System Messages

The Device Kit − Table of Contents

System Messages

This chapter lists and lightly describes the messages that are defined by Be. The list includes messages that are sent from the system to your application, messages that your application can create and send to other applications (and servers), and messages that are defined for their formats, but that you don't literally *send* anywhere (such as a clipboard message format).

Most messages are listed by their command constants (**`B_ABOUT_REQUESTED`**, **`B_QUIT_REQUESTED`**, etc.). Where the command constant isn't specific enough, the message is listed by some other field. For example, every Node Monitor message has the command constant **`B_NODE_MONITOR`**, and is further distinguishedand listed belowby its "opcode" field value (**`B_ENTRY_REMOVED`**, **`B_STAT_CHANGED`**, etc.). If a message doesn't have a defined command constant or other descriptive field, it's identified prosaically ("Clipboard Data", for example).

The list presents the messages in alphabetical order (ignoring the leading "B_"). Each message entry looks something like this:

Purpose: Tells you whether the message is meant to be delivered ("Deliverable"), or if it's defined simply for its format ("Format"). A message can be both: It can be used as a deliverable by one part of the system, and used just for its format in another.

If the message is deliverable, it will describe the following:

Source: The sender of the message. An important point here is whether the message is sent by the system, by your app, or by both. If the "**Source:**" line doesn't mention that your app can send the message, then don't send it. Often the source is listed, vaguely, as "the system." This denotes that the message is sent by one of the basic servers (App Server or Input Server, mainly). Other sources (Node Monitor, Roster Monitor, MIME Monitor) are identified more precisely. In any case, the identity of the source is provided mainly so you know which ballpark you're inthe exact identity shouldn't matter to your code.

Target: The target of the message. This is the BHandler object that receives the message, and, if applicable, converts it to a hook function. For example, the target of a **`B_VIEW_MOVED`** message is the view's BWindow, not the BView itself (because the BView never actually sees the **`B_VIEW_MOVED`** messageit only sees the **`ViewMoved()`** hook function). The target is often either **`be_app`** or a BWindow object, but for some messages the target is declared by your app (the target of a BControl's invocation message, for example).

Hook: The hook function that the target invokes when the message is received. If this line is missing, the message doesn't map to a hook function.

This is followed by a brief description that tells you the circumstances under which the message is sent, and how your app is expected to behave when it receives the message. If the description doesn't say what your app is supposed to do, then it doesn't *have* to actively do anything, but it shouldn't interfere with the default mechanism.

After the description is a table that lists the message's Be−defined fields. If it doesn't have any fields, the table is excluded. If a field holds an array of data, the maximum size of the array, if known, is given in brackets after the field name:

```
"byte" [3]
```

In some cases, the array size is the value of some other field. Here, the size of the array in the "argv" field is given by the value of the "argc" field:

```
"argv" ["argc"]
```

General Messages

B_ABOUT_REQUESTED

Purpose: Deliverable

Source: The system or your app.

Target: App−defined; typically **be_app**.

Hook: **BApplication::AboutRequested()** if the target is **be_app**.

The message should be assigned to an **About...** menu item, such that the message is sent when the user clicks the item. Your application is expected to put up an "About This Application" panel when it receives this message.

B_ACQUIRE_OVERLAY_LOCK , B_RELEASE_OVERLAY_LOCK

Source: A graphics driver.

Target: The team owning the overlay.

B_ACQUIRE_OVERLAY_LOCK is sent by a graphics driver when an overlay is acquired. **B_RELEASE_OVERLAY_LOCK** is sent when the overlay is released.

B_APP_ACTIVATED

Purpose: Deliverable

Source: The system.

Target: **be_app**

Hook: **BApplication::AppActivated()**

Sent when an application becomes active or inactive.

"active"	**B_BOOL_TYPE**	**true** if the app has become active; otherwise **false**.

B_ARCHIVED_OBJECT

Purpose: Deliverable and format

Source: A dragged replicant, or your app.

Target: A (remote) application.

Hook: **BShelf::CanAcceptReplicantMessage()**

As a deliverable: The replicant system uses this message as a deliverable. If you're using BDragger and BShelf objects, the message is created and delivered for you. You can also simulate a dragged replicant by archiving a view, setting the archive message's command to **B_ARCHIVED_OBJECT**, and sending the message to a remote application. If the remote application has a BShelf object, the BShelf will pick up the message (through a BMessageFilter) and pass it to the hook function.

To create a simulated replicant message, you call **Archive()** on the view that you want to replicate, and add (at least) the "add_on" field to the archive message.

See BShelf and BDragger for more information about replicants.

As a format: **B_ARCHIVED_OBJECT** should be used as the command constant for all archive messages. When you archive an object, the "class" field is automatically added to the archive message. All other fields must be added by your archiving code. See the BArchivable class for more information about archiving.

"class" [ ]	**B_STRING_TYPE**	An array of class names that gives the class hierarchy of the archived object.
"add_on"	**B_STRING_TYPE**	The signature of the library or application that knows how to create the archived object.
"be:add_on_version"	**B_INT32_TYPE**	The version of the add_on.
"be:load_each_time"	**B_BOOL_TYPE**	**true**: The add_on is loaded each time the object is unarchived. **false**: The add_on is loaded only if it isn't already loaded.
"be:unload_on_delete"	**B_BOOL_TYPE**	Is the add_on unloaded when the unarchived object is deleted?

"shelf_type" (replicants only; optional)	**B_STRING_TYPE**	The "type" of shelf that you want to have display the replicant. A shelf's type is its name, as assigned when it's created.

