LISA TOOLKIT
SELF—PACED TRAINING

Preface

This self–paced training comprises eleven self–study segments. The intent of these segments is to get you started designing applications with the ToolKit. Although the initial segments have no code associated with them, the latter segments include labs allowing you to experiment with actual application code.

A single application is used as the context for this training. This is the Boxer application. Boxer is implemented in stages over 8 of the eleven segments. The result is an application that exhibits the essential features of typical ToolKit applications. What those are is the subject of this training.

CONTENTS

The following table lists the segments, labs, and the code associated with them:

segment number	segment name	lab	code stage
–	Conceptual Foundation of the ToolKit	no	
1	Introduction to the ToolKit	no	
2	What is a Document?	no	
3	Creating from the Generic Application	yes	1Boxer
4	BlankStationery	no	
5	Intro to the Boxer Application	yes	2Boxer
6	Selections and Highlighting in Boxer	yes	3Boxer
7	Moving Boxes	yes	4Boxer
8	Creating a Box, A Second Selection Class	yes	5Boxer
9	Recoloring, Duplicating, and Clear All Commands with Undo	yes	6Boxer
10	Filters	yes	7Boxer
11	Cut & Paste and Mouse Key Events as Commands; Advanced Commands.	yes	8Boxer

The recommended sequence of segments is to start with "Conceptual Foundation of the ToolKit", and then continue sequentially with segments 1 through 11.

PREREQUISITES

You are expected to have read the following documents before starting this self–study:

- Introduction to Clascal
- WorkShop Manual, especially the QuickDraw and Pascal language sections

To your future as a great ToolKit application designer!

Conceptual Foundation of the ToolKit

The ToolKit is an *object-oriented* development system. This means that the code to be executed is selected through the data, which is packaged into record-like constructs called *objects*. This is in direct contrast to a procedure-oriented system. In that kind of system the code to be called is fixed by the designer. The data must fit the called code, rather than vice versa.

The following discussion and slides provide a conceptual foundation for an object-oriented system, and how it contributes to the structure of the ToolKit.

TOOLKIT APPLICATION DESIGN

The ToolKit is best described as a collection of interlocking hierarchies of classes. One special ToolKit class, **TObject**, is the ancestor of every other class.

As the slide: *The ToolKit Application Model* shows, a user application is a collection of classes as well. Specifically it is a layer of class hierarchies descended from those of the ToolKit.

The user application layer may either abut the ToolKit layer directly, or have one or more *building block* layers of insulation. In either case, the user application may create subclasses from any classes in or above its layer.

(The layered approach is also illustrated in a general way in the slide: The ToolKit Application Design Model.)

TPROCESS

From an application's point of view, one class has especial significance - **TProcess**. An application must define its own subclass of TProcess (T**AppProcess** in the slide). It is only through that subclass that the ToolKit is able to establish access to the user application's code.

The first application object created must be one descended from **TProcess**. That object initiates the creation of every other object in the application. Only if the process object is of your subclass, will it be able to create and reference instances of your classes.

The mechanism uses the fact that all of **TProcess**'s methods are inherited by the subclass. The subclass only needs to override one method to provide access to the rest of the application's classes.

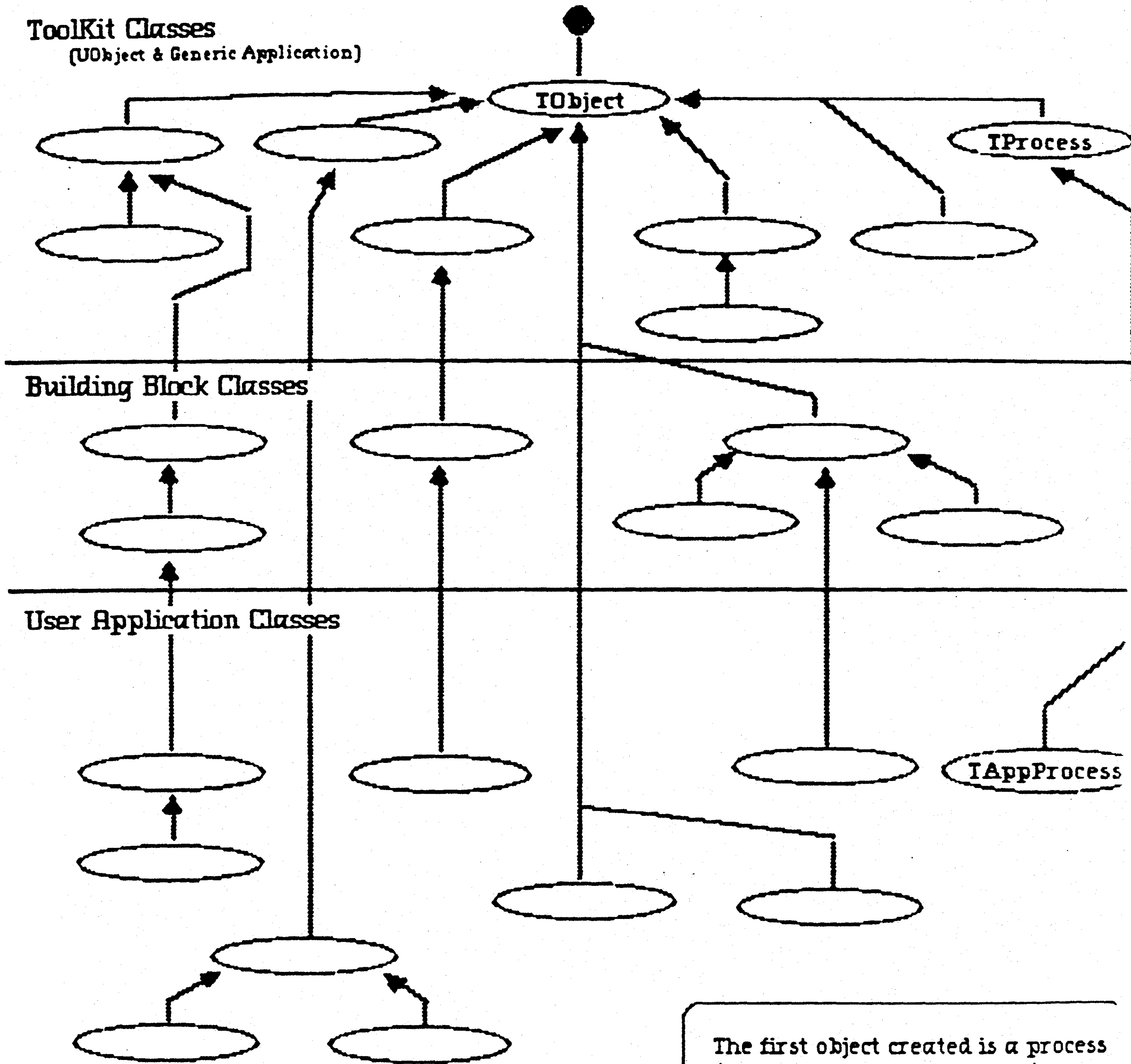
THE GENERIC APPLICATION

The Generic Application is the application defined by the methods of the ToolKit's classes. The Generic Application initiates method calls to your methods through objects of your classes. This is illustrated in the slide: *How Application Code Gets Called*

The primary responsibility of the Generic Application is to route user events, such as mouse downs or keypresses to appropriate methods of your application. It processes all user-generated events directly, freeing you from having to code tedious input/output processing. This also insures that all i/o will be handled consistently from one ToolKit application to the next.

Most significantly, the Generic Application imparts standard behavior to user applications. What it does specifically and how it does this is the subject of the "ToolKit Self-Paced Training".

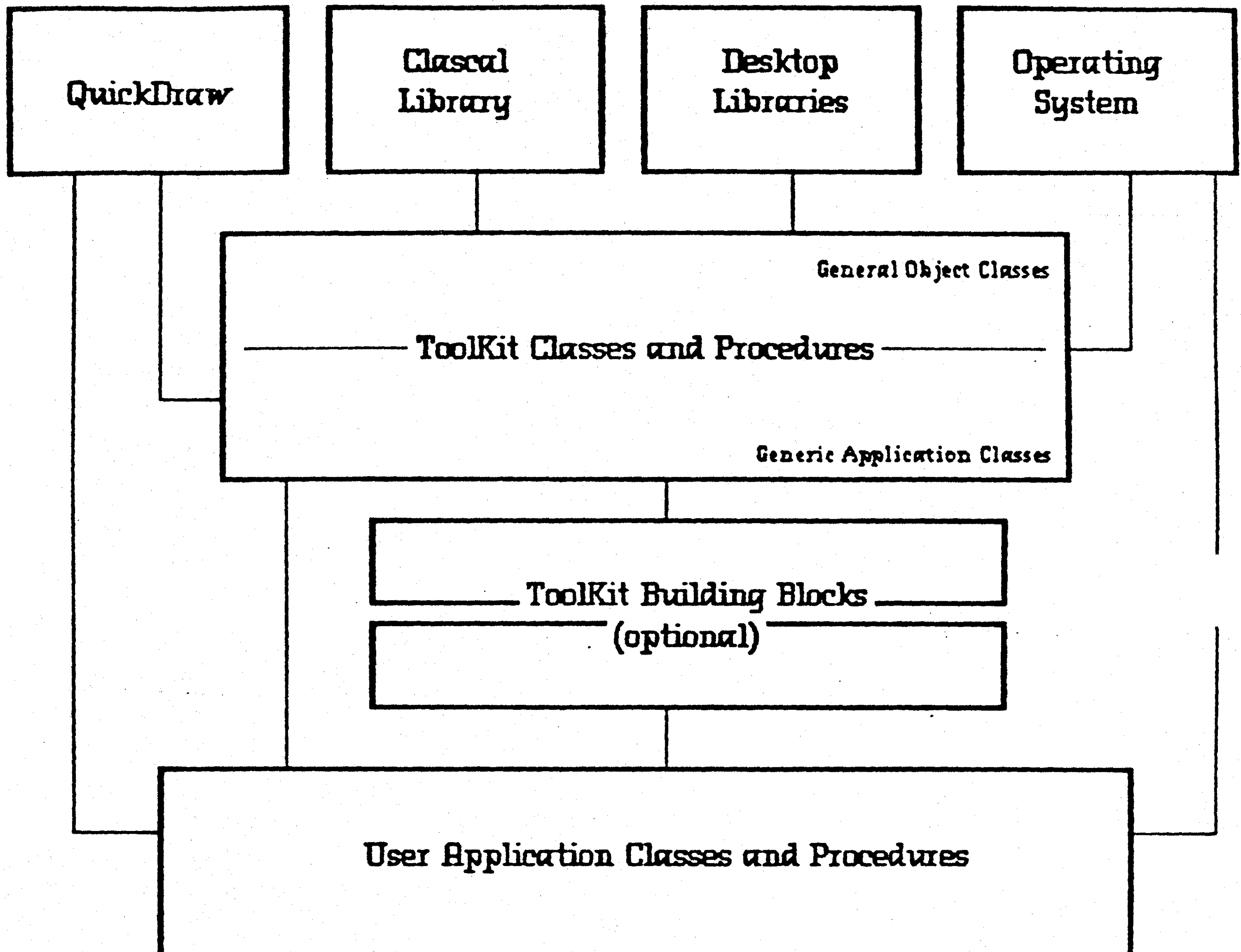
The ToolKit Application Model
(an intricate hierarchy)



The first object created is a process (in this case, an appProcess).

It provides the hook to create o[illegible]r objects from your classes, instead of from the default ToolKit classes.

The ToolKit Application Design Model
(a layered approach)



How Application Code Gets Called
(from the Generic Application)

Generic Application
(ToolKit)

HighLight

Draw

TSelection

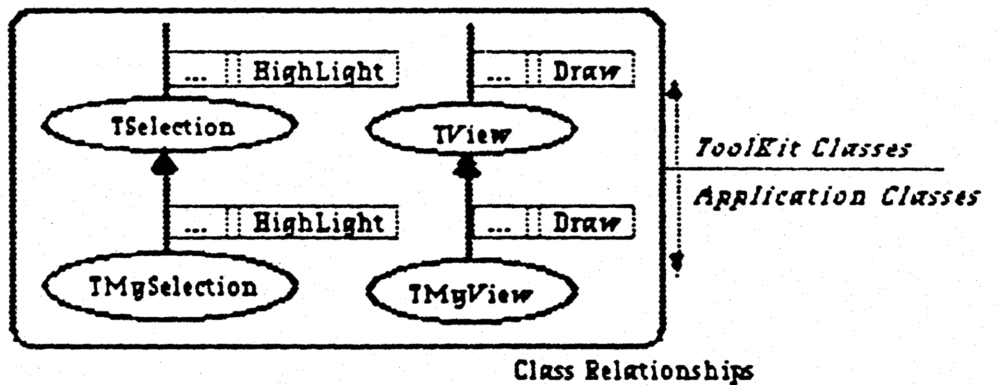
TMySelection

TMyView

TView

... HighLight

... Draw



Class Relationships

If the object was created from one of your classes, then the method selected will be of that class.

CONCLUSION

This concludes the conceptual foundation of the ToolKit.

Please continue with module 1 [Introduction to the ToolKit] of the "ToolKit Self-Paced Training".

[Segment 1]

Introduction to the ToolKit

Purpose of this segment:

To introduce the ToolKit and the Generic Application.

How To use this segment:

This is the first segment of the ToolKit architecture self–paced series. This segment should be only started after having read the **Introduction to Clascal** document.

WHAT THE TOOLKIT IS

The ToolKit is an application development environment for the Lisa 7/7 Office System. Simply stated, it is a package of subroutines and related design aids for developers of software.

The ToolKit enables developers, you, to implement powerful, graphics–oriented applications fully integrated with the Lisa desktop. You use it in the Workshop to design applications that will run on the desktop.

THE GENERIC APPLICATION CONCEPT

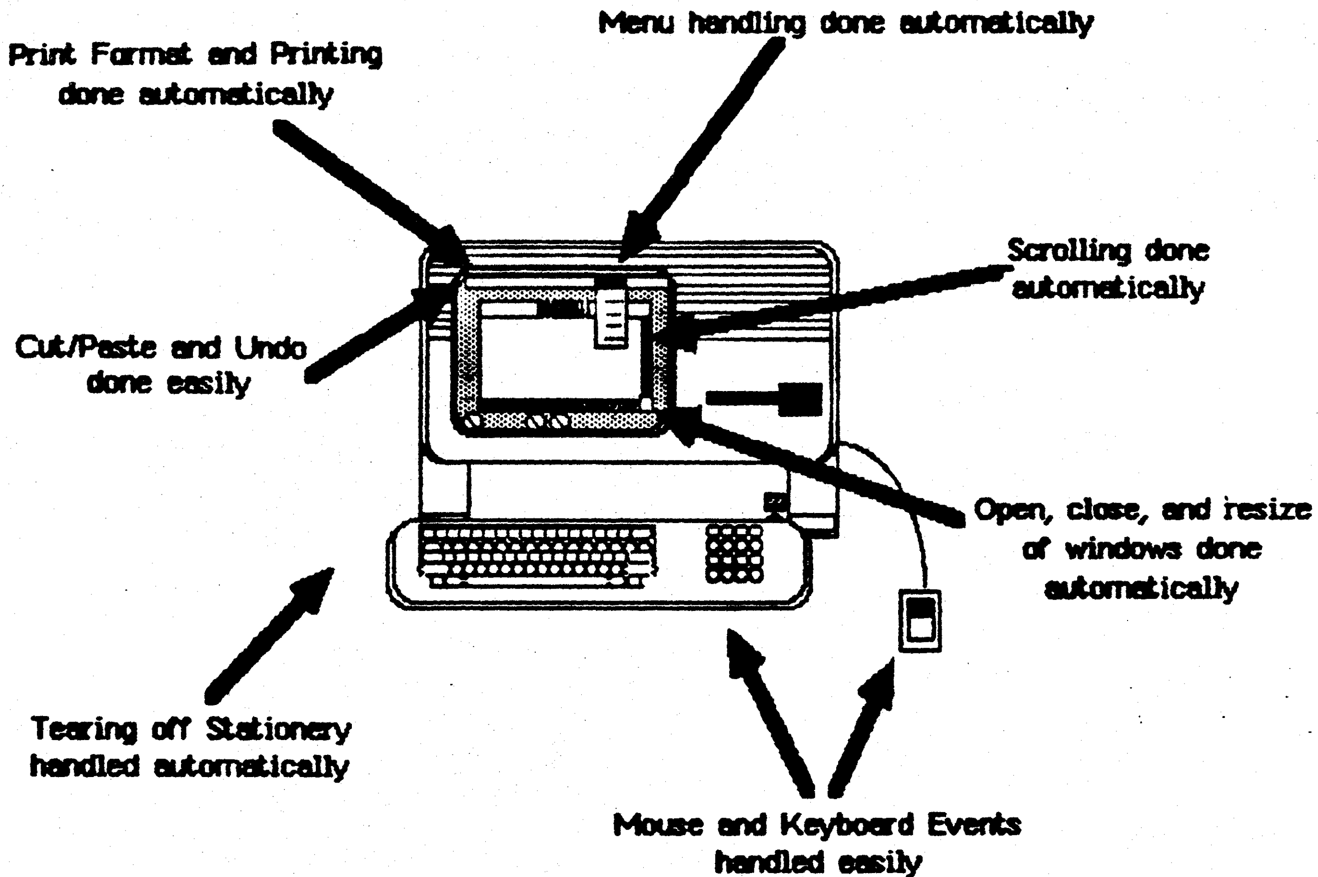
The heart of the ToolKit is the *Generic Application*. The Generic Application is a fully functional desktop application. It opens and closes documents. It creates new documents. It tracks the mouse. It recognizes when keys are pressed. It displays menus. It understands menu commands. It can splits its window into panes. It can even scroll back and forth through the window. It could even print if it had data to manage. But the Generic Application has no user data to manage. You and your users must add the data.

Your job, as a designer, is to customize the Generic Application to manage and display the kinds of data your application will support. The Generic Application is conceived to enhance your productivity in implementing powerful, integrated, user–friendly applications. You just add what is unique to your application on top of the behavior supplied by the Generic Application.

CLASCAL

Apple enhanced the Pascal language to make possible the ToolKit's Generic Application. This enhanced Pascal, called Clascal, supplies a unique structure that

ToolKit Generic Application



couples a data type with code defining the behavior of the variables of that type. The structure is called a *class.*

Classes are the primary data structures in the Generic Application. With Clascal, you customize the Generic Application by defining new classes on top of those it supplies. The class mechanism greatly simplifies your implementation of Lisa—like applications.

You always define a new class in terms of an existing class. Many of your application's classes are derived directly from those in the Generic Application.

INHERITING BEHAVIOR

A new class inherits both the structure and the behavior the class it was descended from. You are free to add new behavior or to redefine any that was inherited.

Inherited class behavior is implemented by sharing code, not copying it. A new class only specifies code unique to the variables of that class.

AN EXAMPLE OF CUSTOMIZING

Let's consider a modification to the Generic Application that you, yourself, might want to make.

You want to implement a document security mechanism. You desire to encode data upon closing a document, and decode data upon opening the document. You start by defining a new class from the ToolKit class, **TDocManager**.

TDocManager supplies two behaviors of immediate interest to you: **Open** and **Close**. In your new docManager class you need to simply redefine **Open** and **Close**. The rest of **TDocManager's** behaviors will be inherited by your new class.

You could rewrite the entire code for **Open** and **Close** to include your encryption mechanism, but this is both difficult and unnecessary. For one thing, you may not have access to the sources for either **Open** or **Close**. For another, Clascal offers a better way to refine the two. You can reference the inherited **Open** and **Close** from within your new **Open** and **Close**, respectively!

Here is how it works. When one of your users selects *close* from the document's File/Print menu, the Generic Application calls the current docManager's **Close** code. Since the current docManager is yours, it uses your **Close**. First, your **Close** tells the document's data to encrypt itself. *This is easy if each type of data is a class.* Next your **Close** calls the **Close** for **TDocManager**, which is Generic Application code. In this fashion you have modified the Generic Application's behavior without having to know any details of the code. *Not many development systems support that!*

To implement **Open**, you first call **TDocManager's Open** code, before commanding your data to decode itself. Your **Open** is called automatically whenever the user selects *open* from the File/Print menu of one of your application's documents.

THE MOST GENERAL CLASS — TObject

Often a new application class will be completely unrelated to any in the Generic Application. Consider a class that describes the behavior of a clock, for instance. To create this kind of class and others, the ToolKit supplies a base class, **TObject**. **TObject** is the ancestor of every class in the Generic Application.

The class **TObject** provides the basic behavior needed by the *instances* of any class. The behavior inherited from **TObject** allows each instance to: duplicate itself, free up any memory it uses, read its value from a character stream, and write its value to a character stream.

TOOLKIT APPLICATION DESIGN

A ToolKit application is fundamentally a hierarchy of classes. You build up the application class by class, sometimes redefining, sometimes enhancing generic behavior. Effectively you layer the new onto the old.

This layering technique is made more powerful by the *unit* mechanism of Lisa Pascal *(which includes Clascal)*. Units enable your application to use previously compiled classes and data types. There is no need to recompile them into your application.

A new application is composed of a four line main program and one or more units referencing ToolKit units. Within each unit is a hierarchy of classes and their implementations.

With a suitable class hierarchy you can maximize the code shared among diverse data types. A well—designed hierarchy can form the base for future applications, as well.

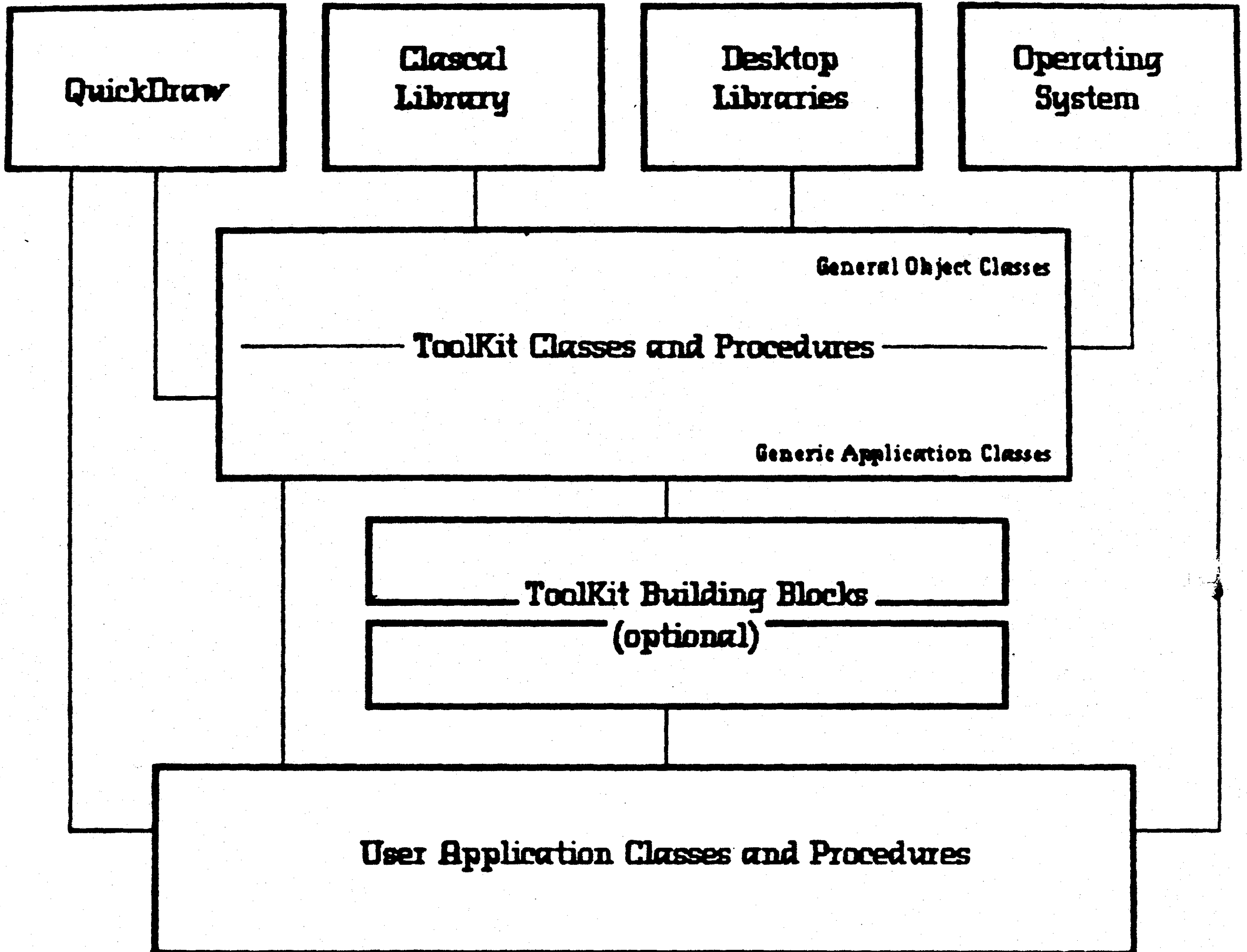
The Generic Application is such a hierarchy. Its classes supply the fundamental behavior of Lisa—like applications. These capabilities include: resizing, scrolling, and updating windows; opening, suspending, and closing application documents; automatic printing of text and graphics; previewing what is to be printed; simplified use of menus and the mouse; and cutting/pasting of information between applications.

BUILDING BLOCKS

Besides the Generic Application, the ToolKit also encompasses application *building blocks*. Each building block is a unit containing an integrated set of classes. Building blocks provide specialized application features such as: text editing and formatting, graphics editing and design, data inquiry dialogs, and international formatting.

You can layer building blocks on top of the Generic Application and underneath your own units. Building blocks enable you to include just the capability you need and nothing more.

The ToolKit Application Design Model
(a layered approach)



In addition to building blocks, the ToolKit also supplies: a high—level debugger specially tailored for ToolKit applications, automated tools for compiling applications and installing them onto the desktop, and utilities for managing numeric, string, and graphics data.

OVERVIEW

The ToolKit is a power tool for application design. Its Generic Application substantially reduces the time needed to create Lisa—like applications.

Harnessing the power of Clascal, the ToolKit maximizes the code shared among your data types. In the multiple process environment of the Lisa, the ToolKit enhances system performance by sharing Generic Application code among concurrently executing applications.

The components of the ToolKit include:

- the Generic Application
- application building blocks
- high—level symbolic debugger
- automated tool for compiling applications
- desktop installation tool
- utilities for dynamic allocation/deallocation of data
- classes and procedures for data conversion and management
- classes and procedures to support large documents

[Segment 2]

What is a Document?

Purpose of this segment:

To introduce documents and their structure as a foundation for understanding the function of the Generic Application.

How to use this segment:

This is the second segment in the ToolKit architecture self—paced series. This segment should be started after the segment "Introduction to the ToolKit". It should be completed before moving on to "Creating from the Generic Application".

INTRODUCTION TO DOCUMENTS

Rarely do end users invoke desktop applications directly. They use them indirectly through *documents*. The structure of the Generic Application is in large part due to the role that documents play on the Lisa desktop.

Lisa documents are special desktop files that you can open, work on, and put away when done. As with other items on the desktop, documents are represented by icons. The document's name is displayed below the icon.

Every document is associated with exactly one application. Typically a document contains data entered during one or more sessions. The document's application determines how this data will be entered, organized, and displayed as information.

A given application may have many documents associated with it. Documents of the same application are represented by the same icon. Each document, of course, can have its own name and its own data.

The data contained within a document can be text, graphics, tables, or any combination thereof. Different applications are able to handle different forms of data. A spreadsheet, for example, excels at processing data in tables. A word processor on the other hand is happiest when working with text, yet may also accommodate graphics.

As for names, each document has two of them. One appears below the document's icon on the desktop. That name *unless it is "Untitled"* is supplied by the user. The other name is not displayed. It is supplied by a built—in desktop tool called the Desktop Manager or Filer. This name associates the document with its application, distinguishes it from all other documents on the same disk, and helps the operating system locate the files that comprise the document.

Documents can vary widely in size. A new graphics document will be virtually empty. A large word processing document may contain hundreds of pages worth of information. *The ToolKit is currently able to support a maximum of 768K bytes in a RAM–based document. This can translate into anything from a 163 page single–spaced "paragraph" to a double–spaced manuscript exceeding 800 pages.* By overriding TDocManager, larger disk–based documents can be implemented.

Documents are subject to the whim of their users. Their contents can be changed; their displayed names can be modified. They can be created or destroyed, opened or closed, all at the discretion of the user.

As far as the Desktop Manager is concerned, a document is composed of its data <u>and</u> some related structure. A ToolKit document, for instance, contains user data as well as all the control information needed to manage it intelligently on the desktop. This additional information ensures that the user interface to the data will be simple, straightforward, and consistent.

Let's follow a moment in the life of an icon to help shed some light on the nature and structure of ToolKit documents. *[Warning: this story may be offensive to hard–core engineers.]*

A Moment in the Life of an Icon

Imagine, if you will, that you are back at the office. You are sitting in your private (five guys around you, and they call it private!) cubicle preparing to do some useful work on your Lisa. You bring up the Office System, and grab a cup of coffee while it is setting up. Ah! The coffee is particularly good this morning. You grab your mouse and send your cursor in search of the icon you clicked on last week. There! There it is, "Memo 1073" (you like to produce memos). A quick double click on "Memo 1073" and things really start to happen.

A window materializes, and the icon disappears. It doesn't even appear that the icon has left a shadow this time. Ah well, nothing to break your heart. You wait a few seconds, and presto, the window fills up with words and pictures. There are some dark black lines splitting up your window. You've tried to get rid of them, but they just won't go away. After that great cup of coffee, though, they don't seem to bother you as much now. Finally! The name in the top bar of the window has changed to inverse video. And, next, the cursor makes its debut. You start typing and mousing merrily away. You are hoping to finish this memo by tonight.

Eleven AM arrives, and you are still busy on "Memo 1073". Your boss walks in. Looks like she has something on her mind. Good god, you forgot to turn in your monthly status report! But first, better put the memo away. Why concern her with the deal you're trying to make – with her boss. You double click on the icon in the top bar of the window; and heave a sigh of relief as the evidence quickly disappears. "Hi Mary! Bet you want that status report now, huh? Yes, I was just ..." The look on her face tells you she's heard this all before. You do derive some comfort, though, that "Memo 1073" is a mere icon again. Amazing how those little icons can hold so much information! But the thought fades quickly as your boss clears her throat....

THE INTERNALS OF A DOCUMENT

The whole system is rather ingenious, when you think about it. The *window* defines where the document data will be displayed. The window is also apparently able to scroll over the data if it can't all be displayed at once.

With some documents you can even have the same data presented in several different ways, simultaneously! The window will be split to display the different representations. For example, LisaGraph documents present their data as both a table of numbers and a graph.

Each representation is framed by its own border. The border belongs to something called a *panel*. Each panel provides a unique view of the document's data. The window is the owner of the panels.

Every panel has at least one *pane*. Usually, there is one pane per panel. If the user has split the view, there will be more than one pane. It is really through the pane that a document's data is displayed.

Scrolling within a pane allows you to see different portions of your document. The total of everything that you could see through that pane is called the *view*. At any one time the pane displays only part of the view. The effect is sort of like looking at the scenery through the window of your car. As the car moves what you actually see changes. But the total landscape is always there. The view, as with the pane, is owned by the panel.

In some panels of some applications, panes can be split (producing more pain). This is done by moving one of the skewers (the stubby, black bar at the end of a panel's scroll bar). Moving the skewer splits both the pane and its rectangle on the view. If there is a scroll bar, it, too, is split to allow each of the newly created panes to focus on a different portion of the view. For example, a top pane could display the start of a document while a bottom pane displays the end.

What about the data? The data is typically associated with the view. Yet, if different representations of the data are to be displayed it is more sensible to keep the data with the window. It is each view, though, that determines how the data is represented. The form of the data, of course, varies with the application.

So far you have seen that documents contain not just data, but also objects such as: windows, panels, panes, scroll bars, scrollers, and views. In addition, there are still other objects that play a role in the document.

Users nearly always want to add to or modify a document's data. Modifying data requires that two more objects be part of the document: *selections* and *commands*.

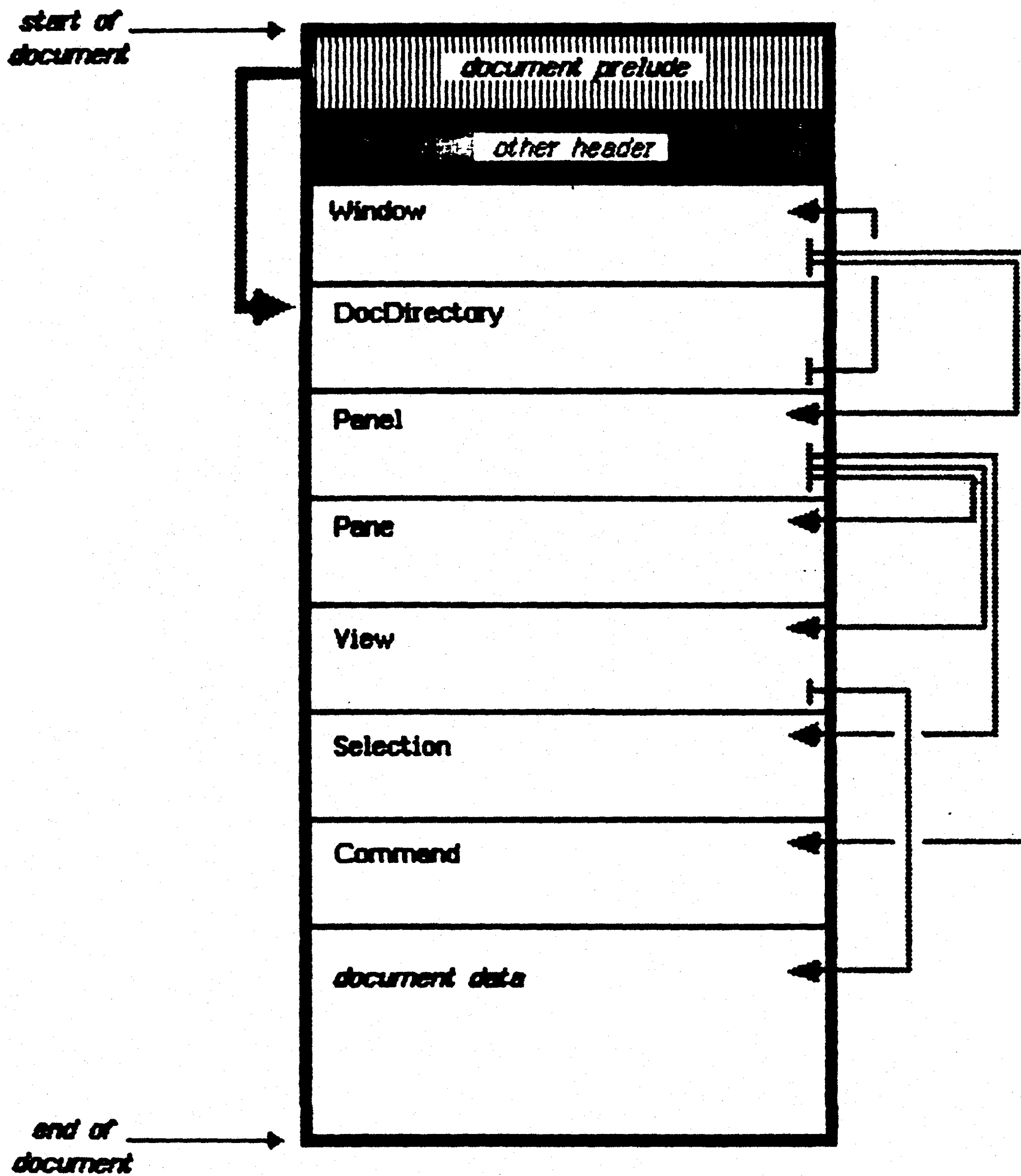
A selection keeps track of that part of the data the user desires to change. Typically, the user uses the mouse to indicate the data to be changed. This may be data that is to be replaced (such as a paragraph), or an insertion point where data is

to be added. A selection always exists, even when no data is to be changed. The selection beloags to the panel. Each panel has a selection.

A command describes how to make a specific change on the data refered to by the selection. Typically, a command is created when the user chooses an action from a menu. For example, one of the menus of a graphics editor may include a ***rotate*** action. It is up to the command to perform the **rotate** action correctly upon whatever graphic object was selected. Only one command is active at any one time. The currently active command belongs to the window.

You have now been introduced to the major components of a document. Here is a diagram of what the inside of the document might look like. The order of the objects in memory may vary from the diagram; what is fixed are the inter—object references shown by the arrows.

Organization of a Document



HEAPS

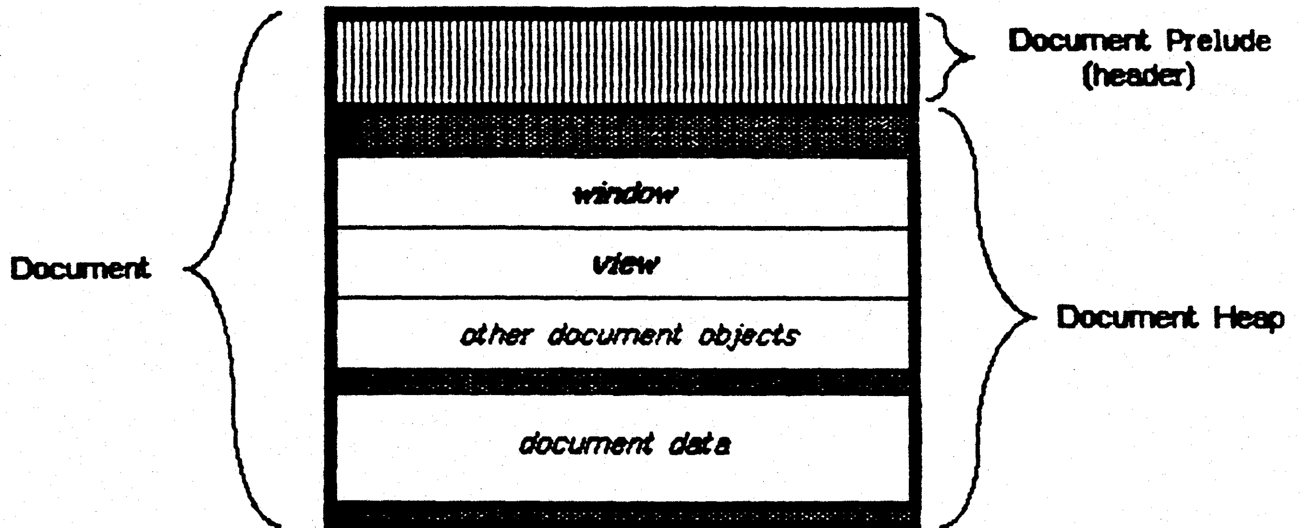
As you have seen, a document is composed of a window, panels, panes, views, selections, commands, and data. What you may not know is that this data is grouped together into something called a *heap*.

A heap is a contiguous span of random—access memory (RAM) where data is stored. The actual size of a heap may vary during execution. All references are relocatable within the heap. This means that the application can access data within the heap without knowing where it is actually stored. This allows the heap manager to dynamically organize heap space in the most efficient way, without disrupting an application's ability to access a document's data.

The language of the ToolKit, Clascal, provides a special feature to refer to data within heaps. This feature is called a *handle*. A handle is a double indirect reference to data. The application refers to a document's data through handles. The handle provides a reliable reference to data whose actual address (within the heap) may vary.

Heaps are associated with one or more *logical data segments*. The logical data segments provide an address space for the data. It is often useful to have the address of a piece of data, but since the data can be relocated at any time, it is much safer to use a handle. Below is a diagram illustrating the relationship between a handle and the heap data it refers to.

A Document and its Heap



A document contains a heap and a document prelude.

The heap contains the data and the objects needed to manage and display the data.

The prelude contains the information needed to initialize the heap from the document's disk file.

PUTTING A DOCUMENT AWAY

The end user can put a document away using a command in the File/Print Menu. The document heap (holding the view, the panel, the window, and whatever else had a hand in managing your document's data) is copied into a file. Then the file gets closed. Finally, the window frame goes away, leaving an icon as the sole reminder of the document.

To assure that a document can be closed and opened at any time, all the state of the document must be in the heap between commands. In particular, no state should be in global variables. Some applications may keep part of the document state in files they manage. An application doing so is responsible to close/open those files when the user puts away/opens the document.

SUMMARY

A document is a user—originated set of files used with a specific application. In addition to data entered by the user, a document also contains the objects needed to supply a Lisa—like user interface to the data. These objects control how the data is displayed and how modifications are performed.

Opening a document brings up a window. The window displays at least one panel. The panel displays one or more panes. Each pane displays a portion of the view associated with a particular panel. The view determines how the data is represented on the screen. Since the view is typically larger than the display area, the pane can scroll over the view. There may be several panes, each displaying a different area of the view.

Closing the document saves the data and the objects that manage and display the data. Thus, the document contains all the information needed to restore its state when later reopened.

[Segment 3]

Creating from the Generic Application

(Getting Started)

Purpose of this segment:

1) To identify the process, docManager, and window methods that you must override to give the ToolKit access to your application's code.
2) To survey the roles of the process, docManager, and window in your documents.
3) To explain how the Generic Application calls application code in response to user events.

How to use this segment:

This segment includes a tutorial and a lab. You should read the tutorial and answer the questions, before proceeding to the "Getting Started" lab.

You should start this segment after the "What is a Document?" segment. You should complete this segment before proceeding to the "BlankStationery" segment.

Terms assumed:

Generic Application, process, window, view, panel, pane, selection, command

INTRODUCTION

The way a ToolKit application works can be stated quite simply — *The Generic Application calls application code in response to user events.* This "application code" is used loosely, since it includes both overidden and inherited methods.

The basic function of the Generic Application is to do as much for you as possible. This includes taking care of standard document behavior: printing, scrolling, and opening, closing, and resizing windows. But, often, it needs to call your code to do so (e.g. drawing a view).

The key to enabling the Generic Application to call your code is to initialize its data structures so that they refer to your objects. Your classes' methods can then be called through those objects.

The following sections describe how to initialize the Generic Application to be able to access your code through your objects.

GETTING STARTED

Every ToolKit application starts by creating a process object. It is from the process that every other object gets created in a new document.

The process is created in the main program block of the application. As you can see from the sample below, the main program exists primarily for creating, initializing, and launching the process.

```
PROGRAM MySample;

USES
          {The following units contain ToolKit procedures and classes}

   {$U UObject   } UObject,          {allocation, deallocation, collection classes}
   {$U QuickDraw } QuickDraw,        {* needed by UDraw *}
   {$U UDraw     } UDraw,            {internal representation to screen conversions}
   {$U UABC      } UABC,             {the classes of the Generic Application}

           {This unit contains the classes (interfaces and implementations) of
            the sample application}

   {$U UMySample } UMySample;


BEGIN
            {One of the application classes is TMyProcess.  It is defined as:
                 TYPE
                     TMyProcess = SUBCLASS OF  TProcess
                       . . .
                     END;}

             {process is a global object reference variable allocated by UObject.
              It is defined as follows:
                 VAR
                     process: TProcess;}

    process := TMyProcess.CREATE;

    process.Commence;              {initializes global variables related to process}

    process.Run;                   {enters the Generic Application event loop}
          {!! The first event queued when a new document is opened is a fileOpen
              event !!}

    process.Complete(TRUE);        {completes with an "all is well" indication}
END.
```

The first object that the process creates is a docManager. The next object created is the window. It is created by the docManager; and is the first object belonging properly to the document.

The window bears the responsibility for creating most of the other document objects. These include: the selection, the view, and the other display objects.

Below are the methods that must be overridden to create the docManager and window objects from your classes.

```
        {creates a new process}
    FUNCTION  TProcess.CREATE: TProcess;

        {returns a newly created docmanager}
    FUNCTION  TProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
                                        : TDocManager;


        {creates a new docmanager}
    FUNCTION  TDocManager.CREATE (object: TObject; itsHeap: THeap; itsPathPrefix:
TFilePath)
                                     : TDocManager;

        {returns a newly created window}
    FUNCTION  TDocManager.NewWindow (heap: THeap; wmgrID: TWindowID): TWindow;



        {creates a new window}
    FUNCTION  TWindow.CREATE (object: TObject;  itsHeap: THeap; wmgrID: TWindowID):
TWindow;
```

Links Between Document Objects

view

panel

view

view

selection

panel

selection

panel

panes

window

window

list

list

panels

selectPanel

window

selectWindow

lastCmd

panel

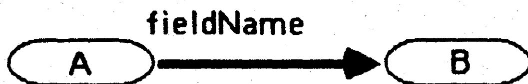
pane

view

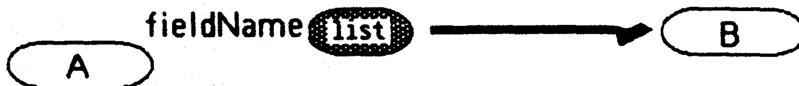
command

image

KEY:



A can access *B* through *fieldName*



A can access *B* through an element of the list *fieldName*

The method below creates the initial display objects (panel, pane, and view) and the selection. This, along with the **CREATE** method below, must be overridden to create a view object from your **TView** subclass.

```
{initializes the document blank stationery}
PROCEDURE TWindow.BlankStationery;


{creates a new view}

FUNCTION  TView.CREATE (object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent:
LRect; itsPrintManager: TPrintManager; itsDfltMargins: LRect; itsFitPagesPerfectly: BOOLEAN;
itsRes: Point; itsMainView: BOOLEAN): TView;
```

BlankStationery plays a major role in initializing the data structures of the Generic Application. *The next segment, "BlankStationery", covers this in more detail.* Yet, these so—called "data structures" are actually a network of cross—referencing objects. What **BlankStationery** does is to create view, panel, pane, and selection objects which cross—reference each other and the window.

The key to having the Generic Application call your code is to insure that the view (and, later on, the selection) is created from one of your classes.

LINKS AMONG DOCUMENT OBJECTS

The slide *Links Between Document Objects* depicts the linkages among document objects. They include: the view, the panel, the pane, the window, the selection, and the command. All of the objects, except for window and command, are created by the window's **BlankStationery** method.

A detailed account of the order of objects created and linked by **BlankStationery** follows.

1) The panel is created as an instance of TPanel.
2) An empty list for the panes is created.
3) An initial pane is created, and appended to the list.
4) The panel appends itself to the window's list of panels. If the panel is the first to be appended, it also becomes the select panel for the window. [The select panel is discussed in the segment on selections.]
5) The view is created as an instance of a subclass of TView.
6) The panel points to the view.

The pane points to the view.

The view points to the panel.

7) An instance of TSelection (the initial selection) is created.

The panel points to that selection.

The selection points to the view and to the panel.

THE STARTING THREE: Process, DocManager, Window

These three objects get your application started; and they play other important roles as well.

process

The process is the heart of the application. There is one and only one process in any ToolKit application.

The process creates one docManager to be responsible for each document. The process, while active, is also owner of the clipboard and the menu bar.

The primary purpose of the process is to direct user events (such as mouse presses, key downs, menu and file events) to the appropriate recipient. Possible recipients are:

the window of the current document,

the dialog box of the active document,

the docManager of the current document,

the selection of the select panel of the current window,

the menu bar,

or the clipboard.

docManager

The primary functions of the docmanager are: to manage the memory used by its document, and to respond to all file events affecting the document.

There is exactly one docmanager for each document owned by a process. The docmanager is responsible for opening, setting aside, resuming, and closing the document.

If the document is newly created, the docmanager allocates space. It then tells the document's window to execute its **BlankStationery** method to create the initial display objects.

window

Each document has a window. Active documents may have an additional window called a dialog box. The window defines the area that the document occupies on the desktop.

The window transmits mouse events received from the process to one or more of its panels. From the panel, these events reach the current selection or view.

USER EVENTS

The heart of the Generic Application is an event loop that identifies and routes user events. Your application's code will be called to process some of these events. The slides, *The Flow of User Events*, indicate some of the methods in your code that could be called.

User events are queued up by the Window Manager and the Filer.

The Window Manager routes mouse downs and key presses to the appropriate application. It sends update events to applications whose documents become exposed on the desktop. Activate events are sent when the user clicks into an inactive window or clicks out of an active one.

*The Filer sends file events, such as **fileOpen** or **fileClose**, to the appropriate application.*

Most user events are handled automatically by the Generic Application. These include: file, menu *(mouse presses on the menu bar)*, update, and activate events. Those that remain, the mouse and key events, are intended to be handled by your application.

After processing an event the Generic Application does something very important. It updates the window. A window update refreshes and redraws the document's display as needed. It is during the update that your application's display code is called. The update tells your view to draw, and your selection to highlight.

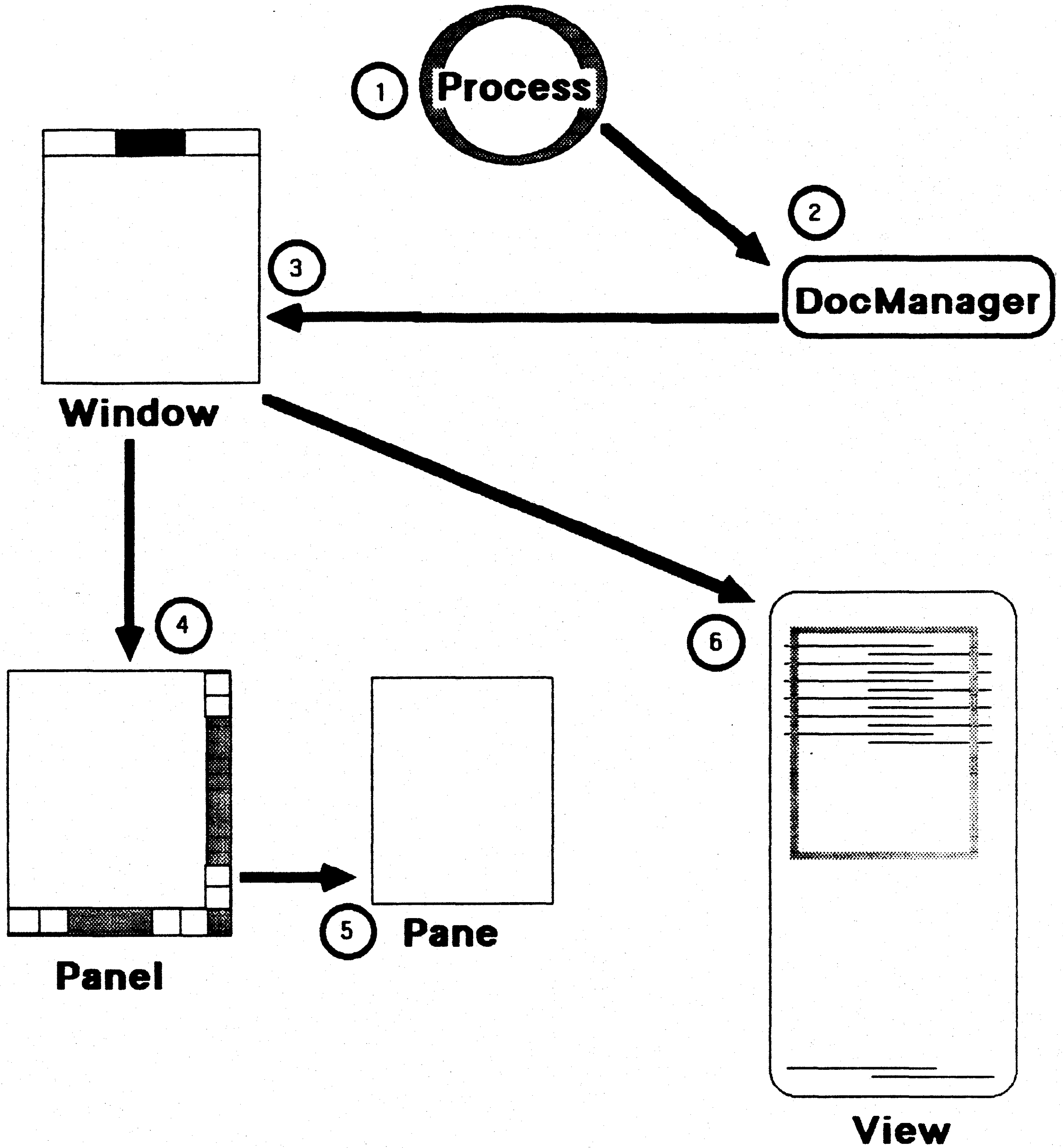
TRUE NATURE OF THE GENERIC APPLICATION

Your application does not call its own display code. The Generic Application does it for you, automatically. This is just one many instances when your code is called automatically. Such is the nature of the Generic Application.

Unlike other programming environments, where you determine when your code will be called, the ToolKit is quite different. Mastering the ToolKit boils down to knowing how and when the Generic Application will call you.

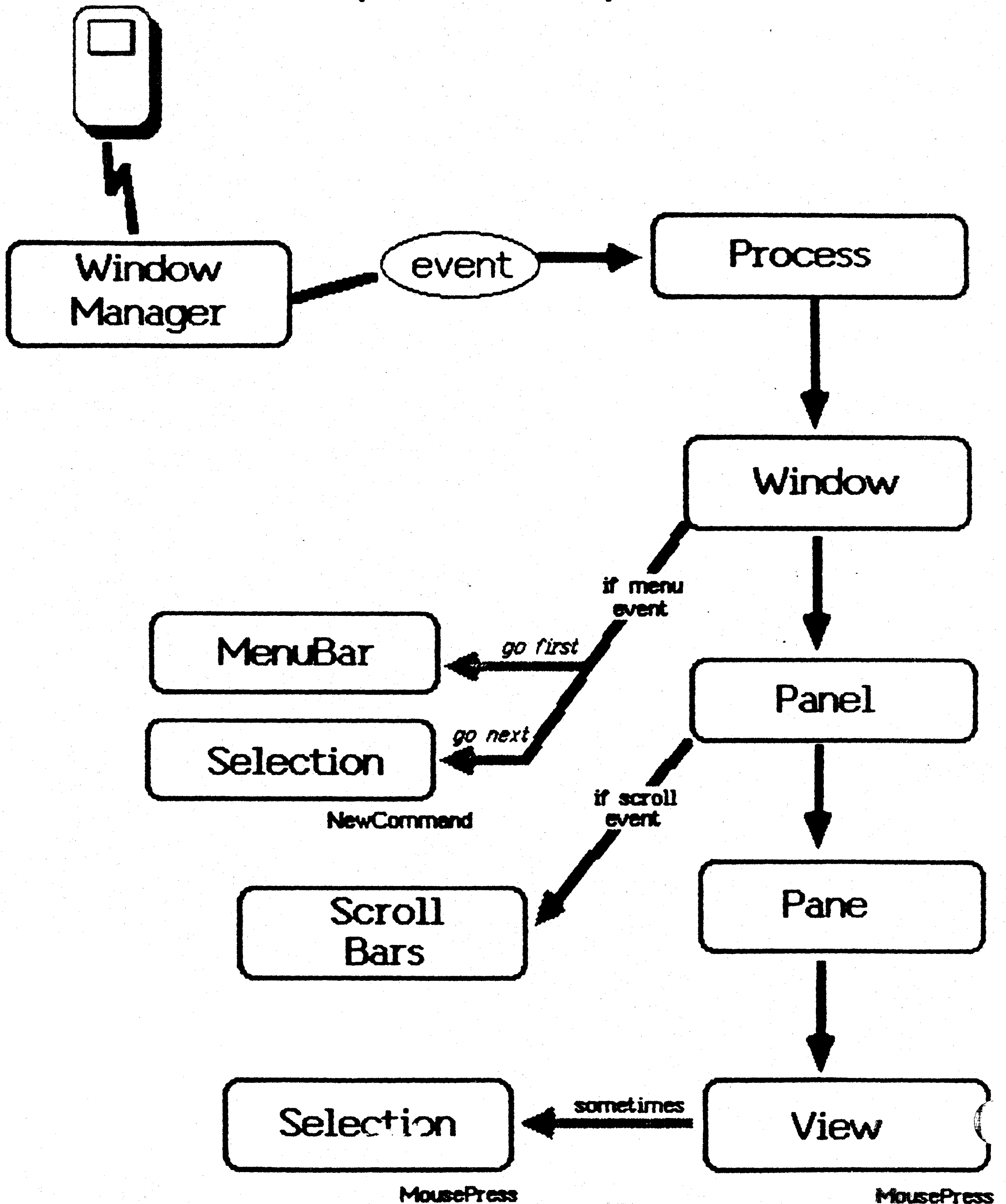
In the accompanying diagrams, "Window" represents your subclass of TWindow, while "TWindow" means the generic class definted in the ToolKit. These diagrams are conceptual. A more detailed diagram is found on the Flow of Control Poster. The ultimate authority is, of course, the ToolKit source code.

Order of Creation



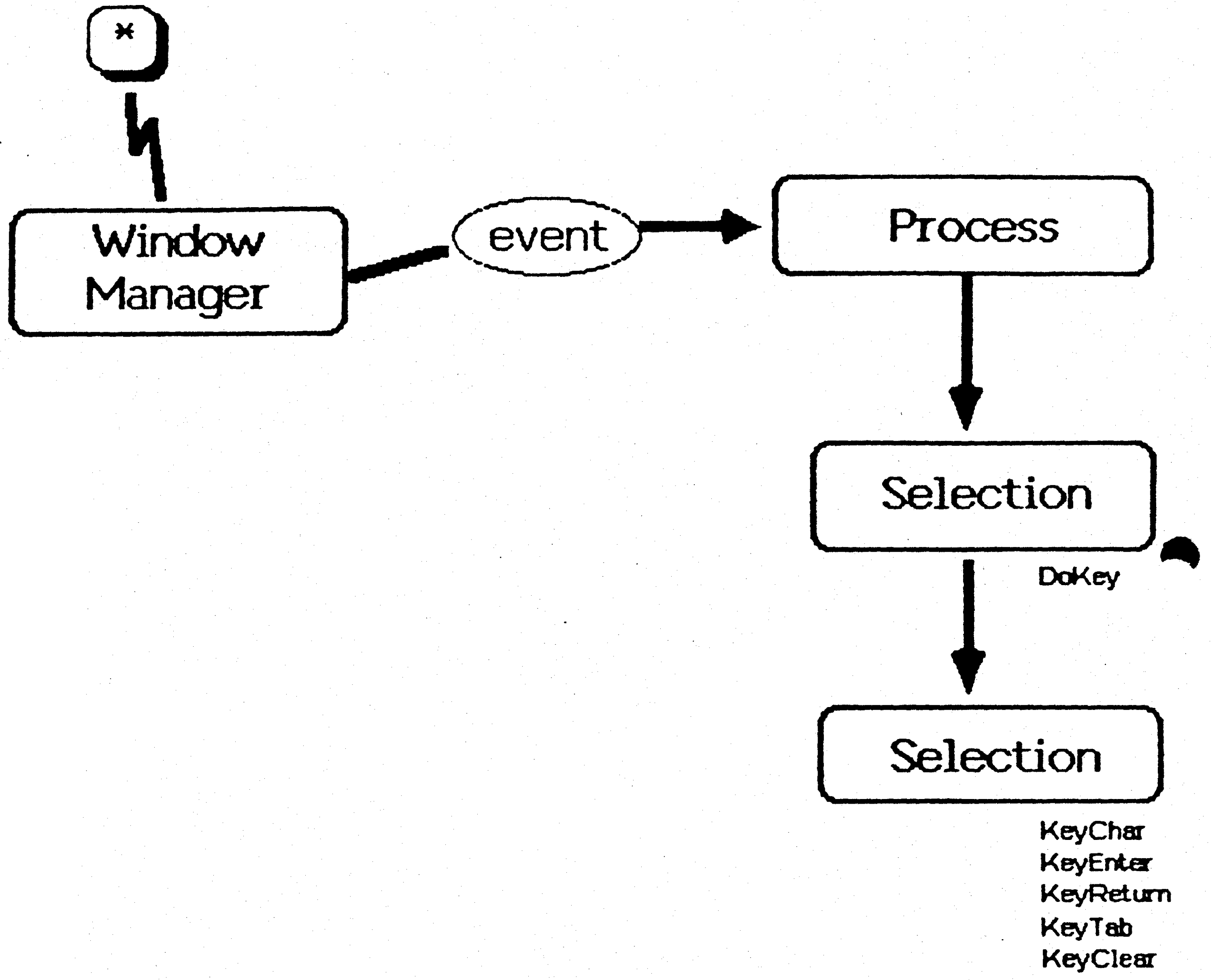
The Flow of User Events

(mouse downs)



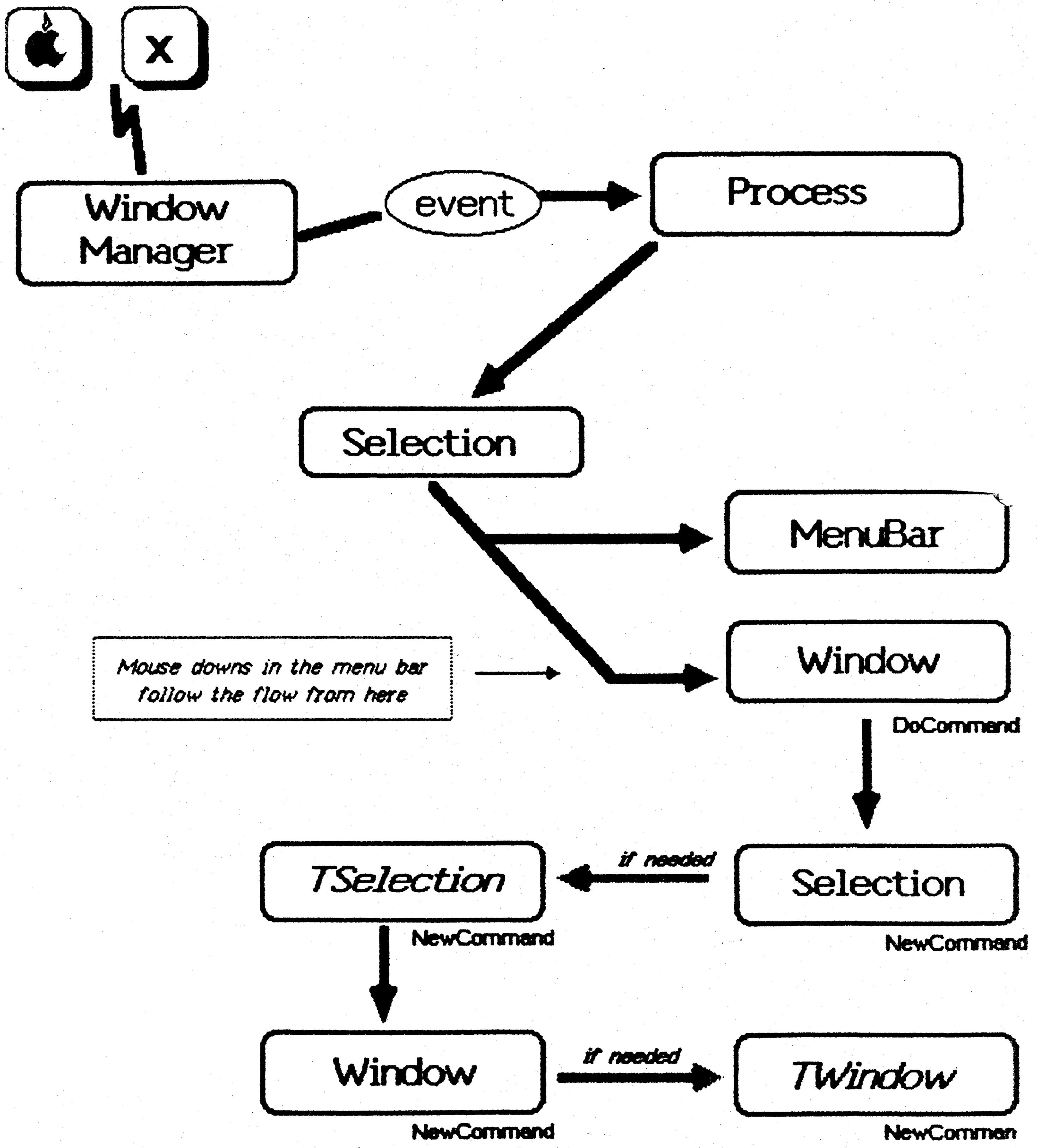
The Flow of User Events

(key presses)



The Flow of User Events

(applekey presses)



Questions:

1) What is the relationship between the window and your view? What role do updates play?
2) Which is created first — the docManager or window? Why?
3) Which is the first application object to receive events?
4) Where are *mouse downs* routed?

