

Dialog Director

Version 0.6, April 1997

Copyright © 1996, 1997 Christopher E. Hyde. All rights reserved.

drjekyll@hilight.demon.co.uk

<http://www.hilight.demon.co.uk/>

Contents

Welcome

Release Notes

Installing Dialog Director

Auto Dialogs and Live Dialogs

Event Handlers

- | | |
|-----------------------|---------------------------------------------------------|
| dd auto dialog | - create, display and interact with a dialog window |
| dd calc dialog bounds | - calculate the bounding rectangle of a dialog window |
| dd count dialogs | - return the number of open dialogs |
| dd delete | - close and delete a dialog window |
| dd get | - get a property from a dialog item |
| dd install | - create and initialise the Dialog Director environment |
| dd interact with user | - interact with the front dialog window |
| dd make dialog | - create and display a dialog window |
| dd set | - set a property of a dialog item |
| dd uninstall | - clean up and remove the Dialog Director environment |

Object Classes

- | | |
|----------------------|-------------------------------------------------|
| class dialog | - data used to create a dialog |
| class dialog item | - a dialog item |
| class push button | - a push button item |
| class check box | - a check box item |
| class radio group | - a group of radio buttons |
| class pop up | - a pop-up menu item |
| class list box | - a scrolling, single column text list |
| class text field | - an optionally labelled editable text item |
| class password field | - an opaque editable text item |
| class static text | - a static text item |
| class group box | - an optionally labelled rectangular frame item |
| class pict | - a picture item |
| class icon | - an icon item |
| class dummy | - a dummy dialog item |
| class color picker | - a colour picking item |
| class gauge | - a graphical magnitude indicator |
| class font spec | - a text font specification |

Dependencies

Examples

Known Problems and Limitations

Other Scripting Additions

Please note that this software is provided “as is” and without any express or implied warranties, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose.

Welcome

Welcome to Dialog Director v0.6. This is the April 1997 public release of the Dialog Director scripting addition. Dialog Director allows you to create dynamic dialog boxes directly from your AppleScript scripts providing you with the ability to write applets, droplets and embedded scripts with the power to interact with your users in a friendly and effective manner.

What's more **Dialog Director is FREE** and I intend to enhance and maintain it based on the feedback I receive from you. You may redistribute Dialog Director to anyone as long as you include the complete Dialog Director package and do not charge for it. This means you may charge for your own software, just not extra for Dialog Director. It also means that you may not sell copies or redistribute parts of the Dialog Director package without the author's express permission.

Please use Dialog Director and tell me what you think. The reason that this release is called v0.6 and not v1.0 is because I am still adding features and the API may change (and has already) as a result of your feedback. However, throughout testing it has proved extremely robust and reliable.

Current features include

Dialog item types

- Push buttons - with user defined default button, return & escape key mapping, actions
- Check boxes
- Radio button groups - one & two dimensional automatic layout
- Pop-up menus
- Scrolling text lists - single column, single & multiple selection, keyboard navigable, disableable
- Static text items - any font, size, style & colour
- Edit text items - optional label, forward & backward tabbing, cut, copy & paste, extended keyboard support
- Password text items - as edit text items but display only bullets
- Rectangular frames - optional label, 3 styles (and horizontal & vertical lines)
- Pictures
- Icons - colour or black & white
- **Editable pop-ups**
- **Colour pickers**
- **Gauges (progress bars & barber poles)**

Other

- Dependencies - enable/disable items depending on other items' states
- Optional dialog timeout
- Extensive error checking and reporting
- Returns an easy to read list of item values
- No need for resource files or use of ResEdit
- Six different window styles including movable modal
- Floating windows - windows that float above all applications [experimental]
- Support for Apple Greyscale(ish) Appearance (AGA)
- Embedded sub-dialogs
- Change default font
- **Separate make, interact and delete dialog events**
- **Get and set values & other properties of dialog items while the dialog is open**

Future features

- Tristate buttons - on, off, neither
- Sliders
- Number text fields
- Balloon help
- Keyboard to item mapping
- Styled text edit
- Picture/icon buttons
- *What would you like Dialog Director to do for you?*

How you can help

I want Dialog Director to be compatible with as many different machines, Mac OS versions, and languages as possible. I would be happy to see it become a standard part of everyone's AppleScript setup. Together with your help I can make Dialog Director even better. Send your comments, ideas, questions, problems and example scripts to me at the above address. I hope to include the best of these in future releases of Dialog Director along with more extensive documentation and an FAQ (Frequently Asked Questions document).

Release Notes

This section briefly lists the changes between the different releases of Dialog Director.

Dialog Director v0.6 (Apr '97)

- Fixed grey background problem in *text field* & *password field* items.
- Fixed typing return in *text fields*.
- Improved redraw speed by pruning unnecessary object redrawing.
- Added <command> + '.' to cancel button mapping.
- Removed **colour** synonym for **color** (due to reports of strange behaviour in QuarkExpress).
- Renamed **do dialog** event to **dd auto dialog**.
- Renamed **calc window bounds** event to **dd calc dialog bounds**.
- Added new **dd install** event. Create and initialise the Dialog Director environment.
- Added new **dd make dialog** event. Create and display a dialog window.
- Added new **dd interact with user** event. Interact with the front dialog window.
- Added new **dd get** event. Get a property from a dialog item.
- Added new **dd set** event. Set a property of a dialog item.
- Added new **dd delete** event. Close and delete a dialog window.
- Added new **dd uninstall** event. Clean up and remove Dialog Director environment.
- Added new **dd count dialog** event. Return the number of open dialogs.
- Changed **the items** property of *dialog* to **contents**.
- Changed **label** property of *push button* to **name**.
- Changed **label** property of *check box* to **name**.
- Changed **the items** property of *radio group* to **contents**.
- Changed **label** property of *pop up* to **name**.
- Completely rewrote *pop up* class. No longer uses System pop-up control.
- Added support to *pop up* for optional type-in **text field**.
- Added support to *pop up* for Apple Greyscale(ish) Appearance (AGA).
- Changed **label width** property of *pop up* to **name width**.
- Changed **the items** property of *pop up* to **contents**.
- Added support to *list box* for initial multiple selections.
- Changed **the items** property of *list box* to **contents**.
- Changed **label** property of *text field* to **name**.
- Changed **label bounds** property of *text field* to **name bounds**.
- Changed **label** property of *password field* to **name**.
- Changed **label bounds** property of *password field* to **name bounds**.
- Changed **label** property of *static text* to **contents**.
- Changed **label** property of *group box* to **name**.
- Changed **data** property of *pict* to **contents**.
- Changed **data** property of *icon* to **contents**.
- Added support to *radio group* for dependencies.
- Changed **depends on** property of all dialog items to **enabled**.
- Added support to for initially disabled items.
- Added new *color picker* class.
- Added new *gauge* class.

