9.9.5

AreaList Pro™

User Manual

Version 9.9.5

e-Node
177 cours de l'Argonne
33000 Bordeaux
France

www.e-node.net

Manual revision: 22 Apr 2021

Contents

Getting Started with AreaList Pro 28

Columns 113

Working with Colors 122

The Advanced Properties Dialog 134

Drag and Drop 143

Properties by Theme 220

Mapping Old Commands to the AreaList Pro v9 API 254

DisplayList 261

Printing with SuperReport Pro 271

Cache Management 275

Appendix I Codes 277

Appendix II Troubleshooting and FAQs 292

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 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
- … and lots more!

Technical Details

Compatibility Information

AreaList Pro version 9.9.x is compatible with 4D v11 to v15, for both MacOS and Windows (including 32-bit and 64-bit servers). It requires MacOS 10.5 or higher and Windows XP SP2 or better.

> 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 9 with little or no change in your code. See the v8.5 manual for legacy commands documentation as well as the Upgrading from Previous versions section.

Technical Support

Technical support for AreaList Pro is provided electronically via e-mail or our online support reporting system.

You are encouraged to use the online web forums.

Items that are new or modified in AreaList Pro version 9.8 are displayed in green characters.

Items that are new or modified in AreaList Pro version 9.9.x are displayed in pink (magenta) characters.

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 e-Node 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 4D SQL Desktop, or with 4D Server, either in interpreted or compiled mode (doesn't make a difference regarding plugin licensing).

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

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

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

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

License keys

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

Free updates

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

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

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

License types

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

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

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

  This license allows deployment (selling new application licenses, updates or subscriptions) while the license is valid. **No new deployment may occur after expiry without a specific license** (merged or regular). End-users running deployments sold during the license validity period remain authorized without time limit, provided that they are no longer charged for the application using the plug-in (including maintenance or upgrades).
- **OEM.** This license allows development (interpreted mode) or deployment (interpreted or compiled mode, including merged) on any number of 4D Servers (any number of users), 4D Standalone or single user/remote merged instances that run your 4D application(s).

  It is a yearly license, under the exact same terms as the Unlimited Single User license described above, except that it also covers server deployments.
- **Unlimited OEM.** This license is a global OEM license, covering any combination of the plug-ins published by e-Node, including AreaList Pro, SuperReport Pro, PrintList Pro, CalendarSet and Internet ToolKit in all configurations.
- **Partner license.** This license matches 4D’s annual Partner subscription and covers all the plug-ins published by e-Node, including AreaList Pro, SuperReport Pro, PrintList Pro, CalendarSet and Internet ToolKit.

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

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

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

Registering your 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,
- though 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 e-Node.
4. The plugin will display an alert indicating that it is now registered.

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

Quick and easy way – Developer online instant activation

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

```
C_LONGINT ($result)
$result:=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.

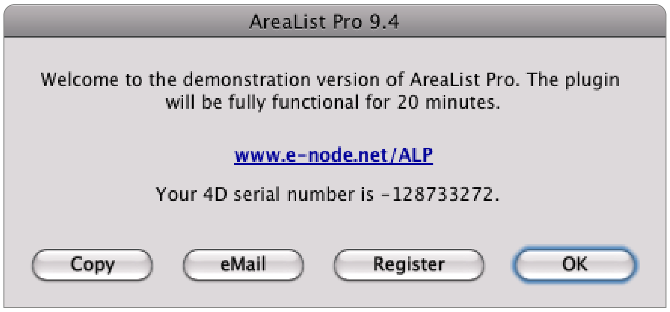
To trigger this display and enable your users to register without actually calling a command or setting up an area, you can also pass an **empty** string to AL_Register and the dialog will show:

```
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 e-Node's registration system and a "Register" button to enter your license key once received:



AreaList Pro 9.4

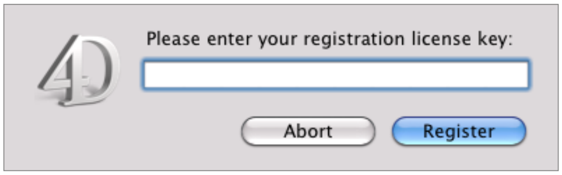
Welcome to the demonstration version of AreaList Pro. The plugin will be fully functional for 20 minutes.

www.e-node.net/ALP

Your 4D serial number is -128733272.

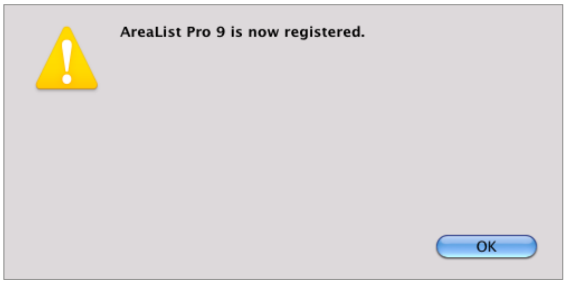
Using the "Register" button

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



Please enter your registration license 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:



AreaList Pro 9 is now registered.

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

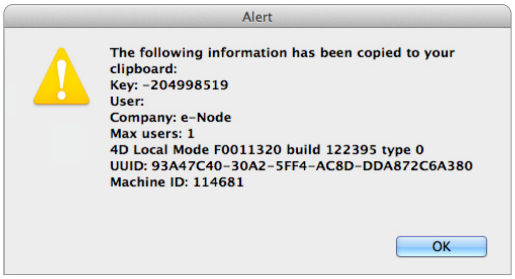
Both methods can be either used with regular or merged servers and client workstations.

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

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



```
The following information has been copied to your
clipboard:
Key: -204998519
User:
Company: e-Node
Max users: 1
4D Local Mode F0011320 build 122395 type 0
UUID: 93A47C40-30A2-5FF4-AC8D-DDA872C6A380
Machine ID: 114681
```

A text file is also saved with the same information.

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

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

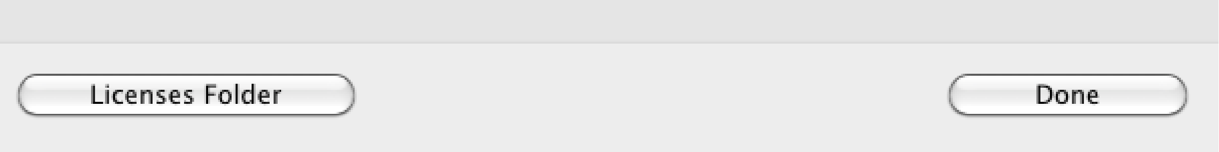
```
C_LONGINT($machineID)
$machineID:= AL_GetAreaLongProperty (0;"mach")
```

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

Using a text file

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

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



The text file **must** be called "ALP9.license4Dplugin" and be a plain text type file.

Just paste all your licenses for AreaList Pro v9.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 "ALP9.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

As of version 9.9, 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 e-Node's license server as described below.

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

Process

If the plugin has not been previously registered (through online 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 e-Node's license server.
2. Ask the server if the master key has not been assigned yet (or if the master key is designed to generate several final keys, if there is any unassigned key up to that number).
3. Send the serial information (regular licenses) or the machine ID (merged licenses) to the license server.
4. If an error is detected (such as master key not matching the current setup) return an error to ***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).

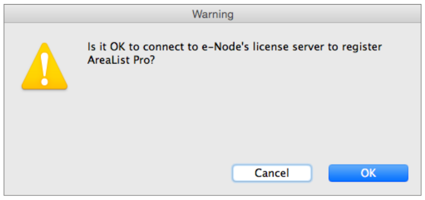
Note: if a final key has already been issued for this serial / machine ID using this master key, it is simply resent.

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
```

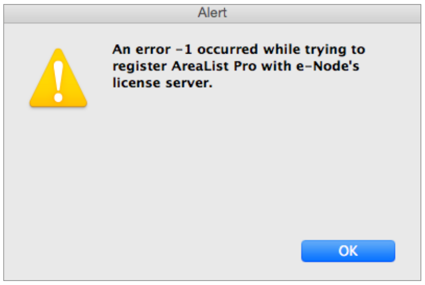
Display a confirmation dialog before step 1



Warning

Is it OK to connect to e-Node's license server to register AreaList Pro?

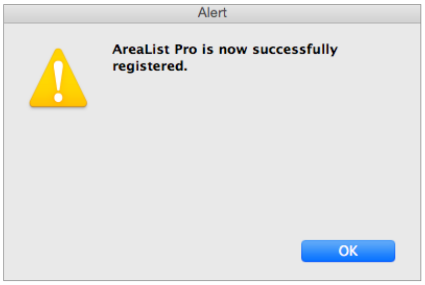
Display an alert at step 4



Alert

An error -1 occurred while trying to register AreaList Pro with e-Node's license server.

Display an alert at step 5



Alert

AreaList Pro is now successfully registered.

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: ALP9 license

  Body:

  > License 123456-123456789-abcdefgh
  >
  > issued to 12.34.56.78

- When a key is resent:

  Title: ALP9 license

  Body:

  > License 123456-123456789-abcdefgh
  >
  > resent to 12.34.56.78

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

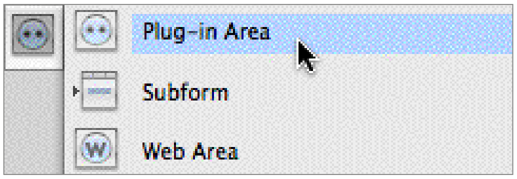
3

Getting Started with 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 **AreaListPro** from the **Type** popup menu. (If the AreaListPro option is not available, please refer to the installing the plugin instructions).
5. Enter a name for your new area in the Variable Name field in the Property List window.
6. Your area will now show the AreaListPro 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.

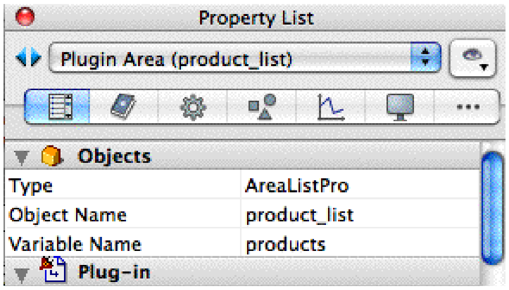
When you do this, the settings specified in the Advanced Properties dialog will be applied when the form is loaded, and then the commands will be applied.

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

We will have a quick overview of the Advanced Properties Dialog here; you'll find a detailed explanation of it later in this manual.

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



Property List

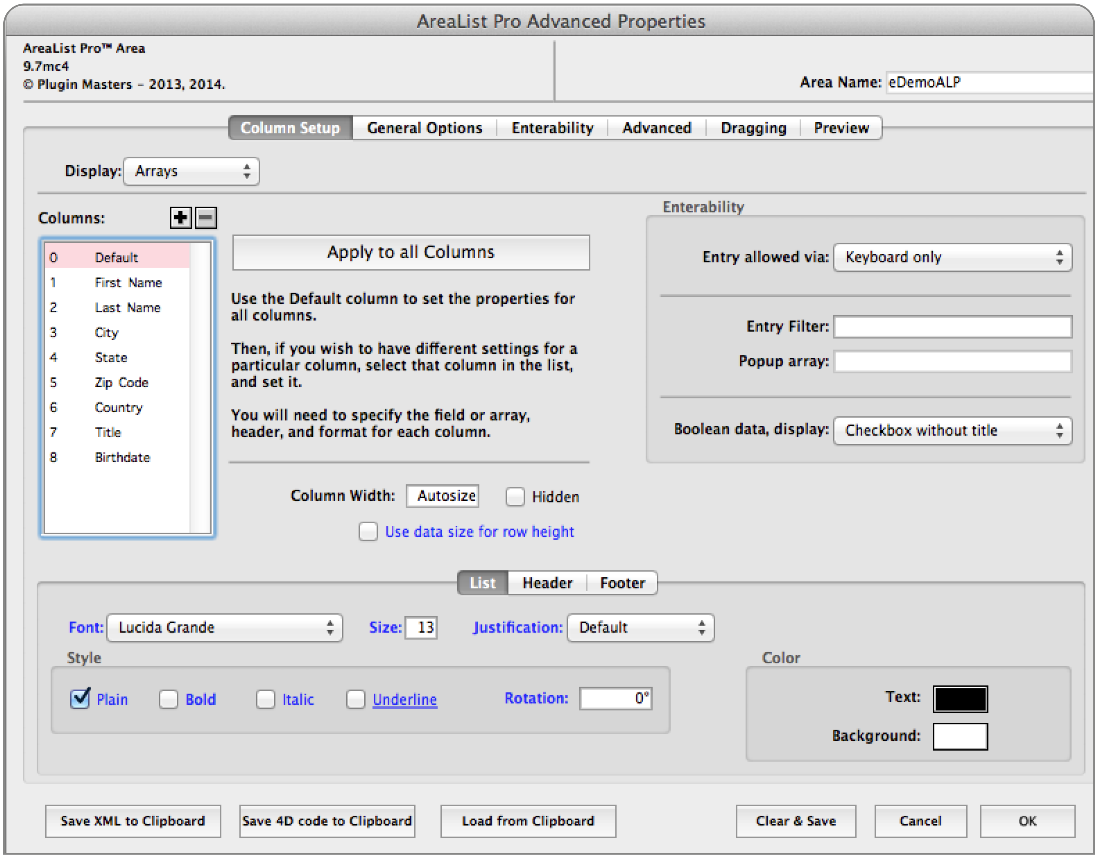
Plugin Area (product_list)

Objects

Type	AreaListPro
Object Name	product_list
Variable Name	products

Plug-in

The Advanced Properties Dialog opens:



AreaList Pro Advanced Properties

AreaList Pro™ Area
9.7mc4
© Plugin Masters - 2013, 2014.

Area Name: eDemoALP

Display: Arrays

Columns:

0	Default
1	First Name
2	Last Name
3	City
4	State
5	Zip Code
6	Country
7	Title
8	Birthdate

Use the Default column to set the properties for all columns.

Then, if you wish to have different settings for a particular column, select that column in the list, and set it.

You will need to specify the field or array, header, and format for each column.

Column Width: Autosize □ Hidden

□ Use data size for row height

Enterability

Entry allowed via: Keyboard only

Entry Filter:

Popup array:

Boolean data, display: Checkbox without title

Font: Lucida Grande Size: 13 Justification: Default

Style

☑ Plain □ Bold □ Italic □ Underline Rotation: 0°

Color

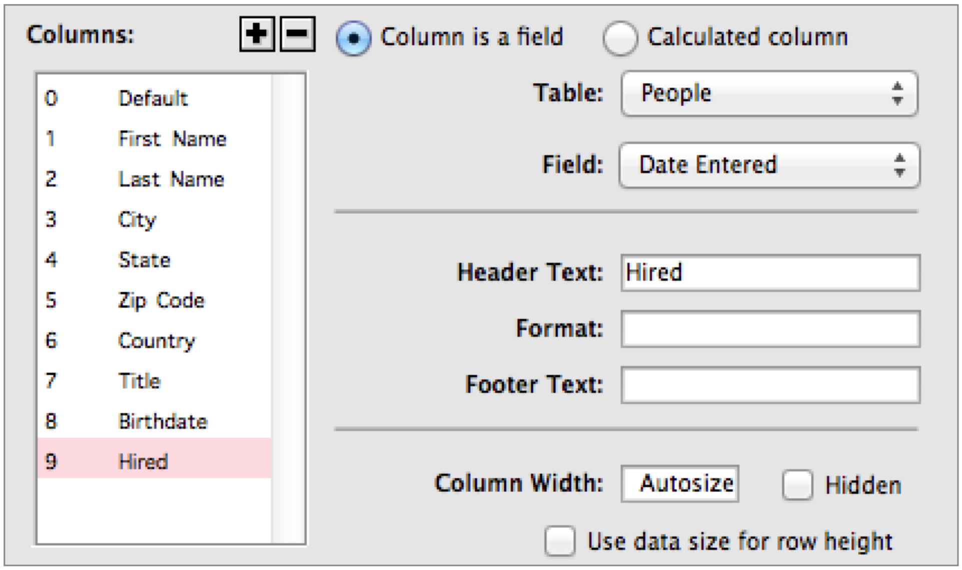
Text: ■

Background: □

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:



Columns: + −

0	Default
1	First Name
2	Last Name
3	City
4	State
5	Zip Code
6	Country
7	Title
8	Birthdate
9	Hired

● Column is a field ○ Calculated column

Table: People

Field: Date Entered

Header Text: Hired

Format:

Footer Text:

Column Width: Autosize □ Hidden

□ Use data size for row height

3. Select the column that you want to assign a field to.
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
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 phase of a layout, 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 phase, 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 phase 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 displayed with the error code, the AreaList Pro command, the calling 4D method and the property selector used.

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.

he 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 true (1) 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 9 is compatible with 4D versions 11 and above.

To upgrade to AreaList Pro version 9, 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

Compatibility Mode

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 9 with little or no change in your code. See the v8.5 manual for legacy commands documentation.

AreaList Pro version 9 will automatically run in v8 compatibility mode and ALP_Area_Compatibility is set to 1 when any of the following conditions are met:

- the area is initialized from the old Advanced Properties
- the area is initialized from XML (Advanced Properties or from code)
- you set the ALP_Area_Compatibility property of ***AL_SetAreaLongProperty*** to 1
- one of the following commands is used:

  AL_SetArraysNam

  AL_InsArraysNam

  AL_SetFields

  AL_InsertFields

  AL_SetRowOpts

  AL_SetColOpts

The developer can switch compatibility mode on or off by setting ALP_Area_Compatibility (1 for on, 0 for off).

Compatibility Mode Behaviour

When running in compatibility mode, the following behaviours are different:

- the ALProEvt variable is created and updated
- visibility of columns is modified according to the number of hidden columns
- the area is made visible on update event
- wrap mode is only set depending on the number of row lines (ALP_XXX_Wrap properties are ignored)
- the area is draggable even if it is not set as draggable in the form properties
- when a row or column is drag and dropped in the same area, it is moved on drag end, not on drop
- headers on Windows 7 are drawn using pictures (eliminates native “white” Win7 headers)
- horizontal scrolling is set to columns (ALP_Area_ScrollColumns = 1)
- auto-selection of an unselected row when clicking a popup icon is disabled (ALP_Area_SelNoAutoSelect = 1)

What’s Changed

Native Look

AreaList Pro version 9 does not support themes as previous versions did: it uses the workstation’s current theme.

For example, to use “classic” Windows theme you must set it in the Windows system settings.

Therefore the appearance is always native (headers, scrollbars, highlight color, checkboxes and entry widget, except when in compatibility mode and Windows 7 - headers are from Vista), only headers can be drawn the legacy v8.x way using the ALP_Area_HeaderMode property.

Note: 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 to ALP_Area_ShowSortIndicator draws the sort (non native) triangle to the right, not on top.

See also Header size and sort indicator in the Troubleshooting / FAQ section.

New API

AreaList Pro version 9 introduces a completely new API, which is based on a full list of properties that the developer can get/set. The previous API is also still available and version 9 is generally compatible with existing version 8 code.

There are now fewer commands that you use to set and get an area’s properties. Each command affects just one property for the area, making your code much easier to understand and debug.

The new 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***

For example, the old AreaList Pro command ***AL_SetCopyOpts*** had four parameters to set copy options for the area:

- Include hidden columns
- Field separator for Edit menu copy
- Record separator for Edit menu copy
- Field wrapper for Edit menu copy

A call to this command would look something like this:

```
AL_SetCopyOpts (area;1;"";"";"")  //include hidden columns in Edit menu Copy
    //use the default Field and Record delimiters for Edit menu Copy; no field wrapper
```

When debugging or modifying the code, it’s difficult to know what each of those parameters means.

In the new API, this would be replaced with four commands, each setting one option.

For example you could use the ***AL_SetAreaPtrProperty*** command:

```
bTrue:=True
tBlank:=""
$err:=AL_SetAreaPtrProperty (area;ALP_Area_CopyHiddenCols;->bTrue)
$err:=AL_SetAreaPtrProperty (area;ALP_Area_CopyFieldSep;->tBlank)
$err:=AL_SetAreaPtrProperty (area;ALP_Area_CopyRecordSep;->tBlank)
$err:=AL_SetAreaPtrProperty (area;ALP_Area_CopyFieldWrapper;->tBlank)
```

Or you could use the ***AL_SetAreaLongProperty*** and ***AL_SetAreaTextProperty*** commands to achieve the same result:

```
AL_SetAreaLongProperty (area;ALP_Area_CopyHiddenCols;1)
AL_SetAreaTextProperty (area;ALP_Area_CopyFieldSep;"")
AL_SetAreaTextProperty (area;ALP_Area_CopyRecordSep;"")
AL_SetAreaTextProperty (area;ALP_Area_CopyFieldWrapper;"")
```

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

Don’t worry though - you will not need to re-write all your AreaList code.

Most of your existing commands will still work; the old commands act as wrappers for the new ones. In fact you will still be able to write new code using the old commands, but if you want to take advantage of the new features, you’ll need to use the new commands.

Some of the old commands are now obsolete or are no longer relevant and should be removed from your code.

These are listed in the table below, along with details about how they should be replaced, where appropriate.

You’ll find a complete list of the old commands and information about how they can be replaced with the new ones in the Mapping Old Commands to the new API chapter, and a complete description of the new commands in the Command Reference section.

You can find a description of the new syntax in the Anatomy of an AreaList Pro Command section.

Controls for Booleans

In previous versions of AreaList Pro, if you wanted to display checkboxes for Boolean values, you had to use a picture.

Now you can set the display and data entry controls using the ALP_Column_DisplayControl and ALP_Column_EntryControl properties of ***AL_SetColumnLongProperty***.

You can also use custom pictures with these properties.

Drag and Drop

In previous versions, AreaList Pro used its own drag and drop manager. It now uses 4D's drag and drop manager.

Also, 4D until version 11 executed the drop in source process context, since version 11 it is executed in destination process context.

This means that you will need to make a few changes to your drag and drop handling.

For more information, see the Drag and Drop chapter for a complete explanation.

Event Handling

If a drag is initiated, AreaList Pro receives a drag event. But this has nothing to do with the actual drop: it could end anywhere (or nowhere if the user pressed Esc - no drop). 4D does not inform AreaList Pro that the drag was not successful.

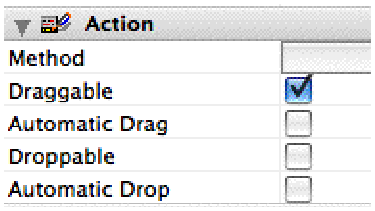
When a drop has occurred, an On Drop event is fired. ***AL_GetAreaLongProperty*** with the ALP_Area_AlpEvent area property will return AL Row drop event, AL Column drop event, AL Cell drop event or AL Object drop event in the destination context.

Accepting Drops from non-AreaList Pro objects

AreaList Pro accepts drags from and to non-AreaList Pro objects such as other 4D objects (in the same process or a different one) and external files.

Object Properties

For an AreaList Pro area that you want to drag or drop to/from, select the appropriate properties in the object's Property List window (note that these properties are ignored in compatibility mode):



Action
Method
Draggable ☑
Automatic Drag □
Droppable □
Automatic Drop □

Automatic Scrolling

When the user drags something to an AreaList Pro area, the area's contents will start to scroll (if necessary) when the object gets near the AreaList Pro border. In AreaList Pro 8.5 this scrolling area was outside the AreaList Pro area; in AreaList Pro version 9 it is inside AreaList Pro area. The default frame size if 30 points; you can change this with the ALP_Area_DragScroll property of ***ALP_SetAreaLongProperty***.

Events

AL_GetLastEvent was added in AreaList Pro version 8.5; it returns the value of ALPEvt for a given area (or for all areas). In version 9, this command will only work if you are using AreaList Pro in compatibility mode.

If you are not using it in compatibility mode, and for all future development, you should use the ALP_Area_AlpEvent parameter of ***AL_GetAreaLongProperty*** to find out what the last event was.

Obsolete Commands

If you are using any of the following commands, you will need to remove them from your code:

Old Command	How to replace it
AL_DoWinResize	Obsolete.
AL_DragMgrAvail	No longer relevant; always true.
AL_GetAdvProps	Unsupported.
AL_GetHeaderOptions	Unsupported.
AL_GetSortEditorParams	Use the ALP_Area_SortTitle property in the ALP Area theme to get the sort editor title and ALP_Area_SortPrompt to set the prompt. The other options are not supported. See the Sorting topic for more information about sorting.
AL_InsertArrays	Replace with AL_AddColumn using a pointer to the array.
AL_InsertFields	Replace with AL_AddColumn using a pointer to the field.
AL_SetArrays	Replace with AL_AddColumn using a pointer to the array.
AL_SetCellFrame	No equivalent in the new API, but the old command will continue to work as before. The call sets properties ALP_Cell_XXXBorderOffset, ALP_Cell_XXXBorderWidth and ALP_Cell_XXXBorderColor (where XXX is Top, Left, Bottom, Right) for all the cells in a rectangle. It is the same as calling AL_SetCellBorder in a loop.
AL_SetHeaderIcon	Unsupported.
AL_SetHeaderOptions	Unsupported.
AL_SetPictEscape	Unsupported.
AL_SetSortEditorParams	Use the ALP_Area_SortTitle property in the ALP Area theme to set the sort editor title. The other options are not supported. See the Sorting topic for more information about sorting.
AL_SetSubSelect	Obsolete.
AL_SetWinLimits	Obsolete: delete it.

Picture Escape Codes

Picture escape codes were mostly used to display controls (checkboxes) when Boolean fields/arrays were displayed. However, AreaList Pro version 9 has the ALP_Column_DisplayControl property and can display native controls as well as custom pictures in the list, so it makes this use of picture escape codes obsolete.

Pictures

The handling of pictures has changed in AreaList Pro version 9:

- Icons, PICTs and cicn from the resource fork are not supported
- The old call ***AL_SetHeaderIcon*** is not supported
- Picture escape codes are not supported
- ***AL_SetCellIcon*** now only supports pictures from the Picture Library. 'cicn' and 'PICT' resources are no longer supported. There is a new mechanism for displaying icons (images) either in a list or in headers.

The new properties are:

- ALP_Cell_LeftIconID
- ALP_Cell_LeftIconFlags
- ALP_Cell_RightIconID
- ALP_Cell_RightIconFlags

For more information about using these properties, please see the Pictures section.

Registering AreaList Pro

The AL_Register command takes just one parameter, and it returns 0 for OK and an integer between 1 and 12 if not OK. Mutliple calls are allowed. There is a list of the registration error codes and their meanings here.

Spelling Checker

The spelling check function is no longer supported by 4D for plugin calls, so this option is not available.

What's New

Caching of Formatted Values

AreaList Pro version 9 caches formatted displayed values. When AreaList Pro redraws the screen it does not need to access 4D.

Column Hiding

In previous versions, you could only specify which columns to hide at the end of the list of columns. Now you can use the ALP_Column_Visible property of ***AL_SetColumnLongProperty*** to hide individual columns by making them invisible, and use ***AL_GetColumnLongProperty*** to find out which columns are invisible.

Individual columns can also be made invisible by selecting the **Hidden** option on the first page of the Advanced Properties dialog.

DisplayList

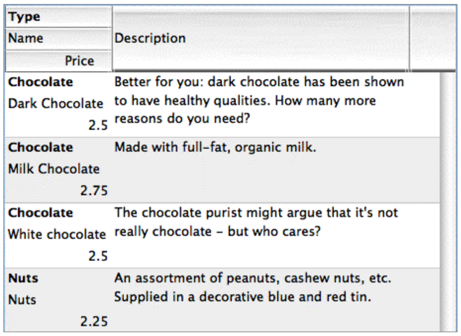
The DisplayList plugin is now included with AreaList Pro for backwards compatibility. You can find more information in the DisplayList chapter.

Dynamic Row Height

Row height can now be set to dynamically auto-size depending upon the contents and styling of the data in the row.

Grid

In previous versions, a row displayed the arrays (columns) in the defined order, and arrays at the end could be hidden. In AreaList Pro version 9, the columns can also be displayed in a table format in which you control how they are arranged within the AreaList Pro area. For example:



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.

In this example, there are four fields (Type, Name, Price and Description) displayed in two columns, with the data in the second column (the Description) spanning three rows - like an HTML table.

See the Grid section for more information and instructions on how to use this feature.

Hierarchical lists

AreaList Pro version 9 is capable of displaying a Finder-like hierarchical list. This is implemented by setting a level (offset to the right) and state (collapsed or expanded) for each row.

For more details on this feature, see the Hierarchical Lists topic.

Multi-styled text

AreaList Pro supports the multi-styled text feature of 4D v12. When 4D passes multi-styled text to AreaList Pro, it should be displayed correctly if the ALP_Column_Attributed option has been set.

If this option is set, special tags can also be used in any text contained in an AreaList Pro area to display styled characters.

See AreaList Pro Text Style Tags.

Native drawing of text

AreaList Pro uses CoreText on Mac and GDI+ on Windows.

Only fonts and font faces supported by these technologies can be used in AreaList Pro. In particular, GDI+ does not support non-TrueType fonts on some Windows versions.

Text styling

Many options are available to set text styles in headers, rows, footers or individual cells.

See the Text Styling topic for more details

Transparency

You can now specify transparency in RGB colors.

See the Working with Colors topic for more info.

Unicode

AreaList Pro supports Unicode for display and data entry. Note that entry filters do not support Unicode, so their functionality is limited to ASCII (original 7-bit) characters. In other words, if a filter is specified, only “printable” Unicode characters with code < 256 can be entered.

Value Mapping

Now you can map values from fields or arrays to defined parameters in a popup menu.

For example, you can map numeric values in a field to text equivalents. For more details, see Value Mapping in the Advanced Topics chapter.

Wrapped Text

Text wrapping is now supported, using the ALP_XXX_Wrap properties of the ***AL_SetXXXLongProperty*** commands. See Text wrapping.

XML

An area’s settings can be saved as XML into a variable or field. Note that if 4D is run in non-Unicode mode, the size of any text is restricted to 32k characters.

This should normally be enough, however it is recommended that you use the application in Unicode mode.

For more information, see the XML section.

Column Automatic Resize

The ALP_Area_AutoResizeColumn and ALP_Area_AutoSnapLastColumn properties can be used to have AreaList Pro automatically resize any given column in an AreaList Pro area defined as horizontally growable in the 4D form editor, or simply because you want a specified column to resize and make the last one snap to the right edge of the area without having to calculate its width.

These properties are also useful when converting AreaList Pro areas from previous versions will slightly modify the area size, including width. This is due to the way the frames are handled in AreaList Pro v9.

- previous versions drew the area frame outside of the plugin area (which will never work in composite windows, for example)
- AreaList Pro v9 always draws inside the plugin area, so the usable area is smaller (2 points for plain box, 4 points for 3D frame)
- AreaList Pro v9 uses native scrollbar size (15 on Mac, previous versions used 16 points) - if the user sets the scrollbar size to 20 (on Windows), it will be used
- AreaList Pro v9 does not draw separator line between list and vertical scrollbar

For example, on MacOS (current scrollbar size is 15):

- for no frame: the list is expanded by 2 points (relative to AreaList Pro 8.5)
- for single line frame: the list size is the same
- for 3D frame (sunken): the list size is shrunken by 2 points

In addition, on Windows not only the scrollbars are bigger, but the focus is drawn inside the AreaList Pro area. Therefore the usable area is smaller by another 2 points. If set, the last visible column will be resized (ALP_Column_Width) to match the area size if there is enough space left.

When initializing an AreaList Pro area from previous versions advanced properties, if all visible columns have a non-zero (fixed, not automatic) width and the sum of all column widths is equal to the area width minus 1 (minus 16 if the vertical scrollbar is visible) and the last column's width after adjustment is at least 5 points, this property will be set to true.

Row hiding

The ALP_Row_Hide and ALP_Object_RowHide properties allow hiding of individual rows.

Calculated columns in array mode

Calculated columns are available in both field and array display modes. See Calculated Columns.

4 Tutorial

The following examples illustrate how to use AreaList Pro.

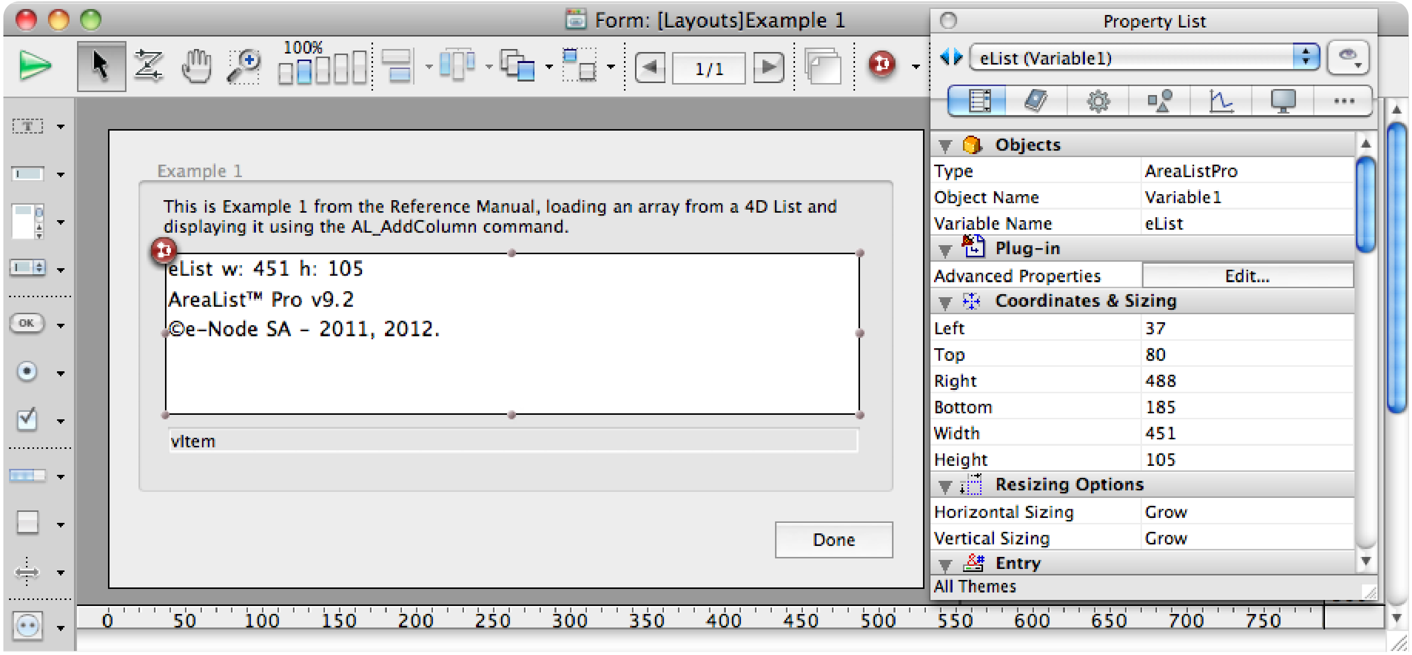
You can find them all in the Examples database.

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:



This is Example 1 from the Reference Manual, loading an array from a 4D List and displaying it using the AL_AddColumn command.

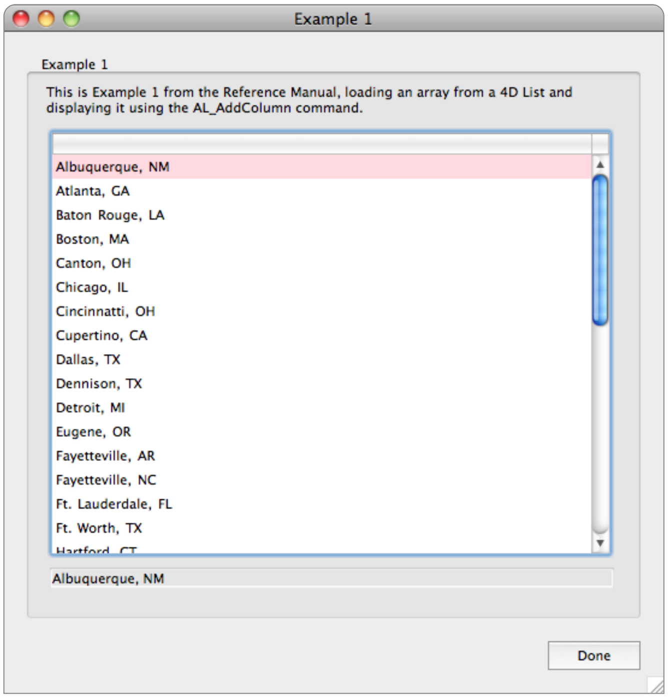
eList w: 451 h: 105
AreaList™ Pro v9.2
©e-Node SA - 2011, 2012.

Objects	
Type	AreaListPro
Object Name	Variable1
Variable Name	eList
Plug-in	
Advanced Properties	Edit...
Coordinates & Sizing	
Left	37
Top	80
Right	488
Bottom	185
Width	451
Height	105
Resizing Options	
Horizontal Sizing	Grow
Vertical Sizing	Grow
Entry	
All Themes	

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

```
Case of
: (Form event=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
    DEMO_Default (eList)  //general display settings
    AL_SetAreaLongProperty (eList;ALP_Area_SelRow;1)  //row 1 selected
    vItem:=aCityState{1}
: (Form event=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 selected
          // OR
        $row:=AL_GetAreaLongProperty (eList;ALP_Area_ClickedRow)  //row the user clicked
        vItem:=aCityState{$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:



This is Example 1 from the Reference Manual, loading an array from a 4D List and displaying it using the AL_AddColumn command.

