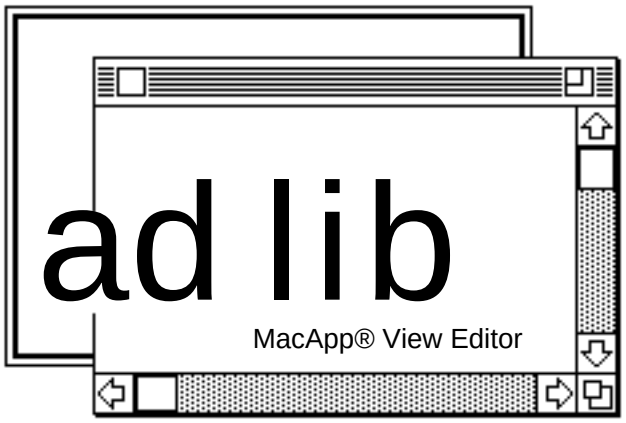


user's guide

Version 2.0

Contents

Overview

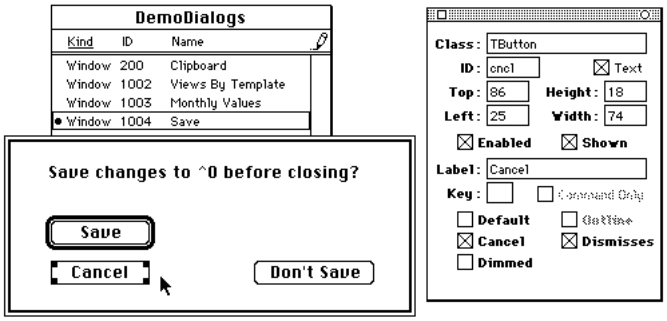
This overview provides information you may find useful as you begin to use Ad Lib. This is the section to read if the user's guide is usually the last thing you look at when using a new program. It explains the features unique to Ad Lib. If you are new to MacApp® you will probably want to use this guide and the Ad Lib software in concert with Apple's MacApp documentation to learn more about MacApp views and their uses. This document assumes that you already understand the basics of views in MacApp.

The overview provides information in the following areas:

- An introduction to view editing with Ad Lib
- Things you should be aware of as you use Ad Lib
- Ad Lib's unique features

You may also want to look over the *Tips and Helpful Hints* section at the end of this guide. At the same time check out the Frequently *Asked Questions* section. If you intend to use Ad Lib with files created by Apple's ViewEdit, you should also read the *Ad Lib and ViewEdit* section.

Windows

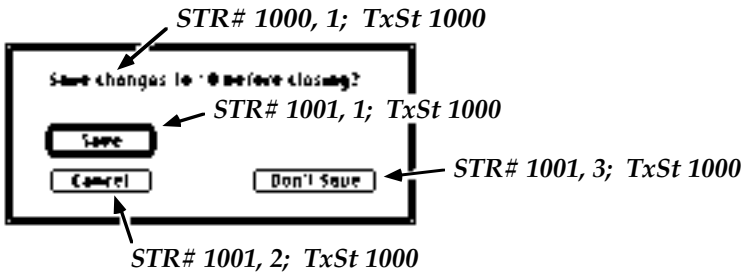


One of the first things you will notice as you start to use Ad Lib is that you are working directly in the window you are creating. In other MacApp view editors, you never see the actual window you are creating. Instead, you edit views within a window "place holder," and attach the window's attributes separately. Ad Lib, in contrast, gives you a choice. With the *Use Actual Windows* option on, Ad Lib doesn't make any distinction between windows and other kinds of views. With the option off, a scrollable window is used in the place of the window you have specified.

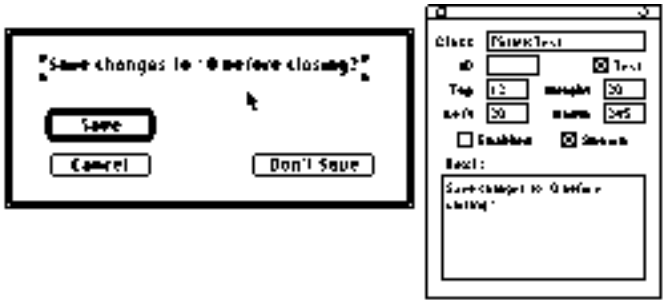
When you add a new view to your document, you specify whether the view is a window or a non-window “view.” Non-window views are used within some other view in your application. After you specify the window type, you edit the window exactly the same way as you edit other views.

Text Resource Management

One of the most convenient features of Ad Lib is its text resource management.



MacApp 3 resource formats separate text strings and style information from the view data. Each type of information has its own resource type. This allows several views to share the same resources, and potentially reduces the size of the resource files. Unfortunately, to take full advantage of this feature, you must keep track of the resources you are using and share an existing resource at the appropriate times. This structure can make view creation cumbersome, since you must create as many as three separate resources for each view. Fortunately, Ad Lib takes care of all this for you.



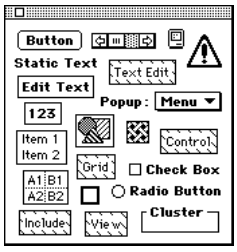
As you create a new view you simply enter the text and style information directly. When you save your file, Ad Lib looks for identical resources and shares one copy among the views that refer to it. To facilitate this, Ad Lib maintains *resource reference count tables*. These reference counts record the number of times each resource is used. They allow Ad Lib to know when you have deleted the last occurrence of a particular resource so it is deleted from your file. These reference counts are stored in your file as `'ALIB'` resources.

Ad Lib also saves a time-stamp with your file. If you change the file with another program, Ad Lib recognizes this and gives you the opportunity to rebuild the reference tables when you open the file. If you are certain that the text resource usage has not changed, you can skip this step by selecting the *Ignore* button in the dialog—however, be very careful about this.

▲ If you have added or removed views, or changed the text attributes of a view outside Ad Lib, the reference tables will be incorrect and Ad Lib may do the wrong thing with your resources if the tables are not rebuilt. To force the reference tables to be rebuilt, remove the 'ALIB' resource ID zero from your file with a resource editor.

Palettes

Ad Lib employs several "floating palette" windows to provide editing functions.



The *Object* palette is used to add new views to a window. Simply drag objects from the palette to your editing window. New objects are always created in a convenient standard size.

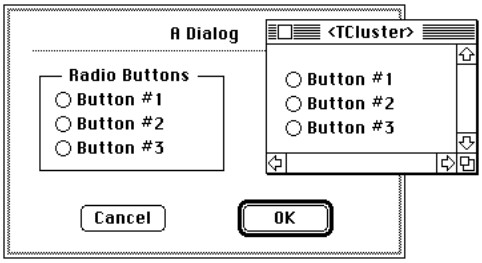
The *Attributes* palette provides an easy mechanism for setting the most frequently used fields of views. Whenever you select a single view, its attributes are displayed in this palette. To change an attribute, just make the change in the palette. Since the Attributes palette is readily available, view editing with Ad Lib is fast and simple.

Occasionally for some classes, you need to change the value of a field that is not available in the *Attributes* palette. For this, use the *Fields* menu. When a single view is selected, this menu contains an item for each inherited class of the view. These menu items invoke a dialog window containing controls to modify all the fields of the corresponding class.

The *Information* palette shows data that are sometimes helpful while editing, such as the coordinates of the cursor with respect to the current window. It also displays the ID of the view under the cursor, and its superview. If you enlarge the palette with the zoom box in the upper right corner, it displays a handy magnifier for those times when you want to get your pixels just right.

Editing Subviews

Occasionally, it is difficult to access the views you want to edit. A superview may be obscured by its subviews, or a subview may be clipped by its superviews.