Note: The 'label' property has been replaced with 'name' (except in the static text class where it has been replaced with 'contents') and the 'data' & 'the items' properties has been replaced with 'contents'. DD v0.6 has extra code to support backward compatibility with DD v0.5.1. All scripts compiled with DD v0.5.1 should continue to function with DD v0.6. However, future versions of DD may not be compatible with 'replaced' properties. It is therefore advisable to update and recompile all your scripts with DD v0.6.

Dialog Director v0.5.1 (6 Dec '96)

- Fixed tabbing direction. (Tab & Shift-Tab were transposed in v0.5 :-).
- Fixed documentation and example inconsistencies. Specifically condense → condensed and centered → center. (All the right characters ... but not necessarily in the right order :-).
- Removed default button outline in floating windows.
- Changed item drawing order to be the same as the item list order.
- Fixed minor sub-window update-on-close problem.
- Added full support for *pict* and *icon* **data** properties.
- Added support for primary, secondary & tertiary style separator lines.
- Added standard notification when in background and trying to opening a window (actually in 0.5 but forgot to mention it).

- Changed grey/colour drawing to take place on > 2 bit deep displays only. (Previously some were > 2, some > 1).
- Added support for typing *carriage returns* in text multi-line fields.
- Added '**last item of the result**' is the final bounds rectangle of the window.
- Added calc window bounds event handler, to calculate the initial window bounds.
- Added support for dialog items with dependencies on specific *list box* items.

- Fixed an annoying TextEdit ‘feature’ if ← or → is pressed when a selection exists.
- Improved the dictionary (I hope :-).
- Added some more examples.
- Included Resource Utilities v1.0b1 osax and examples (no documentation yet).

Dialog Director v0.5 (25 Nov ’96)

This release includes a few fixes, many new features, some changes and a couple of experimental items which are marked and should be used with caution.

- Changed all property and class 4 character codes (in an attempt to avoid future terminology conflicts).
- Added **action** property to *push button* class. An action may be a dialog record or a script handler [experimental]. The value of an **actioned** *push button* is the result of the action.
- Added *font spec* class with **name**, **size**, **style** and **color** properties.
- Added **floating** parameter to do dialog [experimental]. This makes all your DD windows float above all other applications (see **do dialog** below). (I said it was impossible ... I didn’t say I couldn’t do it ;-)
- Added **font** parameter to do dialog.
- Added **greyscale** parameter to do dialog, support for Apple Greyscale Appearance (AGA).
- Added **style**, **name** and **font** properties to *dialog* class.
- Changed *pop up* class to return a *string* when **the items** is a resource type such as ‘FONT’.
- Changed the *nothing’s* that were returned by do dialog to **null’s**. (I previously did not know that a ‘null’ in a script is not actually a **null** object but the type of the **null** object!)
- Replaced **text font**, **text size** & **text style** properties with **font** property in *static text* class.
- Added **justification** property to *static text* class.
- Replaced *frame* class with *group box* class.
- Added **style** property (**primary group**, **secondary group** & **tertiary group**) to *group box* class.
- Added support for enabling and disabling *list boxes*.
- Resolved failure of Event Manager to deliver null events when update events are pending.
- Increased maximum number of *pop up* menus from 10 to 16. This can now be increased even further by the user (see Class pop up below).
- Does not open window until application is in front (blinks Application Menu icon), or if user interaction has been disabled (returns error).
- Addressed all (four) reported problems. There were two terminology conflicts (I suppose that I shouldn’t have assumed that keyAEPropData was a property called **data**) and the above mentioned pop-up menu limit and Event Manager problems.

Dialog Director v0.4 (27 Oct ’96)

This was the first public release of Dialog Director.

Installing Dialog Director

To install Dialog Director all you have to do is copy the “Dialog Director” scripting addition file into the “Scripting Additions” folder of the “Extensions” folder of your System folder. Now you are ready to run the sample scripts and write your own.

Auto Dialogs and Live Dialogs

Dialog Director now supports two different ways of managing dialog windows. These have been named “**Auto Dialogs**” and “**Live Dialogs**” to help distinguish both the scripting implications and the resulting functionality. These two models are fully compatible and may be freely mixed within a script.

Auto Dialogs

Automatic or autonomous dialogs are the same as (**do dialog**) in previous versions of DD where a single call to **dd auto dialog...** creates & displays the dialog, interacts with the user, removes/deletes the dialog and returns the results. Auto Dialogs are the easiest way to manage simple to moderately complex dialogs which require no additional scripting.

Live Dialogs

Live dialogs are new in DD v0.6 and require separate calls

- to create and initialise the Dialog Director environment (**dd install...**),
- to create & display the dialog (**dd make dialog...**),

- to interact with the user (**dd interact with user...**),
- to get properties from dialog items (**dd get...**),
- to set properties of dialog items (**dd set...**),
- to remove/delete the dialog (**dd delete dialog...**),

- and to finally clean up and remove the Dialog Director environment (**dd uninstall**).

Live dialogs allow the management of moderately complex to very advanced dialogs where many user actions may require the intervention of a script. The script may modify the state of the dialog, open an additional dialog or perform a task while the dialog is open (possibly continually displaying the progress of the task).

Event Handlers

This section describes the ten event handlers contained in the Dialog Director suite. It includes details of optional and required parameters, default parameter values, what the event does, general usage, usage notes and examples. More complete (and fully working) examples using all the events are included in the example scripts provided.

Notes and Documentation Conventions

- Typing <command> + <control> + 'Q' will terminate a 'dd auto dialog' call with an error of type 1. (This is handy to know if you manage to create a dialog with no push buttons.)
- All parameters marked with an asterisk* are optional.
- The default value of a parameter is shown in parenthesis following the description.

dd auto dialog: Create, display and interact with a dialog window

dd auto dialog *record* The dialog description record. (see class dialog)
font* *font spec* Global default font for all text items. (see class font spec)
greyscale* *boolean* Use Apple greyscale(ish) appearance. Greyscale is a synonym of greyscale. (false)
floating* *boolean* Experimental. Make windows float above all others. (false)
 Result: *list of anything* The final item values.

Example:

set dItems to [-

```
{class:push button, bounds:[250, 65, 310, 85], name:"OK"}, -
{class:push button, bounds:[170, 65, 230, 85], name:"Cancel"}, -
{class:static text, bounds:[10, 10, 310, 10 + 32], contents:"A very simple dialog box.
```

Just testing!"]]

get dd auto dialog {size:[320, 95], contents:dItems, timeout after:60} **with greyscale**

This event creates an Auto Dialog containing the items described in the dialog record, it displays the dialog, it controls the interaction of the dialog with the user, then (when a *push button* is pressed) it deletes the dialog and returns a list of values, one for each dialog item (and one for the dialog window). This list represents the state of the dialog at termination. Passive items that do not have a value return null. The last item in the list is a rectangle that represents the final bounds of the dialog window. This can be used to set the **bounds** property of the dialog record such that on a subsequent call to **dd auto dialog** the previous window position is restored. See the dialog item classes below to see what values they return.