Getting Started Lab

Purpose of the lab:

1) To provide hands—on experience compiling and installing a simple ToolKit application.

What you are about to do:

1) Review the "Code components of a ToolKit application".
2) Scan the listings for the four files in the sample application base. These are included in the appendix, "Code Samples for this Segment".
3) Copy the following files onto your prefix volume:

   U1Boxer.TEXT

   U1Boxer2.TEXT

   M1Boxer.TEXT

   P1Boxer.TEXT

4) Compile, install, and run the sample application, **1Boxer**. Use **41** as the tool number. *Appendix A (turn page) describes how to compile and install a ToolKit application.*
   a) Split the panel into panes.
   b) Scroll to the end of the view.

The materials you will start with:

1) The code listings for the sample application base *[see "Code Samples for this Segment]*.
2) The following application source files on your disk (or diskette):

   U1Boxer.TEXT

   U1Boxer2.TEXT

   M1Boxer.TEXT

   P1Boxer.TEXT

3) The ToolKit (must be installed on your disk).

CODE COMPONENTS OF A TOOLKIT APPLICATION

The code for a ToolKit application is typically partitioned into four files. The file naming conventions, below, use *YourApp* as the name of the application. Typical contents of each of the four files is included.

The main program outer block (This file is named M *YourApp*.TEXT).

This file lists the units used, and includes a 4 line outer block. One of the units must be U *Your*App.

The application interface part (This file is named U *YourApp*.TEXT).

This file is the interface for the Pascal unit, UYourApp. This file has an include line for the file UYourApp2.TEXT.

The application implementation part (This file is named U *YourApp2*.TEXT).

This file is the implementation for the unit, UYourApp.

The phrase file (This file is named P *YourApp*.TEXT).

This file contains the application's menus, warnings and error messages.

Question on the lab:

You compile a new ToolKit application named MyApp. You create three files named: MMyApp.TEXT, UMyApp.TEXT, and UMyApp2.TEXT. You make and install the program correctly, but it doesn't run. What is the problem?

Appendix A:
Building and Installing a ToolKit Application

Building an Application

To build a ToolKit application, you use the MAKE exec file. This is supplied with your ToolKit system files.

The MAKE exec command line accepts several arguments. For most applications you need to supply only the first two: the application name, and the tool number. For example, the Workshop command line below will build an application named 1Boxer and save the object file as Tool 41 on the desktop.

```
R<MAKE(1Boxer,41)
```

The above command causes MAKE to check for any changes in any of the following four files since the last build:

```
M1Boxer.TEXT
P1Boxer.TEXT
U1Boxer.TEXT
U1Boxer2.TEXT
```

Since, the creation dates of the various files are checked to determine what needs to be recompiled, make sure that your clock is set correctly.

The third argument to MAKE is the volume to put the tool on. The default is the prefix volume. The final four arguments specify any additional units or building blocks that your application uses. The order of these units is critical. With the exception of the application unit, a listed unit must preceec any that refer to it.

When you build your application for the first time, be sure to delete any old copy of the tool (eg. {T41}OBJ) and the phrase file (eg. {T41}PHRASE). This forces the exec file to relink your application.

Installing an application

Once you have built your application, you must install it on the desktop with the INSTALL program. Just run the program and answer the dialog.

For example, if your application was linked as {T41}OBJ, and you wish to install it as the Sample tool on PARAPORT, then you would supply the following responses to the dialog:

INSTALL prompt	your response
device to install tool on:	PARAPORT
tool id number:	**41**
tool creates documents?	[press **RETURN** (for yes)]
tool handles more than one document at a time?	[press **RETURN** (for no)]
change initial stationery rectangle?	[press **RETURN** (for no)]
tool name:	**Sample**

Code Sample
for this
Segment

```
41
SLOT2CHAN1
;no assembler files
$
;no building blocks
$
;no links
$
y
y
n
Boxer Number 1
```

```
;     PBOXER.TEXT    for  Boxer
;Phrase file for Boxer class example
1
3
2500
$-#BOOT-TK/PABC

; Apple building block phrase files can be included here

1000
1Boxer

; Other application alerts can be included here, numbered between 1001 and 32000

0

1
File/Print
Set Aside Everything#101
Set Aside#102
-
Save & Put Away#103
Save & Continue#107
Revert to Previous Version#108
-
Format for Printer ...#104
Print ...#105
Monitor the Printer ...#106

100
Buzzwords
Set Aside ↑Document↑#109

0
```

```
PROGRAM M1Boxer;

USES
    {$U UObject     } UObject,

  {$IFC libraryVersion <= 20}
    {$U UFont}        UFont,
  {$ENDC}

    {$U QuickDraw   } QuickDraw,
    {$U UDraw       } UDraw,
    {$U UABC        } UABC,

    {$U U1Boxer     } U1Boxer;

CONST

    phraseVersion = 1;

BEGIN

    process := TBoxProcess.CREATE;
    process.Commence(phraseVersion);
    process.Run;
    process.Complete(TRUE);

END.
```

```
1    1 --       {LisaBoxer: box-drawing application for the Tool Kit}
1    2 --       {Copyright 1983, Apple Computer Inc.}
1    3 --
1    4 --   UNIT U1Boxer;
1    5 --
1    6 --   INTERFACE
1    7 --
1    8 --   USES
1    9 --       {$U UObject}            UObject,
1   10 --       {$U QuickDraw}          QuickDraw,
1   11 --       {$U UDraw}              UDraw,
1   12 --       {$U UABC}               UABC;
1   13 --
1   14 --   TYPE
1   15 --
1   16 --        TBoxProcess = SUBCLASS OF TProcess
1   17 --
1   18 --          {Creation/Destruction}
1   19 --           FUNCTION  TBoxProcess.CREATE: TBoxProcess;
1   20 --           FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
1   21 --                                                      : TDocManager; OVERRIDE;
1   22 --            END;
1   23 --
1   24 --
1   25 --       TBoxDocManager = SUBCLASS OF TDocManager
1   26 --
1   27 --         {Creation/Destruction}
1   28 --           FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
1   29 --                                                 : TBoxDocManager;
1   30 --           FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID): TWindow; OVERRIDE;
1   31 --           END;
1   32 --
1   33 --
1   34 --
1   35 --       TBoxWindow = SUBCLASS OF TWindow
1   36 --
1   37 --         {Creation/Destruction}
1   38 --           FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
1   39 --
1   40 --         {Document Creation}
1   41 --           PROCEDURE  TBoxWindow. BlankStationery; OVERRIDE;
1   42 --
1   43 --           END;
1   44 --
1   45 --
1   46 --
1   47 --       TBoxView = SUBCLASS OF TView
1   48 --
1   49 --         {Creation/Destruction}
1   50 --           FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
1   51 --                                            : TBoxView;
1   52 --
1   53 --         {Display}
1   54 --           PROCEDURE  TBoxView. Draw; OVERRIDE;
1   55 --          END;
1   56 --
1   57 --
1   58 --   IMPLEMENTATION
1   59 --
1   60 --     {$I U1Boxer2.text}
2    1 --   {U1Boxer2}
2    2 --
2    3 --   METHODS OF TBoxProcess;
2    4 --
2    5 --
2    6 -- A     FUNCTION  TBoxProcess.CREATE: TBoxProcess;
2    7 0- A     BEGIN
2    8 --           {$IFC fTrace}BP(11);{$ENDC}
2    9 --           SELF := TBoxProcess(TProcess.CREATE(NewObject(mainHeap, THISCLASS), mainHeap));
2   10 --           {$IFC fTrace}EP;{$ENDC}
2   11 -0 A     END;
2   12 --
2   13 --
2   14 -- A     FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN): TDocManager;
2   15 0- A     BEGIN
2   16 --           {$IFC fTrace}BP(11);{$ENDC}
2   17 --           NewDocManager := TBoxDocManager.CREATE(NIL, mainHeap, volumePrefix);
2   18 --           {$IFC fTrace}EP;{$ENDC}
2   19 -0 A     END;
2   20 --
2   21 --   END {Methods of TBoxProcess};
2   22 --
2   23 --
2   24 --
2   25 --   METHODS OF TBoxDocManager;
2   26 --
2   27 --
2   28 -- A     FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
2   29 --                                       : TBoxDocManager;
2   30 0- A     BEGIN
2   31 --           {$IFC fTrace}BP(11);{$ENDC}
2   32 --           IF object = NIL THEN
2   33 --               object := NewObject(itsHeap, THISCLASS);
2   34 --           SELF := TBoxDocManager(TDocManager.CREATE(object, itsHeap, itsPathPrefix));
2   35 --           {$IFC fTrace}EP;{$ENDC}
2   36 -0 A     END;
2   37 --
2   38 --
2   39 -- A     FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID): TWindow;
2   40 0- A     BEGIN
2   41 --           {$IFC fTrace}BP(11);{$ENDC}
2   42 --           NewWindow := TBoxWindow.CREATE(NIL, heap, wmgrID);
2   43 --           {$IFC fTrace}EP;{$ENDC}
2   44 -0 A     END;
2   45 --
2   46 --   END {METHODS OF TBoxDocManager};
2   47 --
2   48 --
2   49 --
2   50 --   METHODS OF TBoxWindow;
```

```
2    51 --
2    52 --
2    53 -- A      FUNCTION  TBoxWindow.CREATE(object: TOBject;  itsHeap: THeap;  itsWmgrID: TWindowID): TBoxWindow;
2    54 0- A      BEGIN
2    55 --            {$IFC fTrace}BP(10); {$ENDC}
2    56 --            IF object = NIL THEN
2    57 --               object := NewObject(itsHeap, THISCLASS);
2    58 --            SELF := TBoxWindow(TWindow.CREATE(object,  itsHeap,  itsWmgrID,  TRUE));
2    59 --            {$IFC fTrace}EP; {$ENDC}
2    60 -0 A      END;
2    61 --
2    62 --
2    63 --
2    64 -- A      PROCEDURE TBoxWindow.BlankStationery;
2    65 --       VAR  viewLRect: LRect;
2    66 --            panel:  TPanel;
2    67 --            boxView:  TBoxView;
2    68 --
2    69 0- A      BEGIN
2    70 --           {$IFC fTrace} BP(10);  {$ENDC}
2    71 --                                     {set the view extent LRect}
2    72 --           SetLRect(viewLRect, 0, 0, 5000, 3000);
2    73 --
2    74 --           panel := TPanel.CREATE(NIL,  SELF.Heap,  SELF,  0,  0,
2    75 --                                 [aBar, aScroll, aSplit],  [aBar, aScroll, aSplit]);
2    76 --
2    77 --                                     {initialize the boxView}
2    78 --           boxView := TBoxView.CREATE(NIL,  SELF.Heap,  panel,  zeroLRect);
2    79 --           {$IFC fTrace} EP;  {$ENDC}
2    80 -0 A      END;
2    81 --
2    82 --    END {Methods of TBoxWindow};
2    83 --
2    84 --
2    85 --
2    86 --
2    87 --    METHODS OF TBoxView;
2    88 --
2    89 -- A      FUNCTION  TBoxView.CREATE(object: TObject;  itsHeap: THeap;  itsPanel: TPanel;  itsExtent: LRect)
2    90 --                                : TBoxView;
2    91 0- A      BEGIN
2    92 --            {$IFC fTrace}BP(11); {$ENDC}
2    93 --            IF object = NIL THEN
2    94 --               object := NewObject(itsHeap,  THISCLASS);
2    95 --
2    96 --            SELF := TBoxView(itsPanel.NewView(object,  itsExtent,  NIL,  stdMargins,  FALSE));
2    97 --            {$IFC fTrace}EP; {$ENDC}
2    98 -0 A      END;
2    99 --
2   100 -- A      PROCEDURE TBoxView.Draw;
2   101 0- A      BEGIN
2   102 --            {$IFC fTrace}BP(10); {$ENDC}
2   103 --            {$IFC fTrace}EP; {$ENDC}
2   104 -0 A      END;
2   105 --
2   106 --    END {METHODS OF TBoxView};
2   107 --
2   108 --
1    61 --
1    62 --    END.
```

```
 1. u1boxer.TEXT
 2. U1Boxer2.text
                     41•( 1)   54•( 1)

-B-
 BlankStationery     41•( 1)   64•( 2)

-C-
 CREATE              19 ( 1)   28 ( 1)   38 ( 1)   50 ( 1)    6•( 2)    9 ( 2)   17 ( 2)   28•( 2)   34 ( 2)   42 ( 2)
                     53•( 2)   58 ( 2)   74 ( 2)   78 ( 2)   89•( 2)

-D-
 Draw                54•( 1)  100•( 2)

-N-
 NewDocManager       20•( 1)   14•( 2)   17=( 2)
 NewWindow           30•( 1)   39•( 2)   42=( 2)

-Q-
 QuickDraw           10•( 1)

-T-
 TBoxDocManager      25•( 1)   29 ( 1)   17 ( 2)   25•( 2)   29 ( 2)   34 ( 2)
 TBoxProcess         16•( 1)   19 ( 1)    3•( 2)    6 ( 2)    9 ( 2)
 TBoxView            47•( 1)   51 ( 1)   67 ( 2)   78 ( 2)   87•( 2)   90 ( 2)   96 ( 2)
 TBoxWindow          35•( 1)   38 ( 1)   42 ( 2)   50•( 2)   53 ( 2)   58 ( 2)
 TDocManager         21 ( 1)   25 ( 1)   14 ( 2)   34 ( 2)
 TProcess            16 ( 1)    9 ( 2)
 TView               47 ( 1)
 TWindow             30 ( 1)   35 ( 1)   39 ( 2)   58 ( 2)

-U-
 U1Boxer              4•( 1)
 UABC                12•( 1)
 UDraw               11•( 1)
 UObject              9•( 1)
*** End Xref: 19 id's   67 references       [420232 bytes/4980 id's/42569 refs]
```

[Segment 4]

BlankStationery: The View of a New Document

Purpose of this segment:

1) To describe how a new document is initialized.
2) To introduce the coordinate system of the view.
3) To describe how a document's contents are spatially arranged and displayed.

How to use this segment:

This is the fourth segment in the ToolKit architecture self-paced series. This segment should be started after the segment "Creating from the Generic Application". It should be completed before moving on to the "Intro to Boxer" segment.

THE VIEW OF A NEW DOCUMENT

Each application initializes new documents in its own way. The document's display is drawn within the window by the view. The view generates the displayed representation of a document's data.

The initial views of new documents can vary widely. For example, tearing off a sheet of LisaWrite paper yields a completely blank view. Tearing off a sheet of LisaDraw paper displays a drawing palette and a grid.

In each of your ToolKit applications, you will need to determine how new documents are to be displayed. This segment presents a BlankStationery boilerplate to help you do so.

BLANK STATIONERY

The objects in a ToolKit-based document are Clascal objects. As such each object has a class. The Generic Application contains the superclasses for many of a typical document's classes. For instance, window objects are of the Generic Application class, TWindow. Views are of the class, TView.

A special method of the window, BlankStationery, is called by the Generic Application to initialize the view of a new document. Your application will subclass TWindow and override BlankStationery to initialize its documents.

INITIALIZING A DOCUMENT

the user interface

A user tears off a sheet from an application's stationery pad. An icon named "Untitled" is displayed on the desktop; the user may edit the name. The user opens the icon and the application proceeds to create and initialize a new document.

The window frame is painted for the new document. If the application that owns the stationery is not already running, it is started at that time.

the internals

A *process* object is created when the application starts. Once it has initialized itself, the process creates a *docManager* object for each opened window. *Both the process and the docManager objects reside on a separate heap — the process heap.*

The first things the docmanager does are to: make the document heap, and place a new *window* object on that heap. *There is a separate document heap for each window.*

The window's BlankStationery method creates the panels and the initial view of the document. BlankStationery must do the following things to initialize the view:

a) create a panel. *The panel, in turn, creates an initial pane.*

b) create a view. *The new view is then linked to the panel.*

c) create an initial selection *The default is an instance of TSelection.*

After returning from BlankStationery, the Generic Application tells the window to activate itself. This causes the window to update. The window update frames the panel and pane, then draws the view.

After drawing the view, the Generic Application highlights the window's title bar, and waits for the next user event. The user is now able to work on the new document.

THE VIEW COORDINATE SYSTEM

some definitions

inner rect: the coordinates of the inner boundary of an object.

> Typically this is one pixel less all around than the outer boundary. Inner rects are expressed in 16-bit *QuickDraw* coordinates.

LRect: the 32-bit analogue of QuickDraw's Rect type.

> Each coordinate is a long integer.

extent LRect: an LRect defining the logical boundaries of the view.

> The extent LRect imposes a 32-bit coordinate system upon the view. Only a portion of the extent LRect may be visible at any moment. [See the diagram "View vrs. Pane".]

viewed LRect: the portion of the view visible through a specific pane.

> Each pane has an inner rect and a viewed LRect. The viewed LRect encloses that portion of the view that is to be displayed in the pane's inner rect.
>
> The pane performs a mapping from the 32-bit coordinates of the view to the 16-bit coordinates of the window's grafport.
>
> The coordinate system of the viewed LRect is the same as that of the view's extent LRect. [See the diagram "View vrs. Pane".]

the mechanics of drawing the view

If you recall, QuickDraw is the Apple 32 graphics utility. QuickDraw performs all of its drawing in a 16-bit coordinate space. This means that drawing is limited to between -32768 and 32767 units in either the X or Y direction.

In a typical document, the data objects are arranged by coordinates. For example, a box enclosing text may be defined by the Rect [100, 100, 500, 300]. The first line of text might start at [110, 120]. This is done both to interrelate data objects spatially within the document, and to allow objects to be drawn inside the window's grafport.

Unfortunately, a 16-bit coordinate space supports, for example, no more than 40 pages of text data. There are no more unused points in the coordinate space to associate new text with.

The ToolKit remedies that problem by supporting 32-bit coordinates. In the case of text, a 32-bit coordinate space supports more than a million pages. *This is much more than a document can actually hold.*

ToolKit views support objects specified in 32-bit coordinates. The view *extent LRect* describes the subset of the 32-bit coordinate space in which objects can be defined and through which the view can be scrolled.

Drawing objects expressed in 32–bit coordinates into a 16–bit grafport, though, poses a bit of a problem. A translation must be performed upon the 32–bit coordinates before drawing is possible. That is where the pane steps in.

The pane displays a subset of the view. The pane's *inner rect* resides in the 16–bit coordinate space of the window's grafport. To display 32–bit data objects, the pane also has a *viewed LRect*. This viewed LRect is a viewport into the view's coordinate space. The pane's viewed LRect is a subset of the view's extent LRect.

Typically with the same dimensions as the inner rect, the viewed LRect makes possible a one–to–one mapping of points from a 32–bit view to the 16–bit screen.

Scrolling through the document changes the coordinates of the viewed LRect, enabling other portions of the view to be mapped to the screen.

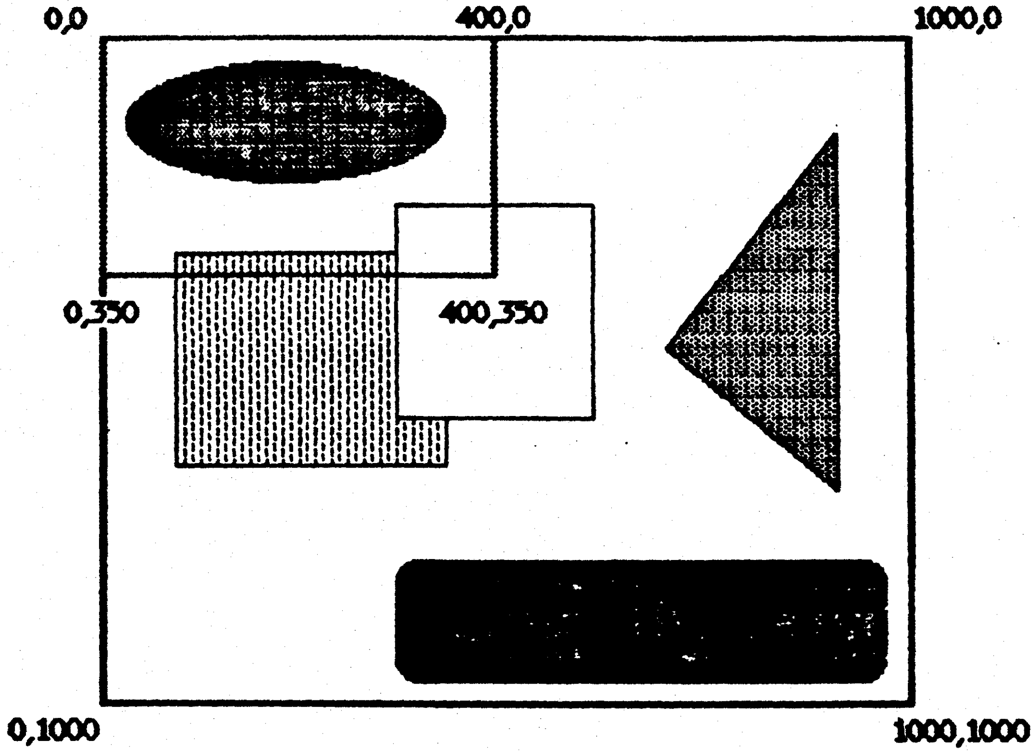
SAMPLE CODE

Sample code for a window BlankStationery method and view CREATE function are listed below. The listing assumes that TMyWindow is created as a subclass of TWindow; and that TMyView is created as a subclass of TView.

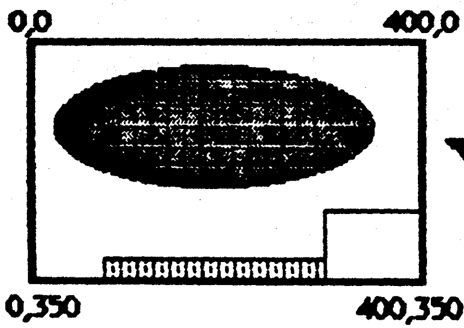
```
PROCEDURE {TMyWindow.} BlankStationery;
VAR  myView:        TMyView;
     panel:         TPanel;
     minHeight,
     minWidth:      INTEGER;
     extentLRect:   LRect;
     docHeap:       THeap;
     itsPrintMgr:   TPrintManager;        {the print manager for the view}

BEGIN
                 {Bear in mind that SELF represents the window object}

 docHeap := SELF.Heap;

                 {create the panel with two scroll bars.  The pane in the panel will
                  be able to be scrolled and split both horizontally and
                  vertically.}
 minHeight := 0;                  {minimum height of the panel inner rect}
 minWidth := 0;                   {minimum width of the panel inner rect}
 panel :=  TPanel.CREATE(NIL, docHeap, SELF, minHeight, minWidth,
                         [aBar,aScroll,aSplit], [aBar,aScroll,aSplit]);

                 {create the view}
                       {define the view extent rect}
 SetLRect(extentLRect, {its left}, {its top}, {its right}, {its bottom});
 myView := TMyView.Create(NIL, docHeap, panel, extentLRect);

END  {TMyWindow.BlankStationery};

{||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||}


FUNCTION {TMyView.} CREATE {(object: TObject; itsHeap: THeap; panel: TPanel; itsExtent: LRect)
                        : TMyView}
CONST  itsFitPerfectlyOnPages = TRUE;   {will resize the view extent rect to fit perfectly
                                         on pages}
```

View vrs. Pane

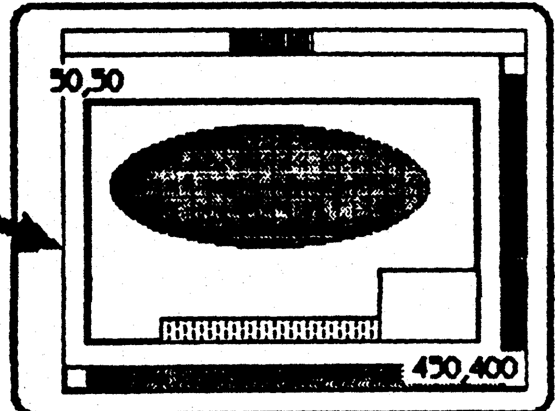
View
ExtentLRect
(32-bit coordinates)



Pane
ViewedLRect
(32-bit view coordinates)



Pane
InnerRect
(16-bit window coordinates)



mapping from view to window

```
VAR  itsPrintMgr:   TPrintManager;
   {stdMargins:    LRect;              (global margin settings in screen pixels)  }

BEGIN

 IF object = NIL THEN   {create a new object of this class}
                              {The compiler construct, THISCLASS, supplies the class pointer
                               automatically}
    object := NewObject(itsHeap, THISCLASS);

                           {create a default print manager}
                                 {This print manager will allow general printing capabilities,
                                  except for margins and headings}
 itsPrintMgr := TPrintMgr.CREATE(NIL, docHeap);

                          {create a new standard view}
                                 {the view will be created as a main view (not a printed or
                                  paginated view) with horizontal and vertical resolutions
                                  the same as the screen}

 SELF := TMyView(panel.NewView(object, itsExtent, itsPrintMgr, stdMargins,
                                                               itsFitPerfectlyOnPages));
                    {!! NOTE !!
                                  The construct, TMyView( … ), is a type cast
                                  A type cast treats its argument as if it were of the specified
                                  class or type.  The method, TPanel.NewView, returns a
                                  value of class TView.  This result is type cast, because
                                  SELF must be assigned a value of class TMyView.

END  {TMyView.Create};
```

[Segment 5]

Intro to the Boxer Application

Purpose of this segment:

1) To introduce the Boxer application.
2) To demonstrate how to draw boxes.

How to use this segment:

This segment includes a tutorial and a lab. You should read the tutorial and answer the questions before proceeding to the lab.

You should start this segment after the "BlankStationery" segment. You should complete this segment before the "Selections & Highlighting" segment.

INTRODUCTION TO THE BOXER APPLICATION

The Boxer application displays and edits boxes. We shall implement Boxer in stages, over the next seven segments.

In Boxer documents the data are boxes, boxes, and boxes. A *list* is used to store the boxes. The list is maintained as a field in the view.

With a list we are able to access any box easily. Using a *list scanner* we can access each box in sequence. The list scanners also provides easy insertion and deletion of boxes *(or any object)*.

Each box is represented internally as an object with several fields. A box's primary attribute, its dimensions, is conserved as a ToolKit LRect. *LRects are uniquely defined by four long integer coordinates.*

As with any document, we must make provision for displaying the data. To display a box, we just draw a frame around its LRect. To display the data in the view, we use a list scanner to step through the list and draw each box. Actual drawing occurs in the view's panel.

Though each box is always *drawn*, at any given time, some boxes may not be displayed. The actual display is determined by the panel's panes. Within a pane, only those boxes intersecting the pane's viewedLRect are displayed. And of those boxes, only that portion contained within the viewed LRect is drawn on the screen. The pane's viewed LRect acts as a temporary clip region. This mechanism is called *focusing*. *When drawing in a panel, the Generic Application focuses each pane before changing bits on the screen.*

Editing boxes follows the normal Lisa model. As with other data, boxes must be selected to be operated upon. Among the editing operations we implement for

boxes are: box moves, color changes, duplicate, and cut & paste. *Editing operations are implemented in subsequent segments.*

BOX OBJECTS

Box objects have certain attributes. These are included in the table below.

Attributes of box objects

modifiable	constant
color	
shape LRect	size

The *color* is one of five standard colors (white, gray, light gray, dark gray, black).

The *shape LRect* determines both the location within the view space and the *size* of the box. *[See the "ToolKit Reference Manual" for more details on LRects].* The shape LRect can be offset, (this is how the box is moved within the view), but it's dimensions remain constant.

A box's attributes are saved in its fields. This is indicated by the partial interface below.

```
TYPE
    TColor = (colorWhite, colorLtGray, colorGray, colorDkGray, colorBlack);

    TBox = SUBCLASS OF TObject

     {fields}
      shapeLRect:  LRect;
      color:       TColor;

     {methods}
      ...
     END;
```

As can be seen from the interface, box objects are instances of the class, TBox.

IMPLEMENTATION STRATEGY

In this segment we create two boxes, and insert them into the view. The view maintains the boxes in an *indexList*. Each created box is appended to the list. Initially, a box's color and shape LRect will not change.

We have built upon the application code presented in the "Generic Application" segment.

We have created a new class, TBox. Boxes are instances of TBox. Within this class we have added the method {TBox.} Draw to enable each box to draw itself. *The implementation for TBox is included in the appendix "Code Sample for this Segment".*

In addition to adding TBox, other application changes were made, most notably in the view. The modifications are listed below:

New Classes

[TBox]

```
TBox = SUBCLASS OF TObject

    {variables}
  shapeLRect:  LRect;
  color:       TColor;

  {Creation}
FUNCTION  {TBox.} CREATE (object: TObject; itsHeap: THeap);

  {Display}
PROCEDURE {TBox.} Draw;
END {of TBox};
```

New Methods (for existing classes)

[TBoxView]

```
PROCEDURE {TBoxView.} InitBoxList (itsHeap: THeap);
```

{TBoxView.}InitBoxList creates an index list, then appends two newly created boxes to the list.

Changed Methods

[TBoxWindow]

```
PROCEDURE {TBoxWindow.} BlankStationery;
```

{TBoxWindow.}BlankStationery now creates the view and tells it to initialize its list of boxes.

theory of operation

Because of the minimal user interface, 1Boxer operates quite simply. If initializes the view in BlankStationery, then waits until the window updates to draw the boxes in the view. The flow of control is depicted below:

```
|    TBoxWindow.Update
|        TBoxView.Draw
v            TBox.Draw   (for each box)
```

APPENDING OBJECTS TO A LIST [optional]

Appending objects to a list is fairly straightforward. Below, we show some sample Clascal code that appends boxes to a list.

```
PROCEDURE  InitBoxList (boxList: TList; itsHeap: THeap);
VAR  box:     TBox;

BEGIN
                        {initialize the list with no elements}
    boxList := TList.Create(NIL, itsHeap, 0);

    box := TBox.Create(NIL, itsHeap);  {create a box}
    boxList.InsLast(box);              {append the box to the list}

    box := TBox.Create(NIL, itsHeap);  {create another box}
    boxList.InsLast(box);              {append it to the list}

END;
```

DRAWING OBJECTS FROM A LIST [optional]:

The boxes in the view's list are drawn one at a time. A list scanner is the best way to step through the objects in a list. The code segment below draws each box in the list, boxList.

```
PROCEDURE  Draw (boxList: TList);
VAR   box:  TBox;
      s:    TListScanner;

BEGIN
          {create a list scanner for the List, boxList}
     s := boxList.Scanner;

          {Draw each box in the list}
     WHILE s.Scan(box) DO
          box.Draw;
END;
```

HOW TO DRAW A BOX [optional]

As was stated before, a box is an object with a shape LRect and a color. The shape LRect is the shape of the box. This reduces the problem of drawing a box to drawing an LRect.

Drawing an LRect in the ToolKit is similar to drawing a Rect in QuickDraw. The main difference is that LRects have 32-bit coordinates, rather than the 16-bit coordinates of Rects.

Drawing a shape is a two step process. First you fill the shape, then you frame it. The fill operation uses the ToolKit procedure, FillLRect. The frame operation uses the ToolKit procedure, FrameLRect.

The FillLRect procedure is analogous to the QuickDraw procedure, FillRect. FillLRect takes two parameters — an LRect and a pattern. The pattern must be of the ToolKit type, LPattern. Fortunately, the ToolKit pre-defines the following color LPatterns:

```
[ lPatWhite, lPatBlack, lPatGray, lPatLtGray, lPatDkGray ].
```

To fill an LRect named myLRect with the color black, you would use the following procedure call:

```
FillLRect (myLRect, lPatBlack)
```

The FrameLRect procedure is analogous to the QuickDraw procedure, FrameRect. FrameLRect takes just one parameter, the LRect. To frame the LRect, myLRect, you would use the following call.

```
FrameLRect (myLRect)
```

Intro to Boxer Lab

Purpose of the lab:

To implement a simple Boxer document.

What you are about to do:

1) Read the section "Desired Application Behavior".
2) Scan the listings for the four files in the sample application. These are included in the appendix, "Code Samples for this Segment".
3) Copy the following files onto your prefix volume:

   U2Boxer.TEXT

   U2Boxer2.TEXT

   M2Boxer.TEXT

   P2Boxer.TEXT

4) Compile, install, and run the sample application, 2Boxer. Use 42 as the tool number.

   Try the following:

   a) Split the panel into panes.

   b) Scroll to the end of the view.

5) Try one or both of the following changes:

   — display one gray box and one black box.

   — draw a box that is bigger than the view extent LRect.

Desired application behavior:

After a piece of stationery is torn off of the 2Boxer pad, the new document should display two boxes as shown in the attached screen shot.

The boxes will have the following attributes:

	shape LRect	color
box #1	[(20,20) , (100,100)]	Gray
box #2	[(200,100) , (300,130)]	Gray

Note that the color is set to gray in the method {TBox.} CREATE.

Questions:

1) Think about how you might select a box and indicate that it is selected.

Code Sample
for this
Segment

```
42
SLOT2CHAN1
;no assembler files
$
;no building blocks
$
;no links
$
y
y
n
BoxNum2
```

```
;     PBOXER.TEXT    for  Boxer
;Phrase file for Boxer class example
1
3
2500
$-#boot-tk/PABC

; Apple building block phrase files can be included here

1000
Boxer

; Other application alerts can be included here, numbered between 1001 and 32000

0

1
File/Print
Set Aside Everything#101
Set Aside#102
-
Save & Put Away#103
Save & Continue#107
Revert to Previous Version#108
-
Format for Printer ... #104
Print ... #105
Monitor the Printer ... #106

100
Buzzwords
Set Aside ↑Document↑#109

0
```

```
PROGRAM M2Boxer;

USES
    {$U UObject     } UObject,

  {$IFC libraryVersion <= 20}
    {$U UFont}              UFont,
  {$ENDC}

    {$U QuickDraw   } QuickDraw,
    {$U UDraw       } UDraw,
    {$U UABC        } UABC,

    {$U U2Boxer     } U2Boxer;

CONST

    phraseVersion = 1;

BEGIN

    process := TBoxProcess.CREATE;
    process.Commence(phraseVersion);
    process.Run;
    process.Complete(TRUE);

END.
```

```
1     1 --    UNIT U2Boxer;
1     2 --
1     3 --    INTERFACE
1     4 --
1     5 --    USES
1     6 --        {$U UObject}            UObject,
1     7 --
1     8 --      {$IFC libraryVersion <= 20}
1     9 --        {$U UFont}              UFont,
1    10 --      {$ENDC}
1    11 --
1    12 --        {$U QuickDraw}          QuickDraw,
1    13 --        {$U UDraw}              UDraw,
1    14 --        {$U UABC}               UABC;
1    15 --
1    16 --
1    17 --    CONST
1    18 --        colorWhite = 1;
1    19 --        colorLtGray = 2;
1    20 --        colorGray = 3;
1    21 --        colorDkGray = 4;
1    22 --        colorBlack = 5;
1    23 --
1    24 --
1    25 --    TYPE
1    26 --
1    27 --        TColor = colorWhite..colorBlack; {color of a box}
1    28 --
1    29 --      {New Classes for this Application}
1    30 --
1    31 --        TBox = SUBCLASS OF TObject
1    32 --
1    33 --            {Variables}
1    34 --            shapeLRect:     LRect;
1    35 --            color:          TColor;
1    36 --
1    37 --          {Creation/Destruction}
1    38 --            FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
1    39 --
1    40 --          {Display}
1    41 --            PROCEDURE TBox.Draw;
1    42 --            END;
1    43 --
1    44 --
1    45 --        TBoxView = SUBCLASS OF TView
1    46 --
1    47 --          {Variables}
1    48 --            boxList:        TList;
1    49 --
1    50 --          {Creation/Destruction}
1    51 --            FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
1    52 --                                    : TBoxView;
1    53 --
1    54 --          {Display}
1    55 --            PROCEDURE  TBoxView.Draw; OVERRIDE;
1    56 --            PROCEDURE TBoxView.InitBoxList(itsHeap: THeap);
1    57 --           END;
1    58 --
1    59 --
1    60 --         TBoxProcess = SUBCLASS OF TProcess
1    61 --
1    62 --          {Creation/Destruction}
1    63 --            FUNCTION  TBoxProcess.CREATE: TBoxProcess;
1    64 --            FUNCTION   TBoxProcess. NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
1    65 --                                               : TDocManager; OVERRIDE;
1    66 --            END;
1    67 --
1    68 --
1    69 --        TBoxDocManager = SUBCLASS OF TDocManager
1    70 --
1    71 --          {Creation/Destruction}
1    72 --            FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
1    73 --                                         : TBoxDocManager;
1    74 --            FUNCTION   TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID): TWindow; OVERRIDE;
1    75 --            END;
1    76 --
1    77 --
1    78 --        TBoxWindow = SUBCLASS OF TWindow
1    79 --
1    80 --          {Creation/Destruction}
1    81 --            FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
1    82 --
1    83 --          {Document Creation}
1    84 --            PROCEDURE  TBoxWindow. BlankStationery; OVERRIDE;
1    85 --
1    86 --            END;
1    87 --
1    88 --
1    89 --    IMPLEMENTATION
1    90 --
1    91 --      {$I U2Boxer2.text}
2     1 --    {UBoxer2}
2     2 --
2     3 --    METHODS OF TBox;
2     4 --
2     5 --
2     6 -- A      FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
2     7 0- A      BEGIN
2     8 --            {$IFC fTrace}BP(11);{$ENDC}
2     9 --            SELF := NewObject(itsHeap, THISCLASS);
2    10 --            WITH SELF DO
2    11 1-                BEGIN
2    12 --                shapeLRect := zeroLRect;
2    13 --                color := colorGray;
2    14 -1                END;
2    15 --            {$IFC fTrace}EP;{$ENDC}
2    16 -0 A      END;
2    17 --
2    18 --        {This draw a particular box}
2    19 -- A      PROCEDURE TBox.Draw;
```

```
2    20 --          VAR lPat:    LPattern;
2    21 0- A        BEGIN
2    22 --              {$IFC fTrace}BP(10);{$ENDC}
2    23 --              PenNormal;
2    24 --
2    25 --              IF LRectIsVisible(SELF.shapeLRect) THEN {this box needs to be drawn}
2    26 1-                  BEGIN
2    27 --                {Get a Quickdraw pattern to represent the box's color}
2    28 2-                  CASE SELF.color OF
2    29 --                      colorWhite:      lPat := lPatWhite;
2    30 --                      colorLtGray:     lPat := lPatLtGray;
2    31 --                      colorGray:       lPat := lPatGray;
2    32 --                      colorDkGray:     lPat := lPatDkGray;
2    33 --                      colorBlack:      lPat := lPatBlack;
2    34 --                      OTHERWISE        lPat := lPatWhite;  {this case should not happen}
2    35 -2                      END;
2    36 --
2    37 --                 {Fill the box with the pattern, and draw a frame around it}
2    38 --                  FillLRect(SELF.shapeLRect, lPat);
2    39 --                  FrameLRect(SELF.shapeLRect);
2    40 -1                  END;
2    41 --              {$IFC fTrace}EP;{$ENDC}
2    42 -0 A        END;
2    43 --
2    44 --      END;
2    45 --
2    46 --
2    47 --
2    48 --
2    49 --      METHODS OF TBoxView;
2    50 --
2    51 -- A        FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
2    52 --                                      : TBoxView;
2    53 0- A        BEGIN
2    54 --              {$IFC fTrace}BP(11);{$ENDC}
2    55 --              IF object = NIL THEN
2    56 --                  object := NewObject(itsHeap, THISCLASS);
2    57 --              SELF := TBoxView(itsPanel.NewView(object, itsExtent, TPrintManager.CREATE(NIL, itsHeap),
2    58 --                                          stdMargins, TRUE));
2    59 --              {$IFC fTrace}EP;{$ENDC}
2    60 -0 A        END;
2    61 --
2    62 --
2    63 --      {This draws the list of boxes}
2    64 -- A        PROCEDURE TBoxView.Draw;
2    65 --          VAR box:     TBox;
2    66 --              s:       TListScanner;
2    67 0- A        BEGIN
2    68 --              {$IFC fTrace}BP(10);{$ENDC}
2    69 --              s := SELF.boxList.Scanner;
2    70 --              WHILE s.Scan(box) DO
2    71 --                  box.Draw;
2    72 --              {$IFC fTrace}EP;{$ENDC}
2    73 -0 A        END;
2    74 --
2    75 -- A        PROCEDURE TBoxView.InitBoxList (itsHeap: THeap);
2    76 --          VAR  box: TBox;
2    77 --               boxList: TList;
2    78 0- A        BEGIN
2    79 --              {$IFC fTrace}BP(10);{$ENDC}
2    80 --              boxList := TList.CREATE(NIL, itsHeap, 0);
2    81 --              SELF.boxList := boxList;
2    82 --
2    83 --                                      {create and append the first box}
2    84 --              box := TBox.CREATE(NIL, itsHeap);
2    85 --        {$H-} SetLRect(box.shapeLRect, 20,20, 100, 100);  {$H+}
2    86 --              SELF.boxList.InsLast(box);
2    87 --
2    88 --                                      {create and append the second box}
2    89 --              box := TBox.CREATE(NIL, itsHeap);
2    90 --        {$H-} SetLRect(box.shapeLRect, 200, 100, 300, 130);  {$H+}
2    91 --              SELF.boxList.InsLast(box);
2    92 --              {$IFC fTrace}EP;{$ENDC}
2    93 -0 A        END;
2    94 --
2    95 --      END;
2    96 --
2    97 --
2    98 --
2    99 --
2   100 --      METHODS OF TBoxProcess;
2   101 --
2   102 --
2   103 -- A        FUNCTION  TBoxProcess.CREATE: TBoxProcess;
2   104 0- A        BEGIN
2   105 --              {$IFC fTrace}BP(11);{$ENDC}
2   106 --              SELF := TBoxProcess(TProcess.CREATE(NewObject(mainHeap, THISCLASS), mainHeap));
2   107 --              {$IFC fTrace}EP;{$ENDC}
2   108 -0 A        END;
2   109 --
2   110 --
2   111 -- A        FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN): TDocManager;
2   112 0- A        BEGIN
2   113 --              {$IFC fTrace}BP(11);{$ENDC}
2   114 --              NewDocManager := TBoxDocManager.CREATE(NIL, mainHeap, volumePrefix);
2   115 --              {$IFC fTrace}EP;{$ENDC}
2   116 -0 A        END;
2   117 --
2   118 --      END;
2   119 --
2   120 --
2   121 --
2   122 --      METHODS OF TBoxDocManager;
2   123 --
2   124 --
2   125 -- A        FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
2   126 --                                          : TBoxDocManager;
2   127 0- A        BEGIN
2   128 --              {$IFC fTrace}BP(11);{$ENDC}
2   129 --              IF object = NIL THEN
```

```
2  130 --              object := NewObject(itsHeap, THISCLASS);
2  131 --          SELF := TBoxDocManager(TDocManager.CREATE(object, itsHeap, itsPathPrefix));
2  132 --          {$IFC fTrace}EP;{$ENDC}
2  133 -0 A     END;
2  134 --
2  135 --
2  136 -- A     FUNCTION TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID):TWindow;
2  137 0- A     BEGIN
2  138 --          {$IFC fTrace}BP(11);{$ENDC}
2  139 --          NewWindow := TBoxWindow.CREATE(NIL, heap, wmgrID);
2  140 --          {$IFC fTrace}EP;{$ENDC}
2  141 -0 A     END;
2  142 --
2  143 --    END;
2  144 --
2  145 --
2  146 --
2  147 --    METHODS OF TBoxWindow;
2  148 --
2  149 --
2  150 -- A     FUNCTION TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
2  151 --       CONST  isResizable = TRUE;
2  152 0- A     BEGIN
2  153 --          {$IFC fTrace}BP(10);{$ENDC}
2  154 --          IF object = NIL THEN
2  155 --              object := NewObject(itsHeap, THISCLASS);
2  156 --          SELF := TBoxWindow(TWindow.CREATE(object, itsHeap, itsWmgrID, isResizable));
2  157 --          {$IFC fTrace}EP;{$ENDC}
2  158 -0 A     END;
2  159 --
2  160 --
2  161 --
2  162 -- A     PROCEDURE TBoxWindow.BlankStationery;
2  163 --       VAR viewLRect:      LRect;
2  164 --           panel:          TPanel;
2  165 --           boxView:        TBoxView;
2  166 --
2  167 0- A     BEGIN
2  168 --         {$IFC fTrace} BP(10); {$ENDC}
2  169 --                                    {set the view extent LRect}
2  170 --         SetLRect(viewLRect,0,0,5000,3000);
2  171 --         panel := TPanel.CREATE(NIL, SELF.Heap, SELF, 0, 0,
2  172 --                                [aBar,aScroll,aSplit], [aBar,aScroll,aSplit]);
2  173 --
2  174 --                                    {initialize the boxView}
2  175 --         boxView := TBoxView.CREATE(NIL, SELF.Heap, panel, viewLRect);
2  176 --         boxView.InitBoxList(SELF.Heap);
2  177 --         {$IFC fTrace} EP; {$ENDC}
2  178 -0 A     END;
2  179 --
2  180 --    END;
1   92 --
1   93 --    END.
```

```
 1. u2boxer.TEXT
 2. U2Boxer2.text
                     64•( 1)  84•( 1)

-B-
  BlankStationery    84•( 1) 162•( 2)
  boxList            48 ( 1)  69 ( 2)  77•( 2)  80=( 2)  81=( 2)  81 ( 2)  86 ( 2)  91 ( 2)

-C-
  color              35•( 1)  13=( 2)  28 ( 2)
  colorBlack         22•( 1)  27 ( 1)  33 ( 2)
  colorDkGray        21•( 1)  32 ( 2)
  colorGray          20•( 1)  13 ( 2)  31 ( 2)
  colorLtGray        19•( 1)  30 ( 2)
  colorWhite         18•( 1)  27 ( 1)  29 ( 2)
  CREATE             38•( 1)  51•( 1)  63 ( 1)  72 ( 1)  81 ( 1)   6•( 2)  51•( 2)  57 ( 2)  80 ( 2)  84 ( 2)
                     89 ( 2) 103•( 2) 106 ( 2) 114 ( 2) 125•( 2) 131 ( 2) 139 ( 2) 150•( 2) 156 ( 2) 171 ( 2)
                    175 ( 2)

-D-
  Draw               41•( 1)  55•( 1)  19•( 2)  64•( 2)  71 ( 2)

-I-
  InitBoxList        56•( 1)  75•( 2) 176 ( 2)

-L-
  LRect              34 ( 1) 163 ( 2)

-N-
  NewDocManager      64•( 1) 111•( 2) 114=( 2)
  NewWindow          74•( 1) 136•( 2) 139=( 2)

-O-
  openAsTool         64•( 1)

-Q-
  QuickDraw          12•( 1)

-S-
  shapeLRect         34 ( 1)  12=( 2)  25 ( 2)  38 ( 2)  39 ( 2)  85 ( 2)  90 ( 2)

-T-
  TBox               31•( 1)  38 ( 1)   3•( 2)   6 ( 2)  65 ( 2)  76 ( 2)  84 ( 2)  89 ( 2)
  TBoxDocManager     69•( 1)  73 ( 1) 114 ( 2) 122•( 2) 126 ( 2) 131 ( 2)
  TBoxProcess        60•( 1)  63 ( 1) 100•( 2) 103 ( 2) 106 ( 2)
  TBoxView           45•( 1)  52 ( 1)  49•( 2)  52 ( 2)  57 ( 2) 165 ( 2) 175 ( 2)
  TBoxWindow         78•( 1)  81 ( 1) 139 ( 2) 147•( 2) 150 ( 2) 156 ( 2)
  TColor             27•( 1)  35 ( 1)
  TDocManager        65 ( 1)  69 ( 1) 111 ( 2) 131 ( 2)
  TFilePath          64 ( 1)
  TList              48 ( 1)  77 ( 2)  80 ( 2)
  TObject            31 ( 1)
  TProcess           60 ( 1) 106 ( 2)
  TView              45 ( 1)
  TWindow            74 ( 1)  78 ( 1) 136 ( 2) 156 ( 2)

-U-
  U2Boxer             1•( 1)
  UABC               14•( 1)
  UDraw              13•( 1)
  UFont               9•( 1)
  UObject             6•( 1)

-V-
  volumePrefix       64•( 1) 114 ( 2)

*** End Xref: 37 id's   131 references      [417000 bytes/4962 id's/42201 refs]
```

[Segment 6]

Selections & Highlighting in Boxer

Purpose of this segment:

1) To introduce the concept of selections.
2) To describe how to highlight a box.

How to use this segment:

This segment deals with selections. It includes a tutorial and a lab. Please complete the tutorial before proceeding to the lab.

You should start this segment after the "Intro to Boxer" segment. You should complete this segment before starting the "Box Moves" segment.

In the previous segment you implemented drawing boxes in the view. The next stage is to select a box, then perform actions (changing its color, moving it, etc.) upon it. This segment covers how to select a box, and indicate that it is selected.

INTRODUCTION TO SELECTIONS

Making a change to a Lisa document, you typically operate upon a particular object (or objects) in the view. Since you select the object to be acted upon (by using the mouse or other means), it is known as the *selected object*.

A *selection* is a ToolKit object that keeps track of the selected object. All changes to a document are routed through selections. A selection verifies that a desired change is appropriate for its selected object, before performing any operation.

Selections in the ToolKit have the following structure:

```
TSelection = SUBCLASS OF TObject

  {fields}
  window:      TWindow;   {the window in which it was made}
  panel:       TPanel;    {the panel in which it was made}
  view:        TView;     {the view (of the panel) in which it was made}
  kind:        INTEGER;   {kind codes are defined by the application's view}
  anchorLPt:   LPoint;    {the place where the mouse went down (view-relative)}
  currLPt:     LPoint;    {the place the mouse was last tracked (view-relative)}
  boundLRect:  LRect;     {the LRect bounding the selection (default is hugeLRect -
```

```
                        an LRect with bounds:  0, 0, $3FFFFFFF, $3FFFFFFF ))

        {other internal fields}
```

A selection exists even if no object is selected. In that case, the selection is null. The selection's kind field, indicates whether the selection is null or not. A special ToolKit constant, nothingKind *(equal to 0)* indicates a null selection.

The selection is either an instance or a descendant of TSelection. A selection exists in each panel of the document.

The main role of the selection is to apply user actions such as mouse moves or menu commands to the selected object *(or objects)*.

SELECTION CONCEPTS

There are four concepts essential to successful implementation of selections in your applications. These are: highlighting, deselection, anchor LPoint, and free and replace.

highlighting

A selected object is distinguished by *highlighting* it. A *highlight* is a visible mark on the object, indicating that it is selected.

The selection's Highlight method is used to both mark an object when it is selected, and to remove the mark when the object is no longer selected. Each subclass of TSelection should override the Highlight method.

deselection

An object that is no longer selected is said to be *deselected*. A selected box, for example, is deselected when one of the following occurs:

1) a different box is selected.

    (for example, the user clicks on another box)

2) the user clicks where there is no box.

anchor LPoint

Users of Lisa applications typically select objects by clicking on them. For this reason, every selection has an *anchor LPoint*. *[An LPoint is analogous to a QuickDraw Point, but with 32-bit coordinates]*. The selection's anchor LPoint is used to save the view-relative location where the mouse went down.

free and replace

Standard ToolKit behavior is to replace the selection after each time a mouse press is detected in the view. This is tricky, since so many ToolKit objects reference the selection. For this reason, selections are replaced using a special method, FreedAndReplacedBy.

FreedAndReplacedBy replaces the selection while retaining its original Clascal handle. Thus, all references to the old selection object now point to the new one.

UPDATE REGION

Another concept important in our implementation is the *update region*.

The update region is an area within the document's window that needs to be redrawn. Any time we change the view, we may need to define an update region to modify the display. The update region is set by the window manager to be the visRgn when the window updates. When the window update completes, the update region is emptied, and the visRgn reverts to its previous state.

An update region is more efficient to refresh than the whole window. Fewer bits need to be changed in the screen bit map. *Modifying a few thousand extra bits after every change in the document can turn a responsive application into a real dog.*

You have two options for updating the display. If the change is trivial, you can update the display in each pane directly. But, if the change affects parts of the document besides the selected object, you need to build an update region. The mechanics of building an update region are covered in the next Boxer segment — "Box Moves".

PARTIAL INTERFACE OF TSELECTION

The partial interface of TSelection is listed below:

```
TSelection = SUBCLASS OF TObject

  {methods}
        {creation}
    FUNCTION  {TSelection.} CREATE(object: TObject; heap: THeap; itsView: TView;
                                    itsKind: INTEGER; itsAnchorLPt: LPoint): TSelection;

                                            {replaces one selection with another}
    FUNCTION  {TSelection.} FreedAndReplacedBy(selection: TSelection): TSelection;

        {selecting}
                                            {highlights the current selection}
    PROCEDURE {TSelection.} Highlight(highTransit: THighTransit); DEFAULT;

                                            {deselects the selected object, then
                                             replaces it with TView.NoSelection}
    PROCEDURE {TSelection.} DeSelect; DEFAULT;

                                            {called when the mouse is pressed}
    PROCEDURE {TSelection.} MousePress(mouseLPt: LPoint); DEFAULT;
                                            {called when the mouse moves}
    PROCEDURE {TSelection.} MouseMove(mouseLPt: LPoint); DEFAULT;
                                            {called when the mouse is released}
    PROCEDURE {TSelection.} MouseRelease; DEFAULT;
```

(Note: DEFAULT indicates that the method is intended to be overridden in subclasses, but some default behavior is implemented)

HIGHLIGHTING

In Boxer, the selected objects are boxes. Your application needs to supply a Highlight method that marks a selected box.

Although you need to tell the Generic Application how to highlight, it determines when to highlight.

The Generic Application calls the current selection's Highlight method to highlight the selected object. This is done automatically during window updates when the update region is not empty.

Highlight methods typically mark the selected object in a reversible way. The same method is used to both paint handles on the selected box, and remove them when the box is deselected.

To deselect your application must call Highlight itself.

The ToolKit defines several special pen states for painting highlights reversibly. For boxes, the two pen states we shall be concerned with are:

hOffToOn {makes highlight marks appear}

hOnToOff {makes highlight marks disappear}

A simple way to highlight a box is to paint tiny rectangles at its corners. Those tiny rectangles are sometimes called *handles*.

IMPLEMENTATION STRATEGY

In this stage of Boxer we implement a particular kind of selection, a boxSelection. We define the class, TBoxSelection, as a subclass of TSelection.

The user is allowed to select one of two boxes in the view. As the partial interface below indicates, a special field, box, in the selection is used to keep track of the selected box.

```
TBoxSelection = SUBCLASS OF TSelection

  {fields}

  box: TBox;   {the selected box, or NIL (if none is selected)}
```

user interface

Two boxes are drawn on the screen. Either box may be selected by the user.

When the user clicks on a box, it is highlighted. Tiny handles appear at the corners of the box. Any previously selected box is deselected. The handles of that box disappear.

When the user clicks where there is no box, none is highlighted, but any previously selected box is still deselected.

theory of operation

The Generic Application initializes the document, calling BlankStationery in the process. When initialization is complete, the Generic Application waits for the first user event. In the case of Boxer, that first event should be a mouse press. Once a mouse press is detected, the following method calls are made:

```
pane.MouseTrack             {TPane.}
  view.MouseTrack           {TImage.}
    view.MousePress         {TImage.} default method
      selection.MousePress  {TBoxSelection.} called by TImage.MousePress
```

Your code resumes from here.

1. Extract the mouse point where the button went down. This is done in {TBoxSelection.}MousePress.
2. Deselect the previously selected box.
3. Determine if the mouse point is inside of one your boxes.

   If yes then make that box the one currently selected, and proceed to step 4.

   If no then do nothing.

After the mouse is released the Generic Application tells the window to update. The update is performed only if the update region is not empty. During the update the currently selected object is highlighted. When an update is performed this is the flow of control:

```
window.Update               {TWindow.}
  ...
  For each pane, do
  pane.Refresh              {TPane.}
    view.Draw               {TBoxView.}
    selection.Highlight     {TBoxSelection.}
```

In this stage of Boxer though, the change is trivial. The application highlights the selected box directly.

4. Highlight the currently selected box.

In your selection's Highlight code do the following:

a. Set the QuickDraw pen to the appropriate pen state.

b. If a box is selected, tell that box to paint its handles.

At this point the Generic Application resumes. The next mouse press event repeats the cycle from step 1.

<u>actual implementation</u>

The actual implementation for 3Boxer is summarized below. The code for 2Boxer is used as a base for the changes and additions described.

New Constants

```
boxSelectionKind = 1;
```

If a box is selected, we set the kind field to boxSelectionKind. If no box is selected, we set the field to nothingKind.

New Classes

```
[TBoxSelection]
TBoxSelection = SUBCLASS OF TSelection

   {fields}
  box:  TBox;

  {Creation}
FUNCTION  {TBoxSelection.} CREATE(object: TObject; itsHeap: THeap; itsView: TView;
                                  itsKind: INTEGER; itsAnchorLPt: LPoint)
                                 :TBoxSelection;
```

{TBoxSelection.}CREATE creates a new boxSelection with the given kind and anchor LPoint.

```
PROCEDURE {TBoxSelection.} HighLight(highTransit: THighTransit); OVERRIDE;
```

{TBoxSelection.}Highlight sets the highlight pen state. It then calls SELF.box.PaintHandles to highlight the selected box.

```
PROCEDURE {TBoxSelection.} MousePress(mouseLPt: LPoint); OVERRIDE;
```

{TBoxSelection.}MousePress deselects any previously selected box; then calls SELF.view.BoxWith to return the current selected box. The returned value is assigned to SELF.box. The kind field is set to boxSelectionKind if a box was selected.

New Methods (for existing classes)

```
[TBoxView]
FUNCTION {TBoxView.} BoxWith(LPt: LPoint): TBox;
```

{TBoxView.}BoxWith returns the box containing the LPoint where the mouse went down. If no box contains the mouse LPoint, then NIL is returned.

```
FUNCTION {TBoxView.} NoSelection: TSelection; OVERRIDE;
```

{TBoxView.}**NoSelection** returns a null boxSelection (with kind set to **nothingKind**).

```
[TBox]
PROCEDURE {TBox.} PaintHandles;
```

{TBox.}**PaintHandles** paints a 6x4 (pixel) handle rectangle at each corner of the box's shape LRect. The highlight pen state is assumed to be properly set.

Changed Methods

```
[TBoxWindow]
PROCEDURE {TBoxWindow.} BlankStationery;
```

{TBoxWindow.}**BlankStationery** replaces the default selection with a null boxSelection.

(Note: The implementation in 3Boxer, the Boxer stage for this segment, differs from the ToolKit guidelines in one respect — the selection is never freed and replaced. In the special case exhibited by this program, only one kind of selection, a boxSelection, ever exists.)

HIGHLIGHTING IN MULTIPLE PANES

Managing displays in multiple panes is a simple matter using the ToolKit. The code segment below uses the method {TPanel.}**HighLight** to take care of highlighting a box that may be displayed in several panes.

```
PROCEDURE {TBoxSelection.} MousePress(( … ));
VAR panel: TPanel;
BEGIN
            {extract the panel containing the selection}
   panel := SELF.panel;
            {highlight the selection in every pane in the panel}
   panel.Highlight(SELF, hOffToOn);
END;
```

When your document's window has multiple panels call {TWindow.}Highlight instead. This calls {TPanel.}**Highlight** for each of its panels.

The Generic Application calls {TWindow.}Highlight to turn down highlighting when the window is deactivated, and to turn up highlighting when the window is reactivated.

Questions:

1) What is the relationship between the *selection* and the *selected object*?
2) What is a null selection?

   How does an application indicate a null selection?
3) What kind of selection does the Generic Application first install in your panel when initializing a document?

   What method should your application use to replace that selection with an instance of TBoxSelection?
4) When a document is initialized the update region is set to inner Rect of the window to draw the initial view. When initialization is complete what is the update region set to?
5) Diagram how a mouse press reaches the current selection. Start with the window.
6) Since this stage of Boxer never alters the update region, is the currently selected object ever highlighted during a window update?
7) When does your application need to deselect?

   What method is called to deselect a box?

Selections Lab

Purpose:

To implement box selections.

What you will do in the lab:

You will compile and run 3Boxer, then optionally modify the source. The steps below describe what you should do:

1) Copy the following files onto your prefix volume:

   U3Boxer.TEXT

   U3Boxer2.TEXT

   M3Boxer.TEXT

   P3Boxer.TEXT

2) Compile, install, and run the sample application, 3Boxer. Use 43 as the tool number.

3) Scan the listings of the four files in the sample application. These are included in the appendix, "Code Smaples for this Segment".

4) *[Optional]* The Lisa user interface recommends that applications toggle the highlighting of the selected object when the mouse button is pressed with the SHIFT key down. If the selected object is already highlighted, it is deselected. Otherwise, the object is selected and highlighted.

   The SHIFT/mouse press combination should not disturb the highlighting of previously selected items. The result is that several objects may comprise the selection.

   In contrast, a normal mouse press (with the SHIFT key up), must deselect all previously selected objects. This results in either a newly selected object or a null selection.

   The state of the shift key is supplied by the global ToolKit Boolean variable, clickState.fShift. The variable is true only if the shift key was down when the mouse press was detected.

   Given the preceding information, implement the selection of multiple boxes.

Things to look out for:

- *Highlighting never gets turned on.*

  Be sure you call the HighLight method of TBoxSelection to highlight the newly selected object when a mouse press occurs.

- *Highlighting never gets turned off.*

  Check to see if you deselect the old selection.

- *When deactivating the window, highlighting goes away.*

  Make sure that the current selection, window.selectPanel.selection, references the selected object.

- *When activating the window, highlighting mysteriously appears.*

  Check to see if you deselect the previously selected box.