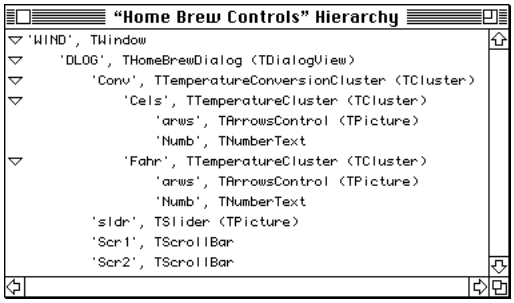


Three commands in the *View* menu are useful in these situations. *Edit Subviews* is enabled when you have selected a single view. This command opens a scrollable window containing the subviews of the view you have selected. You can edit the view's subviews in this window or in the original view. This is particularly handy for views whose extent is clipped by a scroller. You can scroll and resize a subview window to access all of a large view.

Normally, you cannot make the window any larger then the view it represents. However, occasionally you may need access to subviews that are hidden outside a view's frame. When you hold down the *command* key (⌘) as you resize a subview window, you can make the window any size necessary to reveal such views.

The *Subview Access* command is helpful in cases when you are only interested in editing views contained directly in the window. Subview access mode, when on, allows you to select any subview within your window directly. Normally, this mode is on. When it is off, you can only select the topmost views in the window. Occasionally, you may want to use the *Edit Subviews* command on the topmost view.

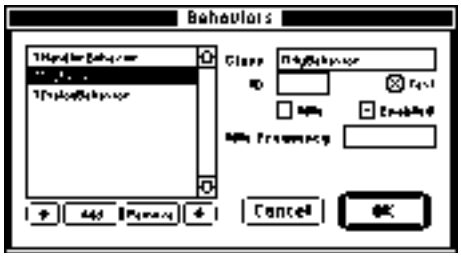
If you are having trouble selecting a particular view in the middle of a hierarchy the *control/arrow* keys are sometimes helpful. When the *control* key is held down the *arrow* keys move the selection around in the view hierarchy. To select the superview of the current selection, hold down the *control* key and type the *up-arrow* key. *Down-arrow* selects the first (i.e., rear-most) subview, and *left* and *right arrow* select the previous and next subview of the selection's superview.



The *Show View Hierarchy* command opens a hierarchy window for the current view. The hierarchy window shows all of the views in outline form. By selecting and dragging items in the hierarchy window, you can change the order and nesting of the views displayed. Using the hierarchy window, you can also locate views that have moved outside of their superviews.

Adorners & Behaviors

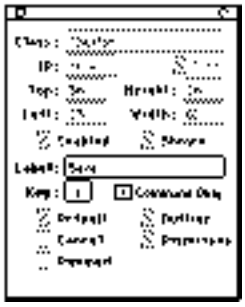
Adorners and behaviors are added to views through their respective commands in the *Object* menu. Both commands invoke similar dialogs for editing a view.



To add new adorners or behaviors select the *Add* button. Items appear in a list by priority and are rearranged with the arrow buttons. The first item in the list is called first by MacApp when your program is running. To customize a selected item, enter the new class name in the *Class* field.

MacApp Extensions

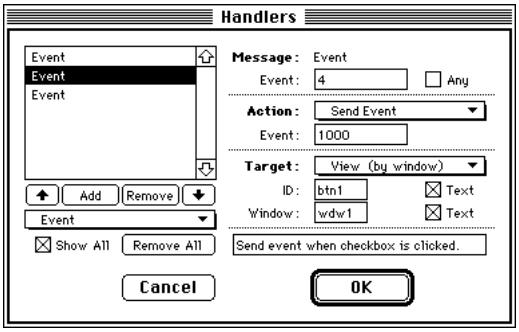
Ad Lib includes custom behaviors to provide some special features. The source code for these extensions is included with Ad Lib.



- In the *Attributes* palette, you can associate a key with buttons, radio buttons, and check boxes. This adds a *TCmdKeyBehavior* to the control and a *TBroadcastCmdKey* behavior to the control's window. When the user types this key (in upper or lower case), the control acts as if it has received a mouse command. For example, buttons in the alerts in MPW respond to the first letter in their labels.

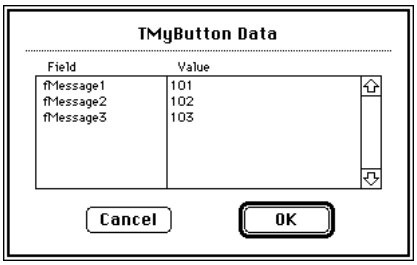
The controls can respond to any key event, or only to key events that occur when the *command* key (⌘) is down.

Ad Lib makes this function simple to provide to your users.



- *Handlers* are powerful, general purpose behaviors that allow your views to respond to specific messages in MacApp. A handler responds to a message by sending new messages to various objects within your program. For more information, refer to the *Handlers* section in the *Option Menu* chapter.

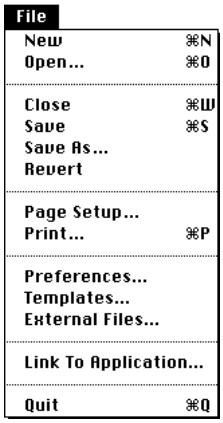
- MacApp supports the use of behaviors in all classes descending from *TEventHandler* such as views and applications. Behaviors are added to views in their resources, but normally the only way to add behaviors to applications is by adding code directly to your application. Ad Lib alleviates this by allowing you to attach behaviors to an Ad Lib document outside the view resource format. When you select a document window, the *Behaviors* and *Handlers* commands remain enabled. When you add behaviors to your document in this way, a resource of type `'BHVR'` is added to your file to store the behaviors. To automatically add these behaviors to your application, derive your application object from *TBehaviorApp* rather than from *TApplication* directly.



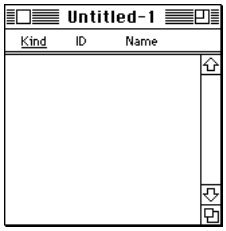
- A final way to extend the standard MacApp objects is to override the objects and add new data fields to them. Ad Lib allows you to create templates for your classes that let you edit your own data fields from within Ad Lib. For more information, refer to the *Templates* section of the *File Menu* chapter, and the *Data* section of the *Object Menu* chapter.

The File Menu

The File menu contains items to create a new document or open, close, save, revert, or print an existing document. You can also set the preferences, edit templates, add external files, link to running applications, and quit the program.



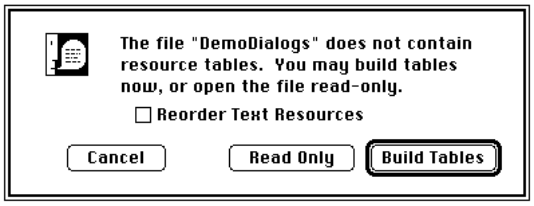
New



Creates a new untitled Ad Lib document. You can have as many documents open at one time as memory allows. The Ad Lib document window lists all of the view resources in a file. Views are displayed sorted by kind (i.e., view or window), resource ID, or resource name.

Open

Opens a file on disk as an Ad Lib document. If the file has never been saved by Ad Lib, you must build the *resource reference tables*, or open the file read-only. For more information about the resource reference tables, refer to the *Text Resource Management* section of the *Overview*.



❑ Reorder Text Resources

Sets all text resource ID values to the default values when the tables are rebuilt. This allows you to organize files containing inconsistent resource numbering. It also replaces all duplicate resources with a single shared resource. Use the *Preferences* command to set the default resource IDs.

❑ Cancel

Don't open this file.

❑ Read Only

Opens the file with read permission only. This does not build the resource tables and you cannot make any changes to the file.

❑ Build Tables

Build the resource tables before opening the file. The next time you save the file, the tables are saved with it.

If the file has been modified by another program since it was saved by Ad Lib, you are asked if you want to rebuild the resource tables, ignore the changes, or open the file read only.

▲ Ignoring the changes saves time but it may corrupt the resource references if the views are changed by a program other than Ad Lib.

❑ Ignore

Ignores the changes and uses the present tables. This saves time but it may corrupt the resource references if the views are changed by a program other than Ad Lib.

❑ Read Only

Opens the file with read permission only. This does not rebuild the tables and you cannot make any changes to the file.

❑ Build Tables

Rebuilds the tables before opening the file. The next time you save the file, the resource tables are saved with it.