Setting the **greyscale** parameter to true forces all DD windows to be drawn with grey backgrounds and item frames & group boxes to adopt a 3D appearance according to Apple's greyscale appearance for System 7.5 (AGA) guidelines.

Note: The **floating** parameter is experimental. Setting it to true makes all the DD windows float above all other applications' windows. This is designed for use with the palette and sideways palette window styles. Keyboard input is not supported in floating windows.

Note: It is best not run scripts that create floating windows from the Script Editor. They should be saved as an application first (you may need to increase their memory requirements too), and run from the Finder.

dd calc dialog bounds: Calculate the bounding rectangle of a dialog window

dd calc dialog bounds
point Size of dialog: [width, height].
 Result: *rectangle* The calculated rectangle.

Example:

set dRect to dd calc window bounds [260, 95]

Use this event to precalculate the **bounds** property of a dialog. The rectangle returned will depend on your monitor size and setup as well as the [width, height] parameter. In combination with the rectangle returned by dd

auto dialog you can use this to remember and restore the last position of a movable dialog window. See the “Remember Pos.as” example script.

dd count dialogs: Return the number of open dialogs

dd count dialogs

Result: *integer* The number of dialogs.

This event returns the current number of open DD windows in target application.

dd delete: Close and delete a dialog window

dd delete

reference

The dialog to delete.

Examples:

set dlog to dd make dialog ... -- dlog is now a dialog ref of the form 'dialog id #'

dd delete dialog 1 -- Delete the front most dialog

- OR -

dd delete dlog -- Delete the dialog 'dlog' and any in front of it

- OR -

dd delete dialog id 1 -- Delete the all dialogs (as dialog id 1 is the back most dialog)

This event closes and deletes the specified dialog and any other dialogs in front of it. References to dialogs become invalid once the dialog is deleted. See **dd make dialog** below for details of dialog numbers and IDs. This event may only be called between **dd install** and **dd uninstall**.

dd get: Get a property from a dialog item

dd get

reference

The property to be returned.

Result: *anything* The data from the dialog item.

Examples:

set dlog to dd make dialog ... -- dlog is now a dialog reference

set n to dd get value of item 7 of dlog -- Retrieve the value of dialog item 7 of dlog

set n to dd get value of dialog item 7 of dialog 1 -- Same as above

set dVals to dd get value of every item of dlog -- Retrieve the value of every item of dlog and the dialog's bounds rect

This event allows the reading of properties of dialog items in any currently open dialog. It retrieves a given property from a given item in a given dialog. See **dd make dialog** below for details of dialog numbers and IDs. This event may only be called between **dd install** and **dd uninstall**.

Note: Currently the only valid property to get is **value**.

dd install: Create and initialise the Dialog Director environment

dd install

font* *font spec*

Global default font for all text items. (see class font spec)

greyscale*

boolean

Use Apple greyscale(ish) appearance. Greyscale is a synonym of greyscale. (false)

floating*

boolean

Experimental. Make windows float above all others. (false)

Example:

dd install with greyscale

The **dd install** event creates and sets up the DD environment within the currently targeted application (i.e. the application that is the argument of the current **tell** statement). The **font**, **greyscale** & **floating** arguments apply to all dialogs created between this call and its matching **dd uninstall** call. Calling **dd install** more than once for the same target application and without calling **dd uninstall** between is safe and is ignored.

The **dd install** call is necessary prior to any calls to **dd make dialog**, **dd interact with user**, **dd get**, **dd set**, **dd delete** and **dd uninstall**. This call is not necessary if your script only makes calls to **dd auto dialog**, **dd calc dialog bounds** and **dd count dialogs**. However, if your script is likely to make more than one call to **dd auto dialog** then it is faster (and better practice) to call **dd install** at the beginning of your script (and **dd uninstall** at the end).

Note: The **font**, **greyscale** & **floating** arguments to any **dd auto dialog** calls between calls to **dd install** and **dd uninstall** are ignored i.e. the values used are those passed to **dd install**.

dd interact with user: Interact with the front dialog window

dd interact with user

for max ticks* *integer* The maximum ticks before returning null if the user has clicked nothing.

Result: *anything* The index of the dialog item clicked or null.

Example:

```
set d to dd make dialog ...
repeat
    set i to dd interact with user -- Wait for user input
    if i = 1 then -- Cancel button
        exit repeat
    else if i = 2 then -- Start button
        StartSomething()
    else if i = 3 then -- Stop button
        StopSomething()
    else
        -- Handle other items here
    end if
end repeat
dd delete d -- Remove the dialog
```

This event enables and handles any user interaction with the front most Live Dialog. Interaction consists of window updating of any previously obscured areas of the window and all user mouse-clicks and keyboard input. The event retains control until the user modifies one of the dialog items in the front dialog window. When this happen the dialog item is updated and the index of that item is returned as the result of the event. If the **for max ticks** parameter is specified then if no suitable user interaction takes place within the specified time span (where a tick is 1/60 second) then the event returns **null**. Thus it is important to call **dd interact with user** repeatedly and often, especially if the **for max ticks** parameter is used, so as to not lock out the user. This event may only be called between **dd install** and **dd uninstall**.

dd make dialog: Create and display a dialog window

dd make dialog *record* A record that describes the dialog.
Result: *dialog reference* A reference to the new dialog.

Example:

```
set dA to dd make dialog ... -- dA is dialog id 1 or dialog 1

set dB to dd make dialog ... -- dA is dialog id 1 or dialog 2
                             -- dB is dialog id 2 or dialog 1

set dC to dd make dialog ... -- dA is dialog id 1 or dialog 3
                             -- dB is dialog id 2 or dialog 2
                             -- dC is dialog id 3 or dialog 1
```

This event creates and displays a Live Dialog containing the items described in the dialog record. A reference to the newly created dialog is returned. This reference may be used in subsequent calls to **dd get**, **dd set** and **dd delete**. The reference is of the form 'dialog id #'. Each open dialog has both a unique and constant ID number and a unique but varying index number. The first dialog created (within a particular DD environment) is always dialog id 1, the second (if the first is still open) is dialog id 2, etc.. The front most dialog is always dialog 1 (or dialog id (dd count dialogs)), the second from the front is dialog 2, etc.. The back most dialog is always dialog id 1 (or dialog -1).

The reference returned by **dd make dialog** is valid until the dialog is deleted. This event may only be called between **dd install** and **dd uninstall**.

dd set: Set a property of a dialog item

dd set *reference* the property to be changed.
to *anything* the new property value.

