Drag and Drop

In this module, we describe how to use the Drag and Drop framework and provide examples of its use. Drag and Drop is a new addition in VisualWorks 2.5. A good example of the use of Drag and Drop features is in the VisualWorks tools themselves, the Launcher, browsers and file list. The Drag and Drop version of the tools may be filed in from `extras/tooldd.st`

1. Introduction

To use the Drag and Drop mechanism it's necessary to identify one or more widgets as a *drop source* (a widget offering data to be dragged) and / or a *drop target* (a widget that will receive data). Currently, only the List widget may act as a drop source. All widgets may act as a drop target (except for Dataforms), including a window.

2. Defining Widget Properties

The properties for a Drop Source are defined in the Properties Tool as shown in Fig.1.

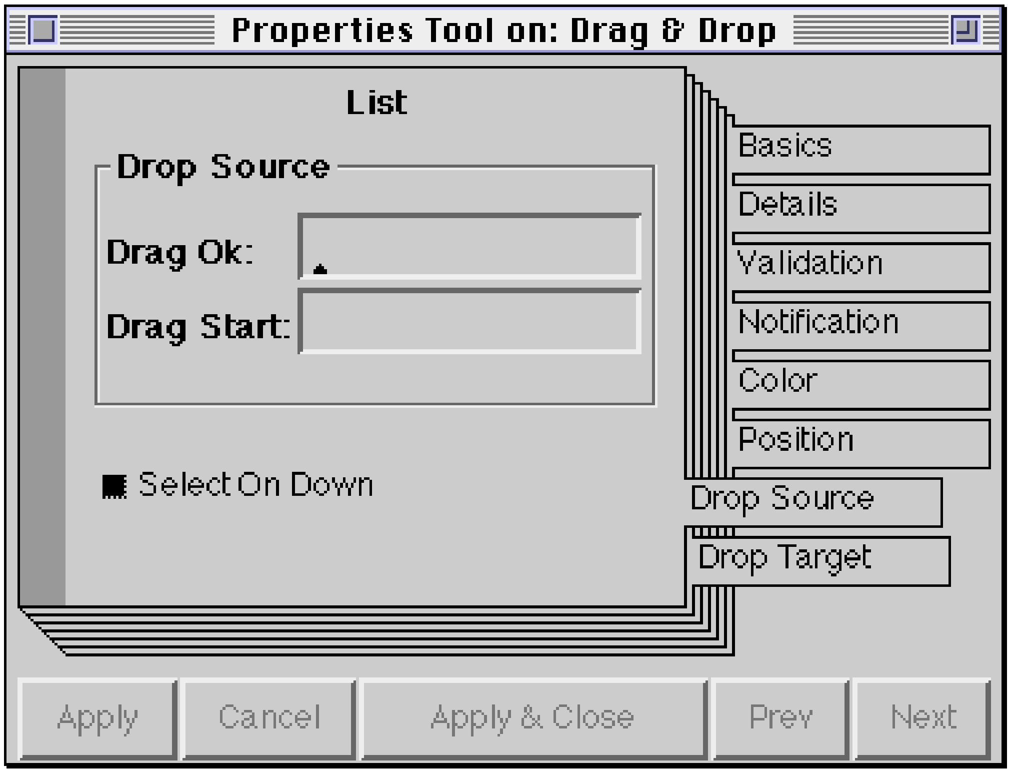

Figure 1: Drop Source Properties

The widget will send the message selector specified by its *Drag Ok* property to its application whenever the user presses the <select> mouse button and moves the cursor in the bounds of the widget. The message is sent to the application model to determine if the drag operation should proceed from this widget. Hence the method corresponding to the message selector must return a Boolean.

If the drag operation proceeds, the message selector specified by the *Drag Start* property is sent to the application model. The corresponding method must create an instance of DragDropManager — it takes responsibility for managing the drag operation.

As the cursor enters a widget, the widget is queried by the DragDropManager to discover if the dragged data can be dropped on it. The DragDropManager queries the widget by sending it the message selector specified by the widget's Drop Target *Entry* property (see Fig.2). The method corresponding to the selector should return an *effect symbol* which identifies a cursor shape (see section 4.1).