B_ARGV_RECEIVED

Source: The system.

Target: **be_app**

Hook: **BApplication::ArgvReceived()**

Forwards arguments that **(a)** the user passes while launching the app from the command line, or **(b)** are passed to **BRoster::Launch()**. Most apps treat command line arguments as filenames that should be opened. If the filename is relative (if it doesn't start with "/"), you should append it to the "cwd" field to reconstruct the full path.

"argc"	**B_INT32_TYPE**	The number of arguments.
"argv" [*argc*]	**B_STRING_TYPE**	The arguments.
"cwd"	**B_STRING_TYPE**	The path name of the current working directory.

B_CANCEL

Purpose: Deliverable

Source: The Application Kit.

Target: Application Server.

Used to cancel an ongoing operation. The Application Kit sends this message to the Application Server to cancel a shutdown if a window refuses to quit.

B_CLIPBOARD_CHANGED

Purpose: Deliverable

Source: The Application Server.

Target: Selected BMessenger.

If you've called BClipboard::**StartWatching()** to monitor a clipboard for changes, this message is sent to the specified BMessenger when the clipboard changes.

B_CONTROL_INVOKED

Purpose: Deliverable

Source: A BControl.

Target: Selected BMessenger.

This message is sent to the targeted messenger when a BControl−derived object is invoked.

B_INPUT_DEVICES_CHANGED

Purpose: Deliverable

Source: The Input Server.

Target: Add−on specified target.

Hook: **BInput::Control()**

This message is sent by the Input Server to send you notification when a device starts or stops, or when the set of registered devices changes. These messages are sent only if you've used the **watch_input_devices()** function to request such notification. The messages are sent to the target indicated in the function call.

"code"	**B_INT32_TYPE**	Operation code for the keyboard device control request being issued. One of: **B_INPUT_DEVICE_ADDED** **B_INPUT_DEVICE_STARTED**

		B_INPUT_DEVICE_STOPPED B_INPUT_DEVICE_REMOVED
"device"	B_STRING_TYPE	The name of the device. If this field doesn't exist, the device is identified by "type" instead.
"type"	B_INT32_TYPE	The device type. If this field doesn't exist, the "device" field identifies the device.
"message"	B_MESSAGE_TYPE	The control message. None of the BeOS standard messages use this field.

B_INPUT_METHOD_EVENT

Purpose: Deliverable

Source: The Input Server.

Target: Input method add−on's BLooper.

This message is sent by the Input Server to an input method add−on to let it know that a method event has occurred. The message's "be:opcode" field indicates which event has occurred.

- B_INPUT_METHOD_STARTED
- B_INPUT_METHOD_STOPPED
- B_INPUT_METHOD_CHANGED
- B_INPUT_METHOD_LOCATION_REQUEST

These are discussed in more detail in the Input Server chapter.

B_KEY_DOWN , B_KEY_UP , B_UNMAPPED_KEY_DOWN , B_UNMAPPED_KEY_UP

Source: The system.

Target: The focus view's BWindow.

Hook: `BView::KeyDown()` (B_KEY_DOWN)
`BView::KeyUp()` (B_KEY_UP)
(The **`...UNMAPPED...`** messages don't map to hook functions.)

B_KEY_DOWN is sent when the user presses (or holds down) a key that's mapped to a character; B_KEY_UP is sent when the user releases the key. B_UNMAPPED_KEY_DOWN and B_UNMAPPED_KEY_UP are sent if the key isn't mapped to a character. This doesn't include modifier keys, which are reported in the B_MODIFIERS_CHANGED message.

"when"	B_INT64_TYPE	Event time, in microseconds since 01/01/70
"key"	B_INT32_TYPE	The code for the physical key that was pressed. See <x> for the key map.
"be:key_repeat" (B_KEY_DOWN only)	B_INT32_TYPE	The "iteration number" of this key down. When the user holds the key down, successive messages are sent with increasing key repeat values. This field isn't present in the initial event; the first repeat message (i.e., the second key down message) has a key repeat value of 1.
"modifiers"	B_INT32_TYPE	The modifier keys that were in effect at the time of the event. See <x> for a list of values.
"states"	B_UINT8_TYPE	The state of all keys at the time of the event. See <x>.
"byte" [3] (B_KEY_DOWN and B_KEY_UP only)	B_INT8_TYPE	The UTF8 data that's generated
"bytes" (B_KEY_DOWN and B_KEY_UP only)	B_STRING_TYPE	The string that's generated. (The string usually contains a single character.)
"raw_char" (B_KEY_DOWN and B_KEY_UP only)	B_INT32_TYPE	Modifier−independent ASCII code for the character.

B_KEY_UP see B_KEY_DOWN

B_KEYBOARD_DEVICE

Purpose: Deliverable

Source: The input server.

Target: A keyboard device add−on.

Hook: `BInput::Control()`

This message is used by the input server to send a request to a keyboard device add−on.

"code"	B_INT32_TYPE	Operation code for the keyboard device control request being issued. One of: B_KEY_MAP_CHANGED B_KEY_LOCKS_CHANGED B_KEY_REPEAT_DELAY_CHANGED B_KEY_REPEAT_RATE_CHANGED
"device"	B_STRING_TYPE	The name of the device. If this field doesn't exist, the device is identified by "type" instead.
"type"	B_INT32_TYPE	The device type. If this field doesn't exist, the "device" field identifies the device.
"message"	B_MESSAGE_TYPE	The control message for the request. None of the BeOS standard messages use this field.