Examples:

```
dd set value of item 3 of dialog 1 to "Some Text"
```

```
set theListItem to a reference to item 5 of dialog id 3 -- A list box item
dd set contents of theListItem to theNewListContents
dd set value of theListItem to theNewListContents's length -- Select last item
```

```
dd set bounds of theMovingButton to [x, y, x + 60, y + 20]
dd set name of theOKButton to "Not OK"
dd set enabled of theOKButton to (true) -- Parenthesis to stop AppleScript changing it
dd set contents of theStaticText to "Not so static, Huh!"
```

This event allows the setting of many different properties of dialog items in any currently open dialog. It is the converse of **dd get**. It sets a given property of a given item in a given dialog. See **dd make dialog** above for details of dialog numbers and IDs. This event may only be called between **dd install** and **dd uninstall**.

dd uninstall: Clean up and remove the Dialog Director environment

dd uninstall

Example:

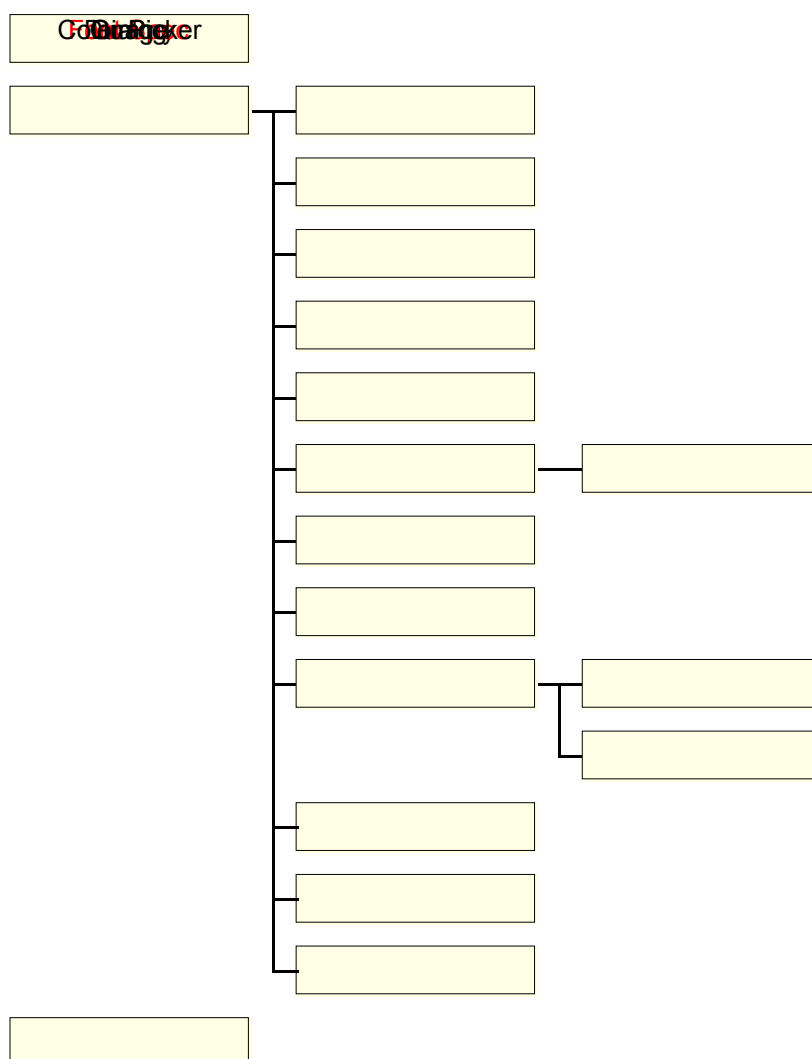
dd uninstall

This event cleans up and removes the DD environment created by an earlier call to **dd install**. The clean up process includes closing and deleting any open/active dialogs. This event may only be called after **dd install**. Further calls to **dd uninstall** result in an error being returned.

Tip: If, during script development, you encounter a script error and the script terminates leaving any ‘dead’ Dialog Director windows (in the Script Editor), you may safely remove these windows by creating and running a script consisting of just `dd uninstall`.

Object Classes

This section describes the many object classes contained within the Dialog Director suite. A complete list of dialog item types supported by this release can be seen in the diagram below. The classes in *italics* are not instantiable, and only exist for implementation purposes.



The Dialog Director class hierarchy

Notes and Documentation Conventions

- All properties marked with an asterisk* are optional.
- The default value of a property is shown in parenthesis following the description.
- All of the 'help' properties are currently ignored.
- All text/string properties are currently limited to a maximum of 255 bytes (characters?) in length.
- ~~The 'label' property was previously called 'name' but I cannot decide whether it should be called 'name', 'label' or 'title'.~~
- Properties marked with [W] are writeable while a dialog is open (via **dd set**).
- Properties marked with [R/W] are readable and writeable while a dialog is open (via **dd get** and **dd set**).

Class dialog: data used to create a dialog

Elements:

dialog item by numeric index

Properties:

default*	<i>integer</i>	Index of the default (outlined) push button. (first push button item)
size*	<i>point</i>	Size of window: [width, height]. Specifying a window size forces the dialog box to be centred on the main screen.
bounds*	<i>rectangle</i>	Bounds of window. (rect of main screen inset by 16 pixels)
help*	<i>string</i>	Help string
timeout after*	<i>integer</i>	Seconds after which the window closes automatically. (no timeout)
style*	<i>window frame style</i>	The style of the window frame. One of:
document window		Standard movable mode less window
modal		Standard modal dialog window (default)
plain dialog		Borderless modal dialog window
movable modal		Standard movable modal dialog window
palette		Standard floating window (requires Mac OS 7.5?)
sideways palette		Floating window with title bar on left (requires Mac OS 7.5?)
name*	<i>string</i>	The title of window. This is displayed in some of the styles of window frame. ("")
font*	<i>font spec</i>	Default font for text items in this window. (see class font spec)
contents	<i>a list of dialog item</i>	The dialog items contained in this window.

Example:

```
dd auto dialog {default:3, size:[260, 95], contents:ditems, timeout after:60}
```

The **size** and **bounds** properties are mutually exclusive. Note that the style of the window has no bearing on its modal/mode less nature. All dialog director windows are modal within their application context, but movable windows allow the user to switch applications. Movable windows are primarily designed for use in applets and droplets. If they are used in embedded or attached scripts then the contents of the application's other windows may not be updated.

Class dialog item: a dialog item

Plural form:

dialog items

Properties:

class	<i>class</i>	This property must be set to the class of the required item.
bounds	<i>rectangle</i>	The bounding rectangle of the item: [left, top, right, bottom]
help*	<i>string</i>	The help text. ("") [currently unused]
enabled*	<i>boolean, integer or list</i>	True, false, or the other items on which this item depends. (true)

All sub-classes of dialog item inherit the properties of this class (with the exception of *dummy item* which does not have a **bounds** property). The **help** and **enabled** properties have no meaning for the passive item classes *static text*, *frame*, *pict*, *icon* and *gauge*. You should never have to create an actual instance of the *dialog item* class. The **enabled** property (from v0.5.1) has been replaced with the **enabled** property. The **enabled** property provides the same functionality as **enabled** plus *dialog items* may be created in a disabled state. This is most useful in live dialogs.

