A. Message Protocols

System Messages 3
System Management Messages 3
B_HANDLERS_REQUESTED 4
B_QUIT_REQUESTED 4
Application Messages 4
B_ABOUT_REQUESTED 4
B_ACTIVATE 5
B_APP_ACTIVATED 5
B_ARGV_RECEIVED 5
B_PANEL_CLOSED 5
B_PULSE 6
B_QUIT_REQUESTED 6
B_READY_TO_RUN 6
B_REFS_RECEIVED 6
Interface Messages 7
B_KEY_DOWN 7
B_KEY_UP 9
B_MINIMIZE 9
B_MOUSE_DOWN 9
B_MOUSE_MOVED 10
B_MOUSE_UP 11
B_PANEL_CLOSED 12
B_PULSE 12
B_QUIT_REQUESTED 13
B_SAVE_REQUESTED 13
B_SCREEN_CHANGED 14
B_VALUE_CHANGED 14
B_VIEW_MOVED 14
B_VIEW_RESIZED 15
B_WINDOW_ACTIVATED 15
B_WINDOW_MOVED 16
B_WINDOW_RESIZED 16
B_WORKSPACE_ACTIVATED 16
B_WORKSPACES_CHANGED 17
B_ZOOM 17

Standard Messages 18
Reply Messages 18
B_HANDLERS_INFO 18
B_MESSAGE_NOT_UNDERSTOOD 18
B_NO_REPLY 19
Editing Messages 19
B_CUT, **B_COPY**, and **B_PASTE** 19
B_SIMPLE_DATA 19
Interapplication Messages 20

A. Message Protocols

This appendix details the formats for all public messages produced and understood by Be system software. The list includes all system messages, all other messages that might find their way to your application (for example, through a drag and drop operation), and all messages that you can deliver to a Be application or a Be-defined class.

For information on the messaging system, see "Messaging" in *The Application Kit* chapter.

System Messages

Messages that are dispatched and handled in a message-specific manner are known as *system messages*. For the most part, these are messages that the system produces and that applications are expected to respond to (by implementing a hook function matched to the message), but some are messages that applications must produce themselves. They fall into three categories:

- *System-management messages* can be delivered to any BLooper,
- *Application messages* are consigned to the BApplication object, and
- *Interface messages* are reported to BWindow objects.

For information on the place of system messages in the messaging system, see "System Messages" in the introduction to *The Application Kit* chapter.

System Management Messages

System management messages are concerned with running the messaging system. The BLooper class in the Application Kit declares hook functions for two such messages. (See also "System Management Messages" on page 15 of *The Application Kit* chapter.)

B_HANDLERS_REQUESTED

This message asks a target BHandler to supply BMessenger objects as proxies for other BHandlers. The BLooper dispatches it by calling the target's **HandlersRequested()** function; the target should respond with a **B_HANDLERS_INFO** reply.

The **HandlersRequested()** functions implemented in the Application and Interface Kits look for the following data entries in the message. See those functions for details.

Data name	Type code	Description
"index"	**B_LONG_TYPE**	An index into a list of BHandlers kept by the target BHandler.
"name"	**B_STRING_TYPE**	The name of a BHandler.
"class"	**B_STRING_TYPE**	The name of a class derived from BHandler or "BHandler" itself.

Since applications initiate **B_HANDLERS_REQUESTED** messages, they are free to use whatever protocols prove useful for requesting BHandler proxies. The data entries listed above are simply those that the Be-defined functions expect.

B_QUIT_REQUESTED

This message contains no data. It simply asks a BLooper to quit its message loop and destroy itself. The Blooper dispatches the message by calling its own **QuitRequested()** function.

This message is reinterpreted by the BApplication object to mean a request to quit the application and by a BWindow object to mean a request to close the window. It's therefore also listed under "Application Messages" and "Interface Messages" below.

Application Messages

Application messages concern the application as a whole, rather than one specific window or thread. They're all received and handled by the BApplication object. See "Application Messages" on page 16 in the introduction to *The Application Kit* chapter for information on when they're produced and how they should be handled.

B_ABOUT_REQUESTED

This message contains no data entries. It requests the BApplication object to put a window on-screen with information about the application. Applications should produce it when the user chooses the "About . . ." item in the main menu. The BApplication object dispatches the message by calling its own **AboutRequested()** function.

