Support Group Application Note
Number: 240
Issue: 1.00
Author: SH/(JB)

Acorn

The RISC OS Selection Model and Clipboard

The second edition of the RISC OS Style Guide recommends the use of clipboard-based (cut-copy-paste) data transfer. This application note describes how this could be implement, 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 Drag-and-Drop System

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

Definitions

Selection: The portion of a document which the user has chosen as the target for subsequent operations. This may be a contiguous selection (as in the case of selected text) or a non-contiguous selection (as in the case of a number of selected files in the Filer).

Caret: the position in a document where typed characters or pasted clipboard contents will be placed. In textual documents, the caret is often shown by an I-beam, but other representations of the caret may be more appropriate for other kinds of data. Some editors, such as !Draw, do not have a visible insertion point, but still "grab the caret" in order to receive keystroke events.

Input focus: the window where keystroke events will be delivered.

Clipboard: a temporary storage area that holds data while the user is copying or moving it (either within one application or between applications).

Cut: the operation that the user invokes to transfer the current selection to the clipboard. The selection is deleted from the source document.

Copy: as cut, but does not delete the original.

Paste: the operation that the user invokes to transfer the clipboard contents into a document.

The Selection Model

There should normally be one caret or selection active in each window. They are mutually exclusive; a window's caret and selection are not normally visible at the same time. When the user clicks to position the caret or makes a selection, any existing selection in the same window should be de-selected. In text editors it is helpful to think of the caret as a zero-width selection; if the user makes a selection, then the caret becomes invisible, and if the user clicks to set the caret. then any selection is de-selected.

Each window has its own selection, even multiple views of the same document. Making a selection in one window should not affect any selections active in other windows.

The Clipboard

Copying and moving data within and between windows is accomplished by means of a temporary holding area known as the clipboard. The clipboard supports three operations. The *Copy* operation copies the selected data from the source window to the clipboard, leaving the selection intact. The *Cut* operation is similar to Copy, but deletes the selected data from the document. The *Paste* operation inserts a copy of the clipboard contents to the caret position of the destination window (or, if there is a selection rather than a caret, it replaces the selection with the contents of the clipboard). The user combines these operations as required to effect the copying and movement of data. The clipboard is a systemwide entity, so the operations work between applications as well as within one application.

From an implementation point of view, the clipboard is not a specific area of memory, but a notional entity which applications claim and keep track of by means of messages. The data "on" the clipboard is held by the application that performed the last "cut" or "copy" operation.

Data entered from the keyboard, or pasted in from the clipboard, should replace the current selection if there is one, or be inserted at the caret position if there is no selection. Graphical applications that do not have a visible caret must choose where to position pasted data in an appropriate way. In any case, the pasted data should be automatically selected so that the user can immediately cut it again should this be desired.

If the user replaces a selection by typing new data, the selection should be replaced by a caret and the deleted data should be cut to the clipboard as a safety measure - the user can paste it back again if the operation was unintended. Do not do this if the selection is replaced by pasting from the clipboard, because it would prevent the same data being pasted multiple times.

Caret / Selection / Clipboard control

Ownership of the caret / selection / clipboard

Each task keeps a separate record of the position of the caret/selection for each of its windows. It also stores flags to indicate whether it currently "owns" the input focus and the clipboard.

To enable applications to track changes in the status these entities, the following message is used:

Message_ClaimEntity (15)

0	message size (24)
4	task handle of task making the claim
8	message id
12	your_ref (0)
16	Message_ClaimEntity
20	flags: bits 0 and 1 set => caret or selection being claimed bit 2 set => clipboard being claimed all other bits reserved (must be 0)

This message should be broadcast to all tasks as the caret / selection or clipboard are claimed.

When the user positions the caret or makes a selection, the application should claim ownership of the input focus by broadcasting this message with bits 0 and 1 set. When positioning the caret, the application can choose whether to use the Wimp's caret or draw its own representation of the caret more appropriate to the type of data being edited. When making a selection, the application must hide the caret; it should do this by setting the Wimp's caret to the window containing the selection, but invisible. This is necessary to direct keystroke events to this window.