- *Highlighting gets turned off in some panes, but not others.*

  Make sure you deselect and highlight in all panes.

- *Process aborts when you press the mouse.*

  Verify both that your kind field is set correctly, and that you are not trying to highlight a nil or otherwise invalid box.

Code Sample
for this
Segment

```
43
SLOT2CHAN1
;no assembler files
$
;no building blocks
$
;no links
$
y
y
n
BoxNum3
```

```
;     PBOXER.TEXT    for  Boxer
;Phrase file for Boxer class example
1
3
2500
$-#BOOT-TK/PABC

; Apple building block phrase files can be included here

1000
Boxer

; Other application alerts can be included here, numbered between 1001 and 32000

0

1
$-#BOOT-TK/PABC~File/Print

2
Page Layout
Preview Actual Pages#401
Preview Page Breaks#402
Don't Preview Pages#403

100
Buzzwords
Set Aside ↑Document↑#109

0
```

```
PROGRAM M3Boxer;

USES
    {$U UObject      } UObject,

  {$IFC libraryVersion <= 20}
    {$U UFont}              UFont,
  {$ENDC}

    {$U QuickDraw    } QuickDraw,
    {$U UDraw        } UDraw,
    {$U UABC         } UABC,

    {$U U3Boxer      } U3Boxer;

CONST

    phraseVersion = 1;

BEGIN

    process := TBoxProcess.CREATE;
    process.Commence(phraseVersion);
    process.Run;
    process.Complete(TRUE);

END.
```

```
1     1 --   UNIT U3Boxer;
1     2 --
1     3 --   INTERFACE
1     4 --
1     5 --   USES
1     6 --       {$U UObject}              UObject,
1     7 --
1     8 --     {$IFC libraryVersion <= 20}
1     9 --       {$U UFont}                UFont,
1    10 --     {$ENDC}
1    11 --
1    12 --       {$U QuickDraw}            QuickDraw,
1    13 --       {$U UDraw}                UDraw,
1    14 --       {$U UABC}                 UABC;
1    15 --
1    16 --   CONST
1    17 --       colorWhite = 1;
1    18 --       colorLtGray = 2;
1    19 --       colorGray = 3;
1    20 --       colorDkGray = 4;
1    21 --       colorBlack = 5;
1    22 --
1    23 --       boxSelectionKind = 1;
1    24 --
1    25 --   TYPE
1    26 --
1    27 --       TColor = colorWhite..colorBlack; {color of a box}
1    28 --
1    29 --
1    30 --
1    31 --     {New Classes for this Application}
1    32 --
1    33 --       TBox = SUBCLASS OF TObject
1    34 --
1    35 --           {Variables}
1    36 --           shapeLRect:     LRect;
1    37 --           color:          TColor;
1    38 --
1    39 --         {Creation/Destruction}
1    40 --           FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
1    41 --
1    42 --           PROCEDURE TBox.PaintHandles;
1    43 --
1    44 --         {Display}
1    45 --           PROCEDURE TBox.Draw;
1    46 --           END;
1    47 --
1    48 --
1    49 --
1    50 --       TBoxView = SUBCLASS OF TView
1    51 --
1    52 --         {Variables}
1    53 --           boxList:        TList;
1    54 --
1    55 --         {Creation/Destruction}
1    56 --           FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
1    57 --                                        : TBoxView;
1    58 --
1    59 --           FUNCTION  {TBoxView.}BoxWith(LPt: LPoint): TBox;
1    60 --
1    61 --           PROCEDURE  TBoxView. Draw; OVERRIDE;
1    62 --           PROCEDURE TBoxView. InitBoxList(itsHeap: THeap);
1    63 --           FUNCTION   TBoxView. NoSelection: TSelection; OVERRIDE;
1    64 --           PROCEDURE TBoxView.MousePress(mouseLPt: LPoint); OVERRIDE;
1    65 --           END;
1    66 --
1    67 --
1    68 --
1    69 --       TBoxSelection = SUBCLASS OF TSelection
1    70 --
1    71 --         {Variables}
1    72 --           box:   TBox;
1    73 --
1    74 --         {Creation/Destruction}
1    75 --           FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsKind: INTEGER;
1    76 --                                          itsAnchorLPt: LPoint): TBoxSelection;
1    77 --
1    78 --         {Drawing - per pad}
1    79 --           PROCEDURE  TBoxSelection. Highlight(highTransit: THighTransit); OVERRIDE;
1    80 --
1    81 --        END;
1    82 --
1    83 --
1    84 --       TBoxProcess = SUBCLASS OF TProcess
1    85 --
1    86 --         {Creation/Destruction}
1    87 --           FUNCTION  {TBoxProcess.}CREATE: TBoxProcess;
1    88 --           FUNCTION   TBoxProcess. NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
1    89 --                                             : TDocManager; OVERRIDE;
1    90 --           END;
1    91 --
1    92 --
1    93 --       TBoxDocManager = SUBCLASS OF TDocManager
1    94 --
1    95 --         {Creation/Destruction}
1    96 --           FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
1    97 --                                        : TBoxDocManager;
1    98 --           FUNCTION   TBoxDocManager. NewWindow(heap: THeap; wmgrID: TWindowID): TWindow; OVERRIDE;
1    99 --           END;
1   100 --
1   101 --
1   102 --
1   103 --       TBoxWindow = SUBCLASS OF TWindow
1   104 --
1   105 --         {Creation/Destruction}
1   106 --           FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
1   107 --
1   108 --         {Document Creation}
1   109 --           PROCEDURE  TBoxWindow. BlankStationery; OVERRIDE;
1   110 --
```

```
1  111 --          END;
1  112 --
1  113 --
1  114 --   IMPLEMENTATION
1  115 --
1  116 --     {$I U3Boxer2.text}
2    1 --   {U3BOXER2}
2    2 --
2    3 --   METHODS OF TBox;
2    4 --
2    5 -- A     FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
2    6 0- A     BEGIN
2    7 --          {$IFC fTrace}BP(11);{$ENDC}
2    8 --          SELF := NewObject(itsHeap, THISCLASS);
2    9 --          WITH SELF DO
2   10 1-              BEGIN
2   11 --              shapeLRect := zeroLRect;
2   12 --              color := colorGray;
2   13 -1              END;
2   14 --          {$IFC fTrace}EP;{$ENDC}
2   15 -0 A     END;
2   16 --
2   17 --   {This draws a particular box}
2   18 -- A     PROCEDURE TBox.Draw;
2   19 --       VAR lPat:    LPattern;
2   20 0- A     BEGIN
2   21 --          {$IFC fTrace}BP(10);{$ENDC}
2   22 --          PenNormal;
2   23 --
2   24 --          IF LRectIsVisible(SELF.shapeLRect) THEN {this box needs to be drawn}
2   25 1-              BEGIN
2   26 --            {Get a Quickdraw pattern to represent the box's color}
2   27 2-              CASE SELF.color OF
2   28 --                  colorWhite:      lPat := lPatWhite;
2   29 --                  colorLtGray:     lPat := lPatLtGray;
2   30 --                  colorGray:       lPat := lPatGray;
2   31 --                  colorDkGray:     lPat := lPatDkGray;
2   32 --                  colorBlack:      lPat := lPatBlack;
2   33 --                  OTHERWISE        lPat := lPatWhite; {this case should not happen}
2   34 -2                  END;
2   35 --
2   36 --             {Fill the box with the pattern, and draw a frame around it}
2   37 --              FillLRect(SELF.shapeLRect, lPat);
2   38 --              FrameLRect(SELF.shapeLRect);
2   39 -1              END;
2   40 --          {$IFC fTrace}EP;{$ENDC}
2   41 -0 A     END;
2   42 --
2   43 --       {This calls the DoToHandle Procedure once for each handle LRect; user of this method must
2   44 --         set up the pen pattern and mode before calling}
2   45 -- A     PROCEDURE TBox.PaintHandles;
2   46 --       VAR hLRect,
2   47 --           shapeLRect: LRect;
2   48 --           dh, dv:     LONGINT;
2   49 --
2   50 -- B        PROCEDURE MoveHandleAndPaint(hOffset, vOffset: LONGINT);
2   51 0- B        BEGIN
2   52 --              OffsetLRect(hLRect, hOffset, vOffset);
2   53 --              PaintLRect(hLRect);
2   54 -0 B        END;
2   55 --
2   56 0- A     BEGIN
2   57 --          {$IFC fTrace}BP(10);{$ENDC}
2   58 --          SetLRect(hLRect, -3, -2, 3, 2);
2   59 --
2   60 --          shapeLRect := SELF.shapeLRect;
2   61 --          WITH shapeLRect DO
2   62 1-              BEGIN
2   63 --              dh := right - left;
2   64 --              dv := bottom - top;
2   65 --              MoveHandleAndPaint(left, top);     {draw top left handle}
2   66 -1              END;
2   67 --          MoveHandleAndPaint(dh, 0);            {then top right}
2   68 --          MoveHandleAndPaint(0, dv);            {then bottom right}
2   69 --          MoveHandleAndPaint(-dh, 0);           {finally bottom left}
2   70 --          {$IFC fTrace}EP;{$ENDC}
2   71 -0 A     END;
2   72 --
2   73 --   END;
2   74 --
2   75 --
2   76 --   METHODS OF TBoxView;
2   77 --
2   78 -- A     FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
2   79 --                                    : TBoxView;
2   80 0- A     BEGIN
2   81 --          {$IFC fTrace}BP(11);{$ENDC}
2   82 --          IF object = NIL THEN
2   83 --              object := NewObject(itsHeap, THISCLASS);
2   84 --          SELF := TBoxView(itsPanel.NewView(object, itsExtent, TPrintManager.CREATE(NIL, itsHeap),
2   85 --                                            stdMargins, TRUE));
2   86 --          {$IFC fTrace}EP;{$ENDC}
2   87 -0 A     END;
2   88 --
2   89 --
2   90 --   {This returns the box containing a certain point}
2   91 -- A     FUNCTION  TBoxView.BoxWith(LPt: LPoint): TBox;
2   92 --       VAR box:    TBox;
2   93 --           s:      TListScanner;
2   94 0- A     BEGIN
2   95 --          {$IFC fTrace}BP(11);{$ENDC}
2   96 --          boxWith := NIL;
2   97 --          s := SELF.boxList.Scanner;
2   98 --          WHILE s.Scan(box) DO
2   99 --              IF LPtInLRect(LPt, box.shapeLRect) THEN
2  100 --                  boxWith := box;
2  101 --          {$IFC fTrace}EP;{$ENDC}
2  102 -0 A     END;
2  103 --
2  104 --
```

```
2  105 --   {This draws the list of boxes}
2  106 -- A     PROCEDURE TBoxView.Draw;
2  107 --      VAR box:    TBox;
2  108 --          s:      TListScanner;
2  109 0- A     BEGIN
2  110 --          {$IFC fTrace}BP(10);{$ENDC}
2  111 --          s := SELF.boxList.Scanner;
2  112 --          WHILE s.Scan(box) DO
2  113 --              box.Draw;
2  114 --          {$IFC fTrace}EP;{$ENDC}
2  115 -0 A     END;
2  116 --
2  117 --
2  118 -- A     PROCEDURE TBoxView.InitBoxList (itsHeap: THeap);
2  119 --      VAR  box: TBox;
2  120 --           boxList: TList;
2  121 0- A     BEGIN
2  122 --          {$IFC fTrace}BP(10);{$ENDC}
2  123 --          boxList := TList.CREATE(NIL, itsHeap, 0);
2  124 --          SELF.boxList := boxList;
2  125 --
2  126 --                                      {create and append the first box}
2  127 --          box := TBox.CREATE(NIL, itsHeap);
2  128 --    {$H-} SetLRect(box.shapeLRect, 20,20, 100, 100);  {$H+}
2  129 --          SELF.boxList.InsLast(box);
2  130 --
2  131 --                                      {create and append the second box}
2  132 --          box := TBox.CREATE(NIL, itsHeap);
2  133 --    {$H-} SetLRect(box.shapeLRect, 200, 100, 300, 130);  {$H+}
2  134 --          SELF.boxList.InsLast(box);
2  135 --          {$IFC fTrace}EP;{$ENDC}
2  136 -0 A     END;
2  137 --
2  138 -- A     FUNCTION TBoxView.NoSelection: TSelection;
2  139 0- A     BEGIN
2  140 --         {$IFC fTrace}BP(11);{$ENDC}
2  141 --          NoSelection := TBoxSelection.CREATE(NIL, SELF.Heap, SELF, nothingKind, zeroLPt);
2  142 --          {$IFC fTrace}EP;{$ENDC}
2  143 -0 A     END;
2  144 --
2  145 --     {This PROCEDURE makes a new selection, when the user presses the mouse button}
2  146 --     {This procedure illustrates the "standard" way of creating a new selection}
2  147 -- A     PROCEDURE TBoxView.MousePress(mouseLPt: LPoint);
2  148 --      VAR newSelection: TSelection;
2  149 --          panel:      TPanel;
2  150 --          box: TBox;
2  151 0- A     BEGIN
2  152 --         {$IFC fTrace}BP(11);{$ENDC}
2  153 --          panel := SELF.panel;
2  154 --          panel.Highlight(self.panel.selection, hOntoOff);        {Turn off the old highlighting}
2  155 --
2  156 --          newSelection := self.panel.selection.FreedAndReplacedBy(TBoxSelection.CREATE(NIL, SELF.heap, SELF,
2  157 --                                                                                     boxSelectionKind,
2  158 --                                                                                     mouseLPt));
2  159 --
2  160 --          self.panel.selection := newSelection;
2  161 --
2  162 --          newSelection.currLPt := mouseLPt;
2  163 --
2  164 --          box := self.BoxWith(mouseLPt);        {Find the box the user clicked on}
2  165 --          IF box = NIL THEN
2  166 --              newSelection.kind := nothingKind
2  167 --          ELSE
2  168 --              newSelection.kind := boxSelectionKind;
2  169 --          TBoxSelection(newSelection).box := box;
2  170 --
2  171 --          panel.Highlight(newSelection, hOffToOn);        {Turn on the highlighting for the newly selected box}
2  172 --
2  173 --          self.panel.selection.MarkChanged;     {Allow the document to be saved so that any changes made}
2  174 --                                                {can become permanent}
2  175 --
2  176 --         {$IFC fTrace}EP;{$ENDC}
2  177 -0 A     END;
2  178 --
2  179 --   END;
2  180 --
2  181 --
2  182 --   METHODS OF TBoxSelection;
2  183 --
2  184 --
2  185 -- A     FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsKind: INTEGER;
2  186 --                                     itsAnchorLPt: LPoint): TBoxSelection;
2  187 0- A     BEGIN
2  188 --          {$IFC fTrace}BP(11);{$ENDC}
2  189 --          IF object = NIL THEN
2  190 --              object := NewObject(itsHeap, THISCLASS);
2  191 --          SELF := TBoxSelection(TSelection.CREATE(object, itsHeap, itsView, itsKind, itsAnchorLPt));
2  192 --
2  193 --          SELF.box := NIL;
2  194 --          {$IFC fTrace}EP;{$ENDC}
2  195 -0 A     END;
2  196 --
2  197 --     {This draws the handles on the selected box}
2  198 -- A     PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit);
2  199 --      VAR box:    TBox;
2  200 0- A     BEGIN
2  201 --          {$IFC fTrace}BP(11);{$ENDC}
2  202 --          IF SELF.kind <> nothingKind THEN
2  203 1-              BEGIN
2  204 --              box := SELF.box;
2  205 --              thePad.SetPenToHighlight(highTransit);  {set the drawing mode according to desired highlighting}
2  206 --              box.PaintHandles;                       {draw the handles on the box}
2  207 -1              END;
2  208 --          {$IFC fTrace}EP;{$ENDC}
2  209 -0 A     END;
2  210 --
2  211 --   END;
2  212 --
2  213 --
2  214 --   METHODS OF TBoxProcess;
```

```
2  215 --
2  216 -- A      FUNCTION  TBoxProcess.CREATE: TBoxProcess;
2  217 0- A      BEGIN
2  218 --           {$IFC fTrace}BP(11);{$ENDC}
2  219 --           SELF := TBoxProcess(TProcess.CREATE(NewObject(mainHeap, THISCLASS), mainHeap));
2  220 --           {$IFC fTrace}EP;{$ENDC}
2  221 -0 A      END;
2  222 --
2  223 --
2  224 -- A      FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN): TDocManager;
2  225 0- A      BEGIN
2  226 --           {$IFC fTrace}BP(11);{$ENDC}
2  227 --           NewDocManager := TBoxDocManager.CREATE(NIL, mainHeap, volumePrefix);
2  228 --           {$IFC fTrace}EP;{$ENDC}
2  229 -0 A      END;
2  230 --
2  231 --    END;
2  232 --
2  233 --
2  234 --
2  235 --    METHODS OF TBoxDocManager;
2  236 --
2  237 -- A      FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
2  238 --                                            : TBoxDocManager;
2  239 0- A      BEGIN
2  240 --           {$IFC fTrace}BP(11);{$ENDC}
2  241 --           IF object = NIL THEN
2  242 --               object := NewObject(itsHeap, THISCLASS);
2  243 --           SELF := TBoxDocManager(TDocManager.CREATE(object, itsHeap, itsPathPrefix));
2  244 --           {$IFC fTrace}EP;{$ENDC}
2  245 -0 A      END;
2  246 --
2  247 --
2  248 -- A      FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID): TWindow;
2  249 0- A      BEGIN
2  250 --           {$IFC fTrace}BP(11);{$ENDC}
2  251 --           NewWindow := TBoxWindow.CREATE(NIL, heap, wmgrID);
2  252 --           {$IFC fTrace}EP;{$ENDC}
2  253 -0 A      END;
2  254 --
2  255 --    END;
2  256 --
2  257 --
2  258 --
2  259 --    METHODS OF TBoxWindow;
2  260 --
2  261 -- A      FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
2  262 0- A      BEGIN
2  263 --           {$IFC fTrace}BP(10);{$ENDC}
2  264 --           IF object = NIL THEN
2  265 --               object := NewObject(itsHeap, THISCLASS);
2  266 --           SELF := TBoxWindow(TWindow.CREATE(object, itsHeap, itsWmgrID, TRUE));
2  267 --           {$IFC fTrace}EP;{$ENDC}
2  268 -0 A      END;
2  269 --
2  270 -- A      PROCEDURE TBoxWindow.BlankStationery;
2  271 --        VAR viewLRect:         LRect;
2  272 --            panel:             TPanel;
2  273 --            boxView:           TBoxView;
2  274 --            aSelection:        TBoxSelection;
2  275 --
2  276 0- A      BEGIN
2  277 --           {$IFC fTrace}BP(10);{$ENDC}
2  278 --           panel := TPanel.CREATE(NIL, SELF.Heap, SELF, 0, 0, [aScroll, aSplit], [aScroll, aSplit]);
2  279 --
2  280 --           SetLRect(viewLRect, 0, 0, 5000, 3000);
2  281 --           boxView := TBoxView.CREATE(NIL, SELF.Heap, panel, viewLRect);
2  282 --           boxView.InitBoxList(SELF.Heap);
2  283 --
2  284 --           {$IFC fTrace}EP;{$ENDC}
2  285 -0 A      END;
2  286 --
2  287 --    END;
2  288 --
1  117 --
1  118 --    END.
```

```
 1. u3boxer.TEXT
 2. U3Boxer2.text
                     61•( 1)   63•( 1)   79•( 1)   88•( 1)   98•( 1)  109•( 1)

-B-
 BlankStationery    109•( 1)  270•( 2)
 box                 72 ( 1)   92•( 2)   98 ( 2)   99 ( 2)  100 ( 2)  107•( 2)  112 ( 2)  113 ( 2)  119•( 2)  127=( 2)
                    128 ( 2)  129 ( 2)  132=( 2)  133 ( 2)  134 ( 2)  150•( 2)  164=( 2)  165 ( 2)  169=( 2)  169 ( 2)
                    193=( 2)  199•( 2)  204=( 2)  204 ( 2)  206 ( 2)
 boxList             53 ( 1)   97 ( 2)  111 ( 2)  120•( 2)  123=( 2)  124=( 2)  124 ( 2)  129 ( 2)  134 ( 2)
 boxSelectionKind    23•( 1)  157 ( 2)  168 ( 2)
 BoxWith             59•( 1)   91•( 2)  164 ( 2)

-C-
 color               37•( 1)   12=( 2)   27 ( 2)
 colorBlack          21•( 1)   27 ( 1)   32 ( 2)
 colorDkGray         20•( 1)   31 ( 2)
 colorGray           19•( 1)   12 ( 2)   30 ( 2)
 colorLtGray         18•( 1)   29 ( 2)
 colorWhite          17•( 1)   27 ( 1)   28 ( 2)
 CREATE              40•( 1)   56•( 1)   75•( 1)   87 ( 1)   96 ( 1)  106 ( 1)    5•( 2)   78•( 2)   84 ( 2)  123 ( 2)
                    127 ( 2)  132 ( 2)  141 ( 2)  156 ( 2)  185•( 2)  191 ( 2)  216•( 2)  219 ( 2)  227 ( 2)  237•( 2)
                    243 ( 2)  251 ( 2)  261•( 2)  266 ( 2)  278 ( 2)  281 ( 2)

-D-
 Draw                45•( 1)   61•( 1)   18•( 2)  106•( 2)  113 ( 2)

-H-
 heap                98•( 1)  156 ( 2)  251 ( 2)
 Highlight           79•( 1)  154 ( 2)  171 ( 2)  198•( 2)
 highTransit         79•( 1)  205 ( 2)

-I-
 InitBoxList         62•( 1)  118•( 2)  282 ( 2)

-L-
 LRect               36 ( 1)   47 ( 2)  271 ( 2)

-M-
 MousePress          64•( 1)  147•( 2)

-N-
 NewDocManager       88•( 1)  224•( 2)  227=( 2)
 NewWindow           98•( 1)  248•( 2)  251=( 2)
 NoSelection         63•( 1)  138•( 2)  141=( 2)

-O-
 openAsTool          88•( 1)

-P-
 PaintHandles        42•( 1)   45•( 2)  206 ( 2)

-Q-
 QuickDraw           12•( 1)

-S-
 shapeLRect          36 ( 1)   11=( 2)   24 ( 2)   37 ( 2)   38 ( 2)   47•( 2)   60=( 2)   60 ( 2)   61 ( 2)   99 ( 2)
                    128 ( 2)  133 ( 2)

-T-
 TBox                33•( 1)   40 ( 1)   59 ( 1)   72 ( 1)    3•( 2)    5 ( 2)   91 ( 2)   92 ( 2)  107 ( 2)  119 ( 2)
                    127 ( 2)  132 ( 2)  150 ( 2)  199 ( 2)
 TBoxDocManager      93•( 1)   97 ( 1)  227 ( 2)  235•( 2)  238 ( 2)  243 ( 2)
 TBoxProcess         84•( 1)   87 ( 1)  214•( 2)  216 ( 2)  219 ( 2)
 TBoxSelection       69•( 1)   76 ( 1)  141 ( 2)  156 ( 2)  169 ( 2)  182•( 2)  186 ( 2)  191 ( 2)  274 ( 2)
 TBoxView            50•( 1)   57 ( 1)   76•( 2)   79 ( 2)   84 ( 2)  273 ( 2)  281 ( 2)
 TBoxWindow         103•( 1)  106 ( 1)  251 ( 2)  259•( 2)  261 ( 2)  266 ( 2)
 TColor              27•( 1)   37 ( 1)
 TDocManager         89 ( 1)   93 ( 1)  224 ( 2)  243 ( 2)
 TFilePath           88 ( 1)
 THeap               98 ( 1)
 THighTransit        79 ( 1)
 TList               53 ( 1)  120 ( 2)  123 ( 2)
 TObject             33 ( 1)
 TProcess            84 ( 1)  219 ( 2)
 TSelection          63 ( 1)   69 ( 1)  138 ( 2)  148 ( 2)  191 ( 2)
 TView               50 ( 1)
 TWindow             98 ( 1)  103 ( 1)  248 ( 2)  266 ( 2)
 TWindowID           98 ( 1)

-U-
 U3Boxer              1•( 1)
 UABC                14•( 1)
 UDraw               13•( 1)
 UFont                9•( 1)
 UObject              6•( 1)

-V-
 volumePrefix        88•( 1)  227 ( 2)

-W-
 wmgrID              98•( 1)  251 ( 2)
••• End Xref: 52 id's   220 references       [412624 bytes/4947 id's/41684 refs]
```

[Segment 7]

Moving Boxes

Purpose of this segment:

1) To describe how to move a box across the view.
2) To introduce the concept of invalidation.

How to use this segment:

This is the seventh segment of the self—paced introduction to the ToolKit. This segment immediately follows the "Selections" segment. The next segment after this is "Creating A Box".

This segment concerns moving (or *dragging*) existing boxes across the screen. It comprises a tutorial and a lab.

So far you have been able to select and deselect one of a fixed number of boxes in your document. That capability will be enhanced in this segment to include the ability to drag a box, *unharmed*, from one place to another.

INTRODUCTION TO DRAGGING (moving an object)

With the aid of the mouse, users are able to drag objects within a window. This is very convenient way to organize items within a document.

The dragging procedure we will study involves the following steps:

1) Identify the selected box at the mouse down event.
2) At each mouse move event:
   a) Determine the box's new position.
   b) Erase the box at its old position.
   c) Redraw the box at its new position.

As you can see dragging is actually erasing and redrawing. An alternative procedure that we will not study erases on mouse down, redraws on mouse up, and XOR's twice at each mouse move. It is faster, but less impressive.

The Lisa user interface suggests a nice way to implement box dragging using the mouse. The mechanism is similar, though not identical, to that used by

LisaDraw. The procedure is to move the box the distance that the mouse moves. If the mouse cursor is dragged past a panel border, the view scrolls so the user can move the box further.

MOUSE MOVEMENTS

As moves the mouse so move our boxes. This section explains how mouse movements are detected and processed by the Generic Application.

Mouse movements are detected only while the mouse button is down. Once the mouse button is depressed, it is the panel's job to track movements of the mouse.

The panel translates mouse movements into **mouse move** events. *Bear in mind that these are not events in the strictest sense, since they are not generated by the window manager.* From the panel mouse move events flow to either the view or the selection.

Although the panel detects very minute movements of the mouse, the first few movements are not translated into events. The main reason for this is *hysteresis*.

hysteresis

"Behind cancer, hysteresis is the number two killer in this country." — a quack.

Hysteresis may sound bad, but it is really quite harmless. Quite simply, hysteresis is the distance the mouse must move before dragging can begin. It is a distance allowance for unsteady hands. This user friendly feature allows all but the most inept users to jiggle the mouse with the button down without disturbing the view.

Specifically, hysteresis is a set of two numbers. Each number represents a distance in pixels. These distances are applied to movements of the mouse to determine whether the first **mouse move** has occurred. One distance is applied horizontally, the other is applied vertically. The ToolKit constants, below, indicate the default hysteresis values:

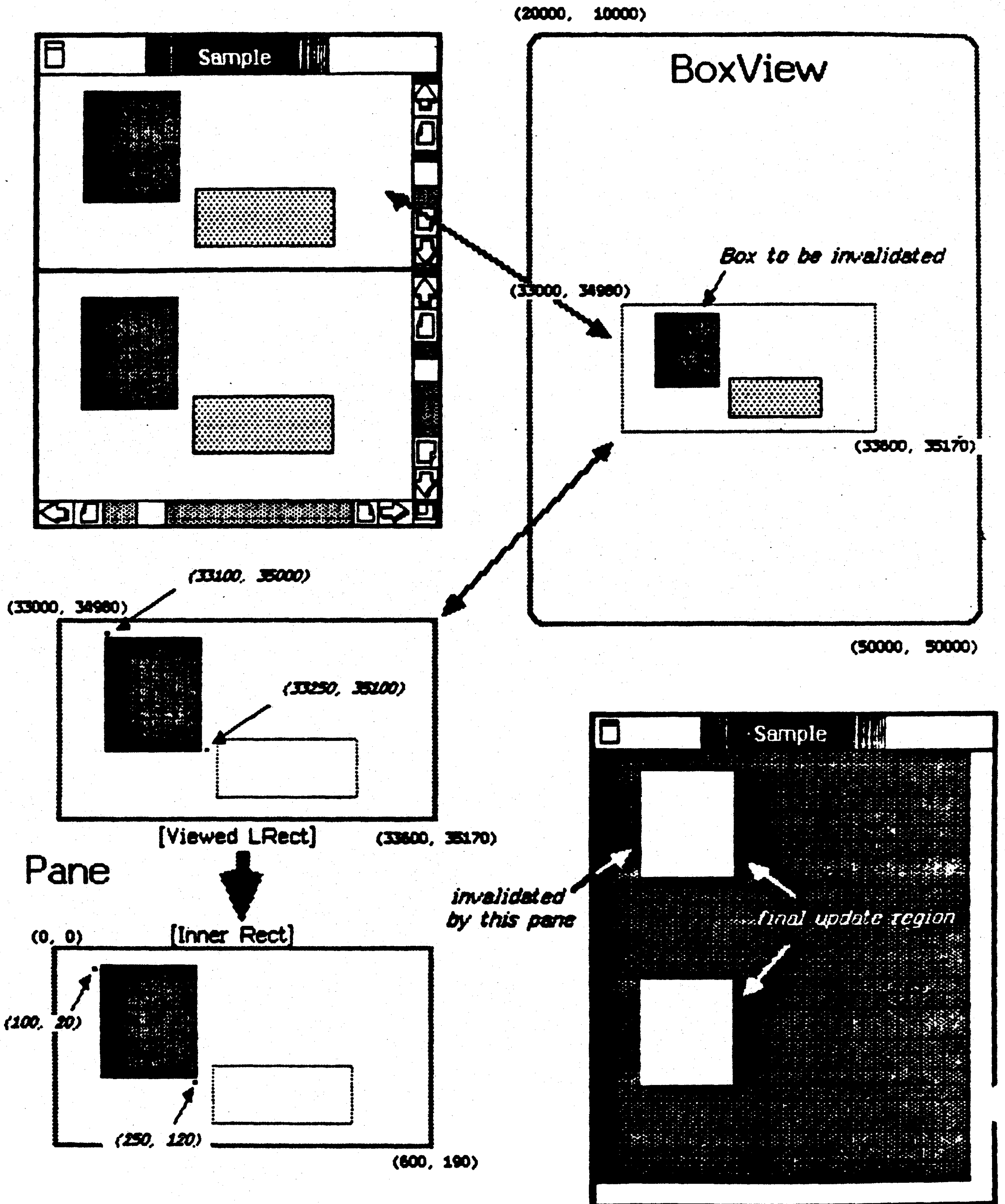
```
stdHHysteresis = 9;    {jiggle allowance horizontally}
stdVHysteresis = 6;    {jiggle allowance vertically}
```

If the mouse movement exceeds neither hysteresis value, horizontally or vertically, the mouse cursor coordinates reported are those of the original position (the selection's anchor LPoint).

Now that you know about mouse moves, the next concept essential to dragging boxes is *invalidation*.

INVALIDATION

Invalidation



Invalidation lets you erase and redraw boxes, or other graphical objects, in a clean and simple way. Particularly important when erasing, invalidation lets you easily restore the view obscured by the box.

Changes such as moving a box around in the view make the window's current display invalid. All or part of the display may need to be redrawn to reflect the altered contents of the view. Specifically, what will be changed are those portions of the screen displaying the areas of the view that the application has declared invalid.

(Note: Scrolling also invalidates the display, but it is taken care of automatically by the Generic Application. See the section "Scrolling" below for a more detailed explanation.)

Invalidation indicates the parts of the window that have become invalid because of a change in the view. Recall that the view is a Clascal object controlling the data objects to be displayed. The boxView's boxList field, for example, holds a list of boxes to be drawn. A box can be added to or removed from the list without immediately affecting the screen.

As you may have guessed, invalidation is intimately related to the window update region.

from LRect to Rect

Invalidation starts in the view, but ends in the window's update region. As the illustration: "Invalidation" shows, a 32-bit LRect in the view is invalidated by the application. This might be a box's shape LRect, for instance.

Next, for each pane in the view's panel the following occurs:

> The intersection of the invalid LRect with the pane's viewed LRect is translated into a corresponding 16-bit Rect (in window coordinates). This invalid Rect is a piece of the screen area to be updated in the window.
>
> If not empty, that invalid Rect is added to the update region. *Recall that the update region is initially empty. It is set to empty after each window update.*

Invalidation is complete when the last pane has added its invalid Rect to the update region.

building the update region

The application builds up the update region by invalidating one or more LRects in the view. As you recall from the previous segment, the update region is the area drawn into when the window updates the display.

The ToolKit method, {TPanel.}InvalLRect is used to invalidate an LRect in the view. The fragmented code below illustrates how this method is used:

```
PROCEDURE {TMySelection.} MouseMove({mouseLPt: LPoint}); {moves the box the distance
                                                          that the mouse has moved}

VAR box:   TBox;
    panel: TPanel;
```

```
BEGIN
   box := SELF.box;        {get the selected box}
   panel := SELF.panel;

              {invalidate the old position of the box}
   panel.InvalLRect(box.shapeLRect);

   {compute the distance the mouse has moved    (code omitted)
    compute new position of the box             (code omitted)
    offset the box's shape LRect appropriately (code omitted)}

              {invalidate the new position of the box}
   panel.InvalLRect(box.shapeLRect);
   {...}
END;
```

DETAILS OF THE WINDOW UPDATE

Once the update region has been built, the window is updated. After a mouse move event, the update is immediate.

One of the most relevant features of the update for box moves is that the update area is erased before redrawing occurs within it. This means that you only need to invalidate a moving box at its former position to erase it. To redraw, you simply invalidate the box at its new position.

The details of how the update works are listed below.

1. Install the update region as the visRgn; save the former visRgn.
2. If the update region is empty, then restore the former visRgn, and return to process the next event.

   Otherwise,
3. Prepare to highlight if needed.
4. Erase the update region by filling it with white.
5. Refresh the window frame (only parts within the update region).
6. Tell its panels to refresh themselves, each of which:

   tells its visible panes to refresh themselves, each of which:

   a. Frames itself, and focuses *(focusing clips to the intersection of the pane's inner Rect with the visRgn).*
   b. Positions the view frame properly within the pane.
   c. Has the view draw itself. *(view.Draw)*
   d. Has the selection highlight itself, if needed. *(selection.Highlight)*

e. Restores the former focus area *(this restores the clip region to its former state).*

7. Restore the former visRgn, and return to process the next event.

IMPLEMENTATION STRATEGY

We modify the previous stage of Boxer, 3Boxer, to support dragging. The user interface is summarized below.

user interface

Two boxes are displayed in the window. The user depresses the mouse button to select a box.

The user keeps the mouse button depressed to move the selected box. Once hysteresis is satisfied, each movement of the mouse causes the selected box to be moved proportionately.

The user releases the mouse button when finished moving the box. The box remains at its final position.

theory of operation

The implementation of dragging follows the algorithm below:

To move a box the application must do the following.

1. When the mouse button is pressed, set the selection anchor LPoint and current LPoint *(the current LPoint is the last place the mouse was tracked. Initially it is the same as the anchor LPoint.)* Determine the selected box.

The Generic Application takes over here. When hysteresis has been satisfied, it sends a mouse move event to the selection according the following flow of control:

```
pane.MouseTrack            {TPane.}
  view.MouseTrack          {TImage.}
    view.MouseMove         {TImage.}  default method
      selection.MouseMove  {TBoxSelection.}
```

Your code is called from here.

2. Extract the current mouse LPoint. Compute the amount the mouse has moved since the last mouse move event. This is done in {TBoxSelection.}MouseMove.
3. If the mouse has moved, then do the following:
   a. Update the selection's current LPoint.

b. Invalidate the box's old shape LRect (the erase phase).

c. Offset the box's shape LRect by the movement of the mouse.

d. Invalidate the box's new shape LRect.

The Generic Application now updates the window. As long as the mouse is down it continues to generate mouse move events. When the mouse button is released, the Generic Application makes the following calls:

```
pane.MouseTrack              {TPane.}
...
 view.MouseRelease           {TImage.}       default method
  selection.MouseRelease     {TSelection.}   default method
```

The default action in {TSelection.}MouseRelease is to do nothing. Your code does not need to change that.

actual implementation

The actual implementation for 4Boxer is summarized below. The code for 3Boxer is used as base for the changes.

New Classes

none

New Methods (for existing classes)

none

Overridden Methods

[TBoxSelection]

PROCEDURE {TBoxSelection.} MouseMove ((mouseLPt: LPoint));

{TBoxSelection.}MouseMove saves the mouse LPoint in SELF.currLPt, a field inherited from TSelection. Before it does that, though, it computes the difference between mouseLPt and SELF.currLPt. This is the amount that the box's shape LRect will be offset.

OFFSETTING A BOX

Offsetting a box means offsetting its shape LRect. The line below uses the ToolKit procedure, OffsetLRect, to offset a box by 30 pixels horizontally and 20 pixels vertically.

```
VAR  yourBox:  TBox;

{$H-}
```

```
OffsetLRect(yourBox.shapeLRect,30,20);
{$H+}
```

In 4Boxer the distance is computed as a point. This point is the difference between the mouse LPoint (mouseLPt) and the current LPoint (selection.currLPt). The ToolKit procedure, LPtMinusLPt, is used to compute the difference.

```
PROCEDURE {TBoxSelection.} MouseMove((mouseLPt: LPoint));
VAR  distance: LPoint;
{...}
            {the second LPoint is subtracted from the first to yield the difference,
             which is returned as the third LPoint}
LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
```

The following line now offsets the box, yourBox, by the distance computed above.

```
OffsetLRect(SELF.box.shapeLRect, diffLPt.h, diffLPt.v);
```

SCROLLING (Optional)

If the cursor passes the panel boundary and the panel was created with the ability to scroll, then the Generic Application scrolls automatically before each call on your mouse move method.

Questions:

1) Does dragging occur while the mouse button is up or down?
2) What is the difference between the selection's current LPoint and its anchor LPoint?
3) Requiring that the mouse moves a minimum distance is known as "enforcing hysteresis". While the mouse button is depressed, how often is hysteresis enforced?
4) Why invalidate a box's shape LRect to erase it instead of erasing the box's shape LRect directly?
5) How many invalidations are needed to move a box?

   What does the update region look like after that (those) invalidations?
6) How is an invalid LRect converted to an invalid Rect?

Box Move Lab

Purpose:

To implement dragging a box.

What you are about to do:

You will compile and run 4Boxer, then optionally modify the source. This should be done in the following steps:

1) Copy the following files onto your prefix volume.

   4UBoxer.TEXT
   4UBoxer2.TEXT
   4MBoxer.TEXT
   4PBoxer.TEXT

2) Compile, install, and run the sample application, **4Boxer**. Use **44** as the tool number.

3) Scan the listings of the four files in the sample application. These are included in the appendix, "Code Samples for this Segment".

4) *[Optional]* Extend the application you designed in step 4 of the "Selections" lab to support dragging multiple boxes.

   Please review the "Selections" lab.

Things to look out for:

- *Little black squares strewn all over the screen.*

  Remember to erase the box highlighting when erasing any box.

- *Box can be moved only a short distance.*

  Check whether you update selection.currLPt.

- *Boxes multiply like rabbits.*

  Check what you are actually invalidating.

Code Sample for this Segment

```
23
SLOT2CHAN1
;no assembler files
$
;no building blocks
$
;no links
$
y
y
n
BoxNum4
```

```
;    PBOXER.TEXT   for  Boxer
;Phrase file for Boxer class example
1
3
2500
$-#BOOT-TK/PABC

; Apple building block phrase files can be included here

1000
Boxer

; Other application alerts can be included here, numbered between 1001 and 32000

0

1
$-#BOOT-TK/PABC~File/Print

2
Page Layout
Preview Actual Pages#401
Preview Page Breaks#402
Don't Preview Pages#403

100
Buzzwords
Set Aside ↑Document↑#109

0
```

```
PROGRAM M4Boxer;

USES
    {$U UObject     } UObject,

  {$IFC libraryVersion <= 20}
    {$U UFont}              UFont,
  {$ENDC}

    {$U QuickDraw   } QuickDraw,
    {$U UDraw       } UDraw,
    {$U UABC        } UABC,

    {$U U4Boxer     } U4Boxer;

CONST

    phraseVersion = 1;

BEGIN

    process := TBoxProcess.CREATE;
    process.Commence(phraseVersion);
    process.Run;
    process.Complete(TRUE);

END.
```

```
1    1 --   UNIT U4Boxer;
1    2 --
1    3 --   INTERFACE
1    4 --
1    5 --   USES
1    6 --       {$U UObject}              UObject,
1    7 --
1    8 --     {$IFC libraryVersion <= 20}
1    9 --       {$U UFont}              UFont,
1   10 --     {$ENDC}
1   11 --
1   12 --       {$U QuickDraw}          QuickDraw,
1   13 --       {$U UDraw}              UDraw,
1   14 --       {$U UABC}               UABC;
1   15 --
1   16 --   CONST
1   17 --       colorWhite = 1;
1   18 --       colorLtGray = 2;
1   19 --       colorGray = 3;
1   20 --       colorDkGray = 4;
1   21 --       colorBlack = 5;
1   22 --
1   23 --       boxSelectionKind = 1;
1   24 --
1   25 --   TYPE
1   26 --
1   27 --       TColor = colorWhite..colorBlack; {color of a box}
1   28 --
1   29 --     {New Classes for this Application}
1   30 --
1   31 --       TBox = SUBCLASS OF TObject
1   32 --
1   33 --           {Variables}
1   34 --           shapeLRect:       LRect;
1   35 --           color:            TColor;
1   36 --
1   37 --         {Creation/Destruction}
1   38 --           FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
1   39 --
1   40 --           PROCEDURE TBox.PaintHandles;
1   41 --           PROCEDURE TBox.Draw;
1   42 --           END;
1   43 --
1   44 --
1   45 --       TBoxView = SUBCLASS OF TView
1   46 --
1   47 --         {Variables}
1   48 --           boxList:          TList;
1   49 --
1   50 --         {Creation/Destruction}
1   51 --           FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
1   52 --                                     : TBoxView;
1   53 --
1   54 --          FUNCTION  TBoxView.BoxWith(LPt: LPoint): TBox;
1   55 --
1   56 --           PROCEDURE TBoxView.Draw; OVERRIDE;
1   57 --           PROCEDURE TBoxView.InitBoxList(itsHeap: THeap);
1   58 --           FUNCTION  TBoxView.NoSelection: TSelection; OVERRIDE;
1   59 --          END;
1   60 --
1   61 --
1   62 --       TBoxSelection = SUBCLASS OF TSelection
1   63 --
1   64 --         {Variables}
1   65 --           box:   TBox;
1   66 --
1   67 --         {Creation/Destruction}
1   68 --           FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsKind: INTEGER;
1   69 --                                          itsAnchorLPt: LPoint): TBoxSelection;
1   70 --
1   71 --         {Drawing - per pad}
1   72 --           PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit); OVERRIDE;
1   73 --
1   74 --         {Selection - per pad}
1   75 --           PROCEDURE TBoxSelection.MousePress(mouseLPt: LPoint); OVERRIDE;
1   76 --           PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1   77 --       END;
1   78 --
1   79 --
1   80 --       TBoxProcess = SUBCLASS OF TProcess
1   81 --
1   82 --         {Creation/Destruction}
1   83 --           FUNCTION  TBoxProcess.CREATE: TBoxProcess;
1   84 --           FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
1   85 --                                                : TDocManager; OVERRIDE;
1   86 --           END;
1   87 --
1   88 --
1   89 --       TBoxDocManager = SUBCLASS OF TDocManager
1   90 --
1   91 --         {Creation/Destruction}
1   92 --           FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
1   93 --                                              : TBoxDocManager;
1   94 --           FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID): TWindow; OVERRIDE;
1   95 --           END;
1   96 --
1   97 --
1   98 --
1   99 --       TBoxWindow = SUBCLASS OF TWindow
1  100 --
1  101 --         {Variables}
1  102 --
1  103 --         {Creation/Destruction}
1  104 --           FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
1  105 --
1  106 --         {Document Creation}
1  107 --           PROCEDURE TBoxWindow.BlankStationery; OVERRIDE;
1  108 --           END;
1  109 --
1  110 --
```

```
1  111 --    IMPLEMENTATION
1  112 --
1  113 --      {$I U4Boxer2.text}
2    1 --    {U4BOXER2}
2    2 --
2    3 --    METHODS OF TBox;
2    4 --
2    5 -- A      FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
2    6 0- A      BEGIN
2    7 --            {$IFC fTrace}BP(11);{$ENDC}
2    8 --            SELF := NewObject(itsHeap, THISCLASS);
2    9 --            WITH SELF DO
2   10 1-                BEGIN
2   11 --                shapeLRect := zeroLRect;
2   12 --                color := colorGray;
2   13 -1                END;
2   14 --            {$IFC fTrace}EP;{$ENDC}
2   15 -0 A      END;
2   16 --
2   17 --       {This draws a particular box}
2   18 -- A      PROCEDURE TBox.Draw;
2   19 --        VAR lPat:    LPattern;
2   20 0- A      BEGIN
2   21 --            {$IFC fTrace}BP(10);{$ENDC}
2   22 --            PenNormal;
2   23 --
2   24 --            IF LRectIsVisible(SELF.shapeLRect) THEN {this box needs to be drawn}
2   25 1-                BEGIN
2   26 --              {Get a Quickdraw pattern to represent the box's color}
2   27 2-                CASE SELF.color OF
2   28 --                    colorWhite:     lPat := lPatWhite;
2   29 --                    colorLtGray:    lPat := lPatLtGray;
2   30 --                    colorGray:      lPat := lPatGray;
2   31 --                    colorDkGray:    lPat := lPatDkGray;
2   32 --                    colorBlack:     lPat := lPatBlack;
2   33 --                    OTHERWISE       lPat := lPatWhite; {this case should not happen}
2   34 -2                    END;
2   35 --
2   36 --               {Fill the box with the pattern, and draw a frame around it}
2   37 --                FillLRect(SELF.shapeLRect, lPat);
2   38 --                FrameLRect(SELF.shapeLRect);
2   39 -1                END;
2   40 --            {$IFC fTrace}EP;{$ENDC}
2   41 -0 A      END;
2   42 --
2   43 --        {This calls the DoToHandle Procedure once for each handle LRect; user of this method must
2   44 --         set up the pen pattern and mode before calling}
2   45 -- A      PROCEDURE TBox.PaintHandles;
2   46 --        VAR hLRect,
2   47 --            shapeLRect:  LRect;
2   48 --            dh, dv:      LONGINT;
2   49 --
2   50 -- B          PROCEDURE MoveHandleAndPaint(hOffset, vOffset: LONGINT);
2   51 0- B          BEGIN
2   52 --                OffsetLRect(hLRect, hOffset, vOffset);
2   53 --                PaintLRect(hLRect);
2   54 -0 B          END;
2   55 --
2   56 0- A      BEGIN
2   57 --            {$IFC fTrace}BP(10);{$ENDC}
2   58 --            SetLRect(hLRect, -3, -2, 3, 2);
2   59 --            shapeLRect := SELF.shapeLRect;
2   60 --            WITH shapeLRect DO
2   61 1-                BEGIN
2   62 --                dh := right - left;
2   63 --                dv := bottom - top;
2   64 --                MoveHandleAndPaint(left, top);     {draw top left handle}
2   65 -1                END;
2   66 --            MoveHandleAndPaint(dh, 0);             {then top right}
2   67 --            MoveHandleAndPaint(0, dv);             {then bottom right}
2   68 --            MoveHandleAndPaint(-dh, 0);            {finally bottom left}
2   69 --            {$IFC fTrace}EP;{$ENDC}
2   70 -0 A      END;
2   71 --
2   72 --    END;
2   73 --
2   74 --
2   75 --    METHODS OF TBoxView;
2   76 --
2   77 -- A      FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
2   78 --                                    : TBoxView;
2   79 0- A      BEGIN
2   80 --            {$IFC fTrace}BP(11);{$ENDC}
2   81 --            IF object = NIL THEN
2   82 --                object := NewObject(itsHeap, THISCLASS);
2   83 --            SELF := TBoxView(itsPanel.NewView(object, itsExtent, TPrintManager.CREATE(NIL, itsHeap),
2   84 --                                              stdMargins, TRUE));
2   85 --            {$IFC fTrace}EP;{$ENDC}
2   86 -0 A      END;
2   87 --
2   88 --
2   89 --       {This returns the box containing a certain point}
2   90 -- A      FUNCTION TBoxView.BoxWith(LPt: LPoint): TBox;
2   91 --        VAR box:     TBox;
2   92 --            s:       TListScanner;
2   93 0- A      BEGIN
2   94 --            {$IFC fTrace}BP(11);{$ENDC}
2   95 --            boxWith := NIL;
2   96 --            s := SELF.boxList.Scanner;
2   97 --            WHILE s.Scan(box) DO
2   98 --                IF LPtInLRect(LPt, box.shapeLRect) THEN
2   99 --                    boxWith := box;
2  100 --            {$IFC fTrace}EP;{$ENDC}
2  101 -0 A      END;
2  102 --
2  103 --
2  104 --      {This draws the list of boxes}
2  105 -- A      PROCEDURE TBoxView.Draw;
2  106 --        VAR box:     TBox;
2  107 --            s:       TListScanner;
```

```
2  108 0- A      BEGIN
2  109 --            {$IFC fTrace}BP(10);{$ENDC}
2  110 --            s := SELF.boxList.Scanner;
2  111 --            WHILE s.Scan(box) DO
2  112 --                box.Draw;
2  113 --            {$IFC fTrace}EP;{$ENDC}
2  114 -0 A      END;
2  115 --
2  116 --
2  117 -- A      PROCEDURE TBoxView.InitBoxList (itsHeap: THeap);
2  118 --        VAR  box: TBox;
2  119 --             boxList: TList;
2  120 0- A      BEGIN
2  121 --            {$IFC fTrace}BP(10);{$ENDC}
2  122 --            boxList := TList.CREATE(NIL, itsHeap, 0);
2  123 --            SELF.boxList := boxList;
2  124 --
2  125 --                                        {create and append the first box}
2  126 --            box := TBox.CREATE(NIL, itsHeap);
2  127 --        {$H-} SetLRect(box.shapeLRect, 20,20, 100, 100);  {$H+}
2  128 --            SELF.boxList.InsLast(box);
2  129 --
2  130 --                                        {create and append the second box}
2  131 --            box := TBox.CREATE(NIL, itsHeap);
2  132 --        {$H-} SetLRect(box.shapeLRect, 200, 100, 300, 130);  {$H+}
2  133 --            SELF.boxList.InsLast(box);
2  134 --            {$IFC fTrace}EP;{$ENDC}
2  135 -0 A      END;
2  136 --
2  137 --
2  138 -- A      FUNCTION TBoxView.NoSelection: TSelection;
2  139 0- A      BEGIN
2  140 --           {$IFC fTrace}BP(11);{$ENDC}
2  141 --            NoSelection := TBoxSelection.CREATE(NIL, SELF.Heap, SELF, nothingKind, zeroLPt);
2  142 --            {$IFC fTrace}EP;{$ENDC}
2  143 -0 A      END;
2  144 --
2  145 --    END;
2  146 --
2  147 --
2  148 --
2  149 --    METHODS OF TBoxSelection;
2  150 --
2  151 -- A      FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsKind: INTEGER;
2  152 --                                       itsAnchorLPt: LPoint): TBoxSelection;
2  153 0- A      BEGIN
2  154 --            {$IFC fTrace}BP(11);{$ENDC}
2  155 --            IF object = NIL THEN
2  156 --                object := NewObject(itsHeap, THISCLASS);
2  157 --            SELF := TBoxSelection(TSelection.CREATE(object, itsHeap, itsView, itsKind, itsAnchorLPt));
2  158 --
2  159 --            SELF.box := NIL;
2  160 --            {$IFC fTrace}EP;{$ENDC}
2  161 -0 A      END;
2  162 --
2  163 --
2  164 --        {This draws the handles on the selected box}
2  165 -- A      PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit);
2  166 0- A      BEGIN
2  167 --            {$IFC fTrace}BP(11);{$ENDC}
2  168 --            IF SELF.kind <> nothingKind THEN
2  169 1-                BEGIN
2  170 --                thePad.SetPenToHighlight(highTransit); {set the drawing mode according to desired highlighting}
2  171 --                SELF.box.PaintHandles;                 {draw the handles on the box}
2  172 -1                END;
2  173 --            {$IFC fTrace}EP;{$ENDC}
2  174 -0 A      END;
2  175 --
2  176 --
2  177 --       {This is another way to make a new selection, when the user presses the mouse button}
2  178 --       {Just keep the old selection object and replace its data fields with new values.  This isn't the}
2  179 --       {standard paradigm for creating new selection objects, but it certainly works (at least in this case).}
2  180 -- A      PROCEDURE TBoxSelection.MousePress(mouseLPt: LPoint);
2  181 --        VAR boxView:    TBoxView;
2  182 --            aSelection: TSelection;
2  183 --            panel:      TPanel;
2  184 --            box:        TBox;
2  185 0- A      BEGIN
2  186 --           {$IFC fTrace}BP(11);{$ENDC}
2  187 --            WITH SELF DO
2  188 1-                BEGIN
2  189 --                anchorLPt := mouseLPt;
2  190 --                currLPt := mouseLPt;
2  191 -1                END;
2  192 --
2  193 --            boxView := TBoxView(SELF.view);
2  194 --            panel := SELF.panel;
2  195 --            panel.Highlight(SELF, hOntoOff);        {Turn off the old highlighting}
2  196 --
2  197 --            box := boxView.BoxWith(mouseLPt);       {Find the box the user clicked on}
2  198 --            IF box = NIL THEN
2  199 --                SELF.kind := nothingKind
2  200 --            ELSE
2  201 --                SELF.kind := boxSelectionKind;
2  202 --            SELF.box := box;
2  203 --
2  204 --            panel.Highlight(SELF, hOffToOn);        {Turn on the highlighting for the newly selected box}
2  205 --
2  206 --            self.MarkChanged;   {Allow the document to be saved so that any changes made can become permanent}
2  207 --
2  208 --           {$IFC fTrace}EP;{$ENDC}
2  209 -0 A      END;
2  210 --
2  211 --
2  212 --       {This is called when the user moves the mouse after pressing the button}
2  213 -- A      PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint);
2  214 --        VAR diffLPt:    LPoint;
2  215 --            shapeLRect: LRect;
2  216 --
2  217 -- B          PROCEDURE InvalTheBox(invalRect: LRect);
```

```
2  218 0- B          BEGIN
2  219 --            {$IFC fTrace}BP(11);{$ENDC}
2  220 --                InsetLRect(invalRect, -3,-2);       {need to expand the invalidation rectangle to invalidate }
2  221 --                SELF.panel.InvalLRect(invalRect);   {highlighting as well as box}
2  222 --            {$IFC fTrace}EP;{$ENDC}
2  223 -0 B          END;
2  224 --
2  225 0- A      BEGIN
2  226 --           {$IFC fTrace}BP(11);{$ENDC}
2  227 --            IF SELF.kind <> nothingKind THEN
2  228 1-                BEGIN
2  229 --                        {How far did mouse move?}
2  230 --                LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
2  231 --
2  232 --                        {Move it if delta is nonzero}
2  233 --                IF NOT EqualLPt(diffLPt, zeroLPt) THEN
2  234 2-                    BEGIN
2  235 --                    SELF.currLPt := mouseLPt;
2  236 --
2  237 --                    shapeLRect := SELF.box.shapeLRect;
2  238 --                        {Compute old and new positions of box}
2  239 --                    InvalTheBox(shapeLRect);
2  240 --                    OffsetLRect(shapeLRect, diffLPt.h, diffLPt.v);
2  241 --                    InvalTheBox(shapeLRect);
2  242 --
2  243 --                    SELF.box.shapeLRect := shapeLRect;
2  244 -2                    END
2  245 -1                END;
2  246 --            {$IFC fTrace}EP;{$ENDC}
2  247 -0 A      END;
2  248 --
2  249 --    END;
2  250 --
2  251 --
2  252 --
2  253 --    METHODS OF TBoxProcess;
2  254 --
2  255 -- A      FUNCTION  TBoxProcess.CREATE: TBoxProcess;
2  256 0- A      BEGIN
2  257 --            {$IFC fTrace}BP(11);{$ENDC}
2  258 --            SELF := TBoxProcess(TProcess.CREATE(NewObject(mainHeap, THISCLASS), mainHeap));
2  259 --            {$IFC fTrace}EP;{$ENDC}
2  260 -0 A      END;
2  261 --
2  262 --
2  263 -- A      FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN): TDocManager;
2  264 0- A      BEGIN
2  265 --            {$IFC fTrace}BP(11);{$ENDC}
2  266 --            NewDocManager := TBoxDocManager.CREATE(NIL, mainHeap, volumePrefix);
2  267 --            {$IFC fTrace}EP;{$ENDC}
2  268 -0 A      END;
2  269 --
2  270 --    END;
2  271 --
2  272 --
2  273 --
2  274 --    METHODS OF TBoxDocManager;
2  275 --
2  276 -- A      FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
2  277 --                                              : TBoxDocManager;
2  278 0- A      BEGIN
2  279 --            {$IFC fTrace}BP(11);{$ENDC}
2  280 --            IF object = NIL THEN
2  281 --                object := NewObject(itsHeap, THISCLASS);
2  282 --            SELF := TBoxDocManager(TDocManager.CREATE(object, itsHeap, itsPathPrefix));
2  283 --            {$IFC fTrace}EP;{$ENDC}
2  284 -0 A      END;
2  285 --
2  286 --
2  287 -- A      FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID):TWindow;
2  288 0- A      BEGIN
2  289 --            {$IFC fTrace}BP(11);{$ENDC}
2  290 --            NewWindow := TBoxWindow.CREATE(NIL, heap, wmgrID);
2  291 --            {$IFC fTrace}EP;{$ENDC}
2  292 -0 A      END;
2  293 --
2  294 --    END;
2  295 --
2  296 --
2  297 --
2  298 --    METHODS OF TBoxWindow;
2  299 --
2  300 -- A      FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
2  301 0- A      BEGIN
2  302 --            {$IFC fTrace}BP(10);{$ENDC}
2  303 --            IF object = NIL THEN
2  304 --                object := NewObject(itsHeap, THISCLASS);
2  305 --            SELF := TBoxWindow(TWindow.CREATE(object, itsHeap, itsWmgrID, TRUE));
2  306 --            {$IFC fTrace}EP;{$ENDC}
2  307 -0 A      END;
2  308 --
2  309 --
2  310 --
2  311 -- A      PROCEDURE TBoxWindow.BlankStationery;
2  312 --        VAR viewLRect:      LRect;
2  313 --            panel:          TPanel;
2  314 --            boxView:        TBoxView;
2  315 --            aSelection:     TBoxSelection;
2  316 --
2  317 0- A      BEGIN
2  318 --            {$IFC fTrace}BP(10);{$ENDC}
2  319 --            panel := TPanel.CREATE(NIL, SELF.Heap, SELF, 0, 0, [aScroll, aSplit], [aScroll, aSplit]);
2  320 --
2  321 --            SetLRect(viewLRect, 0, 0, 5000, 3000);
2  322 --            boxView := TBoxView.CREATE(NIL, SELF.Heap, panel, viewLRect);
2  323 --            boxView.InitBoxList(SELF.Heap);
2  324 --
2  325 --            {$IFC fTrace}EP;{$ENDC}
2  326 -0 A      END;
2  327 --
```

```
2  328 --    END;
2  329 --
1  114 --
1  115 --    END.
```

```
 1. u4boxer.TEXT
 2. U4Boxer2.text

-B-
  BlankStationery   107•( 1)  311•( 2)
  box                65 ( 1)   91•( 2)   97 ( 2)   98 ( 2)   99 ( 2)  106•( 2)  111 ( 2)  112 ( 2)  118•( 2)  126=( 2)
                    127 ( 2)  128 ( 2)  131=( 2)  132 ( 2)  133 ( 2)  159=( 2)  171 ( 2)  184•( 2)  197=( 2)  198 ( 2)
                    202=( 2)  202 ( 2)  237 ( 2)  243 ( 2)
  boxList            48 ( 1)   96 ( 2)  110 ( 2)  119•( 2)  122=( 2)  123=( 2)  123 ( 2)  128 ( 2)  133 ( 2)
  boxSelectionKind   23•( 1)  201 ( 2)
  BoxWith            54•( 1)   90•( 2)  197 ( 2)

-C-
  color              35•( 1)   12=( 2)   27 ( 2)
  colorBlack         21•( 1)   27 ( 1)   32 ( 2)
  colorDkGray        20•( 1)   31 ( 2)
  colorGray          19•( 1)   12 ( 2)   30 ( 2)
  colorLtGray        18•( 1)   29 ( 2)
  colorWhite         17•( 1)   27 ( 1)   28 ( 2)
  CREATE             38•( 1)   51•( 1)   68•( 1)   83 ( 1)   92 ( 1)  104 ( 1)    5•( 2)   77•( 2)   83 ( 2)  122 ( 2)
                    126 ( 2)  131 ( 2)  141 ( 2)  151•( 2)  157 ( 2)  255•( 2)  258 ( 2)  266 ( 2)  276•( 2)  282 ( 2)
                    290 ( 2)  300•( 2)  305 ( 2)  319 ( 2)  322 ( 2)

-D-
  Draw               41•( 1)   56•( 1)   18•( 2)  105•( 2)  112 ( 2)

-H-
  Highlight          72•( 1)  165•( 2)  195 ( 2)  204 ( 2)

-I-
  InitBoxList        57•( 1)  117•( 2)  323 ( 2)

-L-
  LRect              34 ( 1)   47 ( 2)  215 ( 2)  217 ( 2)  312 ( 2)

-M-
  MouseMove          76•( 1)  213•( 2)
  MousePress         75•( 1)  180•( 2)

-N-
  NewDocManager      84•( 1)  263•( 2)  266=( 2)
  NewWindow          94•( 1)  287•( 2)  290=( 2)
  NoSelection        58•( 1)  138•( 2)  141=( 2)

-P-
  PaintHandles       40•( 1)   45•( 2)  171 ( 2)

-Q-
  QuickDraw          12•( 1)

-S-
  shapeLRect         34 ( 1)   11=( 2)   24 ( 2)   37 ( 2)   38 ( 2)   47•( 2)   59=( 2)   59 ( 2)   60 ( 2)   98 ( 2)
                    127 ( 2)  132 ( 2)  215•( 2)  237=( 2)  237 ( 2)  239 ( 2)  240 ( 2)  241 ( 2)  243=( 2)  243 ( 2)

-T-
  TBox               31•( 1)   38 ( 1)   54 ( 1)   65 ( 1)    3•( 2)    5 ( 2)   90 ( 2)   91 ( 2)  106 ( 2)  118 ( 2)
                    126 ( 2)  131 ( 2)  184 ( 2)
  TBoxDocManager     89•( 1)   93 ( 1)  266 ( 2)  274•( 2)  277 ( 2)  282 ( 2)
  TBoxProcess        80•( 1)   83 ( 1)  253•( 2)  255 ( 2)  258 ( 2)
  TBoxSelection      62•( 1)   69 ( 1)  141 ( 2)  149•( 2)  152 ( 2)  157 ( 2)  315 ( 2)
  TBoxView           45•( 1)   52 ( 1)   75•( 2)   78 ( 2)   83 ( 2)  181 ( 2)  193 ( 2)  314 ( 2)  322 ( 2)
  TBoxWindow         99•( 1)  104 ( 1)  290 ( 2)  298•( 2)  300 ( 2)  305 ( 2)
  TColor             27•( 1)   35 ( 1)
  TDocManager        85 ( 1)   89 ( 1)  263 ( 2)  282 ( 2)
  TList              48 ( 1)  119 ( 2)  122 ( 2)
  TObject            31 ( 1)
  TProcess           80 ( 1)  258 ( 2)
  TSelection         58 ( 1)   62 ( 1)  138 ( 2)  157 ( 2)  182 ( 2)
  TView              45 ( 1)
  TWindow            94 ( 1)   99 ( 1)  287 ( 2)  305 ( 2)

-U-
  U4Boxer             1•( 1)
  UABC               14•( 1)
  UDraw              13•( 1)
  UFont               9•( 1)
  UObject             6•( 1)

*** End Xref:  43 id's   208 references       [412360 bytes/4956 id's/41633 refs]
```

[Segment 8]

Creating a Box
(A Second Selection Class)

Purpose of this segment:

1) To add the ability to create a box.
2) To present the conditions warranting multiple selection classes.
3) To explain how to use the view to arbitrate among selections.