Close

Closes the current window. This can be a view window, subview window, or the document. The current window is the window in front of all others except the Ad Lib palettes. When closing a document that you have changed, you are first asked if you want to save it. Selecting the *Close* command performs the same function as clicking in the window's close box, if it has one.

Save/Save As

Saves the current document to disk and updates all text resource references to valid values. The current document is the document that owns the current window (see *Close* command). The text resource references are the IDs and indices of the `'STR#'` and `'TxSt'` resources referenced by each view. If you change the text of a view, Ad Lib releases the string resource index used by that view. When you save the file, Ad Lib searches the string list resource for each changed view. If it finds an identical string the view is made to share the string with other views, otherwise it creates a new string for the view.

The *Save* command overwrites the old copy of the document on disk. The *Save As* command saves the document as a new file on disk with a new name.

Revert

Discards all changes to the document since the last time it was saved. This command cannot be undone.

Page Setup/Print

The *Page Setup* command allows you to select the page settings for printing.

The *Print* command prints the current window.

Preferences

Customizes the default text and other resource IDs for new documents and the current document, and saves the current display and grid settings.

- Set Defaults

  You can set the default IDs for all new Ad Lib documents, or for the current document.

- Resource ID

  The resource ID to use for the resources identified by the *Item* menu.

- Item

  The type of resource ID you are changing with the *Resource ID* field.

- Save Current Display Mode

  Saves the current view display settings as the new defaults. This includes the *Outline, Preview, Actual Window,* and *Subview Access* modes.

- Save Current Grid Settings

  Saves the current grid settings as the new defaults. This includes all of the options in the *Grid* dialog.

- Cancel

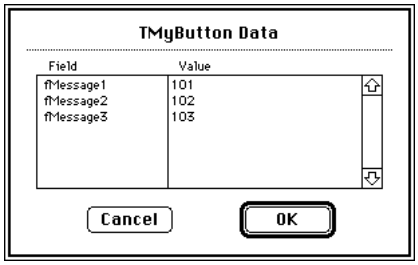
  Closes the dialog without changing the preferences.

- OK

  Closes the dialog and changes the preferences.

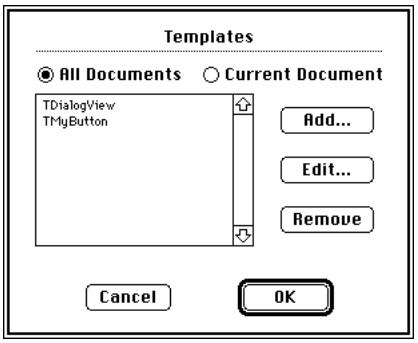
Templates

This command lets you create and edit custom data templates for your classes that are derived from standard MacApp classes. Normally the only class fields you can edit within Ad Lib are those defined by MacApp. For instance, the MacApp class TView contains fields to control the behavior of the Macintosh cursor as it passes over the view in your program. With Ad Lib you can edit these fields using the *TView* item in the *Fields* menu. Similarly you can edit any field of any class defined in MacApp. Sometimes you add custom fields to your derived classes that you would like to edit in Ad Lib and have the values saved in the resource with the object. You can do this by creating a *template* for your custom class. You create a template for a class with a unique class name. Whenever you create an object in Ad Lib and assign it a class name that matches the name of a template, the template is used for that object. For example, if you have a class named *TMyButton* that is derived from the MacApp class *TButton*, you create a template named TMyButton to add custom fields to the class. When you change the class name of a button to TMyButton, the fields described in the template are added to the button. The value of these fields are edited with the *Data* command in the *Object* menu, or through the *Fields* menu. If you change the class name again the template fields are lost.



▲ When reading a file, Ad Lib looks for a template for each custom class that it reads. If a template exists, it is used to figure out how much extra data to read for the object. Without the correct template, it is impossible to determine how much data to read for custom objects. For this reason, it is very important to keep the templates together with the view data. Otherwise, Ad Lib is not able to read the view resource.

You can create templates for either the current document alone, or for use in all documents you edit with Ad Lib. In either case, when a template is used it is placed in the document file as a `'CLSS'` resource. Templates that are currently being used within your document cannot be edited or removed.



- Add

  Adds a new template with the *New Template* dialog described below.

- Edit

  Edits the selected template using the *Edit Template* dialog described below. You cannot edit a template that is currently being used by your document.

- Remove

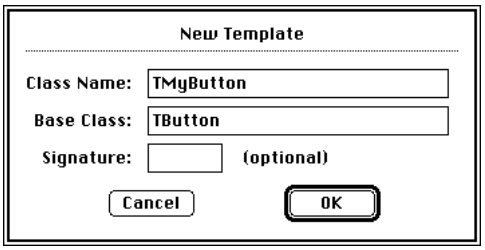
  Deletes the selected template. You cannot remove a template from a document that is currently using the template.

- Cancel

  Closes the dialog without changing the templates.

- OK

  Closes the dialog and makes the changes to the templates.



The *Add* button invokes the New Template dialog. This dialog is used to define your custom class and its relationship to other classes.

- Class Name

  The name of your custom class.

❑ Base Class

The name of the class your custom class is derived from. This can either be a standard MacApp class or a class described in another template. If your class is derived from a custom class that doesn't have a template, use the name of the first parent class that has a template or is a standard MacApp class.

❑ Signature

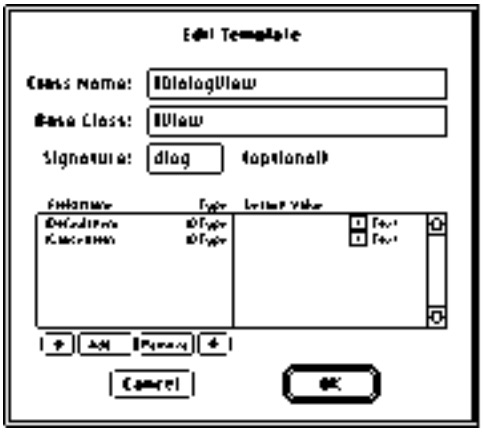
This field is optional. If you leave it blank, Ad Lib automatically uses the signature of the parent class. Signatures are four-character identifiers used within MacApp's view resource format, along with class names, to identify the class of objects being read. If you specify a signature for your template, Ad Lib uses that signature when it saves your class. Normally you can leave this field blank.

❑ Cancel

Closes the dialog without creating a new template.

❑ OK

Closes the dialog and creates a new template.



The *Edit* button invokes the Edit Template dialog. This dialog is used to modify the template's attributes and to edit its fields.

❑ Class Name

The name of your custom class (see above).

❑ Base Class

The name of the class your custom class is derived from (see above).

❑ Signature

The signature of your custom class (see above).

- Field List

  Displays the fields of the template. Fields are listed in the order in which they are saved in the resource file.

- Default Value List

  Displays the default values of the fields in the template.

- Arrow Buttons

  Change the order of the field and default value selected in the list.

- Add

  Adds a new field to the list with the *Add New Field* dialog described below.

- Remove

  Deletes the selected field from the list.

- Cancel

  Closes the dialog without changing the template.

- OK

  Closes the dialog and makes the changes to the template.

The *Add* button invokes the Add New Field dialog.

- Name

  The name of the field.

- Type

  The data type of the field.

- Cancel

  Closes the dialog without adding a field.

- OK

  Closes the dialog and adds the field you have described.

A custom user class that has extra fields must override certain member functions to read the fields from files at the correct time. Objects that are derived from TView must override the *ReadFields()* member. All other objects must override the *ReadFrom()* member. In either case, the overridden function must first call the inherited function, then read the custom fields. All types supported by Ad Lib for custom fields, have corresponding read functions in the *TStream* class in MacApp (e.g., ReadLong and ReadString).

