NAME

dragdrop – Protocol used by drag-and-drop support for Tcl7.4/Tk4.0

DESCRIPTION

The drag-and-drop protocol is built directly on top of the Tk **send** command which implements a RPC mechanism between Tk4.0 (and possibly later) applications sharing the same display.

SERVER SIDE

A drag operation commences with a call to **drag_start** with the absolute location of the mouse cursor when the button was depressed and the types that the source widget is willing to supply. The procedure sets up internal variables so that the rest of the calls that form part of the drag-and-drop protocol work. As part of this it calls the *notify* command associated with the source widget with the **startat** parameter. The last stage of this part of the protocol is an invocation of the next stage of the process (**drag**) to set up the cursor, dependant on whether the widget will accept drops from itself.

Moving the mouse with the button depressed causes calls to be made to **drag** with the absolute location of the mouse cursor. The procedure translates this into destination interpreter and widget by calling **dragdrop_find**. If an interpreter is found, it checks to see if this was the window that the cursor was in at the previous call to this function, and if this not the case, sends a message to old destination interpreter indicating that it has lost the mouse cursor.

Then, a message is sent to the new remote interpreter, telling it the current location of the mouse cursor and the name of the interpreter initiating the drag-and-drop action, and asking whether a drop is allowable at that location. Then, on the basis of the information returned, it configures the cursor and calls the *notify* command with either the **able** or the **unable** parameters.

Upon the release of a mouse button, the **drop** procedure is called with the location of the mouse cursor. This procedure, after finding the drop destination by a call to **dragdrop_find**, informs the drop widget that the mouse pointer has left it, and follows this up with a signal stating that a drop has occurred from the source application. Finally the procedure resets the mouse cursor to normal, and calls the *notify* command with either the **dropped** or the **failed** parameters, depending upon the success of the drop.

dragdrop_find implements the algorithm for searching for applications which will accept data from, trying a cached value of the interpreter first in order to reduce the amount of communication when dragging within the screen area served by a single interpreter. An interpreter is capable of participating in the protocol if, and only if, the cursor is over a widget controlled by the interpreter (determined by seeing if

```
send -- $interpreter_name winfo containing $x $y
```

succeeds with a non-empty value) and the variable **dragdrop** exists (an array counts as a variable for the purposes of this test, as it uses

```
send -- $interpreter_name info globals dragdrop
```

to determine the presence or absence of the variable).

The final element of the source side of the drag-and-drop mechanism (**drop_get_data**) merely returns the data of the requested type, and is there to be invoked directly from the far end of the communication.

CLIENT SIDE

During a drag operation, calls are made to the procedure **drag_at** whenever the mouse is moved with the button down. If the location given in the arguments is a valid one, then it updates the list of data types that the source interpreter is willing to supply if it doesn't already have the current value, and calls the drag handler to determine if the widget actually wants data dropped at the current location.

If the location is invalid, then the message sent was really saying that the mouse cursor had left the widget, which should be notified of the fact.

Drop operations are performed when the server makes a RPC to **drop_at** which transfers the supplied type list if it hasn't already been, and attempts to match it against the list of types accepted (with the acceptance list determining the order of preference). If a match is found, then the data corresponding to the matched type is transferred by making a RPC to the server routine **drop_get_data** and the drop handler for the destination widget is called with the type of the transferred and the data itself, in addition to destination widget, the source interpreter, and the coordinates of the drop location, relative to the destination widget.

SEE ALSO

dragdrop(n), **send**(n)

AUTHOR

Donal K. Fellows: `fellowsd@cs.man.ac.uk`

NAME

dragdrop_types – types of data for transferring via drag-and-drop actions

DESCRIPTION

The type description codes used by the drag-and-drop package may be any (preferably short) strings that do not contain the null character (due to underlying system implementations), but it is recommended that the type codes be restrained to the upper case character, the dash (minus) and number symbols, as this helps type codes stand out from the text surrounding them and can aid greatly in debugging.

There are no fixed guidelines for these symbols, but the following symbols should be treated as being fixed for the sake of compatability:

`STRING`
: Plain textual data.