B_ACTIVATE

This message contains no data entries. It instructs the application to make itself the active application. The BApplication object dispatches it by calling **Activate()**, defined in the BApplication class.

B_APP_ACTIVATED

This message informs the application that it has become the active application, or that it has ceded that status to another application. The BApplication object dispatches the message by calling **AppActivated()**.

It contains one data entry:

Data name	Type code	Description
"active"	**B_BOOL_TYPE**	**TRUE** if the application has just become the active application, and **FALSE** if it just gave up that status.

B_ARGV_RECEIVED

This message passes the BApplication object command-line strings, typically ones the user typed in a shell. The BApplication object dispatches it by calling **ArgvReceived()**.

The message has the two expected data entries for command-line arguments:

Data name	Type code	Description
"argc"	**B_LONG_TYPE**	The number of items in the "argv" array. This will be the same number that **BMessage::GetInfo()** for "argv" would report.
"argv"	**B_STRING_TYPE**	The command-line strings. Each argument is stored as an independent item under the "argv" name—that is, there's an array of data items, each of type **char ***, rather than a single item of type **char ****.

B_PANEL_CLOSED

This message notifies the application that the file panel has been removed from the screen. The BApplication object dispatches it by calling **FilePanelClosed()**.

The message has these data entries:

Data name	Type code	Description
"frame"	**B_RECT_TYPE**	The frame rectangle of the panel at the time it was closed. (The user may have resized it and relocated it on-screen.) The rectangle is recorded in screen coordinates.
"directory"	**B_REF_TYPE**	A **record_ref** reference to the last directory displayed in the panel.
"marked"	**B_STRING_TYPE**	The item that was selected in the Filters list when the panel closed.
"canceled"	**B_BOOL_TYPE**	**TRUE** if the panel was closed because the user operated the "Cancel" button and **FALSE** otherwise.

B_PULSE

This message contains no data entries. It's posted at regularly spaced intervals as a kind of timing mechanism. The BApplication object dispatches it by calling the **Pulse()** function declared in the BApplication class.

B_QUIT_REQUESTED

This message contains no data entries. Its dispatching (by calling **QuitRequested()**) is defined in the BLooper class. When it gets the message, the BApplication object interprets it to be a request to shut the entire application down, not just one thread. It consequently promulgates similar messages to all BWindow objects.

B_READY_TO_RUN

This message contains no data entries. It's delivered to the BApplication object to mark the application's readiness to accept message input after being launched. The BApplication object dispatches it by calling **ReadyToRun()**.

B_REFS_RECEIVED

This message passes the application one or more references to database records. It's typically produced by the Browser when the user chooses some files for the application to open. The BApplication object dispatches it by calling **RefsReceived()**.

The message has one data entry, which might be an array of more than one item:

Data name	Type code	Description
"refs"	**B_REF_TYPE**	One or more **record_ref** items referring to database records. Typically, the records are for documents the application is expected to open.

B_REFS_RECEIVED messages can also be dragged to and from Browser windows.

Interface Messages

Interface messages inform BWindow objects and their BViews about activity in the user interface. Unlike application messages, most of which consist only of a command constant, most interface messages contain data entries describing an event. They're all delivered to a BWindow object, which dispatches some to itself but most to its BViews.

See "Interface Messages" on page 41 in *The Interface Kit* chapter for a discussion of the events these messages report.

B_KEY_DOWN

This message reports that the user pressed a character key on the keyboard. It's dispatched by calling the **KeyDown()** function of the target BView, generally the window's focus view. Most keys produce repeated **B_KEY_DOWN** messages—as long as the user keeps holding the key down and doesn't press another key.

Each message contains the following data entries:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the key went down, as measured in microseconds from the time the machine was last booted.
"key"	**B_LONG_TYPE**	The code for the key that was pressed.
"modifiers"	**B_LONG_TYPE**	A mask that identifies which modifier keys the user was holding down and which keyboard locks were on at the time of the event.
"char"	**B_LONG_TYPE**	The character that's generated by the combination of the key and modifiers.

"states"	**B_UCHAR_TYPE**	A bitfield that records the state of all keys and keyboard locks at the time of the event. Although declared as **B_UCHAR_TYPE**, this is actually an array of 16 bytes.