B_UNMAPPED_KEY_UP see B_KEY_DOWN

B_MEDIA_BUFFER_CREATED , B_MEDIA_BUFFER_DELETED

Purpose: Deliverable

Source: The media server.

Target: Target specified to `BMediaRoster::StartWatching()`.

Sent to indicate that a media buffer has been created or deleted.

B_MEDIA_BUFFER_CREATED's message has the following fields:

"clone_info"	B_RAW_TYPE	An area_info structure describing the buffer's location in memory. This is an array, one entry per buffer created.

B_MEDIA_BUFFER_DELETED looks like this:

"media_buffer_id"	B_INT32_TYPE	The buffer ID number of the buffer being deleted. This is an array, one entry per buffer deleted.

B_MEDIA_BUFFER_DELETED see B_MEDIA_BUFFER_CREATED

B_MEDIA_CONNECTION_BROKEN see B_MEDIA_CONNECTION_MADE

B_MEDIA_CONNECTION_MADE , B_MEDIA_CONNECTION_BROKEN

Purpose: Deliverable

Source: The media server.

Target: Target specified to `BMediaRoster::StartWatching()`.

Sent to indicate that a connection between media nodes has been made or broken.

B_MEDIA_FLAVORS_CHANGED

Purpose: Deliverable

Source: The media server.

Target: Target specified to `BMediaRoster::StartWatching()`.

Sent by the media server to indicate that the flavors supported by a particular add−on have changed.

"be:addon_id"	B_INT32_TYPE	The add−on ID of the add−on whose flavors have changed
"be:new_count"	B_INT32_TYPE	How many new flavors have been added.
"be:gone_count"	B_INT32_TYPE	How many flavors have been removed.

B_MEDIA_FORMAT_CHANGED

Purpose: Deliverable

Source: The media server.

Target: Target specified to BMediaRoster::StartWatching().

Sent by the media server to indicate that the format used on a particular connection has changed.

"be:source"	B_RAW_TYPE	A media_source structure describing the source of the connection whose format changed.
"be:destination"	B_RAW_TYPE	A media_source structure describing the source of the connection whose format changed.
"be:format"	B_RAW_TYPE	A media_format structure describing the new format.

B_MEDIA_NODE_CREATED , B_MEDIA_NODE_DELETED

Purpose: Deliverable

Source: The media server.

Target: Target specified to BMediaRoster::StartWatching().

Sent to indicate that a node has been created or deleted in the media system.

"media_node_id"	B_INT32_TYPE	The ID of the media_node that was created or deleted.

B_MEDIA_NODE_DELETED see B_MEDIA_NODE_CREATED

B_MEDIA_NODE_STOPPED

Purpose: Deliverable

Source: The media server.

Target: Target specified to BMediaRoster::StartWatching().

Indicates that a media node has stopped.

B_MEDIA_PARAMETER_CHANGED , B_MEDIA_NEW_PARAMETER_VALUE

Purpose: Deliverable

Source: The media server.

Target: Target specified to BMediaRoster::StartWatching().

Sent by the media server to indicate that the value of a parameter has changed. B_MEDIA_PARAMETER_CHANGED only tells you which parameter changed (media_node and parameter ID). B_MEDIA_NEW_PARAMTER_VALUE also tells you which parameter changed, and what the new value is.

The B_MEDIA_PARAMETER_CHANGED message looks like this:

"be:node"	B_RAW_TYPE	A media_node structure indicating which node's parameter web has changed.
"be:parameter"	B_INT32_TYPE	The ID number of the parameter whose value has changed.

The B_MEDIA_NEW_PARAMETER_VALUE message is:

"node"	**B_RAW_TYPE**	A media_node structure indicating which node is reporting a changed parameter value.
"parameter"	**B_INT32_TYPE**	The parameter ID of the changed parameter.
"when"	**B_INT64_TYPE**	The performance time at which the change took effect.
"value"	**B_RAW_TYPE**	The parameter's new value.

B_MEDIA_WEB_CHANGED

Purpose: Deliverable

Source: The media server.

Target: Target specified to **BMediaRoster::StartWatching()**.

Sent by the media server to indicate that a particular node's BParameterWeb has.

"node"	**B_RAW_TYPE**	A media_node structure indicating which node's parameter web has changed.

B_MEDIA_WILDCARD

Purpose: Constant only.

Source: Your application.

Target: The media server.

When calling **BMediaRoster::StartWatching()** or **BMediaRoster::StopWatching()**, you can use this constant to match all media messages. This message has no actual format.

(No Be−defined fields)

B_MINIMIZE

Source: The system or your app.

Target: The BWindow that's hidden/unhidden.

Hook: **BWindow::Minimize()**

Sent when the user double−clicks a window's title bar (to hide the window), or selects a window from the **DeskBar**'s window list (to unhide the window).

"when"	**B_INT64_TYPE**	Event time, in microseconds since 01/01/70.
"minimize"	**B_BOOL_TYPE**	**true** if the window is being hidden; **false** if it's being unhidden.

B_MODIFIERS_CHANGED

Source: The system.

Target: The focus view's window.

Sent when the user presses or releases a modifier key.

"when"	**B_INT64_TYPE**	Event time, in microseconds since 01/01/70
"modifiers"	**B_INT32_TYPE**	The current modifier keys. See <x>
"be:old_modifiers"	**B_INT32_TYPE**	The previous modifier keys.
"states"	**B_UINT8_TYPE**	The state of all keys at the time of the event. See <x>.

B_MOUSE_DOWN

Source: The system.