Albuquerque, NM
Atlanta, GA
Baton Rouge, LA
Boston, MA
Canton, OH
Chicago, IL
Cincinnatti, OH
Cupertino, CA
Dallas, TX
Dennison, TX
Detroit, MI
Eugene, OR
Fayetteville, AR
Fayetteville, NC
Ft. Lauderdale, FL
Ft. Worth, TX
Hartford, CT

Albuquerque, NM

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=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:

```
$AL_Area:=$1  //$1 is received as the area reference
AL_SetAreaLongProperty ($AL_Area;ALP_Area_HideHeaders;0)  //Header will be displayed
AL_SetAreaLongProperty ($AL_Area;ALP_Area_ShowFooters;0)  //Hide footer
If (IsWindows =1)  //The IsWindows project method returns True if the platform is Windows
    //Set header properties
   AL_SetColumnTextProperty ($AL_Area;0;ALP_Column_HdrFontName;"Tahoma")
   AL_SetColumnLongProperty ($AL_Area;0;ALP_Column_HdrSize;11)  //Font size
   AL_SetColumnLongProperty ($AL_Area;0;ALP_Column_HdrStyleB;1)  //Bold
    //Set column properties
   AL_SetColumnTextProperty ($AL_Area;0;ALP_Column_FontName;"Tahoma")
   AL_SetColumnLongProperty ($AL_Area;0;ALP_Column_Size;11)
   AL_SetColumnLongProperty ($AL_Area;0;ALP_Column_StyleB;0)
Else  //MacOS
    //Set header properties
   AL_SetColumnTextProperty ($AL_Area;0;ALP_Column_HdrFontName;"Lucida Grande")
   AL_SetColumnLongProperty ($AL_Area;0;ALP_Column_HdrSize;11)
   AL_SetColumnLongProperty ($AL_Area;0;ALP_Column_HdrStyleB;1)
    //Set column properties
   AL_SetColumnTextProperty ($AL_Area;0;ALP_Column_FontName;"Lucida Grande")
   AL_SetColumnLongProperty ($AL_Area;0;ALP_Column_Size;11)
   AL_SetColumnLongProperty ($AL_Area;0;ALP_Column_StyleB;0)
End if
 //Set the area properties
AL_SetAreaLongProperty ($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
```

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:

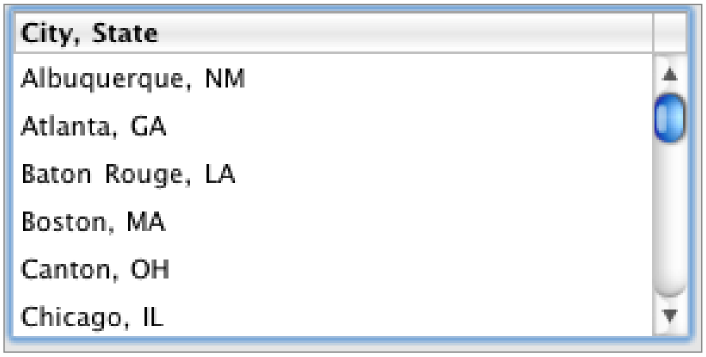
```
Case of
  : (Form event=On Plug in Area)  // respond to user action
    If (AL_GetAreaLongProperty (eList;ALP_Area_AlpEvent)=AL Single click event)
       // single-click on a row (or up / down arrow keys)
      $row:=AL_GetAreaLongProperty (eList;ALP_Area_SelRow)  // get the selected row
      vItem:=aCityState{$row}  // get the value in that element of the array
      End if
```

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:



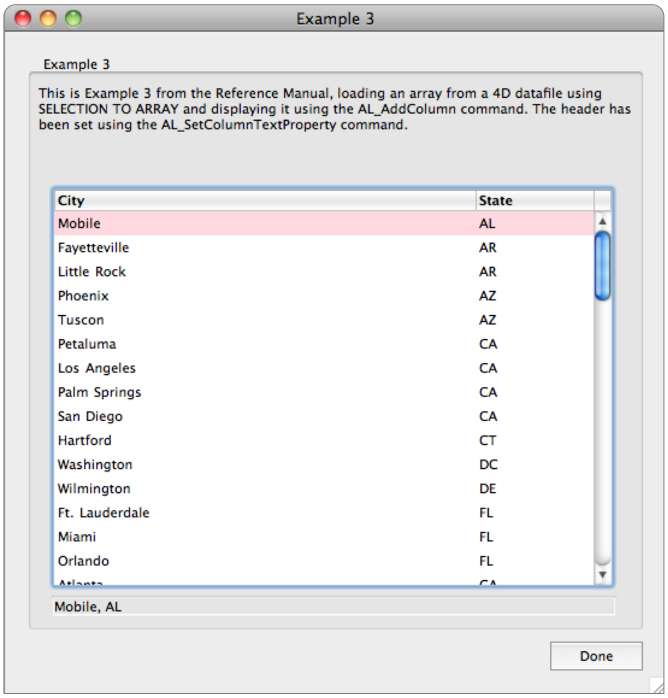
City, State
Albuquerque, NM
Atlanta, GA
Baton Rouge, LA
Boston, MA
Canton, OH
Chicago, IL

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:



This is Example 3 from the Reference Manual, loading an array from a 4D datafile using SELECTION TO ARRAY and displaying it using the AL_AddColumn command. The header has been set using the AL_SetColumnTextProperty command.

City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ
Petaluma	CA
Los Angeles	CA
Palm Springs	CA
San Diego	CA
Hartford	CT
Washington	DC
Wilmington	DE
Ft. Lauderdale	FL
Miami	FL
Orlando	FL
Atlanta	GA

Mobile, AL

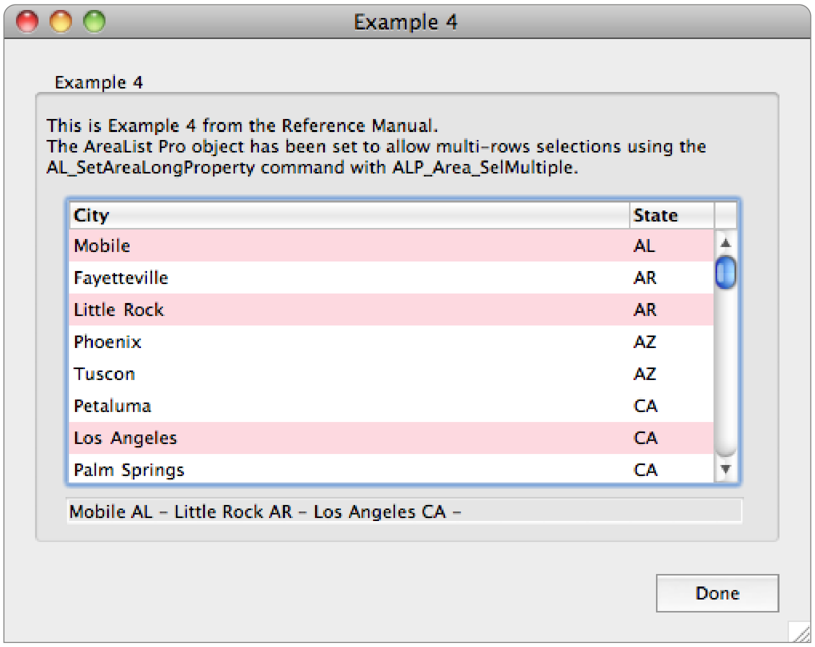
```
ALL RECORDS([Cities])  //load all records in the Cities table
SELECTION TO ARRAY([Cities]City;aCity;[Cities]State;aState)  //copy field values into arrays
$error:=AL_AddColumn (eList;->aCity)  //display array in AreaList Pro object
$error:=AL_AddColumn (eList;->aState)  //display array in AreaList 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:



This is Example 4 from the Reference Manual.
The AreaList Pro object has been set to allow multi-rows selections using the AL_SetAreaLongProperty command with ALP_Area_SelMultiple.

City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ
Petaluma	CA
Los Angeles	CA
Palm Springs	CA

Mobile AL – Little Rock AR – Los Angeles CA –

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
```

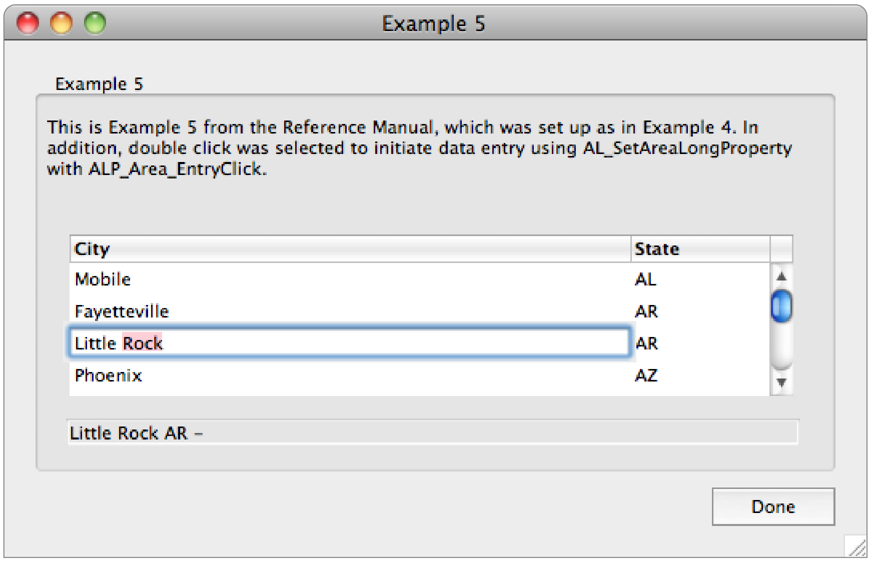
With this option selected, the user can Ctrl-click (Windows) or Cmd-click (Mac) to select multiple rows.

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=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.



This is Example 5 from the Reference Manual, which was set up as in Example 4. In addition, double click was selected to initiate data entry using AL_SetAreaLongProperty with ALP_Area_EntryClick.

City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ

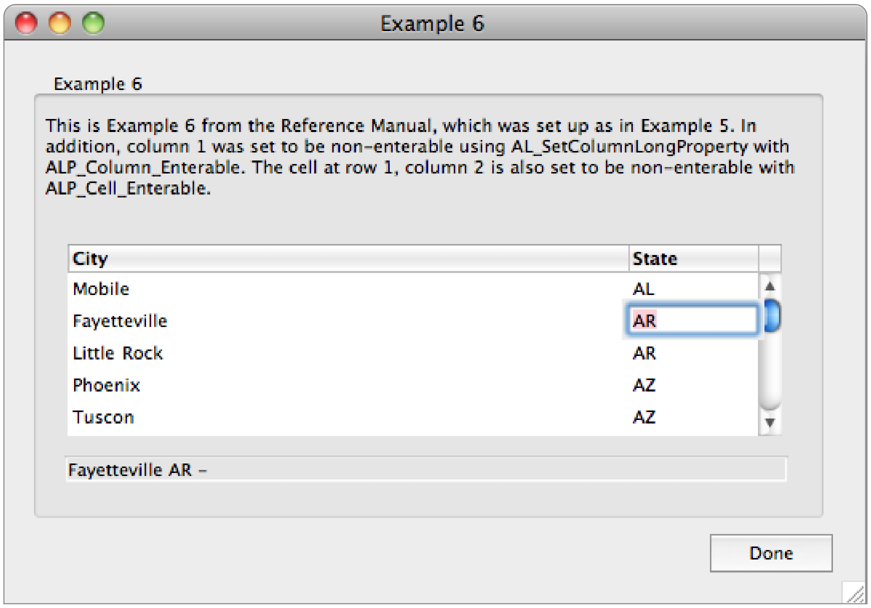
Little Rock AR –

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 is Example 6 from the Reference Manual, which was set up as in Example 5. In addition, column 1 was set to be non-enterable using AL_SetColumnLongProperty with ALP_Column_Enterable. The cell at row 1, column 2 is also set to be non-enterable with ALP_Cell_Enterable.

City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ

Fayetteville AR –

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)

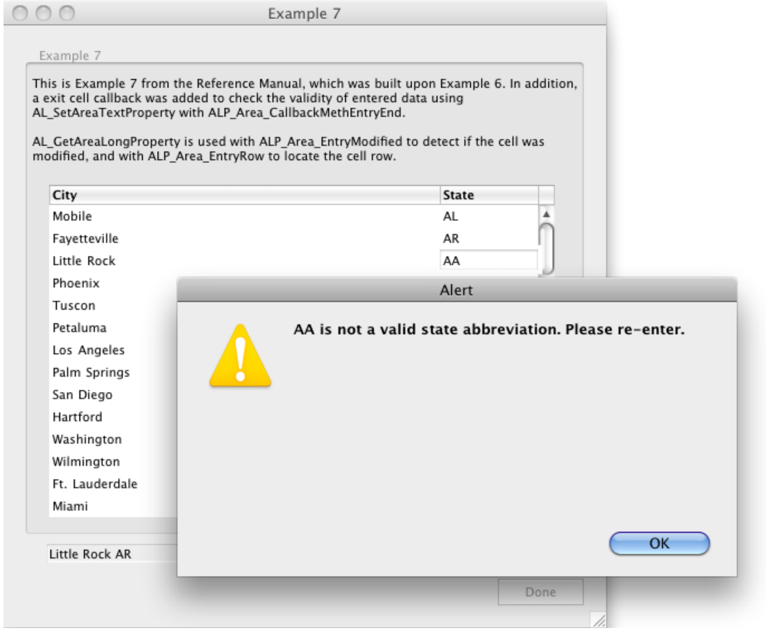
//set the cell at row 1, column 2 to be non-enterable

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 section 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:



This is Example 7 from the Reference Manual, which was built upon Example 6. In addition, a exit cell callback was added to check the validity of entered data using AL_SetAreaTextProperty with ALP_Area_CallbackMethEntryEnd.

AL_GetAreaLongProperty is used with ALP_Area_EntryModified to detect if the cell was modified, and with ALP_Area_EntryRow to locate the cell row.

City	State
Mobile	AL
Fayetteville	AR
Little Rock	AA
Phoenix	
Tuscon	
Petaluma	
Los Angeles	
Palm Springs	
San Diego	
Hartford	
Washington	
Wilmington	
Ft. Lauderdale	
Miami	

Little Rock AR

AA is not a valid state abbreviation. Please re-enter.

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")
```

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")
```

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=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
```

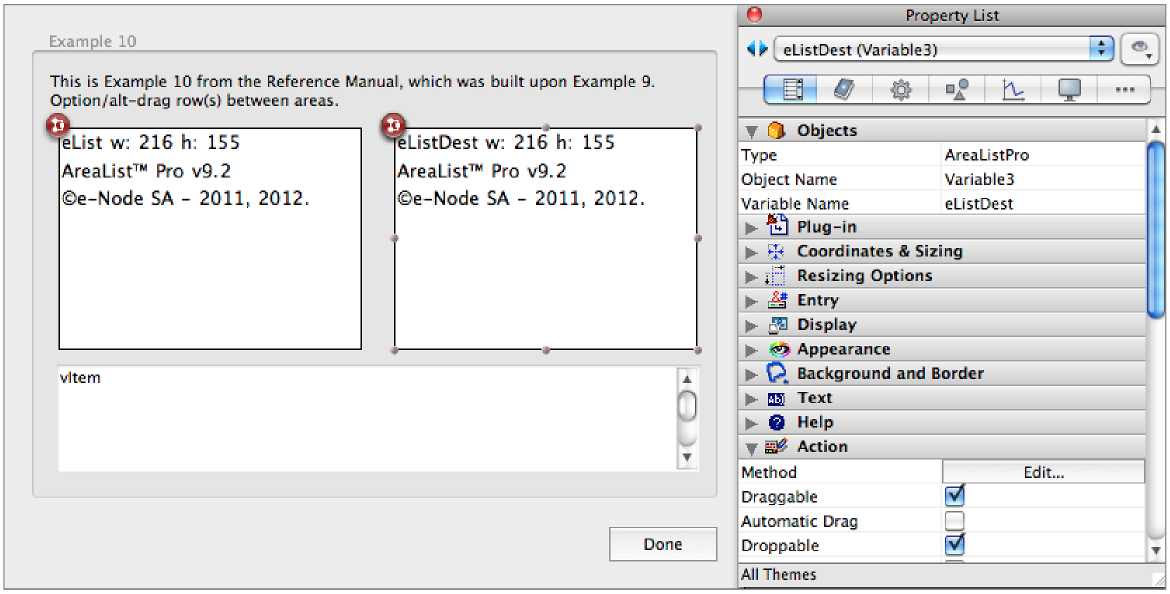
The ***evtUpdateText*** method updates the variable at the bottom of the list:

```
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:



Example 10

This is Example 10 from the Reference Manual, which was built upon Example 9.
Option/alt-drag row(s) between areas.

eList w: 216 h: 155
AreaList™ Pro v9.2
©e-Node SA - 2011, 2012.

eListDest w: 216 h: 155
AreaList™ Pro v9.2
©e-Node SA - 2011, 2012.

vItem

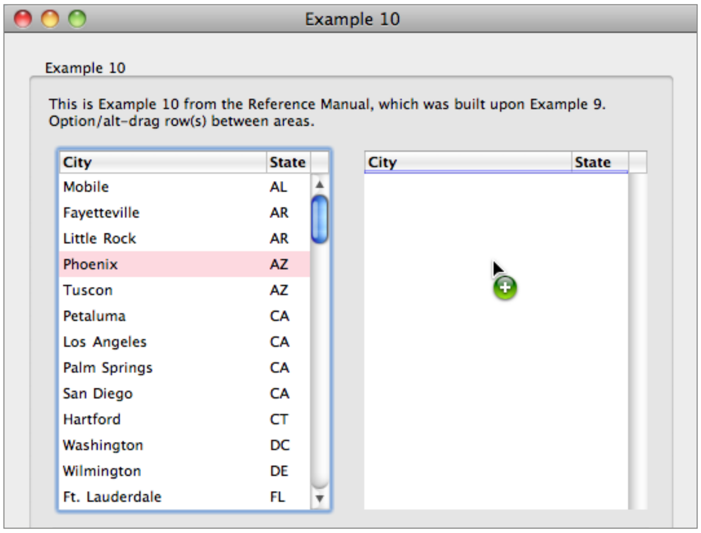
Property List — eListDest (Variable3)

Type	AreaListPro
Object Name	Variable3
Variable Name	eListDest
Method	Edit...
Draggable	☑
Automatic Drag	☐
Droppable	☑

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:



Example 10

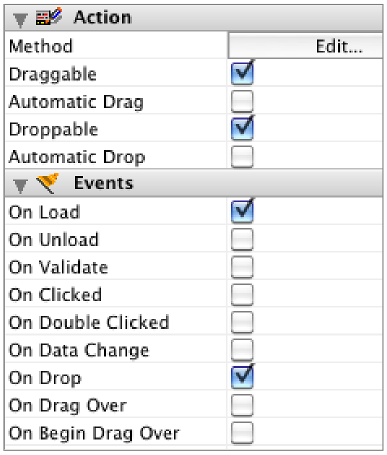
This is Example 10 from the Reference Manual, which was built upon Example 9.
Option/alt-drag row(s) between areas.

City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ
Petaluma	CA
Los Angeles	CA
Palm Springs	CA
San Diego	CA
Hartford	CT
Washington	DC
Wilmington	DE
Ft. Lauderdale	FL

City	State

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.

To control this, the Draggable and Droppable actions are checked in both areas, as well as On Load and On Drop events (with AreaList Pro v9.x the drag and drop is entirely managed in the destination area object method):



Action
Method: Edit...
Draggable ☑
Automatic Drag □
Droppable ☑
Automatic Drop □

Events
On Load ☑
On Unload □
On Validate □
On Clicked □
On Double Clicked □
On Data Change □
On Drop ☑
On Drag Over □
On Begin Drag Over □

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

Left area:

```
Case of
   : (Form event=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=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=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=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
```

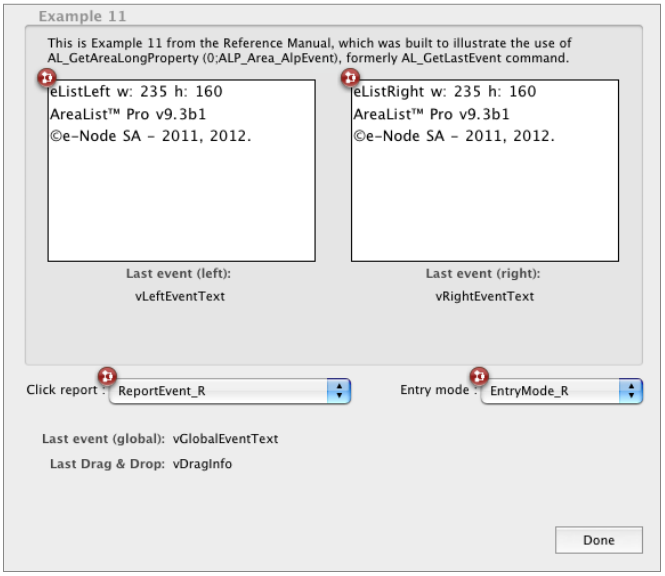
The ***evtRowsDragged*** project method both areas after a row(s) drag and drop between areas:

```
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:



Example 11

This is Example 11 from the Reference Manual, which was built to illustrate the use of AL_GetAreaLongProperty (0;ALP_Area_AlpEvent), formerly AL_GetLastEvent command.

eListLeft w: 235 h: 160
AreaList™ Pro v9.3b1
©e-Node SA - 2011, 2012.

eListRight w: 235 h: 160
AreaList™ Pro v9.3b1
©e-Node SA - 2011, 2012.

Last event (left): vLeftEventText

Last event (right): vRightEventText

Click report : ReportEvent_R

Entry mode : EntryMode_R

Last event (global): vGlobalEventText

Last Drag & Drop: vDragInfo

The user’s actions are monitored through the user 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=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=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=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=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.

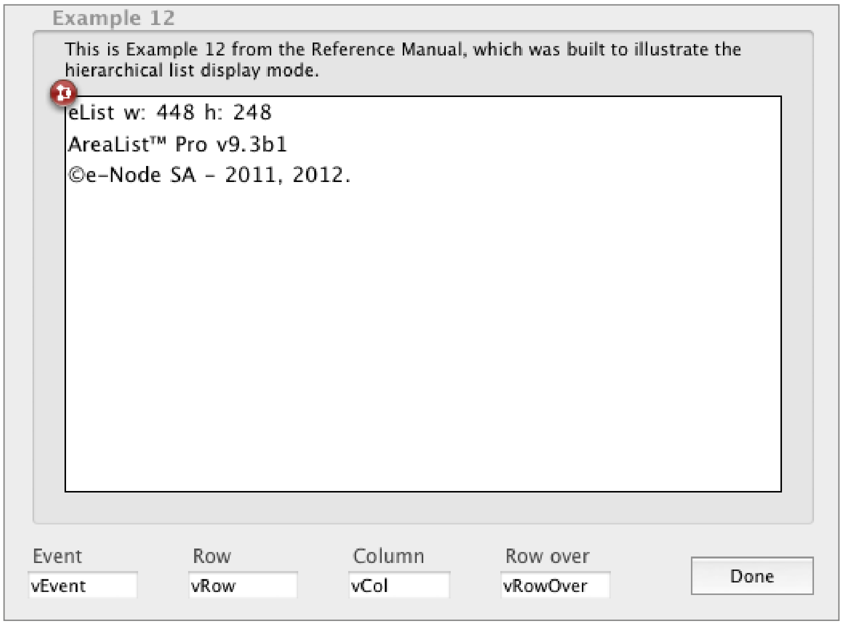
The Event callback method ***EventCallBack11*** is a slight variation from ***EventCallBack10***, including a text variable for each area event, using the received $2 parameter and the same ***AlpEventText*** project method as above:

```
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:



Example 12

This is Example 12 from the Reference Manual, which was built to illustrate the hierarchical list display mode.

```
eList w: 448 h: 248
AreaList™ Pro v9.3b1
©e-Node SA - 2011, 2012.
```

Event	Row	Column	Row over
vEvent	vRow	vCol	vRowOver

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")
```

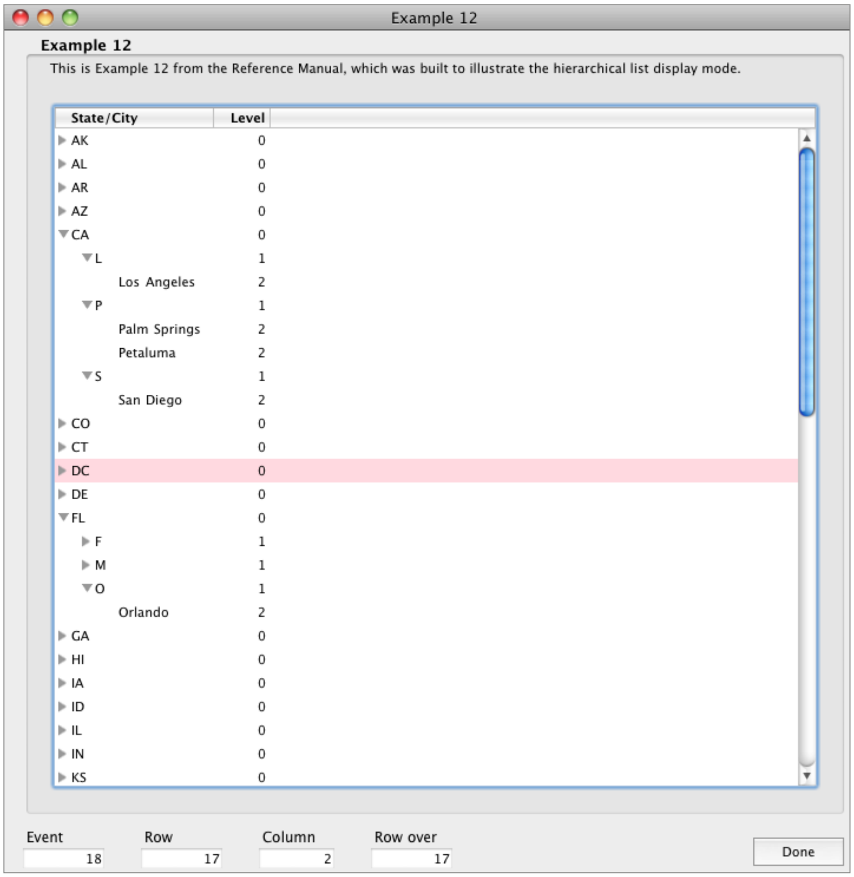
Add some variables to the ***EventCallBack*** method:

```
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:



Example 12

This is Example 12 from the Reference Manual, which was built to illustrate the hierarchical list display mode.

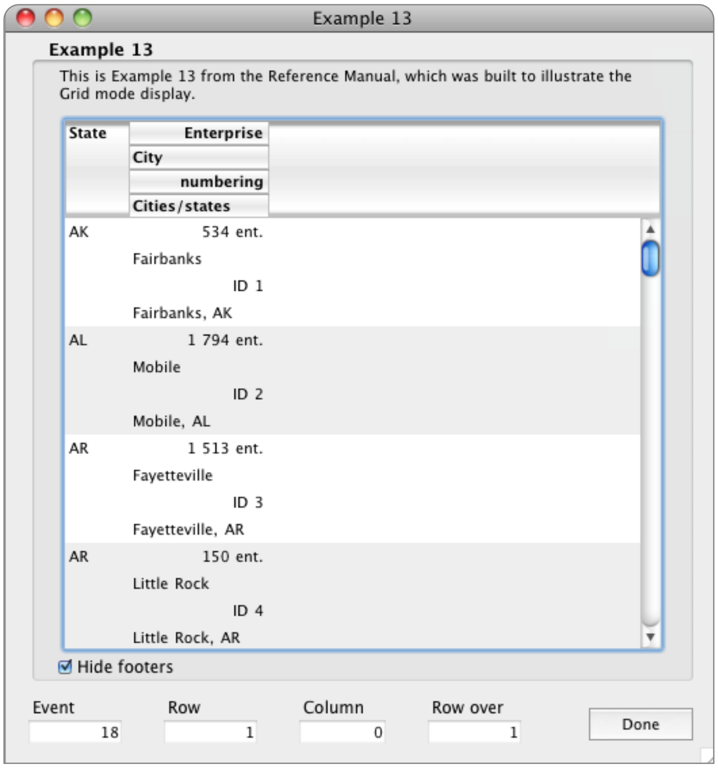
State/City	Level
▶ AK	0
▶ AL	0
▶ AR	0
▶ AZ	0
▼ CA	0
▼ L	1
Los Angeles	2
▼ P	1
Palm Springs	2
Petaluma	2
▼ S	1
San Diego	2
▶ CO	0
▶ CT	0
▶ DC	0
▶ DE	0
▼ FL	0
▶ F	1
▶ M	1
▼ O	1
Orlando	2
▶ GA	0
▶ HI	0
▶ IA	0
▶ ID	0
▶ IL	0
▶ IN	0
▶ KS	0

Event	Row	Column	Row over
18	17	2	17

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:



Example 13

Example 13

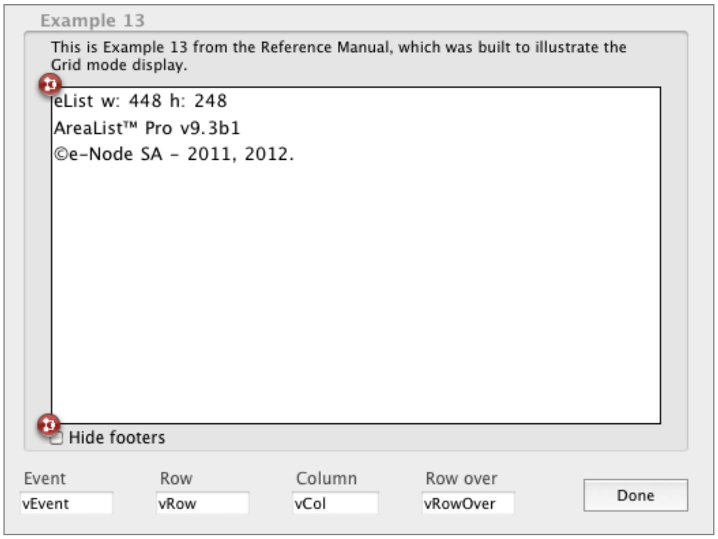
This is Example 13 from the Reference Manual, which was built to illustrate the Grid mode display.