How to use this segment:

This is the eighth segment of the self—paced introduction to the ToolKit. This segment follows the segment, "Moving Boxes", and precedes the segment on commands with undo.

This segment implements multiple selections. It explains when multiple selection classes are useful; and how to specify them.

Having just implemented dragging, the next stage in the Boxer application is to enable users to create new boxes. Once created, those boxes can be selected and dragged around the window.

More so than the previous segments, in this one the user interface plays a critical role in the application design. We start by considering the problem of creating a box.

CREATING A BOX

Users of this stage of Boxer are to be able to create boxes. The design task at hand is to make the creation of boxes both natural and intuitive for the user.

We start by considering how people create boxes on a universally familiar medium — drawing paper.

Tanya is about to draw some boxes for a garden design project. The first box she is adding is the outer boundary of the garden. She locates a suitable starting point on the sheet of paper, and sketches a box of the desired

size. Notice that she has control over both the location and the size of the box.

She now starts to add a second box. This will be a planting area for beans. It will reside within the first box, and will be appreciably smaller. As with the first box, she locates a starting point, then sketches the new box. She follows this basic procedure with every box she adds to the garden design.

Two application design options come to mind. The first is to mimic the above process as closely as possible. This means that users can create boxes of arbitrary size from any starting point on the screen. The second option is more limiting. It allows users to sketch a box, but only from a starting point not within any other box.

We consider the first option — creating a box anywhere on the screen.

user interface options

There are two kinds of places where the user can locate the starting point of a box. The starting point can be either inside of a box or not inside of any box. If the starting point is not enclosed by any other box, then we can proceed to sketch it without reservation.

Yet if the starting point is inside of another box, we have a problem. Unless we know beforehand that the user is creating a box, we have no way of distinguishing it from a box selection *of the type dealt with in the previous two segments.* Can the user let us know that she will be drawing a box before she actually does it? And if so, then how?

The answer to the first question is yes; and several proposals arise in answer to the second. Of these, we can immediately dismiss using a menu command to initiate a box creation mode. For reasons best explained in the Lisa User Interface Guidelines, this is prohibited. A more likely proposal is to implement a control which, minimally, would have two states — box creation and box selection.

A control is a special figure or table that the user can edit or adjust to pass information to an application. Typically controls are used to communicate information that will not be used immediately.

A good example of a control we could use is the pallette in LisaDraw. For Boxer we would only need the *box create* and *drag* indicators. All things considered though, a pallette is a good idea for a more complex application, but for Boxer, it is a bit excessive. A simpler control, such as a two position gauge would be more appropriate.

Yet, despite the advantages, a control is not implemented in Boxer. Instead we pursue the design simplicity of the second implementation option — creating a box only where none exists. LisaProject works this way.

With this option the criterion for creating a box is reduced to a simple test. Is there or is there not a box at the mouse press LPoint? If a box is present, we

interpret the mouse press to be a box selection. If none is present, the mouse press is the starting point for creating a box.

CREATING A BOX

Conceptually a box exists at the moment of creation. Initially this box has no dimensions, since it contains only a single point — the mouse press LPoint.

To grow the box, we follow the example of LisaDraw. We retain the mouse press point as one corner of the box, and use mouse moves to position the far corner of the box.

We use the mouse release to indicate the final position of the box's far corner. At this point the creation is now a full-fledged box. As with other boxes it can be selected, highlighted, and dragged.

Because the responses to mouse moves and mouse releases diverge from those of box selections, box creation is handled as a new class of selection. This is the recommended application design. The table below lists the major behavioral differences between box creation selections and box selections.

	box creation	box selection
CREATE	create a new box	refer to selected box
mouse move	grow the box	move the box
mouse release	end box creation	*(no action)*
highlighting	frame growing box	draw handles around box

IMPLEMENTATION STRATEGY

We modify the previous stage of the Boxer application, 4Boxer, to enable users to create boxes. The user interface is summarized below.

user interface

No box is initially in the view. The user must create all the boxes he or she will use.

The user presses the mouse button at a point inside of no other box to initiate box creation. The box starts as a single point. With the mouse button down, the one corner of the box follows the movements of the mouse. The other corner remains fixed where the mouse went down.

A temporary frame is drawn around the box boundaries after each mouse move. When the user releases the mouse button, the floating corner of the box is fixed at the last mouse move point. As with LisaDraw, if this corner is not a minimal distance from the fixed corner, the creation is nullified, and the screen is restored to its former state.

If of suitable size, a permanent frame is drawn around the box. It is colored the default color, gray. The new box immediately becomes the selected box. As such it is highlighted with tiny handles at its corners.

design considerations

What is the relationship between our two classes of selections? Is one a subclass of the other, or are they both direct descendants of **TSelection**? The subclassing rule of thumb is that if capabilities are not clearly shared between two classes, then make the new class a subclass of the nearest ancestor. In the case of our two selection classes, that ancestor is **TSelection**.

arbitrating between selections

The mouse press determines the class of the selection to be created. We use a built—in ToolKit mechanism to arbitrate between the two selections. As the flow of control below indicates, we can use the **MousePress** method of the view to create the appropriate selection.

[the mouse is pressed]

...

```
  pane.MouseTrack      {TPane.}
    view.MouseTrack    {TView.}
      view.MousePress  {TBoxView.}   overrides the default method in TView
```

Once the selection is created, mouse move events are then routed to that selection.

theory of operation

The implementation of box creation follows the algorithm below:

1. When the mouse button is depressed, determine whether any box is selected. The determination is done by {TBoxView.}**BoxWith**. The mouse press is processed by {TBoxView.}**MousePress**.

   If one is selected, then free—and—replace the current selection with a new instance of **TBoxSelection**.

   If none is selected, then free—and—replace the current selection with a new instance of **TCreateBoxSelection**. This selection creates a box object as one of its data fields. The selection anchor point becomes the fixed vertex of the box.

The Generic Application then routes any mouse move events to that selection. For createBoxSelections the algorithm proceeds as follows:

2. If the mouse has moved then:

   a. Undraw the frame of the box at its current size. This has no effect if the box's shape LRect is a zero LRect.

   b. Set the box's far vertex to the mouse LPoint.

   c. Draw the frame of the resized box.

The Generic Application now updates the window. The process repeats from step 2 as long as the mouse button is down. When the mouse button is released the steps proceed as follows:

3. Undraw the frame of the box at its current (and final) size.
4. Inválidate the box's shape LRect for the window update.
5. Free—and—replace the createBoxSelection with a new instance of **TBoxSelection**, whose anchor LPoint is the fixed corner of the box.
6. If the box is not big enough then throw it away. This is done by setting the kind field of the boxSelection to nothingKind; and by freeing the undersized box.

   Otherwise append the box to the view's box list.

When the Generic Application updates the window at this point, the newly created box is drawn with its default color, then highlighted. If the box was thrown away, the update restores the display underneath the shape LRect of the freed box.

actual implementation

The actual implementation for 5Boxer is summarized below. The code for 4Boxer is used as the base for all changes.

New Constants

```
createBoxSelectionKind = 2;
```

New Classes

```
[TCreateBoxSelection]

TCreateBoxSelection = SUBCLASS OF TSelection
    {fields}
  box: TBox;      {references the box being created}
  {Creation}
FUNCTION  {TCreateBoxSelection.} CREATE(object: TObject; itsHeap: THeap; itsView: TView;
                                        itsAnchorLPt: LPoint): TCreateBoxSelection;
```

{TCreateBoxSelection.}CREATE creates a createBoxSelection with the given anchor LPoint and with kind, createBoxSelectionKind.

```
PROCEDURE {TCreateBoxSelection.} MouseMove(mouseLPt: LPoint);
```

{TCreateBoxSelection.}**MouseMove** ties the far (diagonal) corner of the box to the mouse LPoint. It unframes the box using its old size, and reframes the box using its new size.

```
PROCEDURE {TCreateBoxSelection.} MouseRelease;
```

{TCreateBoxSelection.}**MouseRelease** unframes the box and prepares to redraw it as a normal box. The selection replaces

itself with a new boxSelection referring to the newly created box.

New Methods (for existing classes)

[TBox]

```
PROCEDURE {TBox.} DrawFrame;
```

{TBox.}**DrawFrame** frames the box being created in a reversible way. It uses the QuickDraw pen pattern, patXOr. The framing is performed by FrameLRect.

[TBoxView]

```
PROCEDURE {TBoxView.} InvalBox(invalLRect: LRect);
```

{TBoxView.}**InvalBox** invalidates the created box's shape LRect and enough border to include the highlighting. This allows the invalidation code to be shared.

Overridden Method

[TBoxView]

```
PROCEDURE {TBoxView.} MousePress(mouseLPt: LPoint);
```

{TBoxView.}**MousePress** finds the box the user clicked on. If NIL then it creates a new createBoxSelection. Otherwise it creates a box selection.

Modified Method

[TBoxSelection]

```
PROCEDURE {TBoxSelection.}MouseMove(mouseLPt: LPoint);
```

{TBoxSelection.}**MouseMove** now uses {TBoxView.}**InvalBox** to invalidate boxes.

Deleted Method

[TBoxSelection]

```
PROCEDURE {TBoxSelection.}MousePress(mouseLPt: LPoint);
```

obviated by {TBoxView.}**MousePress**.

FRAMING IN MULTIPLE PANES

Editing operations such as framing a box must work correctly even if the user chooses to split the panel into several panes. A special method in the ToolKit is supplied specifically for this purpose. That method is {TPanel.}**OnAllPadsDo**. Its declaration is listed below:

```
PROCEDURE {TPanel.}OnAllPadsDo(PROCEDURE DoOnThePad);
```

The sole parameter to **OnAllPadsDo** is a procedure, a parameterless procedure. When called, this method executes that procedure on each of the panes in the panel. Each pane that is visible will be focused before executing the procedure.

OnAllPadsDo is typically used to draw reversibly on the display. The code excerpt below demonstrates such a use.

```
PROCEDURE {TCreateBoxSelection.}MouseMove({(mouseLPt: LPoint)});

VAR  box: TBox;

   PROCEDURE FrameTheBox;
   BEGIN
      box.DrawFrame
   END;

BEGIN
  box := SELF.box;      {get the box under creation}
  {...}
  SELF.panel.OnAllPadsDo(FrameTheBox);    {undraw the existing frame in all panes}
  {compute the new frame}
  SELF.panel.OnAllPadsDo(FrameTheBox);    {draw the new frame in all panes}
  {...}
END;
```

Questions for thought:

1. In general, what conditions warrant the creation of a new subclass?
2. How does the view arbitrate between selections?
3. Why is {TBoxSelection.}**MousePress** no longer needed?
4. Why use **FreedAndReplacedBy** to replace one selection with another?
5. Although it wasn't done in this segment, how would you get the Generic Application to frame the box when it highlights the current selection?

Box Creation Lab

Purpose:

To implement box creation.

What you are about to do:

You will compile and run 5Boxer, then optionally modify the source. This should be done in the following steps:

1) Copy the following files onto your prefix volume.

   5UBoxer.TEXT

   5UBoxer2.TEXT

   5MBoxer.TEXT

   5PBoxer.TEXT

2) Compile, install, and run the sample application, 5Boxer. Use 45 as the tool number.

3) Scan the listings of the four files in the sample application. These are included in the appendix, "Code Samples for this Segment".

4) *[Optional]* Instead of merely framing the box, try drawing the box completely during the creation phase. This involves framing and filling the box with an appropriate color.

Things to look out for:

– *Box is not drawn until the mouse is released.*

  Remember to invalidate every time the mouse is moved.

– *Box cannot be undrawn.*

  Check to see if your Highlight method draws the box being created properly.

– *Boxes multiply like rabbits.*

  Check what you are actually invalidating.

Code Sample
for this
Segment

```
24
SLOT2CHAN1
; no assembler files
$
; no building blocks
$
; no links
$
y
y
n
BoxNum5
```

```
;    PBOXER.TEXT   for  Boxer
;Phrase file for Boxer class example
1
3
2500
$-#BOOT-TK/PABC

; Apple building block phrase files can be included here

1000
5Boxer

; Other application alerts can be included here, numbered between 1001 and 32000

0

1
$-#BOOT-TK/PABC~File/Print

2
Page Layout
Preview Actual Pages#401
Preview Page Breaks#402
Don't Preview Pages#403

100
Buzzwords
Set Aside ↑Document↑#109

0
```

```
PROGRAM M5Boxer;

USES
    {$U UObject      } UObject,

  {$IFC libraryVersion <= 20}
    {$U UFont}              UFont,
  {$ENDC}

    {$U QuickDraw    } QuickDraw,
    {$U UDraw        } UDraw,
    {$U UABC         } UABC,

    {$U U5Boxer      } U5Boxer;

CONST

    phraseVersion = 1;

BEGIN

    process := TBoxProcess.CREATE;
    process.Commence(phraseVersion);
    process.Run;
    process.Complete(TRUE);

END.
```

```
1     1 --   UNIT USBoxer;
1     2 --
1     3 --   INTERFACE
1     4 --
1     5 --   USES
1     6 --       {$U UObject}              UObject,
1     7 --
1     8 --     {$IFC libraryVersion <= 20}
1     9 --       {$U UFont}                UFont,
1    10 --     {$ENDC}
1    11 --
1    12 --       {$U QuickDraw}            QuickDraw,
1    13 --       {$U UDraw}                UDraw,
1    14 --       {$U UABC}                 UABC;
1    15 --
1    16 --   CONST
1    17 --       colorWhite  = 1;
1    18 --       colorLtGray = 2;
1    19 --       colorGray   = 3;
1    20 --       colorDkGray = 4;
1    21 --       colorBlack  = 5;
1    22 --
1    23 --       boxSelectionKind = 1;
1    24 --       createBoxSelectionKind = 2;
1    25 --
1    26 --   TYPE
1    27 --
1    28 --       TColor = colorWhite..colorBlack; {color of a box}
1    29 --
1    30 --     {New Classes for this Application}
1    31 --
1    32 --       TBox = SUBCLASS OF TObject
1    33 --
1    34 --           {Variables}
1    35 --           shapeLRect:      LRect;
1    36 --           color:           TColor;
1    37 --
1    38 --         {Creation/Destruction}
1    39 --           FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
1    40 --
1    41 --         { Highlighting support }
1    42 --           PROCEDURE TBox.PaintHandles;
1    43 --
1    44 --         { Framing while creating }
1    45 --           PROCEDURE TBox.DrawFrame;
1    46 --
1    47 --         {Display}
1    48 --           PROCEDURE TBox.Draw;
1    49 --           END;
1    50 --
1    51 --
1    52 --       TBoxView = SUBCLASS OF TView
1    53 --
1    54 --         {Variables}
1    55 --           boxList:         TList;
1    56 --
1    57 --         {Creation/Destruction}
1    58 --           FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
1    59 --                                    : TBoxView;
1    60 --
1    61 --           FUNCTION  TBoxView.BoxWith(LPt: LPoint): TBox;
1    62 --           PROCEDURE TBoxView.InvalBox(invalLRect: LRect);
1    63 --
1    64 --           PROCEDURE TBoxView.MousePress(mouseLPt: LPoint); OVERRIDE;
1    65 --
1    66 --         {Display}
1    67 --           PROCEDURE TBoxView.Draw; OVERRIDE;
1    68 --           PROCEDURE TBoxView.InitBoxList(itsHeap: THeap);
1    69 --           FUNCTION  TBoxView.NoSelection: TSelection; OVERRIDE;
1    70 --          END;
1    71 --
1    72 --
1    73 --       TBoxSelection = SUBCLASS OF TSelection
1    74 --
1    75 --         {Variables}
1    76 --           box:   TBox;
1    77 --
1    78 --         {Creation/Destruction}
1    79 --           FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsBox: TBox;
1    80 --                                          itsKind: INTEGER; itsAnchorLPt: LPoint): TBoxSelection;
1    81 --
1    82 --         {Drawing - per pad}
1    83 --           PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit); OVERRIDE;
1    84 --
1    85 --         {Selection - per pad}
1    86 --           PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1    87 --       END;
1    88 --
1    89 --
1    90 --       TCreateBoxSelection = SUBCLASS OF TSelection
1    91 --
1    92 --         {Variables}
1    93 --           box:   TBox;
1    94 --
1    95 --         {Creation/Destruction}
1    96 --           FUNCTION  TCreateBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView;
1    97 --                                                itsAnchorLPt: LPoint): TCreateBoxSelection;
1    98 --
1    99 --         {Selection - per pad}
1   100 --           PROCEDURE TCreateBoxSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1   101 --           PROCEDURE TCreateBoxSelection.MouseRelease; OVERRIDE;
1   102 --       END;
1   103 --
1   104 --
1   105 --       TBoxProcess = SUBCLASS OF TProcess
1   106 --
1   107 --         {Creation/Destruction}
1   108 --           FUNCTION  TBoxProcess.CREATE: TBoxProcess;
1   109 --           FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
1   110 --                                              : TDocManager; OVERRIDE;
```

```
1  111 --           END;
1  112 --
1  113 --
1  114 --       TBoxDocManager = SUBCLASS OF TDocManager
1  115 --
1  116 --         {Creation/Destruction}
1  117 --           FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
1  118 --                                                           : TBoxDocManager;
1  119 --           FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID): TWindow; OVERRIDE;
1  120 --           END;
1  121 --
1  122 --
1  123 --
1  124 --       TBoxWindow = SUBCLASS OF TWindow
1  125 --
1  126 --         {Creation/Destruction}
1  127 --           FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
1  128 --
1  129 --         {Document Creation}
1  130 --           PROCEDURE TBoxWindow.BlankStationery; OVERRIDE;
1  131 --           END;
1  132 --
1  133 --
1  134 --   IMPLEMENTATION
1  135 --
1  136 --     {$I U5Boxer2.text}
2    1 --   {U5BOXER2}
2    2 --
2    3 --   METHODS OF TBox;
2    4 --
2    5 -- A     FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
2    6 0- A     BEGIN
2    7 --           {$IFC fTrace}BP(11);{$ENDC}
2    8 --           SELF := NewObject(itsHeap, THISCLASS);
2    9 --           WITH SELF DO
2   10 1-               BEGIN
2   11 --               shapeLRect := zeroLRect;
2   12 --               color := colorGray;
2   13 -1               END;
2   14 --           {$IFC fTrace}EP;{$ENDC}
2   15 -0 A     END;
2   16 --
2   17 --     { Draw this box }
2   18 -- A     PROCEDURE TBox.Draw;
2   19 --       VAR lPat:    LPattern;
2   20 0- A     BEGIN
2   21 --           {$IFC fTrace}BP(10);{$ENDC}
2   22 --           PenNormal;
2   23 --
2   24 --           IF LRectIsVisible(SELF.shapeLRect) THEN {this box needs to be drawn}
2   25 1-               BEGIN
2   26 --             {Get a Quickdraw pattern to represent the box's color}
2   27 2-               CASE SELF.color OF
2   28 --                   colorWhite:      lPat := lPatWhite;
2   29 --                   colorLtGray:     lPat := lPatLtGray;
2   30 --                   colorGray:       lPat := lPatGray;
2   31 --                   colorDkGray:     lPat := lPatDkGray;
2   32 --                   colorBlack:      lPat := lPatBlack;
2   33 --                   OTHERWISE        lPat := lPatWhite; {this case should not happen}
2   34 -2                   END;
2   35 --
2   36 --              {Fill the box with the pattern, and draw a frame around it}
2   37 --               FillLRect(SELF.shapeLRect, lPat);
2   38 --               FrameLRect(SELF.shapeLRect);
2   39 -1               END;
2   40 --           {$IFC fTrace}EP;{$ENDC}
2   41 -0 A     END;
2   42 --
2   43 --     { Frame a particular box}
2   44 -- A     PROCEDURE TBox.DrawFrame;
2   45 0- A     BEGIN
2   46 --           {$IFC fTrace}BP(10);{$ENDC}
2   47 --           PenNormal;
2   48 --                     {frame with reversible ink}
2   49 --           PenMode(PatXOr);
2   50 --           FrameLRect(SELF.shapeLRect);
2   51 --           {$IFC fTrace}EP;{$ENDC}
2   52 -0 A     END;
2   53 --
2   54 --       {This procedure paints the handle rectangles for highlighting; user of this method must
2   55 --        set up the pen pattern and mode before calling}
2   56 -- A     PROCEDURE TBox.PaintHandles;
2   57 --       VAR hLRect,
2   58 --           shapeLRect:  LRect;
2   59 --           dh, dv:      LONGINT;
2   60 --
2   61 -- B         PROCEDURE MoveHandleAndPaint(hOffset, vOffset: LONGINT);
2   62 0- B         BEGIN
2   63 --               OffsetLRect(hLRect, hOffset, vOffset);
2   64 --               PaintLRect(hLRect);
2   65 -0 B         END;
2   66 --
2   67 0- A     BEGIN
2   68 --           {$IFC fTrace}BP(10);{$ENDC}
2   69 --           SetLRect(hLRect, -3, -2, 3, 2);
2   70 --           shapeLRect := SELF.shapeLRect;
2   71 --           WITH shapeLRect DO
2   72 1-               BEGIN
2   73 --               dh := right - left;
2   74 --               dv := bottom - top;
2   75 --               MoveHandleAndPaint(left, top);     {draw top left handle}
2   76 -1               END;
2   77 --           MoveHandleAndPaint(dh, 0);           {then top right}
2   78 --           MoveHandleAndPaint(0, dv);           {then bottom right}
2   79 --           MoveHandleAndPaint(-dh, 0);          {finally bottom left}
2   80 --           {$IFC fTrace}EP;{$ENDC}
2   81 -0 A     END;
2   82 --
2   83 --   END;
2   84 --
```

```
2   85 --
2   86 --    METHODS OF TBoxView;
2   87 --
2   88 -- A      FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
2   89 --                                    : TBoxView;
2   90 0- A      BEGIN
2   91 --            {$IFC fTrace}BP(11);{$ENDC}
2   92 --            IF object = NIL THEN
2   93 --                object := NewObject(itsHeap, THISCLASS);
2   94 --             SELF := TBoxView(itsPanel.NewView(object, itsExtent, TPrintManager.CREATE(NIL, itsHeap),
2   95 --                                                 stdMargins, TRUE));
2   96 --            {$IFC fTrace}EP;{$ENDC}
2   97 -0 A      END;
2   98 --
2   99 --
2  100 --      {This returns the box containing a certain point or NIL if point is not within a box}
2  101 -- A      FUNCTION  TBoxView.BoxWith(LPt: LPoint): TBox;
2  102 --        VAR box:    TBox;
2  103 --            s:      TListScanner;
2  104 0- A      BEGIN
2  105 --            {$IFC fTrace}BP(11);{$ENDC}
2  106 --            boxWith := NIL;
2  107 --            s := SELF.boxList.Scanner;
2  108 --            WHILE s.Scan(box) DO
2  109 --                IF LPtInLRect(LPt, box.shapeLRect) THEN
2  110 --                    boxWith := box;
2  111 --            {$IFC fTrace}EP;{$ENDC}
2  112 -0 A      END;
2  113 --
2  114 --    {This draws the list of boxes}
2  115 -- A      PROCEDURE TBoxView.Draw;
2  116 --        VAR box:    TBox;
2  117 --            s:      TListScanner;
2  118 0- A      BEGIN
2  119 --            {$IFC fTrace}BP(10);{$ENDC}
2  120 --            s := SELF.boxList.Scanner;
2  121 --            WHILE s.Scan(box) DO
2  122 --                box.Draw;
2  123 --            {$IFC fTrace}EP;{$ENDC}
2  124 -0 A      END;
2  125 --
2  126 -- A      PROCEDURE  TBoxView.InvalBox(invalLRect: LRect);
2  127 0- A      BEGIN
2  128 --            {$IFC fTrace}BP(10);{$ENDC}
2  129 --            InsetLRect(invalLRect, -3, -2);
2  130 --            SELF.panel.InvalLRect(invalLRect);
2  131 --            {$IFC fTrace}EP;{$ENDC}
2  132 -0 A      END;
2  133 --
2  134 --
2  135 --         {This determines which type of selection to create}
2  136 -- A       PROCEDURE TBoxView.MousePress(mouseLPt: LPoint);
2  137 --         VAR aSelection: TSelection;
2  138 --             panel:      TPanel;
2  139 --             box:        TBox;
2  140 --
2  141 0- A       BEGIN
2  142 --            {$IFC fTrace}BP(11);{$ENDC}
2  143 --            panel := SELF.panel;
2  144 --            panel.Highlight(panel.selection, hOntoOff);            {Turn off the old highlighting}
2  145 --            box := SELF.BoxWith(mouseLPt);          {Find the box the user clicked on}
2  146 --
2  147 --            IF box = NIL THEN
2  148 --                {Create an instance of TCreateBoxSelection}
2  149 --                aSelection := panel.selection.FreedAndReplacedBy(
2  150 --                                    TCreateBoxSelection.CREATE(NIL, SELF.heap, SELF, mouseLPt))
2  151 --            ELSE
2  152 --                {Create an instance of TBoxSelection}
2  153 --                aSelection := panel.selection.FreedAndReplacedBy(
2  154 --                             TBoxSelection.CREATE(NIL, SELF.heap, SELF, box, boxSelectionKind, mouseLPt));
2  155 --
2  156 --            panel.Highlight(panel.selection, hOffToOn);     {Turn on the highlighting for the newly selected box}
2  157 --
2  158 --            self.panel.selection.MarkChanged;     {Allow the document to be saved so that any changes made}
2  159 --                                                  {can become permanent}
2  160 --            {$IFC fTrace}EP;{$ENDC}
2  161 -0 A       END;
2  162 --
2  163 -- A      PROCEDURE TBoxView.InitBoxList (itsHeap: THeap);
2  164 --        VAR  boxList: TList;
2  165 0- A      BEGIN
2  166 --           {$IFC fTrace}BP(11);{$ENDC}
2  167 --            boxList := TList.CREATE(NIL, itsHeap, 0);
2  168 --            SELF.boxList := boxList;
2  169 --            {$IFC fTrace}EP;{$ENDC}
2  170 -0 A      END;
2  171 --
2  172 --
2  173 -- A      FUNCTION TBoxView.NoSelection: TSelection;
2  174 0- A      BEGIN
2  175 --           {$IFC fTrace}BP(11);{$ENDC}
2  176 --           NoSelection := TBoxSelection.CREATE(NIL, SELF.Heap, SELF, NIL, nothingKind, zeroLPt);
2  177 --            {$IFC fTrace}EP;{$ENDC}
2  178 -0 A      END;
2  179 --
2  180 --    END;
2  181 --
2  182 --
2  183 --    METHODS OF TBoxSelection;
2  184 --
2  185 -- A      FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsBox: TBox;
2  186 --                                   itsKind: INTEGER; itsAnchorLPt: LPoint): TBoxSelection;
2  187 0- A      BEGIN
2  188 --            {$IFC fTrace}BP(11);{$ENDC}
2  189 --            IF object = NIL THEN
2  190 --                object := NewObject(itsHeap, THISCLASS);
2  191 --            SELF := TBoxSelection(TSelection.CREATE(object, itsHeap, itsView, itsKind, itsAnchorLPt));
2  192 --
2  193 --            SELF.box := itsBox;
2  194 --            {$IFC fTrace}EP;{$ENDC}
```

```
2  195 -0 A      END;
2  196 --
2  197 --       {This draws the handles on the selected box}
2  198 -- A      PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit);
2  199 0- A      BEGIN
2  200 --           {$IFC fTrace}BP(11);{$ENDC}
2  201 --           IF SELF.kind <> nothingKind THEN
2  202 1-               BEGIN
2  203 --               thePad.SetPenToHighlight(highTransit);   {set the drawing mode according to desired highlight in
2  204 --               SELF.box.PaintHandles;                   {draw the handles on the box}
2  205 -1               END;
2  206 --           {$IFC fTrace}EP;{$ENDC}
2  207 -0 A      END;
2  208 --
2  209 --
2  210 --       {This is called when the user moves the mouse after pressing the button}
2  211 -- A      PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint);
2  212 --       VAR diffLPt:     LPoint;
2  213 --           boxView:     TBoxView;
2  214 --           shapeLRect:  LRect;
2  215 0- A      BEGIN
2  216 --           {$IFC fTrace}BP(11);{$ENDC}
2  217 --           boxView := TBoxView(SELF.view);
2  218 --
2  219 --              {How far did mouse move?}
2  220 --           LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
2  221 --
2  222 --              {Move it if delta is nonzero}
2  223 --           IF NOT EqualLPt(diffLPt, zeroLPt) THEN
2  224 1-               BEGIN
2  225 --               SELF.currLPt := mouseLPt;
2  226 --
2  227 --               shapeLRect := SELF.box.shapeLRect;
2  228 --                     {Compute old and new positions of box}
2  229 --               boxView.InvalBox(shapeLRect);   {invalidate old box (causes it to be erased)}
2  230 --               OffsetLRect(shapeLRect, diffLPt.h, diffLPt.v);
2  231 --               boxView.InvalBox(shapeLRect);    {invalidate new box (causes it to be drawn)}
2  232 --
2  233 --               SELF.box.shapeLRect := shapeLRect;
2  234 -1               END;
2  235 --           {$IFC fTrace}EP;{$ENDC}
2  236 -0 A      END;
2  237 --
2  238 --   END;
2  239 --
2  240 --
2  241 --   METHODS OF TCreateBoxSelection;
2  242 --
2  243 -- A      FUNCTION  TCreateBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView;
2  244 --                                             itsAnchorLPt: LPoint): TCreateBoxSelection;
2  245 --       VAR box: TBox;
2  246 0- A      BEGIN
2  247 --           {$IFC fTrace}BP(11);{$ENDC}
2  248 --           IF object = NIL THEN
2  249 --              object := NewObject(itsHeap, THISCLASS);
2  250 --           SELF := TCreateBoxSelection(TSelection.CREATE(object, itsHeap, itsView, createBoxSelectionKind,
2  251 --                                                   itsAnchorLPt));
2  252 --           box := TBox.CREATE(NIL, SELF.heap);
2  253 --           SELF.box := box;
2  254 --           {$IFC fTrace}EP;{$ENDC}
2  255 -0 A      END;
2  256 --
2  257 --       {This is called when the user moves the mouse after pressing the button}
2  258 -- A      PROCEDURE TCreateBoxSelection.MouseMove(mouseLPt: LPoint);
2  259 --       VAR maxBoxLRect:     LRect;
2  260 --           diffLPt:         LPoint;
2  261 --           boxView:         TBoxView;
2  262 --           box:             TBox;
2  263 --
2  264 -- B          PROCEDURE DrawTheFrame;
2  265 0- B          BEGIN
2  266 --               box.DrawFrame;
2  267 -0 B          END;
2  268 --
2  269 0- A      BEGIN
2  270 --           {$IFC fTrace}BP(11);{$ENDC}
2  271 --           boxView := TBoxView(SELF.view);
2  272 --           box := SELF.box;
2  273 --
2  274 --           { In Boxer it is possible to draw a box greater than allowed by a 16 bit rectangle.  These three
2  275 --             lines force the rectangle to within 16 bits. }
2  276 --     {$H-} WITH SELF.anchorLPt DO
2  277 --               SetLRect(maxBoxLRect, h+10-MAXINT, v+10-MAXINT, h+MAXINT-10, v+MAXINT-10);
2  278 --     {$H+} LRectHaveLPt(maxBoxLRect, mouseLPt);
2  279 --
2  280 --           LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
2  281 --           IF NOT EqualLPt(diffLPt, zeroLPt) THEN
2  282 1-               BEGIN
2  283 --               SELF.currLPt := mouseLPt;
2  284 --
2  285 --               boxView.panel.OnAllPadsDo(DrawTheFrame);
2  286 --               WITH box DO
2  287 2-                   BEGIN
2  288 --                   shapeLRect.topLeft := SELF.anchorLPt;
2  289 --                   shapeLRect.botRight := mouseLPt;
2  290 -2                   END;
2  291 --
2  292 --       {$H-}   RectifyLRect(box.shapeLRect);    {$H+}
2  293 --               boxView.panel.OnAllPadsDo(DrawTheFrame);
2  294 -1               END;
2  295 --           {$IFC fTrace}EP;{$ENDC}
2  296 -0 A      END;
2  297 --
2  298 --
2  299 -- A      PROCEDURE TCreateBoxSelection.MouseRelease;
2  300 --       VAR thisBox:         TBox;
2  301 --           boxView:         TBoxView;
2  302 --           drawnLRect:      LRect;
2  303 --           aSelection:      TSelection;
2  304 --           panel:           TPanel;
```

```
2  305 --
2  306 -- B         PROCEDURE DrawTheFrame;
2  307 0- B         BEGIN
2  308 --               thisBox.DrawFrame;
2  309 -0 B         END;
2  310 --
2  311 0- A     BEGIN
2  312 --           {$IFC fTrace}BP(11);{$ENDC}
2  313 --
2  314 --           boxView := TBoxView(SELF.view);
2  315 --           panel := boxView.panel;
2  316 --           thisBox := SELF.box;
2  317 --           panel.OnAllPadsDo(DrawTheFrame);
2  318 --           drawnLRect := thisBox.shapeLRect;
2  319 --
2  320 --           { Independant of whether we threw the box away or not we must create an instance of TBoxSelection
2  321 --             to replace the now useless instance of TCreateBoxSelection using the kind set above. }
2  322 --           aSelection := SELF.FreedAndReplaceby(
2  323 --                           TBoxSelection.CREATE(NIL, SELF.heap, boxView, thisBox, boxSelectionKind,
2  324 --                                                drawnLRect.topleft));
2  325 --
2  326 --           boxView.InvalBox(drawnLRect);
2  327 --
2  328 --           {If the box is not big enough then throw it away, otherwise put it in the list}
2  329 --           IF (drawnLRect.right - drawnLRect.left <=4) OR (drawnLRect.bottom - drawnLRect.top <=4) THEN
2  330 1-               BEGIN
2  331 --               aSelection.kind := nothingKind;
2  332 --               thisBox.Free;
2  333 -1               END
2  334 --           ELSE
2  335 --               boxView.boxList.InsLast(thisBox);
2  336 --           {$IFC fTrace}EP;{$ENDC}
2  337 -0 A     END;
2  338 --
2  339 --   END;
2  340 --
2  341 --
2  342 --   METHODS OF TBoxProcess;
2  343 --
2  344 -- A     FUNCTION  TBoxProcess.CREATE: TBoxProcess;
2  345 0- A     BEGIN
2  346 --           {$IFC fTrace}BP(11);{$ENDC}
2  347 --           SELF := TBoxProcess(TProcess.CREATE(NewObject(mainHeap, THISCLASS), mainHeap));
2  348 --           {$IFC fTrace}EP;{$ENDC}
2  349 -0 A     END;
2  350 --
2  351 --
2  352 -- A     FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN): TDocManager;
2  353 0- A     BEGIN
2  354 --           {$IFC fTrace}BP(11);{$ENDC}
2  355 --           NewDocManager := TBoxDocManager.CREATE(NIL, mainHeap, volumePrefix);
2  356 --           {$IFC fTrace}EP;{$ENDC}
2  357 -0 A     END;
2  358 --
2  359 --   END;
2  360 --
2  361 --
2  362 --
2  363 --   METHODS OF TBoxDocManager;
2  364 --
2  365 -- A     FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
2  366 --                                       : TBoxDocManager;
2  367 0- A     BEGIN
2  368 --           {$IFC fTrace}BP(11);{$ENDC}
2  369 --           IF object = NIL THEN
2  370 --               object := NewObject(itsHeap, THISCLASS);
2  371 --           SELF := TBoxDocManager(TDocManager.CREATE(object, itsHeap, itsPathPrefix));
2  372 --           {$IFC fTrace}EP;{$ENDC}
2  373 -0 A     END;
2  374 --
2  375 --
2  376 -- A     FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID):TWindow;
2  377 0- A     BEGIN
2  378 --           {$IFC fTrace}BP(11);{$ENDC}
2  379 --           NewWindow := TBoxWindow.CREATE(NIL, heap, wmgrID);
2  380 --           {$IFC fTrace}EP;{$ENDC}
2  381 -0 A     END;
2  382 --
2  383 --   END;
2  384 --
2  385 --
2  386 --
2  387 --   METHODS OF TBoxWindow;
2  388 --
2  389 -- A     FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
2  390 0- A     BEGIN
2  391 --           {$IFC fTrace}BP(10);{$ENDC}
2  392 --           IF object = NIL THEN
2  393 --               object := NewObject(itsHeap, THISCLASS);
2  394 --           SELF := TBoxWindow(TWindow.CREATE(object, itsHeap, itsWmgrID, TRUE));
2  395 --           {$IFC fTrace}EP;{$ENDC}
2  396 -0 A     END;
2  397 --
2  398 --
2  399 -- A     PROCEDURE TBoxWindow.BlankStationery;
2  400 --       VAR viewLRect:      LRect;
2  401 --           panel:          TPanel;
2  402 --           boxView:        TBoxView;
2  403 --           aSelection:     TSelection;
2  404 0- A     BEGIN
2  405 --           {$IFC fTrace}BP(10);{$ENDC}
2  406 --            panel := TPanel.CREATE(NIL, SELF.Heap, SELF, 0, 0, [aScroll, aSplit], [aScroll, aSplit]);
2  407 --
2  408 --           SetLRect(viewLRect, 0, 0, 5000, 3000);
2  409 --           boxView := TBoxView.CREATE(NIL, SELF.Heap, panel, viewLRect);
2  410 --           boxView.InitBoxList(SELF.Heap);
2  411 --
2  412 --           {$IFC fTrace}EP;{$ENDC}
2  413 -0 A     END;
2  414 --
```

```
2  415 --    END;
2  416 --
1  137 --
1  138 --    END.
```

```
 1. u5boxer.TEXT
 2. U5Boxer2.text

-B-
  BlankStationery    130•( 1)  399•( 2)
  box                 76 ( 1)   93 ( 1)  102•( 2)  108 ( 2)  109 ( 2)  110 ( 2)  116•( 2)  121 ( 2)  122 ( 2)  139•( 2)
                     145=( 2)  147 ( 2)  154 ( 2)  193=( 2)  204 ( 2)  227 ( 2)  233 ( 2)  245•( 2)  252=( 2)  253=( 2)
                     253 ( 2)  262•( 2)  266 ( 2)  272=( 2)  272 ( 2)  286 ( 2)  292 ( 2)  316 ( 2)
  boxList             55 ( 1)  107 ( 2)  120 ( 2)  164•( 2)  167=( 2)  168=( 2)  168 ( 2)  335 ( 2)
  boxSelectionKind    23•( 1)  154 ( 2)  323 ( 2)
  BoxWith             61•( 1)  101•( 2)  145 ( 2)

-C-
  color               36•( 1)   12=( 2)   27 ( 2)
  colorBlack          21•( 1)   28 ( 1)   32 ( 2)
  colorDkGray         20•( 1)   31 ( 2)
  colorGray           19•( 1)   12 ( 2)   30 ( 2)
  colorLtGray         18•( 1)   29 ( 2)
  colorWhite          17•( 1)   28 ( 1)   28 ( 2)
  CREATE              39•( 1)   58•( 1)   79•( 1)   96•( 1)  108 ( 1)  117 ( 1)  127 ( 1)    5•( 2)   88•( 2)   94 ( 2)
                     150 ( 2)  154 ( 2)  167 ( 2)  176 ( 2)  185•( 2)  191 ( 2)  243•( 2)  250 ( 2)  252 ( 2)  323 ( 2)
                     344•( 2)  347 ( 2)  355 ( 2)  365•( 2)  371 ( 2)  379 ( 2)  389•( 2)  394 ( 2)  406 ( 2)  409 ( 2)
  createBoxSelecti    24•( 1)  250 ( 2)

-D-
  Draw                48•( 1)   67•( 1)   18•( 2)  115•( 2)  122 ( 2)
  DrawFrame           45•( 1)   44•( 2)  266 ( 2)  308 ( 2)

-H-
  Highlight           83•( 1)  144 ( 2)  156 ( 2)  198•( 2)

-I-
  InitBoxList         68•( 1)  163•( 2)  410 ( 2)
  InvalBox            62•( 1)  126•( 2)  229 ( 2)  231 ( 2)  326 ( 2)

-L-
  LRect               35 ( 1)   58 ( 2)  214 ( 2)  259 ( 2)  302 ( 2)  400 ( 2)

-M-
  MouseMove           86•( 1)  100•( 1)  211•( 2)  258•( 2)
  MousePress          64•( 1)  136•( 2)
  MouseRelease       101•( 1)  299•( 2)

-N-
  NewDocManager      109•( 1)  352•( 2)  355=( 2)
  NewWindow          119•( 1)  376•( 2)  379=( 2)
  NoSelection         69•( 1)  173•( 2)  176=( 2)

-P-
  PaintHandles        42•( 1)   56•( 2)  204 ( 2)

-Q-
  QuickDraw           12•( 1)

-S-
  shapeLRect          35 ( 1)   11=( 2)   24 ( 2)   37 ( 2)   38 ( 2)   50 ( 2)   58•( 2)   70=( 2)   70 ( 2)   71 ( 2)
                     109 ( 2)  214•( 2)  227=( 2)  227 ( 2)  229 ( 2)  230 ( 2)  231 ( 2)  233=( 2)  233 ( 2)  288 ( 2)
                     289 ( 2)  292 ( 2)  318 ( 2)

-T-
  TBox                32•( 1)   39 ( 1)   61 ( 1)   76 ( 1)   93 ( 1)    3•( 2)    5 ( 2)  101 ( 2)  102 ( 2)  116 ( 2)
                     139 ( 2)  245 ( 2)  252 ( 2)  262 ( 2)  300 ( 2)
  TBoxDocManager     114•( 1)  118 ( 1)  355 ( 2)  363•( 2)  366 ( 2)  371 ( 2)
  TBoxProcess        105•( 1)  108 ( 1)  342•( 2)  344 ( 2)  347 ( 2)
  TBoxSelection       73•( 1)   80 ( 1)  154 ( 2)  176 ( 2)  183•( 2)  186 ( 2)  191 ( 2)  323 ( 2)
  TBoxView            52•( 1)   59 ( 1)   86•( 2)   89 ( 2)   94 ( 2)  213 ( 2)  217 ( 2)  261 ( 2)  271 ( 2)  301 ( 2)
                     314 ( 2)  402 ( 2)  409 ( 2)
  TBoxWindow         124•( 1)  127 ( 1)  379 ( 2)  387•( 2)  389 ( 2)  394 ( 2)
  TColor              28•( 1)   36 ( 1)
  TCreateBoxSelect    90•( 1)   97 ( 1)  150 ( 2)  241•( 2)  244 ( 2)  250 ( 2)
  TDocManager        110 ( 1)  114 ( 1)  352 ( 2)  371 ( 2)
  TList               55 ( 1)  164 ( 2)  167 ( 2)
  TObject             32 ( 1)
  TProcess           105 ( 1)  347 ( 2)
  TSelection          69 ( 1)   73 ( 1)   90 ( 1)  137 ( 2)  173 ( 2)  191 ( 2)  250 ( 2)  303 ( 2)  403 ( 2)
  TView               52 ( 1)
  TWindow            119 ( 1)  124 ( 1)  376 ( 2)  394 ( 2)

-U-
  U5Boxer              1•( 1)
  UABC                14•( 1)
  UDraw               13•( 1)
  UFont                9•( 1)
  UObject              6•( 1)
••• End Xref: 48 id's   253 references      [408600 bytes/4951 id's/41173 refs]
```

[Segment 9]

Recoloring, Duplicating, and Clear All
Commands with Undo

Purpose of this segment:

1) To introduce commands.
2) To present the four phases of commands: do, undo, redo, commit.
3) To discuss command generation and command processing.
4) To be able to recolor and duplicate a selected box; and undo these operations. To be able to clear all boxes in a simple, but undoable fashion..

How to use this segment:

This is the ninth segment of the self—paced introduction to the ToolKit. This segment follows the "Creating Boxes" segment; and precedes the segment, "Filters".

The next three segments are devoted to responding to events and creating fully undoable commands. This segment considers only the simplest implementation of undo. This stage of Boxer, 6Boxer, is able to undo selected menu events. The remaining two segments provide the tools to make nearly any operation undoable.

INTRODUCTION TO COMMANDS

With few exceptions, all applications allow users to make changes to a document. The ToolKit supplies a special object to manage such changes. This object is known as a *command*

You are already familiar with such commands as "Cut", "Paste", and "Duplicate". You select some data to be changed, then indicate the operation to perform upon it. For example, you *cut* selected portions of a LisaDraw document to the clipboard; you *duplicate* a selected document on the Desktop.

<u>what do commands do?</u>

Lisa commands are used in many different ways. Lisa commands may:

— Change a selected portion of a document. (eg. *cut, paste, type style*)

- Change the view in a panel or window. (eg. *chronological, preview pages*)
- Use the entire document as data, without changing it. (eg. *save and put away, print, search*)
- Bring up a dialog box, especially when more information is required to make a change. (eg. *format for printer*)
- Display or hide a control. (eg. *hide margin ruler, show document size*)
- Set a control for subsequent changes. (eg. *scale of ruler, set tab*)
- Make a new selection. (eg. *select all of document*)

Normally, a command needs a selection to operate upon. But a command, such as "Save and Put Away" or "Print", may, instead, operate upon the entire window or document, ignoring the current selection.

Both types of commands are legitimate. Both have similar beginnings (and ends).

birth of a command

Commands are born from events. As mouse events give rise to selections, so commands arise from menu events. Certain keyboard events (eg. apple—key combinations) also appear to create commands, but these are actually converted to menu events *by the Menu Manager* before generating any commands.

Application users generate the menu events that precede commands. When the user clicks on the menu bar and releases on a menu item, a menu event is born. This menu event goes immediately to the current selection, which normally generates a command. *If a selection creates the command, the command is given the currently selected object to operate upon.*

doing a command

A command changes a document by operating upon the selected object. Typically, the change is made to the view. The window updates the display of the view to reflect the change before processing the next event.

A command may also operate upon the entire document or window. This is particularly applicable when there is no selected object.

undoing a command

To undo an operation performed by a command, the simplest method is to save in the command object the document data that will be changed. You simply undo the command by restoring the document's state from the data saved in the command.

The mechanism is similar to making change for a dollar. Suppose that you, playing the role of the command, have a dollar. Your friend, playing the role of the document, displays some change. In response to a friendly request, you swap some of his change for your dollar. Now your friend displays a dollar in place of the change. But, for reasons unknown, your friend desires to undo the transaction. Since you

wisely retained his change, you happily restore your friend to his previous currency state.

That simple technique of implementing undo is sufficient for this stage of Boxer. But, as the amount of information needed to restore a document to its pre-changed state grows, more sophisticated undo strategies will be needed. *These strategies are discussed in the subsequent segment, "Filters".*

Whenever possible, all commands should be undoable. This is an extremely useful application feature. Especially important is the ability to restore the selection to what it was before the command. That way the user can proceed from where he or she left off with minimum delay. To this end, the ToolKit provides a simple, but powerful structure for doing and undoing commands.

MENU EVENTS

Menu events are generated by the Menu Manager in response to mouse presses in the menu bar.

The menu closest to the mouse press gets *pulled-down.* The pulled-down menu displays a list of menu items. Releasing the mouse over one of these items generates a menu event with an associated item number. This item number is used to identify the event to the application.

If the menu event's number is legitimate and an object is selected, a command may be generated. But if the current selection is a null selection (kind equals **nothingKind**), a command may not be able to be performed. The menu event may generate a warning instead. *Warnings are displayed in alert boxes.*

Understandably, from the user's point of view, a menu item should be disabled if it cannot be performed. The ToolKit supports this reasoning.

Before a menu is displayed the selection can tell the Generic Application which menu items it will enable and which it will not. *This is the recommended procedure in all applications.* If not enabled, the menu item is grayed. *Grayed menu items cannot generate events.*

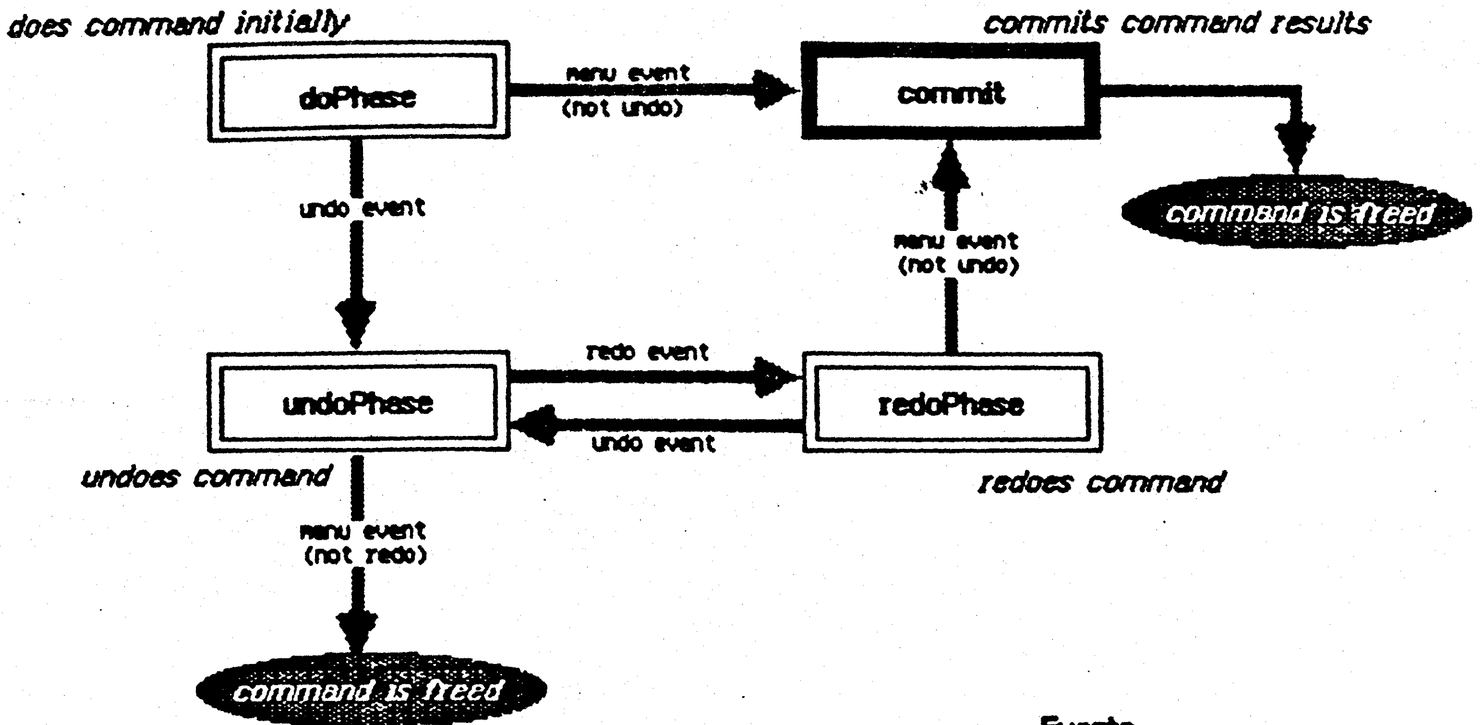
COMMAND CONCEPTS

The following concepts are essential to successful implementation of commands: *command phase, phrase file, command number, revelation, image, and filtering.*

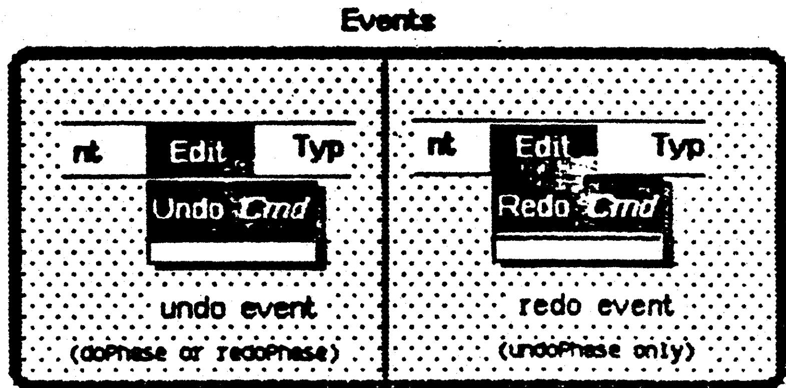
command phase

There are four phases of commands. These are: *doPhase, undoPhase, redoPhase,* and *commit.* The diagram: *Command Phases* illustrates their interrelationships.

Command Phases



Events



Note:
the menu item switches from "undo" to "redo" depending upon the command state

The ToolKit defines the following command phase constants:

```
TCmdPhase = (doPhase, undoPhase, redoPhase);
```

Commands are performed during those three phases only. The *commit* phase terminates the command.

The *doPhase* is the initial command phase. It is entered immediately after the command is created. The command is initially performed during this phase.

The *undoPhase* undoes the command. It is entered when the user initiates an undo menu event during the *doPhase* or the *redoPhase*. When the command is performed during the *undoPhase*, the command's previous changes are undone.

The *redoPhase* undoes an undo. It is entered when the user initiates a redo menu event during the *undoPhase*. When the command is performed during this phase, the command's initial changes are reinstated.

The ToolKit allows only single—level undo. Only the most recently performed command may be undone or redone. When a command is committed it is completed, and can no longer be undone or redone. Redo and undo events switch the command phase between *undoPhase* and *redoPhase*.

Most events other than redo or undo terminate the current command. If the current command phase is *doPhase* or *redoPhase*, the command enters the *commit* phase. The *commit* fixes the command's changes into the document, then deallocates (or frees) the command. If the current command phase is *undoPhase*, the command is simply deallocated.

Some events do not terminate a command, e.g., scrolling, splitting, resizing, and selecting. They do not affect the command phase.

phrase file

The phrase file contains an application's menus, alerts, text, and name on the Desktop. Each application has a phrase file associated with it. Phrase files are covered in a separate document. *The phrase file is also known as an alert and menu file.*

command number

The command number is typically the number of the menu item that initiates the command. Menu item numbers are defined both in the application phrase file and in the application code. This enables the Menu Manager to communicate menu events to the application. A menu item's number uniquely distinguishes it from other items in the phrase file.

revelation

Revelation is the amount to reveal the current selection before performing the current phase of the command.

It is often desirable to scroll the whole selection into view to observe the effects of a command. Sometimes (clearing the screen, for example) you don't care. And other times only part of the selection (recoloring a box, for example) is needed to convey the change.

The ToolKit supplies the following revelation constants:

```
TRevelation = (revealNone, revealSome, revealAll);
```

Revelation tells the Generic Application how much to scroll to reveal the current selection's bound LRect. Scrolling is done in a pane selected by the Generic Application.

RevealNone performs no scrolling. *RevealSome* scrolls to reveal at least a 30x20 pixel portion of the bound LRect. *RevealAll* scrolls to reveal the maximum possible portion of the selection's bound LRect.

image

An image is an area within a view. The command's image defines the portion of the view that the command can affect. Typically, the image will be the same as the view. Images have two fields, one of which is the view in which it lies. Images are of the class **TImage**. *TView is a subclass of TImage.*

```
TImage = SUBCLASS OF TObject
  {fields}
  extentLRect: LRect;   {the bounds of the image}
  view:        TView;   {the view containing the image (or SELF)}
```

filtering

Filtering is a way to make a change to a document without affecting the document's data. A *filtered command* changes the display of the document, but not the document itself. Filtering is like editing an overhead slide by making changes to a sheet of plastic covering the slide. The slide's display is changed without the slide itself being altered.

Only when a filtered command is committed are its effects made permanent in the document.

THE STRUCTURE OF COMMANDS

Commands are descended from the class, **TCommand**. They have the following structure:

```
TCommand = SUBCLASS OF TObject

  {fields}
  cmdNumber:      TCmdNumber;     {the command number of the associated menu item}
  image:          TImage;         {the image or view affecting filtering}
  undoable:       BOOLEAN;        {TRUE iff the command is undoable}
  doing:          BOOLEAN;        {TRUE iff command is in doPhase or redoPhase}
  revelation:     TRevelation;    {how much of the selection to reveal before performing
                                   the command}
  unHiliteBefore: ARRAY[TCmdPhase] OF BOOLEAN  {if TRUE, ToolKit unhighlights all
```

```
                                        selections before performing command}
hiliteAfter:     ARRAY[TCmdPhase] OF BOOLEAN  {if TRUE, ToolKit highlights all
                                        selections after performing command}
```

Some commands, such as "Show Page Ruler", can be made not undoable. Typically, in such cases, a complementing menu item, such as "Hide Ruler", is provided in lieu of undoability.

Some of the relevant methods of commands are listed below:

```
{creation}
FUNCTION  {TCommand.} CREATE(object: TObject; heap: THeap; itsCmdNumber: TCmdNumber;
                        itsImage: TImage; isUndoable: BOOLEAN; itsRevelation: TRevelation)
                        : TCommand;
{destruction}
PROCEDURE {TCommand.} Free; OVERRIDE;   {frees temporary fields if last phase was undoPhase}

{command execution}
PROCEDURE {TCommand.} Commit; DEFAULT;   {commits the command (default is a no-op)}
PROCEDURE {TCommand.} Perform(cmdPhase: TCmdPhase); DEFAULT;  {performs the command in the
                                          given phase (default is a no-op)}
{filtering}
          {those methods are covered in the "Filtering" segment}
```

In this stage of Boxer, we only need to be concerned with **CREATE**, **Free** and **Perform**.

COMMAND GENERATION

The ToolKit's mechanism for generating and processing commands is simple and smart. The Generic Application takes care of everything that you don't. You need to only handle commands specific to your data. The Generic Application handles the rest. For example, in Boxer, you might handle just recoloring and duplicating boxes, while leaving such things as printing or saving the document to the Generic Application.

Your code gets the first opportunity to generate a command in response to a menu event. Your application's selection gets to perform the initial honors.

role of the selection

From the moment the user clicks on the menu bar, the menu event is destined for the selection. Two methods of your selection are particularly important — **CanDoCommand** and **NewCommand**. The default declarations of those methods are listed below.

```
FUNCTION {TSelection.} CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
                                  : BOOLEAN; DEFAULT;  {indicates command availability}
FUNCTION {TSelection.} NewCommand(cmdNumber: TCmdNumber): TCommand; DEFAULT;
                                  {returns the created command}
```

Before the target menu is displayed, the selection is called upon to indicate which menu items are available to the user. The flow of control below includes the method **CanDoCommand**. This method tells the Generic Application which items are enabled and which are not at the moment of the menu click.

```
window.MenuEventAt             {TWindow.}
  window.SetUpMenus            {TWindow.}
    menuBar.BuildCmdName       {TMenuBar.}
    selection.CanDoCommand     {TBoxSelection.}
```

Graying the items that the selection says are not enabled, the window and the menu bar set up the menu.

As the user moves the mouse over the menu, enabled items under the mouse are highlighted. If the mouse is released over an enabled item, the menu item number is returned. Otherwise zero is returned.

From a nonzero menu item number the selection generates the command. The method **NewCommand** creates the command and returns it to the window. This is indicated in the flow of control below.

```
    selection.CanDoCommand     {TBoxSelection.}
  menuBar.DownAt               {TMenuBar.}        returns menu item #
  window.DoCommand             {TWindow.}
    selection.NewCommand       {TBoxSelection.}   returns command
```

which class command?

The **NewCommand** method creates commands in response to menu events. There are four choices of how to process a menu event. These are:

1) Create and return an undoable command.

   You must have defined a subclass of **TCommand** (for example, **TRecolorCmd**) for the particular command. The created command is an instance of that subclass. The **undoable** parameter must be set to true. The **Perform** method of the command will be called by the Generic Application after committing the last active command.

2) Return NIL, rather than a command object, for events that are not commands.

   View—altering events such as "hide ruler" or "select all of document" do not need to be processed as commands. Returning NIL for a menu event

preserves the last active command. If undoable, it can still be undone or redone.

Please read the section "Guidelines for Converting Menu Events Into Commands" for more insight upon this.

3) Pass an unrecognized menu item to the Generic Application for processing.

4) Create and return a command that cannot be undone.

   Typically you define such a command as an instance of the ToolKit class, **TCommand**, with the **undoable** parameter set to false. Before returning you call a method of your creation to execute the command. You may need to commit the last command before the body of your command is executed. Upon return from **NewCommand** the last active command is committed, if you have not already committed it yourself.

   This is done by calling SUPERSELF.**NewCommand**.

Note: Each type of undoable command should be an instance of a unique subclass of TCommand, since each type makes distinct changes to the document. The ***Perform*** *method of each class should reflect the unique way it's instances do and undo changes to a document's data.*

standard commands

While the selection directly generates commands specific to the application, the Generic Application generates commands standard to all applications (eg. *Save and Put Away* and *Print*).

The selection calls the Generic Application to handle menu item numbers it does not recognize. As the flow of control below shows, the selection's **NewCommand** method calls SUPERSELF.**NewCommand** to invoke the Generic Application.

selection.**NewCommand**	{TBoxSelection.}	
SUPERSELF.**NewCommand**	{TSelection.}	*Generic Application*
window.**NewCommand**	{TBoxWindow.}	
SUPERSELF.**NewCommand**	{TWindow.}	*Generic Application*

The first thing that TSelection.NewCommand does is give the selection's *co–selection* a chance to generate the command. *The co–selection is a special selection created by a ToolKit building block to handle events specific to the building block. For example, a co–selection created by UText (the text building block) would handle menu events such as formatting text or changing typestyles. The co–selection is assigned to the application selection's* ***coSelection*** *field.*

If the co–selection does not recognize the menu event, the Generic Application passes the event to the application's window. The window is typically used to handle commands that affect a whole document (such as clear all boxes), rather than a single selected object.

Finally, any menu items not handled by the window are passed back to the Generic Application. It is at this point that a *Print* command, for example, is generated.

summary of command generation

1. The selection gets the menu item number. The selection handles application specific commands affecting the selected object.

