An external client widget is a widget that is part of another program but functions as an Emacs frame. This is intended to be a more powerful replacement for standard text widgets.

1 Using an External Client Widget

There are three different implementations of the external client widget. One is designed for use in Motif applications and is linked with the option `-lextcli_Xm`. Another is designed for non-Motif applications that still use the X toolkit; it is linked with the option `-lextcli_Xt`. The third is designed for applications that do not use the X toolkit; it is linked with the option `-lextcli_Xlib`. In order to use an external client widget in a client program that uses the X toolkit (i.e. either of the first two options described above), simply create an instance of widget type ExternalClient and link your program with the appropriate library. The corresponding header file is called '`ExternalClient.h`'.

Documentation still needs to be provided for using the raw Xlib version of the external client widget.

The external client widget will not do anything until an instance of Emacs is told about this particular widget. To do that, call the function `make-frame`, specifying a value for the frame parameter `window-id`. This value should be a string containing the decimal representation of the widget's X window ID number (this can be obtained by the Xt function `XtWindow()`). In order for the client program to communicate this information to Emacs, a method such as sending a ToolTalk message needs to be used.

Once `make-frame` has been called, Emacs will create a frame that occupies the client widget's window. This frame can be used just like any other frame in Emacs.

2 External Client Widget Resource Settings

The external client widget is a subclass of the Motif widget XmPrimitive and thus inherits all its resources. In addition, the following new resources are defined:

'`deadShell (class DeadShell)`'
: A boolean resource indicating whether the last request to the ExternalShell widget that contains the frame corresponding to this widget timed out. If true, no further requests will be made (all requests will automatically fail) until a response to the last request is received. This resource should normally not be set by the user.

'`shellTimeout (class ShellTimeout)`'
: A value specifying how long (in milliseconds) the client should wait for a response when making a request to the corresponding ExternalShell widget. If this timeout is exceeded, the client will assume that the shell is dead and will fail the request and all subsequent requests until a response to the request is received. Default value is 5000, or 5 seconds.

The shell that contains the frame corresponding to an external client widget is of type ExternalShell, as opposed to standard frames, whose shell is of type TopLevelShell. The ExternalShell widget is a direct subclass of Shell and thus inherits its resources. In addition, the following new resources are defined:

'`window (class Window)`'
: The X window ID of the widget to use for this Emacs frame. This is normally set by the call to `x-create-frame` and should not be modified by the user.

'`deadClient (class DeadClient)`'
: A boolean resource indicating whether the last request to the corresponding ExternalClient widget timed out. If true, no further requests will be made (all requests will automatically fail) until a response to the last request is received. This resource should normally not be set by the user.

'`ClientTimeout (class ClientTimeout)`'
: A value specifying how long (in milliseconds) the shell should wait for a response when making a request to the corresponding ExternalClient widget. If this timeout is exceeded, the shell will assume that the client is dead and will fail the request and all subsequent requests until a response to the request is received. Default value is 5000, or 5 seconds.

Note that the requests that are made between the client and the shell are primarily for handling query-geometry and geometry-manager requests made by parent or child widgets.

3 Motif-Speci c Info About the External Client Widget

By default, the external client widget has navigation type '`XmTAB_GROUP`'.

The widget traversal keystrokes are modified slightly from the standard XmPrimitive keystrokes. In particular, ⟨TAB⟩ alone does not traverse to the next widget (Ctrl- ⟨TAB⟩ must be used instead), but functions like a normal ⟨TAB⟩ in Emacs. This follows the semantics of the Motif text widget. The traversal keystrokes Ctrl- ⟨TAB⟩ and Shift- ⟨TAB⟩ are silently filtered by the external client widget and are not seen by Emacs.

4 External Client Widget Internals

The following text is lifted verbatim from Ben Wing's comments in '`ExternalShell.c`'.

This is a special Shell that is designed to use an externally- provided window created by someone else (possibly another process). That other window should have an associated widget of class ExternalClient. The two widgets communicate with each other using ClientMessage events and properties on the external window.

Ideally this feature should be independent of Emacs. Unfortunately there are lots and lots of specifics that need to be dealt with for this to work properly, and some of them can't conveniently be handled within the widget's methods. Some day the code may be rewritten so that the embedded-widget feature can be used by any application, with appropriate entry points that are called at specific points within the application.

This feature is similar to the OLE (Object Linking & Embedding) feature provided by MS Windows.

Communication between this shell and the client widget:

Communication is through ClientMessage events with message_type EXTW_NOTIFY and format 32. Both the shell and the client widget communicate with each other by sending the message to the same window (the "external window" below), and the data.l[0] value is used to determine who sent the message.

The data is formatted as follows:

data.l[0] = who sent this message: external_shell_send (0) or external_client_send (1) data.l[1] = message type (see enum en_extw_notify below) data.l[2-4] = data associated with this message

EventHandler() handles messages from the other side.

extw_send_notify_3() sends a message to the other side.

extw_send_geometry_value() is used when an XtWidgetGeometry structure needs to be sent. This is too much data to fit into a ClientMessage, so the data is stored in a property and then extw_send_notify_3() is called.

extw_get_geometry_value() receives an XtWidgetGeometry structure from a property.

extw_wait_for_response() is used when a response to a sent message is expected. It looks for a matching event within a particular timeout.

The particular message types are as follows:

1) extw_notify_init (event_window, event_mask)

This is sent from the shell to the client after the shell realizes its EmacsFrame widget on the client's "external window". This tells the client that it should start passing along events of the types specified in event_mask. event_window specifies the window of the EmacsFrame widget, which is a child of the client's external window.

extw_notify_init (client_type)

When the client receives an extw_notify_init message from the shell, it sends back a message of the same sort specifying the type of the toolkit used by the client (Motif, generic Xt, or Xlib).

2) extw_notify_end ()

This is sent from the shell to the client when the shell's EmacsFrame widget is destroyed, and tells the client to stop passing events along.

3) extw_notify_qg (result)

This is sent from the client to the shell when a QueryGeometry request is received on the client. The XtWidgetGeometry structure specified in the QueryGeometry request is passed on in the EXTW_QUERY_GEOMETRY property (of type EXTW_WIDGET_GEOMETRY) on the external window. result is unused.

In response, the shell passes the QueryGeometry request down the widget tree, and when a response is received, sends a message of type extw_notify_qg back to the client, with result specifying the GeometryResult value. If this value is XtGeometryAlmost, the returned XtWidgetGeometry structure is stored into the same property as above. [BPW is there a possible race condition here?]

4) extw_notify_gm (result)

A very similar procedure to that for extw_notify_qg is followed when the shell's RootGeometryManager method is called, indicating that a child widget wishes to change the shell's geometry. The XtWidgetGeometry structure is stored in the EXTW_GEOMETRY_MANAGER property.

5) extw_notify_focus_in (), extw_notify_focus_out ()

These are sent from the client to the shell when the client gains or loses the keyboard focus. It is done this way because Xt maintains its own concept of keyboard focus and only the client knows this information.

Short Contents

Table of Contents