NAME

drag&drop − facilities for handling drag&drop data transfers

SYNOPSIS

drag&drop source
drag&drop source *window* ?*options*?
drag&drop source *window* **handler** ?*dataType*? ?*command arg arg...*?

drag&drop target
drag&drop target *window* **handler** ?*dataType command arg arg...*?

drag&drop target *window* **handle** *dataType* ?*value*?

drag&drop token *window*

drag&drop drag *window x y*
drag&drop drop *window x y*
drag&drop active
drag&drop errors ?*proc*?
drag&drop location ?*x y*?

DESCRIPTION

The **drag&drop** command provides access to a set of facilities for managing drag-and-drop data transfers. Any of the usual Tk widgets can be registered to participate in the drag-and-drop process. Widgets registered as a drag&drop *source* can export data to other widgets registered as a drag&drop *target*. Note that a particular widget can be registered as a source, as a target, or as both.

The drag-and-drop process begins when the user clicks and holds a mouse button in a source window; a token window appears with an icon or message to represent the data being transferred. As the user moves the mouse pointer, the token window follows along, acting as a movable packet of data. Whenever the mouse pointer falls on a valid target window, the border of the token window is changed to a raised (active) state. When the mouse button is released over the target window, a Tcl routine is invoked to send the data to the desired application, and the target window is asked to "handle" the data. If this communication process fails, a rejection symbol (a circle with a line through it) is displayed on the token window to indicate failure.

The details of the communication process are fully configurable by the application developer. In the simplest case, the value that is sent to the target window is a simple string. The target window is simply asked to "handle" that string value. In general, the source window can have a special "handler" procedure to transfer a particular data type by issuing a series of "send" commands. After this, the target window is again asked to "handle" the result.

Both sources and targets can have a list of "handlers" for different data types. As a token window is dragged from its source to various targets, each target is checked to see if it recognizes a handler offered by the source. If it does, it is treated as a valid target. Otherwise, it is ignored. This scheme allows the same source to interact with many different kinds of targets. For example, a source for RGB color samples might have "color" and "string" handlers. This would allow it to communicate with "color" targets (sending RGB data) as well as entry widgets (sending strings of the form "#rrggbb").

This introduction was presented as a brief overview of the communication process; further details are presented below:

drag&drop source

Returns a list of path names for widgets registered as drag&drop sources. Returns an empty string if no widgets have been registered.

drag&drop source *window* ?*options*?

Registers a new drag&drop source window with the given options, or modifies the options for an existing window:

Name: **buttonBinding**
Class: **ButtonBinding**
Switch: **−button** *n*

Specifies the mouse button (integer 1-5) that will invoke the drag&drop operation on the source window. This causes the following bindings to be added to the widget:

bind *win* **<ButtonPress-***n***> {drag&drop drag %W %X %Y}**
bind *win* **<B***n***-Motion> {drag&drop drag %W %X %Y}**
bind *win* **<ButtonRelease-***n***> {drag&drop drop %W %X %Y}**

The default value is button 3. If the value "0" is specified, then no bindings are added; this enables the user to establish bindings manually.

Name: **packageCommand**
Class: **Command**
Switch: **−packagecmd** *command*

Specifies a Tcl command used to establish the appearance of the token window at the start of each drag&drop operation. This command is automatically invoked by the **drag&drop drag** command whenever the token window is about to be mapped for a drag operation. It should update the appearance of the token window to represent the data that is being moved.

The following substitutions are made in the *command* string before it is executed:

%t Replaced with the window path name for the token which represents the data being dragged.

%W Replaced with the window path name for the drag&drop source.

The return value from the package command represents the data being transferred. If the package command returns an empty string, the drag operation is quietly aborted. This can be used to disallow drag&drop operations from certain parts of a widget, if the drag position is inappropriate.

For example, the following package routine will select an item from a listbox and configure the token window to display the selected string. It uses the**drag&dr op location** command to determine the entry in the listbox that the user has selected and it returns this as the data value:

```
proc package_list_item {lbox token} {
   set xy [drag&drop location]
   set y  [expr [lindex $xy 1]-[winfo rooty $lbox]]

   set str [$lbox get [$lbox nearest $y]]
   $token.value configure -text $str
   return $str
}
```

The return value is available later when the source and target communicate. If the source has a command associated with its data handler, then this value is substituted in place of "%v" in the source handler. Otherwise, it is substituted in place of "%v" in the target handler.

