NAME

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

SYNOPSIS

dragdrop *option* ?*arg arg ...*?

HELPER PROCEDURE SYNOPSIS

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

dragdrop-enable-text *window*
dragdrop-enable-entry *window*

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

VARIABLE SYNOPSIS

dragdrop
dragdrop_version
dragdrop_patchLevel
dragdrop_library

DESCRIPTION

This is an extension (written in C and Tcl) that provides inter- and intra-interpreter drag-and-drop support for Tk4.0 (and later versions). Deleting the **dragdrop** command disables all drag-and-drop support. Renaming the command changes the command used for all RPC interactions by the application, so if you rename the command to the same thing in a group of applications, then the user will be able to drag-and-drop between those applications, but no others. Note that the **dragdrop** array is used by all implementations (changing this currently requires alterations to the header files − specifically *src/dragdropInt.h* in the definition of the `MAIN` macro). The legal forms of the command are:

dragdrop containing *rootX rootY*

Returns the path name for the window containing the point given by *rootX* and *rootY*, much as in the **winfo containing** *rootX rootY* command except that several extra checks are applied to the window being returned (that it is mapped, and that it actually does contain the point given, this latter being needed because of bugs in some window managers, or so I expect.

dragdrop data *type*

Returns the data from the source window implied by the *type* argument. *Should only be called by the drag-and-drop mechanism, and not by the user of this extension.*

dragdrop debug ?*boolean*?

If *boolean* is specified, then the state of the debugging flag (global within the scope of the current process) is updated to the value specified (*boolean* must have a proper boolean value, such as **0** or **yes**). Debugging is off by default.

dragdrop destination *window* ?*option arg ...*?

This command is used to manipulate the configuration of the drop target *window*. The behaviour of the command depends on the *option* argument. The following forms of the command are currently supported:

dragdrop destination *window*

Equivalent to the **dragdrop destination** *window* **isdestination** command.

dragdrop destination *window* **dragcmd** ?*command*?

If *command* is specified, sets a command to be called to notify the destination window that a drag is occurring over it. The command will have the pathname of the target window, the x and y location of the mouse pointer (relative to the target window), the source interpreter and the list of types that the source interpreter is prepared to provide all appended to it (as list elements) before execution. If the command returns a true value (as determined by **Tcl_GetBoolean**) then a drop is permitted at the current position, and a drop is not permitted otherwise. If the *command* argument is not specified, then the current current command associated with dragging actions over the window is returned. This command will make window *window* into a drag-and-drop target if it is not one already. When the mouse leaves the window (either because it moves to another window, or because the dragging button is released), a command similar to the above, except with the x and y locations specified as empty strings, will be invoked to notify the target window that it is no longer under a dragging cursor. In this case, the return value is ignored. If the command defined for a window does not return success, then the command for the toplevel containing that window is called (if it is defined). At the termination of a drag operation within a window, the **dragcmd** will be called for the toplevel containing the window if it is defined, and there is no **dragcmd** for the window itself.

Thus, if the command

```
dragdrop destination .window dragcmd can_drop
```

was issued, the command

`can_drop .window` *x y srcInterp typeList*

would be executed every time the mouse moves over window *.window* during a drag operation, and the command

`can_drop .window {} {}` *srcInterp typeList*

would be executed when the mouse leaves the window or the dragging button is released over the window. If, instead, the command

```
dragdrop destination . dragcmd can_drop_toplevel
```

then the command issued for a drag over the window would be

`can_drop_toplevel .window` *x y srcInterp typeList*

provided that there is no **dragcmd** defined for *.window* elsewhere.

dragdrop destination *window* **dropcmd** *type* ?*command*? ?*aftertype*?

If *command* is specified, sets a command to be called to notify the destination window that a drop has occurred over it which offers data of type *type*. The command will have the pathname of the target window, the x and y location of the mouse pointer (relative to the target window), the source interpreter, the type of the dropped data, and the data itself (all formatted as proper lists) all appended to it before execution. If the *command* argument is not specified, then the current current command associated with dropping actions of the given type over the window is returned. If the window *window* is not already a drag-and-drop target, this command will make it into one, and if *command* is specified as an empty string, then the entry in the type list for the type given will be removed. The return value from the dropping action is always ignored.

The *aftertype* argument, where specified, says where in the list of types the type associated with the command should be inserted. This should either be a type already present in the list, or one of the special symbols:

first Make the type first in the list of types.

last Make the type last in the list of types.

