University of Groningen **Faculty of Mathematics and Physical Sciences**

Department of Computing Science

From October 10th, 1995
telephonenumbers will
change to +31.50.363xxxx

The Design and Implementation of an HTML Editor

Victor Smit

begeleider: E. Dijkstra

September 4. 1995

RuG

Contents

Chapter 1

Introduction

The World Wide Web (WWW) [1] is a network of computers providing information and resources on the Internet. The WWW grew out of a hypertext project started at CERN, the European Laboratory for Particle Physices. To access information on the WWW, you use hypertext-based "browser" applications that lead you to the desired documents, then display information on the screen.
The WWW is based on a platform-independent page-desciption language called HTML. At the beginning of '95 there were almost no html editors. The need for such an editor is obvious (see HTML section). Why implement the editor in Oberon System 3? Oberon has a different way of looking at texts, and a 'new' type of graphical user interface. One of the goals of the project is to see how suited Oberon System 3 is for this type of editor. The most important research questions are:

- Is it possible to create a program that you can use both as a browser and as an editor for html documents within the same window?
- Is Oberon System 3 with Gadgets suited for such a program?

Overview
Chapter 2 gives an overview of HTML 2.0. Chapters 3 and 4 describe Oberon and the Gadgets library. Chapter 5 describes the design of the user interface of the editor, chapter 6 the design of the editor itself. Chapter 7 discusses some of the more interesting parts of the implementation. Chapter 8 describes what has not yet been implemented. Chapter 9 gives a brief overview of the new elements of HTML 3.0, and how they can be included in the editor. Chapter 10 evaluates the project, and Chapter 11 give some conclusions.
Appendix A contains a user manual for the Oberon System and the editor, appendix B contains the source code of the editor.

Chapter 2

HTML

2.1 Introduction

HyperText Markup Language (HTML) is a simple markup language used to create internet-wide hypertext documents that are portable from one platform to another. HTML documents are SGML documents [2] with generic semantics that are appropriate for representing information for a wide range of applications. HTML markup can represent hypertext, news, mail, documentation, and multimedia; menus of options, database query results; simple structured documents with in-lined graphics; and hypertext views of existing bodies of information.
HTML has been in use by the World Wide Web (WWW) global information initiative since 1990.
Version 2.0 of the HTML specification [3] introduces forms for user input of information, and adds a distinction between levels of conformance:

- Level 0 indicates the minimum conformance level.
- Level 1 includes Level 0 features plus features such as highlighting and images.
- Level 2 includes all Level 0 and Level 1 features, plus forms.

Currently, HTML version 3.0 is being developed. In Chapter 9 gives an overview of the new elements introduced in this version.

2.2 Overview

HTML describes the structure and logical organisation of a document. It only suggests appropriate presentations of the document when represented on a screen or on paper.
In HTML documents, tags define the start and end of headings, paragraphs, lists, character highlighting and links. Most HTML elements are identified in a document as a start tag, which gives the element name and attributes, followed by the content, followed by the end tag. Start tags are delimited by < and >, and end tags are delimeted by </ and >.

Example:

```
<H1>This is a heading </H1>
```

Creating and browsing HTML documents
HTML-documents are plain ASCII files, so they may be created with ordinary editors. To browse through HTML documents, special viewers like Mosaic or Netscape can be used. These viewers interpret the tags contained in the document. For instance, the text between the tags `<H1>` and `</H1>`, will be shown in a larger font than the other text.
Using an ordinary editor like emacs to create HTML documents however has some disadvantages:

- A browser shows the documently quit differently from what the creator sees while he/she is editing it. So, to make the document look as the creator wants to, he/she will have to switch between the editor and the browser.
- The creator (of the HTML document) has to know the HTML syntax. No online help is available.
- Errors in the syntax and in the references will usually not be found until the document is shown by a HTML browser. This means frequent switching between the editor and browser.

As an illustration, a (part of a) document is shown in figure 2.1 as source text, and in figure 2.2 as Mosaic show it.

2.3 The HTML Tags

This section describes a representative part of all HTML tags. For a more complete list see [3]. First, the tags in the document of figure 2.1 are discussed. Then, some other often used tags are discussed.

The document starts with the <HTML> tag and ends with the </HTML> tag, to show that the document is a HTML text. The text between <!-- and -- > is comment, and will not be shown by browsers.

Head

```
<HEAD> .. </HEAD>
```

The head of an HTML document contains information about the document. The following tags can be used inside the head element.

```
<BASE href="http://www.cs.nl">
<TITLE> Research  </TITLE>
```

```
<html>
<!-- (C) 1994, 1995 PGL, 1995 RS -->
<head> <title>Research</title> </head>
<body>

<h1>Research at Computing Science (RUG)</h1>

<img align=bottom src="http://www.cs.rug.nl/images/cslogo5trans.gif">
<a href="#groups">Research Groups</a>
<br> <img align=bottom src="http://www.cs.rug.nl/images/cslogo5trans.gif">
<a href="#publications">Publications</a>
<br> <img align=bottom src="http://www.cs.rug.nl/images/cslogo5trans.gif">
<a href="#projects">Projects</a>
<br> <img align=bottom src="http://www.cs.rug.nl/images/cslogo5trans.gif">
<a href="#schools">Related Schools and Institutes</a>

<p> <img src="http://www.cs.rug.nl/images/red_line.gif">

<a name="groups">
<h2>Research Groups</h2>

<ul>
<li> <A HREF="http://www.cs.rug.nl/~peterkr/groep.html">Parallel computing,
image processing and applications</A>
<li> <A HREF="research/correctness.shtml">Correctness of programs</A>
<li> <A href="research/specification.shtml">Specification languages and
software engineering</a>
<li> <A href="research/systemprog.shtml">System programming</a>
<li> <A href="research/theory.shtml">Theoretical computer science and
mathematical logic</a>
<li> <a href="research/sdv.shtml">System Design and VLSI</a>
<li> <A href="research/intelligent.shtml">Intelligent Modeling</a>
<li> <A href="research/signal.shtml">Signal and Data Processing</a>
</ul> <hr>

<a name="publications">
<h2>Publications</h2>

Some of the WWW-available publications of the Department are listed here.
```

