
ALP_Area_EntryAllowReturn	✔	✔	✔	bool	no			Allow RETURN in text
ALP_Area_EntryAllowSeconds	✔	✔	✔	bool	no			Allow seconds in time entry
ALP_Area_EntryCell	✔			cell				Row and grid cell number of current entry (row, cell)
ALP_Area_EntryCellColumn	✔			int				Visual cell number (can differ when using transposition)
ALP_Area_EntryCellRow	✔			int				Visual row number (can differ when using break processing/transposition)
ALP_Area_EntryClick	✔	✔	✔	int	0	0	7	How to start an entry: 0 = no way (even click-hold) 1 = single click 2 = double click 3 = command-double click 4 = shift-double click 5 = option-double click 6 = control-double-click 7 = click-hold only
ALP_Area_EntryColumn	✔			int				Column number of edited cell
ALP_Area_EntryEditedEntity	✔			object				Entity selection/collection: parent of the edited entity

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Entry Properties								
ALP_Area_EntryExit	✔	✔		bool				Exit the currently edited cell If there is not a cell being entered then ALP_Area_EntryExit will have no effect
ALP_Area_EntryFirstClickMode	✔	✔	✔	int	0	0	3	Determines how the first click is handled upon beginning entry (when using numeric/text entry) 0 = click is routed to the entry widget (default behavior) 1 = ignore click, select all when the value is NULL or the formatted value is empty string (numeric, date, time) 2 = ignore click, select all when the value is NOT NULL (numeric, date, time, text) 3 = ignore click, always select all (same behavior as when tabbing between the fields) Note: explicit setting of the highlighted text in the Cell entered callback is always honored
ALP_Area_EntryGotoCell	✔	✔		cell				Row and grid cell number to start entry in (row, cell)
ALP_Area_EntryGotoColumn	✔	✔		int				Column number to start entry in (first grid cell showing this column will be used)
ALP_Area_EntryGotoGridCell	✔	✔		int				Grid cell number to start entry in (cell in grid, not column number)
ALP_Area_EntryGotoRow	✔	✔		int				Row number to start entry in
ALP_Area_EntryGridCell	✔			int				Grid cell number of edited cell
ALP_Area_EntryHighlight	✔	✔		range				Entry highlight in the form: **String** ($startOfSelection)+"," +**String** ($endOfSelection)
ALP_Area_EntryHighlightE	✔	✔		int				Entry highlight end
ALP_Area_EntryHighlightS	✔	✔		int				Entry highlight start
ALP_Area_EntryInProgress	✔			bool				A cell is currently being edited
ALP_Area_EntryMapEnter	✔	✔	✔	int	0	0	3	Map Enter key to: 0 = nothing (ignore) 1 = Tab 2 = Return 3 = Tab for text fields, Return otherwise
ALP_Area_EntryMethod	✔	✔		int	depends on locale	0	15	Value is a bit-field defining when internal component has to be used for editing bit 0 (value 1) = styled text - default bit 1 (value 2) = plain text - default for RTL systems bit 2 (value 4) - integer/real/date/time bit 3 (value 8) - picture Default for LTR system is 1, default for RTL system is 3
ALP_Area_EntryModified	✔			bool				Currently edited cell value is modified Note: when a real number is to be edited, the cell is marked as modified if **Num** (**String** (oldValue)) # **Num** (currentValue) in order to ignore “epsilon” differences
ALP_Area_EntryPrevCell	✔			cell				Row and grid cell number of previous entry (row, cell)
ALP_Area_EntryPrevColumn	✔			int				Previously edited column number

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Entry Properties				
ALP_Area_EntryPrevGridCell	✔			int				Previously edited grid cell number
ALP_Area_EntryPrevRow	✔			int				Previously edited row number
ALP_Area_EntryRow	✔			int				Row number of edited cell
ALP_Area_EntrySelectedText	✔	✔		text				Selected text (entry must be in progress)
ALP_Area_EntrySkip	✔	✔		bool				Skip current cell
ALP_Area_EntryText	✔	✔		text				Entry text (entry must be in progress)
ALP_Area_EntryValue	✔	✔						Current entry value
ALP_Area_HideEntryFocus	✔	✔		int	0	0	3	Used as bitfield: bit 0 (value 1) - don't draw focus rectangle for text entry (text, int, real, date, time) bit 1 (value 2) - don't draw focus rectangle for other entry types (checkbox, radio, picture)
ALP_Area_IgnoreMenuMeta	✔	✔	✔	bool				Do not interpret meta characters when building popup menu
ALP_Area_IgnoreSoftDeselect	✔	✔	✔	bool	no			Deprecated since v10.5 Does nothing
ALP_Area_UseDateControls	✔	✔	✔	bool				Unsupported, does nothing
ALP_Area_UseTimeControls	✔	✔	✔	bool				Unsupported, does nothing

AreaList Pro Area Event Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Event Properties				
ALP_Area_AlpEvent	✔			int				Last AreaList Pro event: see AreaList Pro Event codes May be used with AreaRef set to zero (last event in any area)
ALP_Area_CallbackMethDeselect	✔	✔	✔	text				Area deselected callback method (area) See the Callbacks topic for more info
ALP_Area_CallbackMethMenu	✔	✔	✔	text				Edit menu callback function (area; event) -> long:flags See the Callbacks topic for more info See the list of the Edit menu constants
ALP_Area_CallbackMethOnEvent	✔	✔	✔	text				Event callback function (area; alpEvt; 4Devent; column; row; modifiers) See the Callbacks topic for more info
ALP_Area_CallbackMethSelect	✔	✔	✔	text				Area selected callback method (area)
ALP_Area_ClickedCell	✔			int				Last clicked grid cell
ALP_Area_ClickedCol	✔			int				Last clicked column
ALP_Area_ClickedColNum	✔			int				The real column number (differs from ALP_Area_ClickedCol only when using ALP_Area_Transposed)

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Event Properties								
ALP_Area_ClickedRow	✔			int				Last clicked row See Row Numbering
ALP_Area_ClickedRowNum	✔			int				The real row number (differs from ALP_Area_ClickedRow only when using ALP_Area_Transposed)
ALP_Area_ClickedVisualRow	✔			int				When using break processing, the second visual row is very likely the first data row (break header in first row)
ALP_Area_DoubleClick	✔			bool				Last click is double click
ALP_Area_Event	✔			int				Kind of event: 1 = mouse down 3 = key down 5 = auto key 18 = mouse moved 21 = area selected 22 = area deselected 25 = scroll 30 = undo 31 = cut 32 = copy 33 = paste 34 = clear 35 = select all 36 = redo 39 = mouse wheel
ALP_Area_Event_Filter	✔	✔		int	0	0	15	Mask to define which events should not be reported bit 0 (value 1): don't report Mouse Moved event (AL Mouse moved event = 18) bit 1 (value 2): don't report Key event (AL Key event = 19) bit 2 (value 4): don't report vertical scroll (AL Vertical scroll event = 7) bit 3 (value 8): report horizontal scroll (AL Horizontal scroll event = 16)
ALP_Area_EventChar	✔			text				Char (string) from keyboard
ALP_Area_EventKey	✔			int				Key code (e.g. 165 = KEY_F8)
ALP_Area_EventModifiers	✔			int				Event modifiers: 256 = command 512 = shift 1024 = caps lock 2048 = option 4096 = control
ALP_Area_EventPosH	✔			int				Horizontal mouse position
ALP_Area_EventPosV	✔			int				Vertical mouse position
ALP_Area_EventTarget	✔			int				Current area under the mouse (when you click, it will be made the focused target) For the area, it returns the focused target area ID, for sub-ALP it returns the parent row number In single AreaList Pro area, it is always the area, but when you click in a sub-alp column, it is the sub-area in that column
ALP_Area_FocusedTarget	✔			int				Current target for event handling For the area, it returns the focused target area ID, for sub-ALP it returns the parent row number In single AreaList Pro area, it is always the area, but when you click in a sub-alp column, it is the sub-area in that column

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Event Properties								
ALP_Area_HdrPictCallback	✔	✔	✔	text				Callback method for the click on user icon ($1 = area) Called as MyIconHandler (area)
ALP_Area_HdrPictFlags	✔	✔	✔	int	0	0	2	0 - no callback 1 - callback on mouseUp 2 - callback on mouseDown
ALP_Area_IgnoreMouseWheel	✔	✔		int				Used as bitfield: bit 0 (value 1) - ignore horizontal scrolling with mouse wheel bit 1 (value 2) - ignore vertical scrolling with mouse wheel
ALP_Area_RollOverCell	✔			int				Current cell (where the mouse is positioned) See Example 15 in the Tutorial section
ALP_Area_RollOverCol	✔			int				Current column (where the mouse is positioned)
ALP_Area_RollOverColNum	✔			int				The real column number (differs from ALP_Area_ClickedCol only when using ALP_Area_Transposed)
ALP_Area_RollOverRow	✔			int				Current row (where the mouse is positioned) See Row Numbering
ALP_Area_RollOverRowNum	✔			int				The real column number (differs from ALP_Area_ClickedRow only when using ALP_Area_Transposed)
ALP_Area_RollOverVisualRow	✔			int				When using break processing, the second visual row is very likely the first data row (break header in first row)
ALP_Area_SendEvent	✔	✔		int				Set custom event and execute the object method ALP_Area_AlpEvent will contain this event, ALP_Area_Event will be 0 Use e.g. -100 to not confuse your own code with regular AreaList Pro event codes Usable in subforms to **CALL SUBFORM CONTAINER**
ALP_Area_ToolTip	✔	✔		text				Tool Tip text To be set from the event callback function
ALP_Area_TypeAheadFieldMode	✔	✔		bool	no			Set to 1 (True) to allow type-ahead in fields mode
ALP_Area_TypeAheadString	✔	✔		text				This property returns the string typed by the user used in type-ahead Setter emulates the user's type-ahead
ALP_Area_TypeAheadTime	✔	✔		int		-1	300	This is a time in ticks (1/60 s) which is used to determine if a new type-ahead is starting (max time between two key dows to be handled as one string) Setting value to less than 5 will set the value to default, which is 2 * double-click time (from OS user's setting)

AreaList Pro Area Selection Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Selection Properties								
ALP_Area_SelClick	✔	✔	✔	int	2	0	3	What to report when the user clicks: 0 = nothing 1 = single and double click reports single click 2 = report single and double clicks 3 = report either single click or double click, not both (legacy v8.x mode)
ALP_Area_SelCol	✔	✔		int				Selected column
ALP_Area_Select	✔			int				Number of selected rows/cells (for multiple row/cell selection mode only)
ALP_Area_SelGotoRec	✔	✔		bool	no			Load record after selection change using **GOTO SELECTED RECORD** The area must be in fields mode and row selection mode Hint: in read/write mode the record will be locked
ALP_Area_SelHighlightMode	✔	✔	✔	int	0	0	2	Change the way the selected cells/rows are highlighted: 0 - system highlight color used as background 1 - invert colors 2 - blend the system highlight color with 75% alpha channel
ALP_Area_SelMultiple	✔	✔	✔	bool	no			Allow selection of multiple rows (in selection mode: “rows” = 0) Ignored if ALP_Area_SelType is not 0
ALP_Area_SelNoAutoSelect	✔	✔	✔	bool	no			Disable row selection when popup is clicked on unselected row Aautomatically set to true when compatibility is turned on
ALP_Area_SelNoCtrlSelect	✔	✔	✔	bool	no			Disable row selection when control-click occurs on unselected row
ALP_Area_SelNoDeselect	✔	✔	✔	bool	no			Disable row selection (deselection of other rows) when click occurs in already selected row and no modifier keys are held down.
ALP_Area_SelNoHighlight	✔	✔	✔	bool	no			Disable row highlight of selected rows (in selection mode: “rows” = 0) Ignored if ALP_Area_SelType is not 0
ALP_Area_SelNone	✔	✔	✔	bool	no			Allow no selection (in selection mode: “rows” = 0) Ignored if ALP_Area_SelType is not 0
ALP_Area_SelPreserve	✔	✔	✔	bool	no			Preserve row selection when sorting 4D's selection of rows: if set to true and area is in field mode, the row selection will be restored after sort Note: can be time-consuming for large selections, especially on client/server
ALP_Area_SelRow	✔	✔		int				Selected row: last clicked (or first row in selection when cmd-clicked to deselect the row in multiple row selection mode)
ALP_Area_SelSetName	✔	✔		text				Create named set after selection change Area must be in fields mode and row selection mode

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Selection Properties								
ALP_Area_SelType	✔	✔	✔	int	0	0	2	Selection mode: 0 = rows 1 = single cell 2 = multiple cells
ALP_Area_TypeAheadEffect	✔	✔		int	0	-2	2	-2 = report AL Typeahead event (do nothing else) -1 = ignore typeahead (do nothing) 0 = select first matching row (value >= “search”) 1 = select first matching row if selection is empty and scroll view to show first matching row otherwise 2 = change the selection to matching rows (value = “search@”) Set to 1 to get the old AreaList Pro v8 behavior: on typeahead, the selection is not changed when selection mode is multiple rows selection and the current selection is not empty - only the view is scrolled to show the first matching row Was ALP_Area_SelKeepOnTypeAhead (value “selT”) in previous versions

AreaList Pro Area Sort Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Sort Properties								
ALP_Area_AllowSortEditor	✔	✔	✔	bool	no			Allow sort editor by user (cmd-click on header)
ALP_Area_DontSortArrays	✔	✔	✔	bool	no			Do not sort arrays Note: when set to true, arrays are not reordered by the sort Internal array of sort order is maintained: see ***AL_GetObjects*** with ALP_Area_Sort in the Internal Sorting topic
ALP_Area_ShowSortEditor	✔	✔						Show sort editor The ***AL_GetAreaLongProperty*** getter command returns 1 if the user clicked Sort, 0 otherwise
ALP_Area_Sort	✔			int				Number of elements in sort list (the number of columns that were sorted)
ALP_Area_SortCancel	✔	✔	✔	text				Sort editor Cancel button label If empty, defaults to "Cancel" or its translation from the “ALP.xlf” file located in the localized subfolder of the Resources folder in the AreaList Pro bundle
ALP_Area_SortColumn	✔		✔	int				Sorted column number (use ALP_Area_SortList to sort)
ALP_Area_SortDuring	✔	✔	✔	bool	no			Sets a flag for a “permanent” sort = ie, automatically keep the data sorted of it changes
ALP_Area_SortList	✔	✔	✔	text or int				Int: sort column number Text: sort list – The format is a comma-separated list of column numbers Use negative number for descending order

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Sort Properties								
ALP_Area_SortListNS	✔	✔	✔	text or int				“No sort” sort list or number See Taking control of the sort for more information and an example Int: sort column number Text: sort list – The format is a comma-separated list of column numbers Use negative number for descending order Only sets the sort list, does not actually sort the data
ALP_Area_SortOK	✔	✔	✔	text				Sort editor OK button label If empty, defaults to "Sort" or its translation from the “ALP.xlf” file located in the localized subfolder of the Resources folder in the AreaList Pro bundle
ALP_Area_SortOnLoad	✔	✔	✔	bool	no			Sort data on area initialization (when loaded from advanced properties or XML)
ALP_Area_SortPrompt	✔	✔	✔	text				Prompt for sort editor If empty, defaults to "Select columns to sort" or its translation from the “ALP.xlf” file located in the localized subfolder of the Resources folder in the AreaList Pro bundle
ALP_Area_SortTitle	✔	✔	✔	text				Title for sort editor If empty, defaults to "Sort Options" or its translation from the “ALP.xlf” file located in the localized subfolder of the Resources folder in the AreaList Pro bundle
ALP_Area_UserSort	✔	✔	✔	int	1	0	3	Sort by user: 0 = disabled 1 = enabled 2 = bypassed 3 = only indexed fields

AreaList Pro Area Plugin Properties

Note: the following properties always expect 0 as the area reference: the workstation global setting is accessed.