`FILENAME`
: Untyped file reference (filing system specific)

`TYPEDFILENAME`
: A tcl list where the first element is a code describing the type of the file, and the second element is the name of the code as in `FILENAME`.

  Suggested filename types include:

  `TEXT` Plain text

  `HTML` Hypertext Markup Language source

  `C` C language source

  `TCL` Tcl source

  `XBM` X11 bitmap

  `PPM` Portable pixmap (raw-bits)

  `GIF` Graphics interchange format (may have `-87` or `-89` to specify the version)

  `COMPRESSED`
  : File compressed with **compress**(1).

  `GZIPPED`
  : File compressed with **gzip**(1).

  `UUENCODED`
  : File encoded with **uuencode**(1).

  `EXECUTABLE`
  : File which may be executed in the context of the application generating the original message. This should only be allowed if the file is allowed to be executed (execute permission for the user) and can physically be executed by the architecture that would run it. It is anticipated that where `EXECUTABLE` file references are passed, there will follow further IPC between the source and destination applications in order to determine where and how to actually run the application.

  `DIRECTORY`
  : The name passed refers to a directory, and not an ordinary file.

  These file types may be combined into a space-separated list, with the leading type having precedence over the others (so *foo.html.gz.uue* might have the file type list `UUENCODED GZIPPED HTML`)

`URL` Uniform resource locator.

`LIST` List of pairs, where the first element of each pair (really a two element list) is the type of the second element.

Where a you want to define a new type which you want to be private to only a few applications (i.e. not for general use) then you should precede the name with `X-LOCAL-` to avoid confusion.

SEE ALSO

dragdrop(n)

AUTHOR

Donal K. Fellows, `fellowsd@cs.man.ac.uk`

NAME

dragdrop – Drag-and-drop support for Tcl7.4/Tk4.0

PROCEDURE SYNOPSIS

dragdrop_bind *widget data_server generator button ?notifier?*
dragdrop_destination *widget drag_handler types drop_handler*

supplies_string *widget string*
supply_string *widget type*
accepts_drop *x y source_app data_type_list*

drag_start *x y types*
drag *x y*
drop *x y*

drag_at *x y source_app*
drop_at *x y source_app*

drop_get_data *type*
dragdrop_find *x y*

VARIABLE SYNOPSIS

dragdrop
dragdropnotify
dragserver
draghandler
drophandler
dropacceptedtypes

DESCRIPTION

This is a Tcl-level library (actually a combination of two libraries) for inter- and intra-interpreter drag-and-drop for Tk4.0 applications.

dragdrop_bind and **dragdrop_destination** implement the main user interface to the drag-and-drop code, though for complex bindings it might be necessary to access **drag_start, drag** and **drop** directly. **drag_at, drop_at** and **drop_get_data** are designed to only be called by **send** from a remote Tk application, and are not supposed to be called by users of the library unless they are reimplementing part of the library.

supply_string, supplies_string and **accepts_drop** are sample procedures for the drag-and-drop servers and control routines, and their use is demonstrated in **dragdrop_demo**(1).

dragdrop_find implements a search algorithm, finding which interpreter (and which widget within that interpreter) actually has the mouse cursor in it. It uses cached values of the interpreter to reduce the amount of IPC for the average case (motion within an interpreter).

dragdrop, dragdropnotify, dragserver, draghandler, drophandler and **dropacceptedtypes** are arrays used internally to the drag-and-drop system. Mostly, these should not be modified by the user (except where duplicating the action of **dragdrop_bind** or **dragdrop_server),** but there are three user-modifiable members of the **dragdrop** array:

dragdrop(debug)
Set to 1 to turn on debugging (**NB** this is rather verbose). This is intended to be turned on by a remote application, as this is how the author uses it...

dragdrop(yes)
The cursor used when it is possible to drop data at the current position, in any form acceptable to **Tk_GetCursor**(3)

dragdrop(no)
The cursor used when it is not possible to drop data at the current position, in any form acceptable to **Tk_GetCursor**(3)