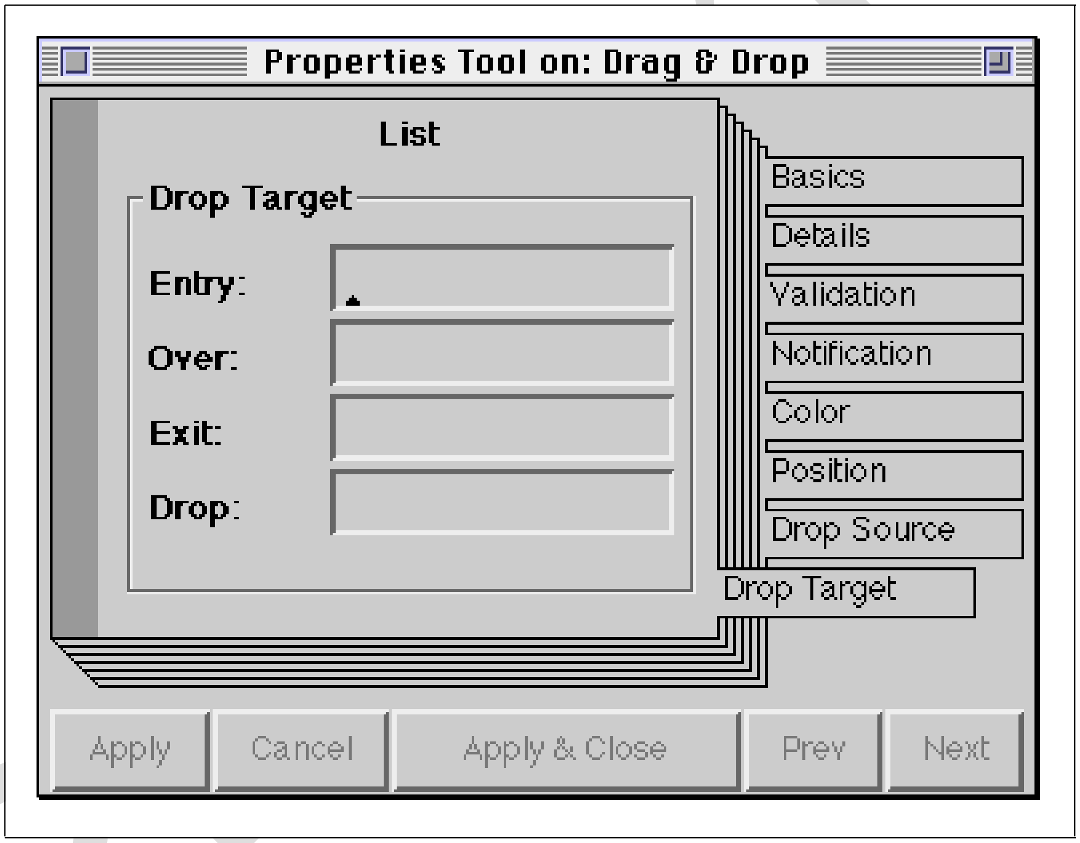

Figure 2: Drop Target Properties

Similarly, the selectors defined by the *Over* and *Exit* properties are sent when the cursor is over, or has just exited, the window or widget. The corresponding methods should also return an effect Symbol.

Lastly, when the mouse is released, the message defined by the *Drop* selector is sent. Typically, the method corresponding to the message will accept the dragged data (after verifying that the data is acceptable) and return an effect Symbol.

NOTE: Although the Properties Tool does not enforce it, all the selectors mentioned above must end with a colon.

3. Defining Drop Source Methods

For the Drag and Drop operation to begin, both the *Drag Ok* and *Drag Start* properties must be specified. The message selectors defined by these properties are sent to the application model at the very beginning of the drag operation. In this section, we describe how to implement the methods corresponding to those message selectors. Note that both selectors should end with a colon, since the widget's controller is passed as an argument to the message.

3.1. Drag OK

This method must return true or false. If it returns true the framework will start the drag operation. In the following example method, a message expression checks that a List widget has a selection:

```
sampleWantsToDrag: aController
    ^self draggableList selection notNil
```

3.2. Drag Start

This method is responsible for creating an instance of DragDropManager. However, a DragDropManager relies on two other objects: an instance of DragDropData and an instance of DropSource.

- An instance of DragDropData holds the data that is to be dragged.
- An instance of DropSource defines the cursor shapes available during the drag operation

A DragDropManager is usually created as follows:

DragDropManager withDropSource: *aDropSource* withData: *aDragDropdata*

where *aDropSource* is a new instance of DropSource and *aDragDropdata* is an instance of DragDropData which has been initialized with information about the source of the drag operation.

Class DragDropdata provides several instance variables — they are described in Table 1.

key	A key which labels the drop. May be queried by potential targets to determine suitability
contextWindow	The current window
contextWidget	The current widget (view or controller)
contextApplication	The current application model

Table 1: Instance Variables of class DragDropData

clientData	Anything — usually an instance of IdentityDictionary with key–value pairs of information specific to this drag operation.