Name: **rejectBackground**
Class: **Background**
Switch: **−rejectbg** *color*

Specifies the color used to draw the background of the rejection symbol on the token window. The rejection symbol (a circle with a line through it--the international "no") appears whenever communication fails.

Name: **rejectForeground**
Class: **Foreground**
Switch: **−rejectfg** *color*

Specifies the color used to draw the foreground of the rejection symbol on the token window.

Name: **rejectStipple**
Class: **Stipple**
Switch: **−rejectstipple** *pattern*

Specifies a stipple pattern used to draw the foreground of the rejection symbol on the token window. Any of the forms acceptable to Tk_GetBitmap can be used.

Name: **selfTarget**
Class: **SelfTarget**
Switch: **−selftarget** *boolean*

If the *boolean* value is true, and if a source widget is also registered as a compatible target, then the source will be able to transmit to itself during drag&drop operations. This is primarily useful for complex sources such as a canvas widget, where items may be moved from place to place within the same widget. By default, this option is disabled.

Name: **send**
Class: **Send**
Switch: **−send** *list*

Specifies a *list* of *dataTypes* enabled for communication. Only *dataTypes* defined by commands of the form "**drag&drop source** *window* **handler** ?*dataType* ?*command arg arg...*?" are allowed. This list also determines the priority of the various *dataTypes*. When a token window is over a potential drag&drop target, this list is searched from start to finish for a *dataType* that is also recognized by the target. The first matching*dataT ype* found determines the value that will be sent if the token is dropped. If no matching *dataType* is found, then the target is incompatible, and is ignored. By default, this option has the value "all", indicating that all *dataTypes* should be considered in the order that they were defined for the source.

Note that this option makes it easy to control a drag&drop source. Setting the value to an empty string disables the source; setting the value back to "all" restores communication.

Name: **siteCommand**
Class: **Command**
Switch: **−sitecmd** *command*

Specifies a Tcl command used to update the appearance of the token window. If specified, this command is automatically invoked by the **drag&drop drag** command whenever the token window is over a compatible drag&drop target.

The following substitutions are made in the *command* string before it is executed:

%s Replaced with "1" if the token window is over a compatible target, and "0" otherwise.

%t Replaced with the window path name for the token which represents the data being dragged.

Regardless of this command, border of the token window will become raised whenever the token is over a valid target. This command can be used to display other visual cues.

Name: **tokenAnchor**
Class: **Anchor**
Switch: **−tokenanchor** *anchor*

Specifies how the token window is positioned relative to the mouse pointer coordinates passed to the **drag&drop drag** command. Must be one of the values n, s, e, w, center, nw, ne, sw or se. For example, "nw" means to position the token such that its upper-left corner is at the mouse pointer. The default value is "center".

Name: **tokenBackground**
Class: **Background**
Switch: **−tokenbg** *color*

Specifies the color used to draw the background of the token window.

Name: **tokenBorderWidth**
Class: **BorderWidth**
Switch: **−tokenborderwidth** *size*

Specifies the width in pixels of the border around the token window. This border becomes raised to indicate when the token is over a compatible drag&drop target site. The value may have any of the forms acceptable to Tk_GetPixels. The default value is "3".

Name: **tokenCursor**
Class: **Cursor**
Switch: **−tokencursor** *cursor*

Specifies the cursor used when a token window is active. The value may have any of the forms acceptable to Tk_GetCursor. The default value is "center_ptr".

drag&drop source *window* **handler** ?*dataType*? ?*command arg arg...*?
With no extra arguments, this command returns a list of all *dataType* names that have been registered for the source *window*. If only the*dataT ype* is specified, then the *dataType* is created if necessary, and the command associated with the *dataType* is returned. Otherwise, it concatenates the *command* and any extra *arg* strings, and registers a new *dataType* with this command.

The following substitutions are made in the *command* string before it is executed:

%i Replaced with the name of the interpreter for the target application.

