Instructions for Setup and Use of BubbleHelp version 3.0

by

Resource Development Associates

Installation

To install RDA's BubbleHelp you must have an alias named BubbleHelp. You can alias to an existing directory or create a separate one for BubbleHelp. This alias is not required to distribute BubbleHelp with your applications but the BubbleHelp forms look for the helpmsg.db table and the RDABubbl.ldl library in this Alias.

Once you have decided where to place the files and created the alias you need to place the RDAbubbl.ldl library and the helpmsg.db, helpmsg.px and helpmsg.mb files there. We recommend that you also put the bublhelp.fdl and bublpart.fsl forms there. Open the bublhelp.fdl and/or the bublpart.fsl and begin applying BubbleHelp to your applications.

Using RDA's BubbleHelp

The most direct way to use RDA's BubbleHelp function requires you to create the help messages in a table and apply the BubbleHelp components. You can use the current helpmsg.db or create a copy of the table for your own messages. The table does not need to have the same name. You may wish to create different ones for different applications. The table must have the same structure although you can extend it if you wish. Extensions to the helpmsg.db table might include fields for descriptions of the object the message is attached to or other system documentation info. A basic data entry component for help messages, along with a message testing component is included on the Bublpart.fsl. You can open this in edit using change table to open the form for entry on a message table other than helpmsg.db The basic structure is:

	Field Name	Type	Size	Key
1	msgid	S		*
2	MessageText	M	20	

If you extend the structure fields 1 and 2 must remain the same. The methods from the BubbleHelp library look for the message id in the first field and the text in the memo from the second field. These items are specified in the variables in the library methods.

You can essentially put in any amount of text in a BubbleHelp message. The sizing of text display objects on forms is not an exact process because of logical screen sizes and proportional fonts. We tried various fixed width fonts but were dissatisfied with the look. The sizing complexity is compounded because, unlike tool tips in Paradox 9, or the help messages you can attach to toolbars, BubbleHelp wraps text after 3.1 logical inches (or 2 logical inches for scrolling messages). While this means that your BubbleHelp messages are only limited by the display size of your forms it means you may sometimes have to manipulate the text to get a display look you are happy with. Unfortunately a simple sizetofit() won't work in this context. Therefor, you may sometimes wish to edit your text a bit to get a display with no blank space at the bottom. Generally it is best to shorten the text a bit if you are not able to see the full text. You can also lengthen the text or pad with spaces to get a cleaner display if you can not shorten the text. But most amounts of text will display cleanly. We have kept the display on the short side because of this. It is not recommend to put tab or return characters in help messages although it is supported. It can also be helpful to users if you put keyboard shortcuts in () at the end of your help bubble messages. There are tips on creating the best display at the end of this document.

The heart of RDA's BubbleHelp are the functions which size and display your help messages. These are called from the library from the items in your application which you want help displayed on. There are 4 primary methods for helpdisplay:

disphelp(msgid smallint, tblname string, formname string)
disphelpleft(msgid smallint, tblname string, formname string)
disphelptop(msgid smallint, tblname string, formname string)
disphelptopleft(msgid smallint, tblname string, formname string)

These display help messages to the bottom right, bottom left, top right and top left respectively. These are generally called from the Mouseenter method for buttons and other general UI Objects and from the Arrive method for fields. But you can call them from any object that can be referenced by the variable **self** and which has display room available for the message. Paradox can not display the messages off a form's boundaries so for all practical purposes this means you can not reference BubbleHelp messages from a page or form object nor any other object which is sized to display so large on a form that there would be no message space. This can be easily worked around by placing other UI objects on the form which can call the BubbleHelp mesages (e.g. text, graphics, etc.)

To embed BubbleHelp in your application you need call and clear routines in the objects that will have BubbleHelp and must include statements in your forms' Arrive, Uses and Var methods. These are detailed below. They can be copied from the page tabs of the bublhelp.fdl or from the methods in the bublpart.fsl. When copying the code be careful not to duplicate the method and endmethod lines when you paste them into your code so as not to cause code errors. You also need to place the hidden, unlabeled field, helptext, on the form at a level which can be seen by all calling objects. Usually on the page is best. It can be sized as small as you like for design purposes since it resizes on the fly at runtime. You should not change the properties of the helptext object since it is designed to work with the sizing code in the BubbleHelp library.