For example:

```
pascal void TMyView::ReadFields(TStream* aStream)
{
      inherited::ReadFields(aStream);
      fShortParam = aStream->ReadInteger();
      fLongParam = aStream->ReadLong();
}

pascal void TMyAdorner::ReadFrom(TStream* aStream)
{
      inherited::ReadFrom(aStream);
      fStringParam = aStream->ReadString();
}
```

External Files

Adds the resources in external files to those considered by Ad Lib when it searches for the resources referred to by views. For instance, the MacApp class TPicture expects to find a `'PICT'` resource to display. Ad Lib displays the correct picture for the object if it can find the resource. If the `'PICT'` resource is in the Ad Lib file containing the view, or within one of the document's external files, the correct picture is displayed.

You can specify external files for all documents in Ad Lib or just the current document.

- Add

  Adds a new external file.

- Remove

  Deletes the selected file.

- Cancel

  Closes the dialog without changing the external files.

❑ OK

Closes the dialog and configures the external files.

Link To Application

Links a running application to an Ad Lib document. When Ad Lib saves a document with a linked application, it sends all of the changes to the linked application to save in its resource fork. This reduces development time by eliminating the need for a build and run cycle to test a view change. Typically you will want to link your document to the application that contains the views you are editing. When you save the file in Ad Lib, you are able to immediately switch to the linked application and try out your new views. This command presents the Macintosh PPC browser for you to use to select the application to link to the current document.

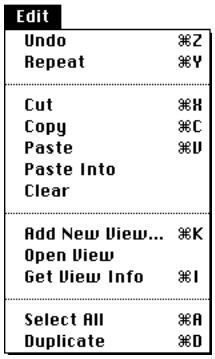
To support the linking feature, your application must include the *TAdLibLink* behavior and ensure that it is not dead-stripped by the linker. Your application must also be derived from the class *TBehaviorApp*, and you must include the file *AdLibLink.r* in your Rez build. All of the files needed for this are contained in the Ad Lib folder.

Quit

Closes all documents and quits the application. You are first asked if you want to save any documents you have changed.

The Edit Menu

The Edit menu contains the standard Macintosh editing commands—Undo, Cut, Copy, Paste, Clear, Select All, and Duplicate. It also contains commands to paste into a specific view, and to edit views in a document.



Undo

Reverses the last operation you performed. Most, but not all, operations can be undone.

Cut/Copy

Moves the selected window or views, or a copy of them, to the clipboard. Ad Lib does not convert the clipboard types into an external form, nor does it copy any referenced resources. For example, the MacApp class TPicture refers to a resource of type 'PICT'. If you copy a TPicture item from one document to another, you will also have to copy the 'PICT' resource using a resource editor.
The *Cut* command removes the selected items from the document, the *Copy* command does not.

Paste

Copies a new window or views from the clipboard into the current document. New views are inserted at the highest level in the view hierarchy. To insert views at some other level, use the *Paste Into* command. If the window is large enough, the new views are placed at the same location they were at in their previous superview.

▲ Care should be taken when working with non-window views, that only one view is placed at the top of the view hierarchy. Since *Paste* places new views at the top of the hierarchy this can be a problem. *Paste Into* is a better command to use in this situation.

Paste Into

Copies new views from the clipboard into the currently selected view. Only one view should be selected for this command to be enabled. If the selected view is large enough, the new views are placed at the same location they were at in their previous superview.

This command is useful for changing the view hierarchy. To move a view from one superview to another, *Cut* the view, select the new superview, and select *Paste Into*.

Clear

Removes the selected window or views from a document.

Add New View

Adds a new view to the current document.

❑ Name

The name of the new view resource.

❑ ID

The ID of the new view resource.

- Kind

  You can add either a window or a view to your document. Windows are the highest item in the view hierarchy. When you add a view, Ad Lib creates a “place holder” window, into which you can add views. MacApp 3 requires a single view to be at the top of a hierarchy. If you place multiple views directly into the place holder window, they are enclosed in a simple TView object when you save the file.

- Cancel

  Closes the dialog without adding a new view.

- OK

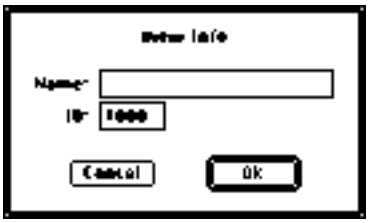
  Closes the dialog and adds the new view.

Open View

Opens the window or view currently selected in the document window. This can also be accomplished by double clicking on the item in the document window.

Get View Info

Changes a view's name and resource ID.



- Name

  The new name of the view resource.

- ID

  The new ID of the view resource. You will only be able to use a particular ID for one view or window in a document.

- Cancel

  Closes the dialog without changing the information.

- OK

  Closes the dialog and changes the information.

Select All

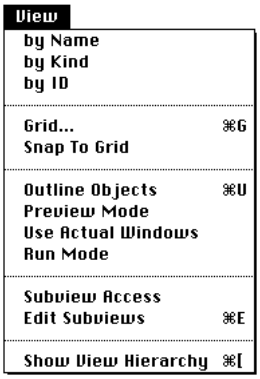
Selects all views in the current window.

Duplicate

Copies the currently selected views and places them slightly offset from the originals. You can also duplicate views by holding down the *option* key as you move them with the mouse.

The View Menu

The View menu contains items to select the display mode for the document and editing windows, control the grid, and work with subviews.

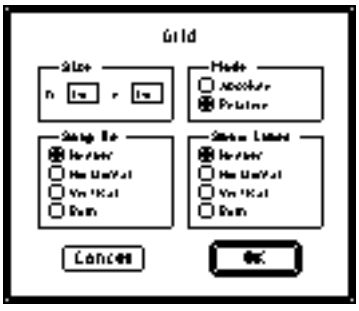


by Name/Kind/ID

Selects the order in which views are displayed in the Ad Lib document window. Views are sorted alphabetically by resource name, numerically by resource ID, or by whether they are windows or simple views. The view display order is also selected by clicking the mouse in the column labels in the document window.

Grid

Sets the grid parameters.



❑ Size

The grid spacing in pixels horizontally and vertically.

- Mode

  The grid mode is either absolute or relative. In absolute mode, objects snap to the grid intersection points. In relative mode, objects always maintain a constant distance from intersection points.

- Snap To

  Controls whether objects “snap” to grid intersections.

- Show Lines

  Controls whether the grid lines appear or remain invisible.

- Cancel

  Closes the dialog without changing the grid.

- OK

  Closes the dialog and changes the grid.

The default grid settings can be set with the *Save Current Grid Settings* option in *Preferences.*

Snap To Grid

Toggles the grid’s “snap” mode without invoking the grid dialog. The snap mode is temporarily toggled by holding down the *command* key (⌘) while dragging or resizing a view. The default grid snap is changed with the *Save Current Grid Settings* option in *Preferences.*

Outline Objects

Toggles the *outline* display mode. When on, views are drawn framed by a dotted line to show their extents. This is helpful if you want to know exactly what the boundaries of your views are when they are surrounded by white space. The default for this mode is changed with the *Save Current Display Mode* option in *Preferences.*

Preview Mode

Toggles the *preview* display mode.

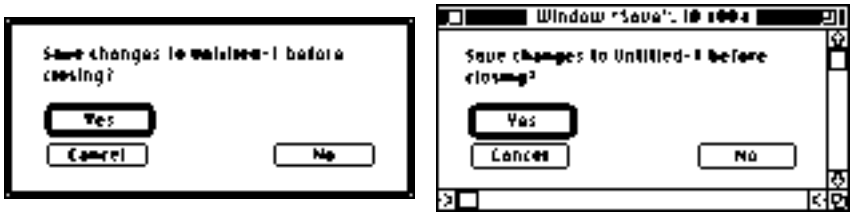
Ad Lib doesn’t know how to draw views based directly on such classes as TView, TControl, and TGridView. In MacApp, they have no drawing code. Normally, you will override these classes in your program to provide the drawing functions.

With preview mode off, unless these views have subviews or contain visible adorners, Ad Lib draws them with a diagonal pattern to show where they are. With preview mode on, these views are not drawn at all, to give you a more accurate idea what the window looks like when used in your program.