State	
	Enterprise
	City
	numbering
	Cities/states
AK	534 ent.
	Fairbanks
	ID 1
	Fairbanks, AK
AL	1 794 ent.
	Mobile
	ID 2
	Mobile, AL
AR	1 513 ent.
	Fayetteville
	ID 3
	Fayetteville, AR
AR	150 ent.
	Little Rock
	ID 4
	Little Rock, AR

☑ Hide footers

Event	Row	Column	Row over
18	1	0	1

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



Example 13

This is Example 13 from the Reference Manual, which was built to illustrate the Grid mode display.

eList w: 448 h: 248
AreaList™ Pro v9.3b1
©e-Node SA - 2011, 2012.

□ Hide footers

Event	Row	Column	Row over
vEvent	vRow	vCol	vRowOver

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.

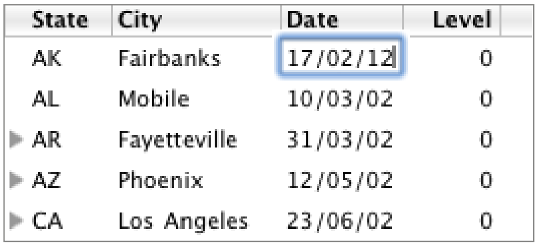
Example 14: Date Formatting Options

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:



State	City	Date	Level
AK	Fairbanks	17/02/12	0
AL	Mobile	10/03/02	0
AR	Fayetteville	31/03/02	0
AZ	Phoenix	12/05/02	0
CA	Los Angeles	23/06/02	0

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")
```

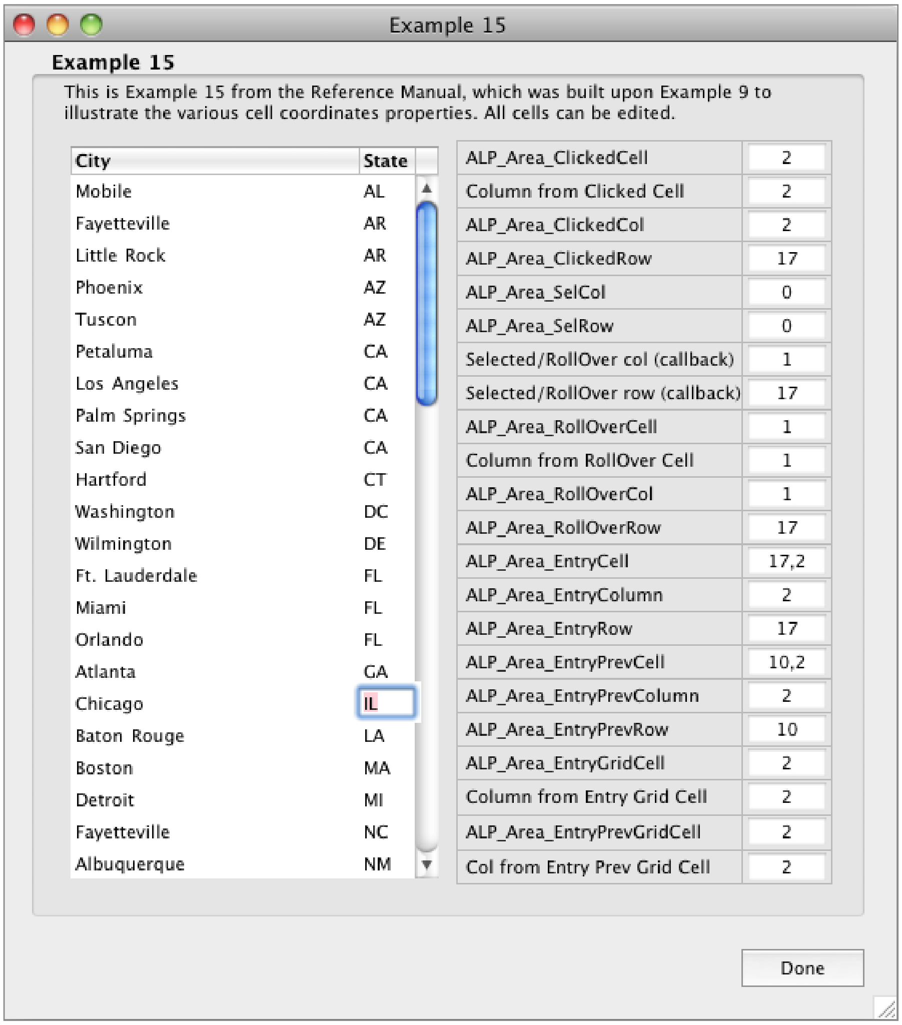
The ***ExitCallbackDate*** method will be invoked when data entry in a cell in the Date column ends.

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:



Example 15

This is Example 15 from the Reference Manual, which was built upon Example 9 to illustrate the various cell coordinates properties. All cells can be edited.

City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ
Petaluma	CA
Los Angeles	CA
Palm Springs	CA
San Diego	CA
Hartford	CT
Washington	DC
Wilmington	DE
Ft. Lauderdale	FL
Miami	FL
Orlando	FL
Atlanta	GA
Chicago	IL
Baton Rouge	LA
Boston	MA
Detroit	MI
Fayetteville	NC
Albuquerque	NM

ALP_Area_ClickedCell	2
Column from Clicked Cell	2
ALP_Area_ClickedCol	2
ALP_Area_ClickedRow	17
ALP_Area_SelCol	0
ALP_Area_SelRow	0
Selected/RollOver col (callback)	1
Selected/RollOver row (callback)	17
ALP_Area_RollOverCell	1
Column from RollOver Cell	1
ALP_Area_RollOverCol	1
ALP_Area_RollOverRow	17
ALP_Area_EntryCell	17,2
ALP_Area_EntryColumn	2
ALP_Area_EntryRow	17
ALP_Area_EntryPrevCell	10,2
ALP_Area_EntryPrevColumn	2
ALP_Area_EntryPrevRow	10
ALP_Area_EntryGridCell	2
Column from Entry Grid Cell	2
ALP_Area_EntryPrevGridCell	2
Col from Entry Prev Grid Cell	2

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:

```
v15EntryCell:=AL_GetAreaTextProperty (eList;ALP_Area_EntryCell)
v15EntryColumn:=AL_GetAreaLongProperty (eList;ALP_Area_EntryColumn)
v15EntryGridCell:=AL_GetAreaLongProperty (eList;ALP_Area_EntryGridCell)
v15EntryGridCellColFromCell:=AL_GetColumnLongProperty (eList;v15EntryGridCell;ALP_Column_FromCell)
v1EntryPrevCell:=AL_GetAreaTextProperty (eList;ALP_Area_EntryPrevCell)
v15EntryPrevColumn:=AL_GetAreaLongProperty (eList;ALP_Area_EntryPrevColumn)
v15EntryPrevGridCell:=AL_GetAreaLongProperty (eList;ALP_Area_EntryPrevGridCell)
v15EntryPrevGridCellColFromCell:=AL_GetColumnLongProperty (eList;v15EntryPrevGridCell;ALP_Column_FromCell)
v15EntryPrevRow:=AL_GetAreaLongProperty (eList;ALP_Area_EntryPrevRow)
v15EntryRow:=AL_GetAreaLongProperty (eList;ALP_Area_EntryRow)
CALL PROCESS(Frontmost process)  //update displayed variables
```

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

```
v15SelColCallback:=$4
v15SelRowCallback:=$5
Example15UpdateVariables
```

The ***EntryCallback15*** 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)
CALL PROCESS(Frontmost process)  //update displayed variables
```

5

Programming the AreaList Pro User Interface

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***:

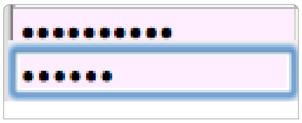
- ALP_Column_DisplayControl must be set to 0, 1, 2 or 4
- ALP_Column_EntryControl is set to 1

Note: you can use pictures to display your own checkboxes, including three-states.
See Displaying custom checkboxes using pictures from the 4D Picture Library.

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:

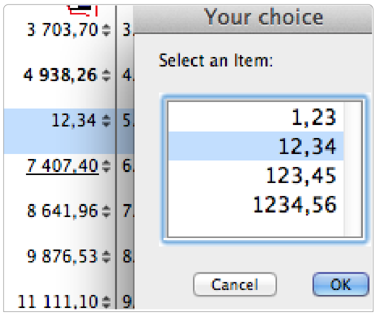
```
AL_SetAreaTextProperty($area;ALP_Area_CallbackMethPopup;"_Alp_PopupCallback")  //entry popup callback
```

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)  //ALP 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 as of version 9.9). The entry will also be ended when a non focusable object is clicked and ALP_Area_IgnoreSoftDeselect is set to true (see the explanation about Soft deselect).
- 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

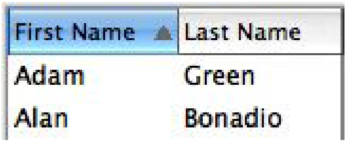
Areas can be sorted either by clicking in a column header or by invoking the AreaList Pro sort editor. If the user clicks in a column header, the area is sorted in ascending order on that column; if he clicks again, the column will be sorted in descending order.

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.

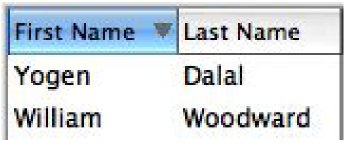
When a column has been sorted, a triangle (sort indicator) appears in the header:

Ascending Order Sort



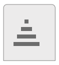
First Name ▲	Last Name
Adam	Green
Alan	Bonadio

Descending Order Sort



First Name ▼	Last Name
Yogen	Dalal
William	Woodward

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



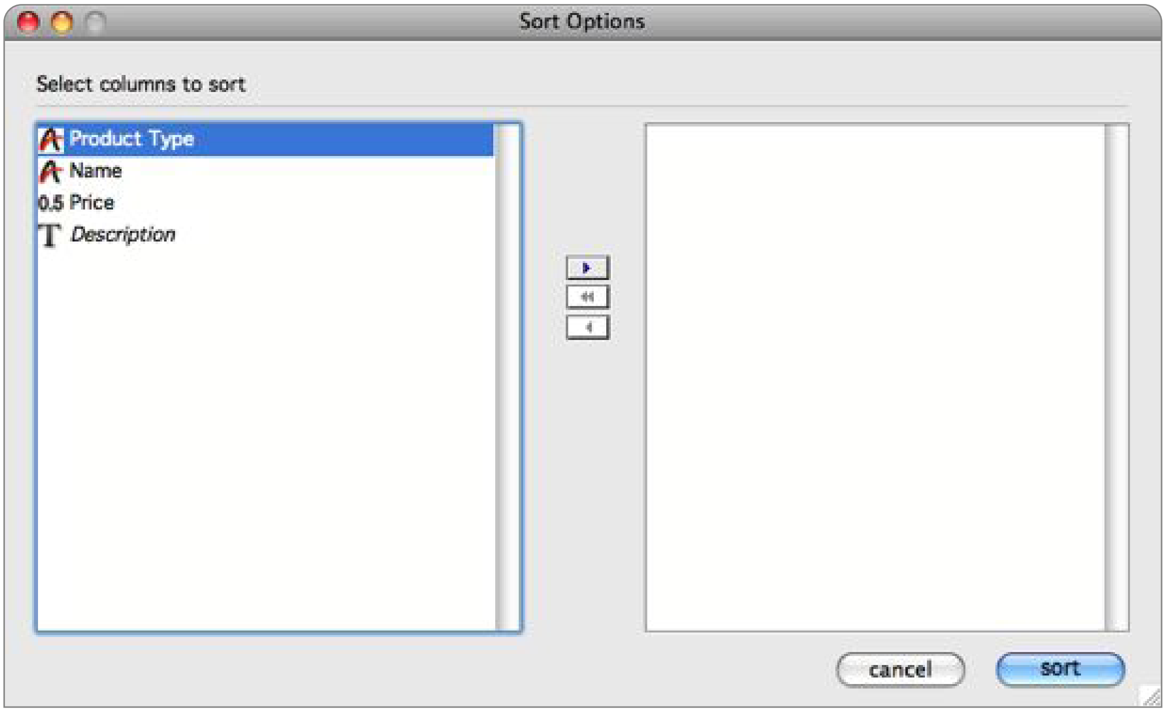
On Windows Vista, 7 and above, value 2 to ALP_Area_ShowSortIndicator draws the sort (non native) triangle to the right, not on top.

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.

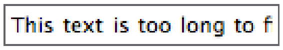
See the demonstration database (AreaList > Configuration options…).

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.

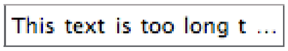
> Note: when word wrapping is enabled and a word does not fit into a column width, the word is split (e.g. “TEXT” can be split into “TE” + “XT” on next line).

This text is too long to fit into the cell:



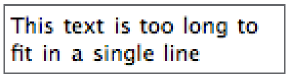
This text is too long to f

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



This text is too long t ...

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



This text is too long to
fit in a single line

> 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)

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.

Area properties

Use these properties with commands in the Area theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Area_EntryAllowReturn	✔	✔	✔	bool	false (0)			Allow RETURN in text
ALP_Area_EntryHighlight	✔	✔		range				Entry highlight in the form: **String** ($startOfSelection)+"," +**String** ($endOfSelection)
ALP_Area_EntryHighlightE	✔	✔		long int				Entry highlight end
ALP_Area_EntryHighlightS	✔	✔		long int				Entry highlight start
ALP_Area_EntryMapEnter	✔	✔	✔	long int	0	0	3	Map Enter key to: 0 = nothing (ignore) 1 = Tab 2 = Return 3 = Return for text fields, Tab otherwise
ALP_Area_EntrySelectedText	✔	✔		text				Selected text (entry must be in progress)
ALP_Area_EntryText	✔	✔		text				Entry text (entry must be in progress)
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_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_HierIndent	✔	✔	✔	real	16	0	64	Indent increment for every hierarchy level (for use with hierarchical lists)
ALP_Area_RowIndent	✔	✔	✔	point	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_UseEllipsis	✔	✔	✔	long 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

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Area_WindowsText	✔	✔		bool	false (0)			1 = change the engine used for drawing on Windows to GDI drawing (ignored on Mac) 0 (default) = use GDI+ 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 May be used with the area reference set to zero (newly created areas will use this mode)

Column Properties

Use these properties with commands in the Columns theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Column_Attributed	✔	✔	✔	bool	false (0)			Use attributed (multi-style) text: 0 = no, 1 = yes See also AreaList Pro Text Style Tags
ALP_Column_FooterText	✔	✔	✔	text				Footer text
ALP_Column_HeaderText	✔	✔	✔	text				Header text
ALP_Column_Length	✔			long int				Size of the alpha 4D field Zero means it is not an alpha (length-limited) field
ALP_Column_Uppercase	✔	✔	✔	bool				Make uppercase
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_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	false (0)			Header font style = bold
ALP_Column_HdrStyleF	✔	✔	✔	long int	0	0	7	Header font style, using 4D style constants (e.g. Bold, Italic, etc.)
ALP_Column_HdrStyleI	✔	✔	✔	bool	false (0)			Header font style = italic

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Column_HdrStyleU	✔	✔	✔	bool	false (0)			Header font style = underlined
ALP_Column_HdrTextColor	✔	✔	✔	color	#FF000000			Header font color Default is black
ALP_Column_HdrVertAlign	✔	✔	✔	long int	2	0	3	Header vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Column_HdrWrap	✔	✔	✔	bool	false (0)			Wrap long lines in header
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_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	false (0)			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	false (0)			Footer font style = italic
ALP_Column_FtrStyleU	✔	✔	✔	bool	false (0)			Footer font style = underlined
ALP_Column_FtrTextColor	✔	✔	✔	color	#FF000000			Footer font color Default is black
ALP_Column_FtrVertAlign	✔	✔	✔	long int	2	0	3	Footer vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Column_FtrWrap	✔	✔	✔	bool	false (0)			Wrap long lines in footer
ALP_Column_FontName	✔	✔	✔	text	Verdana on Windows Lucida Grande on MacOS			List font name

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Column_HorAlign	✔	✔	✔	long int	0	0	5	List horizontal alignment: 0 = default 1 = left 2 = center 3 = right 4 = justify 5 = full justify
ALP_Column_HorizontalScale	✔	✔	✔	real	1	0,1	100	List horizontal scale
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	false (0)			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	false (0)			List font style = italic
ALP_Column_StyleU	✔	✔	✔	bool	false (0)			List font style = underlined
ALP_Column_TextColor	✔	✔	✔	color	#FF000000			List font color Default is black
ALP_Column_VertAlign	✔	✔	✔	long int	0	0	3	List vertical alignment 0 = default 1 = top 2 = center 3 = bottom
ALP_Column_Wrap	✔	✔	✔	bool	false (0)			Wrap long lines in list

Row Properties

Use these properties with commands in the Rows theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
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_FontName	✔	✔	✔	text				Font name
ALP_Row_HorAlign	✔	✔	✔	long 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_Rotation	✔	✔	✔	real		-360	360	Rotation of text
ALP_Row_Size	✔	✔	✔	real		4	128	Font size
ALP_Row_StyleB	✔	✔	✔	bool				Font style = bold

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Row_StyleF	✔	✔	✔	long 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_TextColor	✔	✔	✔	color				Font color
ALP_Row_VertAlign	✔	✔	✔	long int		0	3	Vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Row_Wrap	✔	✔	✔	bool				Wrap long lines

Cell Properties

Use these properties with commands in the Cells theme:

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Cell_BaseLineShift	✔	✔	✔	real		-100	256	Baseline shift
ALP_Cell_ClearStyle		✔		n/a				Clear the style of this cell The area redraws automatically
ALP_Cell_Flags	✔	✔	✔	long 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 Maintained by AreaList Pro and should not normally be changed 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	✔	✔	✔	long 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_Rotation	✔	✔	✔	real		-360	360	Rotation of text
ALP_Cell_Size	✔	✔	✔	real		4	128	Font size

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
ALP_Cell_StyleB	✔	✔	✔	bool				Font style = bold
ALP_Cell_StyleF	✔	✔	✔	long 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_VertAlign	✔	✔	✔	long int		0	3	Vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Cell_Wrap	✔	✔	✔	bool				Wrap long lines

Object Properties

Use these properties with commands in the Objects theme:

Constant	Get	Set	Array Type	Comments
ALP_Object_FooterText	✔	✔	text	Footer text of all columns
ALP_Object_FooterTextNH	✔		text	Footer text of visible columns in grid order
ALP_Object_HeaderText	✔	✔	text	Header text of all columns
ALP_Object_HeaderTextNH	✔		text	Header text of visible columns

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 AreaList 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-").

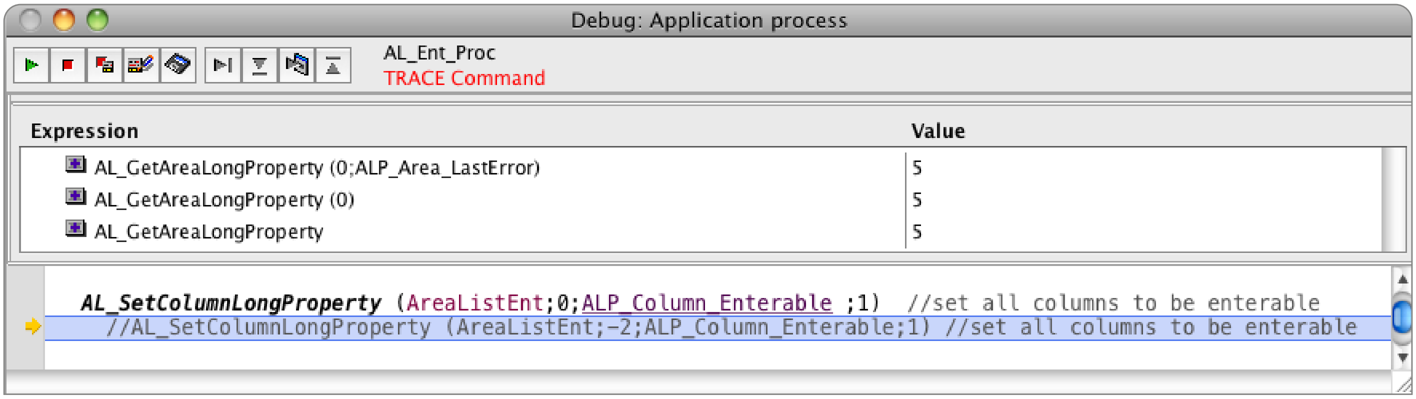
The trailing minus sign "-" means “use empty string for null date”. The following line will do exactly the same as the above example:

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:


Debug: Application process

AL_Ent_Proc
TRACE Command

Expression	Value
AL_GetAreaLongProperty (0;ALP_Area_LastError)	5
AL_GetAreaLongProperty (0)	5
AL_GetAreaLongProperty	5

```
  AL_SetColumnLongProperty (AreaListEnt;0;ALP_Column_Enterable ;1)  //set all columns to be enterable
    //AL_SetColumnLongProperty (AreaListEnt;-2;ALP_Column_Enterable;1) //set all columns to be enterable
```

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

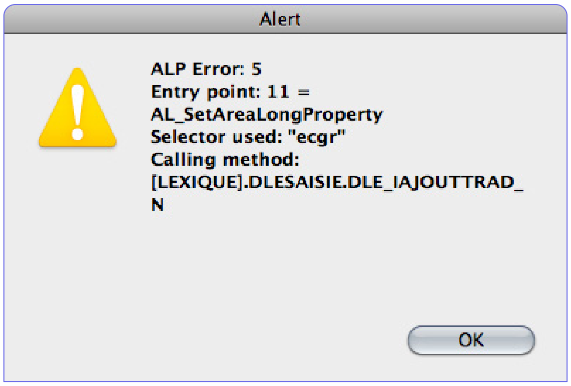
AL_GetAreaLongProperty //short syntax

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):



Alert

ALP Error: 5
Entry point: 11 =
AL_SetAreaLongProperty
Selector used: "ecgr"
Calling method:
[LEXIQUE].DLESAISIE.DLE_IAJOUTTRAD_
N

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

6 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 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 seven 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

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
```

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	✔			long 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”) ALP_Area_IgnoreSoftDeselect must be false (0: default value)
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.

If ALP_Area_IgnoreSoftDeselect is set to 0 (default value), 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.

Note: when the ALP_Area_IgnoreSoftDeselect property is set to true (1) soft deselect is handled as hard deselect.

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!

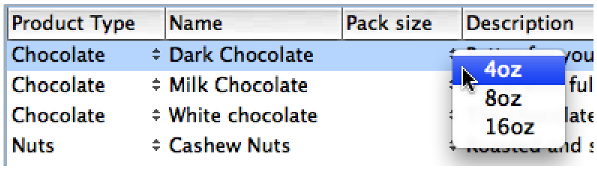
Then we assign a popup callback method to the area:

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

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

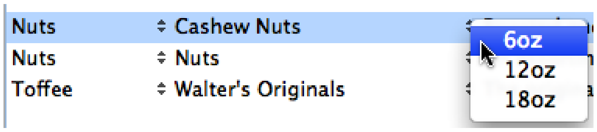
```
C_BOOLEAN($0)  //Return True if handled; False if not handled
C_LONGINT($1)  //AreaList Pro object reference
C_LONGINT($2)  //row
C_LONGINT($3)  //column
C_LONGINT($4)  //data type
$go:=True
Case of
    : ([product]product_type=”chocolate”)
        $pop:=tPopChoc
        $array:=->atChocsizes
    : ([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:



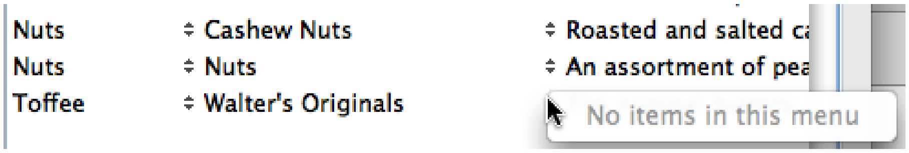
Product Type	Name	Pack size	Description
Chocolate	Dark Chocolate		you
Chocolate	Milk Chocolate		ful
Chocolate	White chocolate		late
Nuts	Cashew Nuts		Roasted and s

This one if it's Nuts:



Nuts	Cashew Nuts	
Nuts	Nuts	
Toffee	Walter's Originals	

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



Nuts	Cashew Nuts	Roasted and salted ca
Nuts	Nuts	An assortment of pea
Toffee	Walter's Originals	

See also Entering data in AreaList Pro with DisplayList.

7 Columns

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

Compatibility mode

Here are the compatibility mode differences in AreaList Pro version 9 regarding column behavior.

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: in AreaList Pro version 9, 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 true (1).

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 used to display column widths in the headers when clicking on a X button located at the area's lower right corner.

AreaList Pro v9 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".

This is also true with the DisplayList module included into AreaList Pro v9.