%v Replaced with the value returned from the "-packagecmd" command.

%w Replaced with the window path name for the target window.

A typical source handler contains one or more "send" commands which transfer data to the remote application. The target window is then asked to handle the new data. Whatever value is returned by the source *command* handler is automatically substituted into the "%v" fields of the target handler.

This separation between the transfer and the handling of the data is important. It allows the same source handler to transfer data for many different targets, and it allows each of the targets to handle the incoming data differently. If an error is encountered during the communication process, the rejection symbol is posted on the token window to indicate failure.

drag&drop target
Returns a list of path names for widgets registered as drag&drop targets. Returns an empty string if no widgets have been registered.

drag&drop target *window* **handler** ?*dataType command arg arg...*?
Registers a new drag&drop target window with a given handler, or modifies the handlers for an existing window. If no*dataT ype* is specified, this command returns the current list of recognized *dataType* strings. Each *dataType* is a symbolic name representing a form of data, and the

corresponding *command* is a Tcl command that specifies how the target will make use of the data. This command is invoked indirectly after a source has transferred data to a target application.

The following substitutions are made in the *command* string before it is executed:

%v In the simplest case, the source window does not have a handler command for the selected *dataType*, and this field is replaced with the result from the "-packagecmd" command. When the source does have a handler command, the result from the "-packagecmd" command is substituted into its "%v" field, and the result from this command is substituted into this field in the target command.

%W Replaced with the window path name for the target window.

drag&drop target *window* handle *dataType* ?*value*?

Searches for the given *dataType* name among the handlers registered for the target *window*, and invokes the appropriate *command*. If a*value* is specified, it is substituted into an y "%v" fields in the handler command associated with the *dataType*. If the*dataT ype* name is not recognized, this command returns an error. This command is invoked automatically by the drag&drop facility when data is being transferred from a source to a target.

drag&drop token *window*

Returns the token window associated with a drag&drop source *window*. The token window is used to represent data as it is being dragged from the source to a target. When a source is first established, its token window must be filled with widgets to display the source data. For example,

```
drag&drop source .foo

set win [drag&drop token .foo]
label $win.label -text "Data"
pack $win.label
```

drag&drop drag *window x y*

Marks the start of (or movement during) a drag&drop operation. If the token window is unmapped when this command is invoked, then the **−packagecmd** for the source *window* is executed. If this command is successful and returns a non-null string, the token window is mapped. On subsequent calls, the token window is moved to the new *x y* location. Unless the "**−button 0**" option is specified for the source, this command is automatically bound to <ButtonPress-*n*> and <B*n*-Motion> events for "**−button** *n*" of the source widget.

drag&drop drop *window x y*

Marks the end of a drag&drop operation. If the mouse pointer is over a compatible target window, then the appropriate send handler for the first compatible *dataType* is invoked to handle the data transfer. If the data transfer is successful, then the token window is unmapped; otherwise, a rejection symbol is drawn on the token window, and the window is unmapped after a small delay. Unless the "**−button 0**" option is specified for the source, this command is automatically bound to the <ButtonRelease-*n*> event for "**−button** *n*" of the source widget.

drag&drop active

Returns "1" if a drag&drop operation is in progress, and "0" otherwise. A drag&drop operation officially starts after the package command has been executed successfully, and ends after the send handler has been executed (successfully or otherwise).

drag&drop errors ?*proc*?

Specifies a Tcl *proc* used to handle errors encountered during drag&drop operations. If a *proc* is not specified, this command returns the current error handler. By default, all errors are sent to the usual **tkerror** command, and therefore appear in a dialog box to the user. This behavior is quite useful when debugging communication protocols, but may not be desirable in a finished

application. Errors can be suppressed entirely (leaving the rejection symbol as the only error indicator) by specifying a null string in place of the *proc* name.

drag&drop location ?*x y*?

Used to set or query the pointer location during a drag&drop operation. The *x y* arguments specify the current location; if these arguments are missing, then the last reported (x,y) location is returned as a list with two elements. This command is issued automatically within the **drag&drop drag** and **drag&drop drop** commands, to keep track of pointer movement.

KEYWORDS

drag&drop, send, bind, widget