   The selection calls the Generic Application to handle any other commands.

2. The Generic Application passes the menu item number to the selection's co—selection. The co—selection handles building block specific commands affecting the (co—selection's) selected object.

   The Generic Application calls the window to handle what the co—selection does not.

3. The window handles application specific commands affecting the whole document.

   The window calls the Generic Application to handle any other commands.

4. The Generic Application handles standard commands.

   If the Generic Application does not recognize the menu item number it alerts the user.

   It puts up a "No Selection" alert for null selections;

   otherwise it puts up an "Unknown Command" alert.

Note: The flow of control applying to ***NewCommand*** *applies to* ***CanDoCommand*** *as well.*

THE ROLE OF THE WINDOW

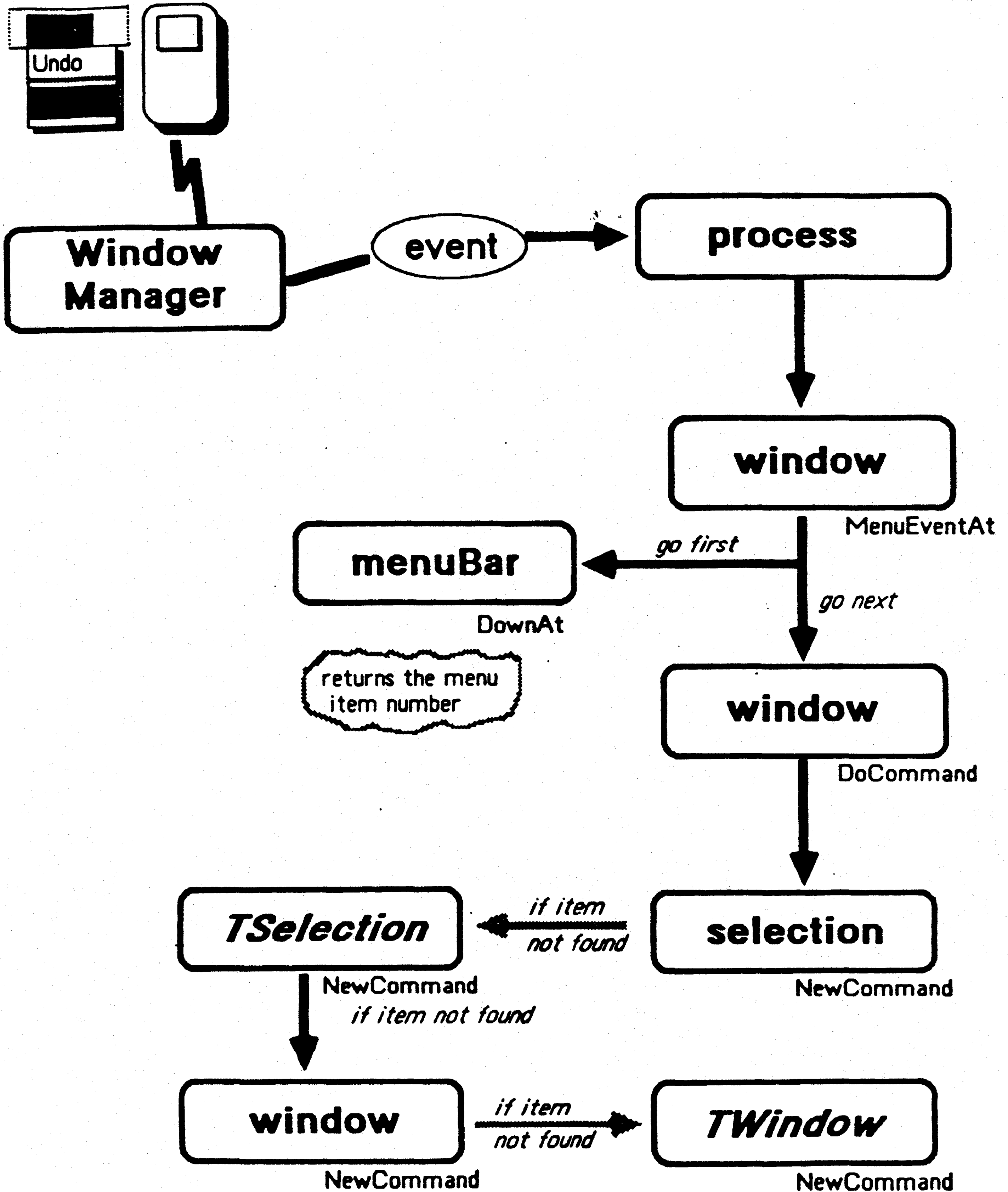
The Generic Application makes the window the owner of the generated command. Per the Lisa User Interface conventions only one command is active at a time.

The following partial interface lists the fields and methods of **TWindow** that apply specifically to commands.

```
TWindow = SUBCLASS OF TObject
   {fields applying to commands}
  lastCmd:       TCommand;   {the last active (uncommitted) command}

  {methods applying to commands}
  FUNCTION  {TWindow.}CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
                                : BOOLEAN; DEFAULT;  {calls currentWindow.CanDoStdCommand}
  FUNCTION  {TWindow.}CanDoStdCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
```

The Flow of User Events

(menu events)



```
                                : BOOLEAN; DEFAULT;  {standard command availability}
PROCEDURE {TWindow.}CommitLast; DEFAULT;  {commits and frees the last active command}
PROCEDURE {TWindow.}DoCommand(cmdNumber: TCmdNumber); DEFAULT;  {creates and performs the
                         command associated with the given menu item number}
PROCEDURE {TWindow.}MenuEventAt(mousePt: Point); DEFAULT;  {identifies the menu item number
                         and calls currentWindow.DoCommand}
FUNCTION  {TWindow.}NewCommand(cmdNumber: TCmdNumber): TCommand; DEFAULT;
                         {calls currentWindow.NewStdCommand}
FUNCTION  {TWindow.}NewStdCommand(cmdNumber: TCmdNumber): TCommand; DEFAULT;
                            {returns a created command}
PROCEDURE {TWindow.}PerformCommand(newCommand: TCommand); {commits last command, then performs
                                the new command}
PROCEDURE {TWindow.}PerformLast(cmdPhase: TCmdPhase);  {performs the current command in the
                                given command phase}
PROCEDURE {TWindow.}SaveCommand(command: TCommand); {saves the command as SELF.lastCmd}
PROCEDURE {TWindow.}SetUpMenus;  {sets up menus before allowing user to select a menu item}
PROCEDURE {TWindow.}UndoLast;  {undoes or redoes the last command}
```

Note that a field of the window, lastCmd, keeps track of the last active command.

The window is primarily responsible for handling undo and redo events, and committing commands.

As you can see from both the diagram: *The Flow of User Events (menu events)* and the following section, the window plays a major role in processing commands.

PROCESSING COMMANDS

The selection creates a command. The window processes it. The important role that the window plays during the various command phases is conveyed by the following flow of control diagrams.

the doPhase (command creation)

The *doPhase* creates a new command, and commits the last active command before performing the new command. The flow of control in 6Boxer is as follows:

```
process.ObeyTheEvent                {TProcess.}
  window.MenuEventAt                {TWindow.}  -> menu item number
    window.DoCommand                {TWindow.}
      selection.NewCommand          {TWindow.}  -> (returns) cmd
      window.PerformCommand         {TWindow.}
        window.CommitLast           {TWindow.}
          (window.lastCmd).Commit   {*}
          (window.lastCmd).Free     {*}
        window.SaveCommand          {TWindow.}  cmd -> window.lastCmd
        window.PerformLast          {TWindow.}  <phase = doPhase>
          window.HighLight          {TWindow.}  turns highlighting off
            selection.Highlight     {TBoxSelection.}
          window.RevealSelection    {TWindow.}
          selection.PerformCommand  {TSelection.} default
            command.Perform         {*}         <phase = doPhase>
          window.Update             {TWindow.}  draws the modified view
          window.Highlight          {TWindow.}  turns highlighting on
            selection.Highlight     {TBoxSelection.}

                                        {*} -  the command's class
```

Note: By default, the Generic Application turns highlighting off before and on after performing the command. This can be deactivated in the command's CREATE method.

the undoPhase

The Generic Application applies the *undoPhase* to the command created in the *doPhase*. The *undoPhase* is entered when the menu item number returned from

{TMenuBar.}**DownAt** equals the ToolKit constant, **uUndoLast**. The flow of control is listed below:

Call	Class	Comment
window.**MenuEventAt**	{TWindow.}	*returns uUndoLast*
window.**DoCommand**	{TWindow.}	
window.**UndoLast**	{TWindow.}	
window.**PerformLast**	{TWindow.}	*<phase = undoPhase>*
{...}		
command.**Perform**	{*}	*<phase = undoPhase>*
{...}		

*Note: It is up to the command's **Perform** method to define the results for the different phases. If not undoable, a command is neither performed in this phase nor the redoPhase.*

<u>the redoPhase</u>

The *redoPhase* undoes the *undoPhase*. Its flow of control is similar to that for the *undoPhase*. The same menu item number, **uUndoLast**, is used to indicate undo and redo events. The command's **doing** field differentiates the two events. The *redoPhase* is entered only if command.**doing** is false.

Call	Class	Comment
window.**MenuEventAt**	{TWindow.}	*returns uUndoLast*
window.**DoCommand**	{TWindow.}	
window.**UndoLast**	{TWindow.}	
window.**PerformLast**	{TWindow.}	*<phase = redoPhase>*
{...}		
command.**Perform**	{*}	*<phase = redoPhase>*
{...}		

RECOLORING, DUPLICATING, AND CLEAR ALL (In Boxer)

Three commands are implemented in this stage of Boxer. These are *recolor*, *duplicate*, and *clear all*.

recolor

Recoloring allows the user to change the color of the selected box. This operation is undoable. The command object needs to remember the original color of the box to be able to undo a color change.

If no box is selected, the recolor menu items are disabled (grayed).

User interface: A separate menu for colors will be displayed. The user will be able to choose any of the following colors from the menu: white, light gray, gray, dark gray, and black. The color selected will be the new color of the selected box. Undoing the command restores the box's original color.

duplicate

Duplicate allows the user to duplicate the selected box. The newly created duplicate becomes the selected box. This box will have the same shape and color as the original.

This operation is undoable. The command needs to keep track of both the new box and the original box to be able to undo and redo itself.

If no box is selected, the *duplicate* menu item is disabled.

User interface: A *duplicate* menu item will be displayed in the Edit menu. When the user chooses this menu item, a duplicate of the selected box will be created and selected. Undoing the command erases the duplicate and reselects the original box.

clear all

Clear all allows the user to clear the boxView of all boxes. For this stage of Boxer this operation is not undoable. As such, a warning needs to be supplied to the user to verify whether the command should be performed.

Since *clear all* operates upon the entire window, and does not depend upon the selection, it is always enabled.

User interface: A *clear all* menu item will be displayed in the Edit menu. When the user chooses this menu item, an alert box is immediately posted. The alert box warns the user that the command cannot be undone. The user is given the choice to cancel the command or perform it. If performed, the window and the view are cleared of all boxes.

Note: According to the Lisa User Interface Standards, *Clear All* should not be a separate command. The user should choose *Select All* and then *Clear*. However, for pedagogical reasons, we implement *Clear All* in this segment.

IMPLEMENTATION

We modify the previous stage of Boxer, 5Boxer, to implement the *recolor, duplicate,* and *clear all* commands.

With the exception of the **CanDoCommand** and **NewCommand** methods in the selection and the window, we can implement each command separately. This is, in fact, what we do.

Implementation of *Duplicate:*

To implement *duplicate* we need to:

1. Insert a *duplicate* menu item into the Edit menu in the phrase file.
2. Insert the menu item number of *duplicate* into the interface. Menu item constants generally begin with a lower case "u" (e.g. **uDuplicate**).
3. Define a new command class, **TDuplicateCmd**. This class defines two fields: **oldBox** for the old box, and **newBox** for the duplicate box.
4. Modify {TBoxSelection.}**CanDoCommand** to enable the *duplicate* menu item when a box is selected.
5. Modify {TBoxSelection.}**NewCommand** to generate an instance of **TDuplicateCmd** when the menu item number supplied is **uDuplicate**.

To define the class, **TDuplicateCmd**, we include the following methods:

{TDuplicateCmd.}**CREATE**

Creates a *duplicate* command object. Also duplicates the old box. Duplicating the box in the **CREATE** method simplifies the **Perform** code.

{TDuplicateCmd.}**Free**

Frees the command. Frees the duplicate if the last phase was *undoPhase*.

{TDuplicateCmd.}**Perform**

Performs the command in the three phases.

in the *doPhase*

Append the new box to the boxView's boxList.

Remake the selection to highlight the new box, not the original.

Invalidate the new box's LRect.

in the *undoPhase*

Delete the new box from the boxList.

Remake the selection to highlight the old box, not the new.

Invalidate the new box's LRect.

in the *redoPhase*

(same as the doPhase)

theory of operation (duplicate)

The operation of the *duplicate* command proceeds as follows:

The Generic Application is waiting in the event loop for an event. When the user depresses the mouse on the menu bar, the Generic Application sets up the menus. It requests information about your menus and menu items by calling:

selection.**CanDoCommand** {TBoxSelection.}

1. Indicates which of your selection-dependent menu items are enabled or disabled. If the command number passed to you is not one your selection handles, then call SUPERSELF.**CanDoCommand** to give control back to the Generic Application.

Since you are not using any building blocks, and thus have no co-selections, the Generic Application promptly calls:

window.**CanDoCommand** {TBoxWindow.}

2. Indicates which of your selection-independent menu items are enabled or disabled. If the command number is not one your window handles, call SUPERSELF.**CanDoCommand** to give control back to the Generic Application.

The Generic Application then processes standard menu items to complete setting up the menus. The Generic Application then passes control to the Menu Manager. The Menu Manager returns the number of the menu item the user released the mouse on. *If no menu item was chosen, zero is returned.*

The Generic Application application gives you the first opportunity to identify the newly generated menu event. It calls:

selection.**NewCommand** {TBoxSelection.}

3. Compares the menu item number supplied with those that your application will process directly. If the menu item number equals **uDuplicate**, then:

   Creates a new instance of **TDuplicateCmd**. You supply the handle of the currently selected box, which is saved as SELF.**oldBox**. {TDuplicateCmd.}CREATE duplicates the selected box and assigns the duplicate to SELF.**newbox**.

4. Returns the *duplicate* command to the Generic Application.

*[From this point on the menu event is assumed to be **uDuplicate**.]*

The Generic Application commits and frees the last command. It then installs *duplicate* in the window's **lastCmd** field. Next, it calls your code to unhighlight the current selection before performing *duplicate:*

selection.**Highlight** {TBoxSelection.}

Now the Generic Application reveals the selection as specified in the command's **revelation** field. *If revealing the selection requires scrolling, it calls view.Draw as needed.* It then calls your code to perform the *doPhase* of the command:

lastCmd.**Perform** {TDuplicateCmd.}

5. Makes the duplicate the new selected object. Invalidates the duplicates's shape LRect. Returns.

The Generic Application now updates the window and highlights the current selection. The duplicate box is thus drawn and highlighted.

Successive undo and redo menu events cause the Generic Application to call your command's **Perform** method with the respective phase.

When a menu event is received that generates a new command, the Generic Application attempts to commit the last command. It does the commit only if the last command phase was *doPhase* or *redoPhase*.

To commit a command, the Generic Application executes the command's **Commit** method. In the case of *duplicate* that method is {TCommand.}**Commit**, which is a no—op. Next, the Generic Application frees the last command. It needs to call your code to do so:

lastCmd.**Free** {TDuplicateCmd.}

6. Frees the duplicate if the last phase was *undoPhase*. Next frees itself.

The Generic Application then proceeds with the new command.

Implementation of *ReColor:*

To implement *recolor* we need to:

1. Add a Color menu to the phrase file, including the following menu items:

   white *light gray* *gray* *dark gray* *black*

2. Insert their menu item numbers into the interface.
3. Define a new command class, **TRecolorCmd**. This class defines two fields: **color** for the new color, and **box** for the selected box.
4. Modify {TBoxSelection.}**CanDoCommand** to enable the recolor menu items when a box is selected.
5. Modify {TBoxSelection.}**NewCommand** to generate an instance of **TRecolorCmd** when one of the recolor menu item numbers is supplied.

To define the class, **TRecolorCmd**, we include the following methods:

{TRecolorCmd.}**CREATE**

Creates a *recolor* command object. Sets the color field to the color chosen.

{TRecolorCmd.}**Perform**

Performs the command in the three phases.

in the *doPhase*

Swap the color of the box with the command's color field.

Invalidate the selected box's LRect.

in the *undoPhase*

(same as the doPhase)

(the effect is that the original color is restored)

in the *redoPhase*

(same as the doPhase)

Implementation of *Clear All:*

The implementation is a little different here, since *clear all* is not undoable. The main change is that no new command class is defined. Instead we create the command as an instance of **TCommand**. We add a special method to the boxWindow to perform the command. *That method is added to the boxWindow, because the command is independent of the boxSelection.*

To implement *clear all* we need to:

1. Insert a *clear all* menu item into the Edit menu in the phrase file.
2. Insert the menu item number of *clear all* into the interface. Use the menu item constant, **uClearAll**.
3. Insert an alert into the phrase file warning the user that *clear all* cannot be undone. *Before the command is performed we want to give the user a chance to cancel; since it is not going to be undoable.*

   To insert the following alert: "The Clear All command cannot be undone", you must:

   a. Put the following text into the phrase file. Along with the alert it defines a corresponding alert number and alert type.

   ```
   ; PROCEDURE {TBoxWindow.}NewCommand    {Where the alert is called from}
   ; cantUndo = 1001;                     {Value of the alert constant}
   1001 caution cancel alert
   You will not be able to undo ClearAll
   ```

   This is a caution cancel alert. It gives the user the option to cancel the named operation, or to continue. To make this alert parameterized by the command name, you could type "You will not be able to undo ^C."

   b. Put a corresponding alert constant into the interface.

   ```
   cantUndo = 1001;
   ```

4. Add a method to the window, {TBoxWindow.}**ClearAll**. This method deletes every box in the boxView's boxList; and replaces the selection with the null selection.
5. Modify {TBoxWindow.}**CanDoCommand** to always enable the *clear all* menu item.
6. Modify {TBoxWindow.}**NewCommand** to generate an instance of **TCommand** when the menu item number supplied is **uClearAll**. Next put up the alert. If the user does not cancel the alert, call SELF.**ClearAll** to clear the boxes.

   *Note: Returning an instance of **TCommand** will commit the last command automatically. The Generic Application will also call {TCommand.}**Perform**, but this method is a no-op. That is why **NewCommand** explicitly calls special code to perform commands that cannot be undone.*

To put up the alert defined above, you must make the following call:

```
process.Caution(cantUndo);  {puts up the caution alert, cantUndo}
```

Implementation Summary

The actual implementation for 6Boxer is summarized below. The code for 5Boxer is used as the base for all changes.

New Constants

```
    {menus}
uWhite  = 1006;
uLtGray = 1007;
uGray   = 1008;
uDkGray = 1009;
uBlack  = 1010;
uDuplicate = 1011;
uClearAll  = 1012;
    {phrases}
cantUndo = 1001;
```

New Classes

[TRecolorCmd]

```
FUNCTION {TRecolorCmd.}CREATE(object: TObject; itsHeap: THeap; itsCmdNumber:
TCmdNumber;
                              itsView: TBoxView; itsBox: TBox; itsColor: TColor)
                              : TRecolorCmd;

PROCEDURE {TRecolorCmd.}Perform; OVERRIDE;
```

[TDuplicateCmd]

```
FUNCTION  {TDuplicateCmd.}CREATE(object: TObject; itsHeap: THeap;
                                 itsCmdNumber: TCmdNumber; itsView: TBoxView;
                                 itsBox: TBox): TDuplicateCmd ;

PROCEDURE {TDuplicateCmd.}Free; OVERRIDE;

PROCEDURE {TDuplicateCmd.}Perform; OVERRIDE;
```

New Methods (for existing classes)

[TBoxSelection]

```
PROCEDURE {TBoxSelection.}CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
                                       : BOOLEAN; OVERRIDE;

PROCEDURE {TBoxSelection.}NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;

PROCEDURE {TBoxSelection.}MouseRelease; OVERRIDE;
```

If the mouse moved with the button down (a *mouse move* event) {TBoxSelection.}**MouseRelease** commits the last command. Otherwise it returns.

```
[TBoxWindow]
 PROCEDURE {TBoxWindow.}CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
                                    : BOOLEAN; OVERRIDE;

 PROCEDURE {TBoxWindow.}NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;

 PROCEDURE {TBoxWindow.}ClearAll;
```

Modified Method

```
[TCreateBoxSelection]
 PROCEDURE {TCreateBoxSelection.}MouseRelease; OVERRIDE;
```

{TCreateBoxSelection.}**MouseRelease** now commits the last command before adding the newly created box to the view.

Something to think about: If box creation did not commit the last command then what would the following series of events do?

1. Duplicate a box
2. Create a box
3. Undo

Hint: Does creating a box normally affect window.**lastCmd**?

GUIDELINES FOR CONVERTING MENU EVENTS TO COMMANDS (optional)

Recolor, duplicate, and *clear all,* as with all commands, should be undoable. It is only for demonstration purposes that clear all is not undoable in this segment.

In general, menu events that change the document, such as *cut* and *paste,* should be undoable. Exceptions can be made if the cost is too great, as in *revert to previous version.*

Menu events that only change the selection or view, such as *select all* or *hide ruler,* should not be undoable.

Consider the following example. A user cuts an object from a document, then selects the entire document. Suddenly she chooses to undo the last command. The last command is the *cut,* not the *select all.* Therefore the *cut* is undone

Sometimes menu events make such drastic changes to the display that it would be confusing to undo the last command. For example, within a bar graph in LisaGraph a user might change the text on the X–axis label, then immediately change the graph to a pie chart. Even though changing to a pie chart does not change

the document, undoing the text change at this point would be confusing to the user because the context is gone.

If a menu event changes an entire view, as opposed to just the selection, it should be made into a command. There are actually no hard and fast rules for what kind of events should be commands. Just use your best judgement.

Questions:

Commands Lab

Purpose:

- To implement *redo* and *duplicate* commands with undo. To implement *clear all* command without undo.

What you are about to do:

You will compile and run 6Boxer, then optionally modify the source. This should be done in the following steps:

1) Copy the following files onto your prefix volume.

   6UBoxer.TEXT
   6UBoxer2.TEXT
   6MBoxer.TEXT
   6PBoxer.TEXT

2) Compile, install, and run the sample application, **6Boxer**. Use **46** as the tool number.

3) Scan the listings of the four files in the sample application. These are included in the appendix, "Code Samples for this Segment".

   Study the edit and color menus in the phrase file.

4) *[Optional]* Make the *clear all* command undoable.

   Hint: Think about how you would save the view during the *doPhase* and the *redoPhase*.

   Warning: Be very careful about what gets restored as the selected object during the undoPhase.

Things to look out for:

- *Previous command is not committed.*

  You must create a command in response to a *clear all* menu event.

- *Boxes disappear, but do not reappear.*

  Are you saving the view's list of boxes when doing the command.

- *Boxes do not disappear.*

  Check what you are actually invalidating.

  Does your command's **Perform** method change the view?

Code Sample
for this
Segment

```
26
SLOT2CHAN1
;no assembler files
$
;no building blocks
$
;no links
$
y
y
n
BoxNum6
```

```
1
3
2500
$-#BOOT-TK/PABC

; Apple building block phrase files can be included here

1000
6Boxer

;     PROCEDURE {TBoxWindow.}NewCommand
;     cantUndo         = 1001;
1001 caution cancel alert
You will not be able to undo ClearAll.

0

1
$-#BOOT-TK/PABC~File/Print

2
Edit
Undo Last Change#205
-
Duplicate/D#1011
-
Clear All/Z#1012
-

3
Shades
White#1006
Light Gray#1007
Gray#1008
Dark Gray#1009
Black#1010

5
$-#BOOT-TK/PABC~Page Layout

99
$-#BOOT-TK/PABC~Debug

100
$-#BOOT-TK/PABC~Buzzwords
Create Box#2000
Move Selection#2001

0
```

```
PROGRAM M6Boxer;

USES
    {$U UObject      } UObject,

  {$IFC libraryVersion <= 20}
    {$U UFont}               UFont,
  {$ENDC}

    {$U QuickDraw    } QuickDraw,
    {$U UDraw        } UDraw,
    {$U UABC         } UABC,

    {$U U6Boxer      } U6Boxer;

CONST

    phraseVersion = 1;

BEGIN

    process := TBoxProcess.CREATE;
    process.Commence(phraseVersion);
    process.Run;
    process.Complete(TRUE);

END.
```

```
1    1 --   {$P}
1    2 --       {This segment of Boxer implements commands with undo}
1    3 --       {Copyright 1983, Apple Computer Inc.}
1    4 --
1    5 --   UNIT U6Boxer;
1    6 --
1    7 --   INTERFACE
1    8 --
1    9 --   USES
1   10 --       {$U UObject}            UObject,
1   11 --
1   12 --     {$IFC libraryVersion <= 20}
1   13 --       {$U UFont}              UFont,
1   14 --     {$ENDC}
1   15 --
1   16 --       {$U QuickDraw}          QuickDraw,
1   17 --       {$U UDraw}              UDraw,
1   18 --       {$U UABC}               UABC;
1   19 --
1   20 --   CONST
1   21 --       colorWhite = 1;
1   22 --       colorLtGray = 2;
1   23 --       colorGray = 3;
1   24 --       colorDkGray = 4;
1   25 --       colorBlack = 5;
1   26 --
1   27 --      { selection kinds }
1   28 --       boxSelectionKind = 1;
1   29 --       createBoxSelectionKind = 2;
1   30 --
1   31 --      { Menus }
1   32 --       uWhite = 1006;
1   33 --       uLtGray = 1007;
1   34 --       uGray = 1008;
1   35 --       uDkGray = 1009;
1   36 --       uBlack = 1010;
1   37 --       uDuplicate = 1011;
1   38 --       uClearAll = 1012;
1   39 --
1   40 --      { Phrases }
1   41 --       cantUndo        = 1001;
1   42 --
1   43 --   TYPE
1   44 --
1   45 --       TColor = colorWhite..colorBlack; {color of a box}
1   46 --
1   47 --     {New Classes for this Application}
1   48 --
1   49 --       TBox = SUBCLASS OF TObject
1   50 --
1   51 --           {Variables}
1   52 --           shapeLRect:      LRect;
1   53 --           color:           TColor;
1   54 --
1   55 --         {Creation/Destruction}
1   56 --           FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
1   57 --
1   58 --         { Highlighting support }
1   59 --           PROCEDURE TBox.PaintHandles;
1   60 --
1   61 --         { Framing while creating }
1   62 --           PROCEDURE TBox.DrawFrame;
1   63 --
1   64 --         {Display}
1   65 --           PROCEDURE TBox.Draw;
1   66 --           END;
1   67 --
1   68 --
1   69 --       TBoxView = SUBCLASS OF TView
1   70 --
1   71 --         {Variables}
1   72 --           boxList:         TList;
1   73 --
1   74 --         {Creation/Destruction}
1   75 --           FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
1   76 --                                       : TBoxView;
1   77 --
1   78 --           FUNCTION  TBoxView.BoxWith(LPt: LPoint): TBox;
1   79 --
1   80 --         {Invalidation}
1   81 --           PROCEDURE TBoxView.InvalBox(invalLRect: LRect);
1   82 --
1   83 --           PROCEDURE TBoxView.MousePress(mouseLPt: LPoint); OVERRIDE;
1   84 --
1   85 --         {Display}
1   86 --           PROCEDURE TBoxView.Draw; OVERRIDE;
1   87 --
1   88 --         {Initialization}
1   89 --           PROCEDURE TBoxView.InitBoxList(itsHeap: THeap);
1   90 --           FUNCTION  TBoxView.NoSelection: TSelection; OVERRIDE;
1   91 --          END;
1   92 --
1   93 --
1   94 --       TBoxSelection = SUBCLASS OF TSelection
1   95 --
1   96 --         {Variables}
1   97 --           box:  TBox;
1   98 --
1   99 --         {Creation/Destruction}
1  100 --           FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsBox: TBox;
1  101 --                                        itsKind: INTEGER; itsAnchorLPt: LPoint): TBoxSelection;
1  102 --
1  103 --         {Drawing - per pad}
1  104 --           PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit); OVERRIDE;
1  105 --
1  106 --         {Selection - per pad}
1  107 --           PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  108 --           PROCEDURE TBoxSelection.MouseRelease; OVERRIDE;
1  109 --
1  110 --         {Command Dispatch}
```

```
1  111 --           FUNCTION TBoxSelection.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  112 --           FUNCTION TBoxSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
1  113 --                                                      : BOOLEAN; OVERRIDE;
1  114 --        END;
1  115 --
1  116 --
1  117 --        TCreateBoxSelection = SUBCLASS OF TSelection
1  118 --
1  119 --          {Variables}
1  120 --            box:   TBox;
1  121 --
1  122 --          {Creation/Destruction}
1  123 --            FUNCTION  TCreateBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView;
1  124 --                                                      itsAnchorLPt: LPoint): TCreateBoxSelection;
1  125 --
1  126 --          {Selection - per pad}
1  127 --            PROCEDURE TCreateBoxSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  128 --            PROCEDURE TCreateBoxSelection.MouseRelease; OVERRIDE;
1  129 --        END;
1  130 --
1  131 --
1  132 --          { This command recolors the selected box and is not undoable}
1  133 --        TRecolorCmd = SUBCLASS OF TCommand
1  134 --            Box:     TBox;
1  135 --            color:   TColor;
1  136 --
1  137 --          {Creation}
1  138 --            FUNCTION  TRecolorCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  139 --                                         itsView: TBoxView; itsBox: TBox; itsColor: TColor): TRecolorCmd;
1  140 --
1  141 --            PROCEDURE TRecolorCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  142 --            END;
1  143 --
1  144 --
1  145 --          { This command duplicates the selected box and is undoable }
1  146 --        TDuplicateCmd = SUBCLASS OF TCommand
1  147 --            oldBox, newBox:     TBox;
1  148 --
1  149 --          {Creation}
1  150 --            FUNCTION  TDuplicateCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  151 --                                           itsView: TboxView; itsBox: TBox): TDuplicateCmd;
1  152 --
1  153 --          {Command Execution}
1  154 --            PROCEDURE TDuplicateCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  155 --            END;
1  156 --
1  157 --
1  158 --        TBoxProcess = SUBCLASS OF TProcess
1  159 --
1  160 --          {Creation/Destruction}
1  161 --            FUNCTION  TBoxProcess.CREATE: TBoxProcess;
1  162 --            FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
1  163 --                                                 : TDocManager; OVERRIDE;
1  164 --            END;
1  165 --
1  166 --
1  167 --        TBoxDocManager = SUBCLASS OF TDocManager
1  168 --
1  169 --          {Creation/Destruction}
1  170 --            FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
1  171 --                                            : TBoxDocManager;
1  172 --            FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID): TWindow; OVERRIDE;
1  173 --            END;
1  174 --
1  175 --
1  176 --        TBoxWindow = SUBCLASS OF TWindow
1  177 --
1  178 --          {Creation/Destruction}
1  179 --            FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
1  180 --
1  181 --          {Document Creation}
1  182 --            PROCEDURE {TBoxWindow.} BlankStationery; OVERRIDE;
1  183 --
1  184 --          {Commands}
1  185 --            PROCEDURE TBoxWindow.ClearAll;
1  186 --            FUNCTION  TBoxWindow.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  187 --            FUNCTION  TBoxWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN; OVERRIDE;
1  188 --            END;
1  189 --
1  190 --
1  191 --
1  192 --    IMPLEMENTATION
1  193 --
1  194 --      {$I U6Boxer2.text}
2    1 --    {U6BOXER2}
2    2 --
2    3 --    METHODS OF TBox;
2    4 --
2    5 -- A      FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
2    6 0- A      BEGIN
2    7 --            {$IFC fTrace}BP(11);{$ENDC}
2    8 --            SELF := NewObject(itsHeap, THISCLASS);
2    9 --            WITH SELF DO
2   10 1-              BEGIN
2   11 --              shapeLRect := zeroLRect;
2   12 --              color := colorGray;
2   13 -1              END;
2   14 --            {$IFC fTrace}EP;{$ENDC}
2   15 -0 A      END;
2   16 --
2   17 --      {$IFC fDebugMethods}
2   18 -- A      PROCEDURE TBox.Fields(PROCEDURE Field(nameAndType: S255));
2   19 0- A      BEGIN
2   20 --            Field('shapeLRect: LRect');
2   21 --            Field('color: INTEGER');
2   22 --            Field('');
2   23 -0 A      END;
2   24 --        {$ENDC}
2   25 --
2   26 --
```

```
2    27 --        {This draws a particular box}
2    28 -- A      PROCEDURE TBox.Draw;
2    29 --        VAR lPat:    LPattern;
2    30 0- A      BEGIN
2    31 --            {$IFC fTrace}BP(10);{$ENDC}
2    32 --            PenNormal;
2    33 --
2    34 --            IF LRectIsVisible(SELF.shapeLRect) THEN {this box needs to be drawn}
2    35 1-                BEGIN
2    36 --              {Get a Quickdraw pattern to represent the box's color}
2    37 2-                CASE SELF.color OF
2    38 --                    colorWhite:     lPat := lPatWhite;
2    39 --                    colorLtGray:    lPat := lPatLtGray;
2    40 --                    colorGray:      lPat := lPatGray;
2    41 --                    colorDkGray:    lPat := lPatDkGray;
2    42 --                    colorBlack:     lPat := lPatBlack;
2    43 --                    OTHERWISE       lPat := lPatWhite; {this case should not happen}
2    44 -2                    END;
2    45 --
2    46 --               {Fill the box with the pattern, and draw a frame around it}
2    47 --                FillLRect(SELF.shapeLRect, lPat);
2    48 --                FrameLRect(SELF.shapeLRect);
2    49 -1                END;
2    50 --            {$IFC fTrace}EP;{$ENDC}
2    51 -0 A      END;
2    52 --
2    53 --        { Frame a particular box}
2    54 -- A      PROCEDURE TBox.DrawFrame;
2    55 0- A      BEGIN
2    56 --            {$IFC fTrace}BP(10);{$ENDC}
2    57 --            PenNormal;
2    58 --            PenMode(PatXOr);
2    59 --            FrameLRect(SELF.shapeLRect);
2    60 --            {$IFC fTrace}EP;{$ENDC}
2    61 -0 A      END;
2    62 --
2    63 --         {This calls the DoToHandle Procedure once for each handle LRect; user of this method must
2    64 --          set up the pen pattern and mode before calling}
2    65 -- A      PROCEDURE TBox.PaintHandles;
2    66 --        VAR hLRect,
2    67 --            shapeLRect: LRect;
2    68 --            dh, dv:     LONGINT;
2    69 --
2    70 -- B          PROCEDURE MoveHandleAndPaint(hOffset, vOffset: LONGINT);
2    71 0- B          BEGIN
2    72 --                OffsetLRect(hLRect, hOffset, vOffset);
2    73 --                PaintLRect(hLRect);
2    74 -0 B          END;
2    75 --
2    76 0- A      BEGIN
2    77 --            {$IFC fTrace}BP(10);{$ENDC}
2    78 --            SetLRect(hLRect, -3, -2, 3, 2);
2    79 --            shapeLRect := SELF.shapeLRect;
2    80 --            WITH shapeLRect DO
2    81 1-                BEGIN
2    82 --                dh := right - left;
2    83 --                dv := bottom - top;
2    84 --                MoveHandleAndPaint(left, top);      {draw top left handle}
2    85 -1                END;
2    86 --            MoveHandleAndPaint(dh, 0);              {then top right}
2    87 --            MoveHandleAndPaint(0, dv);              {then bottom right}
2    88 --            MoveHandleAndPaint(-dh, 0);             {finally bottom left}
2    89 --            {$IFC fTrace}EP;{$ENDC}
2    90 -0 A      END;
2    91 --
2    92 --    END;
2    93 --
2    94 --
2    95 --    METHODS OF TBoxView;
2    96 --
2    97 -- A      FUNCTION TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
2    98 --                                     : TBoxView;
2    99 0- A      BEGIN
2   100 --            {$IFC fTrace}BP(11);{$ENDC}
2   101 --            IF object = NIL THEN
2   102 --                object := NewObject(itsHeap, THISCLASS);
2   103 --             SELF := TBoxView(itsPanel.NewView(object, itsExtent, TPrintManager.CREATE(NIL, itsHeap),
2   104 --                                         stdMargins, TRUE));
2   105 --            {$IFC fTrace}EP;{$ENDC}
2   106 -0 A      END;
2   107 --
2   108 --    {$IFC fDebugMethods}
2   109 -- A      PROCEDURE TBoxView.Fields(PROCEDURE Field(nameAndType: S255));
2   110 0- A      BEGIN
2   111 --            TView.Fields(Field);
2   112 --            Field('boxList: TList');
2   113 -0 A      END;
2   114 --    {$ENDC}
2   115 --
2   116 --
2   117 --        {This returns the box containing a certain point}
2   118 -- A      FUNCTION TBoxView.BoxWith(LPt: LPoint): TBox;
2   119 --        VAR box:    TBox;
2   120 --            s:      TListScanner;
2   121 0- A      BEGIN
2   122 --            {$IFC fTrace}BP(11);{$ENDC}
2   123 --            boxWith := NIL;
2   124 --            s := SELF.boxList.Scanner;
2   125 --            WHILE s.Scan(box) DO
2   126 --                IF LPtInLRect(LPt, box.shapeLRect) THEN
2   127 --                    boxWith := box;
2   128 --            {$IFC fTrace}EP;{$ENDC}
2   129 -0 A      END;
2   130 --
2   131 --
2   132 --        {This draws the list of boxes}
2   133 -- A      PROCEDURE TBoxView.Draw;
2   134 --        VAR box:    TBox;
2   135 --            s:      TListScanner;
2   136 0- A      BEGIN
```

```
2  137 --             {$IFC fTrace}BP(10);{$ENDC}
2  138 --             s := SELF.boxList.Scanner;
2  139 --             WHILE s.Scan(box) DO
2  140 --                 box.Draw;
2  141 --             {$IFC fTrace}EP;{$ENDC}
2  142 -0 A       END;
2  143 --
2  144 --
2  145 --          {This determines which type of selection to create}
2  146 -- A       PROCEDURE TBoxView.MousePress(mouseLPt: LPoint);
2  147 --         VAR aSelection: TSelection;
2  148 --             panel:      TPanel;
2  149 --             box:        TBox;
2  150 --
2  151 0- A        BEGIN
2  152 --             {$IFC fTrace}BP(11);{$ENDC}
2  153 --             panel := SELF.panel;
2  154 --             panel.Highlight(panel.selection, hOntoOff);          {Turn off the old highlighting}
2  155 --             box := SELF.BoxWith(mouseLPt);        {Find the box the user clicked on}
2  156 --
2  157 --             IF box = NIL THEN
2  158 --                 {Create an instance of TCreateBoxSelection}
2  159 --                 aSelection := panel.selection.FreedAndReplacedBy(
2  160 --                                         TCreateBoxSelection.CREATE(NIL, SELF.heap, SELF, mouseLPt))
2  161 --             ELSE
2  162 --                 {Create an instance of TBoxSelection}
2  163 --                 aSelection := panel.selection.FreedAndReplacedBy(
2  164 --                             TBoxSelection.CREATE(NIL, SELF.heap, SELF, box, boxSelectionKind, mouseLPt));
2  165 --
2  166 --             panel.Highlight(panel.selection, hOffToOn);    {Turn on the highlighting for the newly selected box}
2  167 --
2  168 --             self.panel.selection.MarkChanged;    {Allow the document to be saved so that any changes made}
2  169 --                                                  {can become permanent}
2  170 --             {$IFC fTrace}EP;{$ENDC}
2  171 -0 A        END;
2  172 --
2  173 --
2  174 -- A       PROCEDURE TBoxView.InvalBox(invalLRect: LRect);
2  175 0- A       BEGIN
2  176 --             {$IFC fTrace}BP(10);{$ENDC}
2  177 --             InsetLRect(invalLRect, -3, -2);
2  178 --             SELF.panel.InvalLRect(invalLRect);
2  179 --             {$IFC fTrace}EP;{$ENDC}
2  180 -0 A       END;
2  181 --
2  182 --
2  183 -- A       PROCEDURE TBoxView.InitBoxList (itsHeap: THeap);
2  184 --         VAR  boxList: TList;
2  185 0- A       BEGIN
2  186 --            {$IFC fTrace}BP(11);{$ENDC}
2  187 --             boxList := TList.CREATE(NIL, itsHeap, 0);
2  188 --             SELF.boxList := boxList;
2  189 --             {$IFC fTrace}EP;{$ENDC}
2  190 -0 A       END;
2  191 --
2  192 --
2  193 -- A       FUNCTION TBoxView.NoSelection: TSelection;
2  194 0- A       BEGIN
2  195 --            {$IFC fTrace}BP(11);{$ENDC}
2  196 --             NoSelection := TBoxSelection.CREATE(NIL, SELF.Heap, SELF, NIL, nothingKind, zeroLPt);
2  197 --             {$IFC fTrace}EP;{$ENDC}
2  198 -0 A       END;
2  199 --
2  200 --     END;
2  201 --
2  202 --
2  203 --     METHODS OF TBoxSelection;
2  204 --
2  205 -- A       FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsBox: TBox;
2  206 --                                     itsKind: INTEGER; itsAnchorLPt: LPoint): TBoxSelection;
2  207 0- A       BEGIN
2  208 --             {$IFC fTrace}BP(11);{$ENDC}
2  209 --             IF object = NIL THEN
2  210 --                 object := NewObject(itsHeap, THISCLASS);
2  211 --             SELF := TBoxSelection(TSelection.CREATE(object, itsHeap, itsView, itsKind, itsAnchorLPt));
2  212 --
2  213 --             SELF.box := itsBox;
2  214 --             {$IFC fTrace}EP;{$ENDC}
2  215 -0 A       END;
2  216 --
2  217 --      {$IFC fDebugMethods}
2  218 -- A       PROCEDURE TBoxSelection.Fields(PROCEDURE Field(nameAndType: S255));
2  219 0- A       BEGIN
2  220 --             TSelection.Fields(Field);
2  221 --             Field('box: TBox');
2  222 -0 A       END;
2  223 --      {$ENDC}
2  224 --
2  225 --
2  226 --            {This draws the handles on the selected box}
2  227 -- A       PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit);
2  228 0- A       BEGIN
2  229 --             {$IFC fTrace}BP(11);{$ENDC}
2  230 --             IF SELF.kind <> nothingKind THEN
2  231 1-                 BEGIN
2  232 --                 thePad.SetPenToHighlight(highTransit);  {set the drawing mode according to desired highlighting}
2  233 --                 SELF.box.PaintHandles;                  {draw the handles on the box}
2  234 -1                 END;
2  235 --             {$IFC fTrace}EP;{$ENDC}
2  236 -0 A       END;
2  237 --
2  238 --
2  239 --         {This is called when the user moves the mouse after pressing the button}
2  240 -- A       PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint);
2  241 --         VAR diffLPt:    LPoint;
2  242 --             boxView:    TBoxView;
2  243 --             shapeLRect: LRect;
2  244 0- A       BEGIN
2  245 --             {$IFC fTrace}BP(11);{$ENDC}
2  246 --             boxView := TBoxView(SELF.view);
```

```
2  247 --
2  248 --             {How far did mouse move?}
2  249 --          LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
2  250 --
2  251 --             {Move it if delta is nonzero}
2  252 --          IF NOT EqualLPt(diffLPt, zeroLPt) THEN
2  253 1-              BEGIN
2  254 --              SELF.currLPt := mouseLPt;
2  255 --
2  256 --              shapeLRect := SELF.box.shapeLRect;
2  257 --                   {Compute old and new positions of box}
2  258 --              boxView.InvalBox(shapeLRect);
2  259 --              OffsetLRect(shapeLRect, diffLPt.h, diffLPt.v);
2  260 --              boxView.InvalBox(shapeLRect);
2  261 --
2  262 --              SELF.box.shapeLRect := shapeLRect;
2  263 -1              END;
2  264 --          {$IFC fTrace}EP;{$ENDC}
2  265 -0 A     END;
2  266 --
2  267 --
2  268 -- A     PROCEDURE TBoxSelection.MouseRelease;
2  269 0- A     BEGIN
2  270 --          {$IFC fTrace}BP(11);{$ENDC}
2  271 --            { If the mouse moved then commit any outstanding command }
2  272 --          IF NOT EqualLPt(SELF.currLPt, SELF.anchorLPt) THEN
2  273 --              SELF.window.CommitLast;
2  274 --          {$IFC fTrace}EP;{$ENDC}
2  275 -0 A     END;
2  276 --
2  277 --
2  278 -- A     FUNCTION  TBoxSelection.NewCommand(cmdNumber: TCmdNumber): TCommand;
2  279 --       VAR boxView:     TBoxView;
2  280 --           heap:        THeap;
2  281 0- A     BEGIN
2  282 --          {$IFC fTrace}BP(11);{$ENDC}
2  283 --
2  284 --          boxView := TBoxView(SELF.view);
2  285 --          heap := SELF.Heap;
2  286 --
2  287 1-          CASE cmdNumber OF
2  288 --              uWhite, uLtGray, uGray, uDkGray, uBlack:
2  289 --                  NewCommand := TRecolorCmd.CREATE(NIL, heap, cmdNumber, boxView, SELF.box,
2  290 --                                                                   cmdNumber - uWhite + colorWhite);
2  291 --
2  292 --              uDuplicate:
2  293 --                  NewCommand := TDuplicateCmd.CREATE(NIL, heap, cmdNumber, boxView, SELF.box);
2  294 --
2  295 --              OTHERWISE
2  296 --                  NewCommand := SUPERSELF.NewCommand(cmdNumber);
2  297 -1              END;
2  298 --          {$IFC fTrace}EP;{$ENDC}
2  299 -0 A     END;
2  300 --
2  301 --
2  302 -- A     FUNCTION  TBoxSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
2  303 0- A     BEGIN
2  304 --          {$IFC fTrace}BP(11);{$ENDC}
2  305 1-          CASE cmdNumber OF
2  306 --              uWhite, uLtGray, uGray, uDkGray, uBlack,
2  307 --                      uDuplicate:
2  308 --                                CanDoCommand := SELF.kind <> nothingKind;
2  309 --
2  310 --              OTHERWISE
2  311 --                                CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
2  312 -1              END;
2  313 --          {$IFC fTrace}EP;{$ENDC}
2  314 -0 A     END;
2  315 --
2  316 --  END;
2  317 --
2  318 --
2  319 --  METHODS OF TCreateBoxSelection;
2  320 --
2  321 -- A     FUNCTION  TCreateBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView;
2  322 --                                          itsAnchorLPt: LPoint): TCreateBoxSelection;
2  323 --       VAR box: TBox;
2  324 0- A     BEGIN
2  325 --          {$IFC fTrace}BP(11);{$ENDC}
2  326 --          IF object = NIL THEN
2  327 --             object := NewObject(itsHeap, THISCLASS);
2  328 --          SELF := TCreateBoxSelection(TSelection.CREATE(object, itsHeap, itsView, createBoxSelectionKind,
2  329 --                                                          itsAnchorLPt));
2  330 --          box := TBox.CREATE(NIL, SELF.heap);
2  331 --          SELF.box := box;
2  332 --          {$IFC fTrace}EP;{$ENDC}
2  333 -0 A     END;
2  334 --
2  335 --   {$IFC fDebugMethods}
2  336 -- A     PROCEDURE TCreateBoxSelection.Fields(PROCEDURE Field(nameAndType: S255));
2  337 0- A     BEGIN
2  338 --          TSelection.Fields(Field);
2  339 --          Field('box: TBox');
2  340 -0 A     END;
2  341 --    {$ENDC}
2  342 --
2  343 --       {This is called when the user moves the mouse after pressing the button}
2  344 -- A     PROCEDURE TCreateBoxSelection.MouseMove(mouseLPt: LPoint);
2  345 --       VAR maxBoxLRect:     LRect;
2  346 --           diffLPt:         LPoint;
2  347 --           boxView:         TBoxView;
2  348 --           box:             TBox;
2  349 --
2  350 -- B        PROCEDURE DrawTheFrame;
2  351 0- B        BEGIN
2  352 --              box.DrawFrame;
2  353 -0 B        END;
2  354 --
2  355 0- A     BEGIN
2  356 --          {$IFC fTrace}BP(11);{$ENDC}
```

```
2  357 --            boxView := TBoxView(SELF.view);
2  358 --            box := SELF.box;
2  359 --
2  360 --            { In Boxer it is possible to draw a box greater than allowed by a 16 bit rectangle.  These three
2  361 --              lines force the rectangle to within 16 bits. }
2  362 --       {$H-} WITH SELF.anchorLPt DO
2  363 --                SetLRect(maxBoxLRect, h+10-MAXINT, v+10-MAXINT, h+MAXINT-10, v+MAXINT-10);
2  364 --       {$H+} LRectHaveLPt(maxBoxLRect, mouseLPt);
2  365 --
2  366 --            LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
2  367 --            IF NOT EqualLPt(diffLPt, zeroLPt) THEN
2  368 1-                BEGIN
2  369 --                SELF.currLPt := mouseLPt;
2  370 --
2  371 --                boxView.panel.OnAllPadsDo(DrawTheFrame);  {erase old frame}
2  372 --                WITH box DO
2  373 2-                    BEGIN
2  374 --                    shapeLRect.topLeft := SELF.anchorLPt;
2  375 --                    shapeLRect.botRight := mouseLPt;
2  376 -2                    END;
2  377 --
2  378 --         {$H-}    RectifyLRect(box.shapeLRect);   {$H+}
2  379 --                boxView.panel.OnAllPadsDo(DrawTheFrame);  {draw new frame}
2  380 -1                END;
2  381 --            {$IFC fTrace}EP;{$ENDC}
2  382 -0 A      END;
2  383 --
2  384 --
2  385 -- A      PROCEDURE TCreateBoxSelection.MouseRelease;
2  386 --        VAR thisBox:        TBox;
2  387 --            boxView:        TBoxView;
2  388 --            drawnLRect:     LRect;
2  389 --            aSelection:     TSelection;
2  390 --            panel:          TPanel;
2  391 --
2  392 -- B          PROCEDURE DrawTheFrame;
2  393 0- B          BEGIN
2  394 --                thisBox.DrawFrame;
2  395 -0 B          END;
2  396 --
2  397 0- A      BEGIN
2  398 --            {$IFC fTrace}BP(11);{$ENDC}
2  399 --            boxView := TBoxView(SELF.view);
2  400 --            panel := boxView.panel;
2  401 --            thisBox := SELF.box;
2  402 --            panel.OnAllPadsDo(DrawTheFrame);
2  403 --            drawnLRect := thisBox.shapeLRect;
2  404 --
2  405 --            { Independant of whether we threw the boxed away or not we must create an instance of TBoxSelection
2  406 --              to replace the now useless instance of TCreateBoxSelection using the kind set above. }
2  407 --            aSelection := SELF.FreedAndReplacedBy(
2  408 --                            TBoxSelection.CREATE(NIL, SELF.heap, boxView, thisBox, boxSelectionKind,
2  409 --                                                 drawnLRect.topLeft));
2  410 --
2  411 --            boxView.InvalBox(drawnLRect);
2  412 --
2  413 --            {If the box is not big enough then throw it away, otherwise put it in the list}
2  414 --            IF (drawnLRect.right - drawnLRect.left <=4) OR (drawnLRect.bottom - drawnLRect.top <=4) THEN
2  415 1-                BEGIN
2  416 --                aSelection.kind := nothingKind;
2  417 --                thisBox.Free;
2  418 -1                END
2  419 1-            ELSE BEGIN
2  420 --                { Commit any outstanding command }
2  421 --                SELF.window.CommitLast;
2  422 --
2  423 --                boxView.boxList.InsLast(thisBox);
2  424 -1            END;
2  425 --            {$IFC fTrace}EP;{$ENDC}
2  426 -0 A      END;
2  427 --
2  428 --    END;
2  429 --
2  430 --
2  431 --
2  432 --    METHODS OF TRecolorCmd;
2  433 --
2  434 -- A      FUNCTION  TRecolorCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  435 --                                      itsView: TBoxView; itsBox: TBox; itsColor: TColor): TRecolorCmd;
2  436 0- A      BEGIN
2  437 --            {$IFC fTrace}BP(10);{$ENDC}
2  438 --            IF object = NIL THEN
2  439 --                object := NewObject(itsHeap, THISCLASS);
2  440 --            SELF := TRecolorCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealAll));
2  441 --            SELF.color := itsColor;
2  442 --            SELF.box := itsBox;
2  443 --            {$IFC fTrace}EP;{$ENDC}
2  444 -0 A      END;
2  445 --
2  446 --    {$IFC fDebugMethods}
2  447 -- A      PROCEDURE TRecolorCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  448 0- A      BEGIN
2  449 --            TCommand.Fields(Field);
2  450 --            Field('Color: INTEGER');
2  451 --            Field('box: TBox');
2  452 -0 A      END;
2  453 --     {$ENDC}
2  454 --
2  455 --
2  456 -- A      PROCEDURE TRecolorCmd.Perform(cmdPhase: TCmdPhase);
2  457 --        VAR boxView:     TBoxView;
2  458 --            tempColor:   TColor;
2  459 --            box:         TBox;
2  460 0- A      BEGIN
2  461 --            {$IFC fTrace}BP(12);{$ENDC}
2  462 --            boxView := TBoxView(SELF.image);
2  463 --            box := SELF.box;
2  464 --
2  465 1-            CASE cmdPhase OF
2  466 --                undoPhase, redoPhase, doPhase:
```

```
2  467 2-                    BEGIN
2  468 --                    tempColor := SELF.color;
2  469 --                    SELF.color := box.color;
2  470 --                    box.color := tempColor;
2  471 --                    boxView.InvalBox(box.shapeLRect);
2  472 -2                    END
2  473 -1              END {CASE};
2  474 --
2  475 --              self.image.view.panel.selection.MarkChanged;    {allow this document to be saved}
2  476 --             {$IFC fTrace}EP;{$ENDC}
2  477 -0 A         END;
2  478 --
2  479 --      END;
2  480 --
2  481 --
2  482 --
2  483 --      METHODS OF TDuplicateCmd;
2  484 --
2  485 -- A        FUNCTION  TDuplicateCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  486 --                                        itsView: TBoxView; itsBox: TBox): TDuplicateCmd;
2  487 --           VAR box:     TBox;
2  488 0- A         BEGIN
2  489 --               {$IFC fTrace}BP(10);{$ENDC}
2  490 --               IF object = NIL THEN
2  491 --                   object := NewObject(itsHeap, THISCLASS);
2  492 --               SELF := TDuplicateCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealAll));
2  493 --
2  494 --               SELF.oldBox := itsBox;
2  495 --               box := TBox(itsBox.Clone(SELF.heap));
2  496 --        {$H-} OffSetLRect(box.shapeLRect, 20, 20);     {$H+}
2  497 --               SELF.newBox := box;
2  498 --               {$IFC fTrace}EP;{$ENDC}
2  499 -0 A         END;
2  500 --
2  501 --
2  502 -- A        PROCEDURE TDuplicateCmd.Free;
2  503 0- A         BEGIN
2  504 --               {$IFC fTrace}BP(10);{$ENDC}
2  505 --               IF NOT SELF.doing THEN
2  506 --                   Free(SELF.newBox);
2  507 --               SELF.FreeObject;
2  508 --               {$IFC fTrace}EP;{$ENDC}
2  509 -0 A         END;
2  510 --
2  511 --
2  512 --      {$IFC fDebugMethods}
2  513 -- A        PROCEDURE TDuplicateCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  514 0- A         BEGIN
2  515 --               TCommand.Fields(Field);
2  516 --               Field('oldBox: TBox');
2  517 --               Field('newBox: TBox');
2  518 -0 A         END;
2  519 --       {$ENDC}
2  520 --
2  521 --
2  522 -- A        PROCEDURE TDuplicateCmd.Perform(cmdPhase: TCmdPhase);
2  523 --           VAR boxView:              TBoxView;
2  524 --               box:                  TBox;
2  525 --               thisSelection:        TBoxSelection;
2  526 0- A         BEGIN
2  527 --               {$IFC fTrace}BP(12);{$ENDC}
2  528 --               boxView := TBoxView(SELF.image);
2  529 --               thisSelection := TBoxSelection(boxView.panel.selection);
2  530 --
2  531 --                   {------------------------------------------------------------------------------------
2  532 --                    The current selection is unhighlighted before performing the command as the result
2  533 --                    of the following command fields set by TCommand.CREATE:
2  534 --                                             unHiliteBefore[doPhase..redoPhase] <- TRUE
2  535 --
2  536 --                    The resulting selection is highlighted after performing the command as the result of the
2  537 --                    following command fields set by TCommand.CREATE:
2  538 --                                             hiLiteAfter   [doPhase..redoPhase] <- TRUE
2  539 --                    ------------------------------------------------------------------------------------}
2  540 --
2  541 1-               CASE cmdPhase OF
2  542 --                   doPhase, redoPhase:
2  543 2-                       BEGIN
2  544 --                       thisSelection.box := SELF.newBox;
2  545 --                       boxView.boxList.InsLast(SELF.newBox);
2  546 -2                       END;
2  547 --
2  548 --                   undoPhase:
2  549 2-                       BEGIN
2  550 --                       boxView.boxList.DelLast(FALSE);
2  551 --                       WITH thisSelection DO
2  552 --                           box := SELF.oldBox;
2  553 -2                       END;
2  554 -1               END {CASE};
2  555 --
2  556 --               boxView.InvalBox(SELF.newBox.shapeLRect);
2  557 --
2  558 --               self.image.view.panel.selection.MarkChanged;    {allow this document to be saved}
2  559 --               {$IFC fTrace}EP;{$ENDC}
2  560 -0 A         END;
2  561 --
2  562 --      END;
2  563 --
2  564 --
2  565 --
2  566 --      METHODS OF TBoxProcess;
2  567 --
2  568 -- A        FUNCTION  TBoxProcess.CREATE: TBoxProcess;
2  569 0- A         BEGIN
2  570 --               {$IFC fTrace}BP(11);{$ENDC}
2  571 --               SELF := TBoxProcess(TProcess.CREATE(NewObject(mainHeap, THISCLASS), mainHeap));
2  572 --               {$IFC fTrace}EP;{$ENDC}
2  573 -0 A         END;
2  574 --
2  575 --
2  576 -- A        FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN): TDocManager;
```

```
2  577 0- A       BEGIN
2  578 --             {$IFC fTrace}BP(11);{$ENDC}
2  579 --             NewDocManager := TBoxDocManager.CREATE(NIL, mainHeap, volumePrefix);
2  580 --             {$IFC fTrace}EP;{$ENDC}
2  581 -0 A       END;
2  582 --
2  583 --    END;
2  584 --
2  585 --
2  586 --
2  587 --    METHODS OF TBoxDocManager;
2  588 --
2  589 -- A      FUNCTION TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
2  590 --                                        : TBoxDocManager;
2  591 0- A       BEGIN
2  592 --             {$IFC fTrace}BP(11);{$ENDC}
2  593 --             IF object = NIL THEN
2  594 --                object := NewObject(itsHeap, THISCLASS);
2  595 --             SELF := TBoxDocManager(TDocManager.CREATE(object, itsHeap, itsPathPrefix));
2  596 --             {$IFC fTrace}EP;{$ENDC}
2  597 -0 A       END;
2  598 --
2  599 --
2  600 -- A      FUNCTION TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID):TWindow;
2  601 0- A       BEGIN
2  602 --             {$IFC fTrace}BP(11);{$ENDC}
2  603 --             NewWindow := TBoxWindow.CREATE(NIL, heap, wmgrID);
2  604 --             {$IFC fTrace}EP;{$ENDC}
2  605 -0 A       END;
2  606 --
2  607 --    END;
2  608 --
2  609 --
2  610 --
2  611 --    METHODS OF TBoxWindow;
2  612 --
2  613 -- A      FUNCTION TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
2  614 0- A       BEGIN
2  615 --             {$IFC fTrace}BP(10);{$ENDC}
2  616 --             IF object = NIL THEN
2  617 --                object := NewObject(itsHeap, THISCLASS);
2  618 --             SELF := TBoxWindow(TWindow.CREATE(object, itsHeap, itsWmgrID, TRUE));
2  619 --             {$IFC fTrace}EP;{$ENDC}
2  620 -0 A       END;
2  621 --
2  622 --
2  623 -- A      PROCEDURE TBoxWindow.BlankStationery;
2  624 --         VAR viewLRect:        LRect;
2  625 --             panel:            TPanel;
2  626 --             boxView:          TBoxView;
2  627 --             aSelection:       TSelection;
2  628 0- A       BEGIN
2  629 --             {$IFC fTrace}BP(10);{$ENDC}
2  630 --              panel := TPanel.CREATE(NIL, SELF.Heap, SELF, 0, 0, [aScroll, aSplit], [aScroll, aSplit]);
2  631 --
2  632 --             SetLRect(viewLRect, 0, 0, 5000, 3000);
2  633 --             boxView := TBoxView.CREATE(NIL, SELF.Heap, panel, viewLRect);
2  634 --             boxView.InitBoxList(SELF.Heap);
2  635 --
2  636 --             {$IFC fTrace}EP;{$ENDC}
2  637 -0 A       END;
2  638 --
2  639 --
2  640 -- A      PROCEDURE TBoxWindow.ClearAll;
2  641 --         VAR boxView:      TBoxView;
2  642 --             panel:        Tpanel;
2  643 --             box:          TBox;
2  644 --             s:            TListScanner;
2  645 --             aSelection:   TSelection;
2  646 0- A       BEGIN
2  647 --             {$IFC fTrace}BP(10);{$ENDC}
2  648 --
2  649 --             panel := SELF.selectPanel;
2  650 --             boxView := TBoxView(panel.view);
2  651 --
2  652 --             s := boxView.boxList.scanner;
2  653 --             WHILE s.Scan(box) DO
2  654 --                 s.Delete(TRUE);
2  655 --             aSelection := panel.selection.FreedAndReplaceby(boxView.NoSelection);
2  656 --             panel.Invalidate;
2  657 --             {$IFC fTrace}EP;{$ENDC}
2  658 -0 A       END;
2  659 --
2  660 --
2  661 -- A      FUNCTION TBoxWindow.NewCommand(cmdNumber: TCmdNumber): TCommand;
2  662 0- A       BEGIN
2  663 --             {$IFC fTrace}BP(11);{$ENDC}
2  664 --
2  665 1-             CASE cmdNumber OF
2  666 --                 uClearAll:
2  667 --                     { put up an alert saying that this will not be undoable }
2  668 --                     IF process.caution(cantUndo) THEN
2  669 2-                         BEGIN
2  670 --                         NewCommand := TCommand.CREATE(NIL, SELF.heap, cmdNumber, SELF.selectPanel.view,
2  671 --                                                       FALSE, revealNone);
2  672 --                         SELF.ClearAll;
2  673 -2                         END;
2  674 --
2  675 --                 OTHERWISE
2  676 --                     NewCommand := SUPERSELF.NewCommand(cmdNumber);
2  677 -1                 END;
2  678 --             {$IFC fTrace}EP;{$ENDC}
2  679 -0 A       END;
2  680 --
2  681 --
2  682 -- A      FUNCTION TBoxWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
2  683 0- A       BEGIN
2  684 --             {$IFC fTrace}BP(11);{$ENDC}
2  685 1-             CASE cmdNumber OF
2  686 --                 uClearAll:
```

```
2  687 2-                    BEGIN
2  688 --                    CanDoCommand := TRUE;
2  689 -2                    END;
2  690 --               OTHERWISE
2  691 --                  CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
2  692 -1               END;
2  693 --          {$IFC fTrace}EP;{$ENDC}
2  694 -0 A      END;
2  695 --
2  696 --    END;
2  697 --
1  195 --
1  196 --    END.
```

```
 1. u6boxer.TEXT
 2. U6Boxer2.text

-B-
  BlankStationery   182•( 1)  623•( 2)
  Box               134 ( 1)
  box                97 ( 1)  120 ( 1)  119•( 2)  125 ( 2)  126 ( 2)  127 ( 2)  134•( 2)  139 ( 2)  140 ( 2)  149•( 2)
                    155=( 2)  157 ( 2)  164 ( 2)  213=( 2)  233 ( 2)  256 ( 2)  262 ( 2)  289 ( 2)  293 ( 2)  323•( 2)
                    330=( 2)  331=( 2)  331 ( 2)  348•( 2)  352 ( 2)  358=( 2)  358 ( 2)  372 ( 2)  378 ( 2)  401 ( 2)
                    442=( 2)  459•( 2)  463=( 2)  463 ( 2)  469 ( 2)  470 ( 2)  471 ( 2)  487•( 2)  495=( 2)  496 ( 2)
                    497 ( 2)  524•( 2)  544=( 2)  552=( 2)  643•( 2)  653 ( 2)
  boxList            72 ( 1)  124 ( 2)  138 ( 2)  184•( 2)  187=( 2)  188=( 2)  188 ( 2)  423 ( 2)  545 ( 2)  550 ( 2)
                    652 ( 2)
  boxSelectionKind   28•( 1)  164 ( 2)  408 ( 2)
  BoxWith            78•( 1)  118•( 2)  155 ( 2)

-C-
  CanDoCommand      112•( 1)  187•( 1)  302•( 2)  308=( 2)  311=( 2)  311 ( 2)  682•( 2)  688=( 2)  691=( 2)  691 ( 2)
  cantUndo           41•( 1)  668 ( 2)
  ClearAll          185•( 1)  640•( 2)  672 ( 2)
  color              53•( 1)  135•( 1)   12=( 2)   37 ( 2)  441=( 2)  468 ( 2)  469=( 2)  469 ( 2)  470=( 2)
  colorBlack         25•( 1)   45 ( 1)   42 ( 2)
  colorDkGray        24•( 1)   41 ( 2)
  colorGray          23•( 1)   12 ( 2)   40 ( 2)
  colorLtGray        22•( 1)   39 ( 2)
  colorWhite         21•( 1)   45 ( 1)   38 ( 2)  290 ( 2)
  CREATE             56•( 1)   75•( 1)  100•( 1)  123•( 1)  138•( 1)  150•( 1)  161 ( 1)  170 ( 1)  179 ( 1)    5•( 2)
                     97•( 2)  103 ( 2)  160 ( 2)  164 ( 2)  187 ( 2)  196 ( 2)  205•( 2)  211 ( 2)  289 ( 2)  293 ( 2)
                    321•( 2)  328 ( 2)  330 ( 2)  408 ( 2)  434•( 2)  440 ( 2)  485•( 2)  492 ( 2)  568•( 2)  571 ( 2)
                    579 ( 2)  589•( 2)  595 ( 2)  603 ( 2)  613•( 2)  618 ( 2)  630 ( 2)  633 ( 2)  670 ( 2)
  createBoxSelecti   29•( 1)  328 ( 2)

-D-
  Draw               65•( 1)   86•( 1)   28•( 2)  133•( 2)  140 ( 2)
  DrawFrame          62•( 1)   54•( 2)  352 ( 2)  394 ( 2)

-F-
  Fields             18•( 2)  109•( 2)  111 ( 2)  218•( 2)  220 ( 2)  336•( 2)  338 ( 2)  447•( 2)  449 ( 2)  513•( 2)
                    515 ( 2)
  Free              502•( 2)  506 ( 2)

-H-
  Highlight         104•( 1)  154 ( 2)  166 ( 2)  227•( 2)

-I-
  InitBoxList        89•( 1)  183•( 2)  634 ( 2)
  InvalBox           81•( 1)  174•( 2)  258 ( 2)  260 ( 2)  411 ( 2)  471 ( 2)  556 ( 2)

-L-
  LRect              52 ( 1)   67 ( 2)  243 ( 2)  345 ( 2)  388 ( 2)  624 ( 2)

-M-
  MouseMove         107•( 1)  127•( 1)  240•( 2)  344•( 2)
  MousePress         83•( 1)  146•( 2)
  MouseRelease      108•( 1)  128•( 1)  268•( 2)  385•( 2)

-N-
  newBox            147 ( 1)  497=( 2)  506 ( 2)  544 ( 2)  545 ( 2)  556 ( 2)
  NewCommand        111•( 1)  186•( 1)  278•( 2)  289=( 2)  293=( 2)  296=( 2)  296 ( 2)  661•( 2)  670=( 2)  676=( 2)
                    676 ( 2)
  NewDocManager     162•( 1)  576•( 2)  579=( 2)
  NewWindow         172•( 1)  600•( 2)  603=( 2)
  NoSelection        90•( 1)  193•( 2)  196=( 2)  655 ( 2)

-O-
  oldBox            147 ( 1)  494=( 2)  552 ( 2)

-P-
  PaintHandles       59•( 1)   65•( 2)  233 ( 2)
  Perform           141•( 1)  154•( 1)  456•( 2)  522•( 2)

-Q-
  QuickDraw          16•( 1)

-S-
  shapeLRect         52 ( 1)   11=( 2)   34 ( 2)   47 ( 2)   48 ( 2)   59 ( 2)   67•( 2)   79=( 2)   79 ( 2)   80 ( 2)
                    126 ( 2)  243•( 2)  256=( 2)  256 ( 2)  258 ( 2)  259 ( 2)  260 ( 2)  262=( 2)  262 ( 2)  374 ( 2)
                    375 ( 2)  378 ( 2)  403 ( 2)  471 ( 2)  496 ( 2)  556 ( 2)

-T-
  TBox               49•( 1)   56 ( 1)   78 ( 1)   97 ( 1)  120 ( 1)  134 ( 1)  147 ( 1)    3•( 2)    5 ( 2)  118 ( 2)
                    119 ( 2)  134 ( 2)  149 ( 2)  323 ( 2)  330 ( 2)  348 ( 2)  386 ( 2)  459 ( 2)  487 ( 2)  495 ( 2)
                    524 ( 2)  643 ( 2)
  TBoxDocManager    167•( 1)  171 ( 1)  579 ( 2)  587•( 2)  590 ( 2)  595 ( 2)
  TBoxProcess       158•( 1)  161 ( 1)  566•( 2)  568 ( 2)  571 ( 2)
  TBoxSelection      94•( 1)  101 ( 1)  164 ( 2)  196 ( 2)  203•( 2)  206 ( 2)  211 ( 2)  408 ( 2)  525 ( 2)  529 ( 2)
  TBoxView           69•( 1)   76 ( 1)   95•( 2)   98 ( 2)  103 ( 2)  242 ( 2)  246 ( 2)  279 ( 2)  284 ( 2)  347 ( 2)
                    357 ( 2)  387 ( 2)  399 ( 2)  457 ( 2)  462 ( 2)  523 ( 2)  528 ( 2)  626 ( 2)  633 ( 2)  641 ( 2)
                    650 ( 2)
  TBoxWindow        176•( 1)  179 ( 1)  603 ( 2)  611•( 2)  613 ( 2)  618 ( 2)
  TColor             45•( 1)   53 ( 1)  135 ( 1)  458 ( 2)
  TCommand          111 ( 1)  133 ( 1)  146 ( 1)  186 ( 1)  278 ( 2)  440 ( 2)  449 ( 2)  492 ( 2)  515 ( 2)  661 ( 2)
                    670 ( 2)
  TCreateBoxSelect  117•( 1)  124 ( 1)  160 ( 2)  319•( 2)  322 ( 2)  328 ( 2)
  TDocManager       163 ( 1)  167 ( 1)  576 ( 2)  595 ( 2)
  TDuplicateCmd     146•( 1)  151 ( 1)  293 ( 2)  483•( 2)  486 ( 2)  492 ( 2)
  TList              72 ( 1)  184 ( 2)  187 ( 2)
  TObject            49 ( 1)
  TProcess          158 ( 1)  571 ( 2)
  TRecolorCmd       133•( 1)  139 ( 1)  289 ( 2)  432•( 2)  435 ( 2)  440 ( 2)
  TSelection         90 ( 1)   94 ( 1)  117 ( 1)  147 ( 2)  193 ( 2)  211 ( 2)  220 ( 2)  328 ( 2)  338 ( 2)  389 ( 2)
                    627 ( 2)  645 ( 2)
  TView              69 ( 1)  111 ( 2)
  TWindow           172 ( 1)  176 ( 1)  600 ( 2)  618 ( 2)

-U-
  U6Boxer             5•( 1)
  UABC               18•( 1)
  uBlack             36•( 1)  288 ( 2)  306 ( 2)
  uClearAll          38•( 1)  666 ( 2)  686 ( 2)
```

```
uDkGray          35*( 1)  288 ( 2)  306 ( 2)
UDraw            17*( 1)
uDuplicate       37*( 1)  292 ( 2)  307 ( 2)
UFont            13*( 1)
uGray            34*( 1)  288 ( 2)  306 ( 2)
uLtGray          33*( 1)  288 ( 2)  306 ( 2)
UObject          10*( 1)
uWhite           32*( 1)  288 ( 2)  290 ( 2)  306 ( 2)

*** End Xref: 68 id's   419 references       [400264 bytes/4931 id's/40171 refs]
```