This matters because, in determining what type to accept, the drag-and-drop mechanism considers the types in the order specified in this list, with the mechanism preferring types earlier in the list. If this argument is left unspecified, the type will be appended to the list if it is not already present in the list, and otherwise left in its current location. If there is no matching type for the widget, then the type list for the toplevel window containing the widget is also searched for a match.

Thus, if the command

```
dragdrop destination .window dropcmd \
        STRING drop_data first
```

was issued, the command

```
drop_data .window x y srcInterp STRING data
```

would be executed when the dragging button for a dragging action is released over *.window* and one of the types of data offered during the data transfer is STRING, with this taking preference over all other data types.

dragdrop destination *window* **forget**
This makes window *window* cease to be a drag-and-drop destination.

dragdrop destination *window* **isdestination**
This asks whether window *window* is a drag-and-drop destination, returning a boolean value.

dragdrop destination *window* **types**
This returns the list of types of data accepted by window *window* in the current order of preference. An empty string is returned if no types of data are currently accepted (either because no dropping handlers are defined, or because the window is not a drag-and-drop target).

dragdrop drag *rootX rootY*
Updates the drag-and-drop mechanism to take account of the (specified) position of the mouse if called during a dragging action, and ignored otherwise. Should be bound to the appropriate modified motion binding for the dragging button associated with a particular window.

dragdrop drop *rootX rootY*
Terminates the current drag-and-drop action at the given coordinate, causing a transfer of data to occur at this point if the there is an accepting target below the pointer, returning the type of the data that was transferred. Has no effect if a drag is not in progress. Should be bound to the appropriate button release binding for the dragging button associated with a particular window.

dragdrop library

Returns the name of the library directory in which standard drag-n-drop scripts are stored. This is actually the value of the **dragdrop_library** variable and may be changed by setting **dragdrop_library**.

dragdrop move *rootX rootY sourceInterp*

Propagates the current dragging position to the destination interpreter, causing a check to see if any of the offered data types can be accepted. *Should only be called by the drag-and-drop mechanism, and not by the user of this extension.*

dragdrop patchLevel

Returns the current value of the **dragdrop_patchLevel** variable.

dragdrop reset

Resets the internal dragging status flag. Only use if a drag has somehow terminated without a call of the **dragdrop drop** command, and the drag-and-drop mechanism has thus become confused.

dragdrop set *rootX rootY sourceInterp*

Propagates the drop position to the destination interpreter, causing a transfer of data of a type offered by the source and accepted by the destination, with the destination choosing what type by the method specified in the comments on the **dragdrop destination** *window* **dropcmd** command above. *Should only be called by the drag-and-drop mechanism, and not by the user of this extension.*

dragdrop source *window* ?*option arg ...*?

This command is used to manipulate the configuration of the drag source *window*. The behaviour of the command depends on the *option* argument. The following forms of the command are currently supported:

dragdrop source *window*

Equivalent to the **dragdrop source** *window* **issource** command.

dragdrop source *window* **datacmd** *type* ?*command*?

If *command* is specified, then this updates the command that will be executed to generate the data requested of type *type*. This command will only be executed when the data is actually requested by the destination side of a drag-and-drop operation during a drop action. If *command* is specified as an empty string, then the data generator command for the given type is removed, and if *command* is not specified at all, then the current data generator command is returned. If a command is given and the window is not a drag-and-drop source, then it is made into one, though this also really requires some bindings to be made as well. If no **datacmd** is specified for a particular requested type in a particular window, then the **datacmd** for the toplevel containing that window is called (provided it is defined).

When invoked, the data generator command has the name of the window for which it is serving data and the type of the data requested appended (as proper lists) before execution. Thus, if the command

```
dragdrop source .window datacmd STRING get_some_data
```

was issued, the command

```
get_some_data .window STRING
```

would be executed when a drag-and-drop action starting at *.window* requests some data of type STRING in the droping phase of the action.

dragdrop source *window* **forget**

This makes window *window* cease to be a drag-and-drop source (though the drag-and-drop bindings for the window should really be removed as well).

dragdrop source *window* **issource**

This asks whether window *window* is a drag-and-drop source, returning a boolean value.

dragdrop source *window* **notify** ?*command*?