Target: The BWindow of the view the mouse is pointing to.

Hook: `BView::MouseDown()`

Sent when the user presses a mouse button. This message is only sent if no other mouse button is already down.

"when"	`B_INT64_TYPE`	Event time, in microseconds since 01/01/70
"where"	`B_POINT_TYPE`	Mouse location in the view's coordinate system.
"modifiers"	`B_INT32_TYPE`	The modifier keys that were in effect at the time of theevent.

"buttons"	`B_INT32_TYPE`	The mouse button that was pressed, one of:
		`B_PRIMARY_MOUSE_BUTTON`
		`B_SECONDARY_MOUSE_BUTTON`
		`B_TERTIARY_MOUSE_BUTTON`

"clicks"	`B_INT32_TYPE`	1 for a single−click, 2 for double−click, 3 for triple−click, and so on. The counter is reset if the time between clicks exceeds the "Double−click speed" set by the user in the **Mouse** preferences. Note that the counter is *not* reset if the mouse moves between clicks.

B_MOUSE_MOVED

Source: The system.

Target: The BWindow of the view the mouse is pointing to.

Hook: `BView::MouseMoved()`

Sent when the user moves the mouse.

"when"	`B_INT64_TYPE`	Event time, in microseconds since 01/01/70
"where"	`B_POINT_TYPE`	The mouse's new location in window coordinates.

"buttons"	`B_INT32_TYPE`	The mouse buttons that are down. Zero or more of:
		`B_PRIMARY_MOUSE_BUTTON`
		`B_SECONDARY_MOUSE_BUTTON`
		`B_TERTIARY_MOUSE_BUTTON`

B_MOUSE_UP

Source: The system.

Target: The BWindow of the view the mouse is pointing to.

Hook: `BView::MouseUp()`

Sent when the user releases a mouse button. It's only sent if no other mouse button remains down.

"when"	`B_INT64_TYPE`	Event time, in microseconds since 01/01/70
"where"	`B_POINT_TYPE`	Mouse location in the view's coordinate system.
"modifiers"	`B_INT32_TYPE`	The modifier keys that were in effect at the time of the event. See <x> for a list of modifier values.

B_MOUSE_WHEEL_CHANGED

Source: The system.

Target: The BWindow of the view the mouse is pointing to.

Sent when the user moves the mouse wheel (on mice that have them).

"when"	**B_INT64_TYPE**	Event time, in microseconds since 01/01/70
"be:wheel_delta_x"	**B_FLOAT_TYPE**	How much the Y value of the wheel has changed.
"be:wheel_delta_y"	**B_FLOAT_TYPE**	How much the Y value of the wheel has changed.

The standard mouse driver that comes with BeOS only supports a Y−oriented wheel.

B_NODE_MONITOR

Source: The Node Monitor.

Target: App−defined.

Sent when a monitored file changes. All Node Monitor messages contain an **int32** "opcode" field that tells you what happened (a file was removed, an attribute changed, etc.). The format of the rest of the message depends on the opcode. The formats are given under individual entries in this appendix, listed by opcode value:

- **B_ENTRY_CREATED**
- **B_ENTRY_REMOVED**
- **B_ENTRY_MOVED**
- **B_STAT_CHANGED**
- **B_ATTR_CHANGED**
- **B_DEVICE_MOUNTED**
- **B_DEVICE_UNMOUNTED**

These are discussed in the section "Node Monitor Messages".

B_OBSERVER_NOTICE_CHANGE

Source: The system.

Target: Application−defined target.

Sent by BHandlers to all targets that have been registered with the BHandler's **StartWatching()** or **StartWatchingAll()** function.

"be:observe_change_what"	**B_INT32_TYPE**	The "what" code of the message being broadcast.
"be:observe_orig_what"	**B_INT32_TYPE**	The original "what" code of the message, before it was altered by the broadcasting mechanism.

The message may have other fields, depending on what the broadcast message is. Messages are sent to targets in response to the **BHandler::SendNotices()** function. The logic is:

- Take the message to broadcast and place its what code into the "be:observe_change_orig_what" field.
- Add the "be:observe_change_what" field, which is set to the what code specified by the call to **SendNotices()**.
- Set the message's what code to **B_OBSERVER_NOTICE_CHANGE**.

The resulting message is then sent to all interested parties.

See also: BHandler::SendNotices(), BHandler::StartWatching()

B_OPEN_IN_WORKSPACE

Source: The system.

Target: BApplication.

Sent to an application when it's first launched to tell it to open in a specific workspace. The message will be handled during the construction of the BApplication object.

"be:workspace"	**B_INT32_TYPE**	Workspace number into which the application should open.

B_POINTING_DEVICE

Purpose: Deliverable

Source: The input server.

Target: A pointing device add−on.

Hook: **`BInput::Control()`**

This message is used by the input server to send a request to a pointing device add−on.

"code"	**`B_INT32_TYPE`**	Operation code for the pointing device control request being issued. One of: **`B_MOUSE_TYPE_CHANGED`** **`B_MOUSE_MAP_CHANGED`** **`B_MOUSE_SPEED_CHANGED`** **`B_CLICK_SPEED_CHANGED`** **`B_MOUSE_ACCELERATION_CHANGED`**
"device"	**`B_STRING_TYPE`**	The name of the device. If this field doesn't exist, the device is identified by "type" instead.
"type"	**`B_INT32_TYPE`**	The device type. If this field doesn't exist, the "device" field identifies the device.
"message"	**`B_MESSAGE_TYPE`**	The control message for the request. None of the BeOS standard messages use this field.