Because BubbleHelp's helptext object must respect the containership in which it is placed, using the standard BubbleHelp methods on multipage forms was quite complicated in the 1.0 version of BubbleHelp. The 2.0 version added 4 new methods. These methods can be used for multipage forms or in any other case where the containership of the helptext object makes calling it from another object difficult. Generally, however, these methods should only be used on second and subsequent pages in a multipage form. The four methods added in version 2.0 are:

```
disphelpwobject(msgid smallint, tblname string, formname string, helptextname string)
disphelpleftwobject(msgid smallint, tblname string, formname string, helptextname string)
disphelptopwobject(msgid smallint, tblname string, formname string, helptextname string)
disphelptopleftwobject(msgid smallint, tblname string, formname string, helptextname string)
```

These methods function just like their counterparts above with 2 minor exceptions. First, you need to place the hidden, unlabeled field, helptext, on the subsequent pages of forms at a level which can be seen by all calling objects (usually the page) and change it's name. Right click and select properties to change it's name. Do not change any of it's other properties. It is recommended that you just add a number corresponding to the page number on which you place this object to make calling this object simpler. Second, when you use one of these four methods you must include the unlabeled field's name as the string for helptextname. If you have named the page object in the form or place the object in any other named containership you will need to use the whole containership (e.g. "Page2.helptext2") as the helptextname string.

In version 3.0 eight additional methods have been added. The first four methods are:

```
disphelpdirect(stMsg Anytype, formname string)
disphelpleftdirect(stMsg Anytype, formname string)
disphelptopdirect(stMsg Anytype, formname string)
disphelptopleftdirect(stMsg Anytype, formname string)
```

These methods allow you to pass an assigned variable that can be cast as a string (a memo, a string variable, the status message from Paradox's status bar, etc.) These will be passed to the helptext object and sized just as regular BubbleHelp messages are in the primary 4 methods. The primary use for these methods is to capture messages from the status bar or Status method and pass them to a message bubble. This allows you to display status messages such as key violation messages, "record already locked in this session" etc. in a more prominent fashion than would otherwise occur without requiring the use of msgstop(), msginfo() etc. This increases the likelihood that users will see these messages without actually halting processing and requiring user intervention to continue. Of course you can use these methods for displaying anything you can pass as a variable which can be cast as strings so be as creative as you like with these methods. Note, however, that although the sizing code is the same, since the message is not static, any padding required will be more difficult since you will have to assign it to the variable as well. With Paradox generated status messages this may not be worth the effort since you would probably have to evaluate what the message is to know how much padding it might need. You can, of course, create a generic padding routine as well. An example of such a padding routine is included in the status message display object in the bublpart.fsl. Still, most messages should display fine; especially single line messages (ones that take less than 3.1 logical inches) since the single line display code in version 3.0 has been improved for all methods.

Calling these messages only require the assignment of the variable stMsg and, as with all the BubbleHelp methods, the assignment of the formname variable. Examples of how to use these methods are included in the bublpart.fsl. These methods are used in the new message sizing test objects now included on the bublpart.fsl

The other 4 methods which have been added are for use of the new scrolling bubble object which is named helptextscroll. These methods are:

disphelpscroll(msgid smallint, tblname string, formname string)
disphelpleftscroll(msgid smallint, tblname string, formname string)
disphelptopscroll(msgid smallint, tblname string, formname string)
disphelptopleftscroll(msgid smallint, tblname string, formname string)

These methods are called exactly like the original BubbleHelp methods by assigning the msgid, tblname and formname variables and calling the method. These methods require that the helptextscroll object be visible to the calling object. The helptextscroll object behaves differently than other help message objects however. First, it is a fixed size object with a vertical scroll bar. It can hold any amount of text but is more useful in locations where larger help bubbles would be more difficult to fit. This object displays at 2 logical inches wide, including the scroll bar and about 4 text lines high. For a user to see the rest of the message they must scroll it. This requires user intervention and also requires more complex calling and hiding code on the part of the developer. To make the object scrollable the focus must be moved to it. This means that after calling any of these methods you must also assign the helptextscroll object’s tab stop property to on and then move to the helptextscroll object. The helptextscroll object has code to turn the tab stop off again and make itself invisible when it is no longer the active object on the form. You should not change that code in the helptextscroll object. Unlike other help bubble objects you will not turn it off by resetting its visible property but rather removing focus from the bubble and placing it somewhere else. There is an example of how this helptextscroll object works in the bublpart form in the code attached to the Scroll Type button.