Turning preview mode on also turns off *outline* mode and the display of grid lines. You can turn these back on manually, while in preview mode, if you want. The default for this mode is changed with the *Save Current Display Mode* option in *Preferences.*

Use Actual Windows

Toggles the *actual windows* mode.



Actual Window **"Place Holder" Window**

When the actual windows mode is on, Ad Lib uses the window you have specified for your view as the editing window. This allows you to see just how your views look in your application. When the actual windows mode is off, Ad Lib uses a scrollable “place holder” window for the editing window. This allows you to move or scroll the window without affecting your view. The default for this mode is changed with the *Save Current Display Mode* option in *Preferences.*

Run Mode

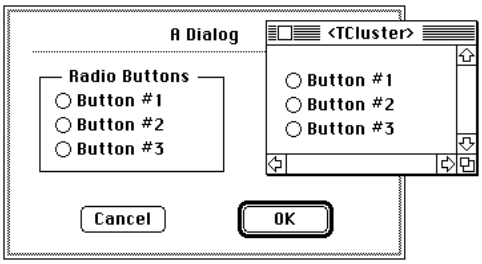
Toggles *run* mode. When run mode is on, your views behave as if you are running your program. This allows you to see how your views look and make sure that things like the default window target and tabbing order are correct. When run mode is off, your views can be edited.

Subview Access

Toggles the *subview access* editing mode. When on, subview access mode allows you to select any subview within your window directly. When off, you can only select the topmost views in the window. When you want to select a view completely obscured by its subviews, turn subview access off. By selecting the topmost view and using the *Edit Subviews* command you are able to access any view in your hierarchy. The default for this mode is changed with the *Save Current Display Mode* option in *Preferences.*

Edit Subviews

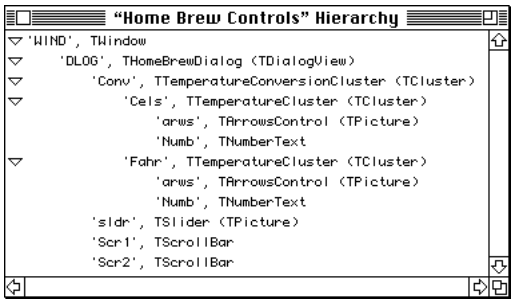
Opens a scrollable window containing all the subviews of the view you have selected.



You can edit the view's subviews in this window or in the original view. You can scroll and resize a subview window to access all parts of the original view. This is useful for views whose extent is clipped by a superview, such as a scroller.
Normally, the size of the subview window is limited to that of the original view. Occasionally, you may find it necessary to place subviews outside the extent of their superviews so they won't appear in your window. You can access these subviews by holding down the command key (⌘) while resizing the subview window. This allows you make the subview window larger than the original view.

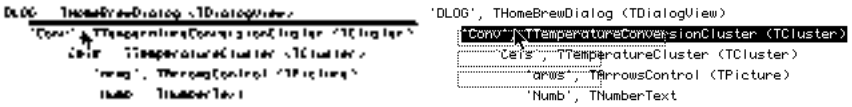
Show View Hierarchy

Opens a *view hierarchy* window for the current view.



The hierarchy window shows all of the views in a window in an outline form similar to that used by the Macintosh Finder for non-icon views. Views containing subviews appear with a small arrow to the left of their entry. By selecting this arrow, you control whether or not the subviews are displayed in the hierarchy window.

The views in the window appear in back to front order. In other words, the first views in the list are drawn behind the others when the window is displayed by MacApp.

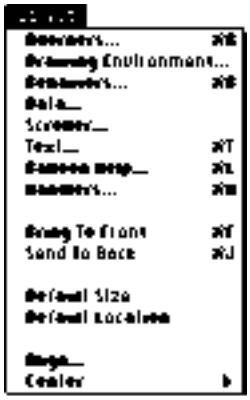


Option Key Up **Option Key Down**

By selecting and dragging items in the hierarchy window, you can change the order and nesting of the views displayed. Normally the selected items are placed between the views you are dragging over. By holding down the *option* key while dragging, you can place the selected items within a view as subviews.

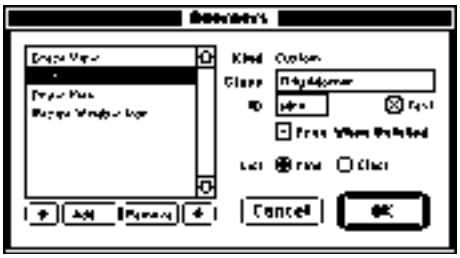
The Object Menu

The Object menu contains items to view the class hierarchy and edit the fields of view objects. View objects are edited through a variety of dialogs. Objects can also be rearranged with respect to one another.



Adorners

Adorners are added to a view with the *Adorners* dialog.



- Adorner List

  Displays the adorners of the view. Adorners appear in their list by priority. The first item in the list is assigned the highest priority in the view's adorner list. Shared adorners are displayed in italic text.

- Arrow Buttons

  Changes the priority of the adorner selected in the list.

- Add

  Adds a new adorner to the list with the *New Adorner* dialog described below.

- Remove

  Deletes the selected adorner from the list.

- Kind

  Displays the kind of adorner selected. Examples of kinds of adorners are Draw, Erase and Frame.

- Class

  The class name of the adorner selected. To customize an adorner, enter the new class name in this field.

- ID

  The ID of the adorner selected.

- List

  Lists the adorners by either their kind or their class name.

- Cancel

  Closes the dialog without changing the adorners.

- OK

  Closes the dialog and changes the adorners.

The *Add* button invokes the New Adorner dialog.

- New Adorner List

  Displays the kinds of adorners available to add. Select one item in the list.

- Use Shared Adorner

  Indicates that a *shared* adorner is used. Shared adorners are the global adorners MacApp maintains. When you use a shared adorner, no new objects are instantiated. You can't edit the attributes of shared adorners since they are program globals. Shared adorners are only available for some adorners in MacApp. If a shared adorner is not available for the kind of adorner you have selected, this option is unavailable.

- Cancel

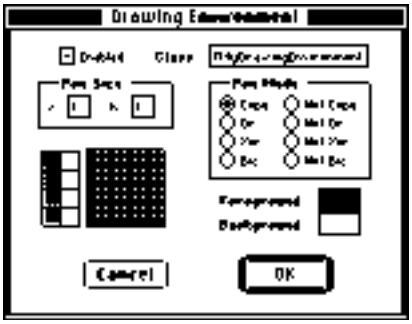
  Closes the dialog without adding a new adorner.

- OK

  Closes the dialog and adds the new adorner.

Drawing Environment

A view may or may not contain a drawing environment to configure special QuickDraw pens and colors.



- Enabled

  Indicates whether or not the view contains a drawing environment object.

- Class

  The class name of the drawing environment object. To customize the drawing environment, enter the new class name in this field.

- Pen Size

  The size of the QuickDraw pen.

- Pen Mode

  The transfer mode of the QuickDraw pen.

- Pen Pattern

  The pattern of the QuickDraw pen. There are two ways to set the pattern, to select one of the supplied patterns, click in the pattern palette on the left side of the dialog. To edit the pattern one pixel at a time, use the enlarged view next to the palette.

- Foreground

  The RGB (i.e., red, green, and blue) foreground color of the drawing environment.

- Background

  The RGB background color of the drawing environment.

❑ Cancel

Closes the dialog without changing the drawing environment.

❑ OK

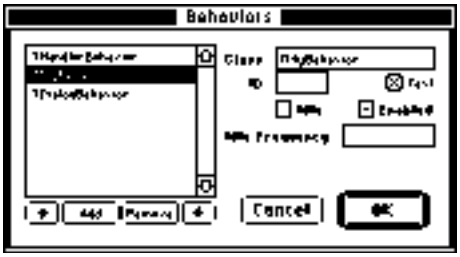
Closes the dialog and changes the drawing environment.

Behaviors

Behaviors are added to an object through the *Behaviors* dialog.

Behaviors added to the Ad Lib document window are saved in a separate `BHVR'` resource. These behaviors are added to your application object automatically by deriving your application object from *TBehaviorApp* rather than from *TApplication* directly.