For most applications, the "char" code is sufficient to distinguish one sort of user action on the keyboard from another. It reflects both the key that was pressed and the effect that the modifiers have on the resulting character. For example, if the Shift key is down when the user presses the *A* key, or if Caps Lock is on, the "char" produced will be uppercase 'A' rather than lowercase 'a'. If the Control key is down, it will be the **B_HOME** character. A section of *The Interface Kit* chapter, "Keyboard Information" on page 47, discusses the mapping of keys to characters in more detail.

The "modifiers" mask explicitly identifies which modifier keys the user is holding down and which keyboard locks are on at the time of the event. The mask is formed from the following constants, which are explained under "Modifier Keys" on page 51 in the introduction to *The Interface Kit* chapter.

B_SHIFT_KEY	**B_COMMAND_KEY**	**B_CAPS_LOCK**
B_LEFT_SHIFT_KEY	**B_LEFT_COMMAND_KEY**	**B_SCROLL_LOCK**
B_RIGHT_SHIFT_KEY	**B_RIGHT_COMMAND_KEY**	**B_NUM_LOCK**
B_CONTROL_KEY	**B_OPTION_KEY**	
B_LEFT_CONTROL_KEY	**B_LEFT_OPTION_KEY**	**B_MENU_KEY**
B_RIGHT_CONTROL_KEY	**B_RIGHT_OPTION_KEY**	

The mask is empty if no keyboard locks are on and none of the modifiers keys are being held down.

The "key" code is an arbitrarily assigned number that identifies which character key the user pressed. All keys on the keyboard, including modifier keys, have key codes (but only character keys produce key-down events). The codes for the keys on a standard keyboard are shown in the "Key Codes" section on page 48 in *The Interface Kit* chapter.

The "states" bitfield captures the state of all keys and keyboard locks at the time of the key-down event. (At other times, you can obtain the same information through BView's **GetKeys()** function.)

Although it's declared as **B_UCHAR_TYPE**, the bitfield is really an array of 16 bytes,

```
uchar states[16];
```

with one bit standing for each key on the keyboard. For most keys, the bit records whether the key is up or down. However, the bits corresponding to keys that toggle keyboard locks record the current state of the lock. To learn how to read the "states" array, see "Key States" on page 56 in *The Interface Kit* chapter.

B_KEY_UP

< Key-up messages are not currently reported. >

B_MINIMIZE

This message instructs a BWindow to "minimize" itself—to replace the window on-screen with a small token—or to remove the token and restore the full window. The message is produced when the user double-clicks the window tab or the window token and is dispatched by calling the BWindow's **Minimize()** function.

It contains the following data:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the user acted, as measured in microseconds from the time the machine was last booted.
"minimize"	**B_BOOL_TYPE**	A flag that's **TRUE** if the window should be minimized to a token representation, and **FALSE** if it should be restored to the screen from its minimized state.

B_MOUSE_DOWN

This message reports that the user pressed a mouse button while the cursor was over the content area of a window. It's produced only for the first button the user presses—that is, only if no other mouse buttons are down at the time. The BWindow dispatches it by calling the target BView's **MouseDown()** function.

The message contains the following information:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the mouse button went down, as measured in microseconds from the time the machine was last booted.
"where"	**B_POINT_TYPE**	Where the cursor was located when the user pressed the mouse button, expressed in the coordinate system of the target BView—the view where the cursor was located at the time of the event.
"modifiers"	**B_LONG_TYPE**	A mask that identifies which modifier keys were down and which keyboard locks were on when the user pressed the mouse button.

"buttons"	**B_LONG_TYPE**	A mask that identifies which mouse button went down.
"clicks"	**B_LONG_TYPE**	An integer that counts the sequence of mouse-down events for multiple clicks. It will be 1 for a single-click, 2 for a double-click, 3 for a triple-click, and so on.

The "modifiers" mask is the same as for key-down events and is described under "Modifier Keys" on page 51 in *The Interface Kit* chapter.

The "buttons" mask identifies mouse buttons by their roles in the user interface. It may be formed from one or more of the following constants:

B_PRIMARY_MOUSE_BUTTON
B_SECONDARY_MOUSE_BUTTON
B_TERTIARY_MOUSE_BUTTON

Because a mouse-down event is reported only for the first button that goes down, the mask will usually contain just one constant.