Figure 2.1: Part of an HTML document

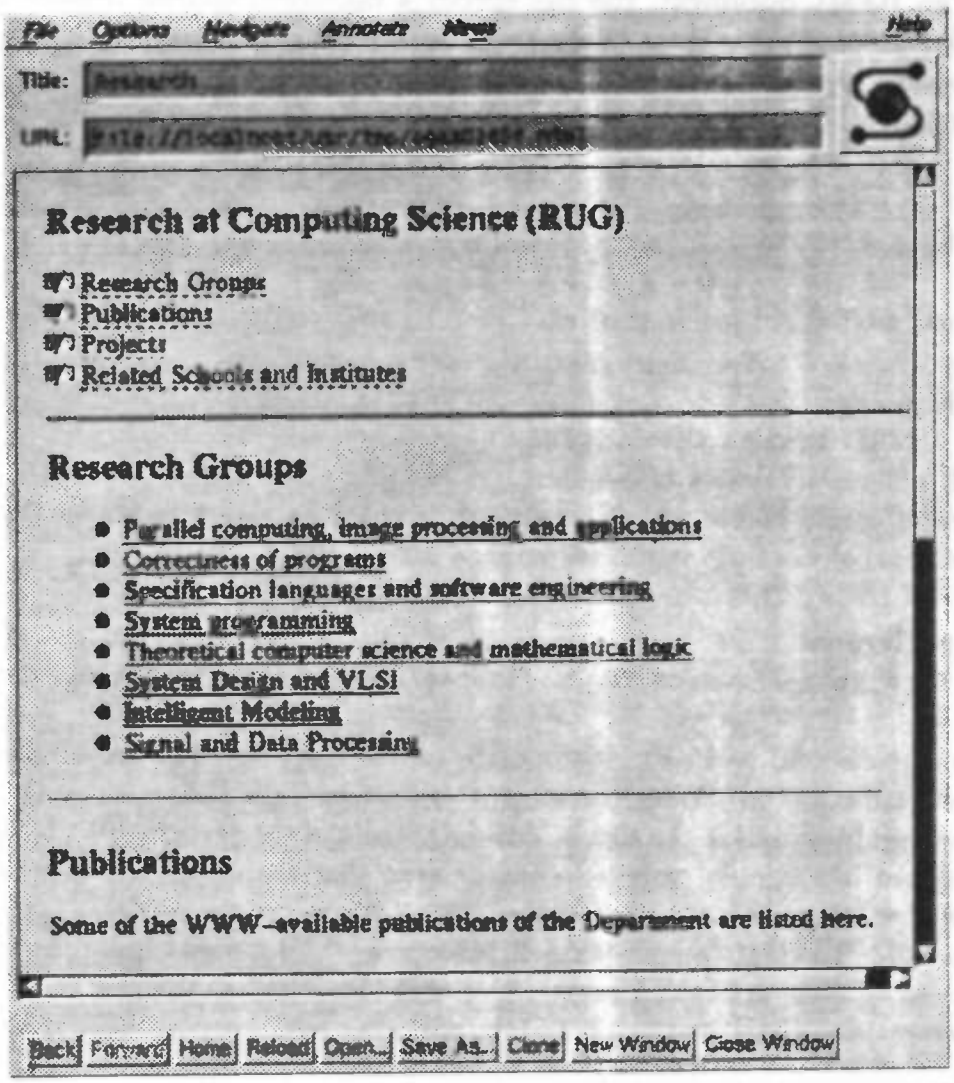

Figure 2.2: The document as shown by Mosaic

The text inside de TITLE tag is the title of the document. Each document must have a title. With the BASE tag you can specify an address.
In HTML terminology, an address is a Universal Resource Locator (URL) [4]. The syntax of a URL is: `<protocol>://<machinename>/<path>/<file>#<label>` A URL contains a protocol (HTTP for HTML documents), a machine name, a filename (including its path), and (optionally) a label wich is defined inside the document.
Inside an HTML document, you can specify a document-address relative to the URL of the current document. These URLs are called *partial* or *relative*.
If an address is set with the BASE tag, relative URLs are specified relative to this address (and not relative to the URL of the current document).

Body

`<BODY> .. </BODY>`

The 'real' text is put inside this tag.

Headings

`<H1> ..</H1>` through `<H6> ..</H6>`.
HTML defines six levels of heading. A heading element implies all the font changes, paragraph breaks before and after, and white space necessary to render the heading. The highest level of headings is H1, followed by H2 ... H6.

Image Element

```
<IMG src="http://www.cs.rug.nl/images/cslogo5trans.gif align=bottom">
```

The image element is used to incorporate in-line graphics. The **src** attribute defines the address of the picture. With the **align** attribute you can define where the text after the picture should begin (at the top, middle or bottom of the picture). With the **alt** attribute (not shown in example) you can define a text that should be shown when the browser cannot show the picture.
It is possible to create a so-called 'clickable image'. Hot spots link different regions in a picture to different actions. A program is needed for handling the user clicks. This program has two main duties:

- Determine the hot spot that contains the clicked position.
- Deliver the WWW page associated with the spot to the user.