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Plugin Properties				
ALP_Area_CalendarColors	✔	✔		text				8 ARGB colors separated with "|" to be used by the date «calendar» popup First 5 colors define object backgrounds: active month, inactive month, selected date, current date, current selected date Next 2 colors define foreground: numbers in active month, numbers in inactive month 8th value is the popup background color; it needs a non-zero alpha channel to be set, e.g. #FFE9F1FF instead of #E9F1FF When not set explicitly, default colors depend on ALP_Area_CalendarLook To restore the default colors (as if ALP_Area_CalendarColors was not set), pass an empty text value Default values are: "#00FFFFDD|#00EEEEEE|#00EEAAAA|#00FF8888|#00FF5555|#00000000|#00444444|#00CCCCCC" for the regular (default) calendar look, and: "#FFFFFFFE|#FFFFFFFE|#00EEEEEE|#00FF8888|#008F8F8F|#00000000|#00444444|#FFFFFFFC" for the alternate Date popup (according to ALP_Area_CalendarLook)
ALP_Area_CalendarLook	✔	✔		bool	no			If set, AreaList Pro will use an alternate Date popup ("Windows look") When ALP_Area_CalendarColors is not set explicitly, the color scheme is different and the "correct one" will be used (actually the ALP_Area_CalendarColors value is updated when ALP_Area_CalendarLook is modified)
ALP_Area_Copyright	✔			text				Copyright of the AreaList Pro plugin
ALP_Area_DefFmtBoolean	✔	✔		text				Default format for Boolean columns Initialized from the "AreaList™ Pro Format Defaults" group from the "ALP.xlf" file located in the localized subfolder of the Resources folder in the AreaList Pro bundle
ALP_Area_DefFmtDate	✔	✔		text				Default format for Date columns Initialized as above
ALP_Area_DefFmtInteger	✔	✔		text				Default format for Integer columns Initialized as above
ALP_Area_DefFmtLong	✔	✔		text				Default format for Long Integer columns Initialized as above
ALP_Area_DefFmtPicture	✔	✔		text				Default format for Picture columns Initialized as above
ALP_Area_DefFmtReal	✔	✔		text				Default format for Real columns Initialized as above

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Plugin Properties								
ALP_Area_DefFmtTime	✔	✔		text				Default format for Time columns Default is “0”
ALP_Area_DPI_X	✔			real				current DPI - X value
ALP_Area_DPI_Y	✔			real				current DPI - Y value
ALP_Area_FillNumberSign	✔	✔		bool	no			If set to 1 (True), string formatting replaces the place holders ‘#’ with non-breaking space (old behavior), otherwise unused part of the format string is removed.
ALP_Area_LastError	✔	✔		int				Last error in ANY AreaList Pro area
ALP_Area_Path	✔			text				Path to the AreaList Pro plugin
ALP_Area_SRPTableTemplate	✔			text				Get the SuperReport Pro template used for report creation (stored in Resources/Table Report.xml) as XML
ALP_Area_TraceOnError	✔	✔		int	1	0	3	Invoke the 4D debugger in interpreted and/or an alert in compiled if a command causes an error, and it is a command that does not return an error code (or if the license does not allow to use the command) 0 - none 1 - TRACE interpreted 2 - ALERT compiled 3 - TRACE interpreted, ALERT compiled Hold down the alt/option key when clicking OK in the alert to kill “endless” ALERT - this will clear bit 1 of ALP_Area_TraceOnError
ALP_Area_Version	✔			text				Version of the AreaList Pro plugin Note that getting this property will not trigger the registration dialog if AreaList Pro is not registered (allows to check version before registering)

AreaList Pro Column Properties

Use these constants with commands in the Columns command theme:

AL_GetColumnLongProperty

AL_GetColumnPtrProperty

AL_GetColumnRealProperty

AL_GetColumnTextProperty

AL_SetColumnLongProperty

AL_SetColumnPtrProperty

AL_SetColumnRealProperty

AL_SetColumnTextProperty

Global Column settings:

- Column = 0 : apply to all columns to be created (defining a column “template”) in either a specific area, or in area 0, -2 or -3 as described in the Area Properties section.
- Column = -2 : apply to all existing columns in either a specific area (from 1 to ALP_Area_Columns), or in area -2, -3 or -3 as above for existing areas.

Column General Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				Column General Properties				
ALP_Column_Attributed	✔	✔	✔	bool	no			Uses attributed (multi-style) text: 0 = no, 1 = yes See also AreaList Pro Text Style Tags
ALP_Column_CalcHeight	✔	✔	✔	bool	no			Automatically set row height based on data in this column Row heights are fixed when ALP_Area_NumRowLines is non-zero Otherwise the row heigh is determined by columns having ALP_Column_CalcHeight = 1 This also holds for headers and footers, see Row Numbering
ALP_Column_Calculated	✔		✔	bool				This column is a calculated column Can only be set in array mode, use AL_AddCalculatedColumn in field mode
ALP_Column_Callback	✔	✔	✔	text				Callback method for a calculated column (area; column; type; ptr; first; count) If the AreaList Pro area displays several Calculated Columns, the callback methods will be called in the column number order
ALP_Column_CallbackFnc	✔	✔	✔	text				Callback function for "formula" column
ALP_Column_ColDivColor	✔	✔	✔	color	#00000000			Column divider color (on right side of this column), set to non-zero to be used (set to 1 to not draw)

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Column General Properties								
ALP_Column_ColDivThickness	✔	✔	✔	real	0	0	5	Column divider thickness (on right side of this column), set to zero to use AreaList Pro's value
ALP_Column_DisplayControl	✔	✔	✔	int	-1	-1	4	Display control type: -1 = default (formatted value) 0 = checkbox without title 1 = small checkbox without title 2 = mini checkbox without title 3 = mapped through ALP_Column_PopupArray +ALP_Column_PopupMap or ALP_Column_PopupMenu (these 3 properties must be defined) 4 = use pictures (see Displaying custom checkboxes using pictures)
ALP_Column_Draggable	✔	✔	✔	bool	yes			Allow column to be dragged
ALP_Column_Enterable	✔	✔	✔	int	1	0	7	Enterablility: 0 = AL Column entry off (not enterable) 1 = AL Column entry typed only (by keyboard) 2 = AL Column entry popup only (by popup) 3 = AL Column entry both (by keyboard & popup) 4 = AL Column entry popup no meta (by popup ignoring menu meta) 5 = AL Column entry both no meta (by keyboard & popup ignoring menu meta) 6 = AL Column entry popup cellwide (click anywhere into the cell is handled the same as click on the popup icon) 7 = AL Column entry popup cell NM (click anywhere into the cell is handled the same as click on the popup icon, ignoring menu meta)
ALP_Column_EntryAllowReturn	✔	✔	✔	Boolean				Allow RETURN in text (changed when you set ALP_Area_EntryAllowReturn) Default value : inherited from area ALP_Area_EntryAllowReturn
ALP_Column_EntryControl	✔	✔	✔	int	0	0	2	Entry control, depending upon column type (boolean or integer/long integer) For boolean columns: 0 = checkbox without title 1 = checkbox with title 2 = radio buttons For integer/long integer columns: 0 = 2-states checkbox (values 0, 1) 1 = 3-states checkbox (values 0, 1, 2) (ALP_Column_DisplayControl must be set to 0, 1, 2 or 4 in order to use checkboxes in integer/long integer columns)
ALP_Column_Filter	✔	✔	✔	text				Entry filter
ALP_Column_FindCell	✔			int				Find the first grid cell number showing data from the column
ALP_Column_FocusableCheckbox	✔	✔	✔	bool	no			When set to 1, column displaying checkboxes (ALP_Column_DisplayControl = 0, 1 or 2) is focusable as any other column type
ALP_Column_FooterText	✔	✔	✔	text				Footer text

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Column General Properties								
ALP_Column_Format	✔	✔	✔	text				Format For picture columns: "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
ALP_Column_FormatResolved	✔			text				Resolved 4D format (when format uses "|Format")
ALP_Column_FromCell	✔			int				Get the column number from the grid cell number
ALP_Column_GroupHdrText	✔	✔	✔	text				Text to use in group header (displayed when ALP_Area_GroupHeader is set to 1 or 2)
ALP_Column_GroupID	✔	✔	✔	int				ID for grouping columns together
ALP_Column_HeaderText	✔	✔	✔	text				Header text
ALP_Column_ID	✔		✔	int				Column number (numbered from 1)
ALP_Column_Indexed	✔			bool				Field is indexed
ALP_Column_Kind	✔		✔	text				Object kind = "Column" To access default column font properties, use AreaRef and Column set to zero
ALP_Column_Length	✔			int				Size of the alpha field Zero means it is not an alpha (length-limited) field
ALP_Column_Locked	✔			bool				Column is locked
ALP_Column_MaxWidth	✔	✔	✔	real	32000	0	32000	Maximum column width
ALP_Column_MinWidth	✔	✔	✔	real	0	0	512	Minimum column width
ALP_Column_PlaceHolder	✔	✔	✔	text				Entry placeholder Usable only with component entry (4D Form), inline entry does not support it
ALP_Column_PopupArray	✔	✔	✔	text/ array				Use a pointer to an array For Get when using an array: the array type must match See the Value Mapping example If text is to be used (not array), elements are separated by **Char**(3) (ASCII ETX) To ignore menu meta characters in a row, start that row with **Char**(1) (ASCII SOH)
ALP_Column_PopupArrayKind	✔	✔	✔	int				Type of the popup array (4D constants): 5 = Is undefined 14 = Is real array 15 = Is integer array 16 = Is longint array 17 = Is date array 18 = Is text array
ALP_Column_PopupEntryType	✔	✔	✔	int	0	0	2	Popup entry type; 0 = default (depends on data type), 1 = date, 2 = time

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				Column General Properties				
ALP_Column_PopupIconID	✔	✔	✔	int	0			IconID to use as popup icon; 0 = use default (depends on data type)
ALP_Column_PopupMap	✔	✔	✔	text/text array				If set, the popup will be built from this array, but values will be used from ALP_Column_PopupArray See the Value Mapping example If text is to be used (not array), elements are separated by **Char**(3) (ASCII ETX) To ignore menu meta characters in a row, start that row with **Char**(1) (ASCII SOH)
ALP_Column_PopupMenu	✔	✔		text				Associated 4D menu Use **Create Menu** See the Value Mapping example
ALP_Column_PopupName	✔	✔	✔	text				Internal (used in advanced properties)
ALP_Column_Proportional	✔	✔	✔	bool	no			When set to 1, this column is used in automatic column resizing see ALP_Area_AutoresizeAllColumns
ALP_Column_Resizable	✔	✔	✔	bool				Allow column to be resized
ALP_Column_Reveal	✔	✔		n/a				Reveal (make visible) this column
ALP_Column_ScrollTo	✔	✔		n/a				If visible, scroll the area to position this column on the left
ALP_Column_Sortable	✔	✔	✔	bool	depends on data type (e.g. no for Picture)			Allow column to be sorted Implicitly set to zero for calculated columns Set it to true for a calculated column in Fields mode to be sorted by the user with ALP_Column_SortFormula
ALP_Column_SortDirection	✔		✔	int	0	-1	1	Current sort direction
ALP_Column_SortFormula	✔	✔	✔	text				Allow sorting selection by calculated column in Fields mode See Sorting a Calculated Column (field mode)
ALP_Column_Source	✔		✔	text				Data source Array name or [MyTable]MyField or entity name (“This.myEntity”)
ALP_Column_SubALP	✔		✔	ALP	0			Sub-ALP handle - area displayed in this column (when column type is AL SubALP Column Type)
ALP_Column_Type	✔		✔	int				Data type (4D constants): 0 = Is Alpha Field 1 = Is Real 2 = Is Text 3 = Is Picture 4 = Is Date 6 = Is Boolean 8 = Is Integer 9 = Is LongInt 11 = Is Time
ALP_Column_Uppercase	✔	✔	✔	bool				Make uppercase
ALP_Column_UserCollection	✔	✔	✔					Collection for free use by developer Use with ***AL_Get/SetColumnPtrProperty*** and a pointer to a collection
ALP_Column_UserText	✔	✔	✔	text				Text for free use by developer (XML or other), associated to the column

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Column General Properties								
ALP_Column_Visible	✔	✔	✔	bool	yes			Column is visible
ALP_Column_Width	✔	✔	✔	real	0	0	32000	Column width in points These two values are related, and the setter will change both properties at the same time if the value is non-zero (both properties will have this same value) A zero value set to either property means automatic width: ALP_Column_WidthUser will return zero, but ALP_Column_Width will return the actual width calculated from the data When the user sets the value by doing a column resize, then again, both properties will be set to the same value But when the user double-clicks in the column resize, ALP_Column_WidthUser will be set to zero and ALP_Column_Width will be calculated from the data ALP_Area_ColumnResize must be set to true to enable column width changes by the setter or the user
ALP_Column_WidthUser	✔	✔	✔	real	0	0	32000	
ALP_Column_XML	✔	✔		text				Full description of the column in XML
PLP_Column_RepeatValues	✔	✔	✔	bool	yes			Repeat duplicated values honored only during printing.

Column Header Style Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				Column Header Style Properties				
ALP_Column_HdrBackColor	✔	✔	✔	color	#00FFFFFF			Header background color (unused when ALP_Area_HeaderMode = 0)
ALP_Column_HdrBaseLineShift	✔	✔	✔	real	0	-100	256	Baseline shift
ALP_Column_HdrFontName	✔	✔	✔	text	Verdana on Windows Lucida Grande on MacOS			Header font name
ALP_Column_HdrHorAlign	✔	✔	✔	long int	0	0	5	Header horizontal alignment: 0 = default 1 = left 2 = center 3 = right 4 = justify 5 = full justify
ALP_Column_HdrHorizontalScale	✔	✔	✔	real	1	0,1	100	Header horizontal scale
ALP_Column_HdrLineSpacing	✔	✔	✔	real	1.0	1	10	Header line spacing
ALP_Column_HdrRotation	✔	✔	✔	real	0	-360	360	Rotation of text in header
ALP_Column_HdrSize	✔	✔	✔	real	12 on Windows 13 on MacOS	4	128	Header font size
ALP_Column_HdrStyleB	✔	✔	✔	bool	no			Header font style = bold
ALP_Column_HdrStyleF	✔	✔	✔	int	0	0	7	Header font style, using 4D style constants (e.g. Bold, Italic, etc.)
ALP_Column_HdrStyleI	✔	✔	✔	bool	no			Header font style = italic
ALP_Column_HdrStyleU	✔	✔	✔	bool	no			Header font style = underlined
ALP_Column_HdrTextColor	✔	✔	✔	color	#FF000000			Header font color Default is black
ALP_Column_HdrUseEllipsis	✔	✔	✔	int	-1	-1	2	Use ellipsis for header of this column -1 - use area's default (ALP_Area_UseEllipsis) 0 - none 1 - on right side for left-aligned, in center otherwise 2 - on right side for left-aligned, on left side for right-aligned, in center otherwise
ALP_Column_HdrVertAlign	✔	✔	✔	long int	2	0	3	Header vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Column_HdrWrap	✔	✔	✔	bool	no			Wrap long lines in header

Column List Style Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
					Column List Style Properties			
ALP_Column_BackColor	✔	✔	✔	color	#00FFFFFF			Background color Default is transparent (no color)
ALP_Column_BaseLineShift	✔	✔	✔	real	0	-100	256	Baseline shift
ALP_Column_FontName	✔	✔	✔	text	Verdana on Windows Lucida Grande on MacOS			List font name
ALP_Column_HorAlign	✔	✔	✔	long int	0	0	5	List horizontal alignment: 0 = default 1 = left 2 = center 3 = right 4 = justify 5 = full justify 6 = align text on the decimal separator (used only for real columns, for other type of data 6 behaves as 3)
ALP_Column_HorizontalScale	✔	✔	✔	real	1	0,1	100	List horizontal scale
ALP_Column_LineSpacing	✔	✔	✔	real	1.0	1	10	List line spacing
ALP_Column_Rotation	✔	✔	✔	real	0	-360	360	Rotation of text in list
ALP_Column_Size	✔	✔	✔	real	12 on Windows 13 on MacOS	4	128	List font size
ALP_Column_StyleB	✔	✔	✔	bool	no			List font style = bold
ALP_Column_StyleF	✔	✔	✔	long int	0	0	7	List font style, using 4D style constants (e.g. Bold, Italic, etc.)
ALP_Column_StyleI	✔	✔	✔	bool	no			List font style = italic
ALP_Column_StyleU	✔	✔	✔	bool	no			List font style = underlined
ALP_Column_TextColor	✔	✔	✔	color	#FF000000			List font color Default is black
ALP_Column_UseEllipsis	✔	✔	✔	int	-1	-1	2	Use ellipsis for data of this column -1 - use area's default (ALP_Area_UseEllipsis) 0 - none 1 - on right side for left-aligned, in center otherwise 2 - on right side for left-aligned, on left side for right-aligned, in center otherwise
ALP_Column_VertAlign	✔	✔	✔	long int	0	0	3	List vertical alignment 0 = default 1 = top 2 = center 3 = bottom
ALP_Column_Wrap	✔	✔	✔	bool	no			Wrap long lines in list

Column Footer Style Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				Column Footer Style Properties				
ALP_Column_FtrBackColor	✔	✔	✔	color	#00FFFFFF			Background color Default is transparent (no color)
ALP_Column_FtrBaseLineShift	✔	✔	✔	real	0	-100	256	Footer baseline shift
ALP_Column_FtrFontName	✔	✔	✔	text	Verdana on Windows Lucida Grande on MacOS			Footer font name
ALP_Column_FtrHorAlign	✔	✔	✔	long int	0	0	5	Footer horizontal alignment: 0 = default 1 = left 2 = center 3 = right 4 = justify 5 = full justify
ALP_Column_FtrHorizontalScale	✔	✔	✔	real	1	0,1	100	Footer horizontal scale
ALP_Column_FtrLineSpacing	✔	✔	✔	real	1.0	1	10	Footer line spacing
ALP_Column_FtrRotation	✔	✔	✔	real	0	-360	360	Rotation of text in footer
ALP_Column_FtrSize	✔	✔	✔	real	12 on Windows 13 on MacOS	4	128	Footer font size
ALP_Column_FtrStyleB	✔	✔	✔	bool	no			Footer font style = bold
ALP_Column_FtrStyleF	✔	✔	✔	long int	0	0	7	Footer font style, using 4D style constants (e.g. Bold, Italic, etc.)
ALP_Column_FtrStyleI	✔	✔	✔	bool	no			Footer font style = italic
ALP_Column_FtrStyleU	✔	✔	✔	bool	no			Footer font style = underlined
ALP_Column_FtrTextColor	✔	✔	✔	color	#FF000000			Footer font color Default is black
ALP_Column_FtrUseEllipsis	✔	✔	✔	int	-1	-1	2	Use ellipsis for footer of this column -1 - use area's default (ALP_Area_UseEllipsis) 0 - none 1 - on right side for left-aligned, in center otherwise 2 - on right side for left-aligned, on left side for right-aligned, in center otherwise
ALP_Column_FtrVertAlign	✔	✔	✔	long int	2	0	3	Footer vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Column_FtrWrap	✔	✔	✔	bool	no			Wrap long lines in footer

AreaList Pro Row Properties

Use these constants with commands in the Rows command theme:

AL_GetRowLongProperty

AL_GetRowPtrProperty

AL_GetRowRealProperty

AL_GetRowTextProperty

AL_SetRowLongProperty

AL_SetRowPtrProperty

AL_SetRowRealProperty

AL_SetRowTextProperty

Row Numbering

Row	Value
Header	0
Body rows	1 to number of rows
Footer	-1
Empty area below last row	-2

The values above are returned by the ALP_Area_ClickedRow and ALP_Area_RollOverRow properties:

```
$clickedRow:=AL_GetAreaLongProperty (area;ALP_Area_ClickedRow)   // last clicked row
$rowUnder:=AL_GetAreaLongProperty (area;ALP_Area_RollOverRow)   // row currently under the pointer
```

They can also be used to set and get properties (except the two row properties above).

