Oberon V4

H. Mössenböck

Oberon V4 is a cleared up version of Oberon V2.2 (The version number V3 refers to "Gadgets Oberon" which has a graphical user interface.) It resulted from the desire to integrate the text editors *Edit* and *Write*. This document explains the differences between V4 and V2.2. What is not described here remains as explained in the book "The Oberon System" by M. Reiser (see reference [1] below).

1. Text Editing

General. The following features make the behavior of text frames more natural:

- A text frame may now contain lines of varying heights. Thus, large fonts can be displayed without being clipped to a fixed line raster.
- Characters entered from the keyboard obtain the font attribute of the character at the caret position.
- A text may contain objects such as pictures, tables or buttons. These objects are called *elements* (see below) and are described in detail in the file *Elem.Guide.Text*. A special element, called a paragraph control character (or *Parc*) controls the formatting of the text between itself and the following Parc. Formatting attributes are described in the file *Edit.Guide.Text*.
- When a text selection expands over two viewers all selected lines are shown in reverse video.
- When a text is modified, an exclamation mark is shown at the end of the corresponding viewer's menu. This indicates that the text should be stored before the viewer is closed.

Backward scrolling. Clicking the right mouse key in the scroll bar moves the respective line to the bottom of the frame. Dragging on the right key tracks lines by underlining them. On release the currently underlined line will move to the bottom of the frame.

Keyboard.

- *Auto–scrolling*. Pressing the return key when the caret is in the last line of a frame scrolls the frame up one line.
- *Auto–indentation*. Pressing the line feed key causes a line break with auto–indentation, i.e., as many tabulator or blank characters are inserted at the beginning of the new line as there are tabulator or blank characters at the beginning of the previous line.
- *Cursor keys*. The caret can be moved one character to the left or to the right by pressing the cursor left or cursor right key.
- *Shifting lines*. A couple of selected lines in the focus frame can be shifted to the left or to the right by pressing the cursor left or cursor right key while a selection exists. Shifting to the right means inserting a tabulator character at the beginning of every line (or a blank at the beginning of every line that already starts with a blank). Shifting to the left means deleting a tabulator or blank character at the beginning of every line.

Menu bar. A user can choose the menu commands of a viewer opened with *Edit.Open*. The text to appear in the menu line should be stored to a file with the name *Edit.Menu.Text*. If such a file exists, *Edit.Open* will display its contents as the viewer's menu. Otherwise, *Edit.Open* will use a standard menu.

A menu text may now also contain elements. A new kind of elements, called *MenuElems* (see below) was especially designed to be used in the menu bar.

An exclamation mark at the end of the menu text indicates that the text in the corresponding contents frame was modified. The mark will disappear after executing *Edit.Store*.

2. Module Edit

The following commands have been added or modified:

- *Edit.Search* searches the selection in the target frame. If the command is invoked from the menu of a viewer, the target frame is that viewer's contents frame; otherwise the target frame is the focus frame. Searching starts at the caret position or at position 0 if no caret is set. Successive invocations of *Edit.Search* will place the caret at the next occurrence of the searched pattern. If the pattern is not found, no caret is set. The next call to *Edit.Search* will start searching from the beginning of the text (wrap–around).
- *Edit.Replace* cooperates with *Edit.Search*. It replaces the previously found pattern by the selection and searches for the next occurrence of the pattern. Successive calls to Edit.Replace allow replacing all occurrences of a pattern in a text.
- *Edit.ReplaceAll* works like Edit.Replace but replaces all occurrences of a previously found pattern from the caret position to the end of the text at once.
- *Edit.ClearReplaceBuffer* clears the replace buffer. If no selection exists, the next call to *Edit.Replace* will replace the previously found pattern by the empty string.
- *Edit.ChangeFont (fontName | "↑")*
  Edit.ChangeColor (colorNumber | "↑")
  Edit.ChangeOffset (voffNumber | "↑")
  Changes the font, color or vertical offset attributes of the selection to *fontName*, *colorNumber* or *voffNumber*, respectively. If the command is followed by a "↑", the parameter is taken from the selection in the command frame.
- *Edit.ChangeBackgroundColor (colorNumber | "↑")*
  Changes the background color of the marked frame to *colorNumber*. If the command is followed by a "↑", the parameter is taken from the selection in the command frame.

3. Elements

Elements are objects such as pictures, tables or buttons that can be inserted in a text and flow with it while the text is edited. Clicking at an element with the middle mouse key causes the element to react in some specific way (most elements will allow editing their contents in response to such a click). The following list shows some currently implemented element kinds:

–	*GraphicElems*	pictures drawn with the *Draw* editor
–	*PictureElems*	bitmaps drawn with the *Paint* editor
–	*TableElems*	formatted tables
–	*PopupElems*	buttons that react as pop up menus
–	*MenuElems*	texts that react as pop up menus (for the menu bar of a viewer)
–	*FoldElems*	hypertext folding of text pieces

– *LineElems*	width–adapting lines of various thickness

A detailed description of how to use these elements can be found in the file *Elem.Guide.Text*.

As an example we look at menu elements that may conveniently be used in the menu of a viewer. The command *MenuElems.Insert* inserts an empty menu element represented by an empty rectangle at the caret position. A middle–left click at the element opens a text viewer into which several commands, each on a separate line, can be entered. The first line is interpreted as the menu name. Invoking *MenuElems.Update* from the viewer's menu will store the commands as the menu element's contents. Note that the file *Edit.Menu.Text* may also contain menu elements.