More information about clickable maps can be found in [5]

Anchors / Hyperlinks

`<A> .. </A>`

This tag is called an 'anchor'. Anchors are used to created hypertext links and for defining labels. Examples:

```
<A NAME="#groups"> Research Groups</A>
<A HREF="document.html">Document containing specification</A>
<A HREF="document.html#glossary">Go to the glossary</A>
```

The first defines a label to which other anchors can refer. The second example is a link to another document called 'document.html'. The last anchor is a link into document 'document.html', pointing to the label 'glossary'.

Line Break

The texts after a <BR> tag is put on a new line.

Paragraph

The paragraph element <P> indicates the begin of a paragraph. Typically, paragraphs are surrounded by a vertical space of one line or half a line. Some browsers indent the first line in a paragraph.

List Elements

```
<UL> .. </UL>      - Unordered list
<DL> .. </DL>      - Definition list
<DIR> .. </DIR>    - Directory list
<MENU> .. </MENU>  - Menu list
<OL> .. </OL>      - Ordered list
```

There are different type of lists, which will be shown differently. For example, the items inside an ordered list will be numbered, items inside an unordered list will be marked with a large dot. The start of a new item is indicated with <LI>. Definition lists use the tags <DT> and <DD>. The term to be defined is put after the <DT> tag, and definition of the term is put after the <DD> tag.

Horizontal Rule

<HR>

A Horizontal Rule element is a divider between sections of text, often shown as a full width horizontal line.

Information Type Elements

Information type elements specify the logical meaning of marked text without causing a paragraph break. Information type elements can be nested; however, HTML browsers are not required to render nested character-level elements distinctly from non-nested elements.

<CITE> .. </CITE>	- text is citation; typically rendered as italics
<EM> .. </EM>	- texts is to be emphasized; typically rendered in italics
<STRONG> .. </STRONG>	- texts is to be emphasized; typically rendered in bold
<VARIABLE> .. </VARIABLE>	- texts is variable name; typically rendered as italics

Character Format Elements

Character format elements are used to specify the format of marked text. Character format elements can be nested.

<B> .. </B>	text rendered in boldface
<I> .. </I>	text rendered in italic
<TT> .. </TT>	text rendered in fixed-width typewriter font

2.4 Forms

Forms allow a document to include text-entry fields, radio boxes, selection lists, check boxes, and buttons. These can be used to gather information for an application "behind" the document, to guide what is offered to the user next. Some typical forms documents include a movie database, questionnaires, surveys, etc.

There are three basic stages in creating a forms-based document:

1. Design the input form and write the corresponding HTML document.
2. Write the application program that interprets the data from the input form.
3. Design the document generated by the program as the reply to the user. Usually, this document is written in HTML, but this is not mandatory (this can also be a GIF picture, plain ASCII, etc.).

The document in step 1 can be created with an HTML editor. The program in step 2 however, cannot be generated automatically, and must be written by the creator of the document. Information about creating such a program can be found in [6].

Form Elements

<FORM> .. </FORM>

The form element is used to delimit a data input form. Forms cannot be nested.

Example:

```
<FORM METHOD="POST" ACTION="http://www.sun.fi/cgi-bin/mtquery"> .. </FORM>
```

The **method** attribute selects variations in the protocol. The **action** attribute specifies a location and program to which the contents of the form is submitted to get a response. In the example, the location is 'www.sun.fi', and the program is 'mtquery', which is in the '/cgi-bin' directory.

Input Element

<INPUT>

The Input element represents a field whose contents may be edited by the user. Attributes of the Input element:

TYPE	defines the type of data the field excepts. Can be CHECKBOX, IMAGE, RADIO, SUBMIT.
VALUE	Initial displayed value of the field.
NAME	Symbolic name used when transferring the form's contents.

Examples:

```
<INPUT type=submit value=" Submit ">
<INPUT type="text" size=30 name="searchText">
```

The first INPUT is a button, which will submit the data of the form when pressed. The second INPUT is a text input field, with name "searchText".

Select Element

<SELECT NAME = ..> .. </SELECT>

The Select element allows the user to choose one of a set of alternatives described by textual labels. Every alternative is represented by the Option element.
Example:

```
Maximum time to wait for response:
<select name="globtime">
<option selected> 5
<option> 15
<option> 30
<option> 60
<option> Forever
</select>
```

The Selected attribute within the option element indicates that this option is initially selected.

Textarea element

<TEXTAREA> .. </TEXTAREA>

The Textarea element lets users enter more than one line of text. Attributes rows and cols determine the visible dimension of the textfield in characters.

Chapter 3

Oberon System 3 with Gadgets

3.1 Different Oberon versions

Oberon is simultaneously the name of a project and of its outcome. The project was started by Niklaus Wirth and Jürg Gutknecht in 1985 with the goal of developing a modern and portable operating system for personal workstations [7].
Currently there are three different Oberon versions: Oberon System 3 with Gadgets (which is used for the editor), which is an experimental system, Oberon V4, which is the 'real' version, and Oberon/F, which is a commercial variant.
The developers of Oberon at ETH maintain an HTML page devoted to Oberon [8].

3.2 Oberon System 3

Oberon System 3 is both a programming language and an operating environment with a graphical user interface. It is the outcome of a research project whose aim was an extensible, integrated operating platform for single-user personal workstations.
System 3 is an object-oriented evolution of the original Oberon system. Gadgets (TM) is a graphical user interface (GUI) kit that has been specifically designed for System 3. The Gadgets are discussed is Chapter 4.
The Oberon system is structured as a tree of Oberon modules in a shared address space. Modules are loaded and linked dynamically when required. All imported modules are also loaded on command. Modules remain in memory until they are explicitly freed. Each module contains one or more implementations of software components and allocates global storage from a system-wide, shared-memory heap. Memory is recycled automatically when not referenced by a mark-and-sweep garbage collector.
Each module may contain exported, parameterless procedures called "commands". Command procedures can be called by the user directly from the UI. The strings of the form "Module.Procedure" are converted to a jump address. Parameters are passed from the UI to commands by global variables.