Table 1: Instance Variables of class DragDropData (Continued)

An example method corresponding to a *Drag Start* message selector is as follows:

```
doItemDrag: aController
    "Drag the selected item from the list "

    | aDataSource aDragDropManager aDragDropData |
    "Instantiate a new instance of DragDropData to handle data"
    aDragDropData:= DragDropData new.
    "Set the key, contextWindow, contextWidget and contextApplication
    instance variables"
    aDragDropData key: #itemMove.
    aDragDropData contextWindow: self builder window.
    aDragDropData contextWidget: aController.
    aDragDropData contextApplication: self.
    "Set up the client data"
    aDragDropData clientData: IdentityDictionary new.
    aDragDropData clientData at: #item put: self sampleList selection.
    aDragDropData clientData at: #color put: self sampleList selection color.
    "Instantiate a new DropSource"
    aDataSource := DropSource new.
    "Instantiate a new DragDropManager using the DropSource
    and the DragDropData from above"
    aDragDropManager:=  DragDropManager
                                    withDropSource: aDataSource
                                    withData: aDragDropData.
    "Let the DragDropManager take over"
    aDragDropManager doDragDrop.
```

4. Defining Drop Target Methods

The Drop Target properties are shown in Fig.2. For a widget[1] to act as a target for a drag operation, its *Drop* property must be specified to be the name of a method that implements the desired response when a drop occurs in that widget. In addition, the presence of a *Drop* property causes the interface builder (a UIBuilder) to provide the widget with an instance of ConfigurableDropTarget. The presence of this object indicates to the DragDropManager that the widget is a drop target.

1. The description that follows is also applicable to a window.

The remaining properties (*Entry*, *Over*, and *Exit*) may be used to specify the names of methods that provide visual feedback to the user when the cursor traverses a widget. For example, they might change the shape of the cursor or the appearance of the widget. However, they are not strictly necessary.

Each of the selectors specified by these properties must end with a colon. When the DragDropManager sends a message to the widget's application model, it includes an instance of DragDropContext as the argument to the message. An instance of DragDropContext is produced by the DragDropManager as a combination of its DragDropData, its DropSource and the widget's ConfigurableDropTarget.

4.1. Drag Enter

If a widget specifies a message selector for its *Entry* property, the DragDropManager will send the selector to its application as soon as the cursor enters the bounds of the widget. It is the responsibility of this method return an effect Symbol — a Symbol that identifies the cursor shape to be used. The default effects are shown in Table 2.

↖⊞	#dropEffectCopy	Indicates that the data will be copied if it is dropped on this widget.
↖▭	#dropEffectMove	Indicates that the data will be moved if it is dropped in this widget (i.e., it will be removed from its source widget).
⊘	#dropEffectNone	Indicates that it is not possible to drop the data in this widget.
↖	#dropEffectNormal	The “normal” cursor shape.

Table 2: Effect Symbols

The choice of cursor shape should be based on whether or not the widget will accept the dragged data. The typical way for a method to determine if it should accept the data is to send the message key to the DragDropContext argument. The message will return the key of its DragDropData. The method below is an example of a method that corresponds to an *Entry* selector:

```
widgetDragEnter: aDragContext
    ^aDragContext key == #itemMove
        ifTrue: [#dropEffectCopy]
        ifFalse: [#dropEffectNone]
```

4.2. Drag Over

The message selector specified by a widget's *Over* property is sent to the application immediately after the message specified for the *Entry* property, and then every time the cursor moves within the widget's bounds. Like the method defined for the *Entry* property, this method should return an effect Symbol to indicate whether or not the data may be dropped in this widget.

4.3. Drag Exit

This message specified by the *Exit* property is send when the cursor is dragged out of the bounds of the widget (with the <select> button pressed). Typically, the corresponding method will return the effect Symbol #dropEffectNone to indicate that the data has not been dropped.

4.4. Drop

This method is responsible for implementing the action to be taken when a drop occurs in a widget, i.e., when the user releases the <select> mouse button. The method has access to an instance of DragDropContext that is the argument to the method, which can be interrogated to determine whether to accept the dragged data and, if so, how to deal with it.

If the data is acceptable, the method should request the DragDropData from the DragDropContext argument. It does this by sending it the message sourceData; this in turn may be sent the message clientData to access the variable of the same name. For example:

```
widgetDrop: aDragContext

    | dict |
    aDragContext key == #itemMove
        ifFalse:[^#dropEffectNone].

    dict := aDragContext sourceData clientData.
    dict keysAndValuesDo: [:k :v |
            Transcript cr;
                        show: k printString;
                        tab;
                        show: v printString].
```

The method should return an effect Symbol for the DragDropManager to return to the *Drag Start* method (see section 3.2) that initiated the drag operation.

Ex 1. (This exercise continues with the version of BondEntry that you built during the “Review of Application Model Framework” module.) Add a graphical Label widget to the Bond Entry window to serve as a “drop” target for an instance of BondTrade. Make the List widget a drop source such that an instance of BondTrade may be dragged from the List widget, onto the Label widget, with the effect of populating all of the entry widgets with the value of the BondTrade.