[Segment 10]

Filters

Purpose of this segment:

1. To introduce filters and filtered commands.
2. To reimplement *recolor* and *duplicate* to utilize filters.
3. To make *clear all* an undoable command using filters.

How to use this segment:

This is the tenth segment in the self—paced introduction to the ToolKit. This segment follows the segment on commands, and precedes the segment on advanced commands with cut & paste.

This segment presents a way to undo commands that make major changes to a document. This undo strategy is based upon the concept of filters.

INTRODUCTION TO FILTERS

When implementing undo there are two basic strategies. The first we used in the previous segment — let the command save the information necessary to restore the document before changing it.

It is easy to imagine cases where this strategy is too complicated to implement or where it requires too much space. For instance, to undo a shade command in LisaDraw you must remember the original color of every box that was affected by the command. To undo a cut in LisaDraw you must remember the objects themselves, their positions in the list, and any relation they have with other objects in the document.

Whenever the size or the complexity of the information necessary to restore the document becomes unreasonable there is a second undo strategy available. The second undo strategy involves a concept called a *filter*.

Filters provide a way to change the display of a document without making any changes to the document's data.

There are basically two types of filters: *transparencies* and *sieves*. The following scenarios illustrate the two types.

transparencies

Ferd Berfel is trying to design a house. He has a blueprint and wants to try out different changes to the blueprint without a damaging it. He lays a transparency, with the changes he is considering on it, over the blueprint. Then, when he views the blueprint through the transparency, he sees the blueprint as if the change were actually made. What he is seeing is the *virtual* blueprint. *The actual blueprint has not been changed.*

The transparency with its additions is the filter through which the blueprint is perceived. In Boxer, the blueprint would be the document's view object. *The view, of course, is displayed through the window.* The virtual view is the actual view plus the changes made by the command.

Transparencies are used by commands, such as *duplicate*, that add boxes to the display.

To undo a change, the document's window is updated with the filter removed. To redo a change, the window is updated with the filter reinstated. When Ferd is satisfied with a change, he commits the change to the blueprint.

sieves

Ferd Berfel is now trying to modify the recipe of the pie baked by a pie—making machine. The pie—making machine has various compartments, each holding a single ingredient (such as flour, butter, pecans, etc.). To make the pie, the machine measures out, in sequence, appropriate quantities of each ingredient in the recipe. During the process various ingredients are mixed together and layered into a pie pan. Finally the pie is baked.

Ferd Berfel decides that pecans are not needed for this pie; so he blocks the hole to the compartment holding the pecans. None the wiser, the pie—making machine proceeds to make another pie. The pie pops out of the oven without pecans.

The recipe for the pie without pecans is a *virtual* recipe. The actual recipe uses all of the original ingredients in the compartments of the machine. Blocking the compartment is like using a sieve to filter out that ingredient. Effectively, Ferd used a sieve to filter out the pecans. *Just as easily Ferd's sieve could have filtered out the pecans and inserted peanuts instead; thus producing a different virtual recipe.*

The sieve is the filter through which the ingredients in the recipe are delivered to the pie. The pie is like a document's display; the recipe is the analogue of the document's view; the ingredients are like the objects in the view. The display *(the pie)* of the virtual view *(the virtual recipe)* may exclude some of the objects *(the ingredients)* in the actual view *(the actual recipe).*

In Boxer, sieves are used by commands that modify a box (eg. *recolor*) or remove it from the display.

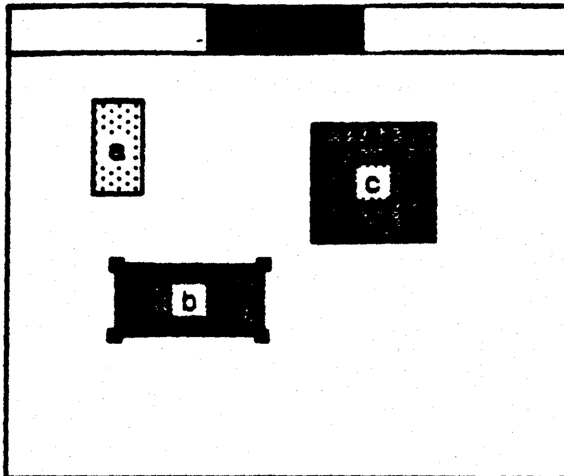
When Ferd is satisfied with his virtual pie, he commits the change to the recipe by replacing the contents of the pecan compartment (either with nothing or with peanuts).

Filtered *Duplicate* Command

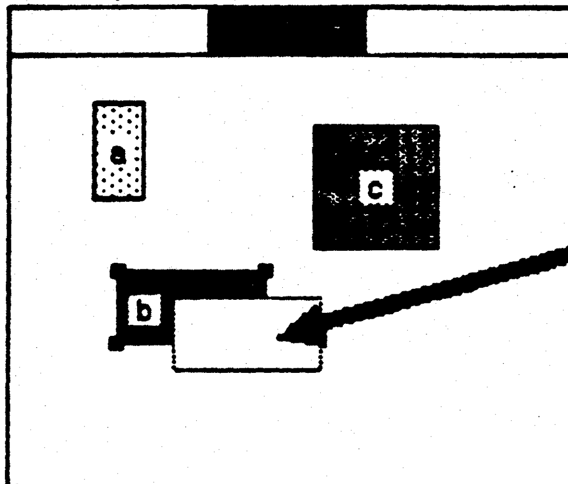
Filtered command is generated to duplicate box b.

Actual View

(Original)

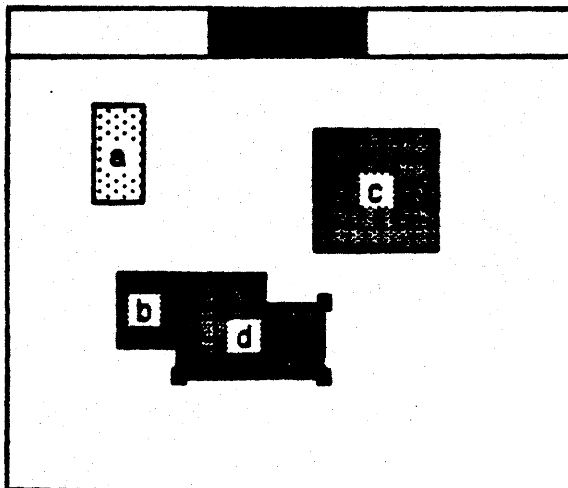


(duplicate box not yet part of actual view)



When done or redone the filtered command yields this virtual view

Virtual View



When the command is undone the actual view is displayed

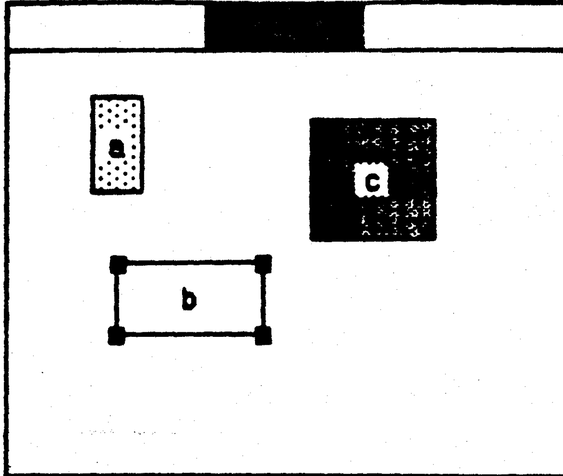
When the command is committed the virtual view becomes the new actual view.

Filtered *Recolor* Command

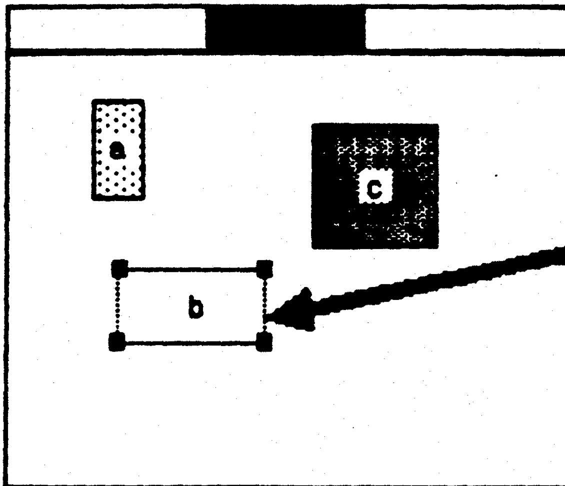
Filtered command is generated to recolor box b black

Actual View

(Original)



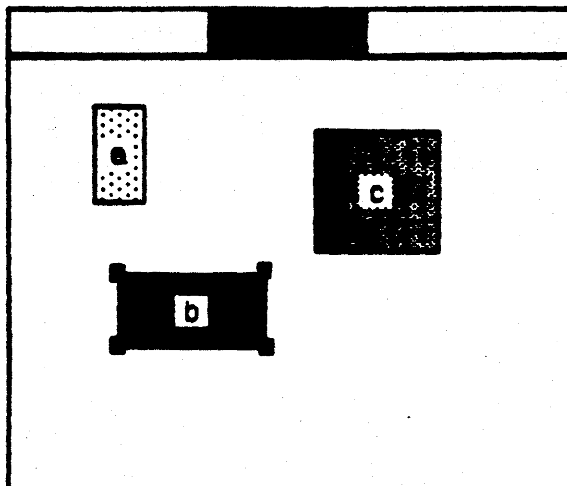
(the new color is not yet part of the actual view)



This box is black

When done or redone the filtered command yields this virtual view

Virtual View



When the command is undone the actual view is displayed

When the command is committed the virtual view becomes the new actual view.

Virtual and Actual Parts

Any document can be broken down into its component parts. The document can be enumerated by listing all of those parts. But a distinction must be made between the parts actually in the document and those added, modified, or removed by the filter.

The parts actually in the document are called *actual parts*. Those passed through by the filter are called *virtual parts*. Virtual parts become actual parts when they are committed to the document.

In Ferd Berfel's pie making machine the ingredients are components of the recipe. Those that are currently in the compartments are the actual ingredients Those that end up in the pie due to the filtering action of the sieve are the virtual ingredients. When Ferd replaces the contents of the blocked compartment with the ingredient (or lack thereof) supplied by the sieve, he is committing the change. He no longer needs the sieve.

filtering in applications

Once an application has been structured to support filtering, any application—defined command can use filters, though none is required to.

Filtering is something your application never needs to worry about. The Generic Application makes sure that filtering only occurs during the *doPhase* and the *redoPhase*. The filter is turned off during the *undoPhase*. When the command is committed, the changes induced by the filter are made permanent in the document.

Note: Many editing commands are more easily implemented without filters. It is not the purpose of this segment to suggest that all commands should be use filters. Just keep in mind that for many commands, it is easier and more efficient to use them.

STRUCTURING APPLICATIONS TO USE FILTERS

Filtering can occur whenever the document, typically its view (or window), is enumerating its parts. Drawing is a prime example of this. In Boxer, for example, the application enumerates the objects in the document by stepping through the view's boxList. To draw, the view has each of its objects draw themselves. Drawing can be easily restructured for filtering.

Up until now our views have always implictly drawn their actual parts. To support filters, ways must be explicitly established to draw both the actual parts of the view and its virtual parts.

the actual parts of the view

Defining an **EachActualPart** method to enumerate the actual parts of the view is the first step. The sample boxView method below enumerates the parts (the boxes) of the view:

```
PROCEDURE {TBoxView.} EachActualPart((PROCEDURE DoToObject(obj: TObject)));
BEGIN
        {SELF refers to the boxView}
```

```
        {The following line supplies each box in the boxList as the sole parameter to
         the procedure -  DoToObject}
    SELF.boxList.Each(DoToObject);
END;
```

{view.}EachActualPart performs some action upon each of the enumerated objects. The complementary method, {view.}EachVirtualPart, is the entry to the filter mechanism. *This method, defined by the ToolKit, is rarely redefined by the application.*

filters at the command level

It is only natural that the command does the actual filtering. The second step in supporting filters is defining the method of filtering for each command.

Each command class that implements filters must define one of two methods: EachVirtualPart or FilterAndDo.

{command.}EachVirtualPart is most often used to implement transparency–type filters. Code for the *duplicate* command's EachVirtualPart method is listed below:

```
    {This method adds the duplicate box as a virtual part.
     The duplicate box is refered to by SELF.newBox}
PROCEDURE {TDuplicateCmd.}EachVirtualPart{(PROCEDURE DoToObject(filteredObj: TObject))};
BEGIN
    {This is a transparency-type filter}
        {First perform the action, DoToObject, upon the actual parts of the view}
    SELF.image.EachActualPart(DoToObject);

        {Next, perform the action on the part to be added}
    DoToObject(SELF.newBox);
END;
```

{command.}FilterAndDo is typically used to implement sieve–type filters. Code for the *recolor* command's FilterAndDo method is listed below:

```
    {This method recolors the selected box.
     The selected box is referred to by SELF.box}
PROCEDURE {TRecolorCmd.}FilterAndDo{(actualObj: TObject;
                                     PROCEDURE DoToObject(filteredObj: TObject))};
VAR  tempColor: TColor;
     box: TBox;
BEGIN
    {This is a sieve-type filter}
    box := TBox(actualObj);   {coerce the actual part to be a box}
        {Check the box to see if it is the one to be modified (or filtered out)}
```

```
        {If it is the box to be modified, then substitute a new color}
    IF box = SELF.box THEN
        BEGIN
        tempColor := SELF.box.color;    {save the box's original actual color}
        SELF.box.color := SELF.color;   {replace the box's color, with the virtual color}
        DoToObject(SELF.box);           {draw the box with the virtual color}
        SELF.box.color := tempColor;    {restore the box's actual color}
        END
        {Otherwise pass the box through unchanged}
    ELSE
        DoToObject(box);
END;
```

Code for a hypothetical *delete* command's FilterAndDo method is listed below:

```
   {This method deletes the selected box}
PROCEDURE {TDeleteCmd.}FilterAndDo((actualObj: TObject;
                                   PROCEDURE DoToObject(filteredObj: TObject)));
VAR  box: TBox;
BEGIN
   {This is a sieve-type filter}
   box := TBox(actualObj);   {coerce the actual part to be a box}
       {Check the box to see if it is the one to be filtered out (or modified)}
       {If it is the box to be filtered out, then do not pass it through}
  IF box <> SELF.box THEN
     DoToObject(box);
END;
```

ACCESSING A VIRTUAL DOCUMENT

Accessing a virtual document is very different from accessing a document's actual parts. The main reason is that the view has no idea what the virtual parts are. Only the current filtering command knows which parts are in the virtual document.

Take drawing boxes for example. The code below only accesses the actual boxes in a Boxer document:

```
   {version of Draw that draws the actual parts only}
PROCEDURE {TBoxView.}Draw;
   PROCEDURE DrawBox(obj: TObject);
   BEGIN
      TBox(obj).Draw;
   END;
BEGIN
```

```
    SELF.boxList.Each(DrawBox);  {equivalent to:  SELF.EachActualPart(DrawBox) }
  END;
```

Whereas the following code is needed to access the boxes in a virtual Boxer document:

```
    {version of Draw that draws the virtual parts}
  PROCEDURE {TBoxView.}Draw;
    PROCEDURE DrawBox(obj: TObject);
    BEGIN
      TBox(obj).Draw;
    END;
  BEGIN
    SELF.EachVirtualPart(DrawBox);
  END;
```

Note: When filtering is not active the virtual document is the same as the actual document.

If your application only needs to operate upon a specific object in the virtual document, there is a special method for that — **FilterAndDo**.

{view}.**FilterAndDo** is a single object filter routine. It performs some action on a specified object once it has been passed through the active filter.

Whenever you want to access the virtual document, you must now call **EachVirtualPart** or **FilterAndDo** and pass in a DoToObject procedure parameter. The key points to remember with procedure parameters are:

1. The procedure passed usually is local to the method that initially passes it.
2. The parameter inside that procedure is of type TObject and must be coerced to be used.

<u>**filtering flow of control**</u>

EachVirtualPart passes the document's parts through the current filter, according to the following flow of control:

	view.**EachVirtualPart**	∨ DrawProc	{TImage.}
	window.**FilterDispatch**	∨ DrawProc	{TWindow.}
->	command.**EachVirtualPart**	∨ DrawProc	{*}

If the current command has overridden **EachVirtualPart** (providing transparency—type filtering), then the flow of control stops here. Otherwise it proceeds as follows:

command.**EachVirtualPart**	∨ DrawProc	{TCommand.}

```
{ FilteredDrawProc:  SELF.FilterAndDo(actualObj, DrawProc) }
```

```
            view.EachActualPart      ∨ FilteredDrawProc     {TBoxView.}
             boxList.Each           ∨ FilteredDrawProc     {TLinkList.}
                {for each box in the box list}
->               command.FilterAndDo ∨ box  ∨ DrawProc      {*}
```

This results in sieve—type filtering if the current command has overridden FilterAndDo. Otherwise the actual parts are passed, unchanged, to DoToObj.

If the current filter is not active (the phase is *undoPhase*), the flow of control is as follows:

```
        view.EachVirtualPart          ∨ DrawProc          {TImage.}
          window.FilterDispatch       ∨ DrawProc          {TWindow.}
            view.EachActualPart       ∨ DrawProc          {TBoxView.}
```

summary of virtual document access methods
(window and view)

The virtual document access methods in the view and the ones in the window are equivalent. Two sets are provided for convienence. Use the methods of the view when your view controls your data, and the methods of window when your window controls the data.

These methods are summarized below:

```
PROCEDURE {TWindow.}EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); DEFAULT;
PROCEDURE {TImage.}EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); DEFAULT;
```

EachActualPart enumerates the actual parts of the document, passing each part to the procedure DoToObject. This must be overridden by the application.

```
PROCEDURE {TWindow.}EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); DEFAULT;
PROCEDURE {TImage.}EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); DEFAULT;
```

EachVirtualPart, via the active filter, enumerates all the virtual parts of the document, passing each part to DoToObject. When no filter is active, only the actual parts are passed to DoToObject. This method should not be reimplemented.

```
PROCEDURE {TWindow.}FilterAndDo(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject));
PROCEDURE {TImage.}FilterAndDo(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject));
```

FilterAndDo should also never be reimplemented. It takes a part and the DoToObject procedure parameter and passes them to any active filter. If no filter is active, it passes the part directly to DoToObject.

Outside of specifying an **EachActualPart** method for the view, the bulk of the implementation rests with the command classes.

command filter methods

In addition to implementing two new methods, **FilterAndDo** and **EachVirtualPart**, we need to modify the function of **Perform** and **Commit**.

```
PROCEDURE {TCommand.} EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
```

EachVirtualPart passes each actual part of the view (or window) to DoToObject; then passes any parts added by the command (the virtual parts) to DoToObject as well.

```
PROCEDURE {TCommand.} FilterAndDo(actualObj: TObject;
                                  PROCEDURE DoToObject(filteredObj: TObject));
```

FilterAndDo passes only those actual parts of the view (or window) to DoToObject that are not affected by the command. The part or parts that are operated on by the command, may either be filtered out or modified.

```
PROCEDURE {TCommand.} Perform(cmdPhase: TCmdPhase);  DEFAULT;
```

Perform typically does very little now. On each of the command phases any document state information that was not saved by the CREATE must be saved; and the screen area that needs to be updated should be invalidated.

```
PROCEDURE {TCommand.} Commit;
```

Commit performs the actual changes to the document.

IMPLEMENTATION

We modify the previous stage of Boxer, 6Boxer, to implement filtered *recolor, duplicate* and *clear all* commands.

user interface

The user interface for this stage of Boxer, 7Boxer, has changed in only one respect — the *clear all* command is now undoable.

Implementation of Filtered *Duplicate*

To modify the implementation of *duplicate* to support filtering, we need to add or modify the following methods:

{TDuplicateCmd.}CREATE

CREATE must commit the last command before duplicating. This, for example, insures that the duplicate of a box which has just been recolored will have that box's new color and not its old color.

{TDuplicateCmd.}**Perform**

Perform is now very simple. The *doPhase* and *redoPhase* are equivalent; and merely reset the selection to the duplicate. The *undoPhase* resets the selection to the original box. All phases invalidate the newly selected box.

{TDuplicateCmd.}**EachVirtualPart**

EachVirtualPart passes to DoToObject all the actual boxes of the document. It does this by calling: view.**EachAcutalPart**(DoToObject). Next it passes the duplicate box by calling: DoToObject(SELF.**newBox**).

{TDuplicateCmd.}**Commit**

Commit now has the honor of inserting the duplicate box into the view's boxList.

Since duplicate does not change any actual box, we do not need to reimplement **FilterAndDo**.

Implementation of Filtered *Recolor*

To modify the implementation of *recolor* to support filtering, we need to add or modify the following methods:

{TRecolorCmd.}**Perform**

Perform simply invalidates the selected box to force a redraw.

{TRecolorCmd.}**FilterAndDo**

If the box passed to **FilterAndDo** is the selected box, its color is changed before being passed to DoToObject, then restored when DoToObject completes.

{TRecolorCmd.}**Commit**

Commit now has the honor of changing the box's color.

Since *recolor* does not add any boxes, we do not need to implement **EachVirtualPart**.

Implementation of Filtered *Clear All*

To modify the implementation of *clear all* to be undoable and support filtering, we need to add or modify the following methods:

{TClearAll.}CREATE

Creates a *clear all* command object. Sets its **kind** field to the kind of the current selection.

{TClearAll.}Perform

Resets the selection's kind to nothingKind on the *doPhase* and *redoPhase*. Resets it to SELF.**kind** on the *undoPhase*. All phases invalidate the panel.

{TClearAll.}EachVirtualPart

EachVirtualPart does nothing; since no boxes are in the virtual document.

{TClearAll.}Commit

Commit deletes all the boxes in the view.

Since *clear all* does not modify any actual box, we do not need to reimplement FilterAndDo. *Note: Clear All does delete actual boxes, but it is a blanket operation rather than one applied to selected boxes.*

Implementation Summary

The actual implementation for 7Boxer is summarized below. The code for 6Boxer is used as the base for all changes.

New Classes

```
[TClearAllCmd]
  FUNCTION  {TClearAllCmd.}CREATE(object: TObject; itsHeap: THeap;
                                  itsCmdNumber: TCmdNumber; itsView: TBoxView)
                                  : TClearAllCmd;

  PROCEDURE {TClearAllCmd.}Commit; OVERRIDE;

  PROCEDURE {TClearAllCmd.}Perform; OVERRIDE;
  PROCEDURE {TClearAllCmd.}EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
```

New Methods (for existing classes)

```
[TBoxView]
  PROCEDURE {TBoxView.}EachActualPart(PROCEDURE DoToObject(filteredObj: TObject));
```

```
                                OVERRIDE;
```

{TBoxView.}**EachActualPart** passes each box in the view's boxList to DoToObject. It calls: SELF.**boxList.Each**(DoToObject).

[TDuplicateCmd]

```
PROCEDURE {TDuplicateCmd.}EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
                                OVERRIDE;
PROCEDURE {TDuplicateCmd.}Commit; OVERRIDE;
```

[TRecolorCmd]

```
PROCEDURE {TRecolorCmd.}FilterAndDo(actualObj: TObject;
                                PROCEDURE DoToObject(filteredObj: TObject));
                                OVERRIDE;
PROCEDURE {TRecolorCmd.}Commit; OVERRIDE;
```

Modified Methods

[TBoxWindow]

```
PROCEDURE {TBoxWindow.}NewCommand; OVERRIDE;
```

{TBoxWindow.}**NewCommand** now creates a *clear all* command in response to a *clear all* menu event.

[TDuplicateCmd]

```
PROCEDURE {TDuplicateCmd.}Perform(cmdPhase: TCmdPhase); OVERRIDE;
```

[TRecolorCmd]

```
PROCEDURE {TRecolorCmd.}Perform(cmdPhase: TCmdPhase); OVERRIDE;
```

Deleted Method

[TBoxWindow]

```
PROCEDURE {TBoxWindow.}ClearAll;
```

Code Sample
for this
Segment

```
27
SLOT2CHAN1
; no assembler files
$
; no building blocks
$
; no links
$
y
y
n
BoxNum7
```

```
1
3
2500
$-#BOOT-TK/PABC

; Apple building block phrase files can be included here

1000
LisaBoxer

;     PROCEDURE {TBoxWindow.}NewCommand
;     cantUndo         = 1001;
1001 caution cancel alert
You will not be able to undo ClearAll.

0

1
$-#BOOT-TK/PABC~File/Print

2
Edit
Undo Last Change#205
-
Duplicate/D#1011
-
Clear All/Z#1012
-

3
Shades
White#1006
Light Gray#1007
Gray#1008
Dark Gray#1009
Black#1010

5
$-#BOOT-TK/PABC~Page Layout

99
$-#BOOT-TK/PABC~Debug

100
$-#BOOT-TK/PABC~Buzzwords
Create Box#2000
Move Selection#2001

0
```

```
PROGRAM M7Boxer;

USES
    {$U UObject      } UObject,

  {$IFC libraryVersion <= 20}
    {$U UFont}                  UFont,
  {$ENDC}

    {$U QuickDraw    } QuickDraw,
    {$U UDraw        } UDraw,
    {$U UABC         } UABC,

    {$U U7Boxer      } U7Boxer;

CONST

    phraseVersion = 1;

BEGIN

    process := TBoxProcess.CREATE;
    process.Commence(phraseVersion);
    process.Run;
    process.Complete(TRUE);

END.
```

```
1     1 --        { This segment of the LisaBoxer sample program implements Filtered Undo}
1     2 --        { Copyright 1983, Apple Computer Inc.}
1     3 --    {$E ERRORS.TEXT}
1     4 --
1     5 --    UNIT U7Boxer;
1     6 --
1     7 --    INTERFACE
1     8 --
1     9 --    USES
1    10 --        {$U UObject}              UObject,
1    11 --
1    12 --      {$IFC libraryVersion <= 20}
1    13 --        {$U UFont}                UFont,
1    14 --      {$ENDC}
1    15 --
1    16 --        {$U QuickDraw}            QuickDraw,
1    17 --        {$U UDraw}                UDraw,
1    18 --        {$U UABC}                 UABC;
1    19 --
1    20 --    CONST
1    21 --        colorWhite = 1;
1    22 --        colorLtGray = 2;
1    23 --        colorGray = 3;
1    24 --        colorDkGray = 4;
1    25 --        colorBlack = 5;
1    26 --
1    27 --       { selection kinds }
1    28 --        boxSelectionKind = 1;
1    29 --        createBoxSelectionKind = 2;
1    30 --
1    31 --       { Menus }
1    32 --        uWhite = 1006;
1    33 --        uLtGray = 1007;
1    34 --        uGray = 1008;
1    35 --        uDkGray = 1009;
1    36 --        uBlack = 1010;
1    37 --        uDuplicate = 1011;
1    38 --        uClearAll = 1012;
1    39 --
1    40 --    TYPE
1    41 --
1    42 --        TColor = colorWhite..colorBlack; {color of a box}
1    43 --
1    44 --               {New Classes for this Application}
1    45 --
1    46 --        TBox = SUBCLASS OF TObject
1    47 --
1    48 --            {Variables}
1    49 --            shapeLRect:       LRect;
1    50 --            color:            TColor;
1    51 --
1    52 --          {Creation/Destruction}
1    53 --            FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
1    54 --
1    55 --          { Highlighting support }
1    56 --            PROCEDURE TBox.PaintHandles;
1    57 --
1    58 --          { Framing while creating }
1    59 --            PROCEDURE TBox.DrawFrame;
1    60 --
1    61 --          {Display}
1    62 --            PROCEDURE TBox.Draw;
1    63 --            END;
1    64 --
1    65 --
1    66 --        TBoxView = SUBCLASS OF TView
1    67 --
1    68 --          {Variables}
1    69 --            boxList:          TList;
1    70 --
1    71 --          {Creation/Destruction}
1    72 --            FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
1    73 --                                      : TBoxView;
1    74 --
1    75 --            FUNCTION  TBoxView.BoxWith(LPt: LPoint): TBox;
1    76 --
1    77 --          {Invalidation}
1    78 --            PROCEDURE TBoxView.InvalBox(invalLRect: LRect);
1    79 --
1    80 --            PROCEDURE TBoxView.MousePress(mouseLPt: LPoint); OVERRIDE;
1    81 --
1    82 --          {Display}
1    83 --            PROCEDURE TBoxView.Draw; OVERRIDE;
1    84 --
1    85 --          {Filtering}
1    86 --           PROCEDURE  TBoxView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1    87 --
1    88 --          {Initialization}
1    89 --            PROCEDURE TBoxView.InitBoxList(itsHeap: THeap);
1    90 --            FUNCTION  TBoxView.NoSelection: TSelection; OVERRIDE;
1    91 --           END;
1    92 --
1    93 --
1    94 --        TBoxSelection = SUBCLASS OF TSelection
1    95 --
1    96 --          {Variables}
1    97 --            box:  TBox;
1    98 --
1    99 --          {Creation/Destruction}
1   100 --            FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsBox: TBox;
1   101 --                                              itsKind: INTEGER; itsAnchorLPt: LPoint): TBoxSelection;
1   102 --
1   103 --          {Drawing - per pad}
1   104 --            PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit); OVERRIDE;
1   105 --
1   106 --          {Selection - per pad}
1   107 --            PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1   108 --            PROCEDURE TBoxSelection.MouseRelease; OVERRIDE;
1   109 --
1   110 --          {Command Dispatch}
```

```
1  111 --            FUNCTION TBoxSelection.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  112 --            FUNCTION TBoxSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
1  113 --                                                  : BOOLEAN; OVERRIDE;
1  114 --        END;
1  115 --
1  116 --
1  117 --        TCreateBoxSelection = SUBCLASS OF TSelection
1  118 --
1  119 --          {Variables}
1  120 --            box:   TBox;
1  121 --
1  122 --          {Creation/Destruction}
1  123 --            FUNCTION  TCreateBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView;
1  124 --                                         itsAnchorLPt: LPoint): TCreateBoxSelection;
1  125 --
1  126 --          {Selection - per pad}
1  127 --            PROCEDURE TCreateBoxSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  128 --            PROCEDURE TCreateBoxSelection.MouseRelease; OVERRIDE;
1  129 --        END;
1  130 --
1  131 --
1  132 --                  { This command recolors the selected box and is undoable}
1  133 --        TRecolorCmd = SUBCLASS OF TCommand
1  134 --            Box:     TBox;
1  135 --            color:   TColor;
1  136 --
1  137 --          {Creation}
1  138 --            FUNCTION  TRecolorCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  139 --                                  itsView: TBoxView; itsBox: TBox; itsColor: TColor): TRecolorCmd;
1  140 --
1  141 --          {Command Execution}
1  142 --            PROCEDURE TRecolorCmd.Commit; OVERRIDE;
1  143 --            PROCEDURE TRecolorCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  144 --            PROCEDURE TRecolorCmd.FilterAndDo(actualObj: TObject;
1  145 --                                    PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  146 --            END;
1  147 --
1  148 --
1  149 --                    { This command duplicates the selected box and is undoable }
1  150 --        TDuplicateCmd = SUBCLASS OF TCommand
1  151 --            oldBox, newBox:     TBox;
1  152 --
1  153 --          {Creation}
1  154 --            FUNCTION  TDuplicateCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  155 --                                     itsView: TboxView; itsBox: TBox): TDuplicateCmd;
1  156 --
1  157 --          {Command Execution}
1  158 --            PROCEDURE TDuplicateCmd.Commit; OVERRIDE;
1  159 --            PROCEDURE TDuplicateCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  160 --            PROCEDURE TDuplicateCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  161 --            END;
1  162 --
1  163 --
1  164 --                    { This command clears the view and is undoable }
1  165 --        TClearAllCmd = SUBCLASS OF TCommand
1  166 --          {Variables}
1  167 --            kind:   INTEGER;
1  168 --
1  169 --          {Creation}
1  170 --            FUNCTION  TClearAllCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  171 --                                  itsView: TboxView): TClearAllCmd;
1  172 --
1  173 --          {Command Execution}
1  174 --            PROCEDURE TClearAllCmd.Commit; OVERRIDE;
1  175 --            PROCEDURE TClearAllCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  176 --            PROCEDURE TClearAllCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  177 --            END;
1  178 --
1  179 --
1  180 --        TBoxProcess = SUBCLASS OF TProcess
1  181 --
1  182 --          {Creation/Destruction}
1  183 --            FUNCTION  TBoxProcess.CREATE: TBoxProcess;
1  184 --            FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
1  185 --                                          : TDocManager; OVERRIDE;
1  186 --            END;
1  187 --
1  188 --
1  189 --        TBoxDocManager = SUBCLASS OF TDocManager
1  190 --
1  191 --          {Creation/Destruction}
1  192 --            FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
1  193 --                                     : TBoxDocManager;
1  194 --            FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID): TWindow; OVERRIDE;
1  195 --            END;
1  196 --
1  197 --
1  198 --        TBoxWindow = SUBCLASS OF TWindow
1  199 --
1  200 --          {Creation/Destruction}
1  201 --            FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
1  202 --
1  203 --          {Document Creation}
1  204 --            PROCEDURE TBoxWindow.BlankStationery; OVERRIDE;
1  205 --
1  206 --          {Commands}
1  207 --            FUNCTION  TBoxWindow.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  208 --            FUNCTION  TBoxWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN; OVERRIDE;
1  209 --            END;
1  210 --
1  211 --
1  212 --
1  213 --    IMPLEMENTATION
1  214 --
1  215 --      {$I U7Boxer2.text}
2    1 --   {U7BOXER2}
2    2 --
2    3 --   METHODS OF TBox;
2    4 --
2    5 -- A      FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
```

```
2    6 0- A     BEGIN
2    7 --           {$IFC fTrace}BP(11); {$ENDC}
2    8 --           SELF := NewObject(itsHeap, THISCLASS);
2    9 --           WITH SELF DO
2   10 1-               BEGIN
2   11 --               shapeLRect := zeroLRect;
2   12 --               color := colorGray;
2   13 -1               END;
2   14 --           {$IFC fTrace}EP; {$ENDC}
2   15 -0 A     END;
2   16 --
2   17 --      {$IFC fDebugMethods}
2   18 -- A     PROCEDURE TBox.Fields(PROCEDURE Field(nameAndType: S255));
2   19 0- A     BEGIN
2   20 --           Field('shapeLRect: LRect');
2   21 --           Field('color: INTEGER');
2   22 --           Field('');
2   23 -0 A     END;
2   24 --       {$ENDC}
2   25 --
2   26 --
2   27 --        {This draws a particular box}
2   28 -- A     PROCEDURE TBox.Draw;
2   29 --       VAR lPat:    LPattern;
2   30 0- A     BEGIN
2   31 --           {$IFC fTrace}BP(10); {$ENDC}
2   32 --           PenNormal;
2   33 --
2   34 --           IF LRectIsVisible(SELF.shapeLRect) THEN {this box needs to be drawn}
2   35 1-               BEGIN
2   36 --             {Get a Quickdraw pattern to represent the box's color}
2   37 2-               CASE SELF.color OF
2   38 --                   colorWhite:       lPat := lPatWhite;
2   39 --                   colorLtGray:      lPat := lPatLtGray;
2   40 --                   colorGray:        lPat := lPatGray;
2   41 --                   colorDkGray:      lPat := lPatDkGray;
2   42 --                   colorBlack:       lPat := lPatBlack;
2   43 --                   OTHERWISE         lPat := lPatWhite;  {this case should not happen}
2   44 -2                   END;
2   45 --
2   46 --              {Fill the box with the pattern, and draw a frame around it}
2   47 --               FillLRect(SELF.shapeLRect, lPat);
2   48 --               FrameLRect(SELF.shapeLRect);
2   49 -1               END;
2   50 --           {$IFC fTrace}EP; {$ENDC}
2   51 -0 A     END;
2   52 --
2   53 --        { Frame a particular box}
2   54 -- A     PROCEDURE TBox.DrawFrame;
2   55 0- A     BEGIN
2   56 --           {$IFC fTrace}BP(10); {$ENDC}
2   57 --           PenNormal;
2   58 --           PenMode(PatXOr);
2   59 --           FrameLRect(SELF.shapeLRect);
2   60 --           {$IFC fTrace}EP; {$ENDC}
2   61 -0 A     END;
2   62 --
2   63 --         {This calls the DoToHandle Procedure once for each handle LRect; user of this method must
2   64 --           set up the pen pattern and mode before calling}
2   65 -- A     PROCEDURE TBox.PaintHandles;
2   66 --       VAR hLRect,
2   67 --           shapeLRect:  LRect;
2   68 --           dh, dv:      LONGINT;
2   69 --
2   70 -- B         PROCEDURE MoveHandleAndPaint(hOffset, vOffset: LONGINT);
2   71 0- B         BEGIN
2   72 --               OffsetLRect(hLRect, hOffset, vOffset);
2   73 --               PaintLRect(hLRect);
2   74 -0 B         END;
2   75 --
2   76 0- A     BEGIN
2   77 --           {$IFC fTrace}BP(10); {$ENDC}
2   78 --           SetLRect(hLRect, -3, -2, 3, 2);
2   79 --           shapeLRect := SELF.shapeLRect;
2   80 --           WITH shapeLRect DO
2   81 1-               BEGIN
2   82 --               dh := right - left;
2   83 --               dv := bottom - top;
2   84 --               MoveHandleAndPaint(left, top);     {draw top left handle}
2   85 -1               END;
2   86 --           MoveHandleAndPaint(dh, 0);           {then top right}
2   87 --           MoveHandleAndPaint(0, dv);           {then bottom right}
2   88 --           MoveHandleAndPaint(-dh, 0);          {finally bottom left}
2   89 --           {$IFC fTrace}EP; {$ENDC}
2   90 -0 A     END;
2   91 --
2   92 --   END;
2   93 --
2   94 --
2   95 --   METHODS OF TBoxView;
2   96 --
2   97 -- A     FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
2   98 --                                 : TBoxView;
2   99 0- A     BEGIN
2  100 --           {$IFC fTrace}BP(11); {$ENDC}
2  101 --           IF object = NIL THEN
2  102 --               object := NewObject(itsHeap, THISCLASS);
2  103 --            SELF := TBoxView(itsPanel.NewView(object, itsExtent, TPrintManager.CREATE(NIL, itsHeap),
2  104 --                                           stdMargins, TRUE));
2  105 --           {$IFC fTrace}EP; {$ENDC}
2  106 -0 A     END;
2  107 --
2  108 --
2  109 --   {$IFC fDebugMethods}
2  110 -- A     PROCEDURE TBoxView.Fields(PROCEDURE Field(nameAndType: S255));
2  111 0- A     BEGIN
2  112 --           TView.Fields(Field);
2  113 --           Field('boxList: TList');
2  114 -0 A     END;
2  115 --   {$ENDC}
```

```
2  116 --
2  117 --
2  118 --           {This returns the box containing a certain point}
2  119 -- A     FUNCTION  TBoxView.BoxWith(LPt: LPoint): TBox;
2  120 -- B         PROCEDURE FindBox(obj: TObject);
2  121 --            VAR box:     TBox;
2  122 0- B          BEGIN
2  123 --                box := TBox(obj);
2  124 --                IF LPtInLRect(LPt, box.shapeLRect) THEN
2  125 --                    BoxWith := box;              {last one found (front one) is returned}
2  126 -0 B          END;
2  127 0- A      BEGIN
2  128 --            {$IFC fTrace}BP(11);{$ENDC}
2  129 --            boxWith := NIL;
2  130 --            SELF.EachVirtualPart(FindBox);
2  131 --            {$IFC fTrace}EP;{$ENDC}
2  132 -0 A      END;
2  133 --
2  134 --
2  135 --            {This draws the list of boxes}
2  136 -- A      PROCEDURE TBoxView.Draw;
2  137 -- B          PROCEDURE DrawBox(obj: TObject);
2  138 --            VAR box:     TBox;
2  139 0- B          BEGIN
2  140 --                box := TBox(obj);
2  141 --                box.Draw;
2  142 -0 B          END;
2  143 0- A      BEGIN
2  144 --            {$IFC fTrace}BP(10);{$ENDC}
2  145 --            SELF.EachVirtualPart(DrawBox);
2  146 --            {$IFC fTrace}EP;{$ENDC}
2  147 -0 A      END;
2  148 --
2  149 --
2  150 -- A      PROCEDURE TBoxView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject));
2  151 0- A      BEGIN
2  152 --            {$IFC fTrace}BP(11);{$ENDC}
2  153 --            SELF.boxList.Each(DoToObject);
2  154 --            {$IFC fTrace}EP;{$ENDC}
2  155 -0 A      END;
2  156 --
2  157 --
2  158 --            {This determines which type of selection to create}
2  159 -- A       PROCEDURE TBoxView.MousePress(mouseLPt: LPoint);
2  160 --         VAR aSelection: TSelection;
2  161 --             panel:      TPanel;
2  162 --             box:        TBox;
2  163 --
2  164 0- A       BEGIN
2  165 --            {$IFC fTrace}BP(11);{$ENDC}
2  166 --            panel := SELF.panel;
2  167 --            panel.Highlight(panel.selection, hOntoOff);          {Turn off the old highlighting}
2  168 --            box := SELF.BoxWith(mouseLPt);        {Find the box the user clicked on}
2  169 --
2  170 --            IF box = NIL THEN
2  171 --                {Create an instance of TCreateBoxSelection}
2  172 --                aSelection := panel.selection.FreedAndReplacedBy(
2  173 --                                        TCreateBoxSelection.CREATE(NIL, SELF.heap, SELF, mouseLPt))
2  174 --            ELSE
2  175 --                {Create an instance of TBoxSelection}
2  176 --                aSelection := panel.selection.FreedAndReplacedBy(
2  177 --                              TBoxSelection.CREATE(NIL, SELF.heap, SELF, box, boxSelectionKind, mouseLPt));
2  178 --
2  179 --            panel.Highlight(panel.selection, hOffToOn);     {Turn on the highlighting for the newly selected box}
2  180 --
2  181 --            self.panel.selection.MarkChanged;    {Allow the document to be saved so that any changes made}
2  182 --                                                 {can become permanent}
2  183 --            {$IFC fTrace}EP;{$ENDC}
2  184 -0 A       END;
2  185 --
2  186 --
2  187 -- A      PROCEDURE  TBoxView.InvalBox(invalLRect: LRect);
2  188 0- A      BEGIN
2  189 --            {$IFC fTrace}BP(10);{$ENDC}
2  190 --            InsetLRect(invalLRect, -3, -2);
2  191 --            SELF.panel.InvalLRect(invalLRect);
2  192 --            {$IFC fTrace}EP;{$ENDC}
2  193 -0 A      END;
2  194 --
2  195 --
2  196 -- A      PROCEDURE TBoxView.InitBoxList (itsHeap: THeap);
2  197 --        VAR  boxList: TList;
2  198 0- A      BEGIN
2  199 --           {$IFC fTrace}BP(11);{$ENDC}
2  200 --            boxList := TList.CREATE(NIL, itsHeap, 0);
2  201 --            SELF.boxList := boxList;
2  202 --            {$IFC fTrace}EP;{$ENDC}
2  203 -0 A      END;
2  204 --
2  205 --
2  206 -- A      FUNCTION TBoxView.NoSelection: TSelection;
2  207 0- A      BEGIN
2  208 --           {$IFC fTrace}BP(11);{$ENDC}
2  209 --            NoSelection := TBoxSelection.CREATE(NIL, SELF.Heap, SELF, NIL, nothingKind, zeroLPt);
2  210 --            {$IFC fTrace}EP;{$ENDC}
2  211 -0 A      END;
2  212 --
2  213 --    END;
2  214 --
2  215 --
2  216 --
2  217 --    METHODS OF TBoxSelection;
2  218 --
2  219 -- A      FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsBox: TBox;
2  220 --                                   itsKind: INTEGER; itsAnchorLPt: LPoint): TBoxSelection;
2  221 0- A      BEGIN
2  222 --            {$IFC fTrace}BP(11);{$ENDC}
2  223 --            IF object = NIL THEN
2  224 --                object := NewObject(itsHeap, THISCLASS);
2  225 --            SELF := TBoxSelection(TSelection.CREATE(object, itsHeap, itsView, itsKind, itsAnchorLPt));
```

```
2  226 --
2  227 --           SELF.box := itsBox;
2  228 --           {$IFC fTrace}EP;{$ENDC}
2  229 -0 A      END;
2  230 --
2  231 --    {$IFC fDebugMethods}
2  232 -- A      PROCEDURE TBoxSelection.Fields(PROCEDURE Field(nameAndType: S255));
2  233 0- A      BEGIN
2  234 --           TSelection.Fields(Field);
2  235 --           Field('box: TBox');
2  236 -0 A      END;
2  237 --    {$ENDC}
2  238 --
2  239 --
2  240 --         {This draws the handles on the selected box}
2  241 -- A      PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit);
2  242 0- A      BEGIN
2  243 --           {$IFC fTrace}BP(11);{$ENDC}
2  244 --           IF SELF.kind <> nothingKind THEN
2  245 1-              BEGIN
2  246 --              thePad.SetPenToHighlight(highTransit);  {set the drawing mode according to desired highlighting}
2  247 --              SELF.box.PaintHandles;                  {draw the handles on the box}
2  248 -1              END;
2  249 --           {$IFC fTrace}EP;{$ENDC}
2  250 -0 A      END;
2  251 --
2  252 --
2  253 --      {This is called when the user moves the mouse after pressing the button}
2  254 -- A      PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint);
2  255 --      VAR diffLPt:     LPoint;
2  256 --           boxView:    TBoxView;
2  257 --           shapeLRect: LRect;
2  258 0- A      BEGIN
2  259 --           {$IFC fTrace}BP(11);{$ENDC}
2  260 --           boxView := TBoxView(SELF.view);
2  261 --
2  262 --              {How far did mouse move?}
2  263 --           LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
2  264 --
2  265 --              {Move it if delta is nonzero}
2  266 --           IF NOT EqualLPt(diffLPt, zeroLPt) THEN
2  267 1-              BEGIN
2  268 --              SELF.currLPt := mouseLPt;
2  269 --
2  270 --              shapeLRect := SELF.box.shapeLRect;
2  271 --                     {Compute old and new positions of box}
2  272 --              boxView.InvalBox(shapeLRect);
2  273 --              OffsetLRect(shapeLRect, diffLPt.h, diffLPt.v);
2  274 --              boxView.InvalBox(shapeLRect);
2  275 --
2  276 --              SELF.box.shapeLRect := shapeLRect;
2  277 -1              END;
2  278 --           {$IFC fTrace}EP;{$ENDC}
2  279 -0 A      END;
2  280 --
2  281 --
2  282 -- A      PROCEDURE TBoxSelection.MouseRelease;
2  283 0- A      BEGIN
2  284 --           {$IFC fTrace}BP(11);{$ENDC}
2  285 --             { If the mouse moved then commit any outstanding command }
2  286 --           IF NOT EqualLPt(SELF.currLPt, SELF.anchorLPt) THEN
2  287 --              SELF.window.CommitLast;
2  288 --           {$IFC fTrace}EP;{$ENDC}
2  289 -0 A      END;
2  290 --
2  291 --
2  292 -- A      FUNCTION  TBoxSelection.NewCommand(cmdNumber: TCmdNumber): TCommand;
2  293 --      VAR boxView:     TBoxView;
2  294 --           heap:       THeap;
2  295 0- A      BEGIN
2  296 --           {$IFC fTrace}BP(11);{$ENDC}
2  297 --
2  298 --           boxView := TBoxView(SELF.view);
2  299 --           heap := SELF.Heap;
2  300 --
2  301 1-           CASE cmdNumber OF
2  302 --              uWhite, uLtGray, uGray, uDkGray, uBlack:
2  303 --                  NewCommand := TRecolorCmd.CREATE(NIL, heap, cmdNumber, boxView, SELF.box,
2  304 --                                                  cmdNumber - uWhite + colorWhite);
2  305 --
2  306 --              uDuplicate:
2  307 --                  NewCommand := TDuplicateCmd.CREATE(NIL, heap, cmdNumber, boxView, SELF.box);
2  308 --
2  309 --              OTHERWISE
2  310 --                  NewCommand := SUPERSELF.NewCommand(cmdNumber);
2  311 -1              END;
2  312 --           {$IFC fTrace}EP;{$ENDC}
2  313 -0 A      END;
2  314 --
2  315 --
2  316 -- A      FUNCTION  TBoxSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
2  317 0- A      BEGIN
2  318 --           {$IFC fTrace}BP(11);{$ENDC}
2  319 1-           CASE cmdNumber OF
2  320 --              uWhite, uLtGray, uGray, uDkGray, uBlack,
2  321 --                    uDuplicate:
2  322 --                             CanDoCommand := SELF.kind <> nothingKind;
2  323 --
2  324 --              OTHERWISE
2  325 --                             CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
2  326 -1              END;
2  327 --           {$IFC fTrace}EP;{$ENDC}
2  328 -0 A      END;
2  329 --
2  330 --    END;
2  331 --
2  332 --
2  333 --
2  334 --    METHODS OF TCreateBoxSelection;
2  335 --
```

```
2  336 -- A     FUNCTION  TCreateBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView;
2  337 --                                                itsAnchorLPt: LPoint): TCreateBoxSelection;
2  338 --       VAR box: TBox;
2  339 0- A     BEGIN
2  340 --           {$IFC fTrace}BP(11);{$ENDC}
2  341 --           IF object = NIL THEN
2  342 --               object := NewObject(itsHeap, THISCLASS);
2  343 --           SELF := TCreateBoxSelection(TSelection.CREATE(object, itsHeap, itsView, createBoxSelectionKind,
2  344 --                                                              itsAnchorLPt));
2  345 --           box := TBox.CREATE(NIL, SELF.heap);
2  346 --           SELF.box := box;
2  347 --           {$IFC fTrace}EP;{$ENDC}
2  348 -0 A     END;
2  349 --
2  350 --
2  351 --   {$IFC fDebugMethods}
2  352 -- A     PROCEDURE TCreateBoxSelection.Fields(PROCEDURE Field(nameAndType: S255));
2  353 0- A     BEGIN
2  354 --           TSelection.Fields(Field);
2  355 --           Field('box: TBox');
2  356 -0 A     END;
2  357 --    {$ENDC}
2  358 --
2  359 --
2  360 --       {This is called when the user moves the mouse after pressing the button}
2  361 -- A     PROCEDURE TCreateBoxSelection.MouseMove(mouseLPt: LPoint);
2  362 --       VAR maxBoxLRect:    LRect;
2  363 --           diffLPt:        LPoint;
2  364 --           boxView:        TBoxView;
2  365 --           box:            TBox;
2  366 --
2  367 -- B         PROCEDURE DrawTheFrame;
2  368 0- B         BEGIN
2  369 --               box.DrawFrame;
2  370 -0 B         END;
2  371 --
2  372 0- A     BEGIN
2  373 --           {$IFC fTrace}BP(11);{$ENDC}
2  374 --           boxView := TBoxView(SELF.view);
2  375 --           box := SELF.box;
2  376 --
2  377 --           { In Boxer it is possible to draw a box greater than allowed by a 16 bit rectangle. These three
2  378 --             lines force the rectangle to within 16 bits. }
2  379 --     {$H-} WITH SELF.anchorLPt DO
2  380 --               SetLRect(maxBoxLRect, h+10-MAXINT, v+10-MAXINT, h+MAXINT-10, v+MAXINT-10);
2  381 --     {$H+} LRectHaveLPt(maxBoxLRect, mouseLPt);
2  382 --
2  383 --           LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
2  384 --           IF NOT EqualLPt(diffLPt, zeroLPt) THEN
2  385 1-               BEGIN
2  386 --               SELF.currLPt := mouseLPt;
2  387 --
2  388 --               boxView.panel.OnAllPadsDo(DrawTheFrame);
2  389 --               WITH box DO
2  390 2-                   BEGIN
2  391 --                   shapeLRect.topLeft := SELF.anchorLPt;
2  392 --                   shapeLRect.botRight := mouseLPt;
2  393 -2                   END;
2  394 --
2  395 --       {$H-}   RectifyLRect(box.shapeLRect);    {$H+}
2  396 --               boxView.panel.OnAllPadsDo(DrawTheFrame);
2  397 -1               END;
2  398 --           {$IFC fTrace}EP;{$ENDC}
2  399 -0 A     END;
2  400 --
2  401 --
2  402 -- A     PROCEDURE TCreateBoxSelection.MouseRelease;
2  403 --       VAR thisBox:        TBox;
2  404 --           boxView:        TBoxView;
2  405 --           drawnLRect:     LRect;
2  406 --           aSelection:     TSelection;
2  407 --           panel:          TPanel;
2  408 --
2  409 -- B         PROCEDURE DrawTheFrame;
2  410 0- B         BEGIN
2  411 --               thisBox.DrawFrame;
2  412 -0 B         END;
2  413 --
2  414 0- A     BEGIN
2  415 --           {$IFC fTrace}BP(11);{$ENDC}
2  416 --           boxView := TBoxView(SELF.view);
2  417 --           panel := boxView.panel;
2  418 --           thisBox := SELF.box;
2  419 --           panel.OnAllPadsDo(DrawTheFrame);
2  420 --           drawnLRect := thisBox.shapeLRect;
2  421 --
2  422 --           { Independant of whether we threw the boxed away or not we must create an instance of TBoxSelection
2  423 --             to replace the now useless instance of TCreateBoxSelection using the kind set above. }
2  424 --           aSelection := SELF.FreedAndReplacedby(
2  425 --                           TBoxSelection.CREATE(NIL, SELF.heap, boxView, thisBox, boxSelectionKind,
2  426 --                                                drawnLRect.topLeft));
2  427 --
2  428 --           boxView.InvalBox(drawnLRect);
2  429 --
2  430 --           {If the box is not big enough then throw it away, otherwise put it in the list}
2  431 --           IF (drawnLRect.right - drawnLRect.left <=4) OR (drawnLRect.bottom - drawnLRect.top <=4) THEN
2  432 1-               BEGIN
2  433 --               aSelection.kind := nothingKind;
2  434 --               thisBox.Free;
2  435 -1               END
2  436 1-           ELSE BEGIN
2  437 --               { Commit any outstanding command }
2  438 --               SELF.window.CommitLast;
2  439 --
2  440 --               boxView.boxList.InsLast(thisBox);
2  441 -1           END;
2  442 --           {$IFC fTrace}EP;{$ENDC}
2  443 -0 A      END;
2  444 --
2  445 --    END;
```

```
2  446 --
2  447 --
2  448 --
2  449 --
2  450 --    METHODS OF TRecolorCmd;
2  451 --
2  452 -- A      FUNCTION  TRecolorCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  453 --                                   itsView: TBoxView; itsBox: TBox; itsColor: TColor): TRecolorCmd;
2  454 0- A      BEGIN
2  455 --            {$IFC fTrace}BP(10);{$ENDC}
2  456 --            IF object = NIL THEN
2  457 --                object := NewObject(itsHeap, THISCLASS);
2  458 --            SELF := TRecolorCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealAll));
2  459 --            SELF.color := itsColor;
2  460 --            SELF.box := itsBox;
2  461 --            {$IFC fTrace}EP;{$ENDC}
2  462 -0 A      END;
2  463 --
2  464 --
2  465 --    {$IFC fDebugMethods}
2  466 -- A      PROCEDURE TRecolorCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  467 0- A      BEGIN
2  468 --            TCommand.Fields(Field);
2  469 --            Field('Color: INTEGER');
2  470 --            Field('box: TBox');
2  471 -0 A      END;
2  472 --     {$ENDC}
2  473 --
2  474 --
2  475 -- A      PROCEDURE TRecolorCmd.Commit;
2  476 0- A      BEGIN
2  477 --            {$IFC fTrace}BP(12);{$ENDC}
2  478 --            SELF.box.color := SELF.color;
2  479 --            {$IFC fTrace}EP;{$ENDC}
2  480 -0 A      END;
2  481 --
2  482 --
2  483 -- A      PROCEDURE TRecolorCmd.Perform(cmdPhase: TCmdPhase);
2  484 --        VAR boxView:     TBoxView;
2  485 --            tempColor:   TColor;
2  486 0- A      BEGIN
2  487 --            {$IFC fTrace}BP(12);{$ENDC}
2  488 --            boxView := TBoxView(SELF.image);
2  489 --            boxView.InvalBox(SELF.box.shapeLRect);
2  490 --            self.image.view.panel.selection.MarkChanged;    {allow this document to be saved}
2  491 --            {$IFC fTrace}EP;{$ENDC}
2  492 -0 A      END;
2  493 --
2  494 --
2  495 -- A      PROCEDURE TRecolorCmd.FilterAndDo(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject));
2  496 --            VAR saveColor:  TColor;
2  497 --                box:        TBox;
2  498 0- A      BEGIN
2  499 --            {$IFC fTrace}BP(12);{$ENDC}
2  500 --            box := TBox(actualObj);
2  501 --            IF box = SELF.box THEN
2  502 1-                BEGIN
2  503 --                saveColor := box.color;
2  504 --                box.color := SELF.Color;
2  505 --                DoToObject(box);
2  506 --                box.color := saveColor;
2  507 -1                END
2  508 --            ELSE
2  509 --                DoToObject(box);
2  510 --            {$IFC fTrace}EP;{$ENDC}
2  511 -0 A      END;
2  512 --
2  513 --    END;
2  514 --
2  515 --
2  516 --
2  517 --    METHODS OF TDuplicateCmd;
2  518 --
2  519 -- A      FUNCTION  TDuplicateCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  520 --                                     itsView: TBoxView; itsBox: TBox): TDuplicateCmd;
2  521 -- B          PROCEDURE CloneBox(filteredObj: TObject);
2  522 --            VAR box:     TBox;
2  523 0- B          BEGIN
2  524 --                box := TBox(filteredObj.Clone(itsHeap));
2  525 --                SELF.newBox := box;
2  526 -0 B          END;
2  527 0- A      BEGIN
2  528 --            {$IFC fTrace}BP(10);{$ENDC}
2  529 --            IF object = NIL THEN
2  530 --                object := NewObject(itsHeap, THISCLASS);
2  531 --            SELF := TDuplicateCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealAll));
2  532 --            TBoxView(SELF.image).FilterAndDo(itsbox, CloneBox);
2  533 --            SELF.oldBox := itsBox;
2  534 --      {$H-} OffSetLRect(SELF.newBox.shapeLRect, 20, 20); {$H+}
2  535 --            {$IFC fTrace}EP;{$ENDC}
2  536 -0 A      END;
2  537 --
2  538 --
2  539 -- A      PROCEDURE TDuplicateCmd.Free;
2  540 0- A      BEGIN
2  541 --            {$IFC fTrace}BP(10);{$ENDC}
2  542 --            Free(SELF.newBox);
2  543 --            SELF.FreeObject;
2  544 --            {$IFC fTrace}EP;{$ENDC}
2  545 -0 A      END;
2  546 --
2  547 --
2  548 --    {$IFC fDebugMethods}
2  549 -- A      PROCEDURE TDuplicateCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  550 0- A      BEGIN
2  551 --            TCommand.Fields(Field);
2  552 --            Field('oldBox: TBox');
2  553 --            Field('newBox: TBox');
2  554 -0 A      END;
2  555 --     {$ENDC}
```

```
2  556 --
2  557 --
2  558 -- A      PROCEDURE TDuplicateCmd.Commit;
2  559 --        VAR boxView:     TBoxView;
2  560 0- A      BEGIN
2  561 --            {$IFC fTrace}BP(12);{$ENDC}
2  562 --            boxView := TBoxView(SELF.image);
2  563 --            boxView.boxList.InsLast(SELF.newBox);
2  564 --            SELF.newBox := NIL;
2  565 --            {$IFC fTrace}EP;{$ENDC}
2  566 -0 A      END;
2  567 --
2  568 --
2  569 -- A      PROCEDURE TDuplicateCmd.Perform(cmdPhase: TCmdPhase);
2  570 --        VAR boxView:             TBoxView;
2  571 --            box:                 TBox;
2  572 --            thisSelection:       TBoxSelection;
2  573 0- A      BEGIN
2  574 --            {$IFC fTrace}BP(12);{$ENDC}
2  575 --            boxView := TBoxView(SELF.image);
2  576 --            thisSelection := TBoxSelection(boxView.panel.selection);
2  577 --
2  578 --               {-------------------------------------------------------------------------------
2  579 --                The current selection is unhighlighted before performing the command as the result
2  580 --                of the following command fields set by TCommand.CREATE:
2  581 --                                           unHiliteBefore[doPhase..redoPhase] <- TRUE
2  582 --
2  583 --                The resulting selection is highlighted after performing the command as the result of the
2  584 --                following command fields set by TCommand.CREATE:
2  585 --                                           hiLiteAfter   [doPhase..redoPhase] <- TRUE
2  586 --                --------------------------------------------------------------------------------}
2  587 --
2  588 --            WITH thisSelection DO
2  589 1-               CASE cmdPhase OF
2  590 --                   doPhase, redoPhase:
2  591 --                         box := SELF.newBox;
2  592 --                   undoPhase:
2  593 --                         box := SELF.oldBox;
2  594 -1               END {CASE};
2  595 --
2  596 --            boxView.InvalBox(SELF.newBox.shapeLRect);
2  597 --
2  598 --            self.image.view.panel.selection.MarkChanged;     {allow this document to be saved}
2  599 --            {$IFC fTrace}EP;{$ENDC}
2  600 -0 A      END;
2  601 --
2  602 --
2  603 -- A      PROCEDURE TDuplicateCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
2  604 0- A      BEGIN
2  605 --            {$IFC fTrace}BP(12);{$ENDC}
2  606 --            TBoxView(SELF.image).EachActualPart(DoToObject);
2  607 --            DoToObject(SELF.newBox);
2  608 --            {$IFC fTrace}EP;{$ENDC}
2  609 -0 A      END;
2  610 --
2  611 --    END;
2  612 --
2  613 --
2  614 --
2  615 --    METHODS OF TClearAllCmd;
2  616 --
2  617 -- A      FUNCTION  TClearAllCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  618 --                                      itsView: TBoxView): TClearAllCmd;
2  619 0- A      BEGIN
2  620 --            {$IFC fTrace}BP(10);{$ENDC}
2  621 --            IF object = NIL THEN
2  622 --                object := NewObject(itsHeap, THISCLASS);
2  623 --            SELF := TClearAllCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealNone));
2  624 --            SELF.kind := SELF.image.view.panel.selection.kind;
2  625 --            {$IFC fTrace}EP;{$ENDC}
2  626 -0 A      END;
2  627 --
2  628 --
2  629 --     {$IFC fDebugMethods}
2  630 -- A      PROCEDURE TClearAllCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  631 0- A      BEGIN
2  632 --            TCommand.Fields(Field);
2  633 --            Field('kind: INTEGER');
2  634 -0 A      END;
2  635 --      {$ENDC}
2  636 --
2  637 --
2  638 -- A      PROCEDURE TClearAllCmd.Commit;
2  639 0- A      BEGIN
2  640 --            {$IFC fTrace}BP(12);{$ENDC}
2  641 --            TBoxView(SELF.image).boxList.DelAll(TRUE);
2  642 --            {$IFC fTrace}EP;{$ENDC}
2  643 -0 A      END;
2  644 --
2  645 --
2  646 -- A      PROCEDURE TClearAllCmd.Perform(cmdPhase: TCmdPhase);
2  647 --        VAR thisSelection: TSelection;
2  648 --            boxView:     TBoxView;
2  649 0- A      BEGIN
2  650 --            {$IFC fTrace}BP(12);{$ENDC}
2  651 --            boxView := TBoxView(SELF.image);
2  652 --            thisSelection := boxView.panel.selection;
2  653 --
2  654 --            WITH thisSelection DO
2  655 1-               CASE cmdPhase OF
2  656 --                   doPhase, redoPhase:
2  657 --                          kind := nothingKind;
2  658 --                   undoPhase:
2  659 --                          kind := SELF.kind;
2  660 -1               END;
2  661 --
2  662 --                  { Invalidate the whole panel }
2  663 --            boxView.panel.Invalidate;
2  664 --
2  665 --            self.image.view.panel.selection.MarkChanged;     {allow this document to be saved}
```

```
2  666 --          {$IFC fTrace}EP;{$ENDC}
2  667 -0 A     END;
2  668 --
2  669 --
2  670 -- A     PROCEDURE TClearAllCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
2  671 0- A     BEGIN
2  672 --          {$IFC fTrace}BP(12);{$ENDC}
2  673 --          {$IFC fTrace}EP;{$ENDC}
2  674 -0 A     END;
2  675 --
2  676 --    END;
2  677 --
2  678 --
2  679 --
2  680 --    METHODS OF TBoxProcess;
2  681 --
2  682 -- A     FUNCTION  TBoxProcess.CREATE: TBoxProcess;
2  683 0- A     BEGIN
2  684 --          {$IFC fTrace}BP(11);{$ENDC}
2  685 --          SELF := TBoxProcess(TProcess.CREATE(NewObject(mainHeap, THISCLASS), mainHeap));
2  686 --          {$IFC fTrace}EP;{$ENDC}
2  687 -0 A     END;
2  688 --
2  689 --
2  690 -- A     FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN): TDocManager;
2  691 0- A     BEGIN
2  692 --          {$IFC fTrace}BP(11);{$ENDC}
2  693 --          NewDocManager := TBoxDocManager.CREATE(NIL, mainHeap, volumePrefix);
2  694 --          {$IFC fTrace}EP;{$ENDC}
2  695 -0 A     END;
2  696 --
2  697 --    END;
2  698 --
2  699 --
2  700 --
2  701 --    METHODS OF TBoxDocManager;
2  702 --
2  703 -- A     FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
2  704 --                                      : TBoxDocManager;
2  705 0- A     BEGIN
2  706 --          {$IFC fTrace}BP(11);{$ENDC}
2  707 --          IF object = NIL THEN
2  708 --              object := NewObject(itsHeap, THISCLASS);
2  709 --          SELF := TBoxDocManager(TDocManager.CREATE(object, itsHeap, itsPathPrefix));
2  710 --          {$IFC fTrace}EP;{$ENDC}
2  711 -0 A     END;
2  712 --
2  713 --
2  714 -- A     FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID):TWindow;
2  715 0- A     BEGIN
2  716 --          {$IFC fTrace}BP(11);{$ENDC}
2  717 --          NewWindow := TBoxWindow.CREATE(NIL, heap, wmgrID);
2  718 --          {$IFC fTrace}EP;{$ENDC}
2  719 -0 A     END;
2  720 --
2  721 --    END;
2  722 --
2  723 --
2  724 --
2  725 --    METHODS OF TBoxWindow;
2  726 --
2  727 -- A     FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
2  728 0- A     BEGIN
2  729 --          {$IFC fTrace}BP(10);{$ENDC}
2  730 --          IF object = NIL THEN
2  731 --              object := NewObject(itsHeap, THISCLASS);
2  732 --          SELF := TBoxWindow(TWindow.CREATE(object, itsHeap, itsWmgrID, TRUE));
2  733 --          {$IFC fTrace}EP;{$ENDC}
2  734 -0 A     END;
2  735 --
2  736 --
2  737 -- A     PROCEDURE TBoxWindow.BlankStationery;
2  738 --       VAR viewLRect:       LRect;
2  739 --           panel:           TPanel;
2  740 --           boxView:         TBoxView;
2  741 --           aSelection:      TSelection;
2  742 0- A     BEGIN
2  743 --          {$IFC fTrace}BP(10);{$ENDC}
2  744 --           panel := TPanel.CREATE(NIL, SELF.Heap, SELF, 0, 0, [aScroll, aSplit], [aScroll, aSplit]);
2  745 --
2  746 --          SetLRect(viewLRect, 0, 0, 5000, 3000);
2  747 --          boxView := TBoxView.CREATE(NIL, SELF.Heap, panel, viewLRect);
2  748 --          boxView.InitBoxList(SELF.Heap);
2  749 --
2  750 --          {$IFC fTrace}EP;{$ENDC}
2  751 -0 A     END;
2  752 --
2  753 --
2  754 -- A     FUNCTION  TBoxWindow.NewCommand(cmdNumber: TCmdNumber): TCommand;
2  755 0- A     BEGIN
2  756 --          {$IFC fTrace}BP(11);{$ENDC}
2  757 --
2  758 1-          CASE cmdNumber OF
2  759 --              uClearAll:
2  760 --                  NewCommand := TClearAllCmd.CREATE(NIL, SELF.heap, cmdNumber,
2  761 --                                                    TBoxView(SELF.selectPanel.view));
2  762 --
2  763 --              OTHERWISE
2  764 --                  NewCommand := SUPERSELF.NewCommand(cmdNumber);
2  765 -1              END;
2  766 --          {$IFC fTrace}EP;{$ENDC}
2  767 -0 A     END;
2  768 --
2  769 --
2  770 -- A     FUNCTION  TBoxWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
2  771 0- A     BEGIN
2  772 --          {$IFC fTrace}BP(11);{$ENDC}
2  773 1-          CASE cmdNumber OF
2  774 --              uClearAll:
2  775 --                      CanDoCommand := TRUE;
```

```
2  776 --                 OTHERWISE
2  777 --                    CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
2  778 -1                 END;
2  779 --              {$IFC fTrace}EP;{$ENDC}
2  780 -0 A      END;
2  781 --
2  782 --    END;
1  216 --
1  217 --    END.
```

```
 1. u7boxer.TEXT
 2. U7Boxer2.text

-B-
  BlankStationery    204•( 1)  737•( 2)
  Box                134 ( 1)
  box                 97 ( 1)  120 ( 1)  121•( 2)  123=( 2)  124 ( 2)  125 ( 2)  138•( 2)  140=( 2)  141 ( 2)  162•( 2)
                     168=( 2)  170 ( 2)  177 ( 2)  227=( 2)  247 ( 2)  270 ( 2)  276 ( 2)  303 ( 2)  307 ( 2)  338•( 2)
                     345=( 2)  346=( 2)  346 ( 2)  365•( 2)  369 ( 2)  375=( 2)  375 ( 2)  389 ( 2)  395 ( 2)  418 ( 2)
                     460=( 2)  478 ( 2)  489 ( 2)  497•( 2)  500=( 2)  501 ( 2)  501 ( 2)  503 ( 2)  504 ( 2)  505 ( 2)
                     506 ( 2)  509 ( 2)  522•( 2)  524=( 2)  525 ( 2)  571•( 2)  591=( 2)  593=( 2)
  boxList             69 ( 1)  153 ( 2)  197•( 2)  200=( 2)  201=( 2)  201 ( 2)  440 ( 2)  563 ( 2)  641 ( 2)
  boxSelectionKind    28•( 1)  177 ( 2)  425 ( 2)
  BoxWith             75•( 1)  119•( 2)  125=( 2)  168 ( 2)

-C-
  CanDoCommand       112•( 1)  208•( 1)  316•( 2)  322=( 2)  325=( 2)  325 ( 2)  770•( 2)  775=( 2)  777=( 2)  777 ( 2)
  color               50•( 1)  135•( 1)   12=( 2)   37 ( 2)  459=( 2)  478=( 2)  478 ( 2)  503 ( 2)  504=( 2)  506=( 2)
  colorBlack          25•( 1)   42 ( 1)   42 ( 2)
  colorDkGray         24•( 1)   41 ( 2)
  colorGray           23•( 1)   12 ( 2)   40 ( 2)
  colorLtGray         22•( 1)   39 ( 2)
  colorWhite          21•( 1)   42 ( 1)   38 ( 2)  304 ( 2)
  Commit             142•( 1)  158•( 1)  174•( 1)  475•( 2)  558•( 2)  638•( 2)
  CREATE              53•( 1)   72•( 1)  100•( 1)  123•( 1)  138•( 1)  154•( 1)  170•( 1)  183 ( 1)  192 ( 1)  201 ( 1)
                       5•( 2)   97•( 2)  103 ( 2)  173 ( 2)  177 ( 2)  200 ( 2)  209 ( 2)  219•( 2)  225 ( 2)  303 ( 2)
                     307 ( 2)  336•( 2)  343 ( 2)  345 ( 2)  425 ( 2)  452•( 2)  458 ( 2)  519•( 2)  531 ( 2)  617•( 2)
                     623 ( 2)  682•( 2)  685 ( 2)  693 ( 2)  703•( 2)  709 ( 2)  717 ( 2)  727•( 2)  732 ( 2)  744 ( 2)
                     747 ( 2)  760 ( 2)
  createBoxSelecti    29•( 1)  343 ( 2)

-D-
  Draw                62•( 1)   83•( 1)   28•( 2)  136•( 2)  141 ( 2)
  DrawFrame           59•( 1)   54•( 2)  369 ( 2)  411 ( 2)

-E-
  EachActualPart      86•( 1)  150•( 2)  606 ( 2)
  EachVirtualPart    160•( 1)  176•( 1)  130 ( 2)  145 ( 2)  603•( 2)  670•( 2)

-F-
  Fields              18•( 2)  110•( 2)  112 ( 2)  232•( 2)  234 ( 2)  352•( 2)  354 ( 2)  466•( 2)  468 ( 2)  549•( 2)
                     551 ( 2)  630•( 2)  632 ( 2)
  FilterAndDo        144•( 1)  495•( 2)  532 ( 2)
  Free               539•( 2)  542 ( 2)

-H-
  Highlight          104•( 1)  167 ( 2)  179 ( 2)  241•( 2)

-I-
  InitBoxList         89•( 1)  196•( 2)  748 ( 2)
  InvalBox            78•( 1)  187•( 2)  272 ( 2)  274 ( 2)  428 ( 2)  489 ( 2)  596 ( 2)

-K-
  kind               167 ( 1)  244 ( 2)  322 ( 2)  433=( 2)  624=( 2)  624 ( 2)  657=( 2)  659=( 2)  659 ( 2)

-L-
  LRect               49 ( 1)   67 ( 2)  257 ( 2)  362 ( 2)  405 ( 2)  738 ( 2)

-M-
  MouseMove          107•( 1)  127•( 1)  254•( 2)  361•( 2)
  MousePress          80•( 1)  159•( 2)
  MouseRelease       108•( 1)  128•( 1)  282•( 2)  402•( 2)

-N-
  newBox             151 ( 1)  525=( 2)  534 ( 2)  542 ( 2)  563 ( 2)  564=( 2)  591 ( 2)  596 ( 2)  607 ( 2)
  NewCommand         111•( 1)  207•( 1)  292•( 2)  303=( 2)  307=( 2)  310=( 2)  310 ( 2)  754•( 2)  760=( 2)  764=( 2)
                     764 ( 2)
  NewDocManager      184•( 1)  690•( 2)  693=( 2)
  NewWindow          194•( 1)  714•( 2)  717=( 2)
  NoSelection         90•( 1)  206•( 2)  209=( 2)

-O-
  oldBox             151 ( 1)  533=( 2)  593 ( 2)

-P-
  PaintHandles        56•( 1)   65•( 2)  247 ( 2)
  Perform            143•( 1)  159•( 1)  175•( 1)  483•( 2)  569•( 2)  646•( 2)

-Q-
  QuickDraw           16•( 1)

-S-
  shapeLRect          49 ( 1)   11=( 2)   34 ( 2)   47 ( 2)   48 ( 2)   59 ( 2)   67•( 2)   79=( 2)   79 ( 2)   80 ( 2)
                     124 ( 2)  257•( 2)  270=( 2)  270 ( 2)  272 ( 2)  273 ( 2)  274 ( 2)  276=( 2)  276 ( 2)  391 ( 2)
                     392 ( 2)  395 ( 2)  420 ( 2)  489 ( 2)  534 ( 2)  596 ( 2)

-T-
  TBox                46•( 1)   53 ( 1)   75 ( 1)   97 ( 1)  120 ( 1)  134 ( 1)  151 ( 1)    3•( 2)    5 ( 2)  119 ( 2)
                     121 ( 2)  123 ( 2)  138 ( 2)  140 ( 2)  162 ( 2)  338 ( 2)  345 ( 2)  365 ( 2)  403 ( 2)  497 ( 2)
                     500 ( 2)  522 ( 2)  524 ( 2)  571 ( 2)
  TBoxDocManager     189•( 1)  193 ( 1)  693 ( 2)  701•( 2)  704 ( 2)  709 ( 2)
  TBoxProcess        180•( 1)  183 ( 1)  680•( 2)  682 ( 2)  685 ( 2)
  TBoxSelection       94•( 1)  101 ( 1)  177 ( 2)  209 ( 2)  217•( 2)  220 ( 2)  225 ( 2)  425 ( 2)  572 ( 2)  576 ( 2)
  TBoxView            66•( 1)   73 ( 1)   95•( 2)   98 ( 2)  103 ( 2)  256 ( 2)  260 ( 2)  293 ( 2)  298 ( 2)  364 ( 2)
                     374 ( 2)  404 ( 2)  416 ( 2)  484 ( 2)  488 ( 2)  532 ( 2)  559 ( 2)  562 ( 2)  570 ( 2)  575 ( 2)
                     606 ( 2)  641 ( 2)  648 ( 2)  651 ( 2)  740 ( 2)  747 ( 2)  761 ( 2)
  TBoxWindow         198•( 1)  201 ( 1)  717 ( 2)  725•( 2)  727 ( 2)  732 ( 2)
  TClearAllCmd       165•( 1)  171 ( 1)  615•( 2)  618 ( 2)  623 ( 2)  760 ( 2)
  TColor              42•( 1)   50 ( 1)  135 ( 1)  485 ( 2)  496 ( 2)
  TCommand           111 ( 1)  133 ( 1)  150 ( 1)  165 ( 1)  207 ( 1)  292 ( 2)  458 ( 2)  468 ( 2)  531 ( 2)  551 ( 2)
                     623 ( 2)  632 ( 2)  754 ( 2)
  TCreateBoxSelect   117•( 1)  124 ( 1)  173 ( 2)  334•( 2)  337 ( 2)  343 ( 2)
  TDocManager        185 ( 1)  189 ( 1)  690 ( 2)  709 ( 2)
  TDuplicateCmd      150•( 1)  155 ( 1)  307 ( 2)  517•( 2)  520 ( 2)  531 ( 2)
  TList               69 ( 1)  197 ( 2)  200 ( 2)
  TObject             46 ( 1)  120 ( 2)  137 ( 2)  521 ( 2)
  TProcess           180 ( 1)  685 ( 2)
  TRecolorCmd        133•( 1)  139 ( 1)  303 ( 2)  450•( 2)  453 ( 2)  458 ( 2)
  TSelection          90 ( 1)   94 ( 1)  117 ( 1)  160 ( 2)  206 ( 2)  225 ( 2)  234 ( 2)  343 ( 2)  354 ( 2)  406 ( 2)
                     647 ( 2)  741 ( 2)
```

```
  TView              66 ( 1)  112 ( 2)
  TWindow           194 ( 1)  198 ( 1)  714 ( 2)  732 ( 2)

-U-
  U7Boxer             5*( 1)
  UABC               18*( 1)
  uBlack             36*( 1)  302 ( 2)  320 ( 2)
  uClearAll          38*( 1)  759 ( 2)  774 ( 2)
  uDkGray            35*( 1)  302 ( 2)  320 ( 2)
  UDraw              17*( 1)
  uDuplicate         37*( 1)  306 ( 2)  321 ( 2)
  UFont              13*( 1)
  uGray              34*( 1)  302 ( 2)  320 ( 2)
  uLtGray            33*( 1)  302 ( 2)  320 ( 2)
  UObject            10*( 1)
  uWhite             32*( 1)  302 ( 2)  304 ( 2)  320 ( 2)

*** End Xref: 72 id's   472 references      [398352 bytes/4927 id's/39940 refs]
```