Class push button: a push button item

Properties:

class	<i>class</i>	This property must be set to <i>push button</i> .
bounds	<i>rectangle</i>	The bounding rectangle of this button: [left, top, right, bottom] [W]
help*	<i>string</i>	The help text for this button. ("") [currently unused]
enabled*	<i>boolean, integer or list</i>	True, false, or the other items on which this button depends. (true) [W]
name	<i>string</i>	The title/name of this button. [W]
action*	<i>anything</i>	Button's action when pressed. (no action)

Return value:

boolean If **action** is null: true if this button was used to dismiss the dialog, otherwise false.
anything If **action** is not null: the result of the action if the button was pressed, otherwise null.

Example:

```
set pb1 to {class:push button, bounds:[430, 390, 490, 410], name:"OK", help:"Dismiss"}
```

~~Currently all push button items dismiss the dialog.~~ If no default button is specified (via the **default** property) in the dialog record then the first push button item becomes the default button. During dialog execution the **return** and **enter** keys map to the default button. The first push button item that is not the default button becomes the cancel button. The **escape** key and <command> + '.' maps to the cancel button.

The **action** property defaults to **null**. This indicates that the *push button* dismisses the dialog. An action may also be:

- a dialog record - in which case a new dialog is opened modally on top of the current one. When dismissed this sub-dialog sets the **value** property of the button as if **dd auto dialog** had been call on the sub-dialog itself.
- a handler script/function - [experimental] the handler (which must have no arguments) is executed and the result becomes the **value** of the button. There are limitations in what may be done in an action script. However, these limitations are still the subject of experimentation. For example: it is possible to call **dd auto dialog** in an action script, and this does work with **some** simple dialogs. ~~However, this feature is not supported.~~ Below is a simple example of an action script.

```
on GetFile()  
    choose file with prompt "Select a file:"  
end GetFile
```

```
set pb3 to {class:push button, bounds:[100, 50, 160, 70], name:"Open...", action:GetFile}
```

Any errors that occur in an action are propagated through and returned by the **dd auto dialog** command. An action will be executed each time its button is presses. For any particular button only the last action's result is returned (or null if the action was never executed).

Tip: To set a *push button* to do a one shot action make it inversely dependent upon itself, and ensure that its action does not return **null**. E.g. if the button's item index is 5 then set its **enabled** property to -5, and its **action** property to a dialog record or function:

```
on Bang()  
    return 1  
end Bang
```

```
{class:push button, bounds:[0, 0, 60, 20], name:"One Shot", action:Bang, enabled:-5}
```

Note: The **value** property of a *push button* may not be set.

Class check box: a check box item

Properties:

class	<i>class</i>	This property must be set to <i>check box</i> .
bounds	<i>rectangle</i>	[W]
help*	<i>string</i>	
enabled*	<i>boolean, integer or list</i>	[W]
name	<i>string</i>	[W]
value*	<i>boolean</i>	If true then the check box is checked. (false) [R/W]

Return value:

boolean True if this check box is checked, otherwise false.

Class radio group: a group of radio buttons

Properties:

class	<i>class</i>	This property must be set to <i>radio group</i> .
bounds	<i>rectangle</i>	The bounding rectangle of the first radio button.
help*	<i>string</i>	
enabled*	<i>boolean, integer or list</i>	[W]
value*	<i>integer</i>	The index of the selected button. (1) [W]
button offset	<i>point</i>	The relative offset of each button: [dx, dy]. If no max down property is specified then the bounding rectangle of each button is offset by (dx, dy) from the previous button's bounds. if a max down property is specified then the bounding rectangle of

each button is offset by (0, dy) from the previous button's bounds for the first **maxDown** buttons then the next button is offset by (dx, 0) from the first button's bounding rectangle. This creates vertical columns of buttons.

max down* *integer* maximum number of buttons in vertical direction before wrapping to the next column. (no wrapping)
contents *a list of strings* The button name list.

Return value:

integer Index of the currently selected button.

Example:

```
set theItems to ["One", "Two", "Three", "Four", 5, 6, 7, 8, 9, 10, 11, 12]
-- Create 2.4 columns of 5 items each
set rg1 to {class:radio group, bounds:[20, 40, 120, 56], button offset:[105, 20], max down:5, contents:theItems}
-- Create 1 diagonal column of 12 items
set rg2 to {class:radio group, bounds:[20, 140, 120, 156], button offset:[10, 20], contents:theItems}
```

Class pop up: a pop-up menu item

Properties:

class *class* This property must be set to *pop up*.
bounds *rectangle*
enabled* *boolean, integer or list* [W]
name *string* Label/title of pop-up item.
name width* *integer* Amount of space (in pixels) to reserve on the left of the bounding rectangle for its name. (width of name + 8)
value* *integer or string* Default selection. (1) [W]
text field* *integer* The index of a type-in text field. (no type-in)
contents *list of strings* The menu item list. This consists of either a single string or a list of strings. If the single string form is used with a string of exactly 4 characters then the AppendResMenu toolbox function is called to create a menu of the names of the resources of that type. Common uses are "FONT" (to create a menu of all fonts) and "DRVR" (to create a menu of the items in the "Apple Menu Items" folder). If the string is not 4 characters long then it may be longer than the 255 character limit imposed on all other strings. Each string may define one or more menu items. You may embed metacharacters in the string(s) to define various characteristics of a menu item.

Here are the metacharacters that you may specify in the data parameter:

<u>Metacharacter</u>	<u>Description</u>
;	Separates menu items.
^	When followed by an icon number, defines the icon for the item. If the keyboard equivalent field contains (ASCII character 28), this number is interpreted as a script code. [NOT functional]
!	When followed by a character, defines the mark for the item.
<	When followed by one or more of the characters B, I, U, O, and S, defines the character style of the item to Bold, Italic, Underline, Outline, or Shadow, respectively.
/	When followed by a character, defines the keyboard equivalent for the item. To specify that the menu item has a script code, small icon, or reduced icon, use the SetItemCmd procedure to set the keyboard equivalent field to (ASCII character 28), (ASCII character 29), or (ASCII character 30), respectively. [NOT functional]
(Defines the menu item as disabled. Use "(-" for a grey dividing line.

Return value:

integer Index of the currently selected menu item if **contents** is **not** a 4 char resource type.
string Text of the currently selected menu item if **contents** is a 4 char resource type.