3.3 The language Oberon

The Oberon language has a lot in common with languages such as Pascal. The main differences with Pascal are:

- The syntax is slightly different. The following code shows some of the differences:

```
PROCEDURE nop (x, y : INTEGER) : INTEGER;
BEGIN
   IF (x>0) & (y>0) THEN
      x:=6;
      RETURN (x+y);
   ELSIF (y<0) THEN
      RETURN 3;
   ELSE
      RETURN 4;
   END;
END nop;
```

  Notice the following differences:

  - The keyword PROCEDURE is used both for procedures and functions.
  - The procedure name is repeated after the END.
  - The pascal AND is in Oberon &. (OR is still OR in Oberon)
  - The syntax of the IF .. THEN .. ELSE .. construct is different. All control structures have open- and close symbols, to avoid problems with nesting (dangling else).
  - The result of the procedure is passed to the caller with RETURN (Pascal: nop:=3, Oberon: RETURN 3).

- You can make subtypes of records. Example:

```
TYPE
   super = RECORD
     a, b : INTEGER;
   END;
   sub = RECORD (super)
     c : INTEGER;
   END;
```

  A variable of type super has the fields a and b. A variable of type sub has the fields a, b and c.

- Dynamic type information is available. Oberon has the `<variable> IS <type>` construct, which evaluates to TRUE if the variable is of the given type or of a subtype, and FALSE otherwise. For the given example, it can be used to create a function that returns the total of all attributes:

```
PROCEDURE total(s : super) : INTEGER;
BEGIN
   IF s IS sub THEN
       RETURN (s.a + s.b + s(sub).c);
   ELSE
       RETURN (s.a + s.b);
END total;
```

Note that the variable s is type-casted to type sub.

3.4 The standard Modules (libaries) of Oberon System 3

With these mechanisms, Objects are created. Here is the definition of type Object.

```
TYPE
   Handler = PROCEDURE (obj: Object; VAR m : ObjMsg);
   Object = POINTER TO ObjDesc;
   ObjDesc = RECORD
     stamp : LONGINT;
     dlink, slink: Object;
     lib : Library;
     ref : INTEGER;
     handle : Handler;
   END;
   ObjMsg = RECORD
     stamp : LONGINT;
     dlink : Object;
   END;
```

The handle field is used for handling messages. Messages can be 'sent' to an object by calling its handler. The *self* parameter has to be passed to the message handler. Example:

```
VAR om  : ObjMsg;
    obj : Objects.Object;
```

```
BEGIN
   (* initialise Object Message *)
   om.stap:=0;
   om.dlink:=NIL;

   obj.handle(obj, om);
```

The fields stamp (time stamp), and dlink (dynamic link) are used for message flow control. The time stamp is used by objects to recognize a recursive receipt of the same message, and the dynamic link typically records the current message path. The field slink (static link) is used to build temporary sets of objects in the form of linked lists. The fields lib and ref are used when the object is put in a library (see libraries).

The typical structure of a message handler is

```
PROCEDURE someHandler(obj : Object; VAR m : ObjMsg);
BEGIN
   IF m IS msgtp1 THEN
        ...
   ELSIF m IS msgtp2 THEN
        ...
   ELSIF m IS msgtp3 THEN
        ...
        .
        .
        .
   ELSE superclasshandle.(obj, m);
   END;
END someHandler;
```

Handlers of superclasses can be used to handle messages.

Here are some of the types of messages that can be sent to an object:

- **Attribute Message.**
  With each Object, an individual set of attributes is associated. Each attribute is specified by its name and class, where the possible classes are the predefined types LONGINT, REAL, LONGREAL, CHAR, BOOLEAN and ARRAY OF CHAR.

  Attributes can be seen as an Abstract Data Type (ADT), the interface to attributes is given, but attributes may be implemented any way you like.

  All objects have two attributes with the names 'Gen' and 'Name', but subclasses of class objects may define more attributes. 'Gen' contains the procedurename

which was used to create the object (used for object persistence), and 'Name' contains the name of the object. An extra attribute is usually added to a class by adding a field to the record, and modifying the handler of the class (See second example).

With an attribute message you can set the value of an attribute, get the value of an attribute, or do an enumeration over all attributes. The definition of AttrMsg is:

```
TYPE
  AttrMsg = RECORD (ObjMsg)
    id    : INTEGER;        (* get, set or enum *)
    Enum  : PROCEDURE (name: ARRAY OF CHAR);
    name  : Name;           (* ARRAY OF CHAR *)
    res   : INTEGER;        (* result *)
    class: INTEGER;         (* type of attribute *)
    i : LONGINT;
    x : REAL;
    y : LONGREAL;
    c : CHAR;
    b : BOOLEAN;
    s : ARRAY 64 OF CHAR;
  END;
```

Example: set the value of attribute 'X' to 10

```
PROCEDURE SetX(VAR o : Object.Objects);
VAR am : Objects.AttrMsg;
BEGIN
  am.id    := Objects.set;
  am.name  := "X";
  am.res   := -1;
  am.class := Objects.Int;
  am.i     := 10;
  o.handle(o, am);
END SetX;
```