The "clicks" integer counts clicks. It's incremented each time the user presses the mouse button within a specified interval of the previous mouse-down event, and is reset to 1 if the event falls outside that interval. The interval is a user preference that can be set with the Mouse preferences application.

Note that the only test for a multiple-click is one of timing between mouse-down events. There is no position test—whether the cursor is still in the vicinity of where it was at the time of the previous event. It's left to applications to impose such a test where appropriate.

B_MOUSE_MOVED

This message is produced when the user moves the cursor into, within, or out of a window. Each message captures a small portion of that movement. Messages aren't produced if the cursor isn't over a window or isn't moving. The BWindow dispatches each message by calling the **MouseMoved()** function of every BView the cursor touched in its path from its last reported location.

The message contains the following data entries:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the event occurred, as measured in microseconds from the time the machine was last booted.

"where"	**B_POINT_TYPE**	The new location of the cursor, where it has moved to, expressed in window coordinates.
"area"	**B_LONG_TYPE**	The area of the window where the cursor is now located.
"buttons"	**B_LONG_TYPE**	Which mouse buttons, if any, are down.

The "area" constant records which part of the window the cursor is over. It will be one of the following constants:

B_CONTENT_AREA	The cursor is over the content area of the window.
B_CLOSE_AREA	The cursor is over the close button in the title tab.
B_ZOOM_AREA	The cursor is over the zoom button in the title tab.
B_TITLE_AREA	The cursor is inside the title tab, but not over either the close button or zoom button.
B_RESIZE_AREA	The cursor is over the area in the right bottom corner where the window can be resized.
B_MINIMIZE_AREA	< *Currently unused*. >
B_UNKNOWN_AREA	It's unknown where the cursor is, probably because it just left the window.

The "buttons" mask is formed from one or more of the following constants:

B_PRIMARY_MOUSE_BUTTON
B_SECONDARY_MOUSE_BUTTON
B_TERTIARY_MOUSE_BUTTON

If no buttons are down, the mask is 0.

B_MOUSE_UP

This message reports that the user released a mouse button. It's produced only for the last button the user releases—that is, only if no other mouse button remains down. The BWindow does not dispatch this message.

The message contains the following data entries:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the mouse button went up again, as measured in microseconds from the time the machine was last booted.

"where"	**B_POINT_TYPE**	Where the cursor was located when the user released the mouse button, expressed in the coordinate system of the target BView—the view where the cursor was located when the button went up.
"modifiers"	**B_LONG_TYPE**	A mask that identifies which of the modifier keys were down and which keyboard locks were in effect when the user released the mouse button.

The "modifiers" mask is the same as for key-down events and is described under "Modifier Keys" on page 51 in *The Interface Kit* chapter.

B_PANEL_CLOSED

This message is delivered to the BWindow when the application or the user closes the save panel associated with the window. The BWindow dispatches it by calling its own **SavePanelClosed()** function.

The message contains the following data entries:

Data name	Type code	Description
"frame"	**B_RECT_TYPE**	The frame rectangle of the save panel at the time the panel was closed. (The user may have resized it and relocated it on-screen before it was closed.) The rectangle is specified in the screen coordinate system.
"directory"	**B_REF_TYPE**	A **record_ref** reference to the last directory displayed in the panel.
"canceled"	**B_BOOL_TYPE**	An indication of whether or not the panel was closed by user. It's **TRUE** if the user closed the panel by operating the "Cancel" button and **FALSE** otherwise.

B_PULSE

This message serves as a simple timing mechanism. It's posted at regularly spaced intervals and is dispatched by calling the **Pulse()** function of every BView that wants to participate.

The message typically lacks any data entries, but may contain this one:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the event occurred, as measured in microseconds from the time the machine was last booted.

B_QUIT_REQUESTED

This message is interpreted by a BWindow object as a request to close the window. It's dispatched by calling **QuitRequested()**, which is generally implemented by application classes derived from BWindow.

When the Application Server produces the message (for example, when the user clicks the window's close button), it adds the following data entry:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the event occurred, as measured in microseconds from the time the machine was last booted.

However, this information is not crucial to the interpretation of the event. You don't need to add it to **B_QUIT_REQUESTED** messages that are posted in application code.

