



Full credit is given to the above companies including the Operating System (OS) that this PDF file was generated!

Rocky Enterprise Linux 9.2 Manual Pages on command 'XtSetEventDispatcher.3'

\$ man XtSetEventDispatcher.3

XtInsertEventHandler(3) XT FUNCTIONS XtInsertEventHandler(3)

NAME

XtInsertEventHandler, XtRemoveEventHandler, XtRegisterExtensionSelector, XtSet? EventDispatcher, XtDispatchEventToWidget - extension event handling

SYNTAX

```
#include <X11/Intrinsic.h>

void XtInsertEventHandler(Widget widget, int event_type, XtPointer select_data,
    XtEventHandler proc, XtPointer client_data, XtListPosition position);

void XtRemoveEventHandler(Widget widget, int event_type, XtPointer select_data,
    XtEventHandler proc, XtPointer client_data);

void XtRegisterExtensionSelector(Display* display, int min_event_type, int max_event_type,
    XtExtensionSelectProc proc, XtPointer client_data);

XtEventDispatchProc XtSetEventDispatcher(Display* display, int event_type, XtEventDis?
    patchProc proc);

Boolean XtDispatchEventToWidget(Widget widget, XEvent* event);
```

ARGUMENTS

widget Specifies the widget for this event handler. Must be of class Core or any sub? class thereof.

event_type
 Specifies the event type.

select_data
 Specifies data used to select or deselect events from the server.

proc Specifies the proc.

client_data

Specifies additional data to be passed to the event handler.

position Specifies when the event handler is to be called relative to other previously registered handlers.

display Specifies the display.

min_event_type, max_event_type

Specifies the range of event types for this extension.

event Specifies a pointer to the event to be dispatched.

DESCRIPTION

The `XtInsertEventTypeHandler` function registers a procedure with the dispatch mechanism that is to be called when an event that matches the specified `event_type` is dispatched to the specified widget.

If `event_type` is one of the core X protocol events then `select_data` must be a pointer to a value of type `EventMask`, indicating the event mask to be used to select for the desired event. This event mask will be included in the value returned by `XtBuildEventMask`. If the widget is realized `XtInsertEventTypeHandler` calls `XSelectInput` if necessary. Specifying `NULL` for `select_data` is equivalent to specifying a pointer to an event mask containing 0. This is similar to the `XtInsertRawEventHandler` function.

If `event_type` specifies an extension event type then the semantics of the data pointed to by `select_data` are defined by the extension selector registered for the specified event type.

In either case the Intrinsics are not required to copy the data pointed to by `select_data`, so the caller must ensure that it remains valid as long as the event handler remains registered with this value of `select_data`.

The `position` argument allows the client to control the order of the invocation of event handlers registered for the same event type. If the client does not care about the order, it should normally specify `XtListTail`, which registers this event handler after any previously registered handlers for this event type.

The `XtRemoveEventTypeHandler` function unregisters an even handler registered with `XtInsertEventTypeHandler` for the specified event type. The request is ignored if `client_data` does not match the value given with the handler was registered.

If `event_type` specifies on of the core X protocol events, `select_data` must be a pointer to a value of type `EventMask`, indicating the mask to be used to deselect for the appropriate

event. If the widget is realized, `XtRemoveEventHandler` calls `XSelectInput` if necessary. Specifying `NULL` for `select_data` is equivalent to specifying a pointer to an event mask containing 0. This is similar to the `XtRemoveRawEventHandler` function.

If `event_type` specifies an extension event type then the semantics of the data pointed to by `select_data` are defined by the extension selector registered for the specified event type.

The `XtRegisterExtensionSelector` function registers a procedure to arrange for the delivery of extension events to widgets.

If `min_event_type` and `max_event_type` match the parameters to a previous call to `XtRegisterExtensionSelector` for the same display, the `proc` and `client_data` replace the previously registered values. If the range specified by `min_event_type` and `max_event_type` overlaps the range of the parameters to a previous call for the same display in any other way, an error results.

The `XtSetEventDispatcher` function registers the event dispatcher procedure specified by `proc` for events with the type `event_type`. The previously registered dispatcher (or the default dispatcher if there was no previously registered dispatcher) is returned. If `proc` is `NULL`, the default procedure is restored for the specified type.

In the future, when `XtDispatchEvent` is called with an event of `event_type`, the specified `proc` (or the default dispatcher) will be invoked to determine a widget to dispatch the event.

The `XtDispatchEventToWidget` function scans the list of registered event handlers for the specified widget and calls each handler that has been registered for the specified event type, subject to the `continue_to_dispatch` value returned by each handler. The Intrinsic behave as if event handlers were registered at the head of the list for `Expose`, `NoExpose`, `GraphicsExpose`, and `VisibilityNotify` events to invoke the widget's expose procedure according to the exposure compression rules and to update the widget's visible field if `visible_interest` is `True`. These internal event handlers never set `continue_to_dispatch` to `False`.

`XtDispatchEventToWidget` returns `True` if any event handler was called and `False` otherwise.

SEE ALSO

`XtGetKeyboardFocusWidget(3)`

X Toolkit Intrinsic - C Language Interface

Xlib - C Language X Interface