▲ Do not add behavior classes containing additional data, such as *TDialogBehavior* or *TTabber*, through the *Behaviors* dialog unless you have created a template for the class. *TDialogBehavior* is supported by the dialog functions in the *Attributes* window when a window is selected. You may, however, customize such a behavior already present in an object by changing its class name.



❑ Behavior List

Displays the behaviors of the view. Behaviors are displayed in the list in the same order as they occur in the view. The first behavior in the list is the first behavior called by MacApp when your application is running.

❑ Arrow Buttons

Changes the priority of the behavior selected in the list.

❑ Add

Adds a new behavior of class *TBehavior* to the list.

❑ Remove

Deletes the selected behavior from the list.

- Class

  The class name of the behavior selected. To customize a behavior, enter the new class name in this field.

- ID

  The ID of the behavior selected.

- Idle

  Indicates that the selected behavior receives idle messages from MacApp.

- Enabled

  Indicates that the selected behavior is called by MacApp when your application is running.

- Idle Frequency

  The frequency at which the behavior receives idle messages if *Idle* is selected above.

- Cancel

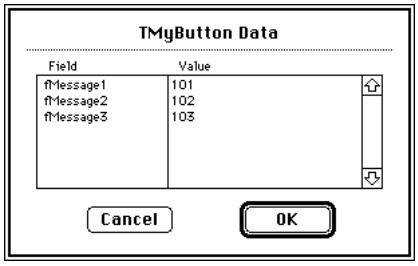
  Closes the dialog without changing the behaviors.

- OK

  Closes the dialog and changes the behaviors.

Data

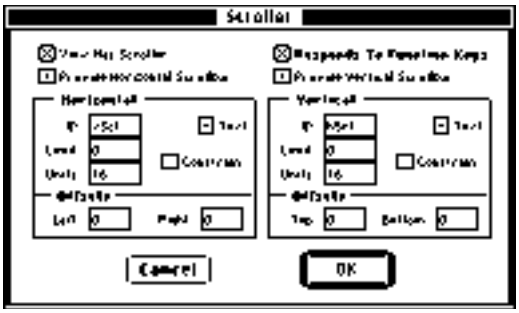
If the currently selected object is using a class template, the Data command is available to edit the fields specified in the template. This works for views as well as behaviors and adorners.



Scroller

A scroller is a special view that scrolls its subviews. Normally, a scroller contains only one subview. Ad Lib treats scrollers as an attribute of the view to be scrolled. The *Scroller* dialog allows you to add a scroller as a superview of a view. You can also add scroll bars at this time.

▲ Care must be taken to assign unique identifiers to the scroll bars. When MacApp reads a scroller from a file, it looks for scroll bars by ID, so the identifiers must be unique or the scroller won't find its scroll bars.



❏ View Has Scroller

Indicates whether or not the view's superview is a scroller.

❏ Responds To Function Keys

Indicates the state of the scroller's *fRespondsToFunctionKeys* field.

❏ Provide Horizontal Scrollbar

If selected, a horizontal scrollbar is added for the scroller.

❏ Provide Vertical Scrollbar

If selected, a vertical scrollbar is added for the scroller.

❏ Horizontal

The parameters of the horizontal scrollbar.

❏ Vertical

The parameters of the vertical scrollbar.

❏ Cancel

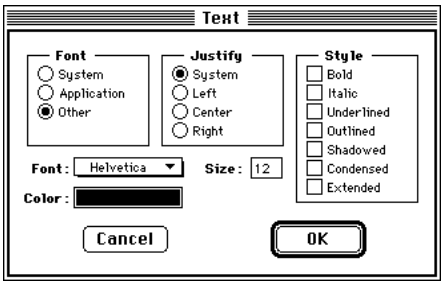
Closes the dialog without changing the scroller.

❑ OK

Closes the dialog and changes the scroller.

Text

The text style of a view or views are modified with the *Text* dialog. Ad Lib automatically creates text style resources and assigns the correct resource IDs to your views. To set the text style of several views to the same values, select all of the views and choose this command.



❑ Font

The font to use for the text. You can choose the current system or application font, or another font available on your system. To choose a font by name, select *Other* and choose the font name in the *Font* popup menu.

❑ Justify

The justification of the text. This option is not available for all view classes. If the selected view has no justification field, this option is unavailable.

❑ Style

The style of the text.

❑ Size

The point size of the text. A value of zero indicates the Macintosh's default text size.

❑ Color

The RGB (i.e., red, green, and blue) color of the text.

❑ Cancel

Closes the dialog without changing the text style.

❑ OK

Closes the dialog and changes the text style.

Balloon Help

Balloon Help resources are assigned to a view with the *Balloon Help* dialog. To use balloon help you must provide `'hdlg'` resources for your application with another program. For more information on balloon help in MacApp, refer to the MacApp documentation.

- Resource ID

  The ID of the `'hdlg'` resource to use for balloon help for the view. A value of -1 indicates that no help resource is available.

- Index

  The index of the help information in the `'hdlg'` resource.

- Handles Help

  The view handles help messages.

- Subviews Handle Help

  The view passes help messages to its subviews.

- Cancel

  Closes the dialog without changing the balloon help assignment.

- OK

  Closes the dialog and changes the balloon help assignment.

Handlers

Handlers are special behaviors that extend MacApp to allow your views to respond to specific messages. A handler responds to a message by sending new messages to various objects within your program. Handlers are very useful for many functions. You can perform simple tasks without overriding classes and writing new code. A partial list of things you can do with handlers includes:

- Perform a menu function from a button, or other control, by sending a menu command to the target chain when the control gets a mouse down message or an event such as mButtonHit (i.e., #3).

- Fast prototyping by adding application handlers to enable menus in response to the menu setup message. Other handlers, in response to menu command messages, open the appropriate windows.
- Create objects that respond to custom event numbers, and add handlers to buttons to send those events, and control your objects.

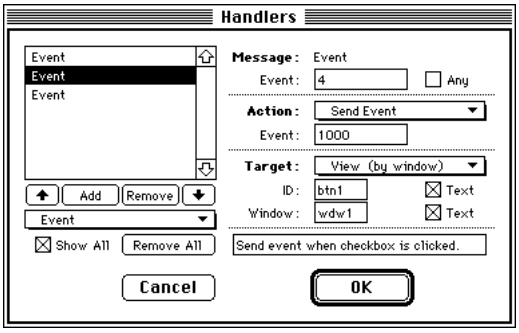
When you add one or more handlers to an object, Ad Lib adds a behavior of type *THandlerBehavior* to your view. For your convenience, the source code for THandlerBehavior is included with Ad Lib.

Handlers respond to specific messages as you might in your application by overriding a view's methods such as *DoEvent()* or *DoMouseCommand()*. For messages with a parameter, such as a key code, you can specify a specific parameter value or have the handler respond to all parameters. Within a single object, several handlers can respond to the same message or to different messages. Handlers are called in the order in which they appear in the *Handlers* dialog.

In response to a message, a handler performs some action. Actions either send messages such as events or menu commands, or perform simple functions, such as opening a window or enabling a menu. A handler action cannot invoke another handler in the same object. A user defined action can be performed by overriding the *DoUserAction*() member of *THandlerBehavior*. To modify the handler class, change the class name of the *THandlerBehavior* object using the *Behaviors* dialog.

Most actions are performed on a target object. The target can be the application object, the document, the head of the target chain, or some other object such as a view specified by its identifier. If an object contains a handler for a specific message, the handler catches the message before the object gets it. A special action, *Forward Message*, passes the current message directly to the target. If you want the object containing a handler to receive the same message, add a handler to the object to forward the message to *Self*. The exception to this is the *Initialize* message. A handler cannot block initialization, it can only perform some action at that time.

Since they are implemented as behaviors, handlers are added to your application object by adding them to the Ad Lib document window and making your application object a descendant of *TBehaviorApp*.



❏ Handler List

Displays the handlers of the view. Handlers appear in the list in the same order as they are executed.

❏ Arrow Buttons

Changes the execution order of the handler selected in the list.