B_SAVE_REQUESTED

This message is delivered to a BWindow when the user operates the save panel to request that a document be saved. It has the following data entries:

Data name	Type code	Description
"directory"	**B_REF_TYPE**	A **record_ref** reference to the directory where the document should be saved.
"name"	**B_STRING_TYPE**	The name of the file in which the document should be saved.

These entries are added to all messages reporting save-requested events. Generally, the message has **B_SAVE_REQUESTED** as its **what** data member. However, you can define a custom message to report the event, one with another constant and additional data entries.

If the command constant is **B_SAVE_REQUESTED**, the message is dispatched by calling the BWindow's **SaveRequested()** function; otherwise, it's not treated as a system message. See **RunSavePanel()** in the BWindow class of the Interface Kit.

B_SCREEN_CHANGED

This message reports that the screen configuration has changed. The BWindow dispatches it by calling its own **ScreenChanged()** function.

The message contains these data entries:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the screen changed, as measured in microseconds from the time the machine was last booted.
"frame"	**B_RECT_TYPE**	A rectangle with the same dimensions as the pixel grid the screen displays.
"mode"	**B_LONG_TYPE**	The color space of the screen—currently **B_COLOR_8_BIT** or **B_RGB_32_BIT**.

B_VALUE_CHANGED

This message reports that the Application Server changed a value associated with a scroll bar—something that will happen repeatedly as the user drags the scroll knob and presses the scroll buttons. The BWindow dispatches it by calling the BScrollBar object's **ValueChanged()** function.

The message has these data entries:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the value changed, as measured in microseconds from the time the machine was last booted.
"value"	**B_LONG_TYPE**	The new value of the object.

B_VIEW_MOVED

This message reports that a view moved within its parent's coordinate system. Repeated messages may be produced if the movement is caused by the user resizing the window, which in turn resizes the parent view. The BWindow dispatches each one by calling its **FrameMoved()** function.

The message contains the following data:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the view moved, as measured in microseconds from the time the machine was last booted.

"where"	**B_POINT_TYPE**	The new location of the left top corner of the view's frame rectangle, expressed in the coordinate system of its parent.

B_VIEW_RESIZED

This message reports that a view has been resized. Repeated messages are produced if the resizing is an automatic consequence of the window being resized. The BWindow dispatches each one by calling its **FrameResized()** function.

The message holds the following data.

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the view was resized, as measured in microseconds from the time the machine was last booted.
"width"	**B_LONG_TYPE**	The new width of the view's frame rectangle.
"height"	**B_LONG_TYPE**	The new height of the view's frame rectangle.
"where"	**B_POINT_TYPE**	The new location of the left top corner of the view's frame rectangle, expressed in the coordinate system of its parent. (A "where" entry is present only if the view was moved while being resized.)

The message has a "where" entry only if resizing the view also served to move it. The new location of the view would first be reported in a **B_VIEW_MOVED** BMessage.

B_WINDOW_ACTIVATED

This message reports that the window has become the active window or has relinquished that status. The BWindow dispatches the message by calling its **WindowActivated()** function, which notifies every BView with a similar function call.

The message contains two data entries:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the window's status changed, as measured in microseconds from the time the machine was last booted.

"active"	**B_BOOL_TYPE**	A flag that records the new status of the window. It's **TRUE** if the window has become the active window, and **FALSE** if it is giving up that status.

B_WINDOW_MOVED

This message reports that the window has been moved in the screen coordinate system. Repeated messages are generated when the user drags a window. The BWindow dispatches each one by calling its **WindowMoved()** function.

The message has the following entries:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the window moved, as measured in microseconds from the time the machine was last booted.
"where"	**B_POINT_TYPE**	The new location of the left top corner of the window's content area, expressed in screen coordinates.

B_WINDOW_RESIZED

This message reports that the window has been resized. It's generated repeatedly as the user moves a window border. The BWindow dispatches each message by calling **WindowResized()**.

The message holds these data entries:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the window was resized, as measured in microseconds from the time the machine was last booted.
"width"	**B_LONG_TYPE**	The new width of the window's content area.
"height"	**B_LONG_TYPE**	The new height of the window's content area.

B_WORKSPACE_ACTIVATED

This message reports that the active workspace has changed. It's delivered to all BWindow objects associated with the workspace that was previously active and with the one just activated. Each BWindow dispatches the message by calling its own **WorkspaceActivated()** function.