The default style is None (the Column style is used: all rows default to column properties, give or take alternate row coloring when in effect).

The row number value -2 has two meanings, depending on whether it is used to get the row information or to set a property:

- Global Row settings: if the row number is -2, a settable row property will be applied to all rows in either a specific area (from 1 to ALP_Area_Rows), or in area -2 or -3 as described in the Area Properties section.
- The “Empty area below last row” (-2) value is reported by ALP_Area_ClickedRow or ALP_Area_RollOverRow when a click occurs or the pointer is over the area between the last row and the footer / horizontal scrollbar / bottom of the AreaList Pro area (i.e. the space without data rows).

Row General Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Row General Properties								
ALP_Row_Clear		✔		n/a				Clear all properties of this row
ALP_Row_Height	✔			real	area's row height	0	32000	Height of the row in points
ALP_Row_Hide	✔			bool	no			Set to 1 to hide a row
ALP_Row_Kind	✔		✔	text				Object kind = "Row"
ALP_Row_Reveal		✔		n/a				Reveal (make visible) this row
ALP_Row_RowOffset	✔			real				Offset from top
ALP_Row_ScrollTo		✔		n/a				If visible, scroll the area to position this row on the top ***AL_SetRowLongProperty*** (area; $row; ALP_Row_ScrollTo) Note: no property value is needed
ALP_Row_XML	✔	✔		text				Full description of the row in XML Note: XML does not contain Style – use ALP_Row_StyleXML for that

Row Style Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Row Style Properties								
ALP_Row_BackColor	✔	✔	✔	color				Background color
ALP_Row_BaseLineShift	✔	✔	✔	real		-100	256	Baseline shift
ALP_Row_ClearStyle		✔		n/a				Clear the style of this row The area redraws automatically
ALP_Row_Flags	✔	✔	✔	int				Bit-mask of set features Properties not set are inherited from the column settings The following flags indicate what style options have been set at the row level: 2 = font name 4 = font size 8 = font style 16 = text color 32 = background color 64 = horizontal alignment 128 = vertical alignment 256 = wrap 512 = rotation 1024 = baseline shift 2048 = horizontal scale 4096 = line spacing 8192 - ellipsis Maintained by AreaList Pro and should not normally be changed You can clear the flag if you want to force AreaList Pro to abandon row-specific settings

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Row Style Properties								
ALP_Row_FontName	✔	✔	✔	text				Font name
ALP_Row_HorAlign	✔	✔	✔	int		0	5	Horizontal alignment: 0 = default 1 = left 2 = center 3 = right 4 = justify 5 = full justify
ALP_Row_HorizontalScale	✔	✔	✔	real		0,1	100	Horizontal scale
ALP_Row_LineSpacing	✔	✔	✔	real	1.0	1	10	Line spacing
ALP_Row_Rotation	✔	✔	✔	real		-360	360	Rotation of text
ALP_Row_RowDivColor	✔	✔	✔	color				Row divider color
ALP_Row_Size	✔	✔	✔	real		4	128	Font size
ALP_Row_StyleB	✔	✔	✔	bool				Font style = bold
ALP_Row_StyleF	✔	✔	✔	int		0	7	Font style, using 4D style constants (e.g. Bold, Italic, etc.)
ALP_Row_StyleI	✔	✔	✔	bool				Font style = italic
ALP_Row_StyleU	✔	✔	✔	bool				Font style = underlined
ALP_Row_StyleXML	✔	✔		text				Description of the row style in XML
ALP_Row_TextColor	✔	✔	✔	color				Font color
ALP_Row_UseEllipsis	✔	✔	✔	int	0	0	2	Use ellipsis for data of this row -1 - use area's default (ALP_Area_UseEllipsis) 0 - none 1 - on right side for left-aligned, in center otherwise 2 - on right side for left-aligned, on left side for right-aligned, in center otherwise
ALP_Row_VertAlign	✔	✔	✔	int		0	3	Vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Row_Wrap	✔	✔	✔	bool				Wrap long lines

Row Hierarchy Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				Row Hierarchy Properties				
ALP_Row_ChildCount	✔			int				Hierarchy: count of immediate children
ALP_Row_ChildCountAll	✔			int				Hierarchy: count of all children
ALP_Row_Children	✔			integer/ longint array				Hierarchy: get immediate children row numbers
ALP_Row_ChildrenAll	✔			integer/ longint array				Hierarchy: get all children rows numbers
ALP_Row_Collapse	✔	✔		bool				Collapse this row (all children will be invisible)
ALP_Row_CollapseAll		✔		bool				"Deep collapse": collapse this row and all its children (all children will be invisible and collapsed)
ALP_Row_Expand	✔	✔		bool				Show children of this row If any child was not collapsed, more levels will be visible
ALP_Row_ExpandAll		✔		bool				"Deep expand": show children of this row and all children (all children will be visible and fully expanded)
ALP_Row_Level	✔			int				Returns the level associated with this row (set using ALP_Object_Hierarchy)
ALP_Row_Parent	✔			int				Returns the immediate parent of this row (zero if this row is at the top level)
ALP_Row_Visible	✔			bool				Returns whether this row is visible (all parents of this row are expanded) This has nothing to do with real visibility on screen, but with the expanded state of all parents

AreaList Pro Cell Properties

Use these constants with commands in the commands in the Cells command theme:

AL_GetCellLongProperty

AL_GetCellPtrProperty

AL_GetCellRealProperty

AL_GetCellTextProperty

AL_SetCellLongProperty

AL_SetCellPtrProperty

AL_SetCellRealProperty

AL_SetCellTextProperty

Global Cell settings:

- If the Row Number is -2, the property will be applied to all rows displaying data (from 1 to ALP_Area_Rows) for the specified Column Number.
- If the Column Number is -2, the property will be applied to all columns in the area (from 1 to ALP_Area_Columns) for the specified Row Number.
- If both the Row Number and Column Number are -2, the property will be applied to all cells in the area.

For example, to clear all special formatting for all cells:

AL_SetCellLongProperty (area; -2; -2; ALP_Cell_ClearStyle; 0)

To clear all special formatting for cells in column 3:

AL_SetCellLongProperty (area; -2; 3; ALP_Cell_ClearStyle; 0)

To clear all special formatting for cells in row 3:

AL_SetCellLongProperty (area; 3; -2; ALP_Cell_ClearStyle; 0)

Cell General Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				Cell General Properties				
ALP_Cell_BottomBorderColor	✔	✔	✔	color				Bottom border color
ALP_Cell_BottomBorderOffset	✔	✔	✔	int				Bottom border offset in points
ALP_Cell_BottomBorderWidth	✔	✔	✔	int				Bottom border width in points
ALP_Cell_Clear		✔		n/a				Clear all properties of this cell The area redraws automatically
ALP_Cell_Enterable	✔	✔	✔	int	-1	-1	7	Enterablility: 0 = AL Cell entry off (not enterable) 1 = AL Cell entry typed only (by keyboard) 2 = AL Cell entry popup only (by popup) 3 = AL Cell entry both (by keyboard & popup) 4 = AL Cell entry popup no meta (by popup ignoring menu meta) 5 = AL Cell entry both no meta (by keyboard & popup ignoring menu meta) 6 = AL Cell entry popup cellwide (click anywhere into the cell is handled the same as click on the popup icon) 7 = AL Cell entry popup cellwide_NM (click anywhere into the cell is handled the same as click on the popup icon, ignoring menu meta)
ALP_Cell_FillColor	✔	✔	✔	color				Color used to fill the border rectangle
ALP_Cell_Format	✔	✔	✔	text				
ALP_Cell_FormatResolved	✔			text				Resolved 4D format (when format uses "|Format")
ALP_Cell_FormattedValue	✔			text				Formatted cell value
ALP_Cell_Invisible	✔	✔	✔	bool	no			If set to yes, cell content is made invisible Invisible cells draw nothing (except borders and disclosure triangle), are implicitly not enterable and are not copied when using Copy or Drag
ALP_Cell_Kind	✔		✔	text				Object kind = “CellOptions”
ALP_Cell_LeftBorderColor	✔	✔	✔	color				Left border color
ALP_Cell_LeftBorderOffset	✔	✔	✔	int				Left border offset
ALP_Cell_LeftBorderWidth	✔	✔	✔	int				Left border width
ALP_Cell_LeftIconFlags	✔	✔	✔	int				Offset/width Horizontal position Vertical position Scaling Mask See the section on Icon Flags for more
ALP_Cell_LeftIconID	✔	✔	✔	int				Left icon ID (see AL_SetIcon)
ALP_Cell_Reveal		✔		n/a				Reveal (make visible) this cell
ALP_Cell_RightBorderColor	✔	✔	✔	color				Right border color

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Cell General Properties								
ALP_Cell_RightBorderOffset	✔	✔	✔	int				Right border offset in points
ALP_Cell_RightBorderWidth	✔	✔	✔	int				Right border width in points
ALP_Cell_RightIconFlags	✔	✔	✔	int				Offset/width Horizontal position Vertical position Scaling Mask See the section on Icon Flags for more
ALP_Cell_RightIconID	✔	✔	✔	int				Right icon ID (see AL_SetIcon)
ALP_Cell_RowDivColor	✔	✔	✔	color				
ALP_Cell_ScrollTo	✔	✔		n/a				If visible, scroll the area to position this cell on the top left
ALP_Cell_TopBorderColor	✔	✔	✔	color				Top border color
ALP_Cell_TopBorderOffset	✔	✔	✔	int				Top border offset in points
ALP_Cell_TopBorderWidth	✔	✔	✔	int				Top border width in points
ALP_Cell_Value	✔	✔						Cell value (depending on the column type)
ALP_Cell_XML	✔	✔		text				Full description of the cell options in XML This will return an empty value if no options have been set for the specified cell

Cell Style Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Cell Style Properties								
ALP_Cell_BackColor	✔	✔	✔	color				Background color
ALP_Cell_BaseLineShift	✔	✔	✔	real		-100	256	Baseline shift
ALP_Cell_ClearStyle		✔		n/a				Clear the style of this cell The area redraws automatically

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
					Cell Style Properties			
ALP_Cell_Flags	✔	✔	✔	int				Bit-mask of set features Properties not set are inherited from the row settings, then the column settings The following flags indicate what style options have been set at the cell level: 2 = font name 4 = font size 8 = font style 16 = text color 32 = background color 64 = horizontal alignment 128 = vertical alignment 256 = wrap 512 = rotation 1024 = baseline shift 2048 = horizontal scale 4096 = line spacing 8192 - ellipsis Maintained by AreaList Pro and should not normally be changed Properties not set are inherited from row and then from column You can clear the flag if you want to force AreaList Pro to abandon cell-specific settings
ALP_Cell_FontName	✔	✔	✔	text				Font name
ALP_Cell_HorAlign	✔	✔	✔	int		0	5	Horizontal alignment: 0 = default 1 = left 2 = center 3 = right 4 = justify 5 = full justify
ALP_Cell_HorizontalScale	✔	✔	✔	real		0,1	100	Horizontal scale
ALP_Cell_LineSpacing	✔	✔	✔	real	1.0	1	10	Line spacing
ALP_Cell_Rotation	✔	✔	✔	real		-360	360	Rotation of text
ALP_Cell_Size	✔	✔	✔	real		4	128	Font size
ALP_Cell_StyleB	✔	✔	✔	bool				Font style = bold
ALP_Cell_StyleF	✔	✔	✔	int		0	7	Font style, using 4D style constants (e.g. Bold, Italic, etc.)
ALP_Cell_StyleI	✔	✔	✔	bool				Font style = italic
ALP_Cell_StyleU	✔	✔	✔	bool				Font style = underlined
ALP_Cell_TextColor	✔	✔	✔	color				Font color
ALP_Cell_UseEllipsis	✔	✔	✔	int	0	0	2	Use ellipsis for data of this cell -1 - use area's default (ALP_Area_UseEllipsis) 0 - none 1 - on right side for left-aligned, in center otherwise 2 - on right side for left-aligned, on left side for right-aligned, in center otherwise
ALP_Cell_VertAlign	✔	✔	✔	int		0	3	Vertical alignment: 0 = default 1 = top 2 = center 3 = bottom

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Cell Style Properties								
ALP_Cell_Wrap	✔	✔	✔	bool				Wrap long lines

Breaks

Use these properties with commands in the Break Processing command theme:

AL_GetBreakLongProperty

AL_GetBreakPtrProperty

AL_GetBreakRealProperty

AL_GetBreakTextProperty

AL_SetBreakLongProperty

AL_SetBreakPtrProperty

AL_SetBreakRealProperty

AL_SetBreakTextProperty

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Break Properties								
ALP_Break_Clear		✔		int				Clear (remove) the break or cell definition
ALP_Break_ClearBreaks		✔		n/a				Clear (remove) all break definitions
ALP_Break_IsHeader	✔		✔	bool				
ALP_Break_Level	✔		✔	int				
ALP_Break_LineColor	✔	✔	✔	color	black			
ALP_Break_LineShowColumnDivider	✔	✔	✔	bool	no			
ALP_Break_LineThickness	✔	✔	✔	real	0	0	2	Range is 0.0 - 1.0 or 2.0
ALP_Break_MinRowHeight	✔	✔	✔	real	0	0	256	Identical use as ALP_Area_MinRowHeight, but applied to this break only
ALP_Break_NumRowLines	✔	✔	✔	int	1	1	64	Identical use as ALP_Area_NumRowLines, but applied to this break only See PL_SetBrkHeight/PL_SetBkHHeight
ALP_Break_RowID	✔			int				The row number for this break - needed when you want to set row or cell options To be used with break row/cell properties: break footer 0 uses row -4, break header 0 uses -5, break footer 1 uses -6, …
ALP_Break_RowIndent	✔	✔	✔	point	3;1			Identical use as ALP_Area_RowIndent, but applied to this break only Pair of reals separated with semicolon (horizontal;vertical) See PL_SetBrkHeight/PL_SetBkHHeight

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
					Break Properties			
ALP_Break_RowIndentH	✔	✔	✔	real	3	0	64	See ALP_BreakRowIndent (above)
ALP_Break_RowIndentV	✔	✔	✔	real	1	0	64	
ALP_Break_RowsInGrid	✔	✔	✔	int	1	0	20	
ALP_Break_SelRow	✔	✔		int				Selected break row Value is the data row number for the specified break
ALP_BreakCell_ColSpan	✔	✔	✔	int	1			ColSpan of the break cell grid
ALP_BreakCell_MergeCenter		✔		int				Number of adjacent columns to use when overflowing to left & right of the cell
ALP_BreakCell_MergeDefault		✔		int				Default is get from the cell's column, then one of the 3 specialized selectors is used
ALP_BreakCell_MergeLeft		✔		int				Number of adjacent cells to use when overflowing to the left of the cell
ALP_BreakCell_MergeRight		✔		int				Number of adjacent cells to use when overflowing to the right of the cell
ALP_BreakCell_RowSpan	✔	✔	✔	int	1			RowSpan of the break cell grid
ALP_BreakCell_Source	✔	✔	✔	int				Data source (column number) for the cell (default is inherited from the data grid)
ALP_BreakCell_Value	✔	✔	✔	text				Break text value. Can include PrintList Pro break functions like “\\BreakValue”
PLP_BreakPage	✔	✔	✔	bool	no			

AreaList Pro Object Properties

Use these properties with commands in the Objects command theme:

AL_GetObjects

AL_GetObjects2

AL_SetObjects

AL_SetObjects2

Persistency depends on the underlying property, e.g.:

- Setting ALP_Object_HeaderText is the same as setting ALP_Column_HeaderText for all columns using individual values (array elements): it is persistent.
- ALP_Object_RowHide sets the underlying ALP_Row_Hide property: it is **not** persistent.

Constant	Get	Set	Array Type	Comments
				Object Properties
ALP_Object_AltHdrGrid	✔	✔	int / 2D int	Column numbers in alternate header Use 2D array to access colSpan & rowSpan, too
ALP_Object_AutoHierarchy	✔	✔	int	See ALP_Area_AutoHierarchy
ALP_Object_Columns	✔	✔	pointer	Pointers to data source All columns: add columns to the area or get a list of the area's columns
ALP_Object_ColumnWidth	✔	✔	real	Current width of all columns Must use **ARRAY REAL**
ALP_Object_ColumnWidthUser	✔	✔	real	Width of all columns as defined by the programmer or the user resizes Set to zero when user double-clicks in the resize area (column divider ± ALP_Area_ColumnResizeDistance). See ALP_Column_Width and ALP_Column_WidthUser
ALP_Object_DragDstCellCodes	✔	✔	text	Drag destination cell codes Can be used with DropArea
ALP_Object_DragDstColCodes	✔	✔	text	Drag destination column codes
ALP_Object_DragDstRowCodes	✔	✔	text	Drag destination row codes
ALP_Object_DragSrcCellCodes	✔	✔	text	Drag source cell codes
ALP_Object_DragSrcColCodes	✔	✔	text	Drag source column codes
ALP_Object_DragSrcRowCodes	✔	✔	text	Drag source row codes
ALP_Object_ExportList	✔	✔	int	Set the list of columns to export with ALP_Area_ExportToFile When setting ALP_Area_ExportOptions, include the AL Export Using List option to use the list of columns The array contains the column numbers to export The list can also be set with ALP_Area_ExportList
ALP_Object_Fields	✔		int	Table/field numbers of all columns 2D or two arrays
ALP_Object_FooterText	✔	✔	text	Footer text of all columns
ALP_Object_FooterTextNH	✔		text	Footer text of visible columns in grid order
ALP_Object_FtrGrid	✔	✔	int / 2D int	Column numbers in footer Use 2D array to access colSpan & rowSpan, too

Constant	Get	Set	Array Type	Comments
			Object Properties	
ALP_Object_Grid	✔	✔	int / 2D int	Column numbers Use a 2D array to access colSpan & rowSpan, too
ALP_Object_HeaderText	✔	✔	text	Header text of all columns
ALP_Object_HeaderTextNH	✔		text	Header text of visible columns
ALP_Object_Hierarchy	✔	✔	int	Hierarchy: level, expanded 2D or two arrays: you can call ***AL_SetObjects*** with a 2-dimensional array or ***AL_SetObjects2*** with two arrays
ALP_Object_RowHide	✔	✔	bool / int	Set to True or non-zero to hide a row
ALP_Object_RowSelection	✔	✔	int	When displaying records in row selection mode, get record numbers of selected rows / set selection using record numbers Note: can be time-consuming for large selections, especially on client/server
ALP_Object_Selection	✔	✔	int / 2D int	Selection 2D or two arrays: you can call ***AL_SetObjects*** / ***AL_GetObjects*** with a 2-dimensional array or ***AL_SetObjects2*** / ***AL_GetObjects2*** with two arrays if the selection mode is not row selection ([multiple] cell selection mode)
ALP_Object_Sort	✔		int	Sort order = to be used with ALP_Area_DontSortArrays
ALP_Object_SortList	✔	✔	int	Sort list – use negative number for descending order
ALP_Object_SortListNS	✔	✔	int	Sort list – use negative number for descending order Set only the sort list, do not actually sort the data
ALP_Object_Source	✔		text	Data source of all columns
ALP_Object_Type	✔		int	Data type of all columns: 4D type constants can be used Note: Is Time is returned for longint arrays formatted as time
ALP_Object_Visible	✔	✔	bool / int	Visible status of columns Set to True or non-zero to show the column For ***AL_SetObjects***: if the array is shorter than the number of columns, the remaining columns visible status will not be modified In compatibility mode visibility of the columns is always reset according to ALP_Area_CompHideCols

Printing

Note: many of these Printing properties also apply to Display in an AreaList Pro area, and conversely.