CREATING A DRAG-AND-DROP SERVER

The command

dragdrop_bind *widget data_server generator button ?notifier?*

makes the widget described by the *widget* argument a source of drag-and-drop data. The other arguments are:

data_server
Specifies a procedure to call to supply the data requested when during a drop. This takes the arguments:

widget the source widget

type type of data requested

generator
This is a script to evaluate at the start of a drag operation which informs the dragging engine of the types of data which the data_server is capable of supplying. It can also be used to move the relevant parts of the state of the source widget into a temporary buffer. See **dragdrop_types**(7) for details of the type scheme.

button This is which mouse button the dragging operation should be bound to. Ideally this would be button 1, but this tends to conflict with other bindings on the widget, so great care should be exercised where such a tactic is adopted.

notifier This is the name of a procedure to call to signify the progress of the current dragging operation (defaults to none). The first argument indicates what sort of message is being passed. The actual forms generated are:

notifier **startat** *widget x y*
This call is generated when the mouse button is pressed to begin the drag operation, and could be used to bring up a window indicating the data being carried by the drag operation.

notifier **able** *widget x y drop_interpreter drop_widget*
This call is generated when a drop operation would be possible at the *drop_widget* given in the *drop_interpreter.* This is only generated from mouse movement with the button down.

notifier **unable** *widget x y*
This call is generated when a drop operation would not be bossible at the current location (either because you are not over a Tk4 application, or the widget you are currently over does not support dropping at the current location in the destination widget/interpreter). This is only generated from mouse movement with the button down.

notifier **dropped** *widget x y drop_interpreter drop_widget*
This call is generated when the data has been successfully dropped into the *drop_interpreter* at the location *x,y* which lies within the *drop_widget.* This can be used to withdraw or destroy any windows created by the **startat** call and to also delete the data from the source widget if this is so required. This is only generated by a release of the dragging button.

notifier **failed** *widget x y*
This call is generated when the drag-and-drop engine failed to transfer the data from the source to any destination, and is called when the dragging button is released. It can be used to delete or withdraw any windows created by the **startat** call.

An example usage of the **dragdrop_bind** command is:

```
pack [entry .e]
dragdrop_bind .e supply_string {
      supplies_string .e [.e get]
} 3
```

This creates an entry widget and makes it into a source for drag-and-drop operations. It only supplies data of type *STRING* where the data supplied is the contents of the entry, as supplied by the .e get widget command.

CREATING A DRAG-AND-DROP CLIENT

The command

dragdrop_destination *widget drag_handler types drop_handler*

makes the widget described by the *widget* argument a destination for drag-and-drop data. The other arguments are:

drag_handler
This is a procedure which returns a boolean value (as determined by **Tcl_GetBoolean**(3)) stating whether a drop operation is possible at the location given within the widget, this information being used to select the cursor used, so relaying whether a drop would succeed to the user. The procedure is called as:

handler widget x y source_interpreter data_types

The arguments passed are:

widget The widget underneath the mouse cursor.

x y The coordinates of the mouse cursor. Unlike the coordinates used throughout the rest of the drag-and-drop engine, these are relative to the destination widget. If these are empty strings then the mouse cursor has left the widget.

source_interpreter
The interpreter which will supply the data to be dropped. **NB** The data is not necessarily available at this point.

data_types
List of types of data that the *source_interpreter* is prepared to supply the data as.

types This is a list of the types accepted by the *drop_handler* of the widget in order of preference. The drag-and-drop mechanism selects the first member of this list that the data source is prepared to supply and retrieves the data associated to this type from the source. Some suggested types are listed elsewhere in this document.

drop_handler
This is a procedure which performs the actual handling of the dropped data. It is called as:

handler widget x y source_interpreter data_type data

The arguments passed are:

widget The widget underneath the mouse cursor.

x y The coordinates of the mouse cursor. Unlike the coordinates used throughout the rest of the drag-and-drop engine, these are relative to the destination widget. If these are empty strings then the mouse cursor has left the widget.

source_interpreter
The interpreter that supplied the data dropped.

data_type
The type of the data dropped.

data The data dropped. **NB** This data is not necessarily still available in the *source_interpreter*

An example usage of the **dragdrop_destination** command is:

```
proc drop_on_listbox {w x y s t d} {
      insert end {}
      insert "@$x,$y" $d
      delete end
}
pack [listbox .l]
dragdrop_destination .l accepts_drop STRING drop_on_listbox
```

This creates an entry widget and makes it into a source for drag-and-drop operations. It only supplies data of type *STRING* where the data supplied is the contents of the entry, as supplied by the .e get widget command.

