Support Group Application Note
Number: 241
Issue: 1.00
Author: SH/(JB)

Acorn

The RISC OS Drag-and-Drop System

This application note describes how drag-and-drop data transfer within and between applications could be implemented, and how to make sure that your applications co-operate with others to promote consistent data transfer facilities.

This application note is based on a protocol designed by Iota Software Ltd.

Applicable Hardware :	All Acorn RISC OS based computers
Related Application Notes:	The RISC OS Selection Model and Clipboard

Copyright © Acorn Computers Limited 1993

Every effort has been made to ensure that the information in this leaflet is true and correct at the time of printing. However, the products described in this leaflet are subject to continuous development and improvements and Acorn Computers Limited reserves the right to change its specifications at any time. Acorn Computers Limited cannot accept liability for any loss or damage arising from the use of any information or particulars in this leaflet. ACORN, ECONET and ARCHIMEDES are trademarks of Acorn Computers Limited.

Support Group
Acorn Computers Ltd
Acorn House
Vision Park, Histon
Cambridge CB4 4AE
England

Drag and Drop

The goal of this protocol is to allow the user to copy and move data, both within and between applications, by direct manipulation. This means that the user should be able to make a selection in a document, pick the selected objects up with the mouse, drag them to their destination and release the mouse button. To facilitate precise positioning of the dragged objects, provision is made for the receiving task to display a *ghost caret*, which tracks the mouse pointer in the target window and indicates exactly where the data will be inserted. For text, the ghost caret should look similar to the normal Wimp caret, and as its follows the mouse pointer it should "snap" to the nearest inter-character gap. The appearance of a ghost caret for non-textual selections will vary, but might typically be the bounding box of the dragged data, scaled according to the destination window's zoom factors.

This document assumes that you have implemented the selection system described in the application note *The RISC OS Selection Model and Clipboard.*

User Interface

The following mouse controls exist on a text area. Non-textual documents will work similarly, but the details of the selection model and visual feedback may differ according to the nature of the data being manipulated.

Clicking Select where there is no selection sets the caret position

Clicking Adjust when there is a caret creates a selection between the old caret position and the mouse position.

Dragging with Select where there is no selection starts a new selection, and subsequent dragging moves the other end.

Dragging with Adjust alters the ‘nearest’ end of the selection.

Clicking Select over the selection has no effect (to allow the drag to occur properly - see below).

Dragging with Select over the selection causes a box the size of the selection to be dragged. While the box is being dragged over a potential destination, a ghost caret may be displayed by the application that owns the destination window.

While a drag is in progress, placing the mouse pointer close to (but inside) an edge of the window should cause the window to scroll automatically in the appropriate direction. Scrolling is the responsibility of the application that owns the destination window.

Releasing Select at the destination causes the selection to be copied or moved to the destination. If the destination is within the same window, then the data should be moved, unless SHIFT is held down, when the data should be copied instead. However, if the destination is a different window, the sense of the SHIFT key should be reversed so that the default action is a copy.

While the selection is being dragged, the following message protocol is used to implement automatic scrolling and 'ghost caret' positioning:

Message_Dragging (17)

0	message size
4	task handle of task performing the drag
8	message id
12	your_ref (0)
16	Message_Dragging
20	window handle
24	icon handle
28	x
32	y
36	flags: bit 1 set => sending data from selection bit 2 set => sending data from clipboard bit 3 set => source data will be deleted bit 4 set => do NOT claim this message all other bits reserved (must be 0)
40	x0, y0, x1, y1: bounding box of data relative to the pointer, in 1/72000" (not scaled according to the zoom factors of the source window) x0 > x1 => source data size unknown
56	list of filetypes in sender's order of preference, terminated by -1

Message_DragClaim (18)

0	message size
4	task handle of task replying to Message_Dragging
8	message id
12	your_ref (message id of the Message_Dragging received)
16	Message_DragClaim
20	flags: bit 0 set => pointer shape has been changed bit 1 set => remove wimp dragbox bit 3 set => source data should be deleted all other bits reserved (must be 0)
24	list of filetypes in receiver's order of preference, terminated by -1

All messages are sent using type 18 (reply required), and the 'your_ref' field of each message is set to the message id of that to which it is replying, unless otherwise stated.

During the drag, the dragging task enables null events using Wimp_PollIdle with a minimum return time of about 0.25 second. On each null event, it finds the current mouse position using Wimp_GetPointerInfo. It then constructs the message block for Message_Dragging using the information gleaned.

If no task has claimed control of the pointer (as none has at the start), the message is sent to the owner of the window/icon pair returned by Wimp_GetPointerInfo.

If the dragging task receives a Message_DragClaim in reply, it stores the task / your_ref / flags of the message to indicate that the given task has claimed the drag process.

On the next null event, if the drag has been claimed, the Message_Dragging is sent to the claimant (with your_ref set as appropriate), otherwise it is sent to the window/icon pair returned by Wimp_GetPointerInfo as before.

If the Message_Dragging bounces (note that a DragClaim reply is required for each Message_Dragging that is sent), and the drag was being claimed, the record of the claimant is reset and the Message_Dragging is resent, this time to the window/icon pair (with a your_ref of 0).

The claimant will normally only claim the drag again if the pointer is still within its window, but it may continue if it has started auto-scrolling. If bit 4 of the flags in the Message_Dragging is set, the claimant MUST relinquish the drag, since this indicates that the sender is terminating the drag process.

When the drag terminates, the dragging task sets a flag and calls the null event code once more (to ensure that the mouse position is up-to-date). This results in a Message_Dragging being sent, either to the claimant or to the window/icon pair under the pointer.