B_PRINTER_CHANGED

Source: The Print Server.

Target: Everyone.

Sent whenever the user changes printers. Applications that support printing should watch for this message, and if they receive it, they should suggest that the user check their page setup the next time they choose to print.

B_PULSE

Source: The system.

Target: **`be_app`** or BWindow object.

Hook: **`BApplication::Pulse()`** and **`BView::Pulse()`**

Sent at regular intervals, but with no particular intent. You can implement **`Pulse()`** to do whatever you want. The message is to the BWindow only if a BView within the window declares **`B_PULSE_NEEDED`** in its constructor flags.

B_QUERY_UPDATE

Source: The system.

Target: App−defined.

Sent when the results of a live query change: If a new file meets the query requirements, **`B_ENTRY_CREATED`** is sent. If a file that previously passed the query requirements no longer does, **`B_ENTRY_REMOVED`** is sent.

B_QUIT_REQUESTED

Source: The system or your app.

Target: **`be_app`**, BWindow closed by the user, or other BLooper object.

Hook: **`BLooper::QuitRequested()`**.

Automatically sent **(a)** to **`be_app`** when the user types **Command**+**q**, and **(b)** to a BWindow when the user clicks the window's close box. Applications can also manufacture and send the message themselves. A looper that receives this message is expected to quit, or at least consider quitting.

"shortcut"	**`B_BOOL_TYPE`**	true if the user typed **Command**+**q**.

B_READY_TO_RUN

Source: The system.

Target: **be_app**

Hook: **BApplication::ReadyToRun()**

Sent when an application has finished configuring itself and is ready to start running.

(No Be−defined fields)

B_REFS_RECEIVED

Source: The system or your app.

Target: **be_app**, or other app−defined target.

Hook: **BApplication::RefsReceived()**

Automatically sent to **be_app** when **(a)** the user double−clicks a file that has a type that's supported by the app, and **(b)** when the user confirms some files (or directories) in an **Open File** panel (the target is **be_app** by default; it can be changed in **BFilePanel::SetTarget()**). You can also create, stuff, and send a **B_REFS_RECEIVED** message yourself. When it receives this message, an app is expected to open the files that the message refers to.

"refs" [*i*]	**B_REF_TYPE**	**entry_ref** items, one for each file or directory.

A BStatusBar object can be controlled synchronously by calling its **Reset()** and **Update()** functions. It can also be controlled asynchronously by sending it messages corresponding to the two functions; the object calls the function when it receives the message. Each message contains fields for the arguments passed to the function.

B_RELEASE_OVERLAY_LOCK see B_ACQUIRE_OVERLAY_LOCK

B_RESET_STATUS_BAR

Source: Your app.

Hook: **BStatusBar::Reset()**

Target: The BStatusBar you're resetting.

You construct and send this message to a BStatusBar object to tell it to (asynchronously) reset itself. The message also lets you reset the object's labels. To send the message, invoke BWindow's **PostMessage()** naming the target BStatusBar as the handler:

```
statusBar->Window()->PostMessage(theMessage, statusBar);
```

"label"	**B_STRING_TYPE**	The object's new label (**NULL**−terminated).
"trailing_label"	**B_STRING_TYPE**	The object's new trailing label (**NULL**−terminated)..

B_SCREEN_CHANGED

Source: The system.

Target: Every BWindow in the screen that changed (even hidden windows).

Hook: **BWindow::ScreenChanged()**

Sent when the screen's dimensions or color space changesbecause the user played with the **Screen** preferences app, for example.

"when"	**B_INT64_TYPE**	Event time, in microseconds since 01/01/70
"frame"	**B_RECT_TYPE**	The screen's dimensions.
"mode"	**B_INT32_TYPE**	The screen's color space: **B_CMAP8**, **B_RGB15**, **B_RGB15**, or **BRGB32**.

B_SILENT_RELAUNCH

Source: The system.

Target: **be_app**

Sent to a single−launch application when it's activated by being launched (for example, if the user double−clicks its icon in Tracker).

(No Be−defined fields)

B_SOME_APP_ACTIVATED , B_SOME_APP_LAUNCHED , B_SOME_APP_QUIT

Source: The Roster Monitor.

Target: App−defined.

Sent as apps are launched, activated, or quit. You get these messages by invoking **BRoster::StartWatching()** passing a one or more of **B_REQUEST_ACTIVATED**, **B_REQUEST_LAUNCHED**, and **B_REQUEST_QUIT**.

"mime_sig"	**B_STRING_TYPE**	The ap signature.
"team"	**B_INT32_TYPE**	The app's team id.
"thread"	**B_INT32_TYPE**	The id of the app's main thread.
"flags"	**B_INT32_TYPE**	The app's app flags (**B_SINGLE_LAUNCH**, **B_BACKGROUND_APP**, etc).
"ref"	**B_REF_TYPE**	The **entry_ref** of the app's executable.

B_UPDATE_STATUS_BAR

Source: Your app.

Hook: **BStatusBar::Update()**

Target: The BStatusBar you're updating.

You construct and send this message to a BStatusBar object to tell it to (asynchronously) update its progress. To send the message, invoke BWindow's **PostMessage()** naming the target BStatusBar as the handler:

```
statusBar->Window()->PostMessage(theMessage, statusBar);
```

"delta"	**B_FLOAT_TYPE**	An increment to the object's current value.
"text"	**B_STRING_TYPE**	The object's new text (**NULL**−terminated).
"trailing_text"	**B_STRING_TYPE**	The object's new trailing text (**NULL**−terminated)..