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
					Properties			
ALP_Area_PrintArea	✔	✔		n/a				Print the area using current **PRINT SETTINGS** and ALP_Area_PrintOptions
ALP_Area_PrintOptions	✔	✔	✔	Object				For a full description of the supported properties, see below,
PLP_BreakDivColor	✔	✔	✔	color				Color of break divider line
PLP_BreakDivWidth	✔	✔	✔	real				Width of break divider line, in points
PLP_ColDivThickness	✔	✔	✔	real	1	0	5	Column divider weight, in points
PLP_FrameColor	✔	✔	✔	color				Color to use for the frame during printing
PLP_FrameWidth	✔	✔	✔	real	1.0	0	1	Frame width in fractions of a point Not modified when printing To be used, ALP_Area_DrawFrame must be 1
PLP_HeaderDivColor	✔	✔	✔	color				Color for the header dividing line
PLP_HeaderDivThickness	✔	✔	✔	real				Weight of the header divider in points
PLP_MinimumAreaHeight	✔	✔	✔	real	0	0		Minimum height of the area when pritning, in points
PLP_PageEndCallback	✔	✔	✔	text				Callback method to use on end of page during printing.
PLP_PrintHeaders	✔	✔	✔	int	0	0	2	0 - never 1 - on first page only 2 - on all pages
PLP_PrintLastBreak	✔	✔	✔	bool	no			Use with PL_SetBrkOpts
PLP_RowDivThickness	✔	✔	✔	real	1	0	1	Row divider weight, in points
PLP_ShrinkArea	✔	✔	✔	int	0	0	2	0 - enlarge last visible column to fit the area (old PrintList Pro behavior) 1 - shrink the area, don't enlarge the last visible column when needed width is smaller than provided 2 - same as 1, but centered horizontally in provided space

ALP_Area_PrintOptions object description

Properties recognized by AreaList Pro when doing standalone printing with ALP_Area_PrintArea - ALP_Area_PrintOptions:

Property	Type	Default	Description
drawFrame	int (0-2)		When present and in range, ALP_Area_DrawFrame is set to this value, otherwise ALP_Area_DrawFrame is set to 1 when it is 2
frameWidth	real (0.0-1.0)		When present and in range, PLP_FrameWidth is set to this value (nothing is done otherwise)

Property	Type	Default	Description
ftrLeft ftrCenter ftrRight	text		[styled] text to use for left, center and right footer Variable substitution is available for variables “<%Date%>”, “<%Time%>”, “<%Page%>” and “<%Pages%>”; only the first occurrence of every variable in every text is replaced
ftrHeight	real	Automatically computed height	Height of the footer
ftrIndent	real	+2 if automatic height	Footer indent (in points). Indent is between footer/body; margins define position of the header from top/footer from bottom
hdrLeft hdrCenter hdrRight	text		[styled] text to use for left, center and right header Variable substitution is available for variables “<%Date%>”, “<%Time%>”, “<%Page%>” and “<%Pages%>”; only first occurrence of every variable in every text is replaced
hdrHeight	real	Automatically computed height	Height of the header
hdrIndent	real	+2 if automatic height	Header indent (in points). Indent is between header/body; margins define position of the header from top/footer from bottom
jobName	text	Default is from ALP_Area_Name	Name of the job to use for print queue
landscape	boolean	Use what is set in the print settings (yes, current PRINT SETTINGS are used for the printing)	When true, use landscape; portrait otherwise
margins	text	Default is from the printer driver (e.g. 12 pt on all sides for a laser printer, but usually zero with Microsoft Print To PDF on Windows)	“left,top,right,bottom” (or separated with “;”) always using decimal dot, always using portrait (don’t rotate the rectangle for landscape)
minMargins	text	Minimum margins to use; when printer margins are bigger, nothing happens (the driver defined margins will be used), when they are smaller, this minimum is used	“left,top,right,bottom” (or separated with “;”) always using decimal dot, always using portrait (don’t rotate the rectangle for landscape)
noSortIndicator	boolean		Turns ALP_Area_ShowSortIndicator off - no space in header for sort order triangle
printHeaders	int (0-2)		When present and in range, PLP_PrintHeaders is set to this value, otherwise PLP_PrintHeaders is set to 0 for ALP_Area_HideHeaders=1 and 2 for ALP_Area_HideHeaders=0
progress	Formula (object)		The progress callback is done before every page is processed/printed and at the end The Formula receives no parameters, only **This** is defined; something like: $abort:=$ALPArea.ALP_Area_PrintOptions.progress.call($PrintProgressObject) The object attached as **This** lives during the printing; you can store anything you need there, like WindowRef Non-zero return value means “abort printing”, it does not matter if you return 1, 3.14 or TRUE
scale	real		Page scaling (implemented in AreaList Pro, no need for scaling support from the printer, e.g. on Windows)

Properties defined by AreaList Pro

Property	Type	Default	Description
counting	boolean		First phase of counting page is in progress, “numPages” is unknown
currentPage	int		Current page being processed
firstPage	int		First page to print as specified in the print job.
lastPage	int		Last page to print as specified in the print job (it is INT_MAX when “print all pages” was selected).
numPages	int		Number of pages (when AreaList Pro was forced to 2-phase run - to count number of pages; zero otherwise)
printedAreaRef	int		The area ID of the area being printed.
printing	boolean		Printing in progress
screenAreaRef	int		The area ID where you asked for printing.
status	int		Available in the last call after finishing the printing (“counting” is False, “printing” is False), the error code of the printing process; aborted is -128

AreaList Pro Area DropArea Properties

For more information about using Drag and Drop, see the Drag and Drop section.

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area DropArea Properties								
ALP_Drop_DragAcceptColumn	✔	✔	✔	bool	yes			Accept column drag when source does not have ALP_Area_DragSrcColCodes set OBSOLETE: use Drag codes instead
ALP_Drop_DragAcceptLine	✔	✔	✔	bool	yes			Accept line drag when source does not have ALP_Area_DragSrcRowCodes set OBSOLETE: use Drag codes instead
ALP_Drop_DragDstCodes	✔	✔	✔	text				Drag destination codes The format is a list of codes separated by ‘|’
ALP_Drop_DragProcessID	✔			int				4D’s process ID of this area
ALP_Drop_DragSrcArea	✔			int				The dropped AreaList Pro area
ALP_Drop_Kind	✔		✔	text				Object kind = “DropArea”
ALP_Drop_Name	✔			text	area name from design			
ALP_Drop_XML	✔	✔		text				Full description of the drop area in XML

16 DisplayList

About DisplayList

DisplayList is a kind of mini-AreaList Pro, without any need to use a layout and a plug-in area.

It's an easy-to-use tool to implement scrolling lists for user interaction.

It is now fully included with AreaList Pro and backwards compatible (except the items listed below). We have included a command list for reference purposes.

See also Entering data in AreaList Pro with DisplayList.

Incompatibilities

Patterns are no longer supported. They are interpreted by AreaList Pro version 11 as transparency ratios.

The font styles Outline, Shadow, Extended, and Condensed are no longer supported.

ArraySort: This was a substitute for **MULTI SORT ARRAY**, before this call was introduced to 4D. It must be replaced with **MULTI SORT ARRAY**.

DisplayList Commands

DisplayList

(array1;…arrayN) ➞SelectedLine

Parameter	Type	Description
➞ array1…N	array	array
↔ selectedLine	Integer	Line selected by user: -1: another DIsplayList window is currently displayed in another process 0: user clicked cancel button, or no elements were selected >0: line number selected by user

DisplayList displays the arrays in the DisplayList window.

Any formatting commands must be executed prior to DisplayList.

SetListHeaders

(header1;…headerN)

Parameter	Type	Description
➞ header1…N	string	Header text to display in each column

SetListHeaders is used to specify the value to display in the header for each column.

It is executed prior to DisplayList, and the parameters for the two commands correspond (i.e., the first parameter for ***SetListHeaders*** sets the header for the first parameter, or first column, of DisplayList).

SetListButtons

(OKtext;cancelText;promptText;button3Text;button3Cmd;button4Text;button4Cmd;button5Text;button5Cmd;button6Text;button6Cmd)

Parameter	Type	Description
OKText	string	value to display in OK button
cancelText	string	value to display in Cancel button
promptText	string	value to display in upper left of the DisplayList window
button3Text	string	value to display in the third button
button3Cmd	string	cmd key equivalent for the third button
button4Text	string	value to display in the fourth button
button4Cmd	string	cmd key equivalent for the fourth button
button5Text	string	value to display in the fifth button
button5Cmd	string	cmd key equivalent for the fifth button
button6Text	string	value to display in the sixth button
button6Cmd	string	cmd key equivalent for the sixth button

SetListButtons is used to specify the values to display in the buttons, and the Prompt message in the upper left of the DisplayList window. The button widths will automatically be adjusted so that the values will fit. If the values can't be displayed in the size window, then the ends of the button values will be truncated.

SetListSize

(windowHeight;windowWidth;location)

Parameter	Type	Description
➞ windowHeight	Integer	Height of DisplayList window
➞ windowWidth	Integer	Width of DisplayList window
➞ location	Integer	Index value for window location

SetListSize is used to control the size or location of the window.

The possible values for Location are:

Value	Window location
0	Centered on screen (default)
1	Centered in top 4D window
2	At top left of screen
3	At mouse position

DisplayList will not allow a window to be displayed larger than the screen being used. If the parameters passed will result in a window larger than the screen, then the window will be displayed at the maximum possible size to fit on the screen.

If ***SetListSize*** is not called, the default size is based on arrays content/columns width and buttons and can grow to full (main) screen.

Pass a zero (0) for either window size parameter to force auto-sizing for that dimension.

Value 3 (at mouse position) may be useful when DisplayList is used for "popup" entry in AreaList Pro.

SetListWidths

(column1;…columnN)

Parameter	Type	Description
➞ column1…N	Integer	Width for each displayed column

SetListWidths is used to set the point width for one or more columns.

Each passed parameter corresponds to the array passed in the ***DisplayList*** command.

SetListFormats

(columnNumber; format)

Parameter	Type	Description
➞ columnNumber	Integer	Column to format
➞ format	string	Format to apply to data displayed in ColumnNumber

The display format for the contents of a column is set with ***SetListFormats***.

Any 4D-supported format for number, boolean, date, time (**ARRAY LONGINT**) and string arrays may be used.

Developer-created styles defined in the Design Environment may also be used.

Be sure to use a string as the second parameter!

SetListHdrStyle

(fontName;size;styleNumber)

Parameter	Type	Description
➞ fontName	string	Name of font. If not called, or the specified FontName is not found, the headers will be displayed in Geneva 12 point Plain
➞ size	integer	Size of font
➞ styleNumber	integer	Number for style to apply to font. A Macintosh font style code. By adding the codes together, you can combine styles

SetListHdrStyle is used to format the DisplayList column headers.

The numeric codes for StyleNumber are shown below:

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4

SetListStyle

(fontName;size;styleNumber)

Parameter	Type	Description
➞ fontName	string	Name of font. If not called, or the specified FontName is not found, the headers will be displayed in Geneva 12 point Plain
➞ size	integer	Size of font
➞ styleNumber	integer	Number for style to apply to font. A Macintosh font style code. By adding the codes together, you can combine styles

SetListStyle is used to format the DisplayList arrays, or list.

The numeric codes for StyleNumber are shown below:

Style	Number
Plain	0
Bold	1
Italic	2
Underline	4

SetListBehavior

(multiLines;allowColumnResize;sortColumn;preSort;userSort;displayPointWidth;hideLastColumn;swapCancelOK)

Parameter	Type	Description
➞ multiLines	integer	Single or multiple-line selection: 1 - allow user to command-click, shift-click, or drag to select multiple lines 0 - allow only one line to be selected (default)
➞ allowColumnResize	integer	User-resizable columns: 1 - allow user to resize columns (default) 0 - prevent user from resizing columns
➞ sortColumn	integer	column for presort
➞ preSort	integer	presort off, ascending or descending
➞ userSort	integer	allow user to sort
➞ displayPointWidth	integer	display column widths
➞ hideLastColumn	integer	don't display last array passed to DisplayList
➞ swapCancelOK	integer	OK and Cancel buttons are swapped when they are visible

SetListBehavior is used to control several DisplayList options.

Each parameter is an integer.

SetListColor

(foreColor1;foreColor2;backColor1;backColor2)

Parameter	Type	Description
→ foreColor1	string	Foreground color from DisplayList's palette
→ foreColor2	integer	Foreground color from 4D's palette
→ backColor1	string	Background color from DisplayList's palette
→ backColor2	integer	Background color from 4D's palette

SetListColor is used to set the foreground and background colors of the list area.

DisplayList has its own palette.

It contains the following colors:

White	Green
Black	Blue
Magenta	Yellow
Red	Gray
Cyan	Light Gray

SetListHdrColor

(foreColor1;foreColor2;backColor1;backColor2)

Parameter	Type	Description
→ foreColor1	string	Foreground color from DisplayList's palette
→ foreColor2	integer	Foreground color from 4D's palette
→ backColor1	string	Background color from DisplayList's palette
→ backColor2	integer	Background color from 4D's palette

SetListHdrColor is used to set the colors for the header area.

SetListDividers

(colDividerPattern;colDividerColor1;colDividerColor2;rowDividerPattern;rowDividerColor1;rowDividerColor2)

Parameter	Type	Description
→ colDividerPattern	string	pattern of the column divider
→ colDividerColor1	string	color from DisplayList's palette for the column divider
→ colDividerColor2	integer	color from 4D's palette for the column divider
→ rowDividerPattern	string	pattern of the row divider
→ rowDividerColor1	string	color from DisplayList's palette for the row divider
→ rowDividerColor2	integer	color from 4D's palette for the row divider

SetListDividers is used to set the pattern and the color of the column and row dividers.

Patterns are no longer supported. They are interpreted by AreaList Pro as transparency ratios since version 9.

SetListLine

(line number)

Parameter	Type	Description
→ line number	integer	line number to select (highlight)

SetListLine sets the line to be highlighted.

The list will automatically scroll to display the selected line at the top of the list, if possible.

SetListSelect

(array)

Parameter	Type	Description
→ array	integer array	contains element numbers to select (highlight)

SetListSelect sets the lines to be highlighted.

The list will be automatically scrolled to display the first selected line at the top of the list, if possible.

If this command is not used, then DisplayList will display the arrays with the first line selected. ***SetListSelect*** can only be used in multiline mode. If DisplayList is in single-line mode, you must use ***SetListLine***.

GetListButton

→ buttonHit

Parameter	Type	Description
→ buttonHit	integer	integer button that the user selected

GetListButton is used to get the button that the user selected.

ButtonHit — The possible values are:

Value	Button Selected
1	OKButton
2	CancelButton
3	Button3
4	Button4
5	Button5
6	Button6

GetListWidths

(column1;…columnN)

Parameter	Type	Description
→ column1…N	integer	width for each column

GetListWidths is used to get the widths of the columns after ***DisplayList*** has been displayed, to allow any user changes to the column widths to be saved for future use.

Variables must be used as the passed parameters; this function will not work with fields. ***GetListWidths*** must be executed after ***DisplayList.***

GetListSelect

(array) → result code

Parameter	Type	Description
→ array	array	list of selected lines
↔ result code	longint	indicates if enough memory was available

GetListSelect is used to determine which items were selected by the user when the Multi-line option is enabled using ***SetListBehavior***, and they have selected multiple lines.

Each element of the array contains a line number that the user selected when the list was displayed. The array must be an integer or longint array.

For compatibility purposes, the result is 1 if everything was OK.

SetListDone

SetListDone is used to inform DisplayList that you are done using it in the current 4D process.

Use ***SetListDone*** when you are done with all of the DisplayList commands in a 4D process, to free up the memory used by DisplayList for that process.

Normally you will call this routine at the end of a process.

Troubleshooting

This section lists several common problems, and their solutions, encountered when working with DisplayList.

When troubleshooting a problem, use all of the tools at your disposal, including the 4D debug window.

Many problems can be quickly resolved by stepping through each line of code, and checking the values of variables and arrays.

Why are one or more of my columns missing?

Ensure that all arrays are of the same size.