Examples:

```
-- For mu1 items through 12 of theItems are automatically coerced to strings
set theItems to ["One", "Two", "Three", "Four", 5, 6, 7, 8, 9, 10, 11, 12]
set mu1 to {class:pop up, name:"Nums:", bounds:[300, 250, 480, 270, value:10, contents:theItems}
-- In mu2 item 5 is a grey dividing line and items 7, 8 & 9 are in the named styles
set mu2 to {class:pop up, name:"Test:", bounds:[10, 40, 130, 59], name width:30, contents:~
"1;2;3;4;(-;6;<BBold;<IItalic;<UUnderline;10"}

-- This complete example presents a dialog with a type-in style pop-up
set dTypeInPopUp to {size:[120, 80], contents:[~
{class:push button, bounds:[50, 50, 110, 70], name:"OK"}, ~
{class:pop up, bounds:[95, 10, 130, 29], contents:"9;10;12;14;18;24;36;48", text field:3}, ~
{class:text field, bounds:[50, 11, 90, 27], name:"Size:", name bounds:[10, 11, 50, 27], value:9}]}
dd auto dialog dTypeInPopUp with greyscale
```

Currently each pop-up menu requires a dummy menu resource (type 'MENU'). There are 16 menu resources included in the "Dialog Director" scripting addition file numbered sequentially from 32720. This limits the maximum number of menus in all open DD windows to 16 per application. If you need more than 16 menus at a time you can add extra menu resources to either your script document (preferred) or the "Dialog Director" scripting addition file using ResEdit. These extra menus should have IDs numbered sequentially from 32736. Pop Up dialog items provide pop-up menu style controls. DD v0.6 completely reimplements pop-up controls and introduces new style pop-ups when the **greyscale** dialog option is set to true. If a resource-names style menu is created (such as with contents:"FONT") then the value may be initialised to a string e.g. value:"Helvetica". In this case calls of the form **dd get value of refToThisPopUp** return a string (the name of the current menu selection) and calls of the form **dd set value of refToThisPopUp** to **newValue** require a string (the name of a menu item) for the **to** (newValue) parameter.

Class list box: a scrolling, single column text list

Properties:

class	<i>class</i>	This property must be set to <i>list box</i> .
bounds	<i>rectangle</i>	
enabled*	<i>boolean, integer or list</i>	[W]
value*	<i>integer or list</i>	The selected item or items. (0 if no selection) [R/W]
flags*	<i>integer</i>	Selection flags for this list. (130) This property controls the selection algorithm. The constants below may be added together to define the required behaviour for a particular list.
Constant	Description	
2	disable highlighting of empty cells	
4	allow use of shift key to deselect items	
8	shift-drag selects items passed by cursor	
16	reset list before responding to shift-click	
32	prevent discontinuous selections	
64	enable multiple item selection without shift key	
128	allow only one item to be selected at once	
The values of the flags property for standard list behaviours are 130 for single item selection, and 0 for arbitrary multiple item selection.		
contents	<i>list of strings</i>	The list items. Each string represents a single list item. Items that can be coerced to strings are also valid e.g. numbers and data types. [W]

Return value:

if only single selections are allowed:

integer Index of the currently selected item or 0 if no selection.

if multiple selections are allowed:

list of integers List of indices of the currently selected items or 0 if no selection.

List boxes are manipulable via the keyboard (i.e. are keyboard focusable):

home	select first item
end	select last item
up arrow	move selection to previous item
down arrow	move selection to next item
page up	move selection up by list box height
page down	move selection down by list box height
tab	move keyboard focus to next focusable item
shift-tab	move keyboard focus to previous focusable item
backspace	delete last character in internal buffer (see <i>other</i>)
clear	clear internal buffer (see <i>other</i>)
<i>other</i>	characters typed within 1 second intervals are appended to an internal buffer which is used to select the first matching item. This is not case sensitive and does not require the list to be in alphabetical order.

Example:

```
-- List the items in the System folder and force the user to select one.
set lbItems to list folder (path to system folder)
set sysFolderDlg to {size:[220, 220], contents:[-
    {class:list box, bounds:[10, 10, 210, 172], contents:lbItems}, -
    {class:push button, bounds:[150, 190, 210, 210], name:"OK", enabled:1}}
dd auto dialog sysFolderDlg
```

The height of a list box should be a multiple of the row height + 2 (for the frame). For the standard Chicago font the row height is 16. Therefore **height of list box** = (**rows displayed**) * 16 + 2. A list box may be

dependent on other items. The text of a disabled list box is draw in grey and the scrollbar is hidden.
Note: The **value** property may now be initialised to a list of cell indices to highlight.

Class text field: an optionally labelled editable text item

Properties:

class	<i>class</i>	This property must be set to <i>text field</i> .
bounds	<i>rectangle</i>	
enabled*	<i>boolean, integer or list</i>	[W]
name*	<i>string</i>	Label/prompt text of the item. (no name)
name bounds*	<i>rectangle</i>	Bounding box of the name text. This must be specified if a name is specified. (no name bounds)
value*	<i>string</i>	Initial editable text of the item. ("") [R/W]

Return value:

string The current editable text of the item.

Example:

```
set ed1 to {class:text field, bounds:[60, 160, 200, 176], name bounds:[10, 160, 60, 180], name:"Text:", value:"default text"}
```

Text fields and password fields fully support the extended keyboards including:

f2	cut
f3	copy
f4	paste
del	delete forward
home	move insertion point to start of text
end	move insertion point to end of text
page up	move insertion point up by height of field
page down	move insertion point down by height of field
clear	delete selection
tab	move keyboard focus to next focusable item
shift-tab	move keyboard focus to previous focusable item

Class password field: an opaque editable text item

Properties:

class	<i>class</i>	This property must be set to <i>password field</i> .
bounds	<i>rectangle</i>	
enabled*	<i>boolean, integer or list</i>	[W]
name*	<i>string</i>	
name bounds*	<i>rectangle</i>	
value*	<i>string</i>	As for an exit text item, but always displayed as bullets. [R/W]

Return value:

string The current editable text of the item (the actual text, not the bullets).

Example:

```
set pwd to {class:password field, bounds:[10, 36, 250, 36 + 16], name bounds:[10, 10, 250, 26], name:"Enter Password:"}
```

Class static text: a static text item

Properties:

class	<i>class</i>	This property must be set to <i>static text</i> .
bounds	<i>rectangle</i>	
enabled*	<i>boolean, integer or list</i>	If disabled the text appears greyish. [W]
contents	<i>string</i>	The displayed text of the item. [W]
font*	<i>font spec</i>	The font specification. (see class font spec)
justification*	<i>text alignment</i>	How to align/justify the text. One of: flush default Aligned according to the primary line direction (default) left Left aligned for all scripts center Centred for all scripts right Right aligned for all scripts

Return value:

null This class of item returns nothing.

Example:

```
-- Display "Some Text" using 18pt Times in bold, italic style  
{class:static text, contents:"Some Text", bounds:[10, 10, 220, 30], font:{size:18, name:"Times", style:3}}  
-- Display aString using the system font and size in bold, condensed style
```

set aString to "A sample string!"

set st1 to {class:static text, contents:**aString**, bounds:[410, 10, 520, 42], font:{style:[bold, condensed]}}

Class group box: an optionally labelled rectangular frame item

Properties:

class	<i>class</i>	This property must be set to <i>group box</i> .
bounds	<i>rectangle</i>	
name*	<i>string</i>	Label/title text of the frame. (no name)
style*	<i>frame style</i>	The style in which the group box is to be drawn. (primary group)
frame style	AGA	Normal
primary group	etched line	black frame
secondary group	indented box	grey frame
tertiary group	raised box	light grey frame

Return value:

null This class of item returns nothing.