B_VALUE_CHANGED

Source: The system.

Target: The manipulated scrollbar's BWindow.

Hook: **BScrollBar::ValueChanged()**

Sent when the user plays with a scrollbar.

"when"	**B_INT64_TYPE**	Event time, in microseconds since 01/01/70
"value"	**B_INT32_TYPE**	The scrollbar's new value.

B_VIEW_MOVED

Source: The system.

Target: The moved view's BWindow.

Hook: **BView::FrameMoved()**

Sent when a view's origin (left top corner) changes relative to the origin of its parent. The message isn't sent if the view doesn't have the **B_FRAME_EVENTS** flag set.

"when"	**B_INT64_TYPE**	Event time, in microseconds since 01/01/70

"where"	**B_POINT_TYPE**	The view's new origin in the coordinate system of its parent.

B_VIEW_RESIZED

Source: The system.

Target: The resized view's BWindow.

Hook: `BView::FrameResized()`

Sent when the size of the view's frame changes. The message isn't sent if the view doesn't have the **B_FRAME_EVENTS** flag set.

"when"	**B_INT64_TYPE**	Event time, in microseconds since 01/01/70
"width"	**B_INT32_TYPE**	The view's new width.
"height"	**B_INT32_TYPE**	The view's new height.
"where"	**B_POINT_TYPE**	The view's new origin expressed in the coordinate system of its parent. This field is only included if the view actually moved while being resized, and can always be ignored: If the view *did* move, you'll hear about it in a separate **B_VIEW_MOVED** BMessage.

B_WINDOW_ACTIVATED

Source: The system.

Target: The activated/deactivated BWindow.

Hook: **`BWindow::WindowActivated()`** and **`BView::WindowActivated()`**

Sent just after a window is activated or deactivated. Note that the BWindow invokes **`WindowActivated()`** on each of its views.

"when"	**B_INT64_TYPE**	Event time, in microseconds since 01/01/70
"active"	**B_BOOL_TYPE**	**`true`** if the window is now active; **`false`** if not.

B_WINDOW_MOVE_BY

Purpose: Deliverable.

Source: Your application.

Target: The BWindow to be moved.

You can send this message to a window to resize it by the specified deltas.

"data"	**B_POINT_TYPE**	The amount by which to move the window's X and Y coordinates.

B_WINDOW_MOVE_TO

Purpose: Deliverable.

Source: Your application.

Target: The BWindow to be moved.

You can send this message to a window to resize it to the specified size.

"data"	**B_POINT_TYPE**	The width and height (in X and Y) to resize the window to.

B_WINDOW_MOVED

Source: The system.

Target: The BWindow that moved.

Hook: `BWindow::FrameMoved()`

Sent when a window's origin (left top corner) changes within the screen coordinate system.

"when"	`B_INT64_TYPE`	Event time, in microseconds since 01/01/70
"where"	`B_POINT_TYPE`	The window's new origin in screen coordinates.

B_WINDOW_RESIZED

Source: The system.

Target: The resized BWindow.

Hook: `BWindow::FrameResized()`

Sent when the size of the window's frame changes. Note that the "width" and "height" fields measure the window's content areathey don't include the window border or window tab.

"when"	`B_INT64_TYPE`	Event time, in microseconds since 01/01/70
"width"	`B_INT32_TYPE`	The width of the window's content area.
"height"	`B_INT32_TYPE`	The height of the window's content area.

B_WORKSPACE_ACTIVATED

Source: The system.

Target: Every BWindow in the activated and deactivated workspaces.

Hook: `BWindow::WorkspaceActivated()`

Sent when the active workspace changes.

"when"	`B_INT64_TYPE`	Event time, in microseconds since 01/01/70
"workspace"	`B_INT32_TYPE`	The index of the window's workspace.
"active"	`B_BOOL_TYPE`	`true` if the workspace is now active; `false` if not.

B_WORKSPACES_CHANGED

Source: The system.

Target: The BWindow whose set of workspaces changed.

Hook: `BWindow::WorkspacesChanged()`

Sent when there's a change to the set of workspaces in which a window can appear.

"when"	`B_INT64_TYPE`	Event time, in microseconds since 01/01/70
"old"	`B_INT32_TYPE`	The window's old workspace set, given as a vector of workspace indices.
"new"	`B_INT32_TYPE`	The window's new workspace set, given as a vector of workspace indices.

B_ZOOM

Source: The system.

Target: The BWindow that was zoomed.

Hook: `BWindow::Zoom()`

Sent when the user clicks a window's zoom button.

The message has just one data field:

"when"	**B_INT64_TYPE**	Event time, in microseconds since 01/01/70

Data Containers

A few constants identify messages as data containers. The system currently uses these constants to mark the containers it constructs for drag−and−drop operations.

B_MIME_DATA

This message constant indicates that all the data in the message is identified by MIME type names. The type code of every data field is **B_MIME_TYPE** and the name of each field is the MIME type string.

As an example, a BTextView object puts together a **B_MIME_DATA** message for drag−and−drop operations. The message has the text itself in a field named "text/plain"; the **text_run_array** structure that describes the character formats of the text is in a field named "application/x−vnd.Be−text_run_array".

B_SIMPLE_DATA

This message is a package for a single data element. If there are multiple data fields in the message, they present the same data in various formats.