If *command* is specified, then this updates the command that will be executed to notify the source window *window* of the status of drag-and-drop actions that start from it. If *command* is specified as an empty string, then the status notification command for the given type is removed, and if *command* is not specified at all, then the current status notification command is returned. This also makes *window* into a drag-and-drop source if it was not already one (and a command is specified). If a source window has no **notify** command, but the toplevel containing that window does have a **notify** command, then that will be called instead (with exactly the same arguments).

Currently definied notification actions are as follows:

command **startat** *srcWindow rootX rootY*

A drag-and-drop action has started in *srcWindow* at the given absolute location.

command **able** *srcWindow rootX rootY destInterp destWindow*

A drag-and-drop action started in *srcWindow* is able to drop in *destWindow* of interpreter *destInterp* at the given absolute location.

command **unable** *srcWindow rootX rootY*

A drag-and-drop action started in *srcWindow* is unable to drop at the given absolute location.

command **dropped** *srcWindow rootX rootY destInterp destWindow*

A drag-and-drop action started in *srcWindow* has succeeded in dropping data in *destWindow* of interpreter *destInterp* at the given absolute location.

command **failed** *srcWindow rootX rootY*

A drag-and-drop action started in *srcWindow* has failed to drop any data at the given absolute location.

dragdrop source *window* **types**

Returns a list of all the types that window *window* knows how to serve, in no particular order. An empty list will be returned if no data generators are defined, or if *window* is not a drag-and-drop source.

dragdrop start *rootX rootY offeredTypes*

Starts a drag-and-drop action at the absolute location indicated, offering types *offeredTypes* for transfer. Should be bound to the appropriate button press binding for the dragging button associated with a particular window.

dragdrop version

Returns the current value of the **dragdrop_version** variable.

Providing drag-and-drop support for entries and text widgets is simple, as there are predefined procedures to do just that:

dragdrop-enable-text, and
dragdrop-enable-entry

These procedures make their target window (given by the argument) into a drag-and-drop source and destination, capable of allowing data to be dragged about using the third mouse-button (mainly because it doesn't clash with any of the standard Tk bindings). These enhanced text (entry) widgets will generate and accept data packets of type `STRING`, `TEXT-INTERNAL-COPY` (`ENTRY-INTERNAL-COPY`) and `TEXT-INTERNAL-MOVE` (`ENTRY-INTERNAL-MOVE`), with the last two being done by holding down the Control (Shift) key while beginning the drag. No moving from a disabled text or entry widget is allowed (the same is true for show-restricted entries), and no widget accepts data that it cannot consume. The only caveats are the **dragcmd** associated with these widgets will change to another procedure (which actually chains the old procedure) and the fact that you need to add the information out the non-standard types that you are prepared to supply. Luckily the latter is easy - add something like

`set dragdropgenerator(%W)` *TYPE*

to the widget binding for button 3.

In order to facilitate the use of simple drag-and-drop schemes, the Tcl procedures **dragdrop_bind** and **dragdrop_destination** are provided. They implement a simple user interface to the drag-and-drop code, though for complex bindings it is necessary to access **dragdrop start**, **dragdrop drag** and **dragdrop drop** directly. **supply_string**, **supplies_string** and **accepts_drop** are sample procedures for the drag-and-drop servers and control routines, and their use of all these procedures is demonstrated in **dragdrop_demo**(1).

dragdrop is an array used internally to the drag-and-drop system. Most of the members should not be modified by the user (except where duplicating the action of **dragdrop_bind** or **dragdrop_destination**), but there are two user-modifiable members of the **dragdrop** array:

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).

THE DRAG-AND-DROP PROTOCOL

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 **dragdrop start** with the absolute location of the mouse cursor when the button was depressed and the types that the source window 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 window with the **startat** parameter. The last stage of this part of the protocol is an invocation of the next stage of the process (**dragdrop drag**) to set up the cursor, dependant on whether the window will accept drops from itself.

Moving the mouse with the button depressed causes calls to be made to **dragdrop_drag** with the absolute location of the mouse cursor. The list of current interpreters is searched for an interpreter that has a window that covers the given point (equivalent to invoking the

winfo containing *x y*

command in every interpreter, except with extra some checks that are performed by the **dragdrop containing** command). In order to speed this process up, the first process checked is the process that last succeeded at this operation (allowing motion within an interpreter to be handled much more efficiently). 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 (using the **dragdrop move** command, 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 **dragdrop drop** command is called with the location of the mouse cursor. The destination window is determined using the same method as in **dragdrop drag** above (actually a call to the same internal routine), and that window is informed that the mouse pointer has left it, and that message is immediately followed by a message stating that a drop has occurred from the source application (by remote invocation of the **dragdrop set** command). 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.