Example:

```
set box1 to {class:group box, name:" Box ", bounds:[300, 16, 490, 112], style:secondary group}
```

If the width of the bounding rectangle ≤ 4 then the item is drawn as a vertical separator. If the height of the bounding rectangle ≤ 4 then the item is drawn as a horizontal separator. In any case the specified style is used to draw the rectangular frame, vertical separator or horizontal separator.

Class pict: a picture item

Properties:

class	<i>class</i>	This property must be set to <i>pict</i> .
bounds	<i>rectangle</i>	The picture is scaled to fit this rectangle.
contents	<i>picture or integer</i>	A Quickdraw picture or ID number of a picture ('PICT') resource.

Return value:

null This class of item returns nothing.

Example:

```
set pic1 to {class:pict, bounds:[10, 90, 10 + 198, 90 + 47], contents:131}
```

The **res id** and **data** properties have been replaced with the **contents** property.

Class icon: an icon item

Properties:

class	<i>class</i>	This property must be set to <i>icon</i> .
bounds	<i>rectangle</i>	The icon is scaled to fit this rectangle.
contents	<i>icon or integer</i>	A colour or black & white icon or ID number of the icon ('icn' or 'ICON') resource.

Return value:

null This class of item returns nothing.

Example:

```
set icn1 to {class:icon, bounds:[160, 10, 192, 42], contents:128}
```

The **res id** and **data** properties have been replaced with the **contents** property.

Class dummy: a dummy dialog item

Properties:

class	<i>class</i>	This property must be set to <i>dummy</i> .
enabled*	<i>boolean, integer or list</i>	[W]

Return value:

boolean The current state of the item, as determined by its **enabled** property.

Example:

-- If items 1, 2 & 3 are text or password fields then this item's state would be on
-- only when all 3 fields contained some text (see Dependencies section below)

```
set dum1 to {class:dummy, enabled:[dAnd, 1, 2, 3]}
```

The only purpose of dummy dialog items is to enable the combination of dependencies into even more complex

structures (or possibly to return the combined state of several items). Any item that depends on a dummy item must appear after that dummy item in the dialog item list.

Class color picker: a colour picking item

Properties:

class	<i>class</i>	This property must be set to <i>color picker</i> .
bounds	<i>rectangle</i>	
enabled*	<i>boolean, integer or list</i>	[W]
value	<i>RGB colour</i>	The colour of the picker's swatch. [R/W]
flags*	<i>integer</i>	The color picker flags. (0) [Ignored]

Return value:

RGB colour The current colour of the item's swatch.

Example:

```
dd auto dialog {size:[260, 100], contents:[-
  {class:push button, bounds:[190, 70, 250, 90], name:"OK"}, -
  {class:color picker, bounds:[10, 10, 90, 90], value:[11100, 22200, 33300]} -
]}
```

A *color picker* is an interactive dialog item that displays a framed, rectangular swatch of colour that when clicked allows the user to choose a colour using the standard color picker dialog.

The **value** property is in [red, green, blue] format, where 0 is no colour and 65535 is maximum colour.

Class gauge: a graphical magnitude indicator

Properties:

class	<i>class</i>	This property must be set to <i>gauge</i> .
bounds	<i>rectangle</i>	
enabled*	<i>boolean, integer or list</i>	[W]
value*	<i>integer</i>	The size of the gauge indicator bar. (0) [R/W]
max value*	<i>integer</i>	The maximum value of the gauge. (100)

Return value:

integer The current value of the gauge.

Example:

```
property theMax : 1234
dd install with greyscale
set p to dd make dialog {size:[300, 50], contents:[-
  {class:static text, contents:"Thinking...", bounds:[8, 4, 160, 20]}, -
  {class:gauge, bounds:[10, 25, 290, 25 + 12], value:0, max value:theMax} -
]}
repeat with n from 1 to theMax
  dd set value of item 2 of p to n
end repeat
dd delete p
dd uninstall
```

A *gauge* is a passive dialog item that displays an integer value as a horizontal indicator bar that fills a relative proportion of the left side of an enclosing frame. Alternatively it may display a “barber pole” that fills the entire frame.

When **max value** is a positive integer then setting **value** to an integer between 0 and **max value** displays a bar that fills the left **value/max value** of the part of the gauge. Setting **value** to ≤ 0 displays an empty gauge, and setting **value** to $\geq \text{max value}$ displays a full gauge. Setting **max value** to a negative integer causes a barber pole to be drawn. Repeatedly incrementing **value** causes the barber pole to “rotate” forward, Repeatedly decrementing **value** causes the barber pole to “rotate” backward.

Class font spec: a text font specification

Properties:

name*	<i>string</i>	Name of font the family for this font spec. ("Chicago" [actually the system font]).
size*	<i>integer</i>	The point size of the text. (12 [actually 0 which means use the system font size])
style*	<i>style</i>	An integer, style name or list of style names indicating the style of the text. (plain)
color*	<i>RGB colour</i>	The colour of the text in [red, green, blue] format, where 0 is no colour and 65535 is maximum colour. ([0, 0, 0])

Example:

```
-- This is the default system font spec on English systems
set font1 to {name:"Chicago", size:0, style:plain, color:[0, 0, 0]}
```

-- This font spec produces **this style of text**
set font2 to {name:"Geneva", size:9, style:[bold, underline]}

A font spec can be used to override the default settings for the context (and all sub-contexts) within which it is specified. That is the global default font (the **font** parameter of **dd auto dialog**) overrides the system font spec for all DD dialogs, the dialog default font (the **font** property of a **dialog**) overrides the global font spec for all text in that **dialog**, and the **font** property of a **static text** overrides the dialog default font for that **static text** item. Where a partial **font spec** is used the missing parts are automatically filled in from the next larger context. **Note:** Currently the **color** property is only used correctly by **static text** items, and is best left to default to black elsewhere.

Dependencies

This section explains how dependencies (i.e. the **enabled** property) works. It does not need to be understood completely if you only need to create simple dialog boxes.

Item dependencies allow interactive dialog items to be enabled or disabled, dynamically, dependent on the state of one or more other items. e.g. The OK button could be greyed out/disabled until some text is entered into a text field (as in the password dialog example below). The dependencies of all items are reevaluated whenever the state of any item changes.

There are three types of dependency: none, single item and multiple items. By default a dialog item has no dependencies i.e. it has no **enabled** property and it is always active. A single item (simple) dependency consists of the **enabled** property being a single integer which gives the index of the item upon which this item depends. (An item's index is determined by the position of the item in the **contents** property of the *dialog* record passed to **dd auto dialog**.)

Negative values indicate inverse dependencies. i.e. if item 4 is a check box then an item with property depends on:4 would be enabled only when the check box is checked, and an item with property depends on:-4 would be enabled only when the check box is **not** checked.