Example: create a new class with an attribute with name 'myint'.

```
TYPE
  newType = POINTER TO newTypeDesc;
  newTypeDesc = RECORD (Objects.Object);
     i : INTEGER;
  END;
```

```
PROCEDURE HandleAttr(o : Objects.Object; VAR am : AttrMsg);
BEGIN
  IF (am.id=Objects.set) & (am.class=Objects.Int)
        & (am.name="myint")
  THEN
    o(newType).i:=am.i;
    am.res:=0;
  ELSIF (am.id=Objects.get) & (am.class=Objects.Int)
        & (am.name="myint")
  THEN
    am.i:=o(newType).i;
    am.res:=0;
  ELSIF (am.id=Objects.enum)
  THEN
    am.enum("myint");
    (* call handler of superclass *)
    Objects.ObjectHandler(o, am);
  ELSE
    (* call handler of superclass*)
    Objects.ObjectHandler(o, am);
END HandleAttr;
```

Notice that there is no need store the value of the attribute in a record/object field with the same name. There is no such thing as an 'Attribute' class, only attribute MESSAGES are defined. How these messages are handled is defined in the message handler. Attributes are an abstract concept.

- **File Message.**
  The object has to save or load itself to a file. This message can be used to make objects persistent. Notice that pointers to other objects cannot be 'saved', since the objects will usually not always be in the same memory location. To be able to store the pointers, the Library concept is introduced (see later).

- **Copy Message.**
  The object has to copy itself. There are two types: shallow and deep. In case of shallow copy, references to other objects are just copied. With a deep copy, the referenced objects are also copied, and references to the new objects are stored in the copy.

- Oberon System 3 has a standard mechanism for grouping objects together: **Libraries**. Libraries can be used to make objects persistent. Libaries can also be used by applications for sharing objects (one application places objects in a library, other applications can use/modify these objects).

  A library is a mapping of Integer → Object.

This is the type definition of libraries:

```
TYPE
  Library = POINTER TO LibDesc;
  LibDesc = RECORD
     ind : Index;
     name : Name;       (* ARRAY OF CHAR *)
     dict : Dictionary;
     maxref : INTEGER; (* Number of objects in the library *)
     GenRef : PROCEDURE (L : Library; VAR ref : INTEGER);
     GetObj : PROCEDURE (L : Library; ref : INTEGER; VAR obj : Object);
     PutObj : PROCEDURE (L : Library; ref : INTEGER; obj : Object);
     FreeObj: PROCEDURE (L : Library; ref : INTEGER);
     Load   : PROCEDURE (L : Library);
     Store  : PROCEDURE (L : Library);
  END;
```

To insert an object in the library, first a new reference has to be generated with the procedure GenRef. Then, the object can be put in the library with PutObj. When an object is in a library, its field `lib` points to the library, and the field `ref` contains its reference number within the library. Each object can only be part of one library.

An object can be retrieved from a library with GetObj given its reference number.

Libraries can be stored to file. Libraries can be used to make objects persistent. If an object contains a pointer to another object, this pointer can be stored using the next two steps:

1. If the object to which is pointed, is not in a library, put it in one.
2. Do not store the pointer, store the pair (library, reference number). (The library is actually a pointer to a record but can be stored by saving its name (a string). Alternaltively, you can 'number' the used libraries, and create a list of (libraryname, libnumber) at the beginning of the file, and use the libnumber instead of the string).

   The pointer can be restored by loading the library and finding the pointer with the procedure GetObj.

- Fonts. A font is just a library. To get the bitmap of the character 'A', just get the object with reference number `ORD('A')` of the corresponding font-library.

- Text. A Text is a list of pairs (library, reference number). The library field is a pointer to a library record, the reference number is a value 0...255. The objects need not necessarily be bitmap-characters, but can be any type of objects. A Text is just a data structure, and cannot visualise itself. To show or edit a Text, a View (e.g. TextGadget) is needed (see gadgets). This view may only be able to show certain type of objects (including, of course (bitmap)characters).

Chapter 4

Gadgets

4.1 Introduction

Gadgets is a GUI management system for Oberon System 3 [9, 10, 11, 12], developed by Hannes Marais [13]. Each dialog element, or gadget, is an object that can be embedded in any UI or application. Gadgets can be part of a text and be embedded in a panel interface, graphic editor, etc.. Figure 4.1 show some often used gadgets.
Container gadgets manage other gadgets as their children. Container gadgets can be used to build larger components out of smaller ones. Important container gadgets are the panel gadgets (two-dimensional edit surfaces), and the text gadgets (complete text editors with support for embedding other gadgets). Most containers specify relatively few rules for parentship, meaning that almost all gadgets can be inserted into all containers.
Gadgets can be modified and used wherever they are located. The construction of a UI is reduced to document editing. Oberon System 3 users can create new UIs or modify existing ones in typical drag-and-drop fashion. For instance, to move a button in a panel, you can just pick it up (press the middle mouse button near the border of the button), and drop it in the position you want. A UI can be frozen with the Inspector (a small utility), but users can unlock a UI at any time to make adjustments.
Gadgets can have attributes. Attributes are (name, value) pairs, which can be modified by users (and by programs). Each gadget has an attribute **name**, which is used by programs to 'find' the gadget. Each value that might be interesting for users to modify, gadgets contain an attribute. With the inspector, the attributes of a gadget can be modified.
An example of an attribute is the **Cmd** attribute of buttons. This attribute contains the 'action' that should be performed when the button is clicked. The action is coded as a procedure in the Oberon language.