The message contains the following data:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the workspace was activated or deactivated, as measured in microseconds from the time the machine was last booted.
"workspace"	**B_LONG_TYPE**	The workspace that's the subject of the message.
"active"	**B_BOOL_TYPE**	A flag that records the new status of the workspace—**TRUE** if it has become the active workspace, and **FALSE** if it has ceased being the active workspace.

B_WORKSPACES_CHANGED

This message informs a BWindow object that the set of workspaces with which it is associated has changed. The BWindow dispatches the message by calling its own **WorkspacesChanged()** function.

The message has three data entries:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the set of workspaces associated with the window changed, as measured in microseconds from the time the machine was last booted.
"old"	**B_LONG_TYPE**	The set of workspaces where the window could appear before the change.
"new"	**B_LONG_TYPE**	The set of workspaces where the window can appear after the change.

B_ZOOM

This message instructs the BWindow object to zoom the on-screen window to a larger size—or to return it to its normal size. The message is produced when the user operates the zoom button in the window's title tab. The BWindow dispatches it by calling **Zoom()**, declared in the BWindow class.

The message has just one data entry:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the zoom button was clicked, as measured in microseconds from the time the machine was last booted.

Standard Messages

The software kits produce a few standard messages that aren't system messages—that aren't matched to a specific hook function. They're classified below as:

- Messages that are sent as replies, sometimes automatically, to other messages, and
- Messages that convey editing instructions.

Reply Messages

The following three messages are sent as replies to other messages.

B_HANDLERS_INFO

The various **HandlersRequested()** functions implemented in the Application and Interface Kits send this message as a reply to a **B_HANDLERS_REQUESTED** system message, which requests BMessenger proxies for BHandler objects. The reply message will contain one of two possible data entries:

Data name	Type code	Description
"handlers"	**B_MESSENGER_TYPE**	An array of one or more BMessenger objects corresponding to the BHandlers specified in the **B_HANDLERS_REQUESTED** message.
"error"	**B_LONG_TYPE**	An error code explaining why there is no "handlers" array.

B_MESSAGE_NOT_UNDERSTOOD

This message doesn't contain any data entries. It's sent as a reply to messages that the receiving thread's chain of BHandlers does not recognize. See **MessageReceived()** in the BHandler class.

B_NO_REPLY

This message doesn't contain any data entries. It's sent as a default reply to another message when the original message is about to be deleted. The default reply is sent only if a synchronous reply is expected and none has been sent. See the **SendReply()** function in the BMessage class.

Editing Messages

A handful of messages pass editable data or give an instruction to edit currently selected data. Because BTextViews are the only kit-defined objects that know how to display editable data, they're the only ones who can respond to these messages.

B_CUT, B_COPY, and B_PASTE

A BWindow posts these messages to its focus view (or to itself, if none of its views is currently in focus) when the user presses the Command-*x*, Command-*c*, and Command-*v* shortcuts. It puts only one data entry in the message:

Data name	Type code	Description
"when"	**B_DOUBLE_TYPE**	When the user pressed the keyboard shortcut, as measured in microseconds from the time the machine was last booted.

BTextView objects respond to these messages. See the BTextView class in the Interface Kit for details.

B_SIMPLE_DATA

This message is a package for a single data element. It can theoretically contain any type of data, but only two entries are currently understood:

Data name	Type code	Description
"text"	**B_ASCII_TYPE**	A null-terminated string of characters.
"char"	**B_LONG_TYPE**	A single character.

A BTextView object can put this message together for a drag-and-drop operation, and can understand the message when it's dropped on or targeted to the view. When it produces the message, it puts the text that's currently selected into a "text" data entry, as described above. It understands the message with either a "text" or a "char" data entry; it inserts the characters at the current selection.

Interapplication Messages

The messages that a user drags and drops on a view might have their source in any application, including applications that come with the Be Operating System. Currently, the Browser is the only source for a published, public message. It will probably be a common source, since it permits users to drag representations of database records. The message in which the Browser packages the dragged information is identical to one that reports a refs-received event. It has a single entry named "refs" containing one or more **record_ref** (**B_REF_TYPE**) items and **B_REFS_RECEIVED** as the command constant. See "**B_REFS_RECEIVED**" above.