Values with a magnitude greater than 255 are indicate a dependency on a sub-item (such as a radio button or menu item). e.g. a value $n = 522$ is interpreted as item no. $= |n| \bmod 256 = 10$, sub-item no. $= |n| \div 256 = 2$. The following function will calculate the numbers for you:

```
on SubItem(itemNo, subItem)
    return itemNo + subItem * 256
end SubItem
```

A multiple item (complex) dependency consists of the **enabled** property being a list composed of an enumeration name (dOr or dAnd) followed by one or more integers as described for the simple dependency e.g. enabled:[dAnd, 3, -5] enables this item only when item 3 is on and 5 is off.

How an item's on/off state is determined depends on the item's class, as follows:

Class	On when	Off when	Comments
push button	never	always	actionless buttons only
push button	after being clicked	before being clicked	sub-dialog buttons only
push button	action returned \neq null	otherwise	handler script buttons only
check box	checked	not checked	
radio group	never	always	
radio group button	selected	not selected	
pop up	never	always	
pop up item	selected	not selected	
list box	has a selection	has no selection	
list box item	selected	not selected	
text field	contains text	empty	
password field	contains text	empty	
static text	never	always	
frame	never	always	
pict	never	always	
icon	never	always	
dummy	dependencies evaluate to true	dependencies evaluate to false	
color picker	never	always	
gauge	never	always	

Examples

1. Very simple dialog

```

set dlog to {size:[320, 95], contents:[-
  {class:push button, bounds:[250, 65, 310, 85], name:"OK"}, -
  {class:push button, bounds:[170, 65, 230, 85], name:"Cancel"}, -
  {class:static text, bounds:[10, 10, 310, 10 + 32], contents:"A very simple dialog box."}] -
}

```

set dVals to dd auto dialog dlog

This creates a small dialog window in the centre of the main screen containing some static text above OK and Cancel push buttons.



The OK button is item 1, the Cancel button is item 2, and the static text is item 3 because that is the order they are defined in **dlog**'s contents list. Clicking the OK button yields the result:

```
{true, false, null, {416, 397, 736, 492}}
```

Clicking the Cancel button yields the result:

```
{false, true, null, {416, 397, 736, 492}}
```

As you can see, dd auto dialog returns a list of four items, one for each dialog item passed to it via the contents property and one that represents the final bounding rectangle of the dialog window. In this example the third item will always be null because static text items contain no value. The values of items 1 and 2 allow you to determine which button was used to dismiss the dialog. The dismissing push button always returns true (whether invoked via the mouse or keyboard), and all other push button's return false. This allows you to write the following sort of code:

```

if item 1 of dVals then
  -- OK was pressed
else if item 2 of dVals then
  -- Cancel was pressed
end if

```

Note that in this example the OK button (being the first push button) defaults to the default button and the Cancel button defaults to the escape button. These could be reversed by including default:2 in the dialog record passed to dd auto dialog.

2. Simple password dialog

```

property dPassword : {size:[260, 95], contents:[-
  {class:push button, bounds:[190, 65, 250, 85], name:"OK", enabled:3}, -
  {class:push button, bounds:[110, 65, 170, 85], name:"Cancel"}, -
  {class:password field, bounds:[10, 36, 250, 36 + 16], name bounds:-
    [10, 10, 250, 26], name:"Enter Password:", value:""]} -
  , timeout after:60}

```

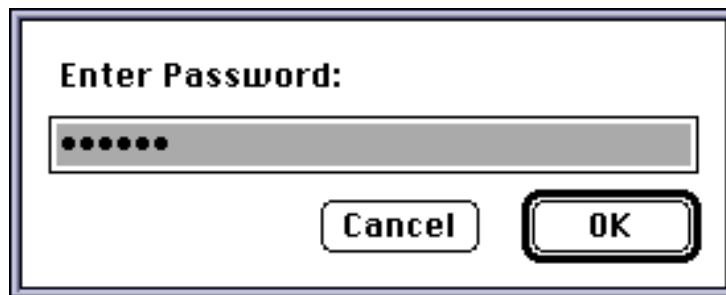
get dd auto dialog dPassword

This creates a small dialog window in the centre of the main screen containing a password field below a prompt and below that are two push buttons. The OK button has a default border and is greyed out/disabled until some text is entered into the password field. If no push button is pressed within 60 seconds the dialog is automatically dismissed as specified by the timeout after:60 property. Note that this example has the dialog record (**dPassword**) defined as a property as opposed to the variable used in example 1. This helps speed up

execution but only where the dialog record is constant. The dialog window should initially look like this:



After typing some text and selecting it the dialog window might appear as below. Note that any character typed will appear as a bullet, and that the OK button is now enabled.



Changing the above script to set the **greyscale** parameter to true and the initial **value** of the password field item to "Test":

```
...  
{class:password field, bounds:[10, 36, 250, 36 + 16], name bounds:-  
  [10, 10, 250, 26], name:"Enter Password:", value:"Test"}} -  
...
```

get dd auto dialog dPassword with greyscale

should produce a dialog window that initially looks like this:



Known Problems and Limitations

This section lists any known problems and limitations with this release of Dialog Director (in no particular order).

- There is a maximum limit of 255 dialog items per dialog window.
- A maximum of 7 dialogs may be open at a time (per application).
- Strings used by DD are limited to 255 bytes (characters) but the *text field* and *password field* classes do not enforce this.
- Many dialog item properties (too many to list) are not available via **dd get** and **dd set**. More will be in latter versions.
- If a live dialog window is left open after the script returns control to the calling application (such as if the script generates an error) then the application may crash. This only effects some applications and is completely avoidable by writing the script thusly:

```
dd install -- plus any options  
try
```

```

        <code to manage my Dialog Director windows>
        dd uninstall
    on error
        dd uninstall
    end try

```

- If **dd interact with user...** is not called frequently enough while a live dialog is active, mouse clicks and key presses may be lost.
- When in use DD consumes at least 80K of memory from the application heap in whose layer the dialog(s) appear. This grows with the complexity and number of dialogs. All this memory should be returned to the application upon DD termination.
- Due to the manner in which DD v0.6 handles backward compatibility with DD v0.5.1 properties that have been replaced – required properties that are missing produce an error specifying the old property ID rather than the new property name.

Other Scripting Additions (Not currently part of this package)

Ask me about the other scripting additions I have written:

Resource Utilities	Read, write, get & set info for any resources and read write any AppleScript data structures as resources (included with DD v0.6).
Picture Utilities	Read, write, print, join, resize and get info about Quickdraw pictures.
Display Picture	Display a Quickdraw picture in a modal, scrollable window with magnification.
Make Picture	Create a picture from within AppleScript. Supports rectangles, ovals, lines, text, polygons, regions, patterns, pens, colours etc.
String Width	Calculate the width, in pixels, of a text string of given font, size and style.