For example, when the user drags selected files and directories from a Tracker window, the Tracker packages **entry_ref** references to them in a **B_SIMPLE_DATA** message. The references are in a "refs" array with a type code of **B_REF_TYPE**. In other words, the message has the same structure as a **B_REFS_RECEIVED** message, but a different **what** constant.

File Panel Messages

The file panel produces three messages: **`B_REFS_RECEIVED`**, **`B_SAVE_REQUESTED`**, and **`B_CANCEL`**. The first of these was discussed under "Application Messages" above. It's produced when the user picks files to open from the panel. The other two messages are described below.

B_SAVE_REQUESTED

The file panel produces this message when the user asks the application to save a document. It has two data fields:

"directory"	**`B_REF_TYPE`**	An **`entry_ref`** referring to the directory where the document should be saved.
"name"	**`B_STRING_TYPE`**	The file name under which the document should be saved.

B_CANCEL

A cancel notification is sent *whenever* a file panel is hidden. This includes the Cancel button being clicked, the panel being closed, and the panel being hidden after an open or a save.

"old_what"	**`B_INT32_TYPE`**	The "previous" **`what`** value. This is only useful (and dependable) if you supplied the BFilePanel with your own BMessage: The **`what`** from your message is moved to the "old_what" field. If you didn't supply a BMessage, you should ignore this field (it could contain garbage).
"source"	**`B_POINTER_TYPE`**	A pointer to the BFilePanel object.

See the BFilePanel class in **The Storage Kit** chapter for more information.

Node Monitor Messages

The Node Monitor is a mechanism that lets you watch for changes to a particular file or directory (or "node"). You ask the Node Monitor to start or stop watching a given node by calling the **watch_node()** function; you can stop all monitoring through the **stop_watching()** function.

When you call **watch_node()**, you tell the Node Monitor which aspects of the node you want to trackchanges to its name, to its size, its attributes, and so on. Each of these "trackable" elements corresponds to a particular type of message (identified by the message's "opcode" field) that's sent back to your application when that element actually changes (when the file is renamed, changes size, gains an attribute, and so on).

B_ENTRY_CREATED

"opcode"	**B_INT32_TYPE**	**B_ENTRY_CREATED** indicates that a new entry was created.
"name"	**B_STRING_TYPE**	The name of the new entry.
"directory"	**B_INT64_TYPE**	The **ino_t** (node) number for the directory in which the entry was created.
"device"	**B_INT32_TYPE**	The **dev_t** number of the device on which the new entry resides.
"node"	**B_INT64_TYPE**	The **ino_t** number of the new entry itself. (More accurately, it identifies the node that corresponds to the entry.)

B_ENTRY_REMOVED

"opcode"	**B_INT32_TYPE**	**B_ENTRY_REMOVED** indicates that an entry was removed.
"directory"	**B_INT64_TYPE**	The **ino_t** (node) number of the directory from which the entry was removed.
"device"	**B_INT32_TYPE**	The **dev_t** number of the device that the removed node used to live on.
"node"	**B_INT64_TYPE**	The **ino_t** number of the node that was removed.

B_ENTRY_MOVED

"opcode"	**B_INT32_TYPE**	**B_ENTRY_MOVED** indicates that an existing entry moved from one directory to another.
"name"	**B_STRING_TYPE**	The name of the entry that moved.
"from directory"	**B_INT64_TYPE**	The **ino_t** (node) number of the directory from that the node was removed from.
"to directory"	**B_INT64_TYPE**	The **ino_t** (node) number of the directory that the node was added to.
"device"	**B_INT32_TYPE**	The **dev_t** number of the device that the moved node entry lives on. (You can't move a file between devices, so this value will be apply to the file's old and new locations.)
"node"	**B_INT64_TYPE**	The **ino_t** number of the node that was removed.

B_STAT_CHANGED

"opcode"	**B_INT32_TYPE**	**B_STAT_CHANGED** indicates that some statistic of a node (as recorded in its **stat** structure) changed.
"node"	**B_INT64_TYPE**	The **ino_t** number of the node.
"device"	**B_INT32_TYPE**	The **dev_t** number of the node's device.

B_ATTR_CHANGED

"opcode"	**B_INT32_TYPE**	**B_ATTR_CHANGED** indicates that some attribute of a node changed.
"node"	**B_INT64_TYPE**	The **ino_t** number of the node.
"device"	**B_INT32_TYPE**	The **dev_t** number of the node's device.

B_DEVICE_MOUNTED

"opcode"	**B_INT32_TYPE**	**B_DEVICE_MOUNTED** indicates that a new device (or file system volume) has been mounted.
"new device"	**B_INT32_TYPE**	The **dev_t** number of the newly−mounted device.
"device"	**B_INT32_TYPE**	The **dev_t** number of the device that holds the directory of the new device's mount point.
"directory"	**B_INT64_TYPE**	The **ino_t** (node) number of the directory that acts as the new device's mount point.

B_DEVICE_UNMOUNTED

"opcode"	**B_INT32_TYPE**	**B_DEVICE_UNMOUNTED** indicates that a device has been unmounted.
"new device"	**B_INT32_TYPE**	The **dev_t** number of the unmounted device.

Reply Messages

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

B_MESSAGE_NOT_UNDERSTOOD

This message doesn't contain any data. The system sends it as a reply to a message that the receiving thread's chain of BHandlers does not recognize. See **`MessageReceived()`** and **`ResolveSpecifier()`** in the BHandler class of the Application Kit.

B_NO_REPLY

This message doesn't contain any data. 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 of the Application Kit.