4.2 Programming with gadgets: Example Counter

To clarify the programming with gadgets, an example is included here. In the example, a simple application is built. For the user interface, a panel is built with 3 buttons and a

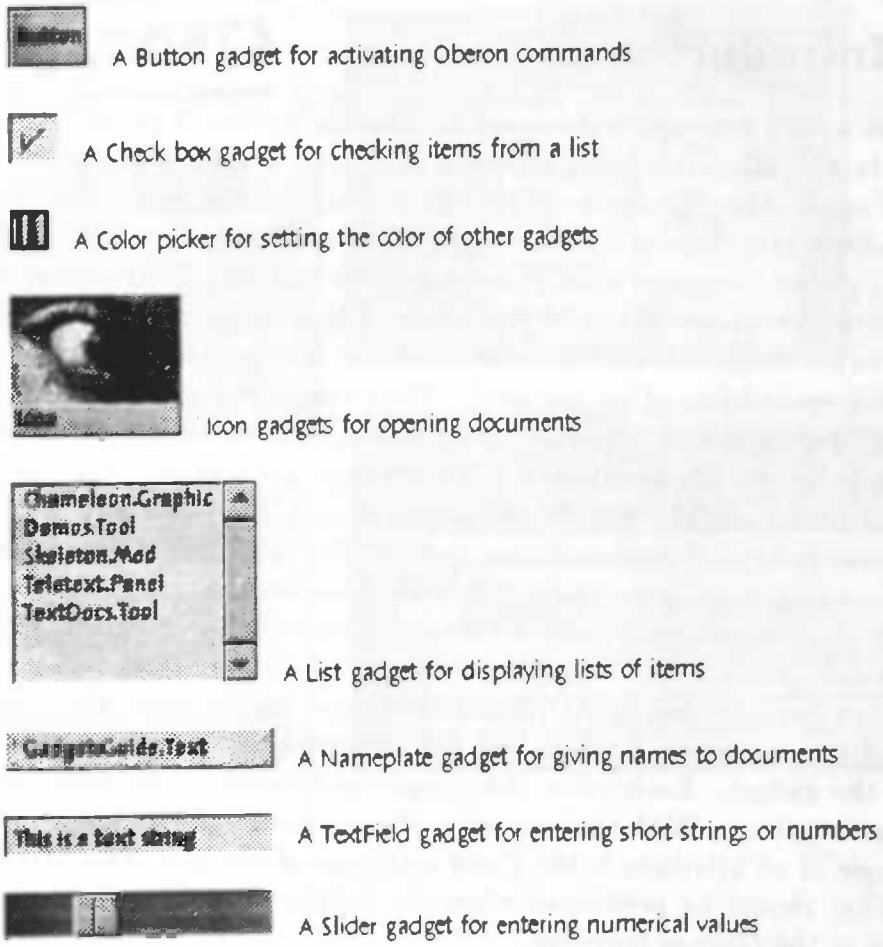

Figure 4.1: Some standard Gadgets

textfield. The textfield shows the value of an integer. With the buttons (labeled Inc, Dec and Reset), the value of this integer can be increased, decreased and set to zero.
First, a desktop is opened by executing the command `Desktops.Open Oberon.Desk~`. The gadgets tool is opened with the command `Desktops.OpenDoc Gadgets.Panel`. An inspector is opened by clicking on the 'inspector' button in the Desktops.Tool.
Figure 4.2 shows how the desktop looks after these commands.
The interface for the counter is created using the following steps:

1. A new Panel is created by clicking on the 'Panel' button in the Desktops.Tool. A window with two buttons (labeled 'close' and 'store') will appear.

2. Create 3 buttons: select a point inside the new panel with the left mouse button. Now click on 'Button' in the View list of the Gadgets.Panel. Repeat this for the other two buttons.

3. Create a textfield (similar to creating buttons)

4. Create an Integer gadget. This is a 'model' gadget (in the Model-View-Controller sense [14, 15]), which will hold the value of the counter. This gadget is not visible. The gadget will be attached to the textfield. 'Attached' means that when the value of the Integer gadget changes, the textfield will be informed with a message (this message is of type LinkMsg, and is broadcasted through the display space). In this way, the textfield will always show the most current value of the Integer gadget.

   The Integer gadget is created by first selecting the textfield (right mouse button), and then clicking on 'integer' inside the Model list of the Gadgets.Tool. When a model gadget is created, it will be attached to all selected gadgets.

5. Some attributes of the newly created buttons are changed, by first selecting a button, and then clicking on the 'inspect' button of the Inspector. The inspector will now show all attributes of the button. The attribute **Caption** contains the string that is shown on the button. Change this value of the buttons into 'Reset', '+' and '−' respectively. The **Cmd** attribute contains an Oberon command which will be executed when the button is clicked. Change this value of the buttons into 'Counter.Reset', 'Counter.Increase' and 'Counter.Decrease'. These oberon command have to be implemented in the module 'Counter'.

6. The name of the Integer gadget is set. This name is used to 'find' the gadget in the Module Counter. Select the textfield, and click on the 'Inspect Model' button of the inspector. Change the attribute **Name** into 'theCount'.

7. A name is given to the panel and it is saved. The name of the panel is shown on the upper left corner of the panel, and can be edited. Then save the panel by clicking on the 'Store' button.