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: in AreaList Pro version 9 (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
```

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.

8 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 in AreaList Pro version 9 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 in AreaList Pro v9 (the scrollbar is drawn, and it is bigger than in 8.x) In 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)

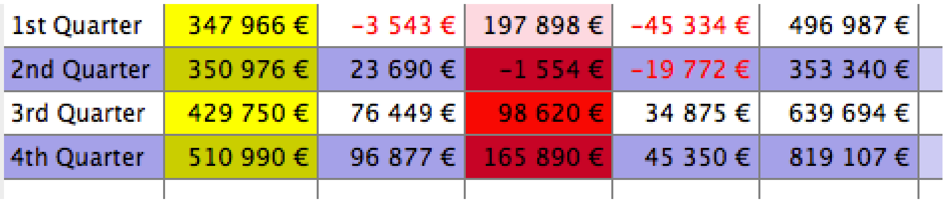
The "empty space below last data row" refers to the area between the last row and the footer/horizontal scrollbar/bottom of the AreaList Pro area, where a click or a rollover reports -2 using ALP_Area_ClickedRow or ALP_Area_RollOverRow.

See Row Numbering.

Combining Alt Row color with Background color

When bit 3 is set to true (1) 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.



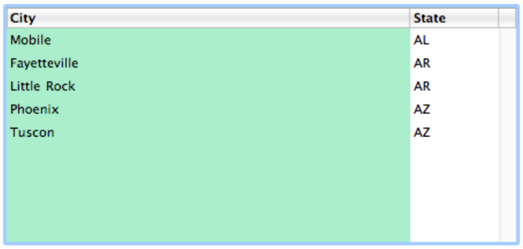
1st Quarter	347 966 €	-3 543 €	197 898 €	-45 334 €	496 987 €
2nd Quarter	350 976 €	23 690 €	-1 554 €	-19 772 €	353 340 €
3rd Quarter	429 750 €	76 449 €	98 620 €	34 875 €	639 694 €
4th Quarter	510 990 €	96 877 €	165 890 €	45 350 €	819 107 €

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:

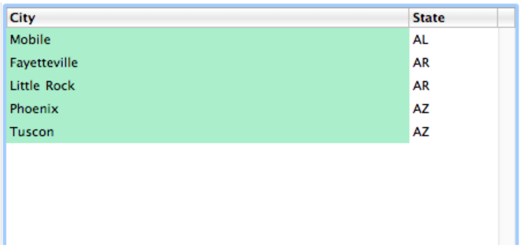
AL_SetColumnLongProperty (eList;1;ALP_Column_BackColor;0xFFAAEECC)



City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ

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”:

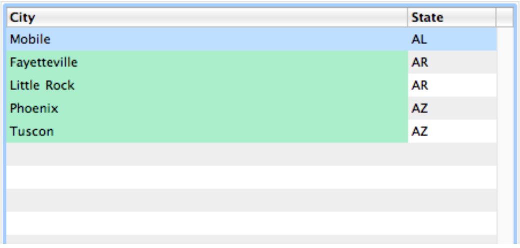
AL_SetCellLongProperty (eList;-2;1;ALP_Cell_BackColor;0xFFAAEECC)



City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ

Alternate row coloring will apply to the “empty” bottom section when bit 2 of ALP_Area_AltRowOptions is set to true (1):

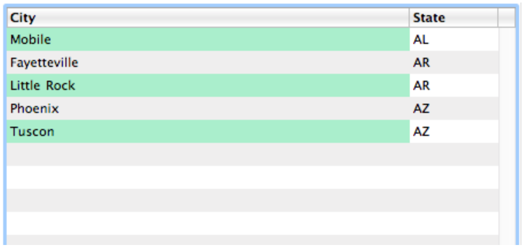
AL_SetAreaLongProperty (eList;ALP_Area_AltRowOptions;5)



City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ

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”):

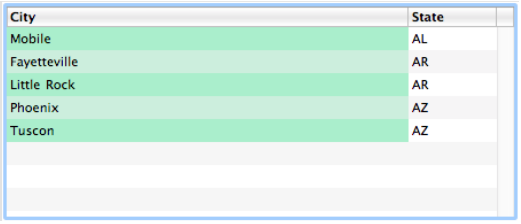
AL_SetAreaLongProperty (eList;ALP_Area_AltRowOptions;13)



City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ

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)

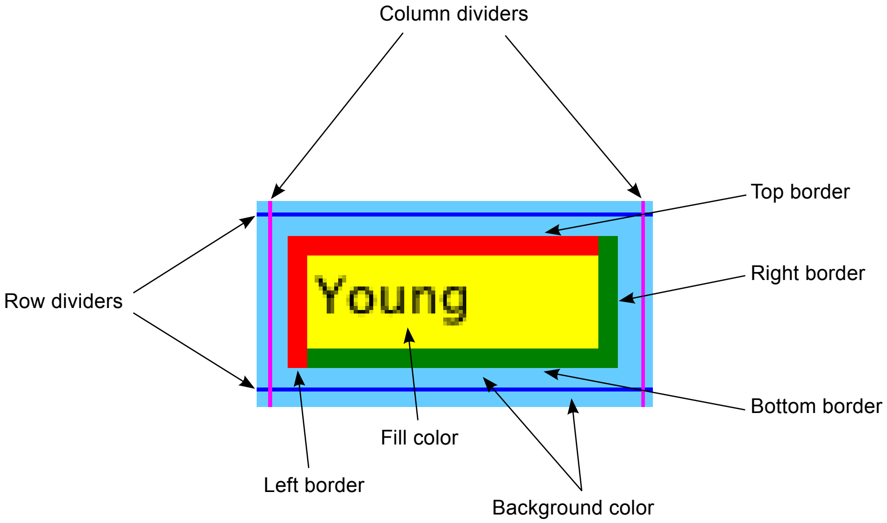


City	State
Mobile	AL
Fayetteville	AR
Little Rock	AR
Phoenix	AZ
Tuscon	AZ

Coloring Cell Sections

Summary

Here are the various parts of the cell, which can be individually set / colored:

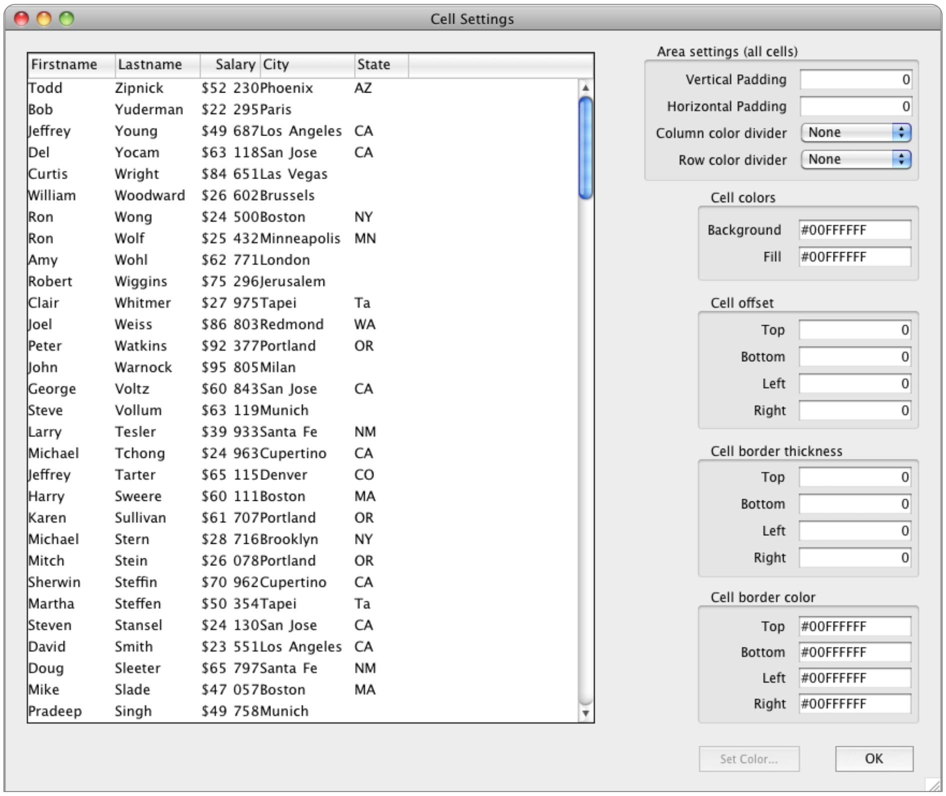


Example

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:



Firstname	Lastname	Salary	City	State
Todd	Zipnick	$52 230	Phoenix	AZ
Bob	Yuderman	$22 295	Paris	
Jeffrey	Young	$49 687	Los Angeles	CA
Del	Yocam	$63 118	San Jose	CA
Curtis	Wright	$84 651	Las Vegas	
William	Woodward	$26 602	Brussels	
Ron	Wong	$24 500	Boston	NY
Ron	Wolf	$25 432	Minneapolis	MN
Amy	Wohl	$62 771	London	
Robert	Wiggins	$75 296	Jerusalem	
Clair	Whitmer	$27 975	Tapei	Ta
Joel	Weiss	$86 803	Redmond	WA
Peter	Watkins	$92 377	Portland	OR
John	Warnock	$95 805	Milan	
George	Voltz	$60 843	San Jose	CA
Steve	Vollum	$63 119	Munich	
Larry	Tesler	$39 933	Santa Fe	NM
Michael	Tchong	$24 963	Cupertino	CA
Jeffrey	Tarter	$65 115	Denver	CO
Harry	Sweere	$60 111	Boston	MA
Karen	Sullivan	$61 707	Portland	OR
Michael	Stern	$28 716	Brooklyn	NY
Mitch	Stein	$26 078	Portland	OR
Sherwin	Steffin	$70 962	Cupertino	CA
Martha	Steffen	$50 354	Tapei	Ta
Steven	Stansel	$24 130	San Jose	CA
David	Smith	$23 551	Los Angeles	CA
Doug	Sleeter	$65 797	Santa Fe	NM
Mike	Slade	$47 057	Boston	MA
Pradeep	Singh	$49 758	Munich	

Area settings (all cells)
Vertical Padding: 0
Horizontal Padding: 0
Column color divider: None
Row color divider: None

Cell colors
Background: #00FFFFFF
Fill: #00FFFFFF

Cell offset
Top: 0
Bottom: 0
Left: 0
Right: 0

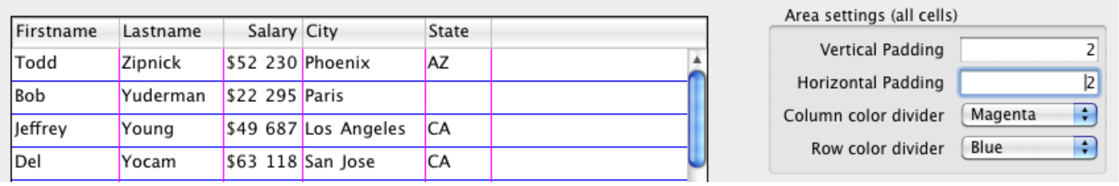
Cell border thickness
Top: 0
Bottom: 0
Left: 0
Right: 0

Cell border color
Top: #00FFFFFF
Bottom: #00FFFFFF
Left: #00FFFFFF
Right: #00FFFFFF

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:

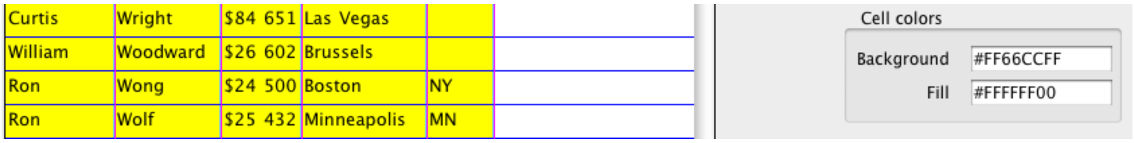


Firstname	Lastname	Salary	City	State
Todd	Zipnick	$52 230	Phoenix	AZ
Bob	Yuderman	$22 295	Paris	
Jeffrey	Young	$49 687	Los Angeles	CA
Del	Yocam	$63 118	San Jose	CA

Area settings (all cells)
Vertical Padding: 2
Horizontal Padding: 2
Column color divider: Magenta
Row color divider: 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:

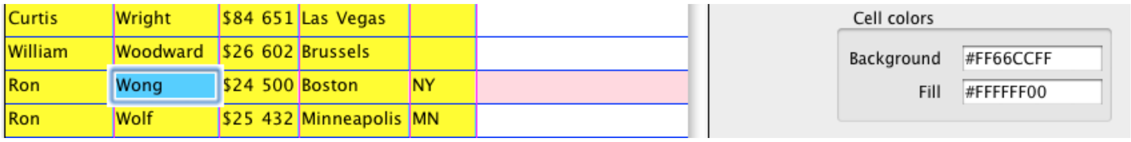


Curtis	Wright	$84 651	Las Vegas	
William	Woodward	$26 602	Brussels	
Ron	Wong	$24 500	Boston	NY
Ron	Wolf	$25 432	Minneapolis	MN

Cell colors
Background: #FF66CCFF
Fill: #FFFFFF00

Text Editing

We will see our blue Background if we enter a cell for text editing:



Curtis	Wright	$84 651	Las Vegas	
William	Woodward	$26 602	Brussels	
Ron	Wong	$24 500	Boston	NY
Ron	Wolf	$25 432	Minneapolis	MN

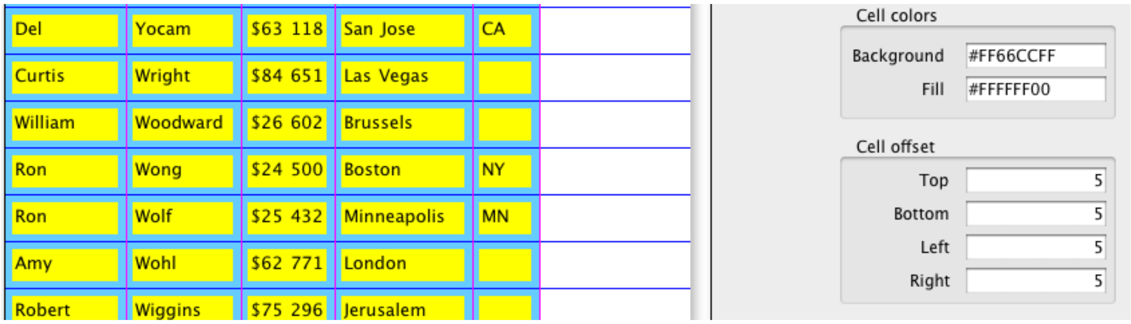
Cell colors

Background #FF66CCFF

Fill #FFFFFF00

Cell Offsets

Setting Cell Offsets will reveal our Background color:



Del	Yocam	$63 118	San Jose	CA
Curtis	Wright	$84 651	Las Vegas	
William	Woodward	$26 602	Brussels	
Ron	Wong	$24 500	Boston	NY
Ron	Wolf	$25 432	Minneapolis	MN
Amy	Wohl	$62 771	London	
Robert	Wiggins	$75 296	Jerusalem	

Cell colors

Background #FF66CCFF

Fill #FFFFFF00

Cell offset

Top 5

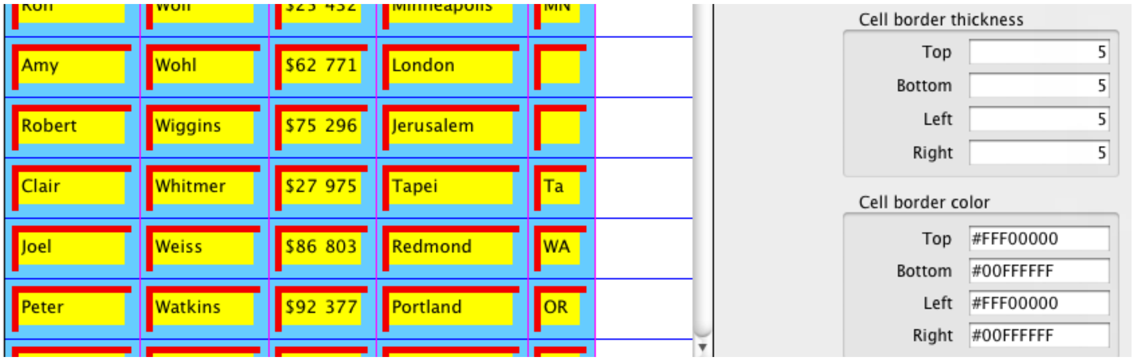
Bottom 5

Left 5

Right 5

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:



Ron	Wolf	$25 432	Minneapolis	MN
Amy	Wohl	$62 771	London	
Robert	Wiggins	$75 296	Jerusalem	
Clair	Whitmer	$27 975	Tapei	Ta
Joel	Weiss	$86 803	Redmond	WA
Peter	Watkins	$92 377	Portland	OR

Cell border thickness

Top 5

Bottom 5

Left 5

Right 5

Cell border color

Top #FFF00000

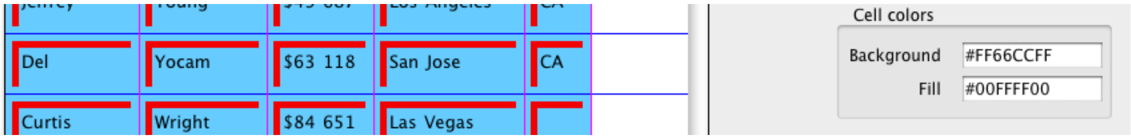
Bottom #00FFFFFF

Left #FFF00000

Right #00FFFFFF

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:



Jeffrey	Young	[illegible]	Los Angeles	CA
Del	Yocam	$63 118	San Jose	CA
Curtis	Wright	$84 651	Las Vegas	

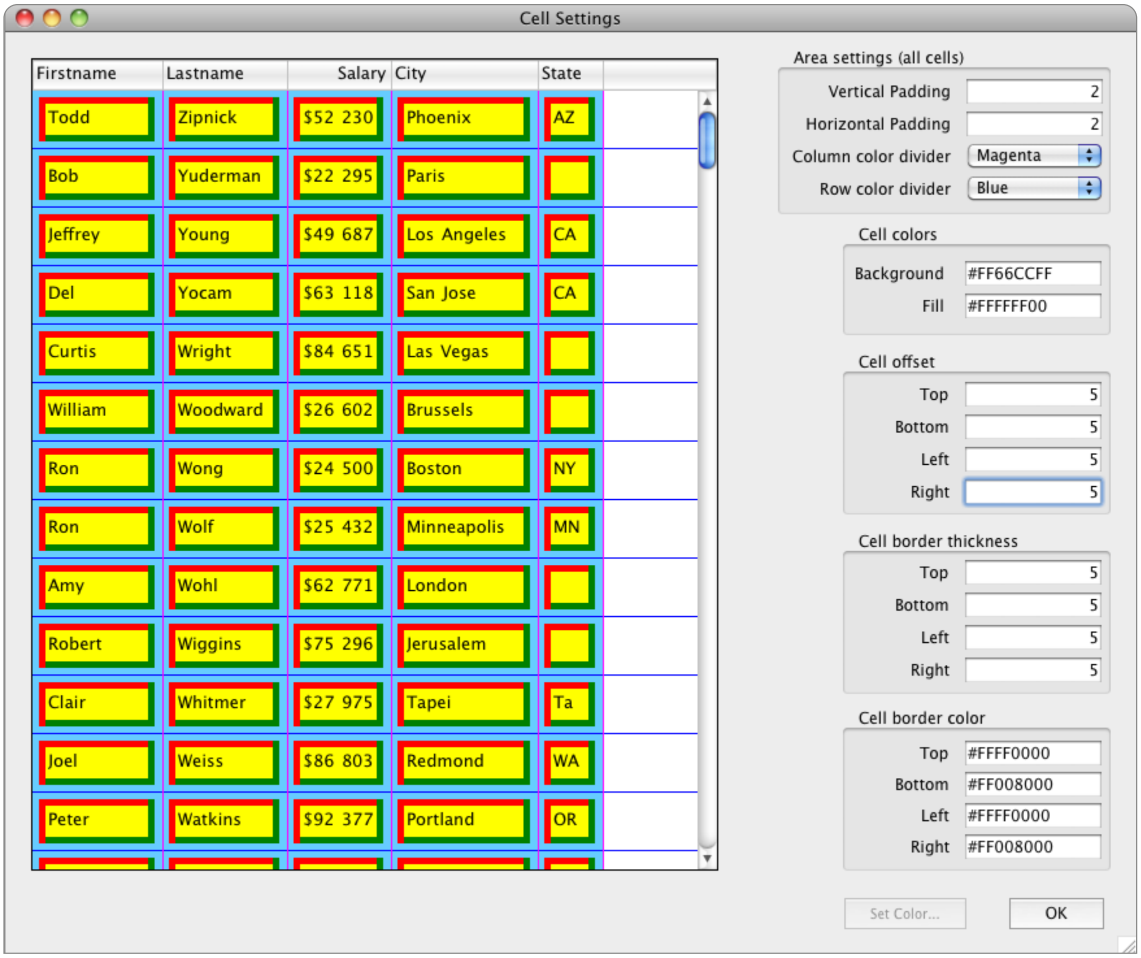
Cell colors

Background #FF66CCFF

Fill #00FFFF00

Final Result

Let’s revert to our opaque yellow Fill and add green right and bottom Borders:



Cell Settings

Firstname	Lastname	Salary	City	State
Todd	Zipnick	$52 230	Phoenix	AZ
Bob	Yuderman	$22 295	Paris	
Jeffrey	Young	$49 687	Los Angeles	CA
Del	Yocam	$63 118	San Jose	CA
Curtis	Wright	$84 651	Las Vegas	
William	Woodward	$26 602	Brussels	
Ron	Wong	$24 500	Boston	NY
Ron	Wolf	$25 432	Minneapolis	MN
Amy	Wohl	$62 771	London	
Robert	Wiggins	$75 296	Jerusalem	
Clair	Whitmer	$27 975	Tapei	Ta
Joel	Weiss	$86 803	Redmond	WA
Peter	Watkins	$92 377	Portland	OR

Area settings (all cells)

Vertical Padding: 2
Horizontal Padding: 2
Column color divider: Magenta
Row color divider: Blue

Cell colors

Background: #FF66CCFF
Fill: #FFFFFF00

Cell offset

Top: 5
Bottom: 5
Left: 5
Right: 5

Cell border thickness

Top: 5
Bottom: 5
Left: 5
Right: 5

Cell border color

Top: #FFFF0000
Bottom: #FF008000
Left: #FFFF0000
Right: #FF008000

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
```

Empty column background color

When the total width of all columns is lesser than the area's display width, and neither ALP_Area_AutoResizeColumn or 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. They are interpreted by AreaList Pro version 9 as transparency ratios (alpha channel value):

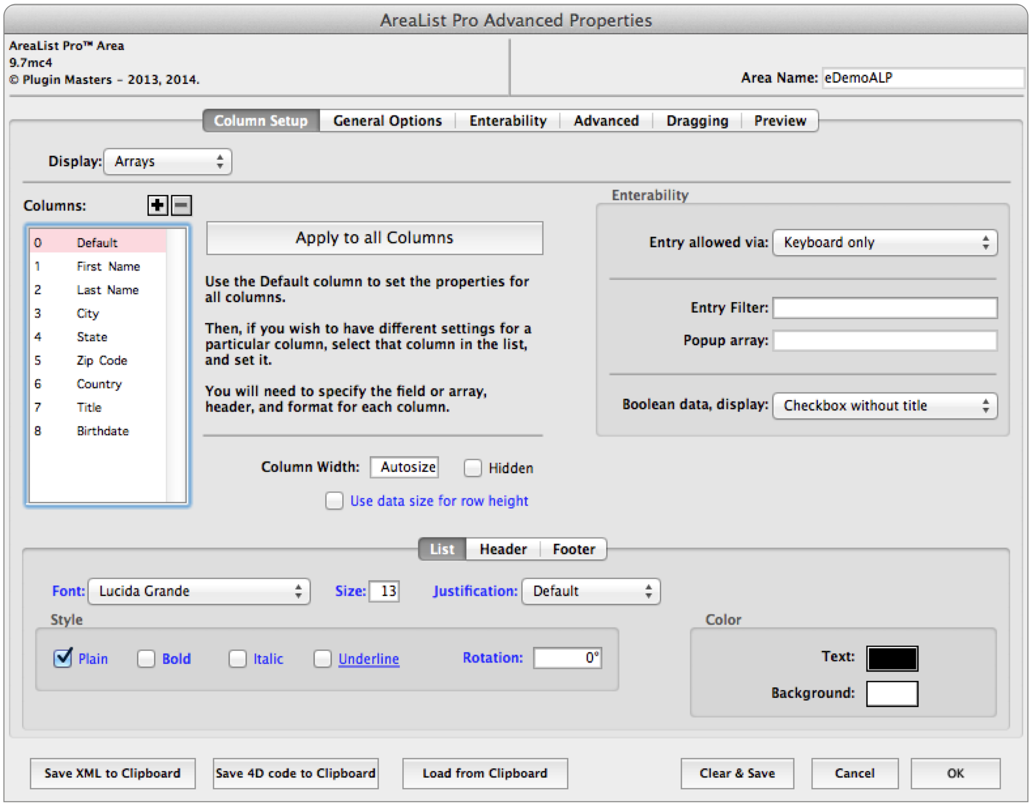
- "black" or 1: 100% (0xFF)
- "darkgray" or 4: 75% (0xC0)
- "gray" or 2: 50% (0x80)
- "lightgray" or 3: 25% (0x40)
- "white" or 0 or "none" or "" or anything else: 0% = no drawing (0x00)

9 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 **Edit …** button next to **Advanced Properties** in the object's Property List window
3. The Advanced Properties window opens:



AreaList Pro Advanced Properties

AreaList Pro™ Area
9.7mc4
© Plugin Masters - 2013, 2014.

Area Name: eDemoALP

Display: Arrays

Columns:

0	Default
1	First Name
2	Last Name
3	City
4	State
5	Zip Code
6	Country
7	Title
8	Birthdate

Use the Default column to set the properties for all columns.

Then, if you wish to have different settings for a particular column, select that column in the list, and set it.

You will need to specify the field or array, header, and format for each column.

Column Width: Autosize □ Hidden

□ Use data size for row height

Enterability

Entry allowed via: Keyboard only

Entry Filter:

Popup array:

Boolean data, display: Checkbox without title

Font: Lucida Grande Size: 13 Justification: Default

Style

☑ Plain □ Bold □ Italic □ Underline Rotation: 0°

Color

Text: ■

Background: □

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.

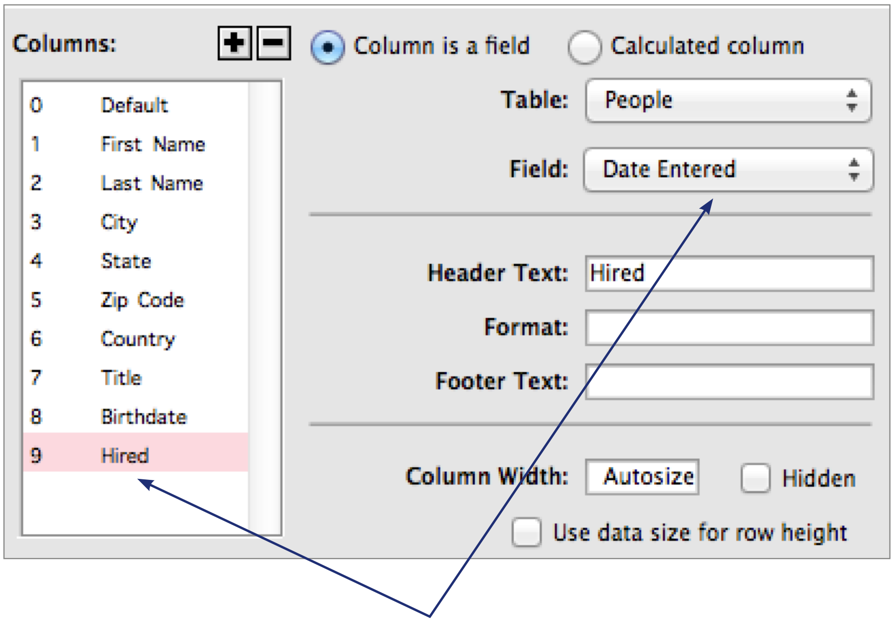
If you have made a change to your Default column and you want to apply that change to all the existing columns, click this button. The current Default column settings will be applied to the properties with blue labels for all columns.

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:



Columns:

0	Default
1	First Name
2	Last Name
3	City
4	State
5	Zip Code
6	Country
7	Title
8	Birthdate
9	Hired

● Column is a field ○ Calculated column

Table: People

Field: Date Entered

Header Text: Hired

Format:

Footer Text:

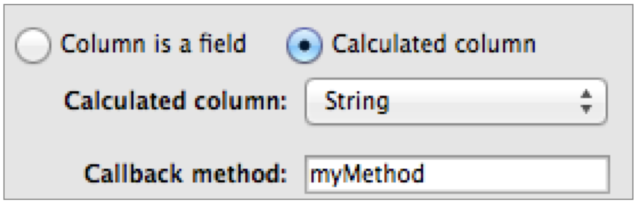
Column Width: Autosize □ Hidden

□ Use data size for row height

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:



○ Column is a field ● Calculated column

Calculated column: String

Callback method: myMethod

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.

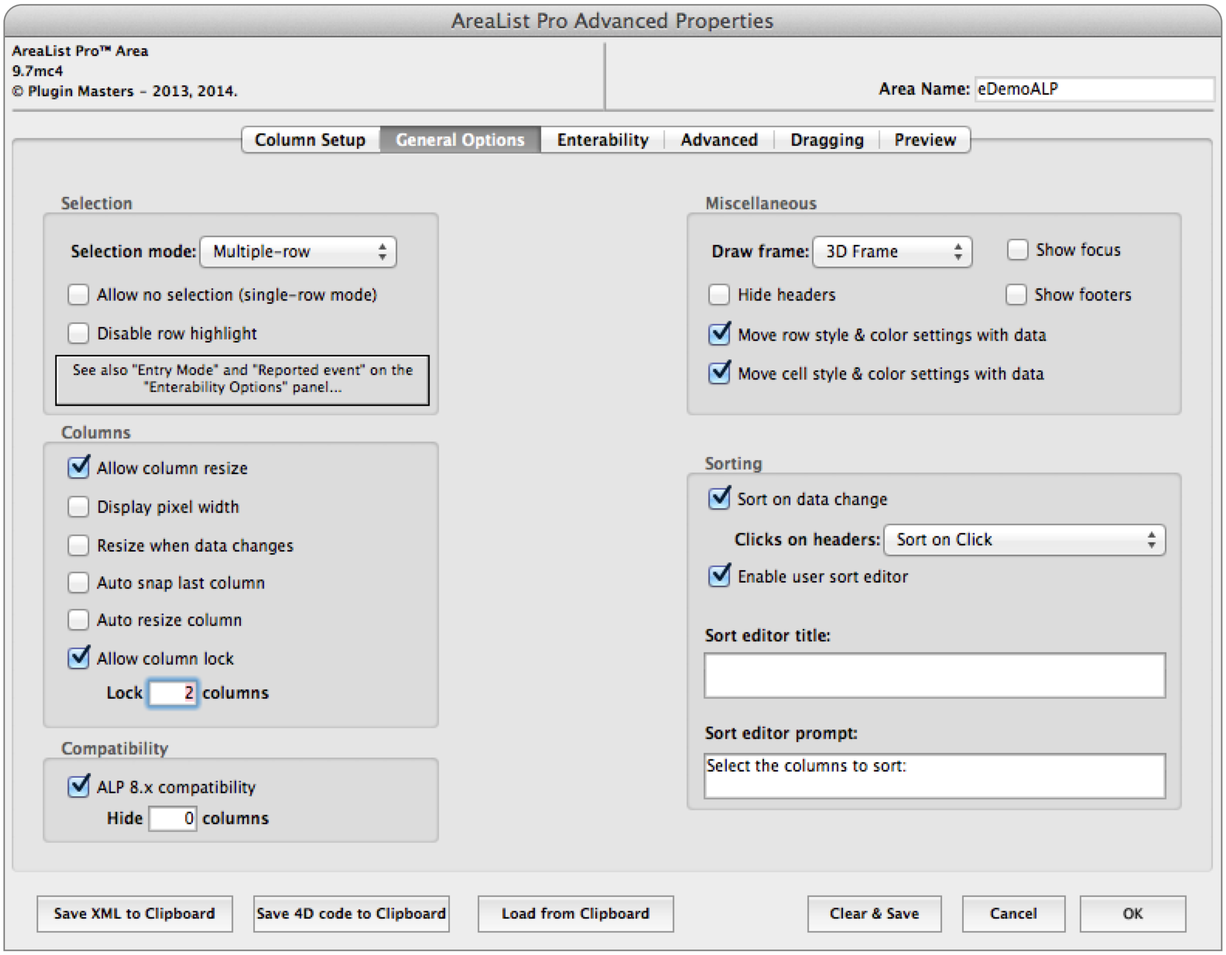
Style options (font, size, color, etc.): You can select any styling for each column.

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

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:



AreaList Pro Advanced Properties

AreaList Pro™ Area
9.7mc4
© Plugin Masters - 2013, 2014.

Area Name: eDemoALP

Selection

Selection mode: Multiple-row

□ Allow no selection (single-row mode)

□ Disable row highlight

See also "Entry Mode" and "Reported event" on the "Enterability Options" panel...

Columns

☑ Allow column resize

□ Display pixel width

□ Resize when data changes

□ Auto snap last column

□ Auto resize column

☑ Allow column lock

Lock 2 columns

Compatibility

☑ ALP 8.x compatibility

Hide 0 columns

Miscellaneous

Draw frame: 3D Frame

□ Show focus

□ Hide headers

□ Show footers

☑ Move row style & color settings with data

☑ Move cell style & color settings with data

Sorting

☑ Sort on data change

Clicks on headers: Sort on Click

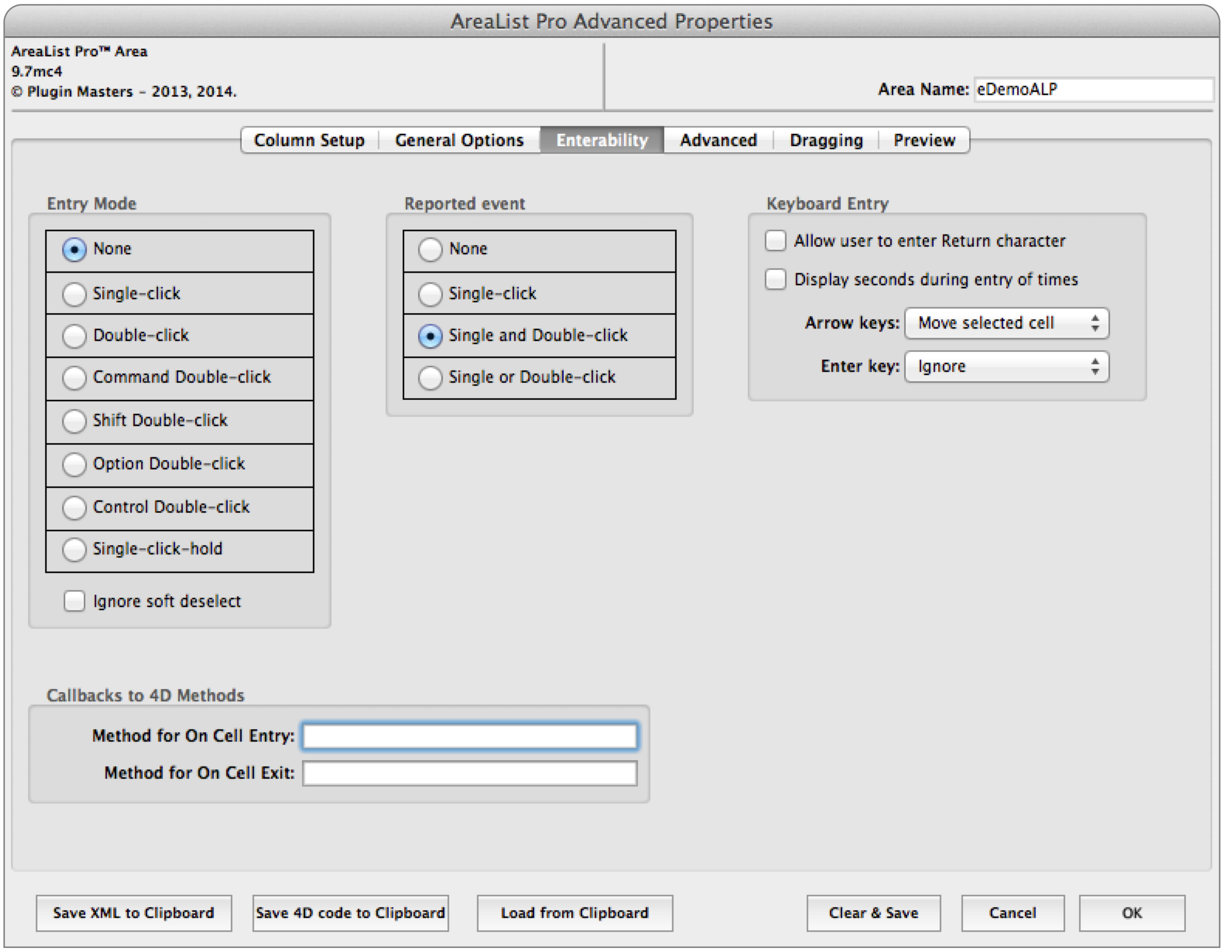
☑ Enable user sort editor

Sort editor title:

Sort editor prompt: Select the columns to sort:

Enterability

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:



AreaList Pro Advanced Properties

AreaList Pro™ Area
9.7mc4
© Plugin Masters - 2013, 2014.

Area Name: eDemoALP

Entry Mode

- ● None
- ○ Single-click
- ○ Double-click
- ○ Command Double-click
- ○ Shift Double-click
- ○ Option Double-click
- ○ Control Double-click
- ○ Single-click-hold
- □ Ignore soft deselect

Reported event

- ○ None
- ○ Single-click
- ● Single and Double-click
- ○ Single or Double-click

Keyboard Entry

- □ Allow user to enter Return character
- □ Display seconds during entry of times

Arrow keys: Move selected cell

Enter key: Ignore

Callbacks to 4D Methods

Method for On Cell Entry:

Method for On Cell Exit:

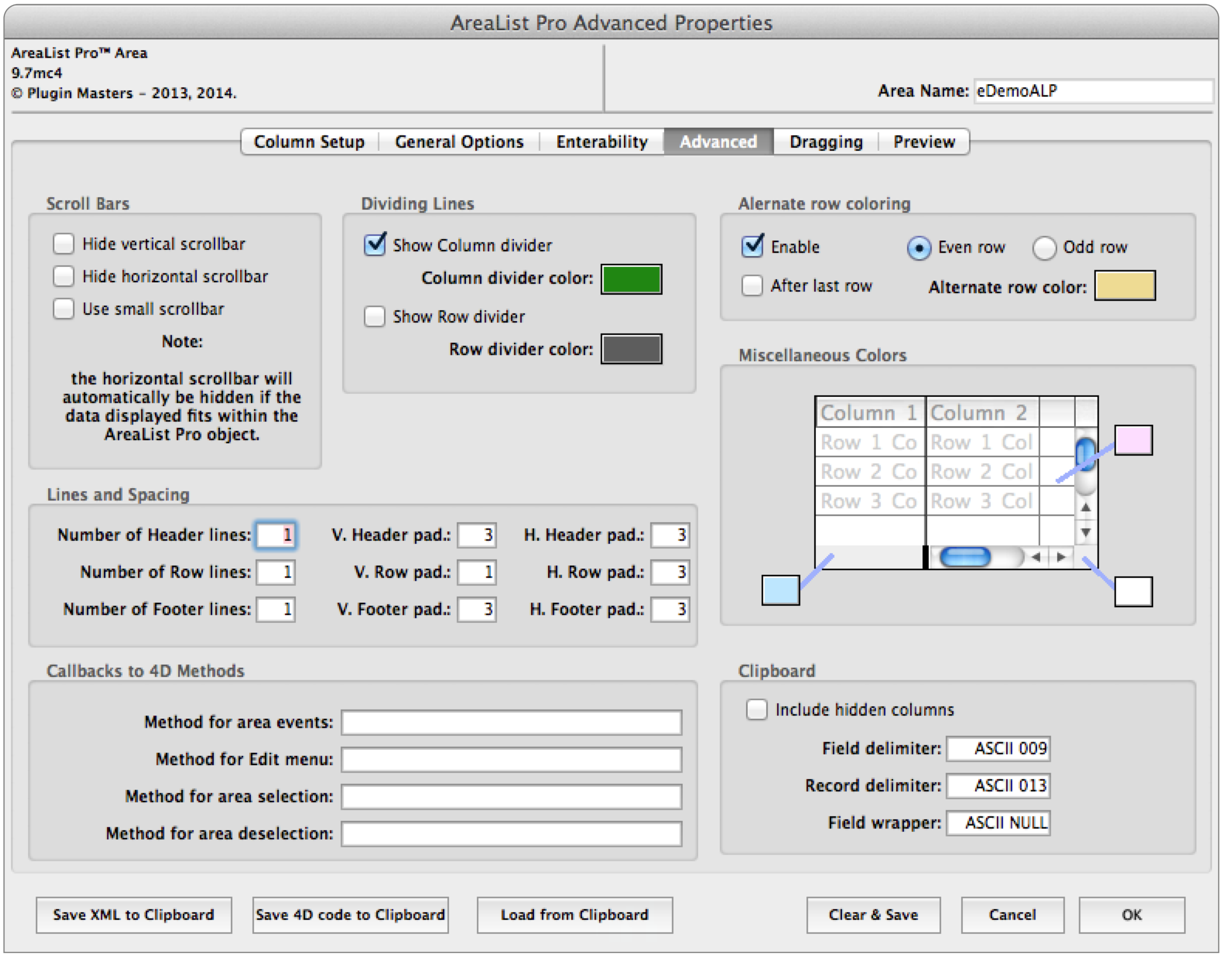
Callbacks: You can specify a callback method that will execute when an enterable cell is entered or exited.

Advanced

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



AreaList Pro Advanced Properties

AreaList Pro™ Area
9.7mc4
© Plugin Masters - 2013, 2014.

Area Name: eDemoALP

Scroll Bars

□ Hide vertical scrollbar
□ Hide horizontal scrollbar
□ Use small scrollbar

Note:

the horizontal scrollbar will automatically be hidden if the data displayed fits within the AreaList Pro object.