A window is considered to be a valid drop zone if:

[1] its interpreter has the command **dragdrop**,

[2] it is the uppermost window under the mouse cursor,

[3] the window is mapped,

[4] the window actually contains the point mentioned (both of these previous two conditions are needed to cope with what I suspect to be bugs in some window managers), and

[5] the patchlevel of the dragdrop mechanism at the remote end matches that at the source end.

The final element of the source side of the drag-and-drop mechanism (**dragdrop data**) merely returns the data of the requested type from the source window (the window used as the source for this operation is that which is cached by the mechanism), 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 **dragdrop move** 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 window 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 window, which should be notified of the fact.

Drop operations are performed when the server makes a RPC to **dragdrop set** 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 **dragdrop data** and the drop handler for the destination window is called with the type of the transferred and the data itself, in addition to destination window, the source interpreter, and the coordinates of the drop location, relative to the destination window.

TYPES OF DATA

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.

`ENTRY-INTERNAL-MOVE`
: Used for moving selected text between entries. Do not support except through the default interface for entries (**dragdrop-enable-entry**).

`ENTRY-INTERNAL-COPY`
Used for copying selected text between entries. Do not support except through the default interface for entries (**dragdrop-enable-entry**).

`TEXT-INTERNAL-MOVE`
Used for moving selected text between entries. Do not support except through the default interface for entries (**dragdrop-enable-text**).

`TEXT-INTERNAL-COPY`
Used for copying selected text between entries. Do not support except through the default interface for entries (**dragdrop-enable-text**).

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.

DESCRIPTION OF VISIBLE STATE

The drag-and-drop system uses one user-visible array:

dragdrop

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

yes cursor (in form acceptable to **Tk_GetCursor**(3)) to use when a drop is possible. Defaults to an arrow, which will be coloured green unless the *MONOCHROME* symbol is defined in the extension header (or makefile), or the root window is determined to be monochrome (by examination of the visual installed).

no cursor (in form acceptable to **Tk_GetCursor**(3)) to use when a drop is not possible. Defaults to a circle, which will be coloured red unless the *MONOCHROME* symbol is defined in the extension header (or makefile), or the root window is determined to be monochrome (by examination of the visual installed).

types possible types of data to transfer.

source source interpreter (in destination).

widget source window.

cursor original cursor used in source window.

rmt_widget
destination window (used for caching)

rmt_interp
destination interpreter (used for caching)

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

FILES

src/.[ch]*
Implements the drag-and-drop protocol (the **dragdrop** command)

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

lib/text.tcl
Implements standard drag-and-drop behaviour for text widgets.

lib/entry.tcl
Implements standard drag-and-drop behaviour for entry widgets.

man/.[1n]*
Documentation (in nroff form, tested using *groff*)

SEE ALSO

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

BUGS

Does not handle multiple displays properly. Mind you, the **send** command didn't until very recently, apparently (according to the changes file at least). Actually, it is not having multiple displays open that is a problem, but rather that the main windows of the interpreters must currently be on the same display...

There was something else, but I've forgotten what it was...

TO DO

Check that this works with Tk4.1 and see if it happy as a 'load'ed extension (I don't see why this shouldn't work unless Tk4.1 lacks a particular internal function which I call directly since it circumvents rather a lot of fiddling around at the Tcl level).

Put some support for fancy multi-colour cursors in. Unfortunately, this is not trivial, as it requires finding out about the shaped window extension... Help with this would be highly appreciated.

Add proper drag and drop support to other publically available code of mine.

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 − further investigation reveals that the problem is actually caused by a **raise**(n) command, but since this is necessary on at least some platforms (at least one that I'm developing on) it has to stay in there.

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

FILES

lib/dragdrop_ifc.tcl
    Quick interface to the drag-and-drop protocol.

lib/.xbm*
    Bitmaps for the rejection cursor.

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 doesn't always sound on a drop failure (when it is turned on). I'm not sure what is wrong here, but I do not like it. This could possibly be due to some bugs in my window manager (**tvtwm**(1) where I'm working at the moment), especially as this doesn't seem to be a problem with **fvwm**(1).

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`