❏ Add

Adds a new handler to respond to the type of message selected in the *Message* popup menu.

❏ Remove

Deletes the selected handler from the list.

❏ Message Popup

Selects the type of handler to add with the *Add* button. If *Show All* is not checked, this menu also selects the type of handlers to display in the list.

❏ Show All

If checked, all types of handlers appear together in the list. If not, only the type of handler selected in the *Message* popup appears.

❏ Remove All

Deletes all of the handlers currently displayed in the list.

❏ Message

Displays the message that the selected handler responds to. Most message types have an associated parameter to specify exactly which message to respond to. For example, the handler for the *Menu Command* message allows you to select which command to handle. To handle all commands, select the *Any* check box.

❏ Action

The action the handler takes when it receives its message. Many actions have a parameter. For example, the *Send Event* action allows you to specify the event number to send.

- Target

  The target the handler performs its action on. Not all actions require a target. If the selected action does not need a target, this item is unavailable. Some target options have a parameter. For example, the *View* target allows you to specify the ID of the view. Usually, for each message you have handlers for, you will want to provide one last handler to forward the message to *Self*. This gives MacApp a chance to perform the usual operations for the message. If your handlers are causing strange effects, verify you're not blocking a needed message by not forwarding it.

- Comment

  Displays the comment for the selected handler. You add comments to handlers to provide documentation of what they are doing.

- Cancel

  Closes the dialog without changing the handlers.

- OK

  Closes the dialog and changes the handlers.

Bring To Front

Places the selected views in front of all others in their superview.

Send To Back

Places the selected views behind all others in their superview.

Default Size

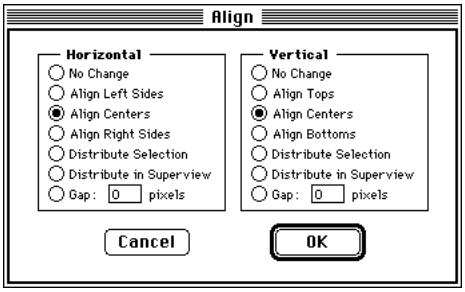
Many views have a natural size that makes sense given their state. For example, icons are almost always 32x32 pixels, and pictures have a size specified in their 'PICT' resource. *Default Size* resizes views to their natural size, if they have one.

Default Location

The Macintosh human interface guidelines specify specific locations for elements of alerts and some dialogs. *Default Location* move icons, static text, and buttons to these locations. Since the default location for buttons is below all the other views in window, you may want to use this command twice. Once for the icon and text, and a second time for the buttons. After the window elements are in position you can select *Default Size* for the window to resize it for the contents.

Align

Aligns the edges or centers of views horizontally or vertically. Views can also be distributed with equal space between them, either within their superview, or within the bounding box of all selected views. To use the *Align* function, first select multiple views within the same superview.



❏ Horizontal

The horizontal alignment and distribution options.

❏ Vertical

The vertical alignment and distribution options.

❏ Cancel

Closes the dialog without aligning or distributing the views.

❏ OK

Closes the dialog and aligns or distributes the views

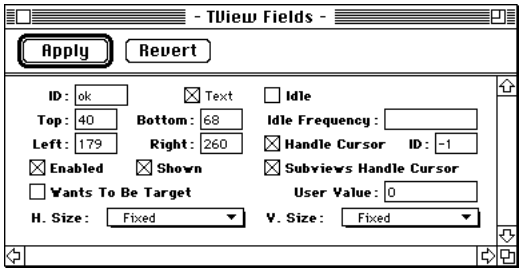
Center Horizontally/Vertically

Centers views within their superview along the horizontal or vertical axis.

The Fields Menu

The Fields menu contains items to view the class hierarchy and edit the fields of objects.

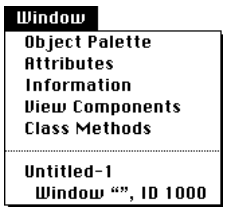
When you have selected a single view, this menu contains an item for each class the view is derived from. Selecting one of these items opens the *Fields* window.



The Fields window displays all of the fields for one class at a time. You can change the class by selecting a new item in the Fields menu. This provides a mechanism to change the fields not available in the *Attributes* palette. If a class contains no fields, it is appears dim in the menu.

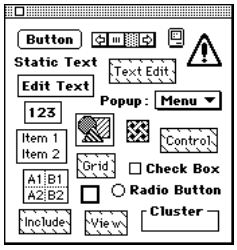
The Window Menu

The Window menu contains items to open Ad Lib's editing palettes and custom windows, and to select any windows created by the application.



Object Palette

The *Object Palette* is used to add new views to a window.



All native MacApp view classes are represented in the object palette. To add a new view to the front window, simply drag a view from the palette to the window. As you drag, a dotted outline follows the cursor. If *Subview Access* mode is on, the new view is placed within whatever view is under the top-left corner of the outline. Otherwise, the new view is placed directly in the window.

Attributes

The *Attributes* palette provides a convenient mechanism for setting a view's most frequently used fields.

Whenever you select a single view, its attributes are displayed in this palette. To change an attribute, just make the change in the palette. For your convenience, some functions available through the Attributes palette perform multiple operations. For example, the Outline option for default buttons in dialogs, when activated, expands the button's extent and insets it by the same amount. It then adds a round rectangle adorner, and sets up a drawing environment with the correct pen size.

A round indicator on the right side of the palette's title bar "locks" the text selection in the palette. Normally, with the lock off, each time you select a new view, any text selected in the Attributes palette is deselected. If you intend to change the same field in several views, lock the selection by selecting this control. When the selection is locked, the same field is selected in the Attributes palette each time you select a new view.

▲ It is important to note one potential pitfall when you are editing a view's attributes with one of the EditText fields in the *Attributes* palette. In this situation, all key events and some editing commands go to the EditText object, not to the object you are editing. Therefore, cut/paste commands and arrow key strokes, for example, affect the text field you have selected, not the view you are expecting to change. If you find that your edit commands or keystrokes do not seem to be working, make sure you're not editing a text field, in the *Attributes* palette, by mistake.

Information

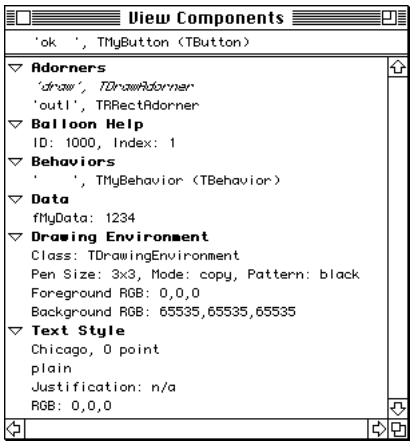
The *Information* palette shows view data sometimes helpful when editing.

The mouse position appears in the current window's coordinates. When the mouse is over a view, the view's identifier and the identifier of its superview are displayed.

Finally, when the window is zoomed to full size, a pixel magnifier shows an enlargement of the area around the cursor. This is sometimes useful for positioning views in relation to others.

View Components

The *View Components* window shows lists of all the components of the currently selected view in a single place. View components are adorners, behaviors, the drawing environment, and the other things that are attached to views.



Class Methods

The *Class Methods* window lists all of the methods defined for the class of the currently selected view. You can invoke the editor for the method's source code by selecting the method and choosing one of the buttons. To use this window you must be running Object Master™*, and the class must be in the current Object Master project.

- ❏ Template

  Opens the text file containing the source for the definition of the selected method.

- ❏ Body

  Opens the text file containing the source for the implementation of the selected method.

- ❏ Browser

  Opens a browser for the displayed class.

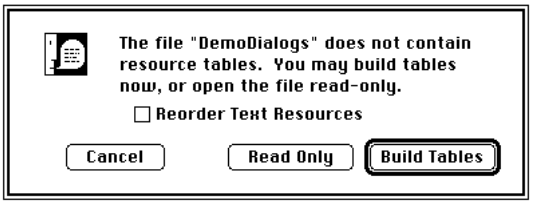
* Object Master is an object-oriented software editing tool from ACI US, Inc.