4. Modifications in the Programming Interface

Texts

- The type *Texts.Elem* is the base type for all elements. Although it was already part of Oberon V2.2, it was not described in the book by M.Reiser (see [1] below). Elements are represented by the character *Texts.ElemChar* in the text. They can be read with *Texts.Read(R, ch)*. If *ch* = *Texts.ElemChar*, *R.elem* contains the element, otherwise *R.elem* = NIL. A faster way to read elements is to use *Texts.ReadElem* and *Texts.ReadPrevElem* which skip characters that are not elements. An element can be written to a buffer with *Texts.WriteElem*. There are several messages understood by elements such as the *FileMsg* for loading and storing elements, the *CopyMsg* for obtaining a copy of an element, and the *IdentifyMsg* for obtaining the name of a command that upon invocation will generate an element of this type and install it in the global variable *Texts.new*. An element is contained in at most one text. This text can be obtained with *Texts.ElemBase*.
- In *Texts.Load* and *Texts.Store* the file parameter has been replaced by a rider parameter:
  Texts.Load(text, file, pos, length) ==> Texts.Load(rider, text)
  Texts.Store(text, file, pos, length) ==> Texts.Store(rider, text)
- *Texts.Reader* and *Texts.Writer* are not extensions of *Files.Rider* any more.
- The new procedure *Texts.Close(text, fileName)* writes a text to the file with the designated name and creates a backup (fileName.Bak) if a file with this name already existed.
- The new procedure *Texts.ElemPos(e)* returns the position of the element e in the text containing e.
- Text files start with a tag and a version number. The tag is 0F0X and the current version number is 01X. Note that only text *files* have a tag and a version but not texts.

TextFrames

Text frames have become more powerful and at the same time their interface has become simpler.

- New or renamed fields in *TextFrames.FrameDesc*:

f.hasSel	true if a selection is visible in *f* (replaces *f.sel > 0*)
f.hasCar	true if a caret is visible in *f* (replaces *f.car > 0*)
f.showsParcs	true if Parcs contained in *f.text* should be visible in *f*
f.barW	width of the scroll bar in *f*
f.focus	frame of the element in *f* that has the focus, or NIL

- *TextFrames.Open(frame, handler, text, org, col, left, right, top, bottom, lsp)* has been replaced by *TextFrames.Open(frame, text, org)*. The other parameters are set to the default values exported in the global variables of TextFrames. Other values can be assigned immediately after the call to Open.
- *TextFrames.Parc* is the type of paragraph control characters that control the formatting of texts. Its fields are described in a report by C. Szyperski (see [2] below). The global variable *defParc* holds the

default Parc implicitly assumed at the beginning of every text. The procedure *ParcBefore(text, pos, parc, parcPos)* returns the first Parc in the specified text before the position *pos*.

– The *InsertElemMsg* can be used to insert an element at the caret position in the focus viewer.
– There are several messages that can be sent to elements. Most of them existed already in Oberon V2.2 but they were not described in the book by M.Reiser (see [1] below).

DisplayMsg	causes an element to display itself on the screen
TrackMsg	is sent to an element when the mouse is clicked at it
FocusMsg	is sent to an element when it becomes the new focus or when the focus is removed from it
NotifyMsg	is the base type of messages that can be used to notify an element of a certain event. Sending a *NotifyMsg* to a frame *f* will cause the message to be distributed to all elements that represent subframes of *f*.

– The following procedures have been removed, since they had the same effect as a *MenuVieweres.ModifyMsg*.

Suspend(f)	*MenuViewers.ModifyMsg* (*id* and *dY* arbitrary, *Y = F.Y, H = 0*)
Extend(F, newY)	*MenuViewers.ModifyMsg* (*dY* arbitrary, *id = extend, Y = newY, H = F.Y + F.H – newY*)
Reduce(F, newY)	*MenuViewers.ModifyMsg* (*dY* arbitrary, *id = reduce, Y = newY, H = F.Y + F.H – newY*)
Modify(F, id, dY, Y, H)	*MenuViewers.ModifyMsg* (parameters in record fields)
Restore(F)	*Suspend* followed by an *Extend* to the original height.

– The following procedures have been removed. They should be replaced by the corresponding message.

Copy	*Oberon.CopyMsg* (*msg.F* must be set to NIL before the message is sent)
CopyOver	*Oberon.CopyOverMsg*
Defocus	*Oberon.ControlMsg* (*id = defocus*)
Delete	*TextFrames.UpdateMsg* (*id = delete*)
Edit	*Oberon.InputMsg* (*id = track*)
GetSelection	*Oberon.SelectionMsg*
Insert	*TextFrames.UpdateMsg* (*id = insert*)
Neutralize	*Oberon.ControlMsg* (*id = neutralize*)
Replace	*TextFrames.UpdateMsg* (*id = replace*)
Update	*TextFrames.UpdateMsg*
Write	*Oberon.InputMsg* (*id = consume*)

– The procedure *TextFrames.Call* is not exported any more.

Oberon

– *Oberon.Task* has a new field *time*. A task *t* is only activated if *t.time <= Oberon.Time()*.
– The procedure *Oberon.ShowMenu* has been eliminated.
– The first parameter of *Oberon.Call* is now a value parameter.

References

[1] M.Reiser: The Oberon System. User Guide and Programmer's Manual. Addison–Wesley, 1991
[2] C.A.Szyperski: Write: An Extensible Text Editor for the Oberon System. ETH report 151, 1991