DisplayList will use the largest array's size, and not display any arrays of non-conforming size.

Also remember that two-dimensional arrays should not be used.

Why doesn't the command key equivalent work for a button?

Make sure you passed a button text for the button. Also make sure none of the preceding buttons have the same command key.

17 PrintList Pro

There are three ways to print your AreaList Pro data: using the (now built-in) PrintList Pro features, using the SuperReport Pro plugin, and directly print the AreaList Pro area with its formatting ("Standalone Printing").

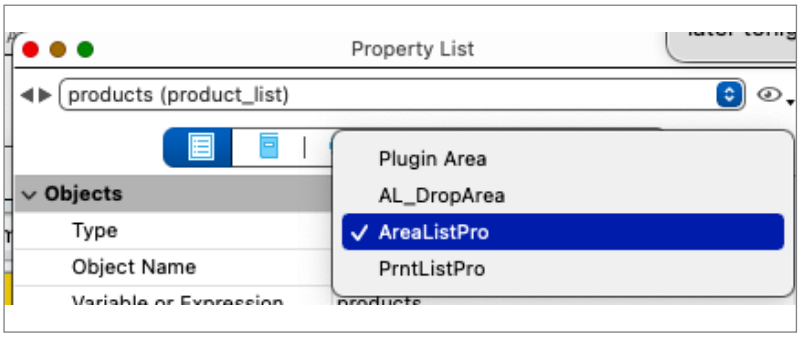
Note: whatever the solution used to print, on Windows you must set the engine to GDI with **SET PRINT OPTION** (Legacy printing layer option;1). Only one call is needed (e.g. **On startup**).

A Historical Note ...

Prior to AreaList Pro Version 11, you could use the features available in our PrintList Pro plugin to produce complex and beautiful printed reports. Now these options are included in AreaList Pro, so you don't need the additional plugin. .

You don't need to do anything special to upgrade from previous versions; simply install AreaList Pro Version 11 and you're all set.

Note that your plugin areas can be selected as either AreaList Pro or PrintList Pro:



They are functionally the same.

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

PrintList Pro is an easy-to-use tool for printing AreaList Pro areas, whether they are based on arrays or fields. I

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.

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).

Getting started with PrintList Pro

Creating a PrintList Pro Area

A PrintList Pro area is actually just another AreaList Pro area. Please see the Getting Started section.

Working with PrintList Pro Commands

The sytax of PrintList Pro commands is the same as the standard AreaList Pro commands (see Command Descriptions and Syntax).

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, PrintList Pro 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 critera, 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	Plain
1	Bold
2	Italic
4	Underline

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 discontiguous 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 AreatList Pro Text Style Tags

Colors

Many PrintList Pro objects can be given color settings. Be sure to set the printer Color/Grayscale option to obtain the proper results. Please see the Working with Colors chapter for detailed information about specifying and using colors.

PrintList Pro's color 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 discontiguous 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 points.

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 point).

Typically, ¼ (.25) point 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_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	Point width(s) of column(s).

PL_SetWidths is used to set the point 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 points 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; printPointWidth:L)

Parameter	Type	Description
➞ areaRef	longint	Reference of PrintList Pro object on layout.
➞ printHeaders	longint	Print the headers above the list.
➞ printPointWidth	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

printPointWidth — Used during development to allow you to easily determine what point 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 point widths

PL_SetHdrOpts(eList;1;1) // print headers on first page, and point 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 point 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 point 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 point 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 point 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 point 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 point 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 point 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 point 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 point 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; numRowLines: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.
➞ numRowLines	longint	Number of text lines in each row.
➞ rowHeightPad	longint	Extra height for each row.
➞ minimumHeight	longint	Minimum point 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 points, 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 points, to give to each row. Default is 0.

minimumHeight — The minimum remaining available height, in points, 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 points (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 points and the rows by 2

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

PL_SetHeight (eList;1;4;1;2;12) // check that 12 points (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.

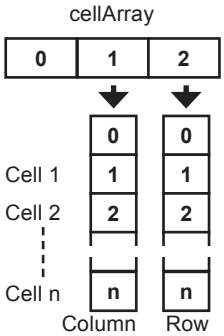
PL_SetCellStyle is used to set the font and/or style of a specific cell, range of cells, or list of cells.

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	Point 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

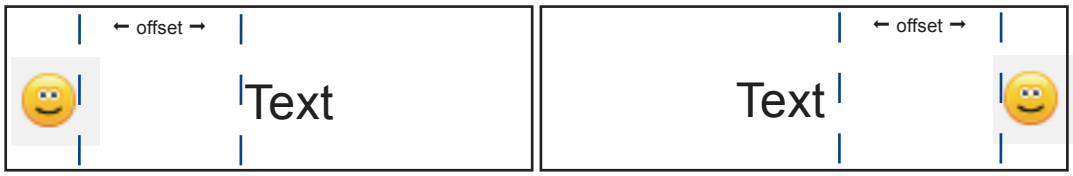
horPosition — One the following options:

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:

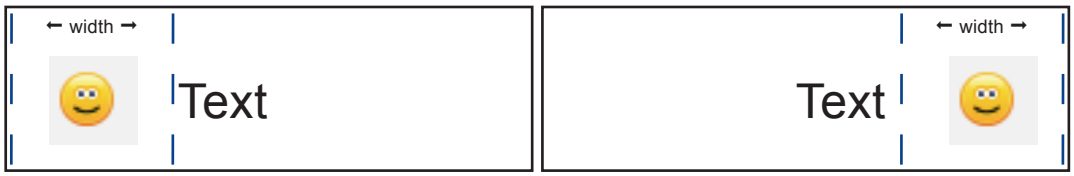
Value	Mode
0	Default (top)
1	Align top left
2	Align center
3	Align bottom

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 points between the text and the icon (left or right):



horPosition = 0

Otherwise the **offsetOrWidth** is the point 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 points and you print a 15 point 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 points.
→ 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 points. 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 points.
→ 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 points. 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 — It 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.

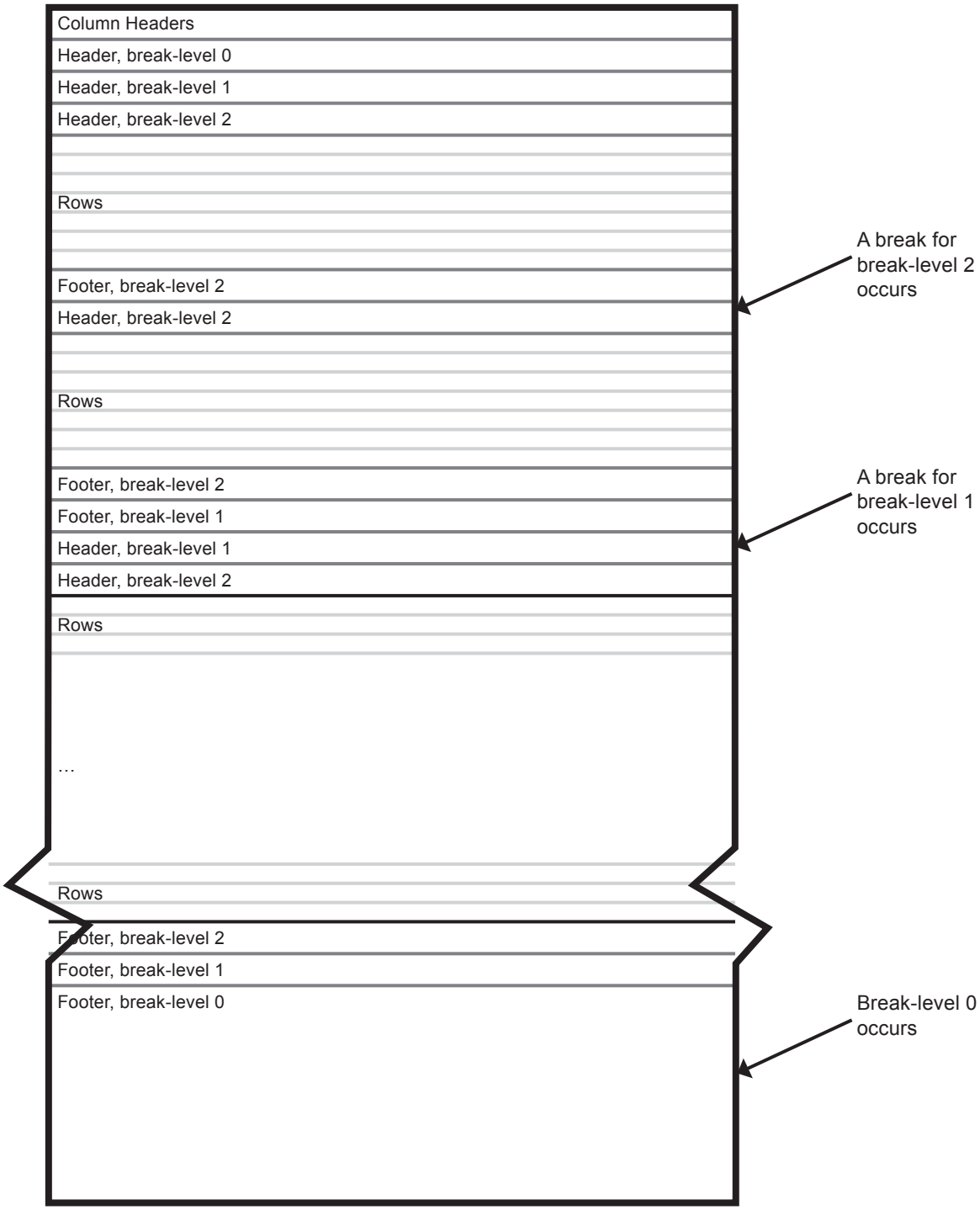
If the same membership list is sorted by country first and then by city, a break for break level one (country) would occur whenever the country changes in the list, and a break for break level two (city) would occur whenever the city changes in the list.

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.

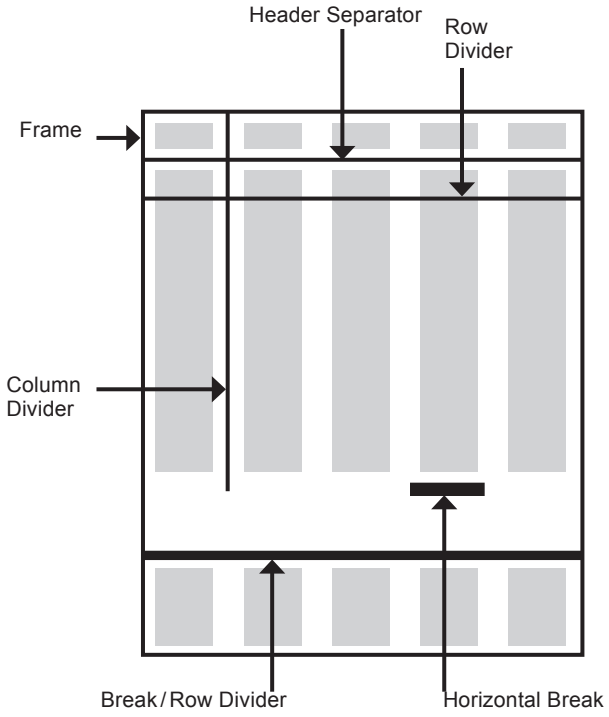
To set the number of lines to be printed in a break level, use PL_SetBrkHeight for break footers and PL_SetBkHHeight for break headers. Please read the section Variable Height Breaks for more information.

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 occurence 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 points, 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 points 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 point 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 points 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 points 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_SetBkHColRGBOpt

(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_SetBkHColRGBOpt 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_SetBkHColOpt, 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 ***PL_ProcessArrays*** call.
➞ column	array	Index of the column to be processed in the dataArrays array set by ***PL_ProcessArrays*** 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.

PrintList Pro 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

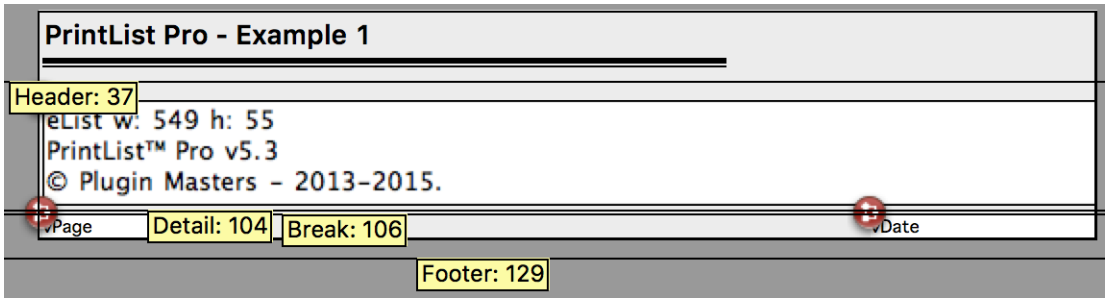
Print a list of information (First name, Last name, Salary, City, State, Zip, Country) contained in a series of seven arrays. Show column headers and print dividing lines between the columns. Do not print dividing lines between the rows.

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	Schwarts	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

Page 1 Mon, Jan 21, 2008

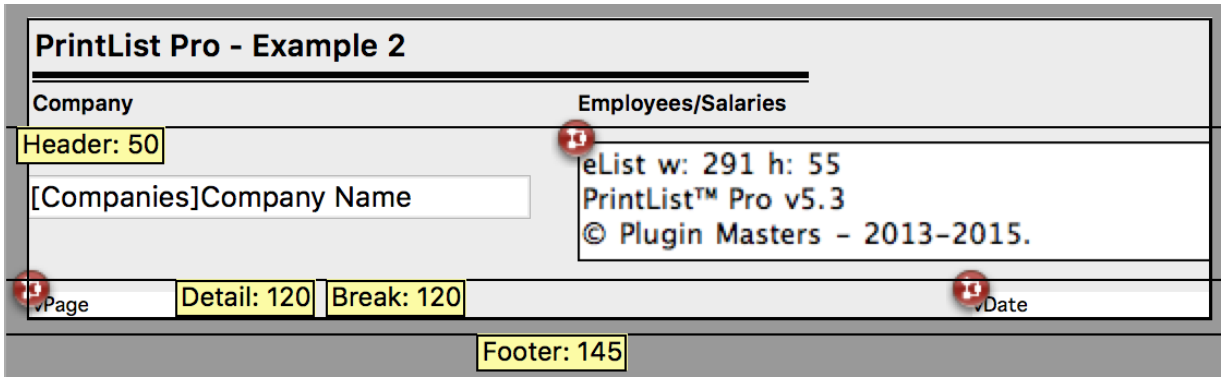
Example 2 — Multiple record current selection

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:



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

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 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

Page 1 Mon, Feb 11, 2008

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.

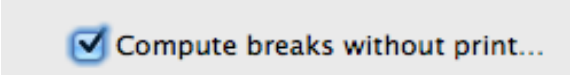
First Name	Last Name	Salary	City	State	Country
Bill	Goodhew	$23 275,98	Podunk	AR	USA
Stan	Getz	$60 956,00	Podunk	AR	USA
Yogen	Dalal	$41 491,24	Podunk	AR	USA
Bill	Coldrick	$42 369,32	Podunk	AR	USA
	Sum	$168 092,54			
	Minimum	$23 275,98		**4 people in Podunk**	
	Average	$42 023,13			
	Maximum	$60 956,00			
	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		**3 people in Phoenix**	
	Average	$34 809,60			
	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		**6 people in Cupertino**	
	Average	$49 310,33			
	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	Schwarts	$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		**9 people in Los Angeles**	
	Average	$46 594,42			
	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

Page 1 2 sept. 2013

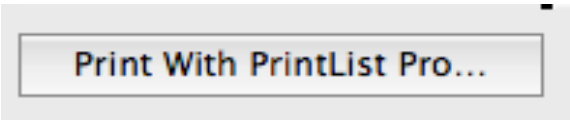
Example 5 — Computed Breaks

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($i)
// Create the arrays from the data
ALL RECORDS([People])
SELECTION TO ARRAY([People]First Name;aFname;[People]Last Name;aLname;[People]Salary;\
    aSalary;[People]City;aCity;[People]State;aState;[People]Country;aCountry)
// Sort by Country (descending), State, City, and Last Name (descending)
MULTI SORT ARRAY(aCountry;<;aState;>;aCity;>;aLname;<;aFname;aSalary)
// Declare the "data" array pointing on the actual arrays on which to perform calculations in our callback
// (we include all arrays in case we'll need them someday, even though we will only use aSalary here)
ARRAY POINTER($dataArrays;0)
APPEND TO ARRAY($dataArrays;->aCountry)
APPEND TO ARRAY($dataArrays;->aState)
APPEND TO ARRAY($dataArrays;->aCity)
APPEND TO ARRAY($dataArrays;->aLname)
APPEND TO ARRAY($dataArrays;->aFname)
APPEND TO ARRAY($dataArrays;->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;$dataArrays;0)=0)
   // 4th parameter is 0 for each break or 1 for each row
   // Now TbreakText_R has been populated by PlpComputeBreak: concatenate breaks into a text variable
   vBrkComputeResult:=""
   For ($i;1;Size of array(TbreakText_R))
      vBrkComputeResult:=vBrkComputeResult+TbreakText_R{$i}+Char(Carriage return)
   End for
   SET TEXT TO PASTEBOARD(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 $dataArrays
 // from the calling method (third parameter to PL_ProcessArrays)
C_TEXT($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.)
```

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	<b>
Italic	<i>
Underline	<u> or <ins>
Strike-through	<del>
Set font size to # points	<s #>
Increase font size by # quarters (1/4) of current size	<s +#>
Decrease font size by # quarters (1/4) of current size	<s -#>
Set font by name	<f "font name"> (needs to be quoted if the name contains more than one word)
Set color (any format can be used, e.g. <c 0xFFFF0000> <c 1.0,0,0> <c P123> <c dark orange>)	<c color name>

4D internal format for styled text is stored as <SPAN STYLE="style attributes"> 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
```

With the above settings:

- Positive numbers will be printed in blue roman characters with a plus sign.
- Negative numbers will be printed in red italic characters with a minus sign.
- Zeros will be printed in green bold, font size increased by 25%, with the text "ZERO".
- Here is the result:

-16 238
-5 526
-31 880
-21 940
+4 137
-100
ZERO
+27 512
-9 330
+21 250
+707
+28 936
-30 953
-24 692
+24 109
-24 352

Note: if the number format is too “small” to hold the number, 4D (and PrintList Pro) will display it as “<<<<<<<<<<<<<<<<<<”, which will interfere with the opening tag character “<” if the column is attributed (multi-styled).