B_REPLY

This constant identifies a message as being a reply to a previous message. The data in the reply depends on the circumstances and, particularly, on the original message. For replies to scripting messages, it generally has a "result" field with requested data and an "error" field with an error code reporting the success or failure of the scripted request.

Scripting Messages

The scripting system defines four generic messages that can operate on the specific properties of an object and two meta−messages that query an object about the messages it can handle. See "Scripting" in **The Application Kit** chapter for a full explanation.

B_COUNT_PROPERTIES

This message requests the number of properties supported by the receiver. It contains no data, but the reply message should contain one field:

"result"	**`B_INT32_TYPE`**	The number of properties supported.

B_GET_SUPPORTED_SUITES

This message requests the names of all message suites that the receiver supports. It doesn't contain any data, but the message that's sent in reply has one field:

"suites"	**`B_STRING_TYPE`**	An array of suite names.

A suite is a named set of messages and specifiers. A BHandler supports the suite if it can respond to the messages and resolve the specifiers.

B_SET_PROPERTY , B_GET_PROPERTY , B_CREATE_PROPERTY , B_DELETE_PROPERTY

These messagesas their names statetarget a particular property under the control of the target handler. They have the following data fields:

"specifiers"	**`B_MESSAGE_TYPE`**	An array of one or more BMessages that specify the targeted property. See **`AddSpecifier()`** in the BMessage class of the Application Kit for details on the contents of a specifier.
"data"	*variable*	For **`B_SET_PROPERTY`** messages only, the data that should be set. The data type depends on the targeted property.

A class can choose to respond to these messages, in any combination, for any set of self−declared properties.

Messages: Master Index

A

B_ACQUIRE_OVERLAY_LOCK	General Messages
B_APP_ACTIVATED	General Messages
B_ARCHIVED_OBJECT	General Messages
B_ARGV_RECEIVED	General Messages
B_ATTR_CHANGED	Node Monitor Messages

C

B_CANCEL	General Messages
B_CLIPBOARD_CHANGED	General Messages
B_CONTROL_INVOKED	General Messages
B_COUNT_PROPERTIES	anon
B_CREATE_PROPERTY	anon

D

Data Containers	Data Containers
B_DELETE_PROPERTY	anon
B_DEVICE_MOUNTED	Node Monitor Messages
B_DEVICE_UNMOUNTED	Node Monitor Messages

E

B_ENTRY_MOVED	Node Monitor Messages
B_ENTRY_REMOVED	Node Monitor Messages

F

File Panel Messages	File Panel Messages

G

General Messages	General Messages
B_GET_PROPERTY	anon

B_GET_SUPPORTED_SUITES	anon

I

B_INPUT_METHOD_EVENT	General Messages

K

B_KEY_UP	General Messages
B_KEYBOARD_DEVICE	General Messages

M

B_MEDIA_BUFFER_CREATED	General Messages
B_MEDIA_BUFFER_DELETED	General Messages
B_MEDIA_CONNECTION_BROKEN	General Messages
B_MEDIA_CONNECTION_MADE	General Messages
B_MEDIA_FLAVORS_CHANGED	General Messages
B_MEDIA_FORMAT_CHANGED	General Messages
B_MEDIA_NEW_PARAMETER_VALUE	General Messages
B_MEDIA_NODE_CREATED	General Messages
B_MEDIA_NODE_DELETED	General Messages
B_MEDIA_NODE_STOPPED	General Messages
B_MEDIA_PARAMETER_CHANGED	General Messages
B_MEDIA_WEB_CHANGED	General Messages
B_MEDIA_WILDCARD	General Messages
B_MESSAGE_NOT_UNDERSTOOD	Reply Messages
B_MIME_DATA	Data Containers
modifier keys	General Messages
B_MODIFIERS_CHANGED	General Messages
B_MOUSE_DOWN	General Messages
B_MOUSE_MOVED	General Messages
B_MOUSE_UP	General Messages
B_MOUSE_WHEEL_CHANGED	General Messages

N

Node Monitor Messages	Node Monitor Messages
Node Monitor Messages	Node Monitor Messages
B_NODE_MONITOR	General Messages

O

B_OPEN_IN_WORKSPACE	General Messages

P

B_POINTING_DEVICE	General Messages
B_PRINTER_CHANGED	General Messages

Q

B_QUIT_REQUESTED	General Messages

R

B_READY_TO_RUN	General Messages
B_REFS_RECEIVED	General Messages
B_RELEASE_OVERLAY_LOCK	General Messages
Reply Messages	Reply Messages
Reply Messages	Reply Messages
B_RESET_STATUS_BAR	General Messages

S

B_SCREEN_CHANGED	General Messages
Scripting Messages	anon
B_SET_PROPERTY	anon
B_SILENT_RELAUNCH	General Messages
B_SIMPLE_DATA	Data Containers
B_SOME_APP_ACTIVATED	General Messages
B_SOME_APP_LAUNCHED	General Messages

B_SOME_APP_QUIT	General Messages
B_STAT_CHANGED	Node Monitor Messages
System Messages	System Messages
System Messages	System Messages

U

B_UNMAPPED_KEY_UP	General Messages
B_UPDATE_STATUS_BAR	General Messages

V

B_VIEW_MOVED	General Messages
B_VIEW_RESIZED	General Messages

W

B_WINDOW_MOVE_BY	General Messages
B_WINDOW_MOVE_TO	General Messages
B_WINDOW_MOVED	General Messages
B_WINDOW_RESIZED	General Messages
B_WORKSPACE_ACTIVATED	General Messages
B_WORKSPACES_CHANGED	General Messages

Z