Now, the interface is ready and the module Counter has to be implemented. Here is a possible implementation:

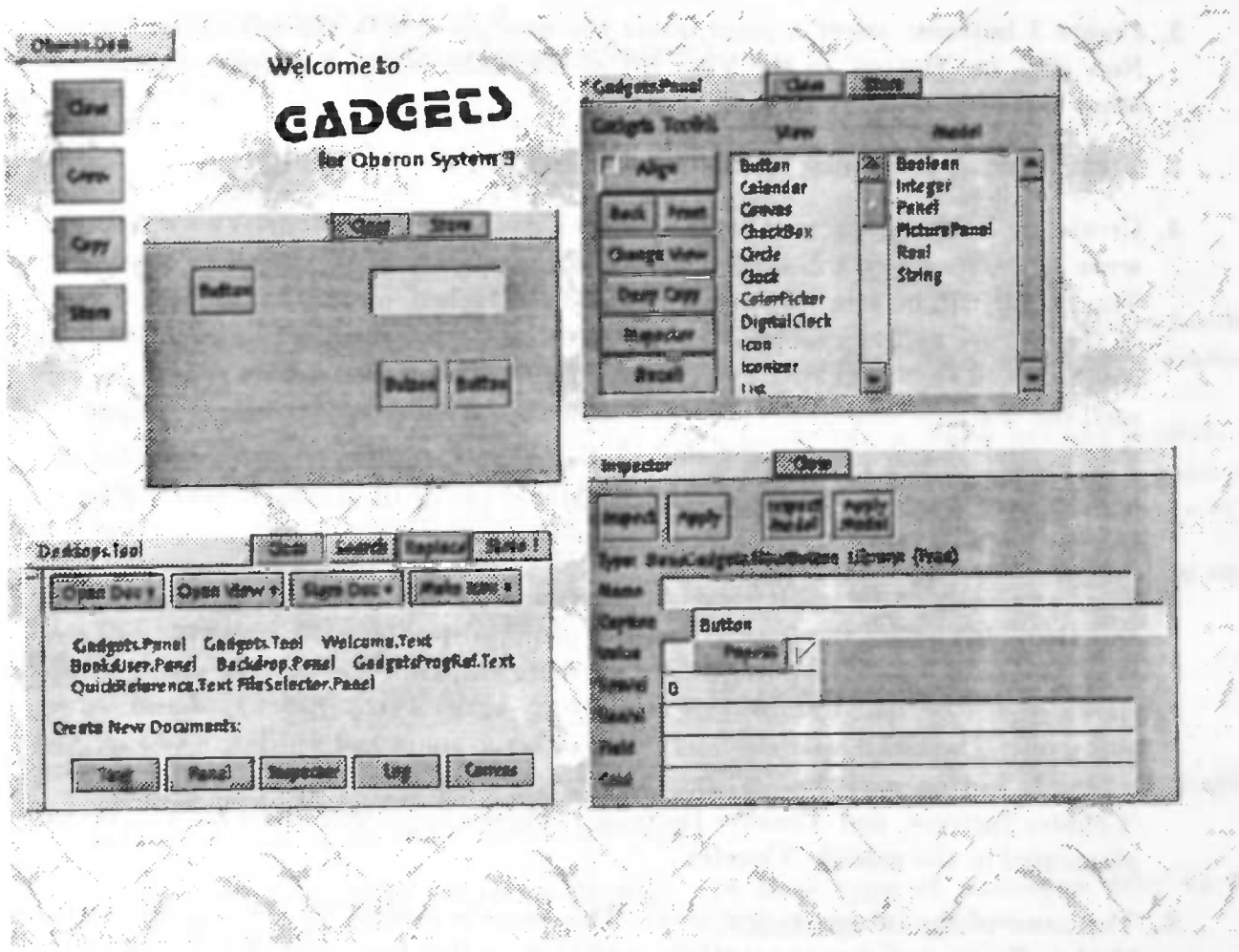

Figure 4.2: The desktop when the interface for the Counter example is created

```
MODULE Counter;
IMPORT Objects, Gadgets, BasicGadgets;


PROCEDURE GetInteger(name: ARRAY OF CHAR): BasicGadgets.Integer;
VAR obj: Objects.Object;
BEGIN
    (* when the button is pressed, the variable Gadgets.context
       is set to the parent (container) of the button.
       The function Gadgets.FindObj searches all children
       of an object for a object with a certain name *)
    obj := Gadgets.FindObj(Gadgets.context, name);
    IF (obj # NIL) & (obj IS BasicGadgets.Integer) THEN
RETURN obj(BasicGadgets.Integer)
    ELSE
   RETURN NIL
    END
END GetInteger;

PROCEDURE Increase*;
VAR counterGadget : BasicGadgets.Integer;
BEGIN
    (* find the model gadget containing the value of the counter *)
    counterGadget := GetInteger("theCount");
    IF (counterGadget=NIL) THEN
       RETURN
    END;

    (* add 1 to the counter *)
    counterGadget.val  := counterGadget.val  + 1;

    (* let all clients of the counterGadget know there has been a change *)
    BasicGadgets.SetValue(counterGadget);
END Increase;

PROCEDURE Decrease*;
VAR counterGadget : BasicGadgets.Integer;
BEGIN
    counterGadget := GetInteger("theCount");
    IF (counterGadget=NIL) THEN
       RETURN
    END;
    counterGadget.val  := counterGadget.val  - 1;
```

```
    BasicGadgets.SetValue(counterGadget);
END Decrease;

PROCEDURE Reset*;
VAR counterGadget : BasicGadgets.Integer;
BEGIN
    counterGadget := GetInteger("theCount");
    IF (counterGadget=NIL) THEN
       RETURN
    END;
    counterGadget.val  := 0;
    BasicGadgets.SetValue(counterGadget);
END Reset;

END Counter.
```

4.3 Creating New Gadgets

To create a new gadget, one has to know which messages a gadget should be able to handle. The gadget class is a subtype of Display.Frame. A Display.Frame is, simply said, a window. Here is part of the Module Display definition:
A frame has a position (X,Y), a width (W) and a height (H). A frame has two pointers to other frames: one to a frame of the same depth (next), and one to a frame of a deeper level (dsc). These pointers are kept for the message broadcasting mechanism (see later).

```
Frame = POINTER TO FrameDesc;          (* base type of all displayable objects *)
FrameDesc = RECORD (Objects.ObjDesc)
   next, dsc: Frame;                   (* brother, child *)
   X, Y, W, H: INTEGER                 (* coordinates *)
END;
```