When a task receives this message with bits 0 or 1 set, it should check to see if any of its windows currently own the input focus. If so, it should update its flag to indicate that it no longer has the focus, and remove any representation of the caret which it has drawn (unless it uses the Wimp caret, which will be undrawn automatically.) It may optionally alter the appearance of its window to emphasize the fact that it does not have the input focus, for example by shading the selection. A task that receives Message_ClaimEntity with only one of bits 0 and 1 set should act as if both bits were set.

When the user performs a Cut or Copy operation. the application should claim ownership of the clipboard by broadcasting this message with bit 2 set.

When a task receives this message with bit 2 set it should set a flag to indicate that the clipboard is held by another application and deallocate the memory being used to store the clipboard contents.

To improve performance, the following optimisation should be made:

When claiming the input focus or clipboard, a task should check to see if it already owns that entity, and if so, there is no need to issue the broadcast. It should then take care of updating the caret / selection / clipboard to the new value (updating the display in the case of the selection).

LoseCaret / GainCaret events

As far as keeping track of whether they have the input focus is concerned, applications implementing the above protocols should ignore LoseCaret/GainCaret messages This is because the Wimp often ‘borrows’ the caret to put it in writable menu icons and when this happens the application should not regard the input focus as being lost.

Older applications which claim the caret without using the claiming mechanism described will also not indulge in the cut/copy/paste protocol so, again, the LoseCaret event should be ignored.

Selection History

To return to a window that does not currently have the input focus, the user clicks in it to set the caret. Because the caret and selection are mutually exclusive within each window, this causes any selection that was pending to be lost. However, the user may occasionally want to return the input focus to a window without losing its selection, particularly if re-making the selection would be fiddly or time-consuming.

Therefore a click *within* the current selection or in a "dead" area of the window (e.g. in a border) should be treated specially. It should cause the window to regain the input focus *without* setting the caret position and hence without losing the current selection. The application should broadcast Message_ClaimEntity with bits 0 and 1 set, and set the Wimp caret to the window. If there is a selection in the window then the caret should be made invisible. If there is no selection then the caret should be shown and restored to its previous position within the window.

Cutting and pasting data

Applications should provide menu entries for Cut, Copy and Paste operations. Refer to the Style Guide for details of where to place these in your menu tree and which keyboard shortcuts to assign to them.

Cut

If the application does not have a selection, then this is a not possible and should be shaded on the menu.

If the application already owns the clipboard, it should free its current contents. If not, it should claim the clipboard by broadcasting Message_ClaimEntity setting flags bit 2. It should then set up its internal representation of the clipboard contents to be a copy of the selected data, and delete the selection.

Copy

As Cut, but the selection should not be deleted.

Paste

The application should first check to see if it owns the clipboard, and use the data directly if so. If is does not own it, it should broadcast the following message:

Message_DataRequest (16)

0	message size
4	task handle of task requesting data
8	message id
12	your_ref (0)
16	Message_DataRequest
20	window handle
24	internal handle to indicate destination of data
28	x
32	y
36	flags: bit 2 set => send data from clipboard (must be 1) all other bits reserved (must be 0)
40	list of filetypes in order of preference, terminated by -1

The sender must set flags bit 2, and the receiver must check this bit, and ignore the message if it is not set. All other flags bits must be cleared by the sender and ignored by the receiver.

If an application receiving this message owns the clipboard, it should choose the earliest filetype in the list that it can provide, and if none are possible it should provide the data its original (native) format. Note that the list can be null, to indicate that the native data should be sent. It should reply using the normal Message_DataSave protocol. Bytes 20 through 35 of the DataSave block should be copied directly from the corresponding bytes of the Message_DataRequest block, whilst the estimated size field, filetype and filename must be filled in.

When the application that initiated the Paste receives the Message_DataSave, it should check the filetype to ensure that it knows how to deal with it - it may be the clipboard owner's native format. If it cannot, it may back out of the transaction by ignoring the message. Otherwise, it should continue with the DataSave protocol as detailed in the Programmer's Reference Manual.

If your application needs to find out whether there is data available to paste, but does not actually want to receive the data, you should broadcast a Message_DataRequest as described above. If no task replies (i.e. you get the message back) then there is no clipboard data available. If a Message_DataSave is received, then you should ignore it (fail to reply), which will cause the operation to be silently aborted by the other task. You can then use the filetype field of the Message_DataSave to determine whether the data being offered by the other task is in a suitable format for you to receive.

This mechanism can be used if you want to shade out the Paste menu item when there is no suitable data for pasting.