Dividing Lines

☑ Show Column divider
Column divider color: 🟩
□ Show Row divider
Row divider color: ■

Alernate row coloring

☑ Enable ● Even row ○ Odd row
□ After last row Alternate row color: 🟨

Miscellaneous Colors

Lines and Spacing

Number of Header lines: 1 V. Header pad.: 3 H. Header pad.: 3
Number of Row lines: 1 V. Row pad.: 1 H. Row pad.: 3
Number of Footer lines: 1 V. Footer pad.: 3 H. Footer pad.: 3

Callbacks to 4D Methods

Method for area events:
Method for Edit menu:
Method for area selection:
Method for area deselection:

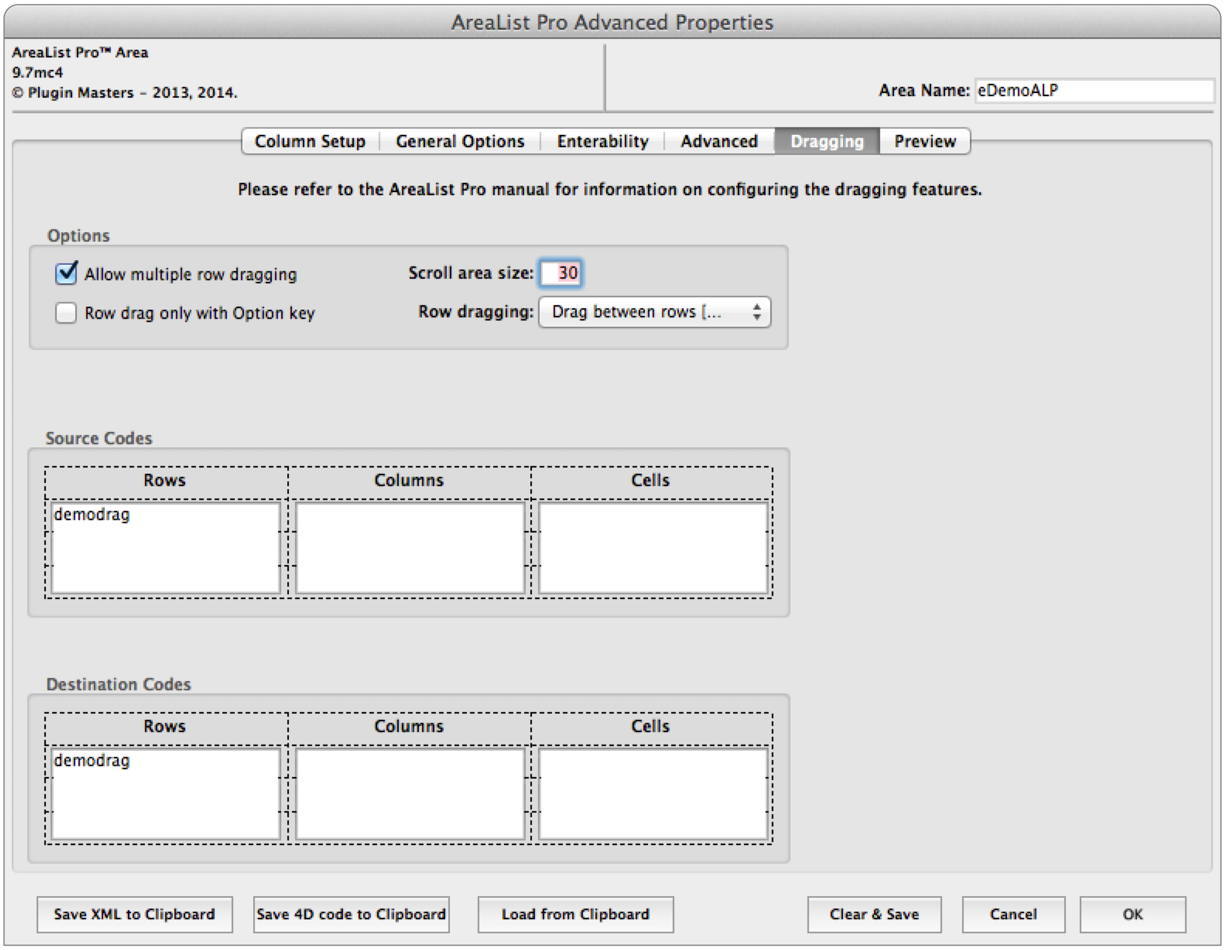
Clipboard

□ Include hidden columns
Field delimiter: ASCII 009
Record delimiter: ASCII 013
Field wrapper: ASCII NULL

Dragging

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:



AreaList Pro Advanced Properties

AreaList Pro™ Area
9.7mc4
© Plugin Masters - 2013, 2014.

Area Name: eDemoALP

Please refer to the AreaList Pro manual for information on configuring the dragging features.

Options

☑ Allow multiple row dragging

☐ Row drag only with Option key

Scroll area size: 30

Row dragging: Drag between rows [...

Source Codes

Rows	Columns	Cells
demodrag		

Destination Codes

Rows	Columns	Cells
demodrag		

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.

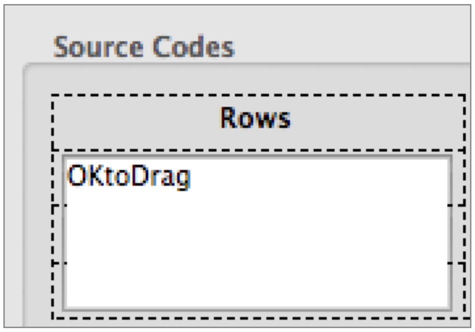
Scroll area size: The size of the frame around the AreaList Pro area border where dragging will start area scrolling. When the user drags something near the AreaList Pro area border, the contents will be scrolled.

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:



Source Codes

Rows

OKtoDrag

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.

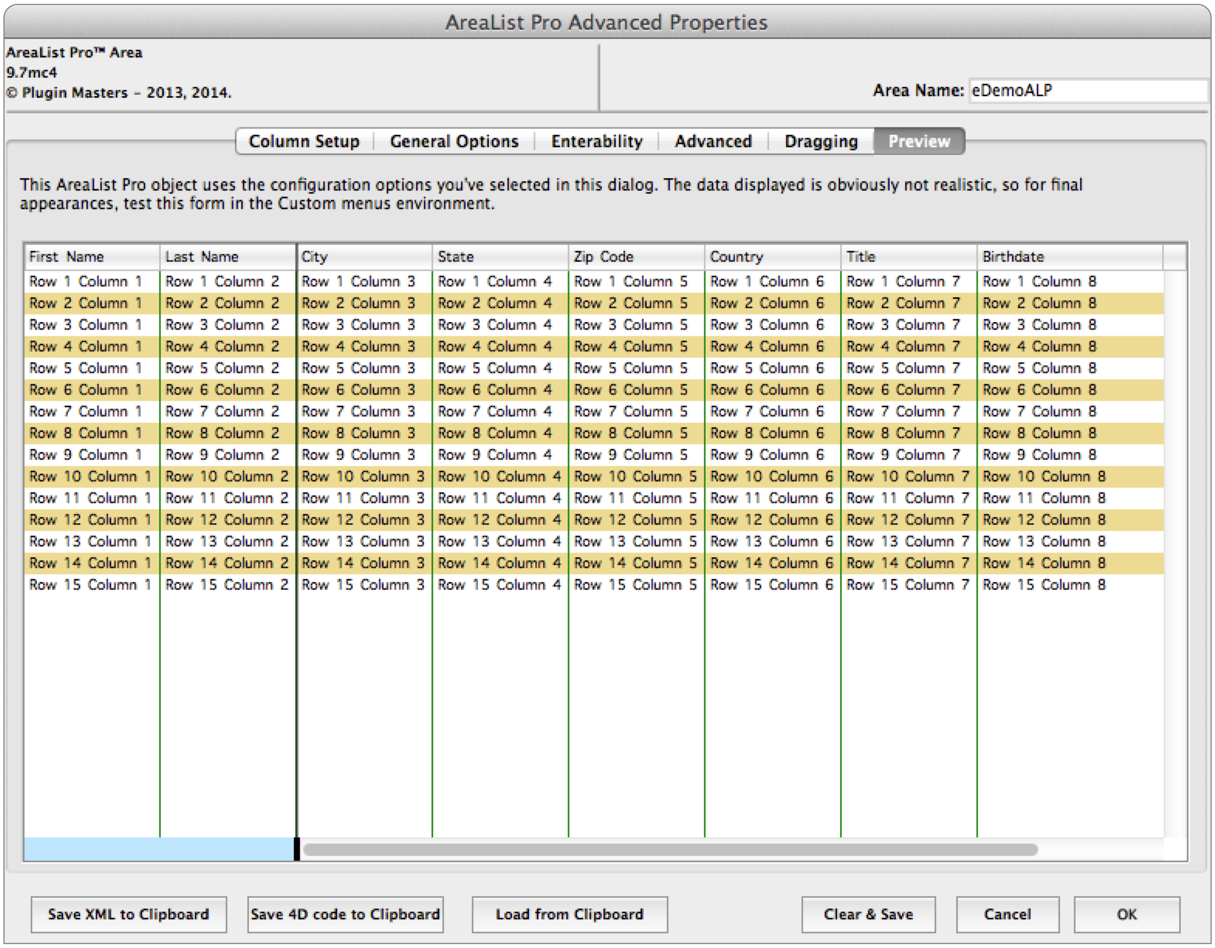
Once an object has been dragged, you will need to handle the Drop event programmatically; AreaList Pro doesn’t know what you want to do with the data that has been dropped, so you must tell it.

Please refer to the Drag and Drop topic for more detailed information and some examples.

Preview

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



AreaList Pro Advanced Properties

AreaList Pro™ Area
9.7mc4
© Plugin Masters - 2013, 2014.

Area Name: eDemoALP

This AreaList Pro object uses the configuration options you've selected in this dialog. The data displayed is obviously not realistic, so for final appearances, test this form in the Custom menus environment.

First Name	Last Name	City	State	Zip Code	Country	Title	Birthdate
Row 1 Column 1	Row 1 Column 2	Row 1 Column 3	Row 1 Column 4	Row 1 Column 5	Row 1 Column 6	Row 1 Column 7	Row 1 Column 8
Row 2 Column 1	Row 2 Column 2	Row 2 Column 3	Row 2 Column 4	Row 2 Column 5	Row 2 Column 6	Row 2 Column 7	Row 2 Column 8
Row 3 Column 1	Row 3 Column 2	Row 3 Column 3	Row 3 Column 4	Row 3 Column 5	Row 3 Column 6	Row 3 Column 7	Row 3 Column 8
Row 4 Column 1	Row 4 Column 2	Row 4 Column 3	Row 4 Column 4	Row 4 Column 5	Row 4 Column 6	Row 4 Column 7	Row 4 Column 8
Row 5 Column 1	Row 5 Column 2	Row 5 Column 3	Row 5 Column 4	Row 5 Column 5	Row 5 Column 6	Row 5 Column 7	Row 5 Column 8
Row 6 Column 1	Row 6 Column 2	Row 6 Column 3	Row 6 Column 4	Row 6 Column 5	Row 6 Column 6	Row 6 Column 7	Row 6 Column 8
Row 7 Column 1	Row 7 Column 2	Row 7 Column 3	Row 7 Column 4	Row 7 Column 5	Row 7 Column 6	Row 7 Column 7	Row 7 Column 8
Row 8 Column 1	Row 8 Column 2	Row 8 Column 3	Row 8 Column 4	Row 8 Column 5	Row 8 Column 6	Row 8 Column 7	Row 8 Column 8
Row 9 Column 1	Row 9 Column 2	Row 9 Column 3	Row 9 Column 4	Row 9 Column 5	Row 9 Column 6	Row 9 Column 7	Row 9 Column 8
Row 10 Column 1	Row 10 Column 2	Row 10 Column 3	Row 10 Column 4	Row 10 Column 5	Row 10 Column 6	Row 10 Column 7	Row 10 Column 8
Row 11 Column 1	Row 11 Column 2	Row 11 Column 3	Row 11 Column 4	Row 11 Column 5	Row 11 Column 6	Row 11 Column 7	Row 11 Column 8
Row 12 Column 1	Row 12 Column 2	Row 12 Column 3	Row 12 Column 4	Row 12 Column 5	Row 12 Column 6	Row 12 Column 7	Row 12 Column 8
Row 13 Column 1	Row 13 Column 2	Row 13 Column 3	Row 13 Column 4	Row 13 Column 5	Row 13 Column 6	Row 13 Column 7	Row 13 Column 8
Row 14 Column 1	Row 14 Column 2	Row 14 Column 3	Row 14 Column 4	Row 14 Column 5	Row 14 Column 6	Row 14 Column 7	Row 14 Column 8
Row 15 Column 1	Row 15 Column 2	Row 15 Column 3	Row 15 Column 4	Row 15 Column 5	Row 15 Column 6	Row 15 Column 7	Row 15 Column 8

Click **OK** when you are happy with your settings, and your area is ready to use.

10

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

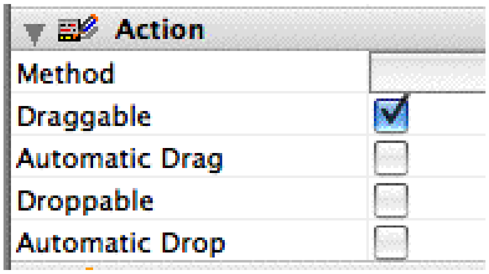
You must configure AreaList Pro to allow dragging out of and into an AreaList Pro area.

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:



Action	
Method	
Draggable	☑
Automatic Drag	□
Droppable	□
Automatic Drop	□

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 ***ALP_SetArea…*** and ***ALP_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
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 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 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 non 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.

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.

See also Drag and Drop from the Upgrading from Previous versions of AreaList Pro section.

11

Advanced Topics

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.

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	✓	✓	✓	long 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)

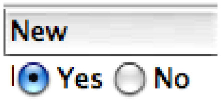
Example 1

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:



New

● Yes ○ No

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	✓	✓	✓	long 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 from the 4D Picture Library)

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)

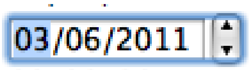
AL_SetColumnTextProperty (area;5;ALP_Column_Format;"Yes;No")

Dates

There are two options for date controls: inline and popup.

Inline Date Control

An inline date control looks like this:



03/06/2011

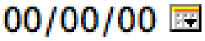
With this type of control, the user selects an element of the date (ie, the day, month, or year) and clicks the up or down arrows to increase or decrease the selected value. It will only accept valid dates.

Inline date controls are specified for an entire area with the ALP_Area_UseDateControls property of ***AL_SetAreaLongProperty*** - for example:

AL_SetAreaLongProperty (area;ALP_Area_UseDateControls;1)

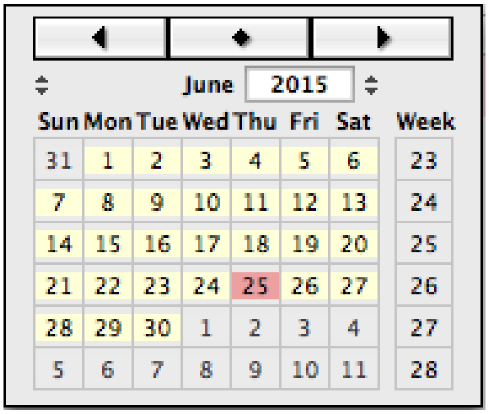
Popup Date Control

A popup date control appears as a little calendar when the data is being edited:



00/00/00

Click on the calendar icon and a calendar opens:



June 2015

Sun	Mon	Tue	Wed	Thu	Fri	Sat	Week
31	1	2	3	4	5	6	23
7	8	9	10	11	12	13	24
14	15	16	17	18	19	20	25
21	22	23	24	25	26	27	26
28	29	30	1	2	3	4	27
5	6	7	8	9	10	11	28

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.

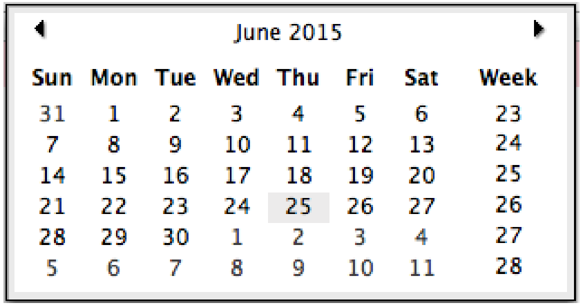
To specify a popup date control, use the "by popup" option of the ALP_Column_Enterable property - for example. to specify a date popup for column 6:

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 true (1):



June 2015

Sun	Mon	Tue	Wed	Thu	Fri	Sat	Week
31	1	2	3	4	5	6	23
7	8	9	10	11	12	13	24
14	15	16	17	18	19	20	25
21	22	23	24	25	26	27	26
28	29	30	1	2	3	4	27
5	6	7	8	9	10	11	28

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.

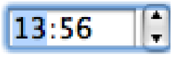
The “Del” key will set the date to the Null value (!00/00/00!) and dismiss the popup.

Time

As with dates, there are two types of control you can use with time data: inline and popup.

Inline Time Control

An inline time control looks like this when the data is being edited:



13:56

Select the value you want to change (hours or minutes) and click the up or down arrows to choose the required value. You can also enter the data manually - you won't be allowed to enter invalid values.

To specify a time control, use the ALP_Area_UseTimeControls property - for example:

AL_SetAreaLongProperty (area;ALP_Area_UseTimeControls;1)

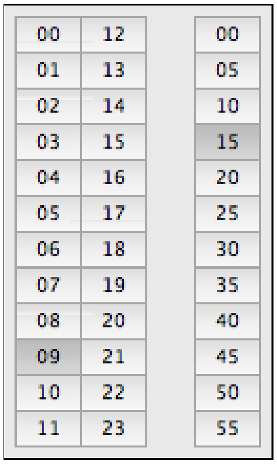
Note: this property applies to the entire area, not individual columns.

Popup Time Control

When you specify a popup time control, a little alarm clock icon appears in each cell in the column:

0

Click on the clock to open the popup time selector:



00	12	00
01	13	05
02	14	10
03	15	15
04	16	20
05	17	25
06	18	30
07	19	35
08	20	40
09	21	45
10	22	50
11	23	55

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:

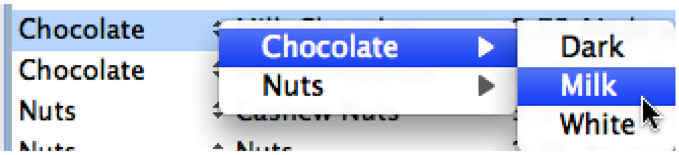
```
AL_SetColumnLongProperty (area;1;ALP_Column_Enterable;2)
```

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)
```

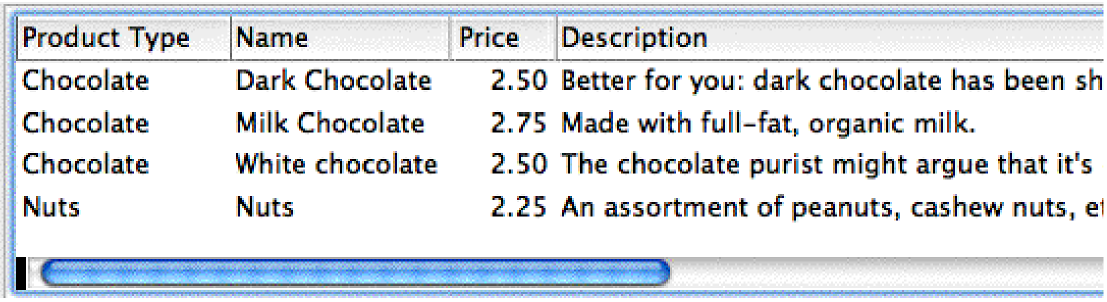
Grids

In addition to displaying fields and arrays in a spreadsheet-style “row and column” layout, AreaList Pro version 9 also enables you to display your data in **grids**.

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.

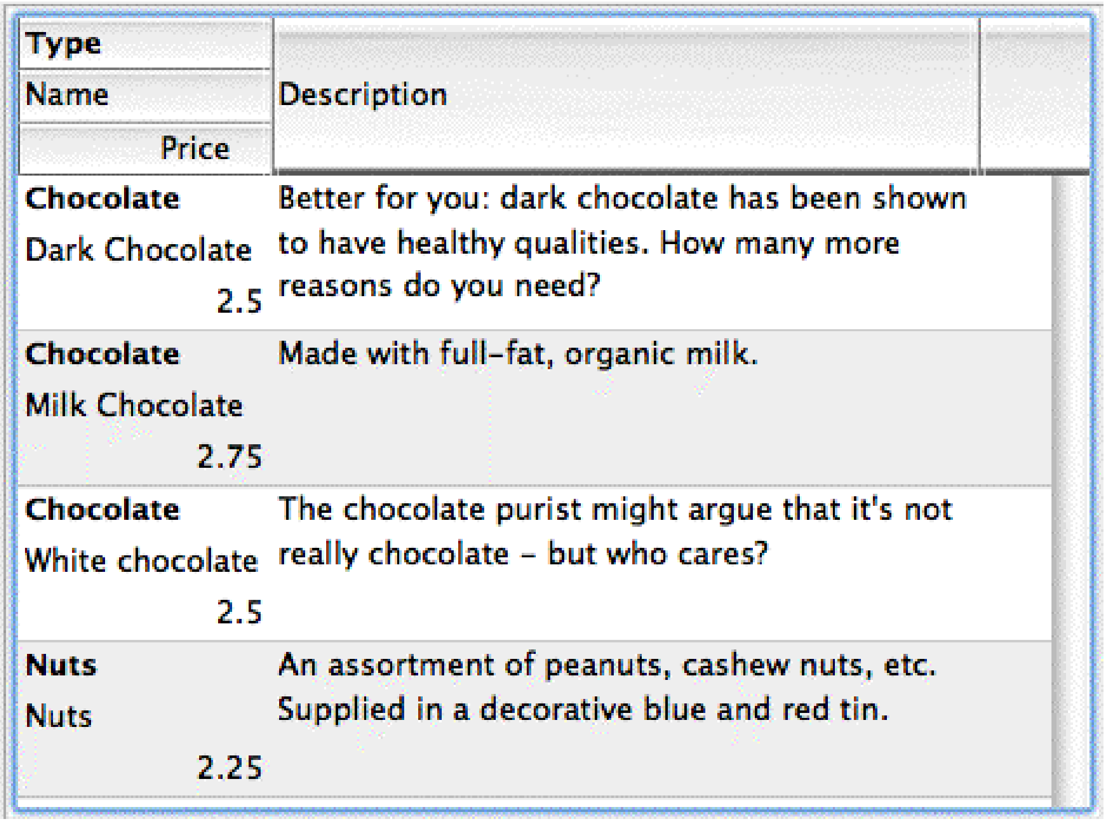
As an example, compare these two AreaList Pro areas:

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

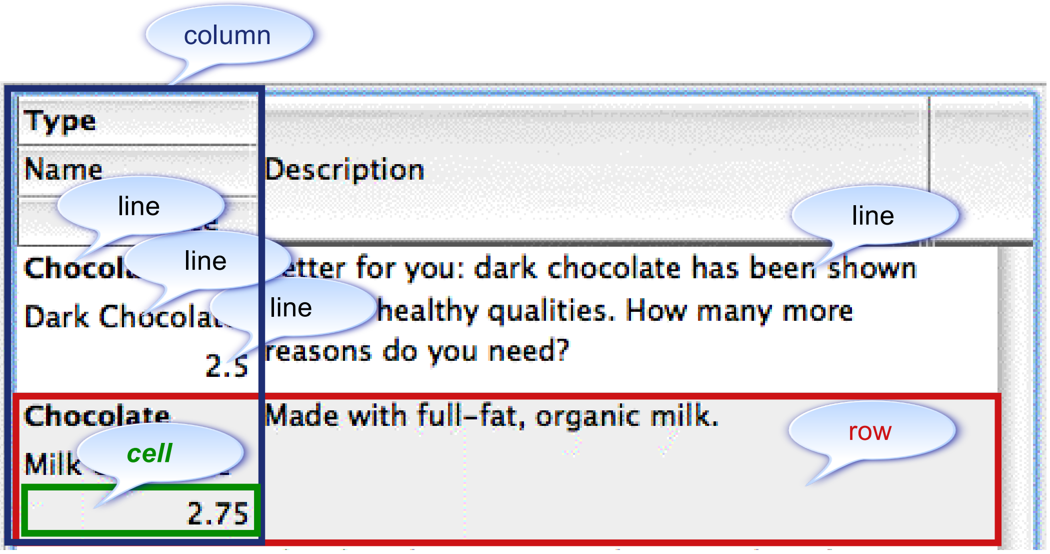
We refer to the parts of a grid as:

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**.



Type
Name
Description

Chocol
Dark Chocola
2.5
etter for you: dark chocolate has been shown
healthy qualities. How many more
reasons do you need?

Chocolate
Milk
2.75
Made with full-fat, organic milk.

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)  //alp, 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	✔	✔	✔	long int	-1	-1		Number of columns in grid
ALP_Area_ColsLocked	✔	✔	✔	long int	0	0		Number of locked columns in grid
ALP_Area_RowsInGrid	✔	✔	✔	long int	1	-1	20	Number of rows in grid
ALP_Area_EntryGotoGridCell	✔	✔		long int				Grid cell number to start entry in (cell in grid, not column number)
ALP_Area_EntryGridCell	✔			long int				Grid cell number of edited cell
ALP_Area_EntryPrevGridCell	✔			long 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	✔	✔	✔	long int				Find the first grid cell number showing data from the column
ALP_Column_FromCell	✔			long int				Get the column number from the grid cell number

Object Properties

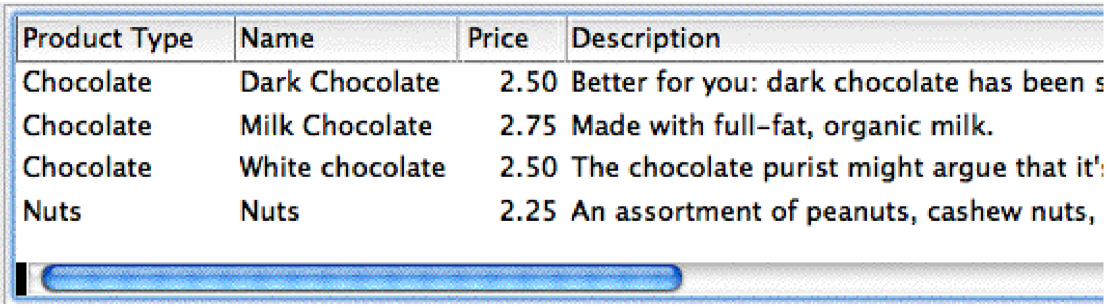
These properties are used with commands in the Objects theme.

Constant	Get	Set	Array Type	Comments
ALP_Object_FooterTextNH	✔	✔	text	Footer text of visible columns in grid order
ALP_Object_Grid	✔		long int	Column numbers Use a 2D array to access colSpan & rowSpan, too

Hierarchical Lists

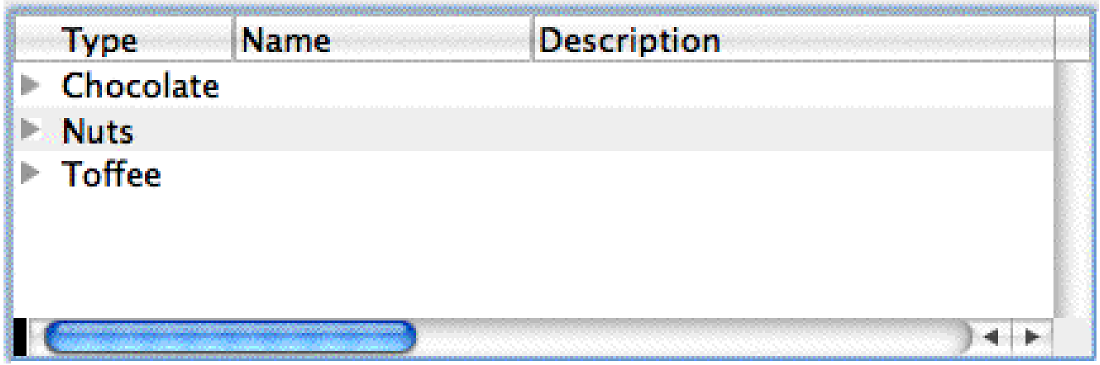
Hierarchical lists are an excellent way to display data that is organised into groups and sub-groups. Consider these two examples:

List Style



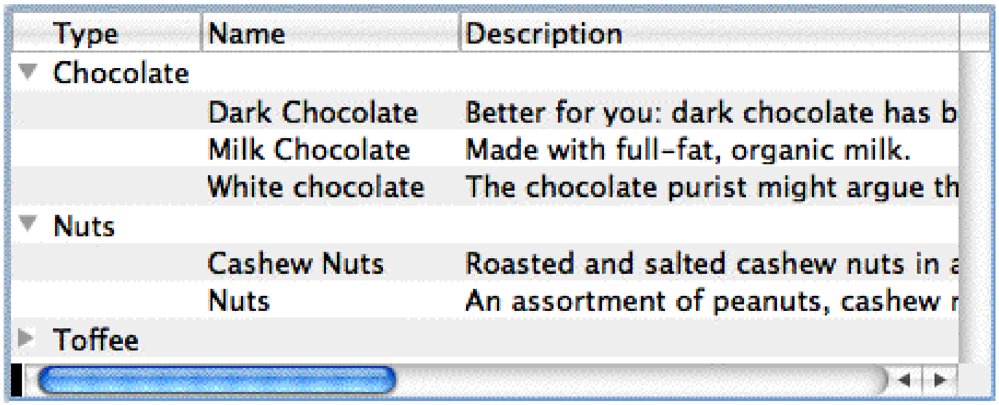
Product Type	Name	Price	Description
Chocolate	Dark Chocolate	2.50	Better for you: dark chocolate has been s
Chocolate	Milk Chocolate	2.75	Made with full-fat, organic milk.
Chocolate	White chocolate	2.50	The chocolate purist might argue that it'
Nuts	Nuts	2.25	An assortment of peanuts, cashew nuts,

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

Hierarchical lists consist of normal AreaList Pro columns (from arrays or fields) plus two additional arrays which specify the hierarchy level and the expansion status of each row.

Hierarchy Level

The hierarchy level is an integer where the top level =0 and successive levels are 1 more. So, in a three-level hierarchy you'll have level numbers 0, 1, and 2.

Expansion status

The expansion status is either 1 or 0: 0 means collapsed (closed) and 1 means expanded.

Here is a simple example to illustrate the way the two additional arrays work:

Data column 1: Product type	Data column 2: Product name	Hierarchy level array	Expansion status array
Chocolate		0	1
	Milk chocolate	1	0
	Dark chocolate	1	0
Nuts		0	1
	Cashews	1	0
	Pecans	1	0

The hierarchical list is initiated by calling the ***AL_SetObjects2*** command with the ALP_Object_Hierarchy parameter, the hierarchy level array, and the expansion status array:

```
$err:=AL_SetObjects2 (area;ALP_Object_Hierarchy;$aiLevel;$aiExpanded)
```

Alternatively, you can call ***AL_SetObjects*** with a two-dimensional array.

Example

In this example, we are going to create the hierarchy shown at the beginning of this topic.

We have a number of product records which each have a Product Type selected (Chocolate, Nuts, etc.).

We will need to:

1. Select the records to include
2. Sort them first by Type and then by Product Name

3. Build the Hierarchy Level array, inserting elements into the arrays for each product type break.

```
// Select the records, sort them, and copy them into arrays:
ALL RECORDS([product])
ORDER BY([product];[product]product_type;[product]product_name)
SELECTION TO ARRAY([product]product_type;atType;[product]product_name;atName;
[product]description;atDescription;[product]retail_price;arPrice)
// Add columns to the AreaList Pro area:
$err:=AL_AddColumn (area;->atType)
$err:=AL_AddColumn (area;->atName)
$err:=AL_AddColumn (area;->atDescription)
$err:=AL_AddColumn (area;->arPrice)
// Create the hierarchy level array. It is initially 1 element bigger than the (current) size of the Product arrays, because we know
that the first row will be the first top-level entry in the hierarchy:
ARRAY LONGINT($aiLevel;Size of array(atType)+1)
// Insert an empty element at the beginning of each of the Product arrays:
INSERT IN ARRAY(atType;1;1)
INSERT IN ARRAY(atName;1;1)
INSERT IN ARRAY(atDescription;1;1)
INSERT IN ARRAY(arPrice;1;1)
// Set up the first row:
$type:=atType{2}  // $type will tell us when the type changes
atType{1}:=$type  // Set the first level topic
$i:=2  // Start the loop at the second row
// Loop through the product arrays, inserting elements when the Type changes:
While ($i<=Size of array(atType))
   If (atType{$i}=$type)  // same type as the previous row
      $aiLevel{$i}:=1
      atType{$i}:=""  // don't need to redisplay the type
   Else
      $type:=atType{$i}  // reset the type variable
      INSERT IN ARRAY($aiLevel;$i;1)
      INSERT IN ARRAY(atType;$i;1)
      INSERT IN ARRAY(atName;$i;1)
      INSERT IN ARRAY(atDescription;$i;1)
      INSERT IN ARRAY(arPrice;$i;1)
      atType{$i}:=$type  // this is the level 1 row
      $aiLevel{$i}:=0  // not strictly necessary, as it will default to 0
      $aiLevel{$i+1}:=1
      atType{$i+1}:=""
      $i:=$i+1
   End if
$i:=$i+1
End while
```

```
//Create the expansion status array:
ARRAY LONGINT($aiExpanded;Size of array($aiLevel))  //0 = not expanded
//Set the hierarchy:
$err:=AL_SetObjects2 (area;ALP_Object_Hierarchy;$aiLevel;$aiExpanded)
//Set the column headers:
AL_SetColumnTextProperty (area;1;ALP_Column_HeaderText;"Type")
AL_SetColumnTextProperty (area;2;ALP_Column_HeaderText;"Name")
AL_SetColumnTextProperty (area;3;ALP_Column_HeaderText;"Description")
AL_SetColumnTextProperty (area;4;ALP_Column_HeaderText;"Price")
//Finally, apply some formatting:
AL_SetAreaLongProperty (area;ALP_Area_AltRowOptions;1)  //alternate row background
```

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	✓			long int				Returns the level associated with this row (set using ALP_Object_Hierarchy)
ALP_Row_Parent	✓			long 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	✓	✓	long 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.

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

Set the header row to a height of 40 points and display an icon in the third column; 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
```

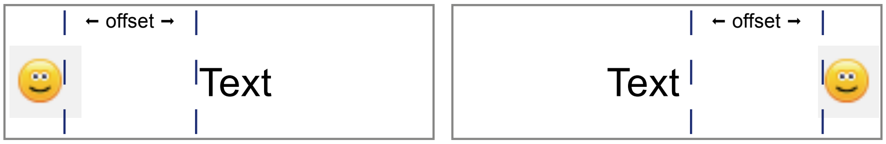
Note: if ***AL_SetIcon*** is not called, AreaList Pro will look for pictures ID 33, 34 and 35 from 4D's picture library.