The following are the code segments for displaying the Bubble Help from the Mousenter method. You can use the same code for the arrive method of a field. You must specify the message ID for the message you want to display, the Alias, if any, and table name where the help messages reside and the form name of the calling form as specified in the form’s title. If you are using the “wobject” versions of the methods you must also include the full container name of the helptext object as described above. These variable are passed on to the library method to be used for displaying the help message. The simplest way to use RDA’s Bubblehelp is just to call one of the Bubblehelp methods. This can be done such as:

method mouseEnter(var eventInfo MouseEvent)
```
;This displays the help to the lower right
;set the message, message table and display form for this bubble help
```

```
bubblehelp.disphelp(3,":BubbleHelp:helpmsg.db", "Bubble help")
endmethod
```

If you were using one of the “wobject” versions of the methods and had placed the helptext object as helptext2 on a page object called page2 the code would be:

```
method mouseEnter(var eventInfo MouseEvent)
;This displays the help to the lower right
;set the message, message table and display form for this bubble help

bubblehelp.disphelpwobject(3,":BubbleHelp:helpmsg.db", "Bubble help", “page2.helptext2")
endmethod
```

If you were using one of the “direct” versions of the methods and wanted to call the memo in the current Helpmsg multirecord object the code would be:

```
method mouseEnter(var eventInfo MouseEvent)
;check to see if the record is in edit mode

if isedit() then

;If it is, post it  if we can
        if Helpmsg.postrecord() then

        ;assign the value to the variable
        stMsg = helpmsg.messagetext
        else

        ;show why would could not post and then kill the process
        Errorshow("Could not post memo record")
        return
        endif
else

;if we are not in edit mode just assign the variable value
stMsg = helpmsg.messagetext
endif

;make sure the variable is not unassigned. It can be a blank memo.
if not stMsg.isassigned() then
msgstop("Message Unnassigned", "You must have a message displayed in the messsage text area
before you can test the disply. Please enter a message and try again.")
else

;call the method by passing the message and the form to display it on.
bubblehelp.disphelptopdirect(stMsg, "Bubble Parts")
endif
endMethod
```

You can call Bubblehelp more conditionally if you wish with the code below. Note that the self.moveto() may be a good idea in some cases but is optional and will need to be removed in cases where another object will need to retain focus when the code is being executed. In cases where the object needs to retain focus you will not be able to use the scrolling type of message since that type of message requires that it receive focus for the user to be able to scroll the message.

```
method mouseEnter(var eventInfo MouseEvent)
;This displays the help to the lower right
Self.moveto()                    ;move to the object
if Helpval = "Yes" then    ;evaluate the value of the helpval var. If its Yes we want a bubble.

;set the message, message table and display form for this bubble help
bubblehelp.disphelp(3,":BubbleHelp:helpmsg.db", "Bubble help")

else ;otherwise set a short message to be displayed in the status bar. This part of the code can be droped
if not desired
hmsg ="Bottom right"

Message(hmsg)
endif
Self.moveto() ;reset the focus if we lost it.
endmethod

method mouseEnter(var eventInfo MouseEvent)
;This displays the help to the lower left
Self.moveto()                    ;move to the object
if Helpval = "Yes" then    ;evaluate the value of the helpval var. If its Yes we want a bubble.

;set the message, message table and display form for this bubble help
bubblehelp.disphelpleft(3,":BubbleHelp:helpmsg.db", "Bubble help")

else ;otherwise set a short message to be displayed in the status bar. This part of the code can be droped
if not desired
hmsg ="Bottom Left"

Message(hmsg)
endif
Self.moveto() ;reset the focus if we lost it.
endmethod


method mouseEnter(var eventInfo MouseEvent)
;This displays the help to the top right
Self.moveto()                    ;move to the object
if Helpval = "Yes" then    ;evaluate the value of the helpval var. If its Yes we want a bubble.

;set the message, message table and display form for this bubble help
bubblehelp.disphelptop(3,":BubbleHelp:helpmsg.db", "Bubble help")

else ;otherwise set a short message to be displayed in the status bar. This part of the code can be droped
if not desired
hmsg ="Top right"

Message(hmsg)
endif
Self.moveto() ;reset the focus if we lost it.
endmethod
```

```
method mouseEnter(var eventInfo MouseEvent)
;This displays the help to the Top Left
Self.moveto()                    ;move to the object
if Helpval = "Yes" then    ;evaluate the value of the helpval var. If its Yes we want a bubble.

;set the message, message table and display form for this bubble help
bubblehelp.disphelptopleft(3,":BubbleHelp:helpmsg.db", "Bubble help")

else ;otherwise set a short message to be displayed in the status bar. This part of the code can be droped
if not desired
hmsg ="Top Left"

Message(hmsg)
endif
Self.moveto() ;reset the focus if we lost it.
endmethod
```