[Segment 11]

Advanced Commands with Cut & Paste

Special Note: The style and structure of this segment differs from the previous ten.

Purpose of this segment:

This segment presents three separate but related topics completing the self—paced introduction to the ToolKit.

1. Implementing cut, copy, and paste within Boxer.
2. Creating command objects from non—menu events, such as mouse and key events.
3. Responding to the tab and clear keys.

How to use this segment:

This is the last segment in the self—paced introduction to Boxer. Try to follow the discussion, and carefully study the sample program.

THE CLIPBOARD

The clipboard is used for intraprocess data transfer and interprocess communication. The clipboard only contains one piece of information at a time but it has 3 parts, all containing different representations of that information.

1) application specific part

If a ToolKit application created the current contents of the clipboard then this representation is a minimum ToolKit document. It contains at least a window, panel, view, and selection. If a non—ToolKit application created the current contents of the clipboard then this part is specific to that application. This part of the clipboard contains the most information about its contents.

2) universal picture part

This is a QuickDraw picture of of the contents of the clipboard. There are no easily accessible semantics associated with a picture. However, LisaDraw or the graphics building block will be able to parse a picture into its component graphical objects.

3) universal text part

This is a text data format understood by all desktop applications. It contains the text and text descriptors, such as font, format, margin, and tab information.

Every time you cut to the clipboard as many representations as possible are generated. The application is responsible for generating the application specific portion (as you will learn). The ToolKit generates the universal picture. If you use the Text Building Block it will automatically generate universal text for any text that is cut or copied.

Every time you paste from the clipboard you may choose which one of the representations you want to paste. Whenever possible this will be the Application specific representation.

FLOW OF CONTROL FOR CUT AND PASTE

Cut, Copy, and Paste are unique commands because they read from or wite to the clipboard. Since there is magic that must be performed to deal with the clipboard, there are two subclasses of **TCommand** that must be subclassed — **TCutCopyCommand** (for cut and copy) and **TPasteCommand** (for paste). The flow of control is the same as other commands except that applications do not reimplement **Perform**. Instead they reimplement **DoCutCopy** or **DoPaste**. Both take as parameters the command phase and the clipboard's selection. For ToolKit cut and paste the selection is used to access all the objects on the clipboard. From the selection it is easy to get to the view, window, or panel.

The **Commit**, **Free**, and all the filter methods can be implemented using techniques already discussed. This segment will implement cut, copy, and paste with filters becuase it is easier, and because it provides another good example of how to use filters. Do not worry about freeing any of the objects placed on the Clipboard, the Clipboard will clear itself on every cut.

IMPLEMENTING CUT AND COPY

To implement cut and copy make a subclass of **TCutCopyCommand** and override **DoCutCopy**. On the *doPhase* **DoCutCopy** must create a new view and selection on the clipBoard, then move into it the data being cut or copied. To do this a new view of type **TBoxView** is created (with boxList and boxes), and passed to {TBoxSelection.}CREATE. ClipSelection, which is a dummy object, is freed and replaced with the new selection. On a cut the data must be removed from the

document. If filters are used then **Commit** and **EachVirtualPart** must check SELF.**isCut** since copying doesn't change the document. As an interesting example 8Boxer's implementation of the cut and copy filters implements both **EachVirtualPart** and **FilterAndDo**.

IMPLEMENTING PASTE

To implement paste make a subclass of **TPasteCommand** and override {TPasteCommand.}**DoPaste**. Instead of freeing and replacing clipSelection with a new selection (as in cut and copy), we retain it. It now points to the data we want to paste. For this section we are only interested in pasting from other Boxer documents. Therefore we will only paste from clipSelection when it is of class **TBoxSelection**. It is possible to paste from other ToolKit applications but that is left for another segment. It is also easy to paste from universal pictures. See the appendix at the end of this segment on pasting from other applications. To paste from clipSelection:

1. On the *doPhase* first check if it is of class **TBoxSelection** using the ToolKit function, InClass, as follows:

   ```
   IF InClass(clipSelection, TBoxSelection) THEN
   ```

   This function returns TRUE only if clipSelection is an instance of TBoxSelection, or an instance of one of its subclasses.
2. If clipSelection is of some other type, put up the *Can't Do It* alert.
3. Otherwise simply copy the data from the view that clipSelection points to.
4. Now you must properly handle the coordinates of the pasted box, since it is always located in the upper left hand corner of the Clipboard. For Boxer we want to paste the box centered around the last place the mouse was pressed (if no box was selected), or centered within the selected box. **TView** has a field, **clickLPt**, which contains the location of the last mouse press.
5. Again, once the box has been copied into the document the do, undo, and redo phases handle the selection, highlighting, and invalidation using existing techniques.

FLOW OF CONTROL FOR COMMANDS NOT ORIGINATED FROM THE MENU BAR

In this section we are going to convert the existing mouse events into command objects. We will also respond to two keyboard events, one which will be converted into a command.

When implementing keyboard or mouse events as commands, we need a mechanism for installing the new command object since we will not be in **NewCommand** when the event occurs.

{window.}**PerformCommand** takes as argument the command object to be performed. Calling **PerformCommand** with a command object produces the same result as returning a command object to **NewCommand**. As you can glean from the flow of control for commands [see segment 9], **PerformCommand** is the next method called after **NewCommand**.

FLOW OF CONTROL FOR MOUSE COMMANDS

To convert our existing mouse events to commands we implement command classes for the mouse events we want to be undoable. The identifiable mouse events that edit the document are: *box move* and *box create*. Upon mouse release in both of these events we want to create an appropriate command object. It will perform the event as a command, and pass the command object to **PerformCommand**.

IMPLEMENTING MOUSE COMMANDS

For *box move*, each mouse move event edits the document. The original state is saved in {selection.}**anchorLPt**. With a little forethought, it is easy to see that *box move* should be an unfiltered command. The original state is simple to remember and restore, and doing the command with a filter would be more complicated than doing it without one. The *doPhase* for this command does nothing since the mouse move code has already edited and properly displayed the document. The undo and redo phases should move the box to the correct location and invalidate.

Suggestion: Move the code in {TBoxSelection.}**MouseMove** that moves a box to a new method of TBox. Then call the new method from both **MouseMove** and {TMoveBoxCmd.}**Perform**.

For *box create*, the mouse move and mouse release code already act as a filter through which the document is viewed. The newly created box is not inserted into boxList until the end of {TBoxCreateSelection.}**MouseRelease**. By replacing the InLast(SELF.**box**) line in **MouseRelease** with a call to **PerformCommand**, we can easily create a filtered command. This command shall be implememnted as **TCreateCmd**. It turns out that **TCreateCmd** is very similar to **TDuplicateCmd**.

FLOW OF CONTROL FOR KEY COMMANDS

Keyboard events go directly from the process to {selection.}**DoKey**. **DoKey** then farms it out to the appropriate method of the selection. For our example, there is no need to override **DoKey**, since the default mapping of keys to methods is acceptable. We will reimplement {selection.}**KeyTab** and {selection.}**KeyClear**, each of which is called by **DoKey** depending upon the key pressed. Since striking the tab key will not edit the document (it only changes the selection), we will not turn it into a command. We will, though, make the clear key event a command. In addition, we will also provide a menu item that is equivalent to striking the clear key.

selection.DoKey

case of

enter, arrow keys:	selection.**KeyEnter**
clear key:	selection.**KeyClear**
backspace key:	selection.**KeyBack**
shift backspace:	selection.**KeyForward**
return key:	selection.**KeyReturn**
tab key:	selection.**KeyTab**
otherwise	selection.**KeyChar**

IMPLEMENTING THE TAB KEY

{TSelection.}**KeyTab** is the selection method that responds to the tab key. The tab key is not undoable, and does not need a command object, but implementing it is an interesting excercise in list managment. *Note: Beware of virtual objects.*

IMPLEMENTING THE CLEAR KEY

{TSelection.}**KeyClear** responds to clear key events. KeyClear calls **PerformCommand** with a command object that implements clearing a box. This same command class is also used for the menu version of clear. The semantics of *clear* are close enough to *cut* to be a subclass of **TCutCopyCommand**.

OTHER CHANGES THAT WERE REQUIRED

Changes to **NewCommand** and **CanDoCommand** for *cut, paste,* and *clear*.

Added {TBox.}**MoveBox** to allow sharing of box move code between {TBoxSelection.}**MouseMove** and {TMoveBoxCmd.}**Perform**.

Reimplemented **TDuplicateCmd** as a subclass of **TCreateCmd**. Introduced a new method, **UpdateSelection**, that updates the selection on undo and redo phases.

Defined two new command constants, uCreateBox and uMoveBox.

Appendix A
Cut and Paste between Applications

PASTING FROM ANOTHER TOOLKIT DOCUMENT

Works as described above.

PASTING FROM A NON-TOOLKIT DOCUMENT

When clipSelection = NIL, or clipboard.hasView = FALSE, there will always be a universal picture to paste from and there will often be universal text to paste. The type of your application and the context being pasted into will determine which one is appropriate.

If your application is largly textual then paste the universal text. If your application is strictly graphical and has no text then paste from the universal picture. Universal text only exists when something textual was cut or copied. But a universal picture is always generated. If universal text is present and you are capable of taking text then paste it instead of the picture.

PASTING A UNIVERSAL PICTURE

Do not just grab the handle, since the picture it points to is in the clipboard's heap. That heap will be unbound from the applications data space immediately after the paste, thus invalidating the handle. Instead, copy the picture onto the document's heap. There are several ways to copy a picture, the simplest is:

1. Save the clipboard heap. The global variable, theHeap, refers to it.
   ```
   tempHeap <- theHeap;
   ```
2. Reset the global heap variable, theHeap, to the document's heap.
   ```
   SetHeap(SELF.heap);
   ```
3. Open a picture to copy the clipboard picture into.
   ```
   myPicture := OpenPicture(thisRect);
   ```
4. Draw the clipboard picture. This effectively copies the picture.
   ```
   DrawPicture(pic);
   ```
5. Close the picture.
   ```
   ClosePicture;
   ```
6. Restore theHeap to the clipboard heap.
   ```
   SetHeap(tempHeap);
   ```

PASTING UNIVERSAL TEXT

The Text Building Block pastes from universal text whenever clipSelection = NIL, and universal text is present.