Alignment and offset

AreaList Pro uses a specific offset/width implementation for icon drawing.

Offset

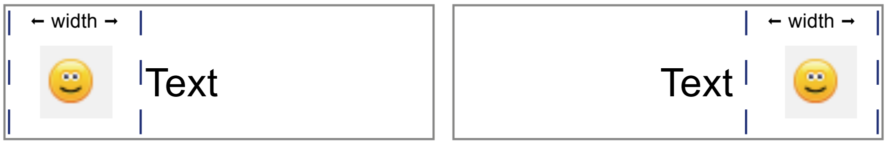
When horizontal alignment in ALP_Cell_XXXIconFlags is zero (default position : left for left icon, right for right icon), the low 8 bits (offset/width) of ALP_Cell_XXXIconFlags are interpreted as the offset, i.e. the distance in points between the text and the icon (left or right):



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:



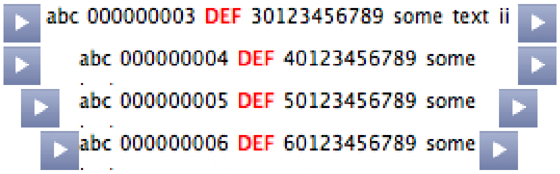
512 - AL Icon Flags Horizontal Center - Horizontally centered

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:



```
abc 000000003 DEF 30123456789 some text ii
abc 000000004 DEF 40123456789 some
abc 000000005 DEF 50123456789 some
abc 000000006 DEF 60123456789 some
```

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).

Note: currently the replacement for internal icons must match the size. This will be enhanced in future releases.

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 though 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)
```

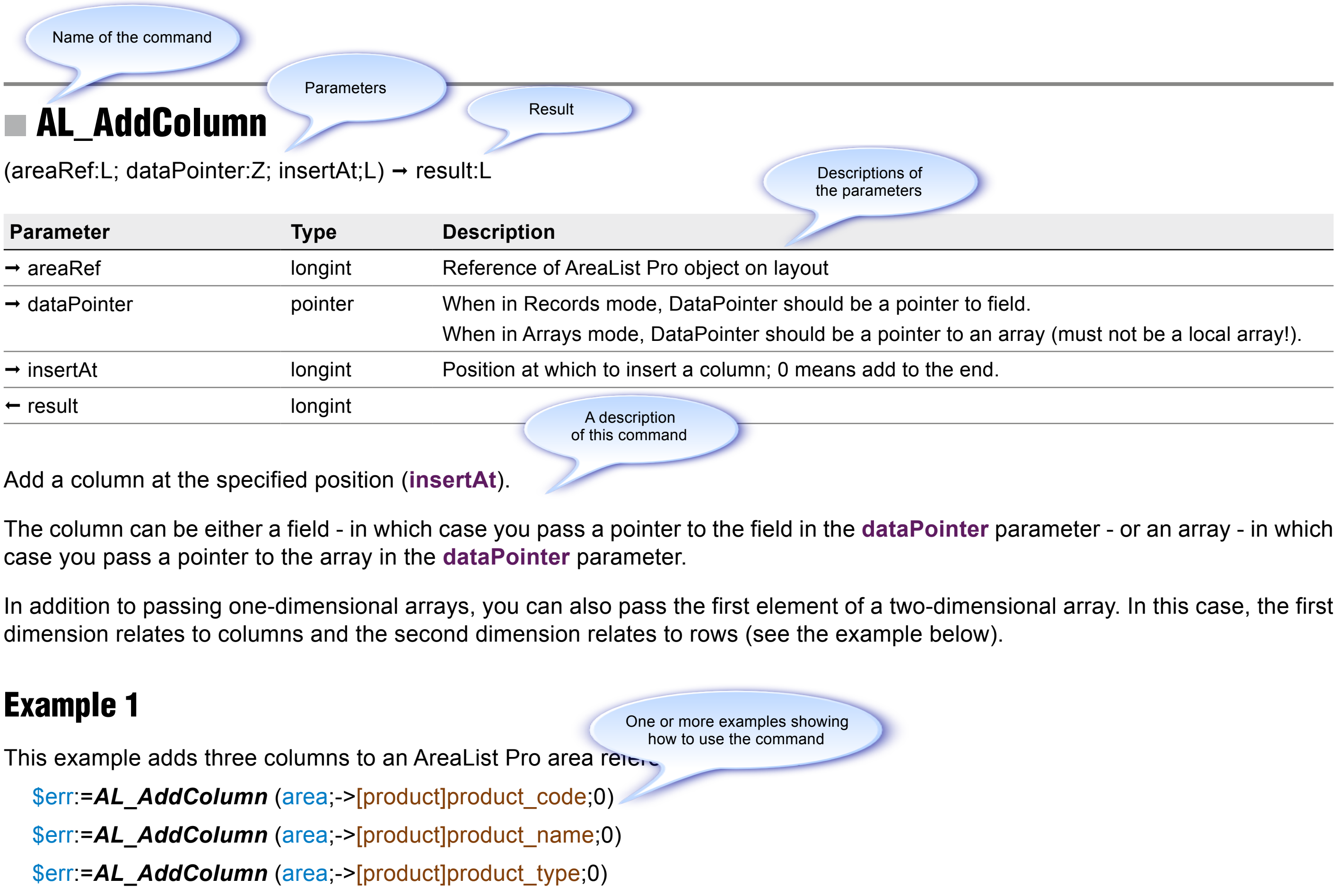
And then just use that menu:

```
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
```

12 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
Parameters
Result
Descriptions of the parameters
A description of this command
One or more examples showing how to use the command

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 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	

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 1

This example adds three columns to an AreaList Pro area refere

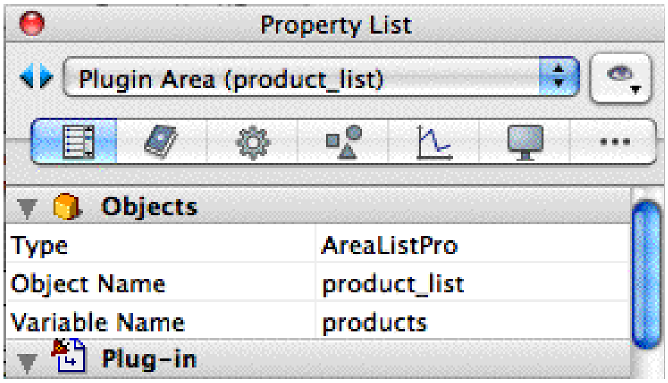
```
$err:=AL_AddColumn (area;->[product]product_code;0)
$err:=AL_AddColumn (area;->[product]product_name;0)
$err:=AL_AddColumn (area;->[product]product_type;0)
```

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:



Type	AreaListPro
Object Name	product_list
Variable Name	products

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 six 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

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.

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.

- AreaRef = 0 means “access workstation global settings”.
- AreaRef # 0 means “access this area’s 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_AddCalculatedColumn

(areaRef:L; dataType:L; callbackMethodName:T; insertAt:L) → result:L

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>→ areaRef</td><td>longint</td><td>Reference of AreaList Pro object on layout.</td></tr>
<tr><td>→ dataType</td><td>longint</td><td>The array type to use:
<table>
<tr><th>Array Type</th><th>Constant</th><th>Value</th></tr>
<tr><td>Alpha</td><td>Is Alpha Field</td><td>0</td></tr>
<tr><td>Boolean</td><td>Is Boolean</td><td>6</td></tr>
<tr><td>Date</td><td>Is Date</td><td>4</td></tr>
<tr><td>Integer</td><td>Is Integer</td><td>8</td></tr>
<tr><td>Longint</td><td>Is Longint</td><td>9</td></tr>
<tr><td>Picture</td><td>Is Picture</td><td>3 or 10</td></tr>
<tr><td>Real</td><td>Is Real</td><td>1</td></tr>
<tr><td>Text</td><td>Is Text</td><td>2</td></tr>
<tr><td>Time</td><td>Is Time</td><td>11</td></tr>
<tr><td>Universal date/time</td><td></td><td>Mapped to 2 (Text)</td></tr>
</table></td></tr>
<tr><td>→ callbackMethodName</td><td>text</td><td>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</td></tr>
<tr><td>→ insertAt</td><td>longint</td><td>Position at which to insert a column; 0 means add to the end.</td></tr>
<tr><td>← result</td><td>longint</td><td></td></tr>
</table>

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_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

To find out the number of columns in an area:

```
C_LONGINT($numColumns)
$err:=AL_GetAreaPtrProperty(area; ALP_Area_Columns;->$numColumns)
```

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_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_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

To show the sort editor:

```
$showSortEditor:=1
$error:=AL_SetAreaPtrProperty (area;ALP_Area_ShowSortEditor;->$showSortEditor)
```

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.

For some of the Column properties (mainly style properties), you can use 0 as the Column Number to accessing the default values for newly created (or re-initialized) columns. If the Column Number is -2, the property will be applied to all existing columns (from 1 to ALP_Area_Columns).

■ 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_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_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.

If the Row Number is -2, properties that can be set will be applied to all existing rows (from row #1 to row #ALP_Area_Rows).

See also 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.

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_Rows) for the specified Row Number.

See examples in the AreaList Pro Cell Properties section.

■ 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 180° (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

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.

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

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)
AL_GetObjects2 (area;ALP_Object_Fields;$arrayTableNos;$arrayFieldNos)
```

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)
```

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_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 e-Node's 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 e-Node’s 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 e-Node's 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

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.

13 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

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=1 or False=0)

Longint: a long integer

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

AreaList Pro Area Properties

Use these properties with commands in the Area command theme:

AL_AddCalculatedColumn

AL_AddColumn

AL_GetAreaLongProperty

AL_GetAreaPtrProperty

AL_GetAreaRealProperty

AL_GetAreaTextProperty

AL_SetAreaLongProperty

AL_SetAreaPtrProperty

AL_SetAreaRealProperty

AL_SetAreaTextProperty

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.

- AreaRef = 0 means “access workstation global settings”.
- AreaRef # 0 means “access this area’s 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.

AreaList Pro Area General Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
					AreaList Pro Area General Properties			
ALP_Area_ArrowsForHierarchy	✓	✓		bool	false (0)			When hierarchy is displayed, left/right arrow keys (without command key) are used to collapse/ expand nodes, not for horizontal scrolling
ALP_Area_AutoResizeColumn	✓	✓	✓	long 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 See Column Automatic Resize
ALP_Area_AutoSnapLastColumn	✓	✓	✓	bool	false (0)			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) See Column Automatic Resize
ALP_Area_Compatibility	✓	✓	✓	bool	depends			AreaList Pro 8.x compatibility mode (defaut value depends on initialization)
ALP_Area_CompHideCols	✓	✓	✓	long int	0	0		Number of columns to hide (compatibility mode only) See Hiding columns
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_DontSetCursor	✓	✓		bool	false (0)			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_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_ListWidth	✓			real				Width of the list area, not including the frame and scrollbars
ALP_Area_MoveCellOptions	✓	✓	✓	bool				Move cell options with cells (on sort)
ALP_Area_MoveRowOptions	✓	✓	✓	bool				Move row options with rows (on sort and drag)
ALP_Area_Name	✓	✓	✓	text				Name of the area (if empty, the variable name is initialized from the form variable name in design mode)

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area General Properties								
ALP_Area_ReadOnly	✔	✔		long int	0	0	7	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
ALP_Area_Redraw		✔		n/a				Redraw the area object (redraw is immediate)
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_SRPTableTemplate	✔			text				Get the SuperReport Pro template used for report creation (stored in Resources/Table Report.xml) as XML
ALP_Area_UserBLOB	✔	✔	✔	BLOB				BLOB for free use by developer
ALP_Area_Visible	✔	✔		bool				Area is visible **Set to false (0) before showing another dialog over AreaList Pro or DropArea to hide scrollbars**
ALP_Area_WindowsClip	✔	✔		bool	true (1)			Area clipping mode on Windows If set to false, the area will not be clipped so that other 4D objects can be positioned over the area Set to false (0) in case you want to use **FORM SCREENSHOT** on Windows May be used with the area reference set to zero (newly created areas will use this mode)
ALP_Area_WindowsText	✔	✔		bool	false (0)			1 = change the engine used for drawing on Windows to GDI drawing 0 (default) = use GDI+ 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 May be used with the area reference set to zero (newly created areas will use this mode)
ALP_Area_XML	✔	✔		text				Full description of the area in XML
ALP_Area_XMLAP	✔			text				Advanced properties from design

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	false (0)			Using Edit->Copy or a row drag, clipboard will contain all columns if set to true
ALP_Area_CopyOptions	✔	✔	✔	bool	false (0)			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 Data Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Data Properties				
ALP_Area_CacheSize	✔	✔	✔	long int	1024	128		Cache size (in number of rows to cache). Can be used with area=0 to access workstation default (used for initialization of new areas)
ALP_Area_CheckData		✔						Check array/selection size and fill the cache with data The number of rows is checked only if explicitly requested with this property
ALP_Area_ClearCache		✔		long int				Clear cells cache and refreshes data Set value to a row number to refresh only that row (see Row Numbering) Set value to -2 to refresh all rows
ALP_Area_Columns	✔		✔	long int				Current number of columns.
ALP_Area_FillCache		✔						Fill the data cache. AreaList Pro 9 uses a true cache for data (at least for the rows on screen) This property will invoke the cache-filling routine If the number of rows is less than the cache size (ALP_Area_CacheSize), all rows are loaded into the cache. Otherwise 128 rows are loaded Note: use only after clearing some/all cached rows – visible rows are refreshed Happens automatically on Update event
ALP_Area_Rows	✔			long int				Current number of rows
ALP_Area_TableID	✔	✔	✔	long int	0			Main table number Zero when showing arrays >0 when showing fields See also No fields from local table in field mode

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Data Properties								
ALP_Area_UpdateData		✔						Clear cells cache, check for selection size change, fill cells cache

AreaList Pro Area Display Properties

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	✔	✔	✔	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_BottomRow	✔			long int				The row number of the last (possibly partially) visible row on screen
ALP_Area_CalcAllRows	✔	✔		long 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	✔	✔	✔	long int	-1	-1		Number of columns in grid
ALP_Area_ColsLocked	✔	✔	✔	long int	0	0		Number of locked columns in grid
ALP_Area_ColumnLock	✔	✔	✔	bool	true (1)			Allow column lock by user
ALP_Area_ColumnResize	✔	✔	✔	bool	true (1)			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_DrawFrame	✔	✔	✔	long int	0	0	2	Draw frame: 0 = none 1 = black rectangle 2 = modern look (sunken)
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

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Display Properties				
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_HeaderMode	✔	✔	✔	long 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	false (0)			Hide headers
ALP_Area_HierIndent	✔	✔	✔	real	16	0	64	Indent increment for every hierarchy level (for use with hierarchical lists)
ALP_Area_LimitRows	✔	✔		long 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_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 OBSOLETE: this area is not customizable in AreaList Pro v9 (the header is drawn)
ALP_Area_MiscColor2	✔	✔	✔	color	#FFEEEEEE			Area color below the vertical scrollbar MODIFIED: this area is not customizable in AreaList Pro v9 (the scrollbar is drawn, and it is bigger than in 8.x) In 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_NumFtrLines	✔	✔	✔	long int	1	0	64	Number of lines within the footer 0 = variable height
ALP_Area_NumHdrLines	✔	✔	✔	long int	1	0	64	Number of lines within the header 0 = variable height
ALP_Area_NumRowLines	✔	✔	✔	long int	1	0	64	Number of lines within the row 0 = variable height
ALP_Area_ResizeDuring	✔	✔	✔	bool	false (0)			
ALP_Area_RowDivColor	✔	✔	✔	color	#FF808080			Row divider color (default is gray)
ALP_Area_RowHeight	✔			real				Initial height of every row

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
								AreaList Pro Area Display Properties
ALP_Area_RowHeightFixed	✔	✔	✔	bool	true (1)			Deprecated, superseded by ALP_Area_NumRowLines / ALP_Area_NumHdrLines / ALP_Area_NumFtrLines with value(s) 0
ALP_Area_RowIndent	✔	✔	✔	point	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	✔	✔	✔	long int	1	-1	20	Number of rows in grid
ALP_Area_ShowColDividers	✔	✔	✔	long int	0	0	7	Show column dividers (using ALP_Area_ColDivColor): bit 0: draw over data and footer bit 1: hide footer dividers (bit 0 ignored if bits 0 and 2 off) bit 2: draw last column divide
ALP_Area_ShowFocus	✔	✔	✔	bool	true (1)			Show focus ring
ALP_Area_ShowFooters	✔	✔	✔	bool	false (0)			Show footers
ALP_Area_ShowHScroll	✔	✔	✔	long 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
ALP_Area_ShowRowDividers	✔	✔	✔	bool	false (0)			Show row dividers (using ALP_Area_RowDivColor)
ALP_Area_ShowSortIndicator	✔	✔	✔	long 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	✔	✔	✔	bool	true (1)			Show vertical scrollbar
ALP_Area_ShowWidths	✔	✔	✔	long int	0	0	2	Show column widths (and column number + column source) in a tooltip when 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) Useful when setting up an area and you want to set the column widths manually 0 = don’t display the width 1 = display (cmd-option-shift down + mouse pointer over a header cell) 2 = display, but only in interpreted mode (when host DB is compiled, same as 0; component can be compiled)
ALP_Area_SmallScrollbar	✔	✔	✔	bool	false (0)			Use small scrollbars
ALP_Area_TopRow	✔			long int				The row number of the first (possibly partially) visible row on screen

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Display Properties								
ALP_Area_UseEllipsis	✔	✔	✔	long 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

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	false (0)			Accept column drag when destination (this area) does not have ALP_Area_DragDstColCodes set OBSOLETE: use Drag access codes instead
ALP_Area_DragAcceptLine	✔	✔	✔	bool	false (0)			Accept line drag when destination (this area) does not have ALP_Area_DragDstRowCodes set OBSOLETE: use Drag access codes instead
ALP_Area_DragColumn	✔	✔	✔	long int	0	0	3	Allow a column drag when source (this area) does not have ALP_Area_DragSrcColCodes set: 0 = no 1 = inside area 2 = outside of area 3 = both inside and outside of area OBSOLETE: use Drag access codes instead
ALP_Area_DragDataType	✔			long int				Dragged data: 1 = row(s) 2 = column 3 = cell(s)
ALP_Area_DragDstArea	✔			long int				Destination area
ALP_Area_DragDstCell	✔			long 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	✔			long int				Destination area column
ALP_Area_DragDstColCodes	✔	✔	✔	text				Drag destination column codes The format is a list of codes separated by '|'
ALP_Area_DragDstProcessID	✔			long int				4D's process ID of destination area
ALP_Area_DragDstRow	✔			long int				Destination area row
ALP_Area_DragDstRowCodes	✔	✔	✔	text				Drag destination row codes The format is a list of codes separated by '|'

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Drag & Drop Properties				
ALP_Area_DragLine	✔	✔	✔	long int	0	0	6	Allow a line drag when source (this area) does not have ALP_Area_DragSrcRowCodes set: 0 = no 1 = inside area 2 = outside of area 3 = both inside and outside of area 4 = inside area w/o option key 5 = outside of area w/o option key 6 = both inside and outside of area w/o option key OBSOLETE: use Drag access codes instead
ALP_Area_DragOptionKey	✔	✔	✔	bool	false (0)			Drag rows using option key
ALP_Area_DragProcessID	✔			long int				4D's process ID of this area
ALP_Area_DragRowMultiple	✔	✔	✔	bool	false (0)			Allow multiple rows to be dragged
ALP_Area_DragRowOnto	✔	✔	✔	bool	true (1)			Show drag feedback: True: onto a row False: between rows
ALP_Area_DragScroll	✔	✔	✔	long int	30	0	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
ALP_Area_DragSrcArea	✔			long int				The dropped AreaList Pro area Zero if a different object was dropped (or AreaList Pro area from another application)
ALP_Area_DragSrcCell	✔			long 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	✔			long int				Source area column
ALP_Area_DragSrcColCodes	✔	✔	✔	text				Drag source column codes The format is a list of codes separated by '|'
ALP_Area_DragSrcRow	✔			long 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 DropArea Properties

AreaList Pro Drop Areas are obsolete, and are included for backwards compatibility. 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	true (1)			Accept column drag when source does not have ALP_Area_DragSrcColCodes set OBSOLETE: use Drag codes instead
ALP_Drop_DragAcceptLine	✓	✓	✓	bool	true (1)			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	✓			long int				4D's process ID of this area
ALP_Drop_DragSrcArea	✓			long 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

AreaList Pro Area Entry Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Entry Properties								
ALP_Area_ClickDelay	✓	✓	✓	long int	30	-2	300	Delay to start editing (in 1/60 sec.): -2 = use current double-click time 0 = disable May be used with the area reference set to 0 (newly created areas will use this mode)
ALP_Area_EntryAllowArrows	✓	✓	✓	bool	false (0)			Arrow keys are used to move to next entry cell
ALP_Area_EntryAllowReturn	✓	✓	✓	bool	false (0)			Allow RETURN in text
ALP_Area_EntryAllowSeconds	✓	✓	✓	bool	false (0)			Allow seconds in time entry
ALP_Area_EntryCell	✓			cell				Row and grid cell number of current entry (row, cell)
ALP_Area_EntryClick	✓	✓	✓	long 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 May be used with the area reference set to zero (newly created areas will use this mode)

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Entry Properties				
ALP_Area_EntryColumn	✔			long int				Column number of edited cell
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	✔	✔	✔	long 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	✔	✔		long int				Column number to start entry in (first grid cell showing this column will be used)
ALP_Area_EntryGotoGridCell	✔	✔		long int				Grid cell number to start entry in (cell in grid, not column number)
ALP_Area_EntryGotoRow	✔	✔		long int				Row number to start entry in
ALP_Area_EntryGridCell	✔			long int				Grid cell number of edited cell
ALP_Area_EntryHighlight	✔	✔		range				Entry highlight in the form: **String** ($startOfSelection)+"," +**String** ($endOfSelection)
ALP_Area_EntryHighlightE	✔	✔		long int				Entry highlight end
ALP_Area_EntryHighlightS	✔	✔		long int				Entry highlight start
ALP_Area_EntryInProgress	✔			bool				A cell is currently being edited
ALP_Area_EntryMapEnter	✔	✔	✔	long int	0	0	3	Map Enter key to: 0 = nothing (ignore) 1 = Tab 2 = Return 3 = Tab for text fields, Return otherwise
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	✔			long int				Previously edited column number
ALP_Area_EntryPrevGridCell	✔			long int				Previously edited grid cell number
ALP_Area_EntryPrevRow	✔			long int				Previously edited row number
ALP_Area_EntryRow	✔			long int				Row number of edited cell

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
				AreaList Pro Area Entry Properties				
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_IgnoreMenuMeta	✔	✔	✔	bool				Do not interpret meta characters when building popup menu May be used with the area reference set to 0 (newly created areas will use this mode)
ALP_Area_IgnoreSoftDeselect	✔	✔	✔	bool	false (0)			Ignore soft deselect (treat it as hard deselect) See the explanation in the Callbacks chapter May be used with the area reference set to 0 (newly created areas will use this mode)
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 See the example in the Callbacks section
ALP_Area_UseDateControls	✔	✔	✔	bool				Use native Date control for date entry Note: can’t be cleared to zero May be used with the area reference set to 0 (newly created areas will use this mode)
ALP_Area_UseTimeControls	✔	✔	✔	bool				Use native Time control for time entry May be used with the area reference set to 0 (newly created areas will use this mode)

AreaList Pro Area Event Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Event Properties								
ALP_Area_AlpEvent	✔			long 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	✔			long int				Last clicked grid cell
ALP_Area_ClickedCol	✔			long int				Last clicked column
ALP_Area_ClickedRow	✔			long int				Last clicked row See Row Numbering
ALP_Area_DoubleClick	✔			bool				Last click is double click
ALP_Area_Event	✔			long 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_EventChar	✔			text				Char (string) from keyboard
ALP_Area_Event_Filter	✔	✔		long int	0	0	1	Mask to define which events should not be reported Currently only bit 0 is defined: don’t report AL Mouse moved event (18) if set to 1
ALP_Area_EventKey	✔			text				Key code
ALP_Area_EventModifiers	✔			long int				Event modifiers: 256 = command 512 = shift 1024 = caps lock 2048 = option 4096 = control
ALP_Area_EventPosH	✔			long int				Horizontal mouse position
ALP_Area_EventPosV	✔			long int				Vertical mouse position

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
								AreaList Pro Area Event Properties
ALP_Area_RollOverCell	✔			long int				Current cell (where the mouse is positioned) See Example 15 in the Tutorial section
ALP_Area_RollOverCol	✔			long int				Current column (where the mouse is positioned)
ALP_Area_RollOverRow	✔			long int				Current row (where the mouse is positioned) See Row Numbering
ALP_Area_SendEvent	✔	✔		long 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

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)

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
					AreaList Pro Area Plugin Properties			
ALP_Area_CalendarLook	✓	✓		bool	false (0)			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 arrays 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 arrays 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_DefFmtInteger	✓	✓		text				Default format for Integer arrays 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_DefFmtLong	✓	✓		text				Default format for Long arrays 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_DefFmtPicture	✓	✓		text				Default format for Picture arrays 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_DefFmtReal	✓	✓		text				Default format for Real arrays 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_FillNumberSign	✓	✓		bool	false (0)			If set to 1 (True), string formatting replaces the "number sign" placeholders '#' with non-breaking space (old behavior), otherwise unused part of the format string is removed
ALP_Area_LastError	✓	✓		long int				Last error in ANY AreaList Pro area
ALP_Area_Path	✓			text				Path to the AreaList Pro plugin
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 bit 0: trace in interpreted (values 1 & 3) bit 1: alert in compiled (values 2 & 3)
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 Area Selection Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
								AreaList Pro Area Selection Properties
ALP_Area_SelClick	✔	✔	✔	long 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	✔	✔		long int				Selected column
ALP_Area_SelCreateSet	✔	✔		text	empty			Name of the set to create after selection change by user The area must be in fields mode and row selection mode This property was also called ALP_Area_SelSetName in some versions
ALP_Area_Select	✔			long int				Number of selected rows/cells (for multiple row/cell selection mode only)
ALP_Area_SelGotoRec	✔	✔		bool	false (0)			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	✔	✔	✔	long int	0	0	2	Change the way the selected cells/rows are highlighted: 0 = system highlight color used as background color (default) 1 = invert colors (legacy mode) 2 = blend the system highlight color with 75% alpha (192)
ALP_Area_TypeAheadEffect	✔	✔		long 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
ALP_Area_TypeAheadFieldMode	✔	✔		bool	false (0)			Set to True (1) to allow typeahead in field mode Note: when ALP_Area_TableID > 0 and ALP_Area_TypeAheadFieldMode = 0, typeahead is fully ignored
ALP_Area_TypeAheadString	✔	✔		text				This property returns the string typed by the user used in typeahead (after the typeahead changed the selection)

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Selection Properties								
ALP_Area_TypeAheadTime	✔	✔		long int	2 * double click time (from OS)	-1	300	Time in ticks (1/60 s) used to determine if a new typeahead is starting (maximum time between two keystrokes to be handled as one string) Note: setting to less than 5 will set the value to default
ALP_Area_SelMultiple	✔	✔	✔	bool	false (0)			Allow selection of multiple rows (in selection mode: “rows” = 0) Ignored if ALP_Area_SelType is not 0
ALP_Area_SelNoCtrlSelect	✔	✔	✔	bool	false (0)			Disable row selection when control-click occurs on unselected row
ALP_Area_SelNoDeselect	✔	✔	✔	bool	false (0)			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	false (0)			Disable row highlight of selected rows (in selection mode: “rows” = 0) Ignored if ALP_Area_SelType is not 0
ALP_Area_SelNoAutoSelect	✔	✔	✔	bool	false (0)			If set to true, a click on a popup in an unselected row does not select the row Automatically set to true when compatibility mode is turned on
ALP_Area_SelNone	✔	✔	✔	bool	false (0)			Allow no selection (in selection mode: “rows” = 0) Ignored if ALP_Area_SelType is not 0
ALP_Area_SelPreserve	✔	✔	✔	bool	false (0)			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	✔	✔		long int				Selected row: last clicked (or first row in selection when cmd-clicked to deselect the row in multiple row selection mode)
ALP_Area_SelType	✔	✔	✔	long int	0	0	2	Selection mode: 0 = rows 1 = single cell 2 = multiple cells

AreaList Pro Area Sort Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Sort Properties								
ALP_Area_AllowSortEditor	✔	✔	✔	bool	false (0)			Allow sort editor by user (cmd-click on header)
ALP_Area_DontSortArrays	✔	✔	✔	bool	false (0)			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 property ALP_Object_Sort See also the Internal Sorting topic
ALP_Area_ShowSortEditor	✔	✔						Show sort editor The ***AL_GetAreaLongProperty*** getter command returns 1 if the user clicked Sort, 0 otherwise

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
AreaList Pro Area Sort Properties								
ALP_Area_Sort	✔			long 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	✔		✔	long int				Sorted column number (use ALP_Area_SortList to sort)
ALP_Area_SortDuring	✔	✔	✔	bool	false (0)			Sets a flag for a “permanent” sort = ie, automatically keep the data sorted of it changes
ALP_Area_SortList	✔	✔	✔	text				Sort list The format is a comma-separated list of column numbers Use negative number for descending order
ALP_Area_SortListNS	✔	✔		text				“No sort” sort list See Taking control of the sort for more information and an example 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	false (0)			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	✔	✔	✔	long int	1	0	3	Sort by user: 0 = disabled 1 = enabled 2 = bypassed 3 = only indexed fields

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

For some of the Column properties (mainly style properties), you can use 0 as the Column Number to accessing the default values for newly created (or re-initialized) columns.

If the Column Number is -2, the property will be applied to all existing columns (from 1 to ALP_Area_Columns).

Column General Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Column General Properties								
ALP_Column_Attributed	✔	✔	✔	bool	false (0)			Uses attributed (multi-style) text: 0 = no, 1 = yes See also AreaList Pro Text Style Tags
ALP_Column_CalcHeight	✔	✔	✔	bool	false (0)			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_DisplayControl	✔	✔	✔	long 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 from the 4D Picture Library)
ALP_Column_Enterable	✔	✔	✔	long int	1	0	5	Enterablility: 0 = not enterable 1 = by keyboard 2 = by popup 3 = by keyboard & popup 4 = by popup ignoring menu meta 5 = by keyboard & popup ignoring menu meta
ALP_Column_EntryControl	✔	✔	✔	long 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

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
					Column General Properties			
ALP_Column_FindCell	✔	✔	✔	long int				Find the first grid cell number showing data from the column
ALP_Column_FooterText	✔	✔	✔	text				Footer text
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_FromCell	✔			long int				Get the column number from the grid cell number
ALP_Column_HeaderText	✔	✔	✔	text				Header text
ALP_Column_ID	✔		✔	long 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	✔			long 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_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	✔	✔	✔	long 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_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)

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Column General Properties								
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_Reveal	✔	✔		n/a				Reveal (make visible) this column
ALP_Column_SortDirection	✔		✔	long int	0	-1	1	Current sort direction
ALP_Column_ScrollTo	✔	✔		n/a				If visible, scroll the area to position this column on the left
ALP_Column_Source	✔		✔	text				Data source Array name or [MyTable]MyField
ALP_Column_Spellcheck	✔	✔	✔	bool				Not implemented
ALP_Column_Type	✔		✔	long 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_UserText	✔	✔	✔	text				Text for free use by developer (XML or other), associated to the column
ALP_Column_Visible	✔	✔	✔	bool	true (1)			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

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	false (0)			Header font style = bold
ALP_Column_HdrStyleF	✔	✔	✔	long int	0	0	7	Header font style, using 4D style constants (e.g. Bold, Italic, etc.)
ALP_Column_HdrStyleI	✔	✔	✔	bool	false (0)			Header font style = italic
ALP_Column_HdrStyleU	✔	✔	✔	bool	false (0)			Header font style = underlined
ALP_Column_HdrTextColor	✔	✔	✔	color	#FF000000			Header font color Default is black
ALP_Column_HdrVertAlign	✔	✔	✔	long int	2	0	3	Header vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Column_HdrWrap	✔	✔	✔	bool	false (0)			Wrap long lines in header

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	false (0)			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	false (0)			Footer font style = italic
ALP_Column_FtrStyleU	✓	✓	✓	bool	false (0)			Footer font style = underlined
ALP_Column_FtrTextColor	✓	✓	✓	color	#FF000000			Footer font color Default is black
ALP_Column_FtrVertAlign	✓	✓	✓	long int	2	0	3	Footer vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Column_FtrWrap	✓	✓	✓	bool	false (0)			Wrap long lines in footer

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	false (0)			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	false (0)			List font style = italic
ALP_Column_StyleU	✔	✔	✔	bool	false (0)			List font style = underlined
ALP_Column_TextColor	✔	✔	✔	color	#FF000000			List font color Default is black
ALP_Column_VertAlign	✔	✔	✔	long int	0	0	3	List vertical alignment 0 = default 1 = top 2 = center 3 = bottom
ALP_Column_Wrap	✔	✔	✔	bool	false (0)			Wrap long lines in list

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:

- if the row number is -2, a settable row property will be applied to all existing rows (from #1 to #ALP_Area_Rows)
- 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 (style is actually the only property)
ALP_Row_Height	✔			real	area's row height	0	32000	Height of the row in points
ALP_Row_Hide	✔	✔		bool				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 Hierarchy Properties

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Row Hierarchy Properties								
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	✔			long int				Returns the level associated with this row (set using ALP_Object_Hierarchy)
ALP_Row_Parent	✔			long 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

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	✔	✔	✔	long 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 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
ALP_Row_FontName	✔	✔	✔	text				Font name
ALP_Row_HorAlign	✔	✔	✔	long 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_Size	✔	✔	✔	real		4	128	Font size
ALP_Row_StyleB	✔	✔	✔	bool				Font style = bold
ALP_Row_StyleF	✔	✔	✔	long 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_VertAlign	✔	✔	✔	long int		0	3	Vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Row_Wrap	✔	✔	✔	bool				Wrap long lines

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

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	✓	✓	✓	long int				Bottom border offset in points
ALP_Cell_BottomBorderWidth	✓	✓	✓	long int				Bottom border width in points
ALP_Cell_Clear		✓		n/a				Clear all properties of this cell The area redraws automatically
ALP_Cell_Enterable	✓	✓	✓	long int	-1	-1	5	Enterability: -1 = use column default (not set) 0 = not enterable 1 = by keyboard 2 = by popup 3 = by keyboard & popup 4 = by popup ignoring menu meta 5 = by keyboard & popup ignoring menu meta

Constant	Get	Set	Per	Type	Default	Min	Max	Comments
Cell General Properties								
ALP_Cell_FillColor	✔	✔	✔	color				Color used to fill the border rectangle
ALP_Cell_FormattedValue	✔			text				Formatted cell value
ALP_Cell_Invisible	✔	✔	✔	bool	false			If set to true (1), 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	✔	✔	✔	long int				Left border offset
ALP_Cell_LeftBorderWidth	✔	✔	✔	long int				Left border width
ALP_Cell_LeftIconFlags	✔	✔	✔	long int				Offset/width Horizontal position Vertical position Scaling Mask *See the section on Icon Flags for more*
ALP_Cell_LeftIconID	✔	✔	✔	long int				Left icon ID (see AL_SetIcon)
ALP_Cell_Reveal		✔		n/a				Reveal (make visible) this cell
ALP_Cell_RightBorderColor	✔	✔	✔	color				Right border color
ALP_Cell_RightBorderOffset	✔	✔	✔	long int				Right border offset in points
ALP_Cell_RightBorderWidth	✔	✔	✔	long int				Right border width in points
ALP_Cell_RightIconFlags	✔	✔	✔	long int				Offset/width Horizontal position Vertical position Scaling Mask *See the section on Icon Flags for more*
ALP_Cell_RightIconID	✔	✔	✔	long int				Right icon ID (see AL_SetIcon)
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	✔	✔	✔	long int				Top border offset in points
ALP_Cell_TopBorderWidth	✔	✔	✔	long 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
ALP_Cell_Flags	✔	✔	✔	long 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 Maintained by AreaList Pro and should not normally be changed 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	✔	✔	✔	long 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	✔	✔	✔	long 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_VertAlign	✔	✔	✔	long int		0	3	Vertical alignment: 0 = default 1 = top 2 = center 3 = bottom
ALP_Cell_Wrap	✔	✔	✔	bool				Wrap long lines

AreaList Pro Object Properties

Use these properties with commands in the Objects command theme:

AL_GetObjects

AL_GetObjects2

AL_SetObjects

AL_SetObjects2

None of these properties can be persistent.

Constant	Get	Set	Array Type	Comments
				Object Properties
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 (or auto-resizes) 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_Fields	✓		long 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_Grid	✓	✓	long 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	✓	✓	long 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	Set to 1 to hide a row
ALP_Object_RowSelection	✓	✓	long 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

Constant	Get	Set	Array Type	Comments
Object Properties				
ALP_Object_Selection	✔	✔	long int	Selection 2D or two arrays: you can call ***AL_SetObjects*** with a 2-dimensional array or ***AL_SetObjects2*** with two arrays if the selection mode is not row selection
ALP_Object_Sort	✔		long int	Sort order = to be used with ALP_Area_DontSortArrays
ALP_Object_SortList	✔	✔	long int	Sort list = use negative number for descending order
ALP_Object_SortListNS	✔	✔	long 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	✔		long 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	✔	✔	boolean, integer or longint	Visible status of an object (e.g. column) For ***AL_SetObjects***: if the array is shorter than the number of columns, the remaining columns visible status will not be modified

14

Mapping Old Commands to the AreaList Pro v9 API

The following table lists all the legacy (pre-version 9) AreaList Pro commands and describes which new command should be used in place of each one. See the v8.5 manual for legacy commands documentation.

Where appropriate, the new commands are shown with the property that should be used.

Note: mapping of colors to the v9 format is supported, but patterns are no longer supported. They are interpreted by AreaList Pro version 9 as transparency ratios. See Patterns.

The old commands will continue to work (other than those which are now obsolete), so you don't have to update all your AreaList Pro code, although it is recommended that all new or generic code is mapped to the v9 API.

For example: ***AL_GetCellEnter*** would be replaced with the new command ***AL_GetCellPtrProperty*** and the property ALP_Cell_Enterable**:**