In addition to toggling between BubbleHelp and status bar messages, as described in the code above, you can have different BubbleHelp messages, for example, a short one and a long one, display by having a second BubbleHelp call to a different message id in the else statement. For example:

```
method mouseEnter(var eventInfo MouseEvent)
;this method displays a Long or short Bubblehelp message and a status message
Self.moveto()
if Helpval = "Yes" then    ;evaluate the value of the helpval var. If its Yes we want the long bubble
message.
bubblehelp.disphelp(42,":BubbleHelp:helpmsg.db", "Bubble help")

else ;otherwise set a short message to be displayed in a bubble
bubblehelp.disphelp(43,":BubbleHelp:helpmsg.db", "Bubble help")
endif

;either way, display a short message in the status bar. This part of the code can be droped if not desired
hmsg ="Open Data Entry Screen."
Message(hmsg)
Self.moveto()
Endmethod
```

You can even get more complex if you wish and use case statements and various variable values to have different messages display on any object depending on multiple contexts and conditions. Get as creative with the calling code as you like.

You must then place the following code segment for clearing the display of the Bubble Help. You should put this code in the Pushbutton and Mouseexit methods for calls from the Mouseenter method and in the depart method for calls from the arrive method for fields. (Remember to change the name of helptext to whatever you have renamed the helptext field to in using the “wobject” versions of the methods.)

```
method pushButton(var eventInfo Event)
```

```
if Helptext.visible  = yes then ;is the Bubble Help display visible
Helptext.visible  = no          ;then clear it
endif

;if you use the optional message call include the message("") call here
message("")

;then your pushbutton code here
```

endmethod

In the form’s Arrive method, or Open method if you prefer, you need to place the code below. Some of the components are optional. If you wish BubbleHelp to always be on you do not need to use the helpon and helpoff buttons nor set their initial visible state. You can also reverse the states shown below if you want to start with Bubblehelp on as the default. If you want BubbleHelp to be on all the time do not use the Helpval variable or set it to Yes. If you want it to be on by default but able to be turned off then set the helpval to Yes. If you place the RDAbubbl.ldl in a different alias than the BubbleHelp then change the alias in the open() line for the library.

method arrive(var eventInfo MoveEvent)

```
        if eventInfo.isPreFilter() then
                ;// This code executes for each object on the form

        else
                ;// This code executes only for the form

;if you use the helpon and help off buttons this sets their state
;reverse the yes and no if you want to start with bubblehelp active

helpon.visible = yes
helpoff.visible = no

;Then set the help value to no or yes depending on the state you want bubblehelp in

Helpval = "No"

;Open the library (make sure you specify the correct alias if you don't use the one specified here)

bubblehelp.open(":Bubblehelp:rdabubbl.ldl")

;your additional code here

        endIf
```

endMethod

In the form’s Uses method you need to place the code below. (Note that you only have to declare methods which you will actually use, not every method on the list)

Uses objectpal

```
disphelp(msgid smallint, tblname string, formname string)
disphelpleft(msgid smallint, tblname string, formname string)
disphelptop(msgid smallint, tblname string, formname string)
disphelptopleft(msgid smallint, tblname string, formname string)
disphelpwobject(msgid smallint, tblname string, formname string, helptextname string)
disphelpleftwobject(msgid smallint, tblname string, formname string, helptextname string)
disphelptopwobject(msgid smallint, tblname string, formname string, helptextname string)
disphelptopleftwobject(msgid smallint, tblname string, formname string, helptextname string)
disphelpdirect(stMsg Anytype, formname string)
disphelpleftdirect(stMsg Anytype, formname string)
disphelptopdirect(stMsg Anytype, formname string)
disphelptopleftdirect(stMsg Anytype, formname string)
disphelpscroll(msgid smallint, tblname string, formname string)
disphelpleftscroll(msgid smallint, tblname string, formname string)
disphelptopscroll(msgid smallint, tblname string, formname string)
disphelptopleftscroll(msgid smallint, tblname string, formname string)

;any other objectpal uses statements you need here
endUses
```