Code Sample
for this
Segment

```
28
SLOT2CHAN1
; no assembler files
$
; no building blocks
$
; no links
$
y
y
n
BoxNum8
```

```
;  P8BOXER.TEXT
1
3
2500
$-#BOOT-TK/PABC

; Apple building block phrase files can be included here

1000
8Boxer

; Other application alerts can be included here, numbered between 1001 and 32000

0

1
$-#BOOT-TK/PABC~File/Print

2
Edit
Undo Last Change#205
-
Cut/X#202
Copy/C#201
Paste/V#203
Duplicate/D#1011
-
Clear Box#208
Clear All/Z#1012
-

3
Shades
White#1006
Light Gray#1007
Gray#1008
Dark Gray#1009
Black#1010

5
$-#BOOT-TK/PABC~Page Layout

99
$-#BOOT-TK/PABC~Debug

100
$-#BOOT-TK/PABC~Buzzwords
Create Box#2000
Move Box#2001

0
```

```
PROGRAM M8Boxer;

USES
    {$U UObject     } UObject,

  {$IFC libraryVersion <= 20}
    {$U UFont}               UFont,
  {$ENDC}

    {$U QuickDraw   } QuickDraw,
    {$U UDraw       } UDraw,
    {$U UABC        } UABC,

    {$U U8Boxer     } U8Boxer;

CONST

    phraseVersion = 1;

BEGIN

    process := TBoxProcess.CREATE;
    process.Commence(phraseVersion);
    process.Run;
    process.Complete(TRUE);

END.
```

```
1    1 --       {This LisaBoxer sample implements cut and paste, move and create undo, Tab and clear keys with undo}
1    2 --       {Copyright 1983, Apple Computer Inc.}
1    3 --
1    4 --   UNIT U8Boxer;
1    5 --
1    6 --   INTERFACE
1    7 --
1    8 --   USES
1    9 --       {$U UObject}            UObject,
1   10 --
1   11 --     {$IFC libraryVersion <= 20}
1   12 --       {$U UFont}              UFont,
1   13 --     {$ENDC}
1   14 --
1   15 --       {$U QuickDraw}          QuickDraw,
1   16 --       {$U UDraw}              UDraw,
1   17 --       {$U UABC}               UABC;
1   18 --
1   19 --   CONST
1   20 --       colorWhite = 1;
1   21 --       colorLtGray = 2;
1   22 --       colorGray = 3;
1   23 --       colorDkGray = 4;
1   24 --       colorBlack = 5;
1   25 --
1   26 --     { selection kinds }
1   27 --       boxSelectionKind = 1;
1   28 --       createBoxSelectionKind = 2;
1   29 --
1   30 --     { Menus }
1   31 --       uWhite = 1006;
1   32 --       uLtGray = 1007;
1   33 --       uGray = 1008;
1   34 --       uDkGray = 1009;
1   35 --       uBlack = 1010;
1   36 --       uDuplicate = 1011;
1   37 --       uClearAll = 1012;
1   38 --
1   39 --     { Implied commands }
1   40 --       uCreateBox = 2000;
1   41 --       uMoveBox = 2001;
1   42 --
1   43 --   TYPE
1   44 --
1   45 --       TColor = colorWhite..colorBlack; {color of a box}
1   46 --
1   47 --     {New Classes for this Application}
1   48 --
1   49 --       TBox = SUBCLASS OF TObject
1   50 --
1   51 --           {Variables}
1   52 --           shapeLRect:     LRect;
1   53 --           color:          TColor;
1   54 --
1   55 --         {Creation/Destruction}
1   56 --           FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
1   57 --
1   58 --         { Display }
1   59 --           PROCEDURE TBox.Draw;
1   60 --           PROCEDURE TBox.DrawFrame;
1   61 --
1   62 --         { Editing and Display }
1   63 --           PROCEDURE TBox.MoveBox(boxView: TBoxView; deltaLPt: LPoint);
1   64 --
1   65 --         { Highlighting support }
1   66 --           PROCEDURE TBox.PaintHandles;
1   67 --           END;
1   68 --
1   69 --
1   70 --       TBoxView = SUBCLASS OF TView
1   71 --
1   72 --         {Variables}
1   73 --           boxList:        TList;
1   74 --
1   75 --         {Creation/Destruction}
1   76 --           FUNCTION  TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
1   77 --                                       : TBoxView;
1   78 --
1   79 --           FUNCTION  TBoxView.BoxWith(LPt: LPoint): TBox;
1   80 --
1   81 --         {Invalidation}
1   82 --           PROCEDURE TBoxView.InvalBox(invalLRect: LRect);
1   83 --
1   84 --           PROCEDURE TBoxView.MousePress(mouseLPt: LPoint); OVERRIDE;
1   85 --
1   86 --         {Display}
1   87 --           PROCEDURE TBoxView.Draw; OVERRIDE;
1   88 --
1   89 --         {Filtering}
1   90 --          PROCEDURE  TBoxView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1   91 --
1   92 --         {Initialization}
1   93 --           PROCEDURE TBoxView.InitBoxList(itsHeap: THeap);
1   94 --           FUNCTION  TBoxView.NoSelection: TSelection; OVERRIDE;
1   95 --          END;
1   96 --
1   97 --
1   98 --       TBoxSelection = SUBCLASS OF TSelection
1   99 --
1  100 --         {Variables}
1  101 --           box:   TBox;
1  102 --
1  103 --         {Creation/Destruction}
1  104 --           FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsBox: TBox;
1  105 --                                        itsKind: INTEGER; itsAnchorLPt: LPoint): TBoxSelection;
1  106 --
1  107 --         {Drawing - per pad}
1  108 --           PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit); OVERRIDE;
1  109 --
1  110 --         {Selection - per pad}
```

```
1  111 --            PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  112 --            PROCEDURE TBoxSelection.MouseRelease; OVERRIDE;
1  113 --
1  114 --          {Command Dispatch}
1  115 --            FUNCTION TBoxSelection.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  116 --            FUNCTION TBoxSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN)
1  117 --                                              : BOOLEAN; OVERRIDE;
1  118 --        END;
1  119 --
1  120 --
1  121 --        TCreateBoxSelection = SUBCLASS OF TSelection
1  122 --
1  123 --          {Variables}
1  124 --            box:   TBox;
1  125 --
1  126 --          {Creation/Destruction}
1  127 --            FUNCTION  TCreateBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView;
1  128 --                                           itsAnchorLPt: LPoint): TCreateBoxSelection;
1  129 --
1  130 --          {Selection - per pad}
1  131 --            PROCEDURE TCreateBoxSelection.MouseMove(mouseLPt: LPoint); OVERRIDE;
1  132 --            PROCEDURE TCreateBoxSelection.MouseRelease; OVERRIDE;
1  133 --        END;
1  134 --
1  135 --
1  136 --                 { This command recolors the selected box and is not undoable
1  137 --                   should it instead return an instance of TCommand }
1  138 --        TRecolorCmd = SUBCLASS OF TCommand
1  139 --            Box:     TBox;
1  140 --            color:   TColor;
1  141 --
1  142 --          {Creation}
1  143 --            FUNCTION  TRecolorCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  144 --                                 itsView: TBoxView; itsBox: TBox; itsColor: TColor): TRecolorCmd;
1  145 --
1  146 --            PROCEDURE TRecolorCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  147 --            PROCEDURE TRecolorCmd.FilterAndDo(actualObj: TObject;
1  148 --                                       PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  149 --            END;
1  150 --
1  151 --
1  152 --        TCreateCmd = SUBCLASS OF TCommand
1  153 --
1  154 --          {Variables}
1  155 --            newBox: TBox;
1  156 --
1  157 --          {Creation and Destruction}
1  158 --            FUNCTION  TCreateCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  159 --                                itsView: TBoxView; itsBox: TBox): TCreateCmd;
1  160 --            PROCEDURE TCreateCmd.Commit; OVERRIDE;
1  161 --            PROCEDURE TCreateCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  162 --            PROCEDURE TCreateCmd.UpdateSelection(thisSelection: TBoxSelection; cmdPhase: TCmdPhase);
1  163 --            PROCEDURE TCreateCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  164 --            END;
1  165 --
1  166 --
1  167 --                  { This command duplicates the selected box and is undoable }
1  168 --        TDuplicateCmd = SUBCLASS OF TCreateCmd
1  169 --
1  170 --          {Variables}
1  171 --            oldBox:      TBox;
1  172 --
1  173 --          {Creation}
1  174 --            FUNCTION  TDuplicateCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  175 --                                  itsView: TboxView; itsBox: TBox): TDuplicateCmd;
1  176 --
1  177 --          {Command Execution}
1  178 --            PROCEDURE TDuplicateCmd.UpdateSelection(thisSelection: TBoxSelection; cmdPhase: TCmdPhase);
1  179 --                                        OVERRIDE;
1  180 --            END;
1  181 --
1  182 --
1  183 --        TBoxCutCopyCmd = SUBCLASS OF TCutCopyCommand
1  184 --
1  185 --          {Variables}
1  186 --            selTopLeft:      LPoint;
1  187 --            box:             TBox;
1  188 --
1  189 --          {Creation}
1  190 --            FUNCTION  TBoxCutCopyCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  191 --                                  itsView: TView; isCutCmd: BOOLEAN; itsBox: TBox): TBoxCutCopyCmd;
1  192 --
1  193 --          {Command Execution}
1  194 --            PROCEDURE TBoxCutCopyCmd.Commit; OVERRIDE;
1  195 --            PROCEDURE TBoxCutCopyCmd.DoCutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN;
1  196 --                                      cmdPhase: TCmdPhase); OVERRIDE;
1  197 --            PROCEDURE TBoxCutCopyCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  198 --            END;
1  199 --
1  200 --
1  201 --        TBoxPasteCmd = SUBCLASS OF TPasteCommand
1  202 --
1  203 --          {Variables}
1  204 --            pasteH:          LONGINT;
1  205 --            pasteV:          LONGINT;
1  206 --            pasteBox:        TBox;
1  207 --
1  208 --          {Creation and Destruction}
1  209 --            FUNCTION  TBoxPasteCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  210 --                                 itsView: TBoxView; itsH, itsV: LONGINT): TBoxPasteCmd;
1  211 --            PROCEDURE TBoxPasteCmd.Free; OVERRIDE;
1  212 --
1  213 --          {Command Execution}
1  214 --            PROCEDURE TBoxPasteCmd.Commit; OVERRIDE;
1  215 --            PROCEDURE TBoxPasteCmd.DoPaste(clipSelection: TSelection; pic: PicHandle; cmdPhase: TCmdPhase);
1  216 --                                     OVERRIDE;
1  217 --            PROCEDURE TBoxPasteCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject)); OVERRIDE;
1  218 --            END;
1  219 --
1  220 --
```

```
1  221 --              { This command duplicates the selected box and is undoable }
1  222 --        TClearAllCmd = SUBCLASS OF TCommand
1  223 --          {Variables}
1  224 --            kind:   INTEGER;
1  225 --
1  226 --          {Creation}
1  227 --            FUNCTION  TClearAllCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  228 --                                            itsView: TboxView): TClearAllCmd;
1  229 --
1  230 --          {Command Execution}
1  231 --            PROCEDURE TClearAllCmd.Commit; OVERRIDE;
1  232 --            PROCEDURE TClearAllCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  233 --            PROCEDURE TClearAllCmd.EachVirtualPart(PROCEDURE D  Object(filteredObj: TObject)); OVERRIDE;
1  234 --            END;
1  235 --
1  236 --
1  237 --        TClearCmd = SUBCLASS OF TBoxCutCopyCmd
1  238 --
1  239 --          {Creation}
1  240 --            FUNCTION  TClearCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  241 --                                         itsView: TBoxView; itsBox: TBox): TClearCmd;
1  242 --
1  243 --          {Command Execution}
1  244 --            PROCEDURE TClearCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  245 --            END;
1  246 --
1  247 --
1  248 --        TMoveBoxCmd = SUBCLASS OF TCommand
1  249 --
1  250 --          {Variables}
1  251 --            hOffset:     LONGINT;
1  252 --            vOffset:     LONGINT;
1  253 --            movedBox:    TBox;
1  254 --
1  255 --          {Creation}
1  256 --            FUNCTION  TMoveBoxCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
1  257 --                                           itsView: TBoxView; itsBox: TBox; itsHOffset, itsVOffset: LONGINT)
1  258 --                                         : TMoveBoxCmd;
1  259 --
1  260 --          {Command Execution}
1  261 --            PROCEDURE TMoveBoxCmd.Perform(cmdPhase: TCmdPhase); OVERRIDE;
1  262 --            END;
1  263 --
1  264 --
1  265 --        TBoxProcess = SUBCLASS OF TProcess
1  266 --
1  267 --          {Creation/Destruction}
1  268 --            FUNCTION  TBoxProcess.CREATE: TBoxProcess;
1  269 --            FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN)
1  270 --                                                : TDocManager; OVERRIDE;
1  271 --            END;
1  272 --
1  273 --
1  274 --        TBoxDocManager = SUBCLASS OF TDocManager
1  275 --
1  276 --          {Creation/Destruction}
1  277 --            FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
1  278 --                                              : TBoxDocManager;
1  279 --            FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID): TWindow; OVERRIDE;
1  280 --            END;
1  281 --
1  282 --
1  283 --        TBoxWindow = SUBCLASS OF TWindow
1  284 --
1  285 --          {Creation/Destruction}
1  286 --            FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
1  287 --
1  288 --          {Document Creation}
1  289 --            PROCEDURE {TBoxWindow.} BlankStationery; OVERRIDE;
1  290 --
1  291 --          {Commands}
1  292 --            FUNCTION  TBoxWindow.NewCommand(cmdNumber: TCmdNumber): TCommand; OVERRIDE;
1  293 --            FUNCTION  TBoxWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN; OVERRIDE;
1  294 --            END;
1  295 --
1  296 --
1  297 --
1  298 --    IMPLEMENTATION
1  299 --
1  300 --      {$I U8Boxer2.text}
2    1 --    {U8BOXER2}
2    2 --
2    3 --    METHODS OF TBox;
2    4 --
2    5 -- A      FUNCTION  TBox.CREATE(object: TObject; itsHeap: THeap): TBox;
2    6 0- A      BEGIN
2    7 --            {$IFC fTrace}BP(11);{$ENDC}
2    8 --            SELF := NewObject(itsHeap, THISCLASS);
2    9 --            WITH SELF DO
2   10 1-                BEGIN
2   11 --                shapeLRect := zeroLRect;
2   12 --                color := colorGray;
2   13 -1                END;
2   14 --            {$IFC fTrace}EP;{$ENDC}
2   15 -0 A      END;
2   16 --
2   17 --      {$IFC fDebugMethods}
2   18 -- A      PROCEDURE TBox.Fields(PROCEDURE Field(nameAndType: S255));
2   19 0- A      BEGIN
2   20 --            Field('shapeLRect: LRect');
2   21 --            Field('color: INTEGER');
2   22 --            Field('');
2   23 -0 A      END;
2   24 --      {$ENDC}
2   25 --
2   26 --
2   27 --        {This draws a particular box}
2   28 -- A      PROCEDURE TBox.Draw;
2   29 --        VAR lPat:   LPattern;
2   30 0- A      BEGIN
```

```
2   31 --              {$IFC fTrace}BP(10);{$ENDC}
2   32 --              PenNormal;
2   33 --
2   34 --              IF LRectIsVisible(SELF.shapeLRect) THEN {this box needs to be drawn}
2   35 1-                  BEGIN
2   36 --                {Get a Quickdraw pattern to represent the box's color}
2   37 2-                  CASE SELF.color OF
2   38 --                      colorWhite:       lPat := lPatWhite;
2   39 --                      colorLtGray:      lPat := lPatLtGray;
2   40 --                      colorGray:        lPat := lPatGray;
2   41 --                      colorDkGray:      lPat := lPatDkGray;
2   42 --                      colorBlack:       lPat := lPatBlack;
2   43 --                      OTHERWISE         lPat := lPatWhite; {this case should not happen}
2   44 -2                      END;
2   45 --
2   46 --                 {Fill the box with the pattern, and draw a frame around it}
2   47 --                  FillLRect(SELF.shapeLRect, lPat);
2   48 --                  FrameLRect(SELF.shapeLRect);
2   49 -1                  END;
2   50 --              {$IFC fTrace}EP;{$ENDC}
2   51 -0 A        END;
2   52 --
2   53 --          { Frame a particular box}
2   54 -- A        PROCEDURE TBox.DrawFrame;
2   55 0- A        BEGIN
2   56 --              {$IFC fTrace}BP(10);{$ENDC}
2   57 --              PenNormal;
2   58 --              PenMode(PatXOr);
2   59 --              FrameLRect(SELF.shapeLRect);
2   60 --              {$IFC fTrace}EP;{$ENDC}
2   61 -0 A        END;
2   62 --
2   63 --
2   64 -- A        PROCEDURE TBox.MoveBox(boxView: TBoxView; deltaLPt: LPoint);
2   65 0- A        BEGIN
2   66 --              {$IFC fTrace}BP(10);{$ENDC}
2   67 --              WITH SELF DO
2   68 1-          {$H-}   BEGIN
2   69 --                  boxView.InvalBox(shapeLRect);
2   70 --                  OffsetLRect(shapeLRect, deltaLPt.h, deltaLPt.v);
2   71 --                  boxView.InvalBox(shapeLRect);
2   72 -1          {$H+}   END;
2   73 --              {$IFC fTrace}EP;{$ENDC}
2   74 -0 A        END;
2   75 --
2   76 --
2   77 --           {This calls the DoToHandle Procedure once for each handle LRect; user of this method must
2   78 --            set up the pen pattern and mode before calling}
2   79 -- A        PROCEDURE TBox.PaintHandles;
2   80 --          VAR hLRect,
2   81 --              shapeLRect: LRect;
2   82 --              dh, dv:       LONGINT;
2   83 --
2   84 -- B            PROCEDURE MoveHandleAndPaint(hOffset, vOffset: LONGINT);
2   85 0- B            BEGIN
2   86 --                  OffsetLRect(hLRect, hOffset, vOffset);
2   87 --                  PaintLRect(hLRect);
2   88 -0 B            END;
2   89 --
2   90 0- A        BEGIN
2   91 --              {$IFC fTrace}BP(10);{$ENDC}
2   92 --              IF NOT EmptyLRect(SELF.shapeLRect) THEN
2   93 1-                 BEGIN
2   94 --                 SetLRect(hLRect, -3, -2, 3, 2);
2   95 --                 shapeLRect := SELF.shapeLRect;
2   96 --                 WITH shapeLRect DO
2   97 2-                     BEGIN
2   98 --                     dh := right - left;
2   99 --                     dv := bottom - top;
2  100 --                     MoveHandleAndPaint(left, top);     {draw top left handle}
2  101 -2                     END;
2  102 --                 MoveHandleAndPaint(dh, 0);            {then top right}
2  103 --                 MoveHandleAndPaint(0, dv);            {then bottom right}
2  104 --                 MoveHandleAndPaint(-dh, 0);           {finally bottom left}
2  105 -1                 END;
2  106 --              {$IFC fTrace}EP;{$ENDC}
2  107 -0 A        END;
2  108 --
2  109 --      END;
2  110 --
2  111 --
2  112 --
2  113 --      METHODS OF TBoxView;
2  114 --
2  115 -- A        FUNCTION TBoxView.CREATE(object: TObject; itsHeap: THeap; itsPanel: TPanel; itsExtent: LRect)
2  116 --                                      : TBoxView;
2  117 0- A        BEGIN
2  118 --              {$IFC fTrace}BP(11);{$ENDC}
2  119 --              IF object = NIL THEN
2  120 --                 object := NewObject(itsHeap, THISCLASS);
2  121 --               SELF := TBoxView(itsPanel.NewView(object, itsExtent, TPrintManager.CREATE(NIL, itsHeap),
2  122 --                                                  stdMargins, TRUE));
2  123 --              {$IFC fTrace}EP;{$ENDC}
2  124 -0 A        END;
2  125 --
2  126 --
2  127 --      {$IFC fDebugMethods}
2  128 -- A        PROCEDURE TBoxView.Fields(PROCEDURE Field(nameAndType: S255));
2  129 0- A        BEGIN
2  130 --              SUPERSELF.Fields(Field);
2  131 --              Field('boxList: TList');
2  132 -0 A        END;
2  133 --      {$ENDC}
2  134 --
2  135 --
2  136 --              {This returns the box containing a certain point}
2  137 -- A        FUNCTION TBoxView.BoxWith(LPt: LPoint): TBox;
2  138 -- B            PROCEDURE FindBox(obj: TObject);
2  139 --              VAR box:     TBox;
2  140 0- B            BEGIN
```

```
2  141 --                   box := TBox(obj);
2  142 --                   IF LPtInLRect(LPt, box.shapeLRect) THEN
2  143 --                       BoxWith := box;             {last one found (front one) is returned}
2  144 -0 B             END;
2  145 0- A         BEGIN
2  146 --               {$IFC fTrace}BP(11);{$ENDC}
2  147 --               boxWith := NIL;
2  148 --               SELF.EachVirtualPart(FindBox);
2  149 --               {$IFC fTrace}EP;{$ENDC}
2  150 -0 A         END;
2  151 --
2  152 --
2  153 --               {This draws the list of boxes}
2  154 -- A         PROCEDURE TBoxView.Draw;
2  155 -- B             PROCEDURE DrawBox(obj: TObject);
2  156 --               VAR box:    TBox;
2  157 0- B             BEGIN
2  158 --                   box := TBox(obj);
2  159 --                   box.Draw;
2  160 -0 B             END;
2  161 0- A         BEGIN
2  162 --               {$IFC fTrace}BP(10);{$ENDC}
2  163 --               SELF.EachVirtualPart(DrawBox);
2  164 --               {$IFC fTrace}EP;{$ENDC}
2  165 -0 A         END;
2  166 --
2  167 --
2  168 -- A         PROCEDURE TBoxView.EachActualPart(PROCEDURE DoToObject(filteredObj: TObject));
2  169 0- A         BEGIN
2  170 --               {$IFC fTrace}BP(11);{$ENDC}
2  171 --               SELF.boxList.Each(DoToObject);
2  172 --               {$IFC fTrace}EP;{$ENDC}
2  173 -0 A         END;
2  174 --
2  175 --
2  176 --                {This determines which type of selection to create}
2  177 -- A          PROCEDURE TBoxView.MousePress(mouseLPt: LPoint);
2  178 --            VAR aSelection: TSelection;
2  179 --                panel:      TPanel;
2  180 --                box:        TBox;
2  181 --
2  182 0- A          BEGIN
2  183 --               {$IFC fTrace}BP(11);{$ENDC}
2  184 --               panel := SELF.panel;
2  185 --               panel.Highlight(panel.selection, hOntoOff);           {Turn off the old highlighting}
2  186 --               box := SELF.BoxWith(mouseLPt);         {Find the box the user clicked on}
2  187 --
2  188 --                IF box = NIL THEN
2  189 --                   {Create an instance of TCreateBoxSelection}
2  190 --                   aSelection := panel.selection.FreedAndReplacedBy(
2  191 --                                       TCreateBoxSelection.CREATE(NIL, SELF.heap, SELF, mouseLPt))
2  192 --               ELSE
2  193 --                   {Create an instance of TBoxSelection}
2  194 --                   aSelection := panel.selection.FreedAndReplacedBy(
2  195 --                               TBoxSelection.CREATE(NIL, SELF.heap, SELF, box, boxSelectionKind, mouseLPt));
2  196 --
2  197 --               panel.Highlight(panel.selection, hOffToOn);     {Turn on the highlighting for the newly selected box}
2  198 --               {$IFC fTrace}EP;{$ENDC}
2  199 -0 A          END;
2  200 --
2  201 --
2  202 -- A         PROCEDURE  TBoxView.InvalBox(invalLRect: LRect);
2  203 0- A         BEGIN
2  204 --               {$IFC fTrace}BP(10);{$ENDC}
2  205 --               InsetLRect(invalLRect, -3, -2);
2  206 --               SELF.panel.InvalLRect(invalLRect);
2  207 --               {$IFC fTrace}EP;{$ENDC}
2  208 -0 A         END;
2  209 --
2  210 --
2  211 -- A         PROCEDURE TBoxView.InitBoxList (itsHeap: THeap);
2  212 --           VAR  boxList: TList;
2  213 0- A         BEGIN
2  214 --             {$IFC fTrace}BP(11);{$ENDC}
2  215 --               boxList := TList.CREATE(NIL, itsHeap, 0);
2  216 --               SELF.boxList := boxList;
2  217 --               {$IFC fTrace}EP;{$ENDC}
2  218 -0 A         END;
2  219 --
2  220 --
2  221 -- A         FUNCTION TBoxView.NoSelection: TSelection;
2  222 0- A         BEGIN
2  223 --             {$IFC fTrace}BP(11);{$ENDC}
2  224 --             NoSelection := TBoxSelection.CREATE(NIL, SELF.Heap, SELF, NIL, nothingKind, zeroLPt);
2  225 --               {$IFC fTrace}EP;{$ENDC}
2  226 -0 A         END;
2  227 --
2  228 --     END;
2  229 --
2  230 --
2  231 --
2  232 --     METHODS OF TBoxSelection;
2  233 --
2  234 -- A         FUNCTION  TBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView; itsBox: TBox;
2  235 --                                   itsKind: INTEGER; itsAnchorLPt: LPoint): TBoxSelection;
2  236 0- A         BEGIN
2  237 --               {$IFC fTrace}BP(11);{$ENDC}
2  238 --               IF object = NIL THEN
2  239 --                   object := NewObject(itsHeap, THISCLASS);
2  240 --               SELF := TBoxSelection(TSelection.CREATE(object, itsHeap, itsView, itsKind, itsAnchorLPt));
2  241 --
2  242 --               SELF.box := itsBox;
2  243 --               {$IFC fTrace}EP;{$ENDC}
2  244 -0 A         END;
2  245 --
2  246 --      {$IFC fDebugMethods}
2  247 -- A         PROCEDURE TBoxSelection.Fields(PROCEDURE Field(nameAndType: S255));
2  248 0- A         BEGIN
2  249 --               SUPERSELF.Fields(Field);
2  250 --               Field('box: TBox');
```

```
2  251 -0 A      END;
2  252 --     {$ENDC}
2  253 --
2  254 --
2  255 --            {This draws the handles on the selected box}
2  256 -- A      PROCEDURE TBoxSelection.Highlight(highTransit: THighTransit);
2  257 0- A      BEGIN
2  258 --          {$IFC fTrace}BP(11);{$ENDC}
2  259 --          IF SELF.kind <> nothingKind THEN
2  260 1-              BEGIN
2  261 --              thePad.SetPenToHighlight(highTransit);   {set the drawing mode according to desired highlighting}
2  262 --              SELF.box.PaintHandles;                   {draw the handles on the box}
2  263 -1              END;
2  264 --          {$IFC fTrace}EP;{$ENDC}
2  265 -0 A      END;
2  266 --
2  267 --
2  268 -- A      PROCEDURE TBoxSelection.KeyClear;
2  269 0- A      BEGIN
2  270 --          {$IFC fTrace}BP(12);{$ENDC}
2  271 --          SELF.window.PerformCommand(TClearCmd.CREATE(NIL, SELF.Heap, uClear, TBoxView(SELF.view), SELF.box));
2  272 --          {$IFC fTrace}EP;{$ENDC}
2  273 -0 A      END;
2  274 --
2  275 --
2  276 -- A      PROCEDURE TBoxSelection.KeyTab;
2  277 --        VAR thisPanel:   TPanel;
2  278 --            nextBox:     TBox;
2  279 --            boxView:     TBoxView;
2  280 --            GetTheNextBox:  BOOLEAN;
2  281 --
2  282 -- B          PROCEDURE  DoToObject(filteredObj: TObject);
2  283 --            VAR box:     TBox;
2  284 0- B          BEGIN
2  285 --                box := TBox(filteredObj);
2  286 --                IF GetTheNextBox = TRUE THEN
2  287 1-                    BEGIN
2  288 --                    nextBox := box;
2  289 --                    GetTheNextBox := FALSE;
2  290 -1                    END;
2  291 --                IF box = SELF.box THEN
2  292 --                    GetTheNextBox := TRUE;
2  293 -0 B          END;
2  294 --
2  295 0- A      BEGIN
2  296 --          {$IFC fTrace}BP(12);{$ENDC}
2  297 --          thisPanel := SELF.panel;
2  298 --          boxView := TBoxView(SELF.view);
2  299 --          IF SELF.kind = nothingKind THEN
2  300 --              SELF.cantDoIt
2  301 --          ELSE
2  302 1-              BEGIN
2  303 --              GetTheNextBox := FALSE;
2  304 --              nextBox := NIL;
2  305 --              boxView.EachVirtualPart(DoToObject);
2  306 --              IF nextBox = NIL THEN
2  307 2-                  BEGIN
2  308 --                  GetTheNextBox := TRUE;
2  309 --                  boxView.EachVirtualPart(DoToObject);
2  310 -2                  END;
2  311 -1              END;
2  312 --          thisPanel.Highlight(SELF, hOnToOff);
2  313 --          SELF.box := nextBox;
2  314 --          thisPanel.Highlight(SELF, hOffToOn);
2  315 --          {$IFC fTrace}EP;{$ENDC}
2  316 -0 A      END;
2  317 --
2  318 --
2  319 --            {This is called when the user moves the mouse after pressing the button}
2  320 -- A      PROCEDURE TBoxSelection.MouseMove(mouseLPt: LPoint);
2  321 --        VAR diffLPt:     LPoint;
2  322 0- A      BEGIN
2  323 --          {$IFC fTrace}BP(11);{$ENDC}
2  324 --              {How far did mouse move?}
2  325 --          LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
2  326 --
2  327 --          {Move it if delta is nonzero}
2  328 --          IF NOT EqualLPt(diffLPt, zeroLPt) THEN
2  329 1-              BEGIN
2  330 --              SELF.currLPt := mouseLPt;
2  331 --              SELF.box.MoveBox(TBoxView(SELF.view), diffLPt);
2  332 -1              END;
2  333 --          {$IFC fTrace}EP;{$ENDC}
2  334 -0 A      END;
2  335 --
2  336 --
2  337 -- A      PROCEDURE TBoxSelection.MouseRelease;
2  338 --        VAR deltaLPt:    LPoint;
2  339 0- A      BEGIN
2  340 --          {$IFC fTrace}BP(11);{$ENDC}
2  341 --              { If the mouse moved then commit any outstanding command }
2  342 --          IF NOT EqualLPt(SELF.currLPt, SELF.anchorLPt) THEN
2  343 1-              BEGIN
2  344 --              LPtMinusLPt(SELF.currLPt, SELF.anchorLPt, deltaLPt);
2  345 --              SELF.window.PerformCommand(TMoveBoxCmd.CREATE(NIL, SELF.Heap, uMoveBox, TBoxView(SELF.view),
2  346 --                                                            SELF.box, deltaLPt.h, deltaLPt.v));
2  347 -1              END;
2  348 --          {$IFC fTrace}EP;{$ENDC}
2  349 -0 A      END;
2  350 --
2  351 --
2  352 -- A      FUNCTION  TBoxSelection.NewCommand(cmdNumber: TCmdNumber): TCommand;
2  353 --        VAR boxView:     TBoxView;
2  354 --            heap:        THeap;
2  355 --            pasteH, pasteV:  LONGINT;
2  356 0- A      BEGIN
2  357 --          {$IFC fTrace}BP(11);{$ENDC}
2  358 --
2  359 --          boxView := TBoxView(SELF.view);
2  360 --          heap := SELF.Heap;
```

```
2  361 --
2  362 1-           CASE cmdNumber OF
2  363 --               uWhite, uLtGray, uGray, uDkGray, uBlack:
2  364 --                   NewCommand := TRecolorCmd.CREATE(NIL, heap, cmdNumber, boxView, SELF.box,
2  365 --                                             cmdNumber - uWhite + colorWhite);
2  366 --               uClear:
2  367 --                   NewCommand := TClearCmd.CREATE(NIL, heap, cmdNumber, boxView, SELF.box);
2  368 --
2  369 --               uCut, uCopy:
2  370 --                   NewCommand := TBoxCutCopyCmd.CREATE(NIL, heap, cmdNumber, boxView, cmdNumber = uCut,
2  371 --                                             SELF.box);
2  372 --               uPaste:
2  373 2-                   BEGIN
2  374 --                   clipboard.Inspect;
2  375 --                   IF clipboard.hasView THEN
2  376 3-                       BEGIN
2  377 --                       WITH SELF DO
2  378 --                           IF kind = nothingKind THEN
2  379 4-                               BEGIN
2  380 --                               pasteH := boxView.clickLPt.h;
2  381 --                               pastev := boxView.clickLPt.v;
2  382 -4                               END
2  383 --                           ELSE
2  384 --                               WITH box.shapeLRect DO
2  385 4-                                   BEGIN
2  386 --                                   pasteH := (left + right) DIV 2;
2  387 --                                   pasteV := (top + bottom) DIV 2;
2  388 -4                                   END;
2  389 --                       NewCommand := TBoxPasteCmd.CREATE(NIL, heap, cmdNumber, boxView, pasteH, pasteV);
2  390 -3                       END
2  391 --                   ELSE
2  392 --                       process.Stop(phUnkClip);
2  393 -2                   END;
2  394 --
2  395 --               uDuplicate:
2  396 --                   NewCommand := TDuplicateCmd.CREATE(NIL, heap, cmdNumber, boxView, SELF.box);
2  397 --
2  398 --               OTHERWISE
2  399 --                   NewCommand := SUPERSELF.NewCommand(cmdNumber);
2  400 -1               END;
2  401 --           {$IFC fTrace}EP;{$ENDC}
2  402 -0 A      END;
2  403 --
2  404 --
2  405 -- A      FUNCTION TBoxSelection.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
2  406 0- A      BEGIN
2  407 --           {$IFC fTrace}BP(11);{$ENDC}
2  408 1-           CASE cmdNumber OF
2  409 --               uWhite, uLtGray, uGray, uDkGray, uBlack,
2  410 --               uClear,
2  411 --               uDuplicate,
2  412 --               uCut, uCopy:
2  413 --                   CanDoCommand := SELF.kind <> nothingKind;
2  414 --
2  415 --               uPaste:
2  416 2-                   BEGIN
2  417 --                   clipboard.Inspect;
2  418 --                   CanDoCommand := clipboard.hasView;
2  419 -2                   END;
2  420 --
2  421 --               OTHERWISE
2  422 --                   CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
2  423 -1               END;
2  424 --           {$IFC fTrace}EP;{$ENDC}
2  425 -0 A      END;
2  426 --
2  427 --   END;
2  428 --
2  429 --
2  430 --
2  431 --   METHODS OF TCreateBoxSelection;
2  432 --
2  433 -- A      FUNCTION TCreateBoxSelection.CREATE(object: TObject; itsHeap: THeap; itsView: TView;
2  434 --                                   itsAnchorLPt: LPoint): TCreateBoxSelection;
2  435 --       VAR box: TBox;
2  436 0- A      BEGIN
2  437 --           {$IFC fTrace}BP(11);{$ENDC}
2  438 --           IF object = NIL THEN
2  439 --               object := NewObject(itsHeap, THISCLASS);
2  440 --           SELF := TCreateBoxSelection(TSelection.CREATE(object, itsHeap, itsView, createBoxSelectionKind,
2  441 --                                   itsAnchorLPt));
2  442 --           box := TBox.CREATE(NIL, SELF.heap);
2  443 --           SELF.box := box;
2  444 --           {$IFC fTrace}EP;{$ENDC}
2  445 -0 A      END;
2  446 --
2  447 --
2  448 --    {$IFC fDebugMethods}
2  449 -- A      PROCEDURE TCreateBoxSelection.Fields(PROCEDURE Field(nameAndType: S255));
2  450 0- A      BEGIN
2  451 --           SUPERSELF.Fields(Field);
2  452 --           Field('box: TBox');
2  453 -0 A      END;
2  454 --    {$ENDC}
2  455 --
2  456 --
2  457 --       {This is called when the user moves the mouse after pressing the button}
2  458 -- A      PROCEDURE TCreateBoxSelection.MouseMove(mouseLPt: LPoint);
2  459 --       VAR maxBoxLRect:    LRect;
2  460 --           diffLPt:        LPoint;
2  461 --           boxView:        TBoxView;
2  462 --           box:            TBox;
2  463 --
2  464 -- B          PROCEDURE DrawTheFrame;
2  465 0- B          BEGIN
2  466 --               box.DrawFrame;
2  467 -0 B          END;
2  468 --
2  469 0- A      BEGIN
2  470 --           {$IFC fTrace}BP(11);{$ENDC}
```

```
2  471 --           boxView := TBoxView(SELF.view);
2  472 --           box := SELF.box;
2  473 --
2  474 --           { In Boxer it is possible to draw a box greater than allowed by a 16 bit rectangle. These three
2  475 --             lines force the rectangle to within 16 bits. }
2  476 --       {$H-} WITH SELF.anchorLPt DO
2  477 --               SetLRect(maxBoxLRect, h+10-MAXINT, v+10-MAXINT, h+MAXINT-10, v+MAXINT-10);
2  478 --       {$H+} LRectHaveLPt(maxBoxLRect, mouseLPt);
2  479 --
2  480 --           LPtMinusLPt(mouseLPt, SELF.currLPt, diffLPt);
2  481 --           IF NOT EqualLPt(diffLPt, zeroLPt) THEN
2  482 1-               BEGIN
2  483 --               SELF.currLPt := mouseLPt;
2  484 --
2  485 --               boxView.panel.OnAllPadsDo(DrawTheFrame);
2  486 --               WITH box DO
2  487 2-                   BEGIN
2  488 --                   shapeLRect.topLeft := SELF.anchorLPt;
2  489 --                   shapeLRect.botRight := mouseLPt;
2  490 -2                   END;
2  491 --
2  492 --       {$H-}   RectifyLRect(box.shapeLRect);   {$H+}
2  493 --               boxView.panel.OnAllPadsDo(DrawTheFrame);
2  494 -1               END;
2  495 --           {$IFC fTrace}EP;{$ENDC}
2  496 -0 A      END;
2  497 --
2  498 --
2  499 -- A      PROCEDURE TCreateBoxSelection.MouseRelease;
2  500 --        VAR thisBox:        TBox;
2  501 --            boxView:        TBoxView;
2  502 --            drawnLRect:     LRect;
2  503 --            aSelection:     TSelection;
2  504 --            panel:          TPanel;
2  505 --
2  506 -- B          PROCEDURE DrawTheFrame;
2  507 0- B          BEGIN
2  508 --                thisBox.DrawFrame;
2  509 -0 B          END;
2  510 --
2  511 0- A      BEGIN
2  512 --            {$IFC fTrace}BP(11);{$ENDC}
2  513 --            boxView := TBoxView(SELF.view);
2  514 --            panel := boxView.panel;
2  515 --            thisBox := SELF.box;
2  516 --            panel.OnAllPadsDo(DrawTheFrame);
2  517 --            drawnLRect := thisBox.shapeLRect;
2  518 --
2  519 --            { Independant of whether we threw the boxed away or not we must create an instance of TBoxSelection
2  520 --              to replace the now useless instance of TCreateBoxSelection using the kind set above. }
2  521 --            aSelection := SELF.FreedAndReplaceby(
2  522 --                            TBoxSelection.CREATE(NIL, SELF.heap, boxView, thisBox, boxSelectionKind,
2  523 --                                                 drawnLRect.topleft));
2  524 --
2  525 --            boxView.InvalBox(drawnLRect);
2  526 --
2  527 --            {If the box is not big enough then throw it away, otherwise put it in the list}
2  528 --            IF (drawnLRect.right - drawnLRect.left <=4) OR (drawnLRect.bottom - drawnLRect.top <=4) THEN
2  529 1-                BEGIN
2  530 --                aSelection.kind := nothingKind;
2  531 --                thisBox.Free;
2  532 -1                END
2  533 --            ELSE
2  534 --                panel.window.PerformCommand(TCreateCmd.CREATE(NIL, SELF.Heap, uCreateBox, boxView, thisBox));
2  535 --            {$IFC fTrace}EP;{$ENDC}
2  536 -0 A      END;
2  537 --
2  538 --    END;
2  539 --
2  540 --
2  541 --
2  542 --
2  543 --    METHODS OF TRecolorCmd;
2  544 --
2  545 -- A      FUNCTION  TRecolorCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  546 --                              itsView: TBoxView; itsBox: TBox; itsColor: TColor): TRecolorCmd;
2  547 0- A      BEGIN
2  548 --            {$IFC fTrace}BP(10);{$ENDC}
2  549 --            IF object = NIL THEN
2  550 --                object := NewObject(itsHeap, THISCLASS);
2  551 --            SELF := TRecolorCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealAll));
2  552 --            SELF.color := itsColor;
2  553 --            SELF.box := itsBox;
2  554 --            {$IFC fTrace}EP;{$ENDC}
2  555 -0 A      END;
2  556 --
2  557 --
2  558 --    {$IFC fDebugMethods}
2  559 -- A      PROCEDURE TRecolorCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  560 0- A      BEGIN
2  561 --            TCommand.Fields(Field);
2  562 --            Field('Color: INTEGER');
2  563 --            Field('box: TBox');
2  564 -0 A      END;
2  565 --     {$ENDC}
2  566 --
2  567 --
2  568 -- A      PROCEDURE TRecolorCmd.Commit;
2  569 0- A      BEGIN
2  570 --            {$IFC fTrace}BP(12);{$ENDC}
2  571 --            SELF.box.color := SELF.color;
2  572 --            {$IFC fTrace}EP;{$ENDC}
2  573 -0 A      END;
2  574 --
2  575 --
2  576 -- A      PROCEDURE TRecolorCmd.Perform(cmdPhase: TCmdPhase);
2  577 0- A      BEGIN
2  578 --            {$IFC fTrace}BP(12);{$ENDC}
2  579 --            TBoxView(SELF.image).InvalBox(SELF.box.shapeLRect);
2  580 --
```

```
2  581 --            self.image.view.panel.selection.MarkChanged;    {allow this document to be saved}
2  582 --            {$IFC fTrace}EP;{$ENDC}
2  583 -0 A       END;
2  584 --
2  585 --
2  586 -- A       PROCEDURE TRecolorCmd.FilterAndDo(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject));
2  587 --            VAR saveColor:   TColor;
2  588 --                box:         TBox;
2  589 0- A       BEGIN
2  590 --            {$IFC fTrace}BP(12);{$ENDC}
2  591 --            box := TBox(actualObj);
2  592 --            IF box = SELF.box THEN
2  593 1-                BEGIN
2  594 --                saveColor := box.color;
2  595 --                box.color := SELF.Color;
2  596 --                DoToObject(box);
2  597 --                box.color := saveColor;
2  598 -1                END
2  599 --            ELSE
2  600 --                DoToObject(box);
2  601 --            {$IFC fTrace}EP;{$ENDC}
2  602 -0 A       END;
2  603 --
2  604 --    END;
2  605 --
2  606 --
2  607 --
2  608 --    METHODS OF TCreateCmd;
2  609 --
2  610 -- A       FUNCTION  TCreateCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  611 --                                     itsView: TBoxView; itsBox: TBox): TCreateCmd;
2  612 0- A       BEGIN
2  613 --            {$IFC fTrace}BP(10);{$ENDC}
2  614 --            IF object = NIL THEN
2  615 --                object := NewObject(itsHeap, THISCLASS);
2  616 --            SELF := TCreateCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealAll));
2  617 --            SELF.newBox := itsBox;
2  618 --            {$IFC fTrace}EP;{$ENDC}
2  619 -0 A       END;
2  620 --
2  621 -- A       PROCEDURE TCreateCmd.Free;
2  622 0- A       BEGIN
2  623 --            {$IFC fTrace}BP(10);{$ENDC}
2  624 --            Free(SELF.newBox);
2  625 --            SELF.FreeObject;
2  626 --            {$IFC fTrace}EP;{$ENDC}
2  627 -0 A       END;
2  628 --
2  629 --
2  630 --     {$IFC fDebugMethods}
2  631 -- A       PROCEDURE TCreateCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  632 0- A       BEGIN
2  633 --            SUPERSELF.Fields(Field);
2  634 --            Field('newBox: TBox');
2  635 -0 A       END;
2  636 --     {$ENDC}
2  637 --
2  638 --
2  639 -- A       PROCEDURE TCreateCmd.Commit;
2  640 --         VAR boxView:     TBoxView;
2  641 0- A       BEGIN
2  642 --            {$IFC fTrace}BP(12);{$ENDC}
2  643 --            TBoxView(SELF.image).boxList.InsLast(SELF.newBox);
2  644 --            SELF.newBox := NIL;
2  645 --            {$IFC fTrace}EP;{$ENDC}
2  646 -0 A       END;
2  647 --
2  648 -- A       PROCEDURE TCreateCmd.UpdateSelection(thisSelection: TBoxSelection; cmdPhase: TCmdPhase);
2  649 0- A       BEGIN
2  650 --            {$IFC fTrace}BP(13);{$ENDC}
2  651 --            WITH thisSelection DO
2  652 1-               CASE cmdPhase OF
2  653 --                   doPhase, redoPhase:
2  654 --                         kind := boxSelectionKind;
2  655 --                   undoPhase:
2  656 --                         kind := nothingKind;
2  657 -1               END {CASE};
2  658 --            {$IFC fTrace}EP;{$ENDC}
2  659 -0 A       END;
2  660 --
2  661 --
2  662 -- A       PROCEDURE TCreateCmd.Perform(cmdPhase: TCmdPhase);
2  663 --         VAR boxView:            TBoxView;
2  664 --             box:                TBox;
2  665 --             thisSelection:      TBoxSelection;
2  666 0- A       BEGIN
2  667 --            {$IFC fTrace}BP(12);{$ENDC}
2  668 --            boxView := TBoxView(SELF.image);
2  669 --            thisSelection := TBoxSelection(boxView.panel.selection);
2  670 --            SELF.UpdateSelection(thisSelection, cmdPhase);
2  671 --            boxView.InvalBox(SELF.newBox.shapeLRect);
2  672 --
2  673 --            self.image.view.panel.selection.MarkChanged;    {allow this document to be saved}
2  674 --            {$IFC fTrace}EP;{$ENDC}
2  675 -0 A       END;
2  676 --
2  677 --
2  678 -- A       PROCEDURE TCreateCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
2  679 0- A       BEGIN
2  680 --            {$IFC fTrace}BP(12);{$ENDC}
2  681 --            TBoxView(SELF.image).EachActualPart(DoToObject);
2  682 --            DoToObject(SELF.newBox);
2  683 --            {$IFC fTrace}EP;{$ENDC}
2  684 -0 A       END;
2  685 --
2  686 --    END;
2  687 --
2  688 --
2  689 --
2  690 --    METHODS OF TDuplicateCmd;
```

```
2  691 --
2  692 -- A     FUNCTION  TDuplicateCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  693 --                                      itsView: TBoxView; itsBox: TBox): TDuplicateCmd;
2  694 --       VAR newBox: TBox;
2  695 0- A     BEGIN
2  696 --           {$IFC fTrace}BP(10);{$ENDC}
2  697 --           IF object = NIL THEN
2  698 --               object := NewObject(itsHeap, THISCLASS);
2  699 --           SELF := TDuplicateCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealAll));
2  700 --           SELF.image.view.panel.window.CommitLast;
2  701 --           SELF.oldBox := itsBox;
2  702 --           newBox := TBox(itsBox.Clone(itsHeap));
2  703 --           SELF.newBox := newBox;
2  704 --       {$H-} OffSetLRect(newBox.shapeLRect, 20, 20); {$H+}
2  705 --           {$IFC fTrace}EP;{$ENDC}
2  706 -0 A     END;
2  707 --
2  708 --
2  709 --     {$IFC fDebugMethods}
2  710 -- A     PROCEDURE TDuplicateCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  711 0- A     BEGIN
2  712 --           SUPERSELF.Fields(Field);
2  713 --           Field('oldBox: TBox');
2  714 -0 A     END;
2  715 --     {$ENDC}
2  716 --
2  717 --
2  718 --
2  719 -- A     PROCEDURE TDuplicateCmd.UpdateSelection(thisSelection: TBoxSelection; cmdPhase: TCmdPhase);
2  720 0- A     BEGIN
2  721 --           {$IFC fTrace}BP(13);{$ENDC}
2  722 --           WITH thisSelection DO
2  723 1-               CASE cmdPhase OF
2  724 --                   doPhase, redoPhase:
2  725 --                           box := SELF.newBox;
2  726 --                   undoPhase:
2  727 --                           box := SELF.oldBox;
2  728 -1               END {CASE};
2  729 --           {$IFC fTrace}EP;{$ENDC}
2  730 -0 A     END;
2  731 --
2  732 --   END;
2  733 --
2  734 --
2  735 --
2  736 --   METHODS OF TBoxCutCopyCmd;
2  737 --
2  738 -- A     FUNCTION  TBoxCutCopyCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  739 --                                       itsView: TView; isCutCmd: BOOLEAN; itsBox: TBox): TBoxCutCopyCmd;
2  740 0- A     BEGIN
2  741 --           {$IFC fTrace}BP(10);{$ENDC}
2  742 --           IF object = NIL THEN
2  743 --               object := NewObject(itsHeap, THISCLASS);
2  744 --           SELF := TBoxCutCopyCmd(TCutCopyCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, isCutCmd));
2  745 --           SELF.box := itsBox;
2  746 --           {$IFC fTrace}EP;{$ENDC}
2  747 -0 A     END;
2  748 --
2  749 --
2  750 --       {$IFC fDebugMethods}
2  751 -- A     PROCEDURE TBoxCutCopyCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  752 0- A     BEGIN
2  753 --           SUPERSELF.Fields(Field);
2  754 --           Field('box: TBox');
2  755 -0 A     END;
2  756 --       {$ENDC}
2  757 --
2  758 --
2  759 -- A     PROCEDURE TBoxCutCopyCmd.Commit;
2  760 --       VAR s:          TListScanner;
2  761 --           box:        TBox;
2  762 0- A     BEGIN
2  763 --           {$IFC fTrace}BP(12);{$ENDC}
2  764 --           IF SELF.isCut THEN
2  765 1-               BEGIN
2  766 --               s := TBoxView(SELF.image).boxList.Scanner;
2  767 --               WHILE s.Scan(box) DO
2  768 --                   IF box = SELF.box THEN
2  769 2-                       BEGIN
2  770 --                       s.Delete(TRUE);
2  771 --                       s.Done;
2  772 -2                       END;
2  773 -1               END;
2  774 --           {$IFC fTrace}EP;{$ENDC}
2  775 -0 A     END;
2  776 --
2  777 --
2  778 -- A     PROCEDURE TBoxCutCopyCmd.DoCutCopy(clipSelection: TSelection; deleteOriginal: BOOLEAN;
2  779 --                                          cmdPhase: TCmdPhase);
2  780 --       VAR boxView:            TBoxView;
2  781 --           thisBoxSelection:   TBoxSelection;
2  782 --           clipHeap:           THeap;
2  783 --           clipBoxList:        TList;
2  784 --           clipBoxView:        TBoxView;
2  785 --           clipBoxSelection:   TSelection;
2  786 --           clipBox:            TBox;
2  787 --           deltaH:             LONGINT;
2  788 --           deltaV:             LONGINT;
2  789 --
2  790 --
2  791 0- A     BEGIN
2  792 --           {$IFC fTrace}BP(12);{$ENDC}
2  793 --           boxView := TBoxView(SELF.image);
2  794 --
2  795 --           IF cmdPhase = doPhase THEN
2  796 1-               BEGIN
2  797 --
2  798 --               {prepare to copy}
2  799 --               clipHeap := clipSelection.Heap;
2  800 --
```

```
2  801 --              deltaH := SELF.box.shapeLRect.left;
2  802 --              deltaV := SELF.box.shapeLRect.top;
2  803 --
2  804 --                 {Copy the box to the scrap}
2  805 --              clipBox := TBox(SELF.box.Clone(clipHeap));
2  806 --      {$H-}   OffsetLRect(clipBox.shapeLRect, -deltaH, -deltaV);  {$H+}
2  807 --
2  808 --              clipBoxList := TList.CREATE(NIL, clipHeap, 0);
2  809 --              clipBoxList.InsLast(clipBox);
2  810 --
2  811 --                 {make new clipboard selection}
2  812 --              clipBoxView := TBoxView.CREATE(NIL, clipHeap, clipSelection.panel, clipBox.shapeLRect);
2  813 --              clipBoxView.boxList := clipBoxList;
2  814 --              clipBoxSelection := clipSelection.FreedAndReplacedBy(
2  815 --                      TBoxSelection.CREATE(NIL, clipHeap, clipBoxView, clipBox, boxSelectionKind, zeroLPt));
2  816 -1             END;
2  817 --
2  818 --          { If this is a cut then remake the selection and invalidate the cut box. }
2  819 --          IF SELF.isCut THEN
2  820 1-              BEGIN
2  821 --              thisBoxSelection := TBoxSelection(boxView.panel.selection);
2  822 --              WITH thisBoxSelection DO
2  823 2-                  CASE cmdPhase OF
2  824 --                      doPhase, redoPhase:
2  825 --                          Kind := nothingKind;
2  826 --                      undoPhase:
2  827 --                          Kind := boxSelectionKind;
2  828 -2                      END;
2  829 --       {$H-} boxView.InvalBox(SELF.box.shapeLRect);  {$H+}
2  830 -1              END;
2  831 --
2  832 --          self.image.view.panel.selection.MarkChanged;   {allow this document to be saved}
2  833 --          {$IFC fTrace}EP;{$ENDC}
2  834 -0 A     END;
2  835 --
2  836 --
2  837 -- A     PROCEDURE TBoxCutCopyCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
2  838 0- A     BEGIN
2  839 --          {$IFC fTrace}BP(12);{$ENDC}
2  840 --          IF SELF.isCut THEN
2  841 --              SUPERSELF.EachVirtualPart(DoToObject)
2  842 --          ELSE
2  843 --              SELF.image.EachActualPart(DoToObject);
2  844 --          {$IFC fTrace}EP;{$ENDC}
2  845 -0 A     END;
2  846 --
2  847 --
2  848 -- A     PROCEDURE TBoxCutCopyCmd.FilterAndDo(actualObj: TObject; PROCEDURE DoToObject(filteredObj: TObject));
2  849 --          VAR box: TBox;
2  850 0- A     BEGIN
2  851 --          {$IFC fTrace}BP(12);{$ENDC}
2  852 --          box := TBox(actualObj);
2  853 --          IF (box <> SELF.box) OR NOT SELF.isCut THEN
2  854 --              DoToObject(actualObj);
2  855 --          {$IFC fTrace}EP;{$ENDC}
2  856 -0 A     END;
2  857 --
2  858 --    END;
2  859 --
2  860 --
2  861 --
2  862 --    METHODS OF TBoxPasteCmd;
2  863 --
2  864 -- A     FUNCTION  TBoxPasteCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  865 --                                  itsView: TBoxView; itsH, itsV: LONGINT): TBoxPasteCmd;
2  866 0- A     BEGIN
2  867 --          {$IFC fTrace}BP(10);{$ENDC}
2  868 --          IF object = NIL THEN
2  869 --             object := NewObject(itsHeap, THISCLASS);
2  870 --          SELF := TBoxPasteCmd(TPasteCommand.CREATE(object, itsHeap, itsCmdNumber, itsView));
2  871 --          WITH SELF DO
2  872 1-              BEGIN
2  873 --              pasteH := itsH;
2  874 --              pasteV := itsV;
2  875 --              pasteBox := NIL;
2  876 -1              END;
2  877 --          {$IFC fTrace}EP;{$ENDC}
2  878 -0 A     END;
2  879 --
2  880 --
2  881 --     {$IFC fDebugMethods}
2  882 -- A     PROCEDURE TBoxPasteCmd.Fields(PROCEDURE Field(nameAndType: S255));
2  883 0- A     BEGIN
2  884 --          SUPERSELF.Fields(Field);
2  885 --          Field('pasteH: LONGINT');
2  886 --          Field('pasteV: LONGINT');
2  887 --          Field('pasteBox: TBox');
2  888 -0 A     END;
2  889 --     {$ENDC}
2  890 --
2  891 --
2  892 -- A     PROCEDURE TBoxPasteCmd.Free;
2  893 0- A     BEGIN
2  894 --          {$IFC fTrace}BP(12);{$ENDC}
2  895 --          Free(SELF.pasteBox);
2  896 --          SELF.FreeObject;
2  897 --          {$IFC fTrace}EP;{$ENDC}
2  898 -0 A     END;
2  899 --
2  900 --
2  901 -- A     PROCEDURE TBoxPasteCmd.Commit;
2  902 0- A     BEGIN
2  903 --          {$IFC fTrace}BP(12);{$ENDC}
2  904 --          TBoxView(SELF.image).boxList.InsLast(SELF.pasteBox);
2  905 --          SELF.pasteBox := NIL;
2  906 --          {$IFC fTrace}EP;{$ENDC}
2  907 -0 A     END;
2  908 --
2  909 --
2  910 -- A     PROCEDURE TBoxPasteCmd.DoPaste(clipSelection: TSelection; pic: PicHandle; cmdPhase: TCmdPhase);
```

```
2  911 --        VAR boxView:               TBoxView;
2  912 --            panel:                 TPanel;
2  913 --            clipBoxSelection:      TBoxSelection;
2  914 --            thisBoxSelection:      TBoxSelection;
2  915 --            clipBoxView:           TBoxView;
2  916 --            deltaH:                LONGINT;
2  917 --            deltaV:                LONGINT;
2  918 --            s:                     TListScanner;
2  919 --            clipBox:               TBox;
2  920 --            box:                   TBox;
2  921 --
2  922 0- A     BEGIN
2  923 --            {$IFC fTrace}BP(12);{$ENDC}
2  924 --            boxView :=TBoxView(SELF.image);
2  925 --            panel := boxView.panel;
2  926 --
2  927 --            IF cmdPhase = doPhase THEN
2  928 1-                BEGIN
2  929 --                    { If the clipboard selection is of class TBoxSelection then we can paste it into document,
2  930 --                      otherwise we have to do other things }
2  931 --                IF NOT InClass(clipSelection, TBoxSelection) THEN
2  932 --                    panel.selection.CantDoIt
2  933 --                ELSE
2  934 2-                    BEGIN
2  935 --                    clipBoxSelection := TBoxSelection(clipSelection);
2  936 --
2  937 --                        { Place the box around the point indicated by pasteH and pasteV }
2  938 --            {$H-}   WITH clipBoxSelection.box.shapeLRect DO {box.shapeLRect.topLeft = zeroLPt}
2  939 3-                        BEGIN
2  940 --                        deltaH := Max(0, Min(SELF.pasteH - (right DIV 2),
2  941 --                                              boxView.extentLRect.right - right));
2  942 --                        deltaV := Max(0, Min(SELF.pasteV - (bottom DIV 2),
2  943 --                                              boxView.extentLRect.bottom - bottom));
2  944 -3          {$H+}          END;
2  945 --
2  946 --                    box := TBox(clipBoxSelection.box.Clone(boxView.Heap));
2  947 --          {$H-}     OffsetLRect(box.shapeLRect, deltaH, deltaV);          {$H+}
2  948 --                    SELF.pasteBox := box;
2  949 -2                    END;
2  950 -1                END;
2  951 --
2  952 --            thisBoxSelection := TBoxSelection(panel.selection);
2  953 --            panel.Highlight(thisBoxSelection, hOnToOff);
2  954 --            WITH thisBoxSelection DO
2  955 1-                CASE cmdPhase OF
2  956 --                    doPhase, redoPhase:
2  957 2-                        BEGIN
2  958 --                        kind := boxSelectionKind;
2  959 --                        box := SELF.pasteBox;
2  960 -2                        END;
2  961 --
2  962 --                    undoPhase:
2  963 --                        kind := nothingKind;
2  964 -1                    END;
2  965 --
2  966 --        {$H-}   boxView.InvalBox(SELF.pasteBox.shapeLRect);   {$H+}
2  967 --
2  968 --             self.image.view.panel.selection.MarkChanged;     {allow this document to be saved}
2  969 --            {$IFC fTrace}EP;{$ENDC}
2  970 -0 A      END;
2  971 --
2  972 --
2  973 -- A      PROCEDURE TBoxPasteCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
2  974 0- A      BEGIN
2  975 --            {$IFC fTrace}BP(12);{$ENDC}
2  976 --            SELF.image.EachActualPart(DoToObject);
2  977 --            DoToObject(SELF.pasteBox);
2  978 --            {$IFC fTrace}EP;{$ENDC}
2  979 -0 A      END;
2  980 --
2  981 --    END;
2  982 --
2  983 --
2  984 --
2  985 --    METHODS OF TClearAllCmd;
2  986 --
2  987 -- A      FUNCTION  TClearAllCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2  988 --                                      itsView: TBoxView): TClearAllCmd;
2  989 0- A      BEGIN
2  990 --            {$IFC fTrace}BP(10);{$ENDC}
2  991 --            IF object = NIL THEN
2  992 --                object := NewObject(itsHeap, THISCLASS);
2  993 --            SELF := TClearAllCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealNone));
2  994 --            SELF.kind := SELF.image.view.panel.selection.kind;
2  995 --            {$IFC fTrace}EP;{$ENDC}
2  996 -0 A      END;
2  997 --
2  998 --
2  999 --     {$IFC fDebugMethods}
2 1000 -- A      PROCEDURE TClearAllCmd.Fields(PROCEDURE Field(nameAndType: S255));
2 1001 0- A      BEGIN
2 1002 --            TCommand.Fields(Field);
2 1003 --            Field('kind: INTEGER');
2 1004 -0 A      END;
2 1005 --     {$ENDC}
2 1006 --
2 1007 --
2 1008 -- A      PROCEDURE TClearAllCmd.Commit;
2 1009 0- A      BEGIN
2 1010 --            {$IFC fTrace}BP(12);{$ENDC}
2 1011 --            TBoxView(SELF.image).boxList.DelAll(TRUE);
2 1012 --            {$IFC fTrace}EP;{$ENDC}
2 1013 -0 A      END;
2 1014 --
2 1015 --
2 1016 -- A      PROCEDURE TClearAllCmd.Perform(cmdPhase: TCmdPhase);
2 1017 --        VAR thisSelection: TSelection;
2 1018 --            boxView:     TBoxView;
2 1019 0- A      BEGIN
2 1020 --            {$IFC fTrace}BP(12);{$ENDC}
```

```
2 1021 --          boxView := TBoxView(SELF.image);
2 1022 --          thisSelection := boxView.panel.selection;
2 1023 --
2 1024 --          WITH thisSelection DO
2 1025 1-             CASE cmdPhase OF
2 1026 --                 doPhase, redoPhase:
2 1027 --                        kind := nothingKind;
2 1028 --                 undoPhase:
2 1029 --                        kind := SELF.kind;
2 1030 -1             END;
2 1031 --
2 1032 --                  { Invalidate the whole panel }
2 1033 --          boxView.panel.Invalidate;
2 1034 --
2 1035 --          self.image.view.panel.selection.MarkChanged;    {allow this document to be saved}
2 1036 --          {$IFC fTrace}EP;{$ENDC}
2 1037 -0 A      END;
2 1038 --
2 1039 --
2 1040 -- A      PROCEDURE TClearAllCmd.EachVirtualPart(PROCEDURE DoToObject(filteredObj: TObject));
2 1041 0- A      BEGIN
2 1042 --          {$IFC fTrace}BP(12);{$ENDC}
2 1043 --          {$IFC fTrace}EP;{$ENDC}
2 1044 -0 A      END;
2 1045 --
2 1046 --    END;
2 1047 --
2 1048 --
2 1049 --
2 1050 --    METHODS OF TClearCmd;
2 1051 --
2 1052 -- A      FUNCTION  TClearCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2 1053 --                                  itsView: TBoxView; itsBox: TBox): TClearCmd;
2 1054 0- A      BEGIN
2 1055 --          {$IFC fTrace}BP(10);{$ENDC}
2 1056 --          IF object = NIL THEN
2 1057 --             object := NewObject(itsHeap, THISCLASS);
2 1058 --          SELF := TClearCmd(TBoxCutCopyCmd.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, itsBox));
2 1059 --          {$IFC fTrace}EP;{$ENDC}
2 1060 -0 A      END;
2 1061 --
2 1062 --
2 1063 -- A      PROCEDURE TClearCmd.Perform(cmdPhase: TCmdPhase);
2 1064 --        VAR boxView:          TBoxView;
2 1065 --            thisBoxSelection:  TBoxSelection;
2 1066 --            panel:             TPanel;
2 1067 0- A      BEGIN
2 1068 --          {$IFC fTrace}BP(12);{$ENDC}
2 1069 --          boxView := TBoxView(SELF.image);
2 1070 --          panel := boxView.panel;
2 1071 --          thisBoxSelection := TBoxSelection(panel.selection);
2 1072 --
2 1073 --          WITH thisBoxSelection DO
2 1074 1-              CASE cmdPhase OF
2 1075 --                  doPhase, redoPhase:
2 1076 --                      Kind := nothingKind;
2 1077 --                  undoPhase:
2 1078 --                      Kind := boxSelectionKind;
2 1079 -1                  END;
2 1080 --          boxView.InvalBox(SELF.box.shapeLRect);
2 1081 --
2 1082 --          self.image.view.panel.selection.MarkChanged;    {allow this document to be saved}
2 1083 --          {$IFC fTrace}EP;{$ENDC}
2 1084 -0 A      END;
2 1085 --
2 1086 --    END;
2 1087 --
2 1088 --
2 1089 --
2 1090 --    METHODS OF TMoveBoxCmd;
2 1091 --
2 1092 -- A      FUNCTION  TMoveBoxCmd.CREATE(object: TObject; itsHeap: THeap; itsCmdNumber: TCmdNumber;
2 1093 --                                     itsView: TBoxView; itsBox: TBox; itsHOffset, itsVOffset: LONGINT)
2 1094 --                                     : TMoveBoxCmd;
2 1095 0- A      BEGIN
2 1096 --          {$IFC fTrace}BP(10);{$ENDC}
2 1097 --          IF object = NIL THEN
2 1098 --             object := NewObject(itsHeap, THISCLASS);
2 1099 --          SELF := TMoveBoxCmd(TCommand.CREATE(object, itsHeap, itsCmdNumber, itsView, TRUE, revealAll));
2 1100 --          WITH SELF DO
2 1101 1-              BEGIN
2 1102 --              hOffset := itsHOffset;
2 1103 --              vOffset := itsVOffset;
2 1104 --              movedBox := itsBox;
2 1105 -1              END;
2 1106 --          {$IFC fTrace}EP;{$ENDC}
2 1107 -0 A      END;
2 1108 --
2 1109 --
2 1110 --       {$IFC fDebugMethods}
2 1111 -- A      PROCEDURE TMoveBoxCmd.Fields(PROCEDURE Field(nameAndType: S255));
2 1112 0- A      BEGIN
2 1113 --          TCommand.Fields(Field);
2 1114 --          Field('hOffset: LONGINT');
2 1115 --          Field('vOffset: LONGINT');
2 1116 --          Field('movedBox: TBox');
2 1117 -0 A      END;
2 1118 --        {$ENDC}
2 1119 --
2 1120 --
2 1121 -- A      PROCEDURE TMoveBoxCmd.Perform(cmdPhase: TCmdPhase);
2 1122 --        VAR diffLPt:     LPoint;
2 1123 --
2 1124 0- A      BEGIN
2 1125 --          {$IFC fTrace}BP(12);{$ENDC}
2 1126 --                 { Do nothing on the doPhase, since the box has already been moved }
2 1127 --          IF cmdPhase <> doPhase THEN
2 1128 1-              BEGIN
2 1129 --       {$H-}  WITH SELF DO
2 1130 2-                  CASE cmdPhase OF
```

```
2 1131 --                         redoPhase:
2 1132 --                             SetLPt(diffLPt, hOffset, vOffset);
2 1133 --                         undoPhase:
2 1134 --                             SetLPt(diffLPt, -hOffset, -vOffset);
2 1135 -2         {$H+}          END;
2 1136 --                 SELF.movedBox.MoveBox(TBoxView(SELF.image), diffLPt);
2 1137 -1                 END;
2 1138 --
2 1139 --             self.image.view.panel.selection.MarkChanged;    {allow this document to be saved}
2 1140 --             {$IFC fTrace}EP;{$ENDC}
2 1141 -0 A       END;
2 1142 --
2 1143 --     END;
2 1144 --
2 1145 --
2 1146 --
2 1147 --     METHODS OF TBoxProcess;
2 1148 --
2 1149 -- A       FUNCTION  TBoxProcess.CREATE: TBoxProcess;
2 1150 0- A       BEGIN
2 1151 --             {$IFC fTrace}BP(11);{$ENDC}
2 1152 --             SELF := TBoxProcess(TProcess.CREATE(NewObject(mainHeap, THISCLASS), mainHeap));
2 1153 --             {$IFC fTrace}EP;{$ENDC}
2 1154 -0 A       END;
2 1155 --
2 1156 --
2 1157 -- A       FUNCTION  TBoxProcess.NewDocManager(volumePrefix: TFilePath; openAsTool: BOOLEAN): TDocManager;
2 1158 0- A       BEGIN
2 1159 --             {$IFC fTrace}BP(11);{$ENDC}
2 1160 --             NewDocManager := TBoxDocManager.CREATE(NIL, mainHeap, volumePrefix);
2 1161 --             {$IFC fTrace}EP;{$ENDC}
2 1162 -0 A       END;
2 1163 --
2 1164 --     END;
2 1165 --
2 1166 --
2 1167 --
2 1168 --     METHODS OF TBoxDocManager;
2 1169 --
2 1170 -- A       FUNCTION  TBoxDocManager.CREATE(object: TObject; itsHeap: THeap; itsPathPrefix: TFilePath)
2 1171 --                                          : TBoxDocManager;
2 1172 0- A       BEGIN
2 1173 --             {$IFC fTrace}BP(11);{$ENDC}
2 1174 --             IF object = NIL THEN
2 1175 --                 object := NewObject(itsHeap, THISCLASS);
2 1176 --             SELF := TBoxDocManager(TDocManager.CREATE(object, itsHeap, itsPathPrefix));
2 1177 --             {$IFC fTrace}EP;{$ENDC}
2 1178 -0 A       END;
2 1179 --
2 1180 --
2 1181 -- A       FUNCTION  TBoxDocManager.NewWindow(heap: THeap; wmgrID: TWindowID):TWindow;
2 1182 0- A       BEGIN
2 1183 --             {$IFC fTrace}BP(11);{$ENDC}
2 1184 --             NewWindow := TBoxWindow.CREATE(NIL, heap, wmgrID);
2 1185 --             {$IFC fTrace}EP;{$ENDC}
2 1186 -0 A       END;
2 1187 --
2 1188 --     END;
2 1189 --
2 1190 --
2 1191 --
2 1192 --     METHODS OF TBoxWindow;
2 1193 --
2 1194 -- A       FUNCTION  TBoxWindow.CREATE(object: TObject; itsHeap: THeap; itsWmgrID: TWindowID): TBoxWindow;
2 1195 0- A       BEGIN
2 1196 --             {$IFC fTrace}BP(10);{$ENDC}
2 1197 --             IF object = NIL THEN
2 1198 --                 object := NewObject(itsHeap, THISCLASS);
2 1199 --             SELF := TBoxWindow(TWindow.CREATE(object, itsHeap, itsWmgrID, TRUE));
2 1200 --             {$IFC fTrace}EP;{$ENDC}
2 1201 -0 A       END;
2 1202 --
2 1203 --
2 1204 -- A       PROCEDURE TBoxWindow.BlankStationery;
2 1205 --         VAR viewLRect:      LRect;
2 1206 --             panel:          TPanel;
2 1207 --             boxView:        TBoxView;
2 1208 --             aSelection:     TSelection;
2 1209 0- A       BEGIN
2 1210 --             {$IFC fTrace}BP(10);{$ENDC}
2 1211 --              panel := TPanel.CREATE(NIL, SELF.Heap, SELF, 0, 0, [aScroll, aSplit], [aScroll, aSplit]);
2 1212 --
2 1213 --             SetLRect(viewLRect, 0, 0, 5000, 3000);
2 1214 --             boxView := TBoxView.CREATE(NIL, SELF.Heap, panel, viewLRect);
2 1215 --             boxView.InitBoxList(SELF.Heap);
2 1216 --
2 1217 --             {$IFC fTrace}EP;{$ENDC}
2 1218 -0 A       END;
2 1219 --
2 1220 --
2 1221 -- A       FUNCTION  TBoxWindow.NewCommand(cmdNumber: TCmdNumber): TCommand;
2 1222 0- A       BEGIN
2 1223 --             {$IFC fTrace}BP(11);{$ENDC}
2 1224 --
2 1225 1-             CASE cmdNumber OF
2 1226 --                 uClearAll:
2 1227 --                     NewCommand := TClearAllCmd.CREATE(NIL, SELF.heap, cmdNumber,
2 1228 --                                         TBoxView(SELF.selectPanel.view));
2 1229 --
2 1230 --                 OTHERWISE
2 1231 --                     NewCommand := SUPERSELF.NewCommand(cmdNumber);
2 1232 -1                 END;
2 1233 --             {$IFC fTrace}EP;{$ENDC}
2 1234 -0 A       END;
2 1235 --
2 1236 --
2 1237 -- A       FUNCTION  TBoxWindow.CanDoCommand(cmdNumber: TCmdNumber; VAR checkIt: BOOLEAN): BOOLEAN;
2 1238 0- A       BEGIN
2 1239 --             {$IFC fTrace}BP(11);{$ENDC}
2 1240 1-             CASE cmdNumber OF
```

```
2 1241 --                uClearAll:
2 1242 --                        CanDoCommand := TRUE;
2 1243 --                OTHERWISE
2 1244 --                    CanDoCommand := SUPERSELF.CanDoCommand(cmdNumber, checkIt);
2 1245 -1                END;
2 1246 --            {$IFC fTrace}EP;{$ENDC}
2 1247 -0 A      END;
2 1248 --
2 1249 --    END;
1  301 --
1  302 --    END.
```

```
1. u8boxer.TEXT
2. U8Boxer2.text

-B-
 BlankStationery   289•( 1) 1204•( 2)
 Box               139 ( 1)
 box               101 ( 1)  124 ( 1)  187•( 1)  139•( 2)  141=( 2)  142 ( 2)  143 ( 2)  156•( 2)  158=( 2)  159 ( 2)
                   180•( 2)  186=( 2)  188 ( 2)  195 ( 2)  242=( 2)  262 ( 2)  271 ( 2)  283•( 2)  285=( 2)  288 ( 2)
                   291 ( 2)  291 ( 2)  313=( 2)  331 ( 2)  346 ( 2)  364 ( 2)  367 ( 2)  371 ( 2)  384 ( 2)  396 ( 2)
                   435•( 2)  442=( 2)  443=( 2)  443 ( 2)  462•( 2)  466 ( 2)  472=( 2)  472 ( 2)  486 ( 2)  492 ( 2)
                   515 ( 2)  553=( 2)  571 ( 2)  579 ( 2)  588•( 2)  591=( 2)  592 ( 2)  592 ( 2)  594 ( 2)  595 ( 2)
                   596 ( 2)  597 ( 2)  600 ( 2)  664•( 2)  725=( 2)  727=( 2)  745=( 2)  761•( 2)  767 ( 2)  768 ( 2)
                   768 ( 2)  801 ( 2)  802 ( 2)  805 ( 2)  829 ( 2)  849•( 2)  852=( 2)  853 ( 2)  853 ( 2)  920•( 2)
                   938 ( 2)  946=( 2)  946 ( 2)  947 ( 2)  948 ( 2)  959=( 2) 1080 ( 2)
 boxList            73 ( 1)  171 ( 2)  212•( 2)  215=( 2)  216=( 2)  216 ( 2)  643 ( 2)  766 ( 2)  813=( 2)  904 ( 2)
                  1011 ( 2)
 boxSelectionKind   27•( 1)  195 ( 2)  522 ( 2)  654 ( 2)  815 ( 2)  827 ( 2)  958 ( 2) 1078 ( 2)
 BoxWith            79•( 1)  137•( 2)  143=( 2)  186 ( 2)

-C-
 CanDoCommand      116•( 1)  293•( 1)  405•( 2)  413=( 2)  418=( 2)  422=( 2)  422 ( 2) 1237•( 2) 1242=( 2) 1244=( 2)
                  1244 ( 2)
 color              53•( 1)  140•( 1)   12=( 2)   37 ( 2)  552=( 2)  571=( 2)  571 ( 2)  594 ( 2)  595=( 2)  597=( 2)
 colorBlack         24•( 1)   45 ( 1)   42 ( 2)
 colorDkGray        23•( 1)   41 ( 2)
 colorGray          22•( 1)   12 ( 2)   40 ( 2)
 colorLtGray        21•( 1)   39 ( 2)
 colorWhite         20•( 1)   45 ( 1)   38 ( 2)  365 ( 2)
 Commit            160•( 1)  194•( 1)  214•( 1)  231•( 1)  568•( 2)  639•( 2)  759•( 2)  901•( 2) 1008•( 2)
 CREATE             56•( 1)   76•( 1)  104•( 1)  127•( 1)  143•( 1)  158•( 1)  174•( 1)  190•( 1)  209•( 1)  227•( 1)
                   240 ( 1)  256•( 1)  268 ( 1)  277 ( 1)  286 ( 1)    5•( 2)  115•( 2)  121 ( 2)  191 ( 2)  195 ( 2)
                   215 ( 2)  224 ( 2)  234•( 2)  240 ( 2)  271 ( 2)  345 ( 2)  364 ( 2)  367 ( 2)  370 ( 2)  389 ( 2)
                   396 ( 2)  433•( 2)  440 ( 2)  442 ( 2)  522 ( 2)  534 ( 2)  545•( 2)  551 ( 2)  610•( 2)  616 ( 2)
                   692•( 2)  699 ( 2)  738•( 2)  744 ( 2)  808 ( 2)  812 ( 2)  815 ( 2)  864•( 2)  870 ( 2)  987•( 2)
                   993 ( 2) 1052•( 2) 1058 ( 2) 1092•( 2) 1099 ( 2) 1149•( 2) 1152 ( 2) 1160 ( 2) 1170•( 2) 1176 ( 2)
                  1184 ( 2) 1194•( 2) 1199 ( 2) 1211 ( 2) 1214 ( 2) 1227 ( 2)
 createBoxSelecti   28•( 1)  440 ( 2)

-D-
 DoCutCopy         195•( 1)  778•( 2)
 DoPaste           215•( 1)  910•( 2)
 Draw               59•( 1)   87•( 1)   28•( 2)  154•( 2)  159 ( 2)
 DrawFrame          60•( 1)   54•( 2)  466 ( 2)  508 ( 2)

-E-
 EachActualPart     90•( 1)  168•( 2)  681 ( 2)  843 ( 2)  976 ( 2)
 EachVirtualPart   163•( 1)  197•( 1)  217•( 1)  233•( 1)  148 ( 2)  163 ( 2)  305 ( 2)  309 ( 2)  678•( 2)  837•( 2)
                   841 ( 2)  973•( 2) 1040•( 2)

-F-
 Fields             18•( 2)  128•( 2)  130 ( 2)  247•( 2)  249 ( 2)  449•( 2)  451 ( 2)  559•( 2)  561 ( 2)  631•( 2)
                   633 ( 2)  710•( 2)  712 ( 2)  751•( 2)  753 ( 2)  882•( 2)  884 ( 2) 1000•( 2) 1002 ( 2) 1111•( 2)
                  1113 ( 2)
 FilterAndDo       147•( 1)  586•( 2)  848•( 2)
 Free              211•( 1)  531 ( 2)  621•( 2)  624 ( 2)  892•( 2)  895 ( 2)

-H-
 Highlight         108•( 1)  185 ( 2)  197 ( 2)  256•( 2)  312 ( 2)  314 ( 2)  953 ( 2)
 hOffset           251 ( 1)   84•( 2)   86 ( 2) 1102=( 2) 1132 ( 2) 1134 ( 2)

-I-
 InitBoxList        93•( 1)  211•( 2) 1215 ( 2)
 InvalBox           82•( 1)   69 ( 2)   71 ( 2)  202•( 2)  525 ( 2)  579 ( 2)  671 ( 2)  829 ( 2)  966 ( 2) 1080 ( 2)

-K-
 KeyClear          268•( 2)
 KeyTab            276•( 2)
 kind              224 ( 1)  259 ( 2)  299 ( 2)  378 ( 2)  413 ( 2)  530=( 2)  654=( 2)  656=( 2)  958=( 2)  963=( 2)
                   994=( 2)  994 ( 2) 1027=( 2) 1029=( 2) 1029 ( 2)

-L-
 LPoint            186 ( 1)  321 ( 2)  338 ( 2)  460 ( 2) 1122 ( 2)
 LRect              52 ( 1)   81 ( 2)  459 ( 2)  502 ( 2) 1205 ( 2)

-M-
 MouseMove         111•( 1)  131•( 1)  320•( 2)  458•( 2)
 MousePress         84•( 1)  177•( 2)
 MouseRelease      112•( 1)  132•( 1)  337•( 2)  499•( 2)
 MoveBox            63•( 1)   64•( 2)  331 ( 2) 1136 ( 2)
 movedBox          253•( 1) 1104=( 2) 1136 ( 2)

-N-
 newBox            155 ( 1)  617=( 2)  624 ( 2)  643 ( 2)  644=( 2)  671 ( 2)  682 ( 2)  694•( 2)  702=( 2)  703=( 2)
                   703 ( 2)  704 ( 2)  725 ( 2)
 NewCommand        115•( 1)  292•( 1)  352•( 2)  364=( 2)  367=( 2)  370=( 2)  389=( 2)  396=( 2)  399=( 2)  399 ( 2)
                  1221•( 2) 1227=( 2) 1231=( 2) 1231 ( 2)
 NewDocManager     269•( 1) 1157•( 2) 1160=( 2)
 NewWindow         279•( 1) 1181•( 2) 1184=( 2)
 NoSelection        94•( 1)  221•( 2)  224=( 2)

-O-
 oldBox            171 ( 1)  701=( 2)  727 ( 2)

-P-
 PaintHandles       66•( 1)   79•( 2)  262 ( 2)
 pasteBox          206•( 1)  875=( 2)  895 ( 2)  904 ( 2)  905=( 2)  948=( 2)  959 ( 2)  966 ( 2)  977 ( 2)
 pasteH            204 ( 1)  355•( 2)  380=( 2)  386=( 2)  389 ( 2)  873=( 2)  940 ( 2)
 pasteV            205•( 1)  355•( 2)  387=( 2)  389 ( 2)  874=( 2)  942 ( 2)
 Perform           146•( 1)  161•( 1)  232•( 1)  244•( 1)  261•( 1)  576•( 2)  662•( 2) 1016•( 2) 1063•( 2) 1121•( 2)

-Q-
 QuickDraw          15•( 1)

-S-
 selTopLeft        186 ( 1)
 shapeLRect         52 ( 1)   11=( 2)   34 ( 2)   47 ( 2)   48 ( 2)   59 ( 2)   69 ( 2)   70 ( 2)   71 ( 2)   81=( 2)
                    92 ( 2)   95=( 2)   95 ( 2)   96 ( 2)  142 ( 2)  384 ( 2)  488 ( 2)  489 ( 2)  492 ( 2)  517 ( 2)
                   579 ( 2)  671 ( 2)  704 ( 2)  801 ( 2)  802 ( 2)  806 ( 2)  812 ( 2)  829 ( 2)  938 ( 2)  947 ( 2)
                   966 ( 2) 1080 ( 2)
```

```
-T-
  TBox               49•( 1)   56 ( 1)   79 ( 1)  101 ( 1)  124 ( 1)  139 ( 1)  155 ( 1)  171 ( 1)  187 ( 1)  206 ( 1)
                    253 ( 1)    3•( 2)    5 ( 2)  137 ( 2)  139 ( 2)  141 ( 2)  156 ( 2)  158 ( 2)  180 ( 2)  278 ( 2)
                    283 ( 2)  285 ( 2)  435 ( 2)  442 ( 2)  462 ( 2)  500 ( 2)  588 ( 2)  591 ( 2)  664 ( 2)  694 ( 2)
                    702 ( 2)  761 ( 2)  786 ( 2)  805 ( 2)  849 ( 2)  852 ( 2)  919 ( 2)  920 ( 2)  946 ( 2)
  TBoxCutCopyCmd    183•( 1)  191 ( 1)  237 ( 1)  370 ( 2)  736•( 2)  739 ( 2)  744 ( 2) 1058 ( 2)
  TBoxDocManager    274•( 1)  278 ( 1) 1160 ( 2) 1168•( 2) 1171 ( 2) 1176 ( 2)
  TBoxPasteCmd      201•( 1)  210 ( 1)  389 ( 2)  862•( 2)  865 ( 2)  870 ( 2)
  TBoxProcess       265•( 1)  268 ( 1) 1147•( 2) 1149 ( 2) 1152 ( 2)
  TBoxSelection      98•( 1)  105 ( 1)  195 ( 2)  224 ( 2)  232•( 2)  235 ( 2)  240 ( 2)  522 ( 2)  665 ( 2)  669 ( 2)
                    781 ( 2)  815 ( 2)  821 ( 2)  913 ( 2)  914 ( 2)  931 ( 2)  935 ( 2)  952 ( 2) 1065 ( 2) 1071 ( 2)
  TBoxView           70•( 1)   77 ( 1)  113•( 2)  116 ( 2)  121 ( 2)  271 ( 2)  279 ( 2)  298 ( 2)  331 ( 2)  345 ( 2)
                    353 ( 2)  359 ( 2)  461 ( 2)  471 ( 2)  501 ( 2)  513 ( 2)  579 ( 2)  640 ( 2)  643 ( 2)  663 ( 2)
                    668 ( 2)  681 ( 2)  766 ( 2)  780 ( 2)  784 ( 2)  793 ( 2)  812 ( 2)  904 ( 2)  911 ( 2)  915 ( 2)
                    924 ( 2) 1011 ( 2) 1018 ( 2) 1021 ( 2) 1064 ( 2) 1069 ( 2) 1136 ( 2) 1207 ( 2) 1214 ( 2) 1228 ( 2)
  TBoxWindow        283•( 1)  286 ( 1) 1184 ( 2) 1192•( 2) 1194 ( 2) 1199 ( 2)
  TClearAllCmd      222•( 1)  228 ( 1)  985•( 2)  988 ( 2)  993 ( 2) 1227 ( 2)
  TClearCmd         237•( 1)  241 ( 1)  271 ( 2)  367 ( 2) 1050•( 2) 1053 ( 2) 1058 ( 2)
  TColor             45•( 1)   53 ( 1)  140 ( 1)  587 ( 2)
  TCommand          115 ( 1)  138 ( 1)  152 ( 1)  222 ( 1)  248 ( 1)  292 ( 1)  352 ( 2)  551 ( 2)  561 ( 2)  616 ( 2)
                    699 ( 2)  993 ( 2) 1002 ( 2) 1099 ( 2) 1113 ( 2) 1221 ( 2)
  TCreateBoxSelect  121•( 1)  128 ( 1)  191 ( 2)  431•( 2)  434 ( 2)  440 ( 2)
  TCreateCmd        152•( 1)  159 ( 1)  168 ( 1)  534 ( 2)  608•( 2)  611 ( 2)  616 ( 2)
  TCutCopyCommand   183 ( 1)  744 ( 2)
  TDocManager       270 ( 1)  274 ( 1) 1157 ( 2) 1176 ( 2)
  TDuplicateCmd     168•( 1)  175 ( 1)  396 ( 2)  690•( 2)  693 ( 2)  699 ( 2)
  TList              73 ( 1)  212 ( 2)  215 ( 2)  783 ( 2)  808 ( 2)
  TMoveBoxCmd       248•( 1)  258 ( 1)  345 ( 2) 1090•( 2) 1094 ( 2) 1099 ( 2)
  TObject            49 ( 1)  138 ( 2)  155 ( 2)  282 ( 2)
  TPasteCommand     201 ( 1)  870 ( 2)
  TProcess          265 ( 1) 1152 ( 2)
  TRecolorCmd       138•( 1)  144 ( 1)  364 ( 2)  543•( 2)  546 ( 2)  551 ( 2)
  TSelection         94 ( 1)   98 ( 1)  121 ( 1)  178 ( 2)  221 ( 2)  240 ( 2)  440 ( 2)  503 ( 2)  785 ( 2) 1017 ( 2)
                   1208 ( 2)
  TView              70 ( 1)
  TWindow           279 ( 1)  283 ( 1) 1181 ( 2) 1199 ( 2)

-U-
  UBBoxer             4•( 1)
  UABC               17•( 1)
  uBlack             35•( 1)  363 ( 2)  409 ( 2)
  uClearAll          37•( 1) 1226 ( 2) 1241 ( 2)
  uCreateBox         40•( 1)  534 ( 2)
  uDkGray            34•( 1)  363 ( 2)  409 ( 2)
  UDraw              16•( 1)
  uDuplicate         36•( 1)  395 ( 2)  411 ( 2)
  UFont              12•( 1)
  uGray              33•( 1)  363 ( 2)  409 ( 2)
  uLtGray            32•( 1)  363 ( 2)  409 ( 2)
  uMoveBox           41•( 1)  345 ( 2)
  UObject             9•( 1)
  UpdateSelection   162•( 1)  178•( 1)  648•( 2)  670 ( 2)  719•( 2)
  uWhite             31•( 1)  363 ( 2)  365 ( 2)  409 ( 2)

-V-
  vOffset           252•( 1)   84•( 2)   86 ( 2) 1103•( 2) 1132 ( 2) 1134 ( 2)

--- End Xref: 95 id's   725 references      [380384 bytes/4904 id's/37740 refs]
```