In the example above (using "## ###" as a number format), this will be the case for all numbers exceeding 99,999.

Make sure that the format used will not cause the number to overflow, lest unexpected results might ensue.

18 Other Printing Options

Printing with SuperReport Pro

SuperReport Pro is 4D's powerful printing companion. It can also be used in conjunction with AreaList Pro version 9.4 and above to print AreaList Pro areas. This feature requires SuperReport Pro version 3 or above.

How it works

AreaList Pro allows printing or saving as HTML through SuperReport Pro. It only takes two lines of code to print an AreaList Pro area.

Additional options are available, such as automatic column width and use of SuperReport Pro style properties instead of the existing AreaList Pro area settings.

AreaList Pro v9.7 and above can print the area footers using SuperReport Pro 3.1.2 or higher.

You can either use the built-in SuperReport Pro template to print an AreaList Pro area "on the fly" or create your own.

The AreaList Pro Demonstration database includes a "Print with SuperReport Pro" button in the AreaList > Configuration Options…" dialog. It also includes a SuperReport menu, allowing printing with the default template or a custom template, and editing / creating your own templates.

Command and properties

Use the following command to print with SuperReport Pro:

AL_SuperReport

(areaRef:L; template:T; options:L; styleOptions:L; title:T) ➞ result:T

Parameter	Type	Description
➞ areaRef	longint	Reference of AreaList Pro object on layout.
➞ template	text	XML SuperReport template or full path to a XML template or empty to use AreaList Pro's built-in template.
➞ options	longint	0 = use current columns widths; 1 = use automatic width.
➞ styleOptions	longint	Style properties that should **not** be overtaken by AreaList Pro - see constants in SuperReport Pro manual, Style Features.
➞ title	text	Optional text centered in the header.
← result	text	SuperReport Pro report XML

Use the following property with ***AL_GetAreaTextProperty*** to retrieve the default template in XML format:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Area_SRPTableTemplate	✔			text				Get the SuperReport Pro template used for report creation (stored in Resources/Table Report.xml) as XML

Creating the report

Creating a XML SuperReport Pro report from an AreaList Pro area is performed by the ***AL_SuperReport*** command:

AL_SuperReport (AreaRef:L; Template:T; Options:L; StyleOptions:L; Title:T) -> result:T

- **Template** can be a XML template or full path to a XML template or empty to use AreaList Pro's built-in SuperReport Pro template
- **Options**:

  0: use the current column width (SuperReport Pro Table column's width set to ALP_Object_ColumnWidth)

  Note: the current column width (ALP_Object_ColumnWidth) is used, not the width set by developer, or by the user resizing columns (ALP_Object_ColumnWidthUser).

  Note: ALP_Object_ColumnWidth can be smaller than ALP_Object_ColumnWidthUser when ALP_Area_ScrollColumns = 1 or smaller / bigger when ALP_Area_AutoResizeColumn # 0.

  1: use automatic width (SuperReport Pro Table column's width set to zero)
- **StyleOptions**: bit-field - which style properties should not be overtaken from AreaList Pro
- see constants in SuperReport Pro Style Features (SuperReport Pro v3 manual)
- **Title**: optional text centered in the header

If you want to use your own SuperReport Pro template:

- **Title** will replace any text in all text objects named “Title” in the first header (must not be grouped - only direct children of the header)
- in the body section, the first table object will be filled with headers/columns; the table must have exactly one column (used as the template for all printed columns)

AreaList Pro area's alternate row coloring settings are honored in the SuperReport Pro report.

Example

```
$reportXml:=AL_SuperReport ($area;"";1;SRP_Style_HasFontName | SRP_Style_HasFontSize;
    “My first AreaList Pro area printed using SRP”)
```

The code above means: fill the template but don’t use the font name and font size defined in AreaList Pro (use the one stored in the template default style), columns will be auto-sized by SuperReport Pro (because the fonts are different, the AreaList Pro widths must be ignored)

Then use SuperReport Pro to edit the resulting report, save it, export it as HTML or print it, e.g.:

```
$result:=SR_Print ($reportXml;0;SRP_Print_DestinationPreview | SRP_Print_AskPageSetup; "";0;"";0)
```

Custom templates

AreaList Pro’s built-in SuperReport Pro template is obtained by the ALP_Area_SRPTableTemplate property, which gets the SuperReport Pro template that will be used for report creation by ***AL_SuperReport*** (stored in Resources/TableReport.xml) as XML:

```
$tableReportTemplate:=AL_GetAreaTextProperty (0;ALP_Area_SRPTableTemplate)
 //get the built-in SuperReport Pro template template from AreaList Pro
```

Then in SuperReport Pro you can edit and save your own template anywhere (in the data file or a document) for future use with ***AL_SuperReport***, e.g.:

```
$srpError:=SR_LoadReport ($window;$tableReportTemplate;0)
//load the SuperReport Pro report and display it, save the custom template somewhere with the File menu
```

Demonstration database code examples

Print with SuperReport Pro (default template)

```
C_TEXT($reportXml)
C_LONGINT($result)
$reportXml:=AL_SuperReport (eArea;"";0;0;vTitle)
$result:=SR_Print ($reportXml;0;SRP_Print_DestinationPreview | SRP_Print_AskPageSetup;"";0;"";0)
```

Print with SuperReport Pro (custom template)

```
C_TEXT($reportXml ;$path)
C_LONGINT($result)
$path:=Select document("";".xml;xml";"Select a SuperReport Pro template for AreaList Pro";0)
$path:=Document  //we actually need the full path
If ($path#"")
    $reportXml:=AL_SuperReport (eArea;$path;0;0;vTitle)
    $result:=SR_Print ($reportXml;0;SRP_Print_DestinationPreview | SRP_Print_AskPageSetup;"";0;"";0)
End if
```

Editing a custom template

```
C_TEXT($tableReportTemplate)
C_LONGINT($srpError)
C_LONGINT($window)
$tableReportTemplate:=AL_GetAreaTextProperty (0;ALP_Area_SRPTableTemplate)
    // get the built-in SuperReport Pro template from AreaList Pro
If ($tableReportTemplate#"")
   $window:=Open external window(100;100;800;800;Plain form window;
      “SuperReport Pro template for AreaList Pro";"%SuperReport")  //open the window for editing
   $srpError:=SR_LoadReport ($window;$tableReportTemplate;0)
      // load the SuperReport Pro report and display it, save the custom template somewhere with the File menu
End if  // closing the window will prompt for save if modified
```

Standalone Printing

In addition to the whole PrintList Pro module, now fully included in AreaList Pro v11, a simple AreaList Pro property triggers printing without using PrintList Pro: ALP_Area_PrintArea. The area is printed as-is, with its formatting and other settings. This is the Standalone Printing option.

This "standalone" printing requires one line of code:

```
$Lerror:=AL_GetAreaLongProperty(area; ALP_Area_PrintArea)
```

Of course, there are plenty of options available to customise the printing job.

Use the usual AreaList Pro commands and properties to configure the area for printing.

The print job uses current **PRINT SETTINGS**.

The ALP_Area_PrintOptions property can be used for specific settings See ALP_Area_PrintOptions object description.

Note: the "landscape" attribute, if used, will override the orientation defined in **PRINT SETTINGS**.

Example 1: Setting the font for the headers

To set the font for the headers to Arial Bold:

```
$LcolumnCount_V:=AL_GetAreaLongProperty(MyArea; ALP_Area_Columns)
For ($i; 1; $LcolumnCount_V)
   AL_SetColumnTextProperty(MyArea; $i; ALP_Column_HdrFontName; "Arial")
   AL_SetColumnLongProperty(MyArea; $i; ALP_Column_HdrStyleF; 1)
End for
```

Example 2: Setting various print options

```
C_OBJECT($oProp)
C_LONGINT($err)
PRINT SETTINGS(Page setup dialog+Print dialog)
If (OK=1)
   $err:=AL_GetAreaPtrProperty($area; ALP_Area_PrintOptions; ->$oProp)
   // you can assume $err is 0 when $area is valid, because you ask for an object and provide a pointer to an object
   If ($oProp=Null)  //When ALP_Area_PrintOptions is not defined, create a new object and set the object to ALP
      $oProp:=New object
      $err:=AL_SetAreaPtrProperty($area; ALP_Area_PrintOptions; ->$oProp)
   End if
   $oProp.landscape:=True  //will override PRINT SETTINGS
   $oProp.scale:=0.7  //explicitly set the scaling to 70% (also on Windows!)
   $oProp.hdrLeft:="Left Header with date <%Date%>"
```

```
    $oProp.hdrCenter:="<span style=\"font-family:'Menlo';font-size:12pt;color:#0003FF;font-weight:bold\">ALP PRINTING</
span>"
    $oProp.hdrRight:="<span style=\"font-family:'Times New Roman';font-size:10pt\">Page &lt;%Page%&gt; of
&lt;%Pages%&gt;"
    // <%variable%> must be encoded, because styled text is used
    //<%Pages%> implies AreaList Pro will use 2 loops to count the pages first
    $oProp.hdrHeight:=30
    $oProp.ftrLeft:="<span style=\"font-family:'Lucida Grande';font-size:12pt;color:#FF0000\">printed using "\
    +"<u>ALP_Area_PrintArea</u></span>"  //<u> is a shortcut supported by ALP, we are lazy to use 4D's <SPAN text-
decoration:underline>
    $oProp.ftrRight:="<i>this is the right footer</i>"
    $oProp.ftrHeight:=20
    $oProp.jobName:="AreaList Pro Standalone Printing"
    $err:=AL_GetAreaLongProperty(eList; ALP_Area_PrintArea)
    If ($err#0)
        ALERT("Printing error: "+String($err))
    End if
End if
```

Additional properties

In addition to the settings managed through the ALP_Area_PrintOptions object, some properties can be set for printing (Standalone or PrintList Pro):

- Set PLP_PrintHeaders to 0 for no column headers (default), 1 for headers on first page only, or 2 for headers on each page.
- Set PLP_FrameWidth to a non zero value up to 1 (real property) if you want to draw a frame around the printed area.
- To print the frame, make sure you also set ALP_Area_DrawFrame to 1, and remove it after print to not apply the frame to the displayed area (see the Standalone Print Example in the v11 Features menu of the Tutorial Database).

19 Cache Management

Understanding how the internal AreaList Pro cache works, and how to manage it depending on the area changes performed by the user or programmatically can help optimize your code and know when to refresh displayed data.

Data updating, Data checking and Cache clearing

Three properties

Calling:

AL_SetAreaLongProperty ($area; ALP_Area_UpdateData; 0)

is the same as calling:

AL_SetAreaLongProperty ($area; ALP_Area_ClearCache; -2)
AL_SetAreaLongProperty ($area; ALP_Area_CheckData; 0)

Examples

After a single row was modified in arrays / selection (refresh only this row):

AL_SetAreaLongProperty ($area ; ALP_Area_ClearCache; $rownum)

After several rows were modified (e.g. sorted):

AL_SetAreaLongProperty ($area ; ALP_Area_ClearCache; -2)

or you might call (but it is really different in the amount of work done, depending on settings):

AL_SetAreaLongProperty ($area ; ALP_Area_UpdateData; 0)

After resizing arrays/selection without modifying the data (e.g. adding/removing a row at the end):

AL_SetAreaLongProperty ($area ; ALP_Area_CheckData; 0)

After resizing arrays/selection and the data was modified (e.g. showing a new selection):

AL_SetAreaLongProperty ($area ; ALP_Area_UpdateData; 0)

Upgrading from previous API

To map the AreaList Pro v8.x ***AL_UpdateArrays*** / ***AL_UpdateFields*** commands:

1 → ***AL_SetAreaLongProperty*** ($area ; ALP_Area_ClearCache; -2)

2 → ***AL_SetAreaLongProperty*** ($area ; ALP_Area_UpdateData; 0)

AreaList Pro version 8 refresh commands vs version 11 cache management

Version 8: no data caching, fields/arrays are accessed during the update event to get values to draw.

Version 11: displayed data are fully cached, no access to fields/arrays on update if not needed.

The ALP_Area_ClearCache property with value -2 is similar to v8.x ***AL_UpdateArrays*** (-1): the cache is cleared, new data are fetched, data sorted if ALP_Area_SortDuring is set to yes.

The size of the arrays/selection should not change, however AreaList Pro should survive if it is modified.

Using ALP_Area_UpdateData is similar to ***AL_UpdateArrays*** (-2): the cache is cleared, data are sorted, width of columns is computed…

Using ALP_Area_CheckData is something between the two above: the cache is not cleared, the size of arrays/selection is checked, no sort is performed, column widths are calculated.

When AreaList Pro has to display data and they are not in the cache, they are fetched from arrays/fields. Only visible rows are fetched. Then the area is drawn.

So once an area was displayed (or the cache was filled because it was required by a call), AreaList Pro will not access 4D during the update event.

Cache clearing or Data updating

You may wonder whether “clear cache” means that we are syncing the plugin to the underlying 4D data (arrays/fields) or that we are clearing AreaList Pro’s cache (thereby leaving everything blank).

Cache clearing means that the internal cache is cleared.

ALP_Area_ClearCache does just that: all data to be displayed are re-fetched from 4D.

ALP_Area_UpdateData does much much more: sort selection/arrays, compute column widths, etc. then fill the cache.

Syncing is performed on demand, as described above: during an update event or during a call when it is required (like ALP_Row_Reveal) or explicitly (e.g. using ALP_Area_FillCache).

When ALP_Area_UpdateData, ALP_Area_CheckData or ALP_Area_FillCache is used, the cache is filled with the data.

The optional parameter is the number of rows to fetch. If not specified (< 1), the number of visible rows (as calculated for fixed row height) is used.

Unnecessary updates

There is no need to call ***AL_SetAreaLongProperty*** ($eList; ALP_Area_UpdateData;0) in the On Load event when configuring the area.

If the data is already in the cache, it will be unnecessarily processed again: fetching data, sorting, measuring… For example, with a large selection this means that the selection will be sorted again. However, besides this unneeded re-processing, it should not harm.

Also, when editing an enterable cell, you don’t have to clear the cache if you don’t modify other rows. You can safely modify the currently edited row (the whole row is re-fetched automatically).

20 Codes

AreaList Pro Error Codes

Registration Error Codes

Error codes that may result from a call to AL_Register.

Error Number	Constant	Description
0	AL Registration passed	OK
1		Beta license has expired
2	AL Registration failed	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 registration)
10		Unauthorized master key (Online registration)
11		Can’t connect to the license server to perform Online registration
12		No Online registration license available for this master key (unknown or all used)

Result Codes

All function calls return a longint result code, with 0 meaning that the function executed successfully. All other possible error codes are listed below along with their constants:

Error Number	Constant	Description
-1	ALP_Err_Generic	General error, often returned by submodules (like XML parser).
0	ALP_Err_OK	No error: the function call was successful.
1	ALP_Err_CantLoadXML	The XML variable could not be loaded
2	ALP_Err_CantSaveXML	The XML variable could not be saved.
3	ALP_Err_InvalidAreaRef	You have passed an invalid AreaList Pro area reference to the function.
4	ALP_Err_InvalidObjectRef	Invalid object reference was passed to commands that require an object reference.
5	ALP_Err_InvalidRequest	There are no objects of the requested type.
6	ALP_Err_InvalidArrayType	The wrong type of array was passed.
7	ALP_Err_InvalidNilPointer	An invalid pointer was passed - ie, it does not point to a valid object.
8	ALP_Err_InvalidPointerType	AreaList Pro was not able to cast from the internal type to the types of variables passed in pointer parameters. Any type can be cast to string; Booleans can be cast to number variables.
9	ALP_Err_InvalidArraySize	This error means that you have passed two or more arrays to the function, and they do not all contain the same number of elements.
10	ALP_Err_CantLoadRecord	When using ALP_Cell_Value (old ***AL_SetCellValue***) to modify record (record is locked).
11	ALP_Err_CantSaveRecord	When using ALP_Cell_Value (old ***AL_SetCellValue***) to modify record (**SAVE RECORD** failed).

Error #-9939

The AreaList Pro plugin was not loaded correctly. Please refer to the installation instructions to make sure that you have installed the plugin in the correct place.

AreaList Pro Event Codes

The user's last AreaList Pro-related action is stored in the ALP_Area_ALPEvent property. You can find out what it was by calling the ***AL_GetAreaLongProperty*** command - for example:

$event:= ***AL_GetAreaLongProperty*** ($area;ALP_Area_AlpEvent)

$area can be null (all areas) for most properties.

Events can be captured in the AreaList Pro area's object method or in the Event callback method, but some can **only** be captured in the Event callback method.

With regards to the Drag and Drop events where either option is possible, the preferred method is to use the On Drop form event.

Specifically, AL Row drop event, AL Column drop event and AL Cell drop event are reported during On Drop.