Ad Lib and ViewEdit

ViewEdit is Apple's view editing software included with MacApp. There are a few issues you should be aware of if you intend to edit view files created in ViewEdit with Ad Lib, or vice versa.

Editing ViewEdit files with Ad Lib

When you first open a file created by ViewEdit in Ad Lib, you are asked to build the resource tables.



Ad Lib must do this to perform its automatic resource management. You may also want to set the resource IDs to the defaults. This ensures that new views use the same resource numbering scheme as the existing views. Alternatively, you can configure the resource numbering for the file with the *Preferences* command.

▲ Any time you edit the views with ViewEdit, make sure you let Ad Lib rebuild the resource tables before using it on the file.

ViewEdit has one idiosyncrasy that causes occasional problems. For many views ViewEdit's default `'STR#'` resource (i.e., number 253) is not found in the file but rather, in MacApp itself. When you edit a view containing this reference, Ad Lib tries to be helpful and adds the resource to your file. When you attempt to combine your view file with MacApp's resource, you get an error because there are two resources for `'STR#'` number 253. The easiest way around this problem is to build the reference tables using the default resource IDs when you first open the file in Ad Lib.

Editing Ad Lib files with ViewEdit

ViewEdit does not support custom adorners or drawing environments, or behaviors of any kind. If you add any of these objects to your views, you cannot open these views with ViewEdit.

AppleEvents

Ad Lib supports the required AppleEvents as well as the `Edit'` AppleEvent supported by most resource editing programs. Using this event, you can instruct Ad Lib to open a specific view within a particular file.

Event:
- Class: aLib
- ID: Edit

Parameter: the resource file
- Keyword: FILE
- Type: fss

Parameter: the resource type (must be 'View')
- Keyword: RTYP
- Type: RSRC

Parameter: the resource ID
- Keyword: RID
- Type: short

Tips and Helpful Hints

General

- If you find that your edit commands or keystrokes do not seem to be working, make sure you're not editing a text field in the *Attributes* palette by mistake. To pass events directly to the selected view, use the *option* key.
- Always assign unique identifiers to the default and cancel items in a dialog.
- Always assign unique identifiers to a scrollers scroll bars.
- The *TDialogView* class is obsolete in MacApp 3, to add a *TDialogBehavior* to a window, select the *Dialog* option in the window's *Attributes* palette. If you really need a TDialogView, you can copy it from a dialog created outside of Ad Lib and paste it into your dialog.

Modifier Keys

- *Shift* while selecting to extend the selection.
- *Shift* while dragging an object to constrain to the horizontal or vertical axis.
- *Shift* while resizing an object to change the size in one dimension only.
- *Shift* with the arrow keys to resize the views in the selection by one pixel.
- *Option* while dragging to duplicate a view.
- *Option* while typing into the Attributes palette to pass the event to the selected view(s).
- *Option* while dragging in the Hierarchy window to move the selection into views as subviews rather than between views.
- *Command* (⌘) while dragging or resizing to temporarily toggle the grid mode.
- *Command* while resizing a *Subview Window* to allow the window to be made larger than the original view.
- *Command* while clicking directly in a window to move the window.
- *Command-Shift* while clicking directly in a window to resize the window.

- *Control* while clicking in a window to send the click to the window, not the view under the mouse.
- *Control* with the arrow keys when no more than one view is selected to select another view in the view hierarchy. Up and down arrows select the superview and first subview; left and right arrows select the previous and next views at the same level as the current selection.

Frequently Asked Questions

When should I use TBehaviorApp in my application?

Your application should descend from TBehaviorApp if you have added any behaviors or handlers to Ad Lib's document window. TBehaviorApp reads these behaviors during initialization and adds them to itself. If you are just adding behaviors or handlers to your views, you do not need to use TBehaviorApp.

Why does Ad Lib keep creating a 'STR#' resource with an ID of 253 in my file?

This is most likely caused by an interaction with ViewEdit. Refer to the section *Ad Lib and ViewEdit* for more information.

I have changed my text resource ID scheme. How do I renumber the resources in my view file?

First set up the resource IDs you want to use for the file in the *Preferences* dialog with the *Current Document* option. Next, close the file and, with a resource editor, delete the `'ALIB'` resource ID 0 from your view file. This removes Ad Lib's time stamp from the file and forces it to rebuild the resource tables. When you open the file in Ad Lib, you will see this dialog. First, select the *Reorder Text Resources* option, and then choose *Build Tables.* After the tables have been rebuilt, all of your view resources will reference the resource IDs defined in the Ad Lib preferences.

The arrow keys do not seem to reliably move my views. What's wrong?

You are probably editing text in the *Attributes* window and the arrow keys are moving the text insertion point. You can pass the key events through to your view by holding down the *option* key while pressing the arrows, or stop editing text.

How do I select a window when it is completely covered by its views?

Hold down the *control* key while clicking in the window.

How to I set the tab order of EditText items in my dialogs?

MacApp tabs EditText items from the back of a dialog to the front. Use the Hierarchy window to stack the views in the order you want them to tab in. To determine what the tab order is, select the view that is the default target of the window and press the *right arrow* key while holding down the *control* key. This advances the selection in the same order as tab does when your program is running.

My program crashes when reading a view containing a TViewTabber. What's wrong?

TViewTabber inherits from TTabber, a behavior that reads its *fRecursive* field from the view stream. Ad Lib doesn't have any way to know that this behavior class reads data from the view resource so it doesn't write any extra data for it. When your program reads the view the tabber reads its data and throws the view stream out of sync. To use a TViewTabber in your views, create a template for it containing a boolean field.

Can I create a scroller with more than one subview?

Yes. Once the scroller has been created it behaves the same as any other view.

How does Ad Lib choose text resource IDs and indices?

Each Ad Lib document contains a set of resource IDs that are used for all new views. Ad Lib also saves default IDs to use for new documents. Both of these ID sets are modified with the *Preferences* command.

String list resources (i.e., `'STR#'`) used by views are set individually using the *Field* menus. Ad Lib never changes the `'STR#'` ID used by a view (unless you ask it to reorder the resources), but it does adjust the index within the list. It does this in order to share as many strings between views as possible, and to eliminate, from the lists, indices that are no longer used.

Text style (i.e., `'TxSt'`) resource IDs used by views are adjusted when views are changed to share as many resources as possible between views, and to eliminate resources that are no longer used.

Ad Lib keeps changing my strings in 'STR#' resources. How do I make it stop?

As discussed above, Ad Lib assumes that it owns any `'STR#'` resource referenced by a view. If you add items to one of these lists, Ad Lib will probably move it on you. Therefore, use separate ID numbers for the string lists you use in your program and those used by your views.

How do I use Ad Lib on a project with multiple view files?

The text resource tables that Ad Lib creates to facilitate text editing cause problems if you are using more than one view file in a project. Ad Lib creates `'ALIB'` resources with IDs 0, 1, and 2 to hold information about text resources in the file. There are also `'ALIB'` resources with IDs that correspond to the IDs of the 'STR#' resources used.

There are at least three things you can do in Rez to work around this.

- If you're not using projector, just exclude the `'ALIB'` resources from your resource include statement:

```
include "xxx.view" not 'ALIB';
```

- If you're using projector you'll probably be using the "not" form for the `'ckid'` resource that projector uses. Another solution is to use Rez with the append flag (`-a` on the command line), and use the delete command:

```
include "xxx.view" not 'ckid';
delete 'ALIB';
```

  However, `"delete"` only works in Rez in the append mode.

- Finally, you can include just the resource types you need from the Ad Lib document. Using the "External Files" function (described in the *File Menu* section) you can place non-view resource types (e.g., `'PICT'`, `'ICON'`, `'MENU'`) in another file (e.g., `xxx.rsrc`) and still access them in Ad Lib.

```
include "xxx.view" 'View';
include "xxx.view" 'STR#';
include "xxx.view" 'TxSt';
include "xxx.rsrc" not 'ckid';
```