DESCRIPTION OF VARIABLES

The drag-and-drop system uses six arrays:

dragdrop
This contains the main state of the system, with the following fields:

debug debugging flag.

yes cursor (in form acceptable to **Tk_GetCursor**(3)) to use when a drop is possible. Defaults to a green arrow.

no cursor (in form acceptable to **Tk_GetCursor**(3)) to use when a drop is not possible. Defaults to a red circle.

types possible types of data to transfer.

source source interpreter (in destination).

widget source widget.

cursor original cursor used in source widget.

rmt_widget
destination widget (used for caching)

rmt_interp
destination interpreter (used for caching)

*supplies,**
temporaries used by **supply_string** and **supplies_string**

dragdropnotify
This array (indexed by widget name) is used to store action notify procedure names for the current interpreter (part of the drag-and-drop source side).

dragserver
This array (indexed by widget name) is used to store data generation procedure names for the current interpreter (part of the drag-and-drop source side).

draghandler
This array (indexed by widget name) is used to store drag action handler procedure names for the current interpreter (part of the drag-and-drop destination side).

drophandler
This array (indexed by widget name) is used to store drop action handler procedure names for the current interpreter (part of the drag-and-drop destination side).

dropacceptedtypes
This array (indexed by widget name) is used to store lists of types accepted by the drag-and-drop destination procedures (specifically the drop action handlers).

FILES

dragdrop.tcl
Implements the drag-and-drop protocol (procedures **drag_start**, **drag**, **drop**, **drop_get_data**, **drag_at**, and **drop_at**)

dragdrop_ifc.tcl
Implements a sample programmers interface to the drag-and-drop protocol (procedures **drag-drop_bind**, **dragdrop_destination**, **supplies_string**, **supply_string**, and **accepts_drop**)

tclIndex
Tcl autoload directory

SEE ALSO

Tcl_GetBoolean(3), **Tk_GetCursor**(3), **dragdrop**(7), **dragdrop_demo**(1), **dragdrop_types**(7), **send**(n), **winfo**(n), **wm**(n)

BUGS

Are you kidding? Mind you, I'd like to know of any bugs that you find, both in the library and in this documentation.

TO DO

Document the actual drag-and-drop protocol.

Add hooks to allow (with a suitable C extension) dragging to a Motif application. Allowing dragging in the opposite direction would be tricky though.

Make a real application that uses this package.

AUTHOR

Donal K. Fellows, `fellowsd@cs.man.ac.uk`

NAME

dragdrop_demo – Demonstration application for drag-and-drop in Tcl7.4/Tk4.0

SYNOPSIS

dragdrop_demo

DESCRIPTION

This is a quick and dirty demonstration of the capabilities of the drag-and-drop system documented in **dragdrop**(n). It creates a listbox and an entry, and you may drag a line of text with mouse button 3 from either the listbox or the entry to insert it into the listbox, or replace the contents of the entry with (though replacing the contents of the entry with the contents of the entry is a null operation). Should you select part of the text in the entry, or one of the lines in the listbox, then this will be dragged and dropped.

Note that you may also drag data between multiple copies of this application, running on separate machines. All that is required is that the application windows are on the same X display, and that it is possible to use **send**(n) between Tcl/Tk interpreters on that display.

There is a flag to set which allow you to drag a window which displays the data that is being dragged, but the performance of this option is sometimes not very good on my workstation for some reason. I suspect that this actually has something to do with scheduling, since once the window has actually decided to start moving, everything is nice and quick.

See **wish**(1) for details of other options that can be passed to the interpreter driving this application.

BUGS

The message at the top of the application window does not like having the window resized around it. I have no idea how to fix this.

The bell sounds on a drop on the root window. This is because of either some bugs in my window manager (**tvtwm**(1) where I'm working at the moment) or because of a bug in the **winfo**(n) **containing** command.

Any bugs in the drag-and-drop system apply here also.

SEE ALSO

dragdrop(n), **wish**(1)

AUTHOR

Donal K. Fellows, `fellowsd@cs.man.ac.uk`