This will result in a DragClaim being returned, or the message bouncing. If the message bounces, and there was a claimant, the Message_Dragging is resent, as for a null event.

However, if a DragClaim is received, or the Message_Dragging bounces and there is no claimant, then the 'drag terminated' flag is inspected, and if set, the data can finally be saved to the destination.

The sender is also responsible for detecting the user aborting the drag using Escape - in this case it should set an internal 'Aborted' flag, call Wimp_DragBox with -1 to terminate the drag, and then proceed as for drag finished, terminating the process just before the DataSave would have been sent.

The sender should keep a record of whether the pointer shape has been changed (ie. keep a copy of the last set of DragClaim flags received), and reset the pointer shape whenever the 'pointer altered' bit changes from a 1 to a 0. Thus if the other application first changes the pointer shape, sending a DragClaim with flags bit 0 set, then next time does not wish to change the pointer shape, but nevertheless wants to keep the drag, it will send a DragClaim with flags bit 0 clear, so the sender should reset the pointer shape.

The 'remove wimp dragbox' flag works slightly differently - since the receiver is not allowed to alter the wimp dragbox, it should instead ask the sender to do this by setting bit 1 of the DragClaim message. If at some later time a DragClaim is sent without this bit set, or the Message_Dragging bounces, the sender should restore the wimp dragbox. It should do this by calling Wimp_DragBox again with the appropriate drag type.

The bounding box in internal units allows the receiver to display the exact position of where the data will be placed if the mouse button is released, and is an exact analogy of the ghost caret for text. The receiver should display the box exactly where the data would go, i.e. corrected for grid alignment, snap to frames etc.

Note that sometimes the sender does not know the bounding box of the data, or sometimes the concept may be meaningless (e.g. for text transfer). In this case the bounding box in the Message_Dragging should be set so that x0 > x1, and the receiver should check for this and not use a box to display the position of the data (although it may still use a ghost caret if appropriate). The Wimp drag box should be used if the receiver has no way to display the destination position.

Bit 3 of the DragClaim message can be set to indicate that the sender should delete the source data even if the user did not press the Shift key. This option could be used by a trashcan application, so that objects

dragged to it are simply deleted.

When the sender finally comes to save the data, it should send in its own preferred format, unless a DragClaim is in force and the sender can do one of the the filetypes in the list returned. If so, it must do the first one in the list that it can.

Before sending the data, the sender should now reset the pointer shape to the default if the claimant's last DragClaim had the 'pointer shape altered' flag set.

The normal Message_DataSave protocol is used, except that the your_ref of the Message_DataSave is set to 0 if there was no claimant, and to the stored claimant's last DragClaim message id otherwise.

Thus the receiver of the Message_DataSave knows whether to alter the insertion point according to the x,y position indicated, and whether the 'ghost caret' needs to be removed from view.

In summary, the two tasks should behave as follows:

Sender:

```
start:
   call Wimp_DragBox to start the drag, and enable null events every 0.25s
   set claimant to 'none'
   set drag_finished to 'false'
   set drag_aborted to 'false'
   set lastref to 'none'

null event:
   read mouse position and construct Message_Dragging
   if claimant
     send message (18) to claimant (your_ref = lastref)
   else
     send message (17/18) to window/icon pair (your_ref = 0)
   endif

escape pressed:
   set drag_aborted to 'true'
   call Wimp_DragBox with -1 to terminate the drag
   proceed as for drag finished

drag finished:
   set drag_finished to 'true', then call null event code

Message_DragClaim received:
   if drag_finished
     reset pointer shape if claimant altered it
     unless drag_aborted send data to claimant (your_ref = my_ref of DragClaim)
   else
     reset pointer shape if (old_flags and not new_flags) has bit 0 set
     set claimant to task handle of DragClaim message
     set lastref to my_ref of DragClaim message
   endif
```

```
Message_Dragging bounced:
      if claimant is not 'none'
        reset pointer shape if claimant altered it
        set claimant to ‘none’
        resend message to window/icon pair (type 17/18)
      else
        if drag_finished
          unless drag_aborted send data to window/icon pair (your_ref = 0)
        endif
      endif

      ‘17/18’ means ‘if drag_finished, send as type 18, else as type 17’.
      bit 4 of any Message_Dragging sent is set if drag_aborted is true.
```

Receiver:

```
   start:
      set claiming to ‘false’
      set pointer_altered to ‘false’

   Message_Dragging received:
      if claiming,
        if flags bit 4 clear, and we’re auto-scrolling or in a text area
          reply with Message_DragClaim (type 17)
          perform auto-scroll if required
          update ghost caret if required
       else
          set claiming to 'false'
          undraw ghost caret if necessary
          don’t reply to message
        endif
      else
        if flags bit 4 clear
          if we’re in the auto-scroll area
            change pointer shape to indicate auto-scroll
            set timer for auto-scroll
            set pointer_altered to ‘true’
            reply with Message_DragClaim (type 17)
          elseif we’re in a text area
            draw ghost caret in correct place
            set pointer_altered to ‘false’
            reply with Message_DragClaim (type 17)
          endif
        endif
      endif
```

```
Message_DataSave received:
      if your_ref indicates we're claiming:
        if ghost caret is set up
          set insertion point to ghost caret
          undraw ghost caret
        endif
      endif
      import data to insertion point
```

All Message_DragClaim messages are sent with bit 0 of the flags indicating the current value of the 'pointer_altered' flag.