```
C_LONGINT($Row;$Column;$Result)
$err:=AL_GetCellPtrProperty (AreaRef;$Row;$Column;ALP_Cell_Enterable;->$Result)
```

In some cases there is more than one way to replace the old command.

Old commands	New Commands and Properties
_DatePopupArea	Unsupported
AL_DoWinResize	Obsolete
AL_DragMgrAvail	Not relevant; always true
AL_DropArea	AL_DropArea
AL_ExitCell	AL_SetAreaLongProperty: ALP_Area_EntryExit
AL_GetAdvProps	Unsupported (internal)
AL_GetAreaBLOB	AL_GetAreaPtrProperty: ALP_Area_UserBLOB
AL_GetAreaName	AL_GetAreaTextProperty: ALP_Area_Name
AL_GetArrayNames	AL_GetColumnTextProperty: ALP_Column_Source AL_GetAreaLongProperty: ALP_Area_TableID AL_GetObjects: ALP_Object_Columns
AL_GetCellColor	AL_GetCellLongProperty: ALP_Cell_TextColor, ALP_Cell_BackColor (using any color format, see Working with Colors)
AL_GetCellEnter	AL_GetCellLongProperty: ALP_Cell_Enterable
AL_GetCellHigh	AL_GetAreaTextProperty or AL_GetAreaPtrProperty: ALP_Area_EntryHighlight, ALP_Area_EntryHighlightS, ALP_Area_EntryHighlightE
AL_GetCellMod	AL_GetAreaLongProperty: ALP_Area_EntryModified
AL_GetCellOpts	AL_GetAreaLongProperty: ALP_Area_SelType, ALP_Area_MoveCellOptions

Old commands	New Commands and Properties
AL_GetCellRGBColor	AL_GetCellLongProperty: ALP_Cell_TextColor, ALP_Cell_BackColor • Using any color format, see Working with Colors
AL_GetCellSel	AL_GetObjects: ALP_Object_Selection
AL_GetCellStyle	AL_GetCellTextProperty: ALP_Cell_FontName, AL_GetCellLongProperty: ALP_Cell_StyleF
AL_GetCellValue	AL_GetCellPtrProperty: ALP_Cell_Value
AL_GetClickedRow	AL_GetAreaLongProperty: ALP_Area_ClickedRow
AL_GetColLock	AL_GetAreaLongProperty: ALP_Area_ColsLocked
AL_GetColOpts	AL_GetAreaLongProperty: ALP_Area_ColumnResize, ALP_Area_ResizeDuring, ALP_Area_ColumnLock, ALP_Area_ShowWidths, ALP_Area_DragColumn, ALP_Area_DragAcceptColumn AL_GetColumnLongProperty: ALP_Column_Visible for column hiding
AL_GetColumn	AL_GetAreaLongProperty: ALP_Area_ClickedCol
AL_GetCopyOpts	AL_GetAreaLongProperty: ALP_Area_CopyHiddenCols, ALP_Area_CopyFieldSep, ALP_Area_CopyRecordSep, AL_GetAreaTextProperty: ALP_Area_CopyFieldWrapper • New API uses Text, not single character
AL_GetCurrCell	AL_GetAreaLongProperty: ALP_Area_EntryRow, ALP_Area_EntryColumn
AL_GetDragCol	AL_GetAreaLongProperty: ALP_Area_DragSrcCol, ALP_Area_DragDstCol, ALP_Area_DragDstArea
AL_GetDragLine	AL_GetAreaLongProperty: ALP_Area_DragSrcRow, ALP_Area_DragDstRow, ALP_Area_DragDstArea
AL_GetDrgArea	AL_GetAreaLongProperty: ALP_Area_DragDstArea, ALP_Area_DragDstProcessID
AL_GetDrgDstCol	AL_GetAreaLongProperty: ALP_Area_DragDstCol
AL_GetDrgDstRow	AL_GetAreaLongProperty: ALP_Area_DragDstRow
AL_GetDrgDstTyp	AL_GetAreaLongProperty: ALP_Area_DragDataType
AL_GetDrgSrcCol	AL_GetAreaLongProperty: ALP_Area_DragSrcCol
AL_GetDrgSrcRow	AL_GetAreaLongProperty: ALP_Area_DragSrcRow
AL_GetEditedText	AL_GetAreaTextProperty: ALP_Area_EntryText/ALP_Area_EntrySelectedText
AL_GetFields	AL_GetColumnLongProperty: ALP_Column_Source, AL_GetAreaLongProperty: ALP_Area_TableID AL_GetObjects: ALP_Object_Columns (using [table]field format)/ ALP_Object_Fields
AL_GetFile	AL_GetAreaLongProperty: ALP_Area_TableID
AL_GetFooters	AL_GetColumnTextProperty: ALP_Column_FooterText AL_GetAreaLongProperty :ALP_Column_Visible AL_GetObjects: ALP_Object_FooterText/ALP_Object_FooterTextNH
AL_GetFormat	AL_GetColumnLongProperty: ALP_Column_HorAlign, ALP_Column_HdrHorAlign, ALP_Column_FtrHorAlign, ALP_Column_CalcHeight
AL_GetFtrStyle	AL_GetColumnTextProperty: ALP_Column_FtrFontName AL_GetColumnRealProperty: ALP_Column_FtrSize AL_GetColumnLongProperty: ALP_Column_FtrStyleF
AL_GetHdrStyle	AL_GetColumnTextProperty: ALP_Column_HdrFontName AL_GetColumnRealProperty:ALP_Column_HdrSize AL_GetColumnLongProperty: ALP_Column_HdrStyleF
AL_GetHeaderOptions	Unsupported
AL_GetHeaders	AL_GetColumnTextProperty: ALP_Column_HeaderText AL_GetColumnLongProperty: ALP_Column_Visible AL_GetObjects: ALP_Object_HeaderText/ALP_Object_HeaderTextNH
AL_GetLastEvent	AL_GetAreaLongProperty: ALP_Area_AlpEvent
AL_GetLine	AL_GetAreaLongProperty: ALP_Area_SelRow
AL_GetMiscOpts	AL_GetAreaLongProperty: ALP_Area_HideHeaders, ALP_Area_ShowFocus, ALP_Area_ShowFooters postKey and useModernLook are unsupported

Old commands	New Commands and Properties
AL_GetMode	AL_GetAreaPtrProperty: ALP_Area_TableID
AL_GetPictEscape	Unsupported
AL_GetPluginPath	AL_GetAreaTextProperty: ALP_Area_Path
AL_GetPrevCell	AL_GetAreaLongProperty: ALP_Area_EntryPrevRow, ALP_Area_EntryPrevColumn
AL_GetRowOpts	AL_GetAreaLongProperty: ALP_Area_SelMultiple, ALP_Area_SelNone, ALP_Area_DragLine, ALP_Area_DragAcceptLine, ALP_Area_MoveRowOptions, ALP_Area_SelNoHighlight
AL_GetScroll	AL_GetAreaRealProperty: ALP_Area_ScrollLeft, ALP_Area_ScrollTop • Not in rows/columns, but in points
AL_GetSelect	AL_GetObjects: ALP_Object_Selection
AL_GetSort	AL_GetAreaTextProperty: ALP_Area_SortList AL_GetObjects: ALP_Object_SortList
AL_GetSortedCols	AL_GetAreaTextProperty: ALP_Area_SortList AL_GetObjects: ALP_Object_SortList
AL_GetSortEditorParams	Unsupported
AL_GetStyle	AL_GetColumnTextProperty: ALP_Column_FontName AL_GetColumnRealProperty: ALP_Column_Size AL_GetColumnLongProperty: ALP_Column_StyleF
AL_GetVersion	AL_GetAreaTextProperty: ALP_Area_Version
AL_GetWidths	AL_GetColumnRealProperty: ALP_Column_Width/ALP_Column_WidthUser
AL_GotoCell	AL_SetAreaLongProperty: ALP_Area_EntryGotoColumn, ALP_Area_EntryGotoRow
AL_InsArraysNam	AL_AddColumn using pointer
AL_InsertArrays	Obsolete - replace with AL_AddColumn
AL_InsertFields	Obsolete - replace with AL_AddColumn using pointer
AL_IsValidArea	AL_GetAreaLongProperty: ALP_Area_IsArea
AL_Register	AL_Register. Note that in AreaList Pro v9 this function returns 0 for OK and an integer between 1 and 8 if not OK There is a list of the error codes and their meanings here
AL_RemoveArrays	AL_RemoveColumn. Note that this will only remove one column at a time; supply -2 as the column number to remove all columns in the area
AL_RemoveFields	AL_RemoveColumn. Note that this will only remove one column at a time; supply -2 as the column number to remove all columns in the area
AL_RestoreData	AL_Load (in XML format)
AL_SaveData	AL_Save (in XML format)
AL_SetAltRowClr	AL_SetAreaLongProperty: ALP_Area_AltRowOptions, ALP_Area_AltRowColor • Using any color format, see Working with Colors
AL_SetAltRowColor	AL_SetAreaLongProperty: ALP_Area_AltRowOptions, ALP_Area_AltRowColor • Using any color format, see Working with Colors
AL_SetAreaBLOB	AL_SetAreaPtrProperty: ALP_Column_UserBLOB
AL_SetAreaName	AL_SetAreaTextProperty: ALP_Area_Name
AL_SetArrays	Obsolete
AL_SetArraysNam	AL_SetObjects: ALP_Object_Columns using array of pointers
AL_SetBackClr	AL_SetColumnLongProperty: ALP_Column_BackColor, ALP_Column_FtrBackColor • Using any color format, see Working with Colors
AL_SetBackColor	AL_SetColumnLongProperty: ALP_Column_BackColor, ALP_Column_FtrBackColor • Using any color format, see Working with Colors
AL_SetBackRGBColor	AL_SetColumnLongProperty: ALP_Column_BackColor, ALP_Column_FtrBackColor • Using any color format, see Working with Colors

Old commands	New Commands and Properties
AL_SetCalcCall	AL_SetColumnTextProperty: ALP_Column_Callback
AL_SetCallbacks	AL_SetAreaTextProperty: ALP_Area_CallbackMethEntryStart, ALP_Area_CallbackMethEntryEnd
AL_SetCellBorder	AL_SetCellLongProperty: ALP_Cell_Clear, ALP_Cell_XXXBorderOffset, ALP_Cell_XXXBorderWidth, ALP_Cell_XXXBorderColor with XXX being Top, Left, Bottom, Right • Using any color format, see Working with Colors
AL_SetCellColor	AL_SetCellLongProperty: ALP_Cell_Clear, ALP_Cell_ClearStyle, ALP_Cell_TextColor, ALP_Cell_BackColor • Using any color format, see Working with Colors
AL_SetCellEnter	AL_SetCellLongProperty: ALP_Cell_Clear, ALP_Cell_Enterable
AL_SetCellFrame	No equivalent in the new API
AL_SetCellHigh	AL_SetAreaTextProperty or AL_SetAreaPtrProperty: ALP_Area_EntryHighlight, ALP_Area_EntryHighlightS, ALP_Area_EntryHighlightE
AL_SetCellIcon	AL_SetIcon + AL_SetCellLongProperty: ALP_Cell_LeftIconID, ALP_Cell_RightIconID to set the icon ID • Using ALP_Cell_LeftIconFlags and ALP_Cell_RightIconFlags, to set a combined bitfield of horizontal position, vertical position and scaling
AL_SetCellOpts	AL_SetAreaLongProperty: ALP_Area_SelType, ALP_Area_MoveCellOptions cellMemOptimization is no longer supported
AL_SetCellRGBColor	AL_SetCellLongProperty: ALP_Cell_Clear, ALP_Cell_ClearStyle, ALP_Cell_TextColor, ALP_Cell_BackColor • Using any color format, see Working with Colors
AL_SetCellSel	AL_SetObjects: ALP_Object_Selection
AL_SetCellStyle	AL_SetCellLongProperty: ALP_Cell_Clear, ALP_Cell_ClearStyle, ALP_Cell_StyleF AL_SetCellTextProperty: ALP_Cell_FontName
AL_SetCellValue	AL_SetCellTextProperty: ALP_Cell_FormattedValue/ALP_Cell_Value AL_SetCellPtrProperty: ALP_Cell_Value
AL_SetColLock	AL_SetAreaPtrProperty: ALP_Area_ColsLocked
AL_SetColOpts	AL_SetAreaLongProperty: ALP_Area_ColumnResize, ALP_Area_ResizeDuring, ALP_Area_ColumnLock, ALP_Area_ShowWidths, ALP_Area_DragColumn, ALP_Area_DragAcceptColumn AL_SetColumnLongProperty: ALP_Column_Visible for column hiding
AL_SetCopyOpts	AL_SetAreaLongProperty: ALP_Area_CopyHiddenCols AL_SetAreaTextProperty: ALP_Area_CopyFieldSep, ALP_Area_CopyRecordSep, ALP_Area_CopyFieldWrapper Note: the v9 API uses Text, not single character
AL_SetDefaultFormat	AL_SetAreaTextProperty: ALP_Area_DefFmtInteger, ALP_Area_DefFmtLong, ALP_Area_DefFmtReal, ALP_Area_DefFmtBoolean, ALP_Area_DefFmtDate, ALP_Area_DefFmtPicture
AL_SetDefaultStyle	AL_SetColumnTextProperty: ALP_Column_FontName, ALP_Column_Size, ALP_Column_HdrFontName, ALP_Column_FtrFontName AL_SetColumnLongProperty: ALP_Column_StyleF, ALP_Column_HdrStyleF, ALP_Column_FtrStyleF AL_SetColumnRealProperty: ALP_Column_HdrSize, ALP_Column_FtrSize
AL_SetDividers	AL_SetAreaLongProperty: ALP_Area_ShowColDividers, ALP_Area_ColDivColor, ALP_Area_ShowRowDividers, ALP_Area_RowDivColor, • Using any color format, see Working with Colors
AL_SetDrgDst	AL_SetAreaTextProperty: ALP_Area_DragDstRowCodes, ALP_Area_DragDstColCodes, ALP_Area_DragDstCellCodes
AL_SetDrgOpts	AL_SetAreaLongProperty: ALP_Area_DragOptionKey, ALP_Area_DragRowMultiple, ALP_Area_DragRowOnto, ALP_Area_DragScroll
AL_SetDrgSrc	AL_SetAreaTextProperty: ALP_Area_DragSrcRowCodes, ALP_Area_DragSrcColCodes, ALP_Area_DragSrcCellCodes

Old commands	New Commands and Properties
AL_SetDropDst	AL_SetAreaTextProperty: ALP_Drop_DragDstCodes
AL_SetDropDst	AL_SetAreaTextProperty: ALP_Drop_DragDstCodes • Instead of x arguments, join them using '|' AL_SetObjects: ALP_Object_DragDstRowCodes using array
AL_SetDropOpts	AL_SetAreaLongProperty: ALP_Drop_DragAcceptLine, ALP_Drop_DragAcceptColumn
AL_SetEditedText	AL_SetAreaTextProperty: ALP_Area_EntryText/ ALP_Area_EntrySelectedText
AL_SetEditMenuCallback	AL_SetAreaTextProperty: ALP_Area_CallbackMethMenu
AL_SetEnterable	AL_SetColumnLongProperty: ALP_Column_Enterable AL_SetColumnPtrProperty or AL_SetColumnTextProperty: ALP_Column_PopupArray AL_SetColumnTextProperty: ALP_Column_PopupMenu • Menu is 4D's menu
AL_SetEntryCtls	AL_SetColumnLongProperty: ALP_Column_EntryControl
AL_SetEntryOpts	AL_SetAreaLongProperty: ALP_Area_IgnoreSoftDeselect, ALP_Area_EntryClick, ALP_Area_SelClick (defaults to 3 instead of 2 when legacy ***AL_SetEntryOpts*** is called), ALP_Area_EntryAllowReturn, ALP_Area_EntryAllowSeconds, ALP_Area_EntryMapEnter, decimalChar, useNewPopupIcon are no longer supported (decimalChar on Windows is handled automatically; new popup icon is always used)
AL_SetEventCallback	AL_SetAreaTextProperty: ALP_Area_CallbackMethOnEvent
AL_SetFields	AL_SetObjects: ALP_Object_Columns using array of pointers
AL_SetFile	AL_SetAreaLongProperty: ALP_Area_TableID
AL_SetFilter	AL_SetColumnTextProperty: ALP_Column_Filter
AL_SetFooters	AL_SetColumnTextProperty: ALP_Column_FooterText
AL_SetForeClr	AL_SetColumnLongProperty: ALP_Column_TextColor, ALP_Column_HdrTextColor, ALP_Column_FtrTextColor • Using any color format, see Working with Colors
AL_SetForeColor	AL_SetColumnLongProperty: ALP_Column_TextColor, ALP_Column_HdrTextColor, ALP_Column_FtrTextColor • Using any color format, see Working with Colors
AL_SetForeRGBColor	AL_SetColumnLongProperty: ALP_Column_TextColor, ALP_Column_HdrTextColor, ALP_Column_FtrTextColor • Using any color format, see Working with Colors
AL_SetFormat	AL_SetColumnLongProperty: ALP_Column_Format, ALP_Column_HorAlign, ALP_Column_HdrHorAlign, ALP_Column_FtrHorAlign, ALP_Column_CalcHeight
AL_SetFtrStyle	AL_SetColumnTextProperty: ALP_Column_FtrFontName AL_SetColumnLongProperty: ALP_Column_FtrStyleF AL_SetColumnRealProperty: ALP_Column_FtrSize,
AL_SetHdrStyle	AL_SetColumnTextProperty: ALP_Column_HdrFontName AL_SetColumnRealProperty: ALP_Column_HdrSize AL_SetColumnLongProperty: ALP_Column_HdrStyleF
AL_SetHeaderIcon	Unsupported
AL_SetHeaderOptions	Unsupported
AL_SetHeaders	AL_SetColumnTextProperty: ALP_Column_HeaderText
AL_SetHeight	AL_SetAreaLongProperty: ALP_Area_NumHdrLines, ALP_Area_NumRowLines, ALP_Area_NumFtrLines, instead of Pad use Indent (ALP_Area_HdrIndent, ALP_Area_RowIndent, ALP_Area_FtrIndent)

Old commands	New Commands and Properties
AL_SetInterface	AL_SetAreaLongProperty: ALP_Area_UseEllipsis, ALP_Area_IgnoreMenuMeta, ALP_Area_ClickDelay. ALP_Area_UseDateControls, ALP_Area_UseTimeControls. Appearance is always native (headers, scrollbars, highlight color, checkboxes and entry widget, except when in compatibility mode and Win7 - headers are from Vista), only headers can be drawn the legacy v8.x way using the ALP_Area_HeaderMode property; allowPartialRow is always used; useOldDatePopup is ignored (new form is used)
AL_SetLine	AL_SetAreaLongProperty: ALP_Area_SelRow
AL_SetMainCalls	AL_SetAreaTextProperty: ALP_Area_CallbackMethSelect, ALP_Area_CallbackMethDeselect
AL_SetMinRowHeight	AL_SetAreaRealProperty: ALP_Area_MinRowHeight, ALP_Area_MinHdrHeight
AL_SetMiscColor	AL_SetAreaLongProperty: ALP_Area_MiscColor1 is included for compatibilty but it is ignored because it refer to properties that, in previous versions, customised the look of the area. In Version 9 only the native look for each platform is supported, so this option is irrelevant. For the same reason, ALP_Area_MiscColor2 has been modified. In AreaList Pro v9, this color is used as the background color: before drawing anything, the whole AreaList Pro area is erased using this color ALP_Area_MiscColor3, ALP_Area_MiscColor4 • Using any color format, see Working with Colors areaAboveVertScroll - not implemented; header is drawn areaBelowVertScroll - not implemented; scrollbar is drawn; Used as background color → this color is used to erase the area areaLeftOfHorzScroll - light gray → area left from horizontal scrollbar under locked columns areaRightOfHorzScroll - light gray → rectangle right to horizontal scrollbar and below vertical scrollbar
AL_SetMiscOpts	AL_SetAreaLongProperty: ALP_Area_HideHeaders, ALP_Area_ShowFocus, ALP_Area_ShowFooters postKey and useModernLook are unsupported
AL_SetMiscRGBColor	AL_SetAreaLongProperty: ALP_Area_MiscColor1, ALP_Area_MiscColor2, ALP_Area_MiscColor3, ALP_Area_MiscColor4 • Using any color format, see Working with Colors
AL_SetPictEscape	Unsupported
AL_SetRGBDividers	AL_SetAreaLongProperty: ALP_Area_ShowColDividers, ALP_Area_ColDivColor, ALP_Area_ShowRowDividers, ALP_Area_RowDivColor, Using any color format, see Working with Colors
AL_SetRowColor	AL_SetRowLongProperty: ALP_Row_Clear, ALP_Row_ClearStyle ALP_Row_TextColor, ALP_Row_BackColor • Using any color format, see Working with Colors
AL_SetRowOpts	AL_SetAreaLongProperty: ALP_Area_SelMultiple, ALP_Area_SelNone, ALP_Area_DragLine, ALP_Area_DragAcceptLine, ALP_Area_MoveRowOptions, ALP_Area_SelNoHighlight
AL_SetRowRGBColor	AL_SetRowLongProperty: ALP_Row_Clear, ALP_Row_ClearStyle ALP_Row_TextColor ALP_Row_BackColor • Using any color format, see Working with Colors
AL_SetRowStyle	AL_SetRowLongProperty: ALP_Row_Clear, ALP_Row_ClearStyle AL_SetRowTextProperty: ALP_Row_FontName AL_SetRowLongProperty: ALP_Row_StyleF
AL_SetScroll	AL_SetAreaLongProperty: ALP_Area_Visible, ALP_Area_ShowVScroll, ALP_Area_ShowHScroll AL_SetAreaRealProperty: ALP_Area_ScrollLeft, ALP_Area_ScrollTop AL_SetRowLongProperty: ALP_Row_Reveal
AL_SetSelect	AL_SetObjects: ALP_Object_Selection
AL_SetSort	AL_SetAreaTextProperty: ALP_Area_SortList/ALP_Area_SortListNS AL_SetObjects: ALP_Object_SortList/ALP_Object_SortListNS
AL_SetSortedCols	AL_SetAreaPtrProperty: ALP_Area_SortListNS AL_SetObjects: ALP_Object_SortList/ALP_Object_SortListNS

Old commands	New Commands and Properties
AL_SetSortEditorParams	Use the ALP_Area_SortTitle property in the AreaList Pro Area theme to set the sort editor title and ALP_Area_SortPrompt to set the prompt The other options are not supported
AL_SetSortOpts	AL_SetAreaLongProperty: ALP_Area_SortDuring, ALP_Area_UserSort, ALP_Area_AllowSortEditor AL_SetAreaTextProperty: ALP_Area_SortPrompt
AL_SetSpellCheck	Not supported
AL_SetStyle	AL_SetColumnTextProperty: ALP_Column_FontName AL_SetColumnRealProperty: ALP_Column_Size AL_SetColumnLongProperty: ALP_Column_StyleF
AL_SetSubSelect	Obsolete
AL_SetWidths	AL_SetColumnRealProperty: ALP_Column_Width, ALP_Column_WidthUser
AL_SetWinLimits	Obsolete
AL_ShowSortEd	AL_SetAreaLongProperty: ALP_Area_ShowSortEditor
AL_SkipCell	AL_SetAreaLongProperty: ALP_Area_EntrySkip
AL_UpdateArrays	(1): AL_SetAreaLongProperty: ALP_Area_ClearCache with value -2 (2): AL_SetAreaLongProperty: ALP_Area_UpdateData with value 0
AL_UpdateFields	(1): AL_SetAreaLongProperty: ALP_Area_ClearCache with value -2 (2): AL_SetAreaLongProperty: ALP_Area_UpdateData with value 0

15 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 9 as transparency ratios.

See Patterns.

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 version 9 as transparency ratios.
See Patterns.

■ 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.

16

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 v3. 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 property

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 ALP 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 SRP template from ALP
```

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 SRP 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 SRP template for ALP";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)
$tableReportTemplate:=AL_GetAreaTextProperty (0;ALP_Area_SRPTableTemplate)
    //get the built-in SRP template from ALP
If ($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 SRP report and display it, save the custom template somewhere with the File menu
End if  //closing the window will prompt for save if modified
```

17 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 9 cache management

Version 8: no data caching, fields/arrays are accessed during the update event to get values to draw.

Version 9: 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 true (1).

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).

18

Appendix I Codes

AreaList Pro Error Codes

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	✔	Allow or disallow drop (during drag and drop from an external object) - 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 Mouse moved event	18	✔	Mouse moved (including over the "empty column" on the right)
AL Mouse entry unsel row event	101		Entry by mouse action in a previously unselected row
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

Constant	Value	Callback only	User action
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
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

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

Property Constant	Property Value (selector)	Property XML Name
ALP_Area_AllowSortEditor	soed	allowEditor
ALP_Area_AlpEvent	evtL	
ALP_Area_AltRowColor	altc	altRowColor
ALP_Area_AltRowOptions	alto	altRowColorParams
ALP_Area_ArrowsForHierarchy	arkh	
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_ClickedRow	evcr	
ALP_Area_ColDivColor	colc	colDividerColor
ALP_Area_ColsInGrid	coln	colsInGrid
ALP_Area_ColsLocked	coll	lockedCols
ALP_Area_ColumnLock	clck	allowColumnLock
ALP_Area_ColumnResize	cres	allowColumnResize
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_DataHeight	Adhi	
ALP_Area_DataWidth	Adwd	

Property Constant	Property Value (selector)	Property XML Name
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_DontSetCursor	DSCU	
ALP_Area_DontSortArrays	sono	dontSortArrays
ALP_Area_DoubleClick	evtD	
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	
ALP_Area_DragDstRowCodes	dddR	dstRowCode
ALP_Area_DragLine	ddln	dragLine
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_EntryAllowArrows	ecnr	navigateUsingArrows
ALP_Area_EntryAllowReturn	ecar	allowReturn
ALP_Area_EntryAllowSeconds	ects	allowSeconds
ALP_Area_EntryCell	eceC	
ALP_Area_EntryClick	eccl	entryClick
ALP_Area_EntryColumn	ecec	
ALP_Area_EntryExit	ecex	
ALP_Area_EntryFirstClickMode	ecch	firstClick

Property Constant	Property Value (selector)	Property XML Name
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_EntryModified	ecem	
ALP_Area_EntryPrevCell	ecpC	
ALP_Area_EntryPrevColumn	ecpc	
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_EventChar	evtC	
ALP_Area_Event_Filter	evtF	
ALP_Area_EventKey	evtK	
ALP_Area_EventModifiers	evtM	
ALP_Area_EventPosH	evtH	
ALP_Area_EventPosV	evtV	
ALP_Area_FillCache	data	
ALP_Area_FillNumberSign	fils	
ALP_Area_FtrIndent	frin	footerIndent
ALP_Area_FtrIndentH	frih	
ALP_Area_FtrIndentV	friv	
ALP_Area_HdrIndent	hrin	headerIndent
ALP_Area_HdrIndentH	hrih	
ALP_Area_HdrIndentV	hriv	
ALP_Area_HeaderMode	hdrm	headerMode
ALP_Area_HideHeaders	hhid	hideHeaders
ALP_Area_HierIndent	hiid	indentHierarchy
ALP_Area_IgnoreMenuMeta	emet	ignoreMenuMeta
ALP_Area_IgnoreSoftDeselect	enis	ignoreSoftDeselect
ALP_Area_IsArea	isEA	
ALP_Area_Kind	kind	
ALP_Area_LastError	(empty string)	

Property Constant	Property Value (selector)	Property XML Name
ALP_Area_LimitRows	limR	
ALP_Area_ListHeight	Alhi	
ALP_Area_ListWidth	Alwd	
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_MoveCellOptions	como	moveCellOptions
ALP_Area_MoveRowOptions	romo	moveRowOptions
ALP_Area_Name	name	name
ALP_Area_NumFtrLines	ftrl	footerLines
ALP_Area_NumHdrLines	hdrl	headerLines
ALP_Area_NumRowLines	rowl	rowLines
ALP_Area_Path	path	
ALP_Area_ReadOnly	ronl	
ALP_Area_Redraw	upds	
ALP_Area_ResizeDuring	redu	resizeDuring
ALP_Area_RollOverCell	evlC	
ALP_Area_RollOverCol	evlc	
ALP_Area_RollOverRow	evlr	
ALP_Area_RowDivColor	rowc	rowDividerColor
ALP_Area_RowHeight	rowh	
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_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 by ALP_Area_TypeAheadEffect)	
ALP_Area_SelMultiple	selm	multipleRows

Property Constant	Property Value (selector)	Property XML Name
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
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_SRPTableReport	SRPt	
ALP_Area_TableID	tbid	mainTable
ALP_Area_ToolTip	tips	
ALP_Area_TopRow	artp	
ALP_Area_TraceOnError	TRAC	
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

Property Constant	Property Value (selector)	Property XML Name
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_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_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
ALP_Cell_RightIconID	corI	rightIconID
ALP_Cell_Rotation	rotd	rotation
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

Property Constant	Property Value (selector)	Property XML Name
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_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_DisplayControl	disp	displayControl
ALP_Column_Enterable	ente	enterable
ALP_Column_EntryControl	entc	entryControl
ALP_Column_Filter	entf	filter
ALP_Column_FindCell	coce	
ALP_Column_FontName	fnam	font
ALP_Column_FooterText	ftxt	footerText
ALP_Column_Format	fmt	format
ALP_Column_FromCell	ceco	
ALP_Column_FtrBackColor	fbcl	backColor
ALP_Column_FtrBaseLineShift	fbls	baseLineShift
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_FtrVertAlign	fval	valign
ALP_Column_FtrWrap	fwrp	wrap
ALP_Column_HdrBackColor	hbcl	backColor
ALP_Column_HdrBaseLineShift	hbls	baseLineShift
ALP_Column_HdrFontName	hfnm	font

Property Constant	Property Value (selector)	Property XML Name
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_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
ALP_Column_PopupArray	entp	entryPopup
ALP_Column_PopupArrayKind	entP	kind
ALP_Column_PopupMap	entM	entryMap
ALP_Column_PopupMenu	entm	
ALP_Column_PopupName	entN	entryPopupName
ALP_Column_Reveal	reve	
ALP_Column_Rotation	rotd	rotation
ALP_Column_ScrollTo	scto	
ALP_Column_Size	size	size
ALP_Column_SortDirection	sord	sort
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_TextColor	tclr	textColor
ALP_Column_Type	type	dataType
ALP_Column_Uppercase	entu	uppercase
ALP_Column_UserText	Utxt	userText
ALP_Column_VertAlign	valg	valign

Property Constant	Property Value (selector)	Property XML Name
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	
ALP_Drop_Kind	kind	
ALP_Drop_Name	name	name
ALP_Drop_XML	xml	
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_Fields	Xsrc	
ALP_Object_FooterText	ftxt	
ALP_Object_FooterTextNH	fTXT	
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_Clear	clr	
ALP_Row_ClearStyle	sclr	
ALP_Row_Collapse	hicr	

Property Constant	Property Value (selector)	Property XML Name
ALP_Row_CollapseAll	hica	
ALP_Row_Expand	hier	
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_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_VertAlign	valg	valign
ALP_Row_Visible	visi	
ALP_Row_Wrap	wrap	wrap
ALP_Row_XML	xml	
ALP_Row_XML	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

19

Appendix II Troubleshooting and FAQs

Here are some usual questions received by our technical support. You can find more on the AreaList Pro forum.

AL_Register returns zero

In AreaList Pro previous versions ***AL_Register*** used to return 1 when registered. Now it returns 0.

With AreaList Pro version 9, zero means no error for all commands. Now the plugin is correctly activated if the result code is zero. See AL_Register.

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.

Undefined parameters

A compiler runtime error occurs with a message that the parameters are undefined.

You must define the parameters in the entry started and entry finished callbacks as long integers.

See the Callbacks topic.

Empty titles in 4D v11

Button’s titles are empty (or boolean labels, etc.) with 4D v11.

The resources have been transferred to XLIFF files where you can find various default values. For example, «AreaList™ Pro Format Defaults» is a group in ‘ALP.xlf’ file.

All text strings (except Advanced Properties) are also set from XLIFF files.

4D v11 does not support XLIFF files in plugins: copy ALP.xlf into your database’s Resources.

Scrolling

“Ghost” scrollbars

AreaList Pro’s scroll bars show up on other pages.

If an AreaList Pro area is displayed on a form in a window, and another form is going to be displayed in the window with **DIALOG**, **ADD RECORD** or **MODIFY RECORD** commands, or the user switches to a different page in the same form, you must inform the AreaList Pro object that another form will be displayed.

To do this, use the ALP_Area_Visible property of ***AL_SetAreaLongProperty*** to:

1. deactivate the AreaList Pro area
2. hide the scroll bars

Note that you will need to call ALP_Area_Visible again to reactivate the area.

For example:

```
AL_SetAreaLongProperty(area; ALP_Area_Visible;0)  //hide the area
AL_SetAreaLongProperty(area; ALP_Area_Visible;1)  //show the area
```

This setting may also be useful on Windows, in case you open a dialog in the same window as an AreaList Pro area, and the cursor and typed characters do not appear when they are located over (or close to) the underlying AreaList Pro area coordinates (in the calling layout).

“Reveal” properties

Is the ability to "reveal" a row/column/cell implemented? Those selectors (i.e. ALP_Row_Reveal, etc.) have a type of "n/a" and when I try to use them I get an error = 5. Is there another way to force the area to scroll so that a particular row/column/cell is visible?

Yes, it is implemented. «Reveal» properties do not expect any value (they are not real properties), so you can use any kind of setter with any value (there is no getter implemented - the result is 5 aka ALP_Err_InvalidRequest).

For example, the following line will reveal (scroll to) the row number $row:

```
AL_SetRowLongProperty (area;$row;ALP_Row_Reveal;0)
```

Another way is to set the scrollbars directly - see the ALP_Area_ScrollLeft and ALP_Area_ScrollTop properties (using points, not column / row numbers).

Reveal properties can be called in On Load phase only for areas with fixed row height areas.

Calculating the scrollbar and area width

How do I determine the width of the vertical scrollbar? I need to calculate the entire width of the AreaList Pro area so I can resize the window after changing the columns programmatically.

There is no accessor for scrollbar width (system default is used), but you can use these (read-only) properties: ALP_Area_ListWidth, ALP_Area_ListHeight, ALP_Area_DataWidth, ALP_Area_DataHeight

«List» is the Data area of the AreaList Pro area - without frame, scrollbars, headers, footers - just the data part (on screen).

«Data» is the actual data size - from this the scrollbars are computed

For example, lets have a small AreaList Pro area on a form, ListWidth = 200 (AreaList Pro object size would be around 220 points)

DataWidth = 250 (it is irrelevant how many columns, the grid width is measured)

➞ the horizontal scrollbar will be active (data is wider than screen view)

DataWidth = 150

➞ the horizontal scrollbar will be inactive

So you have to get ALP_Area_ListWidth equal to ALP_Area_DataWidth by resizing the area/window.

Horizontal scrollbar modes

In the ALP_Area_ShowHScroll property, how is «0 = Automatic, Not Shown» different than «Automatic, Shown»? And what is the default value?

The horizontal scrollbar property has two modes: automatic hiding (dependent on data with and area width) and manual.

Both modes can have the scrollbar visible (shown) or hidden (not shown).

- 0 - automatic, hidden: the scrollbar is hidden by AreaList Pro, because the area is wide enough to show all columns
- 1 - automatic, shown: the scrollbar is shown by AreaList Pro, because the area is not wide enough to show all columns
- 2 - manual, always hidden: the developer chose to always hide the scrollbar
- 3 - manual, always shown: the developer chose to always show the scrollbar

Default value is «auto-hide» = «hide when not needed, show otherwise». It is actually initialized to 1, which means it will be 0 or 1 depending on data and area width.

To always show it, set it to 3. To always hide it, set it to 2.

Scrolling to the top

The following command is generating an error: ***AL_SetRowLongProperty*** ($alp_area;1;ALP_Row_Reveal ;0)
The vertical scroll value is set to one (1) but the AreaList Pro object does not have any lines in it.

Generally, when you address an object which does not exist, an error is returned for the pointer variant of Get/Set and **TRACE** is executed for other specialized variants (not returning any error).

When you want to reveal row X and that row does not exist, it results in an error.

Try to use row number 0 (header, see Row Numbering) to scroll to the top.

Also, setting the vertical scroll position to zero will work as well:

AL_SetAreaLongProperty ($area; ALP_Area_ScrollTop; 0)

Note that ALP_Area_ScrollTop and ALP_Area_ScrollLeft use points, not row (and column) numbers.

Fixed row height and scroll position

Why should the scroll position change when you scroll right to the bottom and don't adjust any rows or change the form height? I'd have thought under this circumstance it should always be the same value.

Imagine a set of 10,000 records.

The initial row height is computed from the column fonts, padding and ALP_Area_MinRowHeight.

Let's have it at 16 points.

Multiplied by number of rows it gives us the initial data height in points (with small simplification, it is the maximum value for the vertical scrollbar).

AreaList Pro fetches e.g. 20 rows to be displayed, the rows are measured when displayed.

First row height is 32 (two lines): the array holding the row height is adjusted, the data height and the scrollbar are adjusted accordingly.

Row #200 height is 64: the height (and the scrollbar) will be adjusted when the row is made visible (this is the time when it is measured).

Otherwise AreaList Pro would have to fully measure all rows to get correct full data height.

And this has to be done on every column width change.

Using variable row height slows down processing - don't use it if you don't need it.

Row/cell formatting (setting font/size on row/cell level) can also have an impact even though the height is fixed: the row height computed from the fonts used by columns (default on MacOS is Lucida Grande 13) differs from the row height computed while displaying the rows (from the fonts used by row/cell).

To avoid this effect, instead of setting a font/size to all rows/cells, apply the style to the columns: row height will be correct and it will be faster.

Scrolling to a row

I am having trouble scrolling a list to the selected row. I find it odd that I can select a row by row number but have to scroll by points (ALP_Area_ScrollTop/ALP_Area_ScrollLeft use points, not row/column numbers).

Use this to make $row the topmost row in the view (if possible):

AL_SetRowLongProperty ($area;$row;ALP_Row_ScrollTo)

Alternatively you can use the following code to make the row visible (no scrolling if already visible):

AL_SetRowLongProperty ($area;$row;ALP_Row_Reveal)

How do I determine the number of pixels to scroll to bring the selected row to the top of the list window?

If you use fixed row height, then it can be computed (ALP_Area_RowHeight), but if you use variable row height, you would have to get the individual row heights (ALP_Row_Height) to compute it.

Grids

Defining a grid

I am calling ***AL_GetObjects*** ($area;ALP_Object_Grid;$columns)
But when the call runs, the $columns array is empty, and $err = 0.

This is because the grid is not yet defined.

When no grid is explicitly defined, AreaList Pro will create it (from all visible columns) on demand i.e. at first update event.

You can create it simply by filling $columns with numbers from 1 to number_of_columns_to_be_shown.

- In compatibility mode (ALP_Area_Compatibility), number_of_columns_to_be_shown is determined by the number of columns (ALP_Area_Columns) and the number of columns to hide (ALP_Area_CompHideCols).
- Otherwise it is determined by visible columns (i.e. having ALP_Column_Visible = 1).

In compatibility mode, visibility of all columns is updated automatically during grid creation.

Number of columns/rows in grid

I tried ALP_Area_ColsInGrid, but that returned -1 for every area.

ALP_Area_ColsInGrid defaults to -1 which means «all columns». This property returns -1 for every area regardless of the compatibility mode, as long as it is not changed explicitly.

When I set ALP_Area_RowsInGrid AreaList Pro threw an error when I passed -1.

Default value for ALP_Area_ColsInGrid is -1 (meaning unlimited = all columns).

Default value for ALP_Area_RowsInGrid is 1 (meaning all columns in one row).

You can't have -1 rows displayed.

You can safely set it to 1 (single row).

Setting it to zero means "use as many rows as needed to show all visible columns in ALP_Area_ColsInGrid columns".

But when ALP_Area_ColsInGrid is -1 at the same time, it effectively means: "use one row for all columns".

The order of precedence is ALP_Area_RowsInGrid (if the value > 1), ALP_Area_ColsInGrid (if the value > 1), otherwise all columns in one row.

Lost grid

I'm having a problem when I go to restore the columns back to the default arrangement. Somehow the grid order is being lost.

Yes, the grid is lost (cleared) when:

- a column is added
- a column is removed
- a column's visibility is modified
- ALP_Area_RowsInGrid is set (does not have to be modified)
- ALP_Area_ColsInGrid is set (does not have to be modified)
- ***AL_SetColOpts*** is called with a different fifth argument (columns to hide)

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

Or just use ALP_Object_Grid at the end of the area initialization.

Note that ALP_Column_Visibility and ALP_Object_Grid are interdependent.

If you add a column, it has to be added to the grid. This is the reason 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 modified according to the new grid: only the columns in the grid will be set to visible, all others will be set to invisible.

Grid formatting

When I display an area as a grid and then switch to a template that does not contain a grid, the grid formatting remains.
I've called ***AL_RemoveColumn*** ($eList;-2).

Removing all columns destroys the grid, but does not reset either ALP_Area_RowsInGrid or ALP_Area_ColsInGrid.

Just use ***AL_SetAreaLongProperty*** ($eList;ALP_Area_ColsInGrid;-1) before adding new columns (assuming you have not changed ALP_Area_RowsInGrid).

Regarding "template" loading, using ***AL_Load*** should reset all area properties to defaults before loading that template.

Callbacks

Popup menus

The ALP_Area_CallbackMethPopup property is used when a popup is clicked but no popup array/menu is defined.
How can you click a popup if no popup array/menu is defined?

If a column is enterable by popup and AreaList Pro detects a click on the popup icon:

1. AreaList Pro first checks if it has any PopupMenu or PopupArray/PopupMap defined. If yes, AreaList Pro displays the specified menu in case of PopupMenu or builds and displays the menu in case of PopupArray/PopupMap.
2. If not, then AreaList Pro checks if a ALP_Area_CallbackMethPopup callback method is defined, and if yes, it calls the method defined in this property. This 4D method has the following parameters:

   PopupCallback (area:L; row:L; column:L; dataType:L) -> bool:Handled

   It is up to the developer to write such a 4D method to handle the click - you may, for example, display your own Data Picker dialog. Or build a 4D menu and display it with a Hierarchical Popup menu. Or do nothing and return false. It is also up to the developer to read the current cell value and set a new value as needed.
3. If the ALP_Area_CallbackMethPopup method returns false or is not defined, AreaList Pro displays its own popup: for columns with date/time values it displays predefined DataPicker or TimePicker dialogs. For other column types AreaList Pro displays a menu with one dimmed value "no items".

   See the example in the Callbacks section. See also Entering data in AreaList Pro with DisplayList.

Edit menu callback

How do we setup a callback to trap the Edit menu?

```
AL_SetAreaTextProperty ($area; ALP_Area_CallbackMethMenu; «MyEditMenuFunction»)
```

Once in the callback method, how do we trap for exactly the Copy event (when rows are selected and nothing else (letting the system handle the rest)?

```
C_LONGINT ($0;$1;$2)  //result, area, event mask
If (($2 = AL Edit Menu Copy Mask) & (AL_GetAreaLongProperty ($1; ALP_Area_SelType) = 0))
   //this is a copy operation on selection of row(s)
   //handle the copy
   $0:=AL Edit Menu Handled Mask
End if
```

Event Callback Logic

When we moved to AreaList Pro version 8, it was strongly suggested that we change our programming logic to take advantage of the new callback system that was introduced. Is this type of centralized management of all callback events no longer recommend, and all this code should be put back into the individual AreaList Pro objects?

You can use what you want - both are supported.

However, the two last arguments have been dropped and are no longer supported.

With version 8.x, the ***AL_SetEventCallback*** command passed 8 parameters to the specified method:

1. AreaList Pro Area

2. Event Code

3. Current event from 4D (mouse down = 1, key down = 3, mouse wheel = 39, cursor/mouse moved = 18)

4. Last Clicked Column

5. Last Clicked Row

6. Modifiers

7. Tip String

8. AreaList Pro Area Name

The event callback is now called as (area; alpEvt; 4Devent; column; row; modifiers)

This is identical to AreaList Pro 8.5, just the 2 last arguments are not passed.

To set a tooltip, instead of setting $7 to the tooltip text, call:

AL_SetAreaTextProperty ($1; ALP_Area_ToolTip; "tool-tip text to show")

To get the area name, instead of using $8, call:

$areaName:=***AL_GetAreaTextProperty*** ($1; ALP_Area_Name)

Note that with event 18 (AL Mouse moved event), $4 is the column under the pointer and $5 is the row under the pointer.

Also, note that this AL Mouse moved event will not be reported if ALP_Area_Event_Filter is set to 1.

Properties

Properties setters types

I'm confused about long vs real vs text property.

If you want to get/set a property, use the long, real, text or pointer variant depending on the kind of the value you are accessing.

- any boolean property can be coerced to long, real and text
- any integer property can be coerced to real and text
- any real property can be coerced to text (note: decimal dot, not decimal comma)

For example, let's access a boolean property, like ALP_Area_Visible:

```
C_LONGINT($err)
C_BOOLEAN($bool)
C_LONGINT($long)
C_REAL($real)
C_TEXT($text)
$err:=AL_GetAreaPtrProperty ($area;ALP_Area_Visible;->$bool)
$long:=AL_GetAreaLongProperty ($area;ALP_Area_Visible)
$real:=AL_GetAreaRealProperty ($area;ALP_Area_Visible)
$text:=AL_GetAreaTextProperty ($area;ALP_Area_Visible)
```

All values are equal: **Num** ($bool) = **Num** ($text) = $long = $real

Same with setters:

```
$bool:=True
$err:=AL_SetAreaPtrProperty ($area;ALP_Area_Visible;->$bool)
AL_SetAreaLongProperty ($area;ALP_Area_Visible;Num($bool))  // 1
AL_SetAreaRealProperty ($area;ALP_Area_Visible;Num($bool))  // 1
AL_SetAreaTextProperty ($area;ALP_Area_Visible;String(Num($bool)))  // «1»
```

But for example if you want to set the header text, you will not succeed with long or real variants: you either have to use the text variant with a text value or the pointer variant with a pointer to a text variable/field.

Using the Redraw property

When should I use ALP_Area_Redraw ?

The area is not always redrawn immediately, but after it gets an update event from 4D.

In some situations you may need to have it redrawn immediately:

```
AL_SetAreaLongProperty ($area;ALP_Area_Redraw)
```

For example, when the user control-clicked on an unselected row, this row is made selected (selection changed), but the area is not yet redrawn, the callback method is called and / or the object method is executed.

Use the ALP_Area_Redraw property at this point to redraw the AreaList Pro area before you show a contextual popup (e.g. dynamic pop up menu) on the newly selected row.

Formatting

Font issues

When I try to set a column font to be Geneva 9 point italicized, the font appears non-italicized. Are there some fonts that do not have the capability of being italicized?

AreaList Pro v9 uses CoreText on Mac and GDI+ on Windows to draw text.

Not all fonts contain italic (or bold) variations and those technologies do not synthesize italic (as QuickDraw did).

For example Geneva on most Macs has only the Regular typeface, Arial has Regular, Bold, Italic and Bold Italic typefaces.

Setting the format for a column

What is the third parameter of the old ***AL_SetFormat*** command (format) mapped to in AreaList Pro version 9?
How do we set the physical format for a column, such as "###,##0.00"?

The format is a column property: ALP_Column_Format.Try this:

AL_SetColumnTextProperty ($area; $column; ALP_Column_Format; "###,##0.00")

See Formatting.

Displaying checkmarks

I was using **Char**(18) in the text with AreaList Pro v8.5 to display a checkmark, this no longer works.

AreaList Pro version 9 uses Unicode and the checkmark at position ASCII 18 is only present in some fonts (on Mac).

Try to use Unicode characters instead:

Square Root	0x221A √
Check Mark	0x2713 ✓
Heavy Check Mark	0x2714 ✔

Headers

Header Foreground Color on Windows 7

Does Header Foreground Color work on Windows 7? Do you have a sample?

This should work (regardless of ALP_Area_HeaderMode):

AL_SetColumnTextProperty ($area;$column;ALP_Column_HdrTextColor;"yellow")

AL_SetColumnLongProperty ($area;$column;ALP_Column_HdrTextColor;0xFFDD00DD)

Header size and sort indicator

Is there a way to reduce the size of the AreaList Pro Header on Windows 7?

In AreaList Pro version 8 the sort indicator was right from the header text, since AreaList Pro version 9 the sort indicator is above the column header text. But this wastes some space, especially on forms including many AreaList Pro areas.

- The answer is **no** when using native headers (ALP_Area_HeaderMode = 0) and Aero interface is active (Windows theme). This is simply the way it is used on Windows 7
- The answer is **yes** if the Classic Windows theme is used (XP pictures are used, sorting triangle is at the right side)
- The answer is **yes** when using ALP_Area_HeaderMode = 1 or 2 - then the sorted column is underlined, the order is not directly visible, the background color can be set (ALP_Column_HdrBackColor).

Rows

Dynamic row height

I can’t seem to get the dynamic row height feature to work.

Set ALP_Area_NumRowLines to 0:

```
AL_SetAreaLongProperty (area;ALP_Area_NumRowLines;0)
```

When ALP_Area_NumRowLines is set to 0, variable row height is determined by columns having the ALP_Column_CalcHeight property set to true (1). Use ALP_Column_CalcHeight for the desired column(s) where you want the row heights to be calculated depending on the cell contents:

```
AL_SetColumnLongProperty (area;column;ALP_Column_CalcHeight;1)
```

Row height and header/footer height

How do ALP_Area_RowHeightFixed and ALP_Area_NumRowLines interact?

ALP_Area_RowHeightFixed is deprecated and superseded by ALP_Area_NumRowLines, ALP_Area_NumHdrLines and ALP_Area_NumFtrLines .The proper way is to set the number of rows to zero (setting ALP_Area_NumRowLines to zero).

For example, row heights are fixed when ALP_Area_NumRowLines is non-zero, otherwise the row heigh is determined by columns having ALP_Column_CalcHeight set to 1 as specified above.

You can set different values for rows, headers and footers, e.g. single line + fixed height for data rows but variable multi-line header (when you change the column width, the header height can change).

ALP_Area_RowHeightFixed is now emulated as follows:

Getter

```
(ALP_Area_NumRowLines#0)
```

Setter

```
If (ALP_Area_RowHeightFixed=1)
    If (ALP_Area_NumRowLines=0)
        ALP_Area_NumRowLines:=1
    End if
    If (ALP_Area_NumHdrLines=0)
        ALP_Area_NumHdrLines:=1
    End if
    If (ALP_Area_NumFtrLines=0)
        ALP_Area_NumFtrLines:=1
    End if
Else
    ALP_Area_NumRowLines:=0
    ALP_Area_NumHdrLines:=0
    ALP_Area_NumFtrLines:=0
End if
```

Columns

Double-clicking an enterable column

I have specified that a column should be enterable, but nothing happens when I double-click it.

You need to set the entry mode - for example:

AL_SetAreaLongProperty (area;ALP_Area_EntryClick;2)

Custom column property

It would be helpful to set my own text ID attribute to each column, so that it can be retrieved later.

Each column has a property for free use by the developer: ALP_Column_UserText:

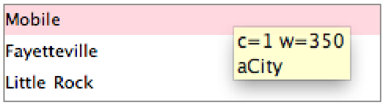
$MyID:="My own custom text id for my column"
AL_SetColumnTextProperty ($area;$columnID;ALP_Column_UserText;$MyID)
AL_GetColumnTextProperty ($area;$columnID;ALP_Column_UserText)

Column width tooltips

An information tooltip is displayed when the modifier keys are pressed even though the ALP_Area_ShowWidths properties is set to zero.

This is 4D's tooltip, not AreaList Pro's.

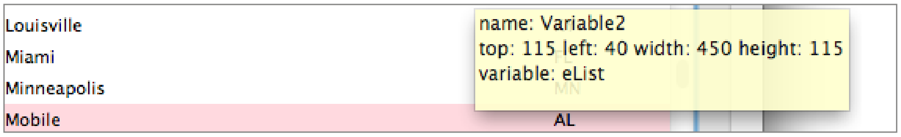
AL_SetAreaLongProperty (eList;ALP_Area_ShowWidths;1) // AreaList Pro information



Mobile
Fayetteville
Little Rock

c=1 w=350
aCity

AL_SetAreaLongProperty (eList;ALP_Area_ShowWidths;0) // 4D information



Louisville
Miami
Minneapolis
Mobile AL

name: Variable2
top: 115 left: 40 width: 450 height: 115
variable: eList

Calculated columns

I am trying to use ***AL_AddCalculatedColumn*** in an AreaList Pro area displaying 4D arrays, but I am getting a ALP_Err_InvalidPointerType error.

AL_AddCalculatedColumn can be only used to set Calculated Columns with AreaList Pro configured to display fields, not arrays.

If you want to use calculated columns with arrays, you need to set the ALP_Column_Calculated property to true (1) and specify the callback method name in the ALP_Column_Callback property.

Specific column color with alternate row coloring

I tested an area with alternate row coloring, I wanted a column with a different color, but there is no 'blending' between the column custom color and the alternate row color. Only the rows with the ‘standard’ (i.e. white) color get the custom column color, while the alternate colored rows remain untouched. Here is the code used for testing:

```
AL_SetColumnTextProperty ($area;3;ALP_Column_BackColor;»#C2C2C2»)  //color for column 3
AL_SetAreaTextProperty ($area;ALP_Area_AltRowColor;»#F2F2F2»)  //color for altrow
AL_SetAreaLongProperty ($area;ALP_Area_AltRowOptions;15)  //bit 3: mix colors
```

When you paint 100% white on a black surface, you will get white.

In other words, the alternate row color needs some transparency (non-100% opacity):

```
AL_SetAreaTextProperty ($area;ALP_Area_AltRowColor;»#80E2E2E2»)  //50% opacity
```

Setting a multi-styled column

We have a text field that has styled text. I cannot get AreaList Pro to display it without the style characters.
I have added ***AL_SetColumnLongProperty*** (eaRecList;0;ALP_Column_Attributed;1) and it still displayed the style characters.

Use the real column number, not zero; e.g.

```
AL_SetColumnLongProperty (eaRecList;4;ALP_Column_Attributed;1)
```

Column # zero is used as default for newly created columns (in any area).

Column # -2 is used for all columns in the specified area.

We don’t recommend using ***AL_SetColumnLongProperty*** (eaRecList;0;ALP_Column_Attributed;1) because all columns in all areas (created after this call) will be multi-styled, causing unnecessary processing.

“Undefined Value” = Different array sizes

What it this ###Undefined Value### that is displayed at the bottom of some columns?

Make sure all arrays are evenly sized! This means that some arrays are larger than others. In this case, previous AreaList Pro versions used to show the minimum row count (present in all arrays).

AreaList Pro version 9 uses the maximum number (largest array size, including invisible columns).

Smaller arrays will display “###Undefined Value###” for any missing items (array elements).

Events

Selecting rows during the On load event

When I try to select all the rows of an area during the On Load event of an form, nothing happens.

I solved it with a timer but this is not OK

This may happen if the selection is empty during On Load and is modified afterwards. Setting ALP_Object_Selection won't operate because AreaList Pro has no rows.

When a Form is displayed:

- AreaList Pro is called to initialize
- If there are valid Advanced Properties, AreaList Pro is initialized from them (columns are added, etc.)
- The number of rows is detected from the current selection (this is always done when adding the first column, even in arrays mode and not only when Advanced Properties are used)
- Form and Object methods are called with the On Load event

At this point you can set the initial selection of rows. In case you modify the 4D record selection, you must inform AreaList Pro with ALP_Area_CheckData (or ALP_Area_UpdateData if the selection was previously not empty and has changed).

The order is important. If you are modifying the 4D selection (especially from an empty one), you need to inform AreaList Pro and only then can you set the row selection, not before.

If you don't inform AreaList Pro of the change, it will detect it automatically on the first update event and show the records, but you can't modify the selected rows until AreaList Pro knows that there are any rows at all. This is the reason why On Timer is working for you.

If the selection is not empty before initializing the area, it will simply work.

Responding to user events

AreaList Pro does not respond appropriately to user events such as single or double clicks.

See the Events topic in the Programming the AreaList Pro Interface chapter, and refer to the list of event codes.

I'm not getting any Event #2 from ALP_Area_Event when I double click.

This property returns the raw event, not the AreaList Pro event. For mouse down it is 1, key down is 3, mouse moved is 18…

To get what you expect ask for the AreaList Pro event, not the general event:

```
$event:=AL_GetAreaLongProperty (eArea;ALP_Area_AlpEvent)
```

Ctrl-Click

Is it normal that on Windows a ctrl-click on a row reports 1 instead of 5, which is reported on Windows with right click and on Mac with ctrl-click?.

Ctrl on Windows is equivalent to Cmd on Mac. Cmd-click on Mac reports 1.

Ctrl-click on Mac is right-click on Windows (old Mac mouses had only one button).

Hierarchy

Fields

How do you specify the hierarchy level an expansion when the AreaList Pro area is set up using fields?

Identically to arrays mode:

- add columns as usual
- use ALP_Object_Hierarchy to set the hierarchy

Note: there could be a problem with sorting… (arrays with defined hierarchy can be "sorted" (the arrays are not sorted in hierarchy mode; internal sort index is used); fields are always sorted by 4D).

Hierarchy arrays

If I use arrays as it says, will they automatically stay in sync with the field columns?

The arrays are used to create a hierarchy (hierarchy of objects, no flat arrays), but they are not used afterwards - they are not in sync even in arrays mode. When you change the selection (or arrays), you have to re-create the hierarchy.

You have to maintain the hierarchy information:

- get the current hierarchy (including the expansion)
- insert the elements into the array AND the hierarchy arrays
- set the new hierarchy
- notify AreaList Pro of updating the arrays

```
ARRAY INTEGER($aLevel;0)
ARRAY BOOLEAN($aExpanded;0)  //ARRAY INTEGER or ARRAY LONGINT can be used, too
$err:=AL_GetObjects2($area;ALP_Object_Hierarchy;$aLevel;$aExpanded)
//insert the row of interest into the data arrays
//insert corresponding row into the $aLevel and $aExpanded
$err:=AL_SetObjects2($area;ALP_Object_Hierarchy;$aLevel;$aExpanded)
AL_SetAreaLongProperty($area;ALP_Area_UpdateData;0)
```

So using arrays instead of fields is probably a better choice.

Note: arrays used to define the hierarchy are not considered as columns (arrays), they are used only once to create the hierarchy.

Compatibility mode

My areas are mostly in compatibility mode, so I add the column property for the columns I want to wrap at the end of my configuration, as well as set the area to be variable row height and to use the same columns that I'm word wrapping as calculated height columns. Yet no wrapping takes place.

Don't use compatibility mode if you want to use some of the new features.

In compatibility mode, e.g. the wrap mode only depends on the number of lines to be shown: ALP_Area_NumHdrLines, ALP_Area_NumRowLines and ALP_Area_NumFtrLines, respectivelly. If the number of lines is 1, wrap mode is set to 0. Any wrap mode set explicitly (ALP_XXX_Wrap) is ignored (any means at any level: column, row or cell).

When ALP_Area_Compatibility is set to 1, some behaviors are different:

- ALProEvt variable is created & updated
- the visibility of columns (ALP_Column_Visible) is modified according to the number of hidden columns (ALP_Area_CompHideCols)
- the area is made visible on update event (ALP_Area_Visible)
- the wrap mode is set depending on the row lines number
- the area is draggable and droppable even if not set as draggable or droppable in form properties
- when a row or column is drag & dropped in same area, it is moved on drag, not on drop
- the headers on Windows are drawn using pictures (eliminates native "white" Windows 7 headers)
- the horizontal scrolling is set to columns (ALP_Area_ScrollColumns = 1)
- auto-selection of unselected row on click into a popup icon is disabled (ALP_Area_SelNoAutoSelect = 1)
- if only one column is to be displayed, it will have the width of the area
- columns are physically reordered on Drag

You can try to simply turn the compatibility mode off (set ALP_Area_Compatibility to zero), but it depends on your code if it will work as you expect…

- no ALProEvt - use ***AL_GetAreaLongProperty*** to get ALP_Area_AlpEvent
- any columns to be hidden have to be maintained
- the area has to be made visible if it was hidden explicitly
- the wrap mode is set depending on the ALP_XXX_Wrap properties (you still need to set the row lines number)
- for drag & drop, the area must have the Draggable and/or Droppable properties enabled in form
- during On Drag event (column = -7, row = -5), the column/row was not yet moved -> use On Drop event (9, 8)
- the columns are not reordered - the order is defined by the grid setup

The rest should work (other properties are not reset to default).

Advanced Properties

I just can’t get the "Entry allowed via" popup to stick to anything other than "Keyboard Only" in the Advanced Properties. I set it, and click the OK button, and when I open the Advanced Properties again it’s set back to "Keyboard Only".

This is because you are changing it for column «Default».

“Default” settings are used for newly added columns in Advanced Properties only.

Enterability is a column property: click on the column that you want to set, then use the “Entry allowed via” popup…

Detecting a modified value in popup entry

AL_GetCellMod is always returning 1 (Modified) when you select a drop-down menu, whether or not you changed the value.

AL_GetCellMod should always return 1 (true) for popup entry.

On the other hand, ALP_Area_EntryModified will return 1 only if the entry is modified.

AL_GetCellMod will return 1 when the entry is modified or if popup entry is used.

This behavior is compatible with AreaList Pro 8.x.

If you want to find out if the entry was actually modified, use:

$modified:=***AL_GetAreaLongProperty***($area;ALP_Area_EntryModified)

No fields from local table in field mode

When I¹m only displaying fields from related tables, but no fields from the local table, the AreaList Pro area only shows a single row: seemingly one record from a related table, rather than all the rows of my local table.

AreaList Pro uses **SELECTION RANGE TO ARRAY** to get the values

- only many to one automatic relations are supported
- if there are no columns in the AreaList Pro area and you add a field, that field's table is the master table

Set the master table before adding the first column:

AL_SetAreaLongProperty ($area;ALP_Area_TableID;$tableID)

And yes, there is a known issue: when AreaList Pro builds the **SELECTION RANGE TO ARRAY** call, the value set in ALP_Area_TableID is not honored when there is no field from the master table (it has to be the first field parameter to **SELECTION RANGE TO ARRAY** to designate the master table).

Add at least one master table's column, make it invisible:

AL_SetColumnLongProperty ($area;$column;ALP_Column_Visible;0)

Note: this trick is no longer needed in v9.9.2 and above. 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.

20 Index

21 Copyrights and Trademarks

All trade names referenced in this document are the trademark or registered trademark of their respective holders.

AreaList Pro is copyright Plugin Masters SAS and exclusively published worldwide by e-Node.

4th Dimension, 4D and 4D Server are trademarks of 4D SAS.

Windows, Excel and Vista are trademarks of Microsoft Corporation.

Macintosh, MacOS and MacOS X are trademarks of Apple, Inc.