In the form's Var method you need to place the code below.

```
Var
bubblehelp library
helpval string
;any other variables you need to declare here
endVar
```

Both the Help On Help Off button combo, the helptext field and the helptextscroll field can be copied from the butnpart.fsl. The helptext field is required except for the "wobject" methods and the "scroll" methods. Those require the helptext field or a renamed copy of it for the "wobject" methods or the helptextscroll object for the "scroll" methods. Do not change their properties as they are set to take advantage of the sizing code in the library. The Help On - Help Off combo is optional.

If you plan to use the Help On and Help Off button copy them, paste them on your form and create the message you want the buttons to display. Help On is presently set only to display a short message to the status bar as if you use it the button will toggle the Button help off so it would not display a BubbleHelp message under any circumstances. Help Off is presently set to display a demo message from the helpmsg table included with the application. Be certain to change the Mouseenter method of the Help Off to point to your message id, message table and form name. You can change the settings and use of the Help On and Help Off buttons as you wish or create your own system for changing the values to initialize BubbleHelp messages.

Once you have installed these components you will have functioning BubbleHelp when you run your forms. BubbleHelp works with delivered forms as well as with running Fsls.

Tips

When a message is not conveniently sized in a box, padding with spaces is often the best way to fix the problem.

Because a proportional font is used you can often observe where the message wraps to a last line and pad with spaces before and after the first and last word in the last line to get a good look since text is centered. Padding before the first word of the last line is the most effective because of the way the sizing code and text centering work. Spaces are sized differently when placed between words in proportional fonts. But the sizing code works on character counts and average character sizes so adding spaces increases the height of the message when the message text wraps.

If you have insufficient height in a multiline message, padding will usually fix the problem as the height is related to the average size of the characters and the character count.

Sometimes moving longer or shorter words around in your text can cause the text to wrap in different spots and get a clean look.

Single line messages should not need padding for height in BubbleHelp 3.0 as they did in earlier versions. They still may need padding to eliminate word wrapping cutting off the last word in a single line message. Padding at the end of single line messages tends to produce the best display because of the way proportional fonts are handled. In some circumstances front padding will improve the centering left to right.

For messages up to 60 characters, sizing code changes at ten character intervals so padding near or across a breakpoint often achieves the desired results.

You can open one session of paradox to be editing the messages and another to your application and switch back and forth while adjusting your messages to check message fit. You must unlock the record, however, for it to be read into the bubble.

You can also test the current message displayed in the bublpart.fsl as you create messages by moving your mouse to one of the light green test objects to the side of the help message edit area.

The edit field on the Bublpart.fsl for the memo text is approximately the correct width, and uses the same font and wrapping as the helptext field. Using it to edit your messages can help you with the wrapping and padding characteristics for the best display of messages. You can copy it to other forms as needed. Then save your form, close it, open it using the change table option to select your message table. You can save it after that if you wish to have a form specifically for editing one applications Bubblehelp messages.

BubbleHelp places the message text display at the corner of the object you select. If there is not enough room on the form for the text object to display from that location to the edge or edges of the form it will push back from the form edge as much as needed to display. This may, however cause the effect of overlapping the calling object.

Since BubbleHelp messages are positioned relative to the container of the helptext object you will get more accurate positioning on multipage forms if you stack the pages rather than tile them.

Otherwise positioning will be off by as much as the part of any nonfocal page which can also be seen in the form window above the top left corner of the focal page.

The BubbleHelp helptext field in the Windows versions uses the MS sans serif 8 point font. Since there is no direct match to this font in Paradox for Linus, the AlbanyAMT 8 font is used. If you are using the same application in both Paradox for Windows and Paradox for Linux you may find you need to adjust the padding slightly differently for each version. This can be done either by copying the message data set and then adjusting messages that need it or by having a second version of the message in the same table and having the Linux version call one with appropriate padding for its font and the Windows version call one with the appropriate padding for its font. Since the fonts are close and the code has been adjusted to compensate for the differences you should not encounter this problem too frequently.