Constant	Value	Callback only	User action
AL Null event	0		No action
AL Single click event	1		Single-click (or up/down arrow keys)
AL Double click event	2		Double-click
AL Empty Area Single click	3		Single-click in an empty part of the area (without displayed data)
AL Empty Area Double click	4		Double-click in an empty part of the area (without displayed data)
AL Single Control Click	5		Control-click (or right mouse click)
AL Empty Area Control Click	6		Control-click (or right mouse click) in an empty part of the area (without displayed data)
AL Vertical Scroll Event	7		Vertical scroll
AL Row drop event	8		Row(s) dropped to the area
AL Column drop event	9		Column(s) dropped to the area
AL Cell drop event	10		Cell(s) dropped to the area
AL Allow drop event	11	✔	Requires ALP_Area_CallbackMethOnEvent Reported during Drag from non-AreaList Pro source - respond in event callback with $0:=1 to allow drop or $0 = 0 to disallow
AL Hierarchy collapse event	12		A hierarchy level was collapsed
AL Hierarchy expand event	13		A hierarchy level was expanded
AL Object drop event	14		Something was dropped from a non-AreaList Pro object (such as a 4D list or variable)
AL Typeahead event	15		Reported when typeahead occurs and ALP_Area_TypeAheadEffect is set to -2
AL Horizontal Scroll Event	16		Horizontal scroll
AL Mouse moved event	18	✔	Mouse moved (including over the "empty column" on the right)
AL Key event	19		Key pressed or posted when ALP_Area_Event_Filter bit 1 is zero
AL Mouse entry unsel row event	101		Obsolete, never reported
AL Sort button event	-1		Sort button
AL Select all event	-2		Edit menu Select All
AL Column resize event	-3		Column resized
AL Column lock event	-4		Column lock changed
AL Row drag event	-5		Row(s) dragged from the area
AL Sort editor event	-6		Sort editor
AL Column drag event	-7		Column(s) dragged from the area
AL Cell drag event	-8		Cell(s) dragged from the area
AL Object resize event	-9		Object and window resized

Constant	Value	Callback only	User action
AL Column click event	-10		User clicked on column header, automatic sort won't be executed
AL Column control click event	-11		Control-click on column header
AL Footer click event	-12		Click on column footer
AL Entry Edit event	-13		Reported on most Edit menu actions while entry is in progress – should be ignored

AreaList Pro Text Style Tags

If the ALP_Column_Attributed option has been set, special tags can also be used in any text contained in an AreaList Pro area to display styled characters.

Note: the tags described below are exacty the same as in SuperReport Pro.

These tags work just like HTML tags: <tag>styled text</tag>.

Style	Tag
Bold	<b>
Italic	<i>
Underline	<u> or <ins>
Strike-through	<del>
Set font size to # points	<s #>
Increase font size by # quarters (1/4) of current size	<s +#>
Decrease font size by # quarters (1/4) of current size	<s -#>
Set font by name	<f "font name"> (needs to be quoted if the name contains more than one word)
Set color (any format can be used, e.g. <c 0xFFFF0000> <c 1.0,0,0> <c P123> <c dark orange>)	<c color name>

4D v12 (and above) internal format for styled text is stored as <SPAN STYLE="style attributes"> where the style attributes used by AreaList 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 ALP_Column_Attributed and ALP_Column_Format properties.

Example for an integer column:

```
AL_SetColumnLongProperty ($area;$column;ALP_Column_Attributed;1)  // the column is "attributed"
AL_SetColumnTextProperty ($area;$column;ALP_Column_Format;\
    "<c blue>+## ###</c>;<i><c red>-## ###</c></i>;<s +1><c green><b>ZERO</b></c></s>")
```

With the above settings:

- Positive numbers will be displayed in blue roman characters with a plus sign.
- Negative numbers will be displayed in red italic characters with a minus sign.
- Zeros will be displayed in green bold, font size increased by 25%, with the text “ZERO”.

Here is the result:

```
-16 238
 -5 526
-31 880
-21 940
 +4 137
   -100
   ZERO
+27 512
 -9 330
+21 250
   +707
+28 936
-30 953
-24 692
+24 109
-24 352
```

Note that if the number format is too “small” to hold the number, 4D (and AreaList Pro) will display it as “<<<<<<<<<<<<<<<<<<”, which will interfere with the opening tag character “<” if the column is attributed (multi-styled).

In the example above (using “## ###” as a number format), this will be the case for all numbers exceeding 99,999.

Make sure that the format used will not cause the number to overflow, lest unexpected results might ensue.

Note: the AL_GetPlainText command converts attributed text to plain text.

Property Values, Constants and XML Names

Properties shown in green were added in Version 10.

Properties shown in magenta were added in Version 11.

Property Constant	Property Value (selector)	Property XML Name
ALP_Area_AccessFocusedTarget	Stgt	
ALP_Area_AllowSortEditor	soed	allowEditor
ALP_Area_AlpEvent	evtL	
ALP_Area_AltHdrClear	AlHC	
ALP_Area_AltHdrRowsInGrid	rowA	rowsInGrid
ALP_Area_AltRowColor	altc	altRowColor
ALP_Area_AltRowOptions	alto	altRowColorParams
ALP_Area_APITarget	ATgt	
ALP_Area_Appearance	appe	appearance
ALP_Area_ArrowsForHierarchy	arkh	
ALP_Area_AutoHierarchy	hilb	autoHierarchy
ALP_Area_AutoResizeAllColumns	Snap	autoResizeAllColumns
ALP_Area_AutoResizeColumn	SNAP	autoResizeColumn
ALP_Area_AutoSnapLastColumn	snap	autoSnapLastCol
ALP_Area_BottomRow	arbt	
ALP_Area_CacheSize	cacs	cacheSize
ALP_Area_CalcAllRows	vwAL	
ALP_Area_CalendarColors	caco	
ALP_Area_CalendarLook	calo	
ALP_Area_CallbackMethDeselect	apde	deselectCallback
ALP_Area_CallbackMethEntryEnd	apef	endCallback
ALP_Area_CallbackMethEntryStart	apes	startCallBack
ALP_Area_CallbackMethMenu	apme	menuCallback
ALP_Area_CallbackMethOnEvent	apcb	eventCallBack
ALP_Area_CallbackMethPopup	appc	popupCallback
ALP_Area_CallbackMethSelect	apse	selectCallback
ALP_Area_CheckData	DATA	
ALP_Area_ClearCache	cach	
ALP_Area_ClickDelay	edel	clickHoldDelay
ALP_Area_ClickedCell	evcC	
ALP_Area_ClickedCol	evcc	
ALP_Area_ClickedColNum	evCc	
ALP_Area_ClickedRow	evcr	
ALP_Area_ClickedRowNum	evCr	
ALP_Area_ClickedVisualRow	evcR	
ALP_Area_ColDivColor	colc	colDividerColor
ALP_Area_Collection	DSco	
ALP_Area_ColsInGrid	coln	colsInGrid

Property Constant	Property Value (selector)	Property XML Name
ALP_Area_ColsLocked	coll	lockedCols
ALP_Area_ColumnLock	clck	allowColumnLock
ALP_Area_ColumnResize	cres	allowColumnResize
ALP_Area_ColumnResizeDistance	cred	columnResizeDistance
ALP_Area_Columns	COLS	numColumns
ALP_Area_Compatibility	comp	compatibility
ALP_Area_CompHideCols	cohc	hideColumns
ALP_Area_CopyFieldSep	ecfd	copyFieldDelimiter
ALP_Area_CopyFieldWrapper	ecfw	copyFieldWrapper
ALP_Area_CopyHiddenCols	echc	copyHiddenColumns
ALP_Area_CopyOptions	ecop	copyOptions
ALP_Area_CopyRecordSep	ecrd	copyRecordDelimiter
ALP_Area_Copyright	copy	
ALP_Area_CurrentEntity	DSce	
ALP_Area_DataClassName	DScn	dataClass
ALP_Area_DataClassObject	DScl	
ALP_Area_DataHeight	Adhi	
ALP_Area_DataSource	DSty	dataSource
ALP_Area_DataWidth	Adwd	
ALP_Area_DefFmtBoolean	dfbo	
ALP_Area_DefFmtDate	dfda	
ALP_Area_DefFmtInteger	dfin	
ALP_Area_DefFmtLong	dflo	
ALP_Area_DefFmtPicture	dfpi	
ALP_Area_DefFmtReal	dfre	
ALP_Area_DefFmtTime	dfti	
ALP_Area_DeleteOffscreen	XDOS	
ALP_Area_DontSetCursor	DSCU	
ALP_Area_DontSortArrays	sono	dontSortArrays
ALP_Area_DoubleClick	evtD	
ALP_Area_DPI_X	dpiX	
ALP_Area_DPI_Y	dpiY	
ALP_Area_DragAcceptColumn	ddac	acceptColumnDrag
ALP_Area_DragAcceptLine	ddal	acceptLineDrag
ALP_Area_DragColumn	ddco	dragColumn
ALP_Area_DragDataType	ddDT	
ALP_Area_DragDstArea	ddDA	
ALP_Area_DragDstCell	ddDc	
ALP_Area_DragDstCellCodes	dddc	dstCellCodes
ALP_Area_DragDstCol	ddDC	
ALP_Area_DragDstColCodes	dddC	dstColCodes
ALP_Area_DragDstProcessID	ddDP	
ALP_Area_DragDstRow	ddDR	

Property Constant	Property Value (selector)	Property XML Name
ALP_Area_DragDstRowCodes	dddR	dstRowCode
ALP_Area_DragLine	ddln	dragLine
ALP_Area_DragMoveColumns	ddmc	dragMoveColumns
ALP_Area_DragOptionKey	ddra	dragWithOptionKey
ALP_Area_DragProcessID	ddpn	
ALP_Area_DragRowMultiple	ddrm	multipleRowDrag
ALP_Area_DragRowOnto	ddro	ontoRow
ALP_Area_DragScroll	ddps	dragScroll
ALP_Area_DragSrcArea	ddSA	
ALP_Area_DragSrcCell	ddSc	
ALP_Area_DragSrcCellCodes	ddsc	srcCellCodes
ALP_Area_DragSrcCol	ddSC	
ALP_Area_DragSrcColCodes	ddsC	srcColCodes
ALP_Area_DragSrcRow	ddSR	
ALP_Area_DragSrcRowCodes	ddsR	srcRowCodes
ALP_Area_DrawFrame	drfr	drawFrame
ALP_Area_EmptyRowColor	emrc	emptyRowColor
ALP_Area_EntitySelection	DSes	
ALP_Area_EntryAllowArrows	ecnr	navigateUsingArrows
ALP_Area_EntryAllowReturn	ecar	allowReturn
ALP_Area_EntryAllowSeconds	ects	allowSeconds
ALP_Area_EntryCell	eceC	
ALP_Area_EntryCellColumn	ecCC	
ALP_Area_EntryCellRow	ecCR	
ALP_Area_EntryClick	eccl	entryClick
ALP_Area_EntryColumn	ecec	
ALP_Area_EntryEditedEntity	eceO	
ALP_Area_EntryExit	ecex	
ALP_Area_EntryFirstClickMode	ecch	firstClick
ALP_Area_EntryGotoCell	ecgC	
ALP_Area_EntryGotoColumn	ecgc	
ALP_Area_EntryGotoGridCell	ecgg	
ALP_Area_EntryGotoRow	ecgr	
ALP_Area_EntryGridCell	eceg	
ALP_Area_EntryHighlight	echi	
ALP_Area_EntryHighlightE	eche	
ALP_Area_EntryHighlightS	echs	
ALP_Area_EntryInProgress	ecea	
ALP_Area_EntryMapEnter	ecme	mapEnterKey
ALP_Area_EntryMethod	enem	
ALP_Area_EntryModified	ecem	
ALP_Area_EntryPrevCell	ecpC	
ALP_Area_EntryPrevColumn	ecpc	

Property Constant	Property Value (selector)	Property XML Name
ALP_Area_EntryPrevGridCell	ecpg	
ALP_Area_EntryPrevRow	ecpr	
ALP_Area_EntryRow	ecer	
ALP_Area_EntrySelectedText	eces	
ALP_Area_EntrySkip	ecsk	
ALP_Area_EntryText	ecet	
ALP_Area_EntryValue	ecev	
ALP_Area_Event	evtT	
ALP_Area_Event_Filter	evtF	
ALP_Area_EventChar	evtC	
ALP_Area_EventKey	evtK	
ALP_Area_EventModifiers	evtM	
ALP_Area_EventPosH	evtH	
ALP_Area_EventPosV	evtV	
ALP_Area_EventTarget	ETgt	
ALP_Area_ExportList	expL	exportList
ALP_Area_ExportOptions	expO	
ALP_Area_ExportToFile	expF	
ALP_Area_FillCache	data	
ALP_Area_FillNumberSign	fils	
ALP_Area_FocusedTarget	FTgt	
ALP_Area_FrameColor	Pfrc	frameColor
ALP_Area_FtrIndent	frin	footerIndent
ALP_Area_FtrIndentH	frih	
ALP_Area_FtrIndentV	friv	
ALP_Area_GroupHeader	grpc	groupHeader
ALP_Area_HdrIndent	hrin	headerIndent
ALP_Area_HdrIndentH	hrih	
ALP_Area_HdrIndentV	hriv	
ALP_Area_HdrPictCallback	upcc	userHdrPictCallBack
ALP_Area_HdrPictFlags	upcf	userHdrPictFlags
ALP_Area_HdrPictID	upci	userHdrPictID
ALP_Area_HeaderDivColor	Phdc	headerDividerColor
ALP_Area_HeaderMode	hdrm	headerMode
ALP_Area_HideEntryFocus	Efoc	hideEntryFocus
ALP_Area_HideHeaders	hhid	hideHeaders
ALP_Area_HideLastHdrDivider	hhld	hideLastHdrDiv
ALP_Area_HierIndent	hiid	indentHierarchy
ALP_Area_HighlightColor	uihc	
ALP_Area_IgnoreMenuMeta	emet	ignoreMenuMeta
ALP_Area_IgnoreMouseWheel	eimw	
ALP_Area_IsArea	isEA	
ALP_Area_Kind	kind	

Property Constant	Property Value (selector)	Property XML Name
ALP_Area_LastError	(empty string)	
ALP_Area_LimitRows	limR	
ALP_Area_ListHeight	Alhi	
ALP_Area_ListLeft	Alrl	
ALP_Area_ListTop	Alrt	
ALP_Area_ListWidth	Alwd	
ALP_Area_MaxCellHeight	cMax	maxCellHeight
ALP_Area_MaxRowLines	rowm	maxRowLines
ALP_Area_MinFtrHeight	fmin	minFooterHeight
ALP_Area_MinHdrHeight	hmin	minHeaderHeight
ALP_Area_MinRowHeight	rmin	minRowHeight
ALP_Area_MiscColor1	mic1	miscColor1
ALP_Area_MiscColor2	mic2	miscColor2
ALP_Area_MiscColor3	mic3	miscColor3
ALP_Area_MiscColor4	mic4	miscColor4
ALP_Area_MiscColor5	mic5	miscColor5
ALP_Area_MoveCellOptions	como	moveCellOptions
ALP_Area_MoveInFieldsMode	fomo	moveInFieldsMode
ALP_Area_MoveRowOptions	romo	moveRowOptions
ALP_Area_Name	name	name
ALP_Area_NamedSelection	nams	
ALP_Area_NewOffscreen	XNOS	
ALP_Area_NumFtrLines	ftrl	footerLines
ALP_Area_NumHdrLines	hdrl	headerLines
ALP_Area_NumRowLines	rowl	rowLines
ALP_Area_OneLineFtr	rowF	singleFooter
ALP_Area_OneLineHdr	rowH	singleHeader
ALP_Area_ParentALP	PAlp	
ALP_Area_ParentColumn	PCol	
ALP_Area_Path	path	
ALP_Area_PrintArea	PRNT	
ALP_Area_PrintOptions	PROP	printOptions
ALP_Area_ReadOnly	ronl	
ALP_Area_Redraw	upds	
ALP_Area_ResizeDuring	redu	resizeDuring
ALP_Area_RollOverCell	evlC	
ALP_Area_RollOverCol	evlc	
ALP_Area_RollOverColNum	evLc	
ALP_Area_RollOverRow	evlr	
ALP_Area_RollOverRowNum	evLr	
ALP_Area_RollOverVisualRow	evlR	
ALP_Area_RowDivColor	rowc	rowDividerColor
ALP_Area_RowHeight	rowh	

Property Constant	Property Value (selector)	Property XML Name
ALP_Area_RowHeightFixed	rowf	rowHeightFixed
ALP_Area_RowIndent	roin	rowIndent
ALP_Area_RowIndentH	roih	
ALP_Area_RowIndentV	roiv	
ALP_Area_Rows	ROWS	
ALP_Area_RowsInGrid	rown	rowsInGrid
ALP_Area_ScrollbarArrowsPos	scap	
ALP_Area_ScrollbarStyle	scst	scrollbarStyle
ALP_Area_ScrollColumns	scco	scrollColumns
ALP_Area_ScrollLeft	scrl	
ALP_Area_ScrollTop	scrt	
ALP_Area_SelClick	selC	selectClick
ALP_Area_SelCol	selc	
ALP_Area_Select	SELC	
ALP_Area_Selected	sele	
ALP_Area_Self	self	
ALP_Area_SelGotoRec	seLR	
ALP_Area_SelHighlightMode	seld	highlightMode
ALP_Area_SelKeepOnTypeAhead	selT (obsolete, replace with ALP_Area_TypeAheadEffect)	
ALP_Area_SelMultiple	selm	multipleRows
ALP_Area_SelNoAutoSelect	selA	noAutoSelect
ALP_Area_SelNoCtrlSelect	selX	noCtrlSelect
ALP_Area_SelNoDeselect	selD	noDeselect
ALP_Area_SelNoHighlight	selh	noHighlight
ALP_Area_SelNone	seln	allowNoSelection
ALP_Area_SelPreserve	selP	preserve
ALP_Area_SelRow	selr	
ALP_Area_SelSetName	seCS	
ALP_Area_SelType	selt	type
ALP_Area_SendEvent	seev	
ALP_Area_ShowColDividers	cold	showColDivider
ALP_Area_ShowFocus	focu	showFocus
ALP_Area_ShowFooters	fshw	showFooters
ALP_Area_ShowHScroll	scrh	showHScroll
ALP_Area_ShowRowDividers	rowd	showRowDivider
ALP_Area_ShowSortEditor	soED	
ALP_Area_ShowSortIndicator	hdrs	showSortIndicator
ALP_Area_ShowVScroll	scrv	showVScroll
ALP_Area_ShowWidths	dwdt	displayPointWidth
ALP_Area_SmallScrollbar	scsm	smallScrollbar
ALP_Area_Sort	SORT	
ALP_Area_SortCancel	soca	cancel
ALP_Area_SortColumn	soco	sortColumn

Property Constant	Property Value (selector)	Property XML Name
ALP_Area_SortDuring	sodu	sortDuring
ALP_Area_SortList	soli	sortList
ALP_Area_SortListNS	soLI	
ALP_Area_SortOK	sook	ok
ALP_Area_SortOnLoad	sold	sortOnLoad
ALP_Area_SortPrompt	sopr	prompt
ALP_Area_SortTitle	soti	title
ALP_Area_SRPTableTemplate	SRPt	
ALP_Area_TableID	tbid	mainTable
ALP_Area_ToolTip	tips	
ALP_Area_TopRow	artp	
ALP_Area_TraceOnError	TRAC	
ALP_Area_Transposed	dtrn	transpose
ALP_Area_TypeAheadEffect	selT	
ALP_Area_TypeAheadFieldMode	tahF	
ALP_Area_TypeAheadString	tahS	
ALP_Area_TypeAheadTime	tahT	
ALP_Area_UpdateData	daup	
ALP_Area_UseDateControls	endc	useDateControls
ALP_Area_UseEllipsis	elip	useEllipsis
ALP_Area_UserBLOB	usrb	userBLOB
ALP_Area_UserCollection	Ucol	userCollection
ALP_Area_UserSort	sous	userSort
ALP_Area_UseTimeControls	entc	useTimeControls
ALP_Area_Version	vers	
ALP_Area_Visible	visi	
ALP_Area_WindowsClip	CLIP	
ALP_Area_WindowsText	DIEN	
ALP_Area_XML	xml	
ALP_Area_XMLAP	Axml	
ALP_Area_Zoom	zoom	
ALP_Break_Clear	clr	
ALP_Break_ClearBreaks	PLcb	
ALP_Break_FooterCalcMethod	PbfC	footerCallback
ALP_Break_HeaderCalcMethod	PbhC	headerCallback
ALP_Break_HideDetails	Pdet	hideDetails
ALP_Break_IsHeader	PBbh	isHeader
ALP_Break_Level	PBbl	level
ALP_Break_LineColor	PLlc	color
ALP_Break_LineShowColumnDivider	PLsd	showColDivider
ALP_Break_LineThickness	PLlt	thickness
ALP_Break_MinRowHeight	PBwm	minRowHeight
ALP_Break_NumRowLines	PBrl	numLines

Property Constant	Property Value (selector)	Property XML Name
ALP_Break_RowID	PBbr	
ALP_Break_RowIndent	PBri	indent
ALP_Break_RowIndentH	PBih	
ALP_Break_RowIndentV	PBiv	
ALP_Break_RowsInGrid	rown	rowsInGrid
ALP_Break_SelRow	selr	
ALP_BreakCell_ColSpan	gcsp	colSpan
ALP_BreakCell_MergeCenter	PBMc	
ALP_BreakCell_MergeDefault	PBMd	
ALP_BreakCell_MergeLeft	PBMl	
ALP_BreakCell_MergeRight	PBMr	
ALP_BreakCell_RowSpan	grsp	rowSpan
ALP_BreakCell_Source	src	colNum
ALP_BreakCell_Value	valu	
ALP_Cell_BackColor	bclr	backColor
ALP_Cell_BaseLineShift	basl	baseLineShift
ALP_Cell_BottomBorderColor	cobc	bottomBorder_color
ALP_Cell_BottomBorderOffset	cobo	bottomBorder_offset
ALP_Cell_BottomBorderWidth	cobw	bottomBorder_width
ALP_Cell_Clear	clr	
ALP_Cell_ClearStyle	sclr	
ALP_Cell_Enterable	ente	enterable
ALP_Cell_FillColor	cofc	fillColor
ALP_Cell_Flags	flgs	features
ALP_Cell_FontName	fnam	font
ALP_Cell_Format	fmt	format
ALP_Cell_FormatResolved	Rfmt	
ALP_Cell_FormattedValue	valf	
ALP_Cell_HorAlign	halg	halign
ALP_Cell_HorizontalScale	hors	hScale
ALP_Cell_Invisible	visi	invisible
ALP_Cell_Kind	kind	
ALP_Cell_LeftBorderColor	colc	leftBorder_color
ALP_Cell_LeftBorderOffset	colo	leftBorder_offset
ALP_Cell_LeftBorderWidth	colw	leftBorder_width
ALP_Cell_LeftIconFlags	colF	leftIconFlags
ALP_Cell_LeftIconID	colI	leftIconID
ALP_Cell_LineSpacing	lisp	lineSpacing
ALP_Cell_Reveal	reve	
ALP_Cell_RightBorderColor	corc	rightBorder_color
ALP_Cell_RightBorderOffset	coro	rightBorder_offset
ALP_Cell_RightBorderWidth	corw	rightBorder_width
ALP_Cell_RightIconFlags	corF	rightIconFlags

Property Constant	Property Value (selector)	Property XML Name
ALP_Cell_RightIconID	corl	rightIconID
ALP_Cell_Rotation	rotd	rotation
ALP_Cell_RowDivColor	rowc	rowDivColor
ALP_Cell_ScrollTo	scto	
ALP_Cell_Size	size	size
ALP_Cell_StyleB	styB	bold
ALP_Cell_StyleF	styF	qdStyle
ALP_Cell_StyleI	styI	italic
ALP_Cell_StyleS	styS	strike-through
ALP_Cell_StyleU	styU	underline
ALP_Cell_TextColor	tclr	textColor
ALP_Cell_TopBorderColor	cotc	topBorder_color
ALP_Cell_TopBorderOffset	coto	topBorder_offset
ALP_Cell_TopBorderWidth	cotw	topBorder_width
ALP_Cell_UseEllipsis	elip	useEllipsis
ALP_Cell_Value	valu	
ALP_Cell_VertAlign	valg	valign
ALP_Cell_Wrap	wrap	wrap
ALP_Cell_XML	xml	
ALP_Column_Attributed	attr	attributed
ALP_Column_BackColor	bclr	backColor
ALP_Column_BaseLineShift	basl	baseLineShift
ALP_Column_CalcHeight	chig	autoHeight
ALP_Column_Calculated	calc	calculated
ALP_Column_Callback	call	callback
ALP_Column_CallbackFnc	calF	formula
ALP_Column_ColDivColor	colc	colDividerColor
ALP_Column_ColDivThickness	Pcdt	colDividerThickness
ALP_Column_DisplayControl	disp	displayControl
ALP_Column_Draggable	cand	draggable
ALP_Column_Enterable	ente	enterable
ALP_Column_EntryAllowReturn	ecar	allowReturn
ALP_Column_EntryControl	entc	entryControl
ALP_Column_Filter	entf	filter
ALP_Column_FindCell	coce	
ALP_Column_FocusableCheckbox	eccf	checkboxFocusable
ALP_Column_FontName	fnam	font
ALP_Column_FooterText	ftxt	footerText
ALP_Column_Format	fmt	format
ALP_Column_FormatResolved	Rfmt	
ALP_Column_FromCell	ceco	
ALP_Column_FtrBackColor	fbcl	backColor
ALP_Column_FtrBaseLineShift	fbls	baseLineShift

Property Constant	Property Value (selector)	Property XML Name
ALP_Column_FtrFontName	ffnm	font
ALP_Column_FtrHorAlign	fhal	halign
ALP_Column_FtrHorizontalScale	fhos	hScale
ALP_Column_FtrLineSpacing	flis	lineSpacing
ALP_Column_FtrRotation	frot	rotation
ALP_Column_FtrSize	fsiz	size
ALP_Column_FtrStyleB	fstB	bold
ALP_Column_FtrStyleF	fstF	qdStyle
ALP_Column_FtrStyleI	fstI	italic
ALP_Column_FtrStyleS	fstS	strike-through
ALP_Column_FtrStyleU	fstU	underline
ALP_Column_FtrTextColor	ftcl	textColor
ALP_Column_FtrUseEllipsis	feli	useEllipsis
ALP_Column_FtrVertAlign	fval	valign
ALP_Column_FtrWrap	fwrp	wrap
ALP_Column_GroupHdrText	gtxt	groupHdrText
ALP_Column_GroupID	gcol	groupID
ALP_Column_HdrBackColor	hbcl	backColor
ALP_Column_HdrBaseLineShift	hbls	baseLineShift
ALP_Column_HdrFontName	hfnm	font
ALP_Column_HdrHorAlign	hhal	halign
ALP_Column_HdrHorizontalScale	hhos	hScale
ALP_Column_HdrLineSpacing	hlis	lineSpacing
ALP_Column_HdrRotation	hrot	rotation
ALP_Column_HdrSize	hsiz	size
ALP_Column_HdrStyleB	hstB	bold
ALP_Column_HdrStyleF	hstF	qdStyle
ALP_Column_HdrStyleI	hstI	italic
ALP_Column_HdrStyleS	hstS	strike-through
ALP_Column_HdrStyleU	hstU	underline
ALP_Column_HdrTextColor	htcl	textColor
ALP_Column_HdrUseEllipsis	heli	useEllipsis
ALP_Column_HdrVertAlign	hval	valign
ALP_Column_HdrWrap	hwrp	wrap
ALP_Column_HeaderText	htxt	headerText
ALP_Column_HorAlign	halg	halign
ALP_Column_HorizontalScale	hors	hScale
ALP_Column_ID	id	id
ALP_Column_Indexed	idx	dataIndexed
ALP_Column_Kind	kind	
ALP_Column_Length	len	dataSize
ALP_Column_LineSpacing	lisp	lineSpacing
ALP_Column_Locked	lock	locked

Property Constant	Property Value (selector)	Property XML Name
ALP_Column_MaxWidth	widM	
ALP_Column_MinWidth	widm	minWidth
ALP_Column_PlaceHolder	entF	placeHolder
ALP_Column_PopupArray	entp	entryPopup
ALP_Column_PopupArrayKind	entP	kind
ALP_Column_PopupEntryType	enPE	popupEntryType
ALP_Column_PopupIconID	entI	popupIconID
ALP_Column_PopupMap	entM	entryMap
ALP_Column_PopupMenu	entm	
ALP_Column_PopupName	entN	entryPopupName
ALP_Column_Proportional	cwid	relativeWidth
ALP_Column_Resizable	canr	resizable
ALP_Column_Reveal	reve	
ALP_Column_Rotation	rotd	rotation
ALP_Column_ScrollTo	scto	
ALP_Column_Size	size	size
ALP_Column_Sortable	cans	sortable
ALP_Column_SortDirection	sord	sort
ALP_Column_SortFormula	cobf	sortFormula
ALP_Column_Source	src	source
ALP_Column_StyleB	styB	bold
ALP_Column_StyleF	styF	qdStyle
ALP_Column_StyleI	styI	italic
ALP_Column_StyleS	styS	strike-through
ALP_Column_StyleU	styU	underline
ALP_Column_SubALP	CAlp	subALP
ALP_Column_TextColor	tclr	textColor
ALP_Column_Type	type	dataType
ALP_Column_Uppercase	entu	uppercase
ALP_Column_UseEllipsis	elip	useEllipsis
ALP_Column_UserCollection	Ucol	userCollection
ALP_Column_UserText	Utxt	userText
ALP_Column_VertAlign	valg	valign
ALP_Column_Visible	visi	visible
ALP_Column_Width	widt	width
ALP_Column_WidthUser	uwdt	userWidth
ALP_Column_Wrap	wrap	wrap
ALP_Column_XML	xml	
ALP_Drop_DragAcceptColumn	ddac	acceptColumnDrag
ALP_Drop_DragAcceptLine	ddal	acceptLineDrag
ALP_Drop_DragDstCodes	dddc	dstCodes
ALP_Drop_DragProcessID	ddpn	
ALP_Drop_DragSrcArea	ddSA	

Property Constant	Property Value (selector)	Property XML Name
ALP_Drop_Kind	kind	
ALP_Drop_Name	name	name
ALP_Drop_XML	xml	
ALP_Object_AltHdrGrid	GRIA	AlternateGrid
ALP_Object_AutoHierarchy	hilb	autoHierarchy
ALP_Object_Columns	COLS	
ALP_Object_ColumnWidth	widt	
ALP_Object_ColumnWidthUser	uwdt	
ALP_Object_DragDstCellCodes	dddc	
ALP_Object_DragDstColCodes	dddC	
ALP_Object_DragDstRowCodes	dddR	
ALP_Object_DragSrcCellCodes	ddsc	
ALP_Object_DragSrcColCodes	ddsC	
ALP_Object_DragSrcRowCodes	ddsR	
ALP_Object_ExportList	expL	
ALP_Object_Fields	Xsrc	
ALP_Object_FooterText	ftxt	
ALP_Object_FooterTextNH	fTXT	
ALP_Object_FtrGrid	GRIF	FooterGrid
ALP_Object_Grid	GRID	
ALP_Object_HeaderText	htxt	
ALP_Object_HeaderTextNH	hTXT	
ALP_Object_Hierarchy	HIER	
ALP_Object_RowHide	rhid	
ALP_Object_RowSelection	ROWS	
ALP_Object_Selection	SELC	
ALP_Object_Sort	SORT	
ALP_Object_SortList	soli	
ALP_Object_SortListNS	soLI	
ALP_Object_Source	src	
ALP_Object_Type	type	
ALP_Object_Visible	visi	
ALP_Row_BackColor	bclr	backColor
ALP_Row_BaseLineShift	basl	baseLineShift
ALP_Row_ChildCount	hicc	
ALP_Row_ChildCountAll	hicC	
ALP_Row_Children	hiac	
ALP_Row_ChildrenAll	hiaC	
ALP_Row_Clear	clr	
ALP_Row_ClearStyle	sclr	
ALP_Row_Collapse	hicr	
ALP_Row_CollapseAll	hica	
ALP_Row_Expand	hier	

Property Constant	Property Value (selector)	Property XML Name
ALP_Row_ExpandAll	hiea	
ALP_Row_Flags	flgs	features
ALP_Row_FontName	fnam	font
ALP_Row_Height	high	height
ALP_Row_Hide	rhid	
ALP_Row_HorAlign	halg	halign
ALP_Row_HorizontalScale	hors	hScale
ALP_Row_Kind	kind	
ALP_Row_Level	hile	
ALP_Row_LineSpacing	lisp	lineSpacing
ALP_Row_Parent	hipa	
ALP_Row_Reveal	reve	
ALP_Row_Rotation	rotd	rotation
ALP_Row_RowDivColor	rowc	rowDivColor
ALP_Row_RowOffset	roff	offset
ALP_Row_ScrollTo	scto	
ALP_Row_Size	size	size
ALP_Row_StyleB	styB	bold
ALP_Row_StyleF	styF	qdStyle
ALP_Row_StyleI	styI	italic
ALP_Row_StyleS	styS	strike-through
ALP_Row_StyleU	styU	underline
ALP_Row_StyleXML	Axml	
ALP_Row_TextColor	tclr	textColor
ALP_Row_UseEllipsis	elip	useEllipsis
ALP_Row_VertAlign	valg	valign
ALP_Row_Visible	visi	
ALP_Row_Wrap	wrap	wrap
ALP_Row_XML	xml	

AreaList Pro Edit Menu Constants

Constant	Value
AL Edit Menu Undo Bit	0
AL Edit Menu Redo Bit	1
AL Edit Menu Cut Bit	2
AL Edit Menu Copy Bit	3
AL Edit Menu Paste Bit	4
AL Edit Menu Clear Bit	5
AL Edit Menu Select All Bit	6
AL Edit Menu Entry Bit	15
AL Edit Menu Setup Bit	16
AL Edit Menu Handled Bit	17
AL Edit Menu Undo Mask	1
AL Edit Menu Redo Mask	2
AL Edit Menu Cut Mask	4
AL Edit Menu Copy Mask	8
AL Edit Menu Paste Mask	16
AL Edit Menu Clear Mask	32
AL Edit Menu Select All Mask	64
AL Edit Menu All Items Mask	127
AL Edit Menu Entry Mask	32768
AL Edit Menu Setup Mask	65536
AL Edit Menu Handled Mask	131072

AreaList Pro Modify Arrays Constants

Constant	Value
AL Modify Insert info	0
AL Modify Insert action	1
AL Modify Delete info	2
AL Modify Delete action	3

AreaList Pro Export Options Constants

Constant	Value
AL Export binary XLS	1
AL Export XLSX	2
AL Export No Header	256
AL Export Selected Rows Only	512
AL Export Formatted Date Time	1024
AL Export Formatted As Picture	2048
AL Export Boolean As Text	4096
AL Export No False	8192
AL Export Mapped Values	16384
AL Export Include SubALP	32768
AL Export Using List	65536

AreaList Pro Internal Icons Constants

Constant	Value
AL Icon Generic Popup PictID	-2147483647
AL Icon Date PictID	-2147483646
AL Icon Time PictID	-2147483645
AL Icon Node Right PictID	-2147483644
AL Icon Node Down PictID	-2147483643
AL Icon CB Unchecked PictID	-2147483642
AL Icon CB Checked PictID	-2147483641
AL Icon CB Mixed PictID	-2147483640
AL Icon RB Unchecked PictID	-2147483547
AL Icon RB Checked PictID	-2147483546
AL Icon Sort Up PictID	-2147483621
AL Icon Sort Down PictID	-2147483620
AL Icon Appearance Offset	1664

21 Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

22 Copyrights and Trademarks

All trade names referenced in this document are the trademark or registered trademark of their respective holders.

AreaList Pro, SuperReport Pro and Internet ToolKit are copyright and exclusively published worldwide by Plugin Masters.

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.