A frame has to be able to handle more types of messages than normal Objects. FrameMsg is the basetype for all frame specific messages.

```
FrameMsg = RECORD (Objects.ObjMsg)  (* base type of messages sent to Frames *)
   F: Frame;                        (* message target *)
   x, y,                            (* message origin *)
   res: INTEGER                     (* result code: <0 = error or no response,
                                       >=0 response and/or invalid *)
END;
```

Messages can often not be sent directly to the target frame itself. For instance, when a frame should become smaller, the frame that contains this frame also changes. Therefore, a message broadcasting mechanism has been introduced.

When a message is broadcasted, it is sent to the root of the display space (the graph formed by all currently visible frames). Each frame that gets a message determines if it should respond to it and to which other frames (the 'next' and the 'dsc' attributes) it should send the message.

The broadcast message should be used when there might be other objects interested in the message besides the sender and the receiver. An example of such a message is the modify message (see later).

The broadcast procedure is defined in module Display:

```
   MsgProc = PROCEDURE (VAR M: FrameMsg);

VAR
   Broadcast: MsgProc;    (* Broadcast to all frames in the display space *)
```

Note: The broadcast procedure is a variable, and can be set by program.
Some of the messages that a frame should be able to handle are:

- **Modify Message.** The Modify Message is used to change the size of a frame. This message has to be sent with the broadcast mechanism: when a frame is resized, some other frames may have to redisplay themselves.

```
ModifyMsg = RECORD (FrameMsg)      (* changes coordinates of dest. frame *)
   id,                             (* reduce, extend, move *)
   mode: INTEGER;                  (* display, state *)
   dX, dY, dW, dH: INTEGER;        (* change from original coordinates *)
   X, Y, W, H: INTEGER             (* new coordinates *)
END;
```

- **Display Message.** When a frame receives a Display Message, it has to redisplay itself.

```
DisplayMsg = RECORD (FrameMsg)     (* display request to destination frame *)
   id: INTEGER;                    (* frame, area *)
   u, v, w, h: INTEGER             (* coordinates of area to be restored *)
END;
```

- **Select Message.** The Select Message can be used to find all selected objects, or to (de-)select objects.

```
SelectMsg = RECORD (FrameMsg)      (* selection control *)
   id: INTEGER;                    (* get, set, reset *)
   time: LONGINT;                  (* time of selection *)
   sel: Frame;                     (* parent of selection *)
   obj: Objects.Object             (* result list of the selection *)
END;
```

- **Consume Message.** When a user 'drops' a frame (gadget) on a another frame, the last frame receives a Consume Message.

```
ConsumeMsg = RECORD (FrameMsg)    (* consume control *)
   id: INTEGER;                   (* drop, integrate *)
   u, v: INTEGER;    (* relative coordinates in destination when drop *)
   obj: Objects.Object            (* list of objects to be consumed *)
END;
```

The Gadget's base type is a type extension of the Frame type and defines further local variables for its visible region on the display, current state, and so on.

Creating new gadget types
To create a new gadget type, two approaches are possible:

1. Start from scratch. This means creating a subclass of Gadgets.Frame.
2. Using an existing gadget, and change it by giving it a new message handler. You can use the old message handler for every type of message for which no new behaviour is needed.

The second approach requires less coding and should therefore (if possible), preferably be used.

4.4 Programming a new gadget: Example LinkGadget

The example is a simplified version of the gadget used in the editor to represent an anchor (hyperlink). The new gadget differs from its superclass TextGadget in the following ways:

- The text inside the textgadget will be blue (when a character is typed, the color will be set to blue).
- The new gadget has an attribute 'href'.
- The gadget will save itself in a special way when a FileMsg is received.
- When the middle mouse button is clicked, the function Loader.Load will be called .

```
MODULE LinkGadgets;

IMPORT
   Objects, Oberon, Texts, TextGadgets, TextGadgets0, Out, Gadgets,
   TextFrames, Loader, Panels, TextFields, BasicGadgets, Files, Store,
   Strings, Display, Display3;
```

```
CONST
    LinkGadgetTextColor*=Display3.blue;

TYPE
    LinkGadget* = POINTER TO LinkGadgetDesc;
    LinkGadgetDesc* = RECORD (TextGadgets.FrameDesc)
        href : ARRAY 200 OF CHAR;
    END;

PROCEDURE LinkAttr(L : LinkGadget; VAR M : Objects.AttrMsg);
(*  AttributeMessage for LinkGadgets *)
BEGIN
    IF ((M.id = Objects.get) & (M.name = "Gen"))
    THEN
        M.class := Objects.String; M.res := 0
        COPY("LinkGadgets.NewLinkGadget", M.s);
    ELSIF ((M.id = Objects.get) & (M.name = "Gen"))
    THEN
        M.class := Objects.String;
        COPY(L.href, M.s); M.res := 0
    ELSIF ((M.id = Objects.set) & (M.name = "Gen") & (M.class = Objects.String))
    THEN
        COPY(M.s, L.href); M.res := 0
    ELSIF M.id = Objects.enum
    THEN
        M.enum("href");
    END;
END LinkAttr;

PROCEDURE StoreLinkGadget*(VAR o :Files.Rider; f : LinkGadget);
VAR a : Tags.Attr;
BEGIN
    Strings.WriteStrToFile(o, "<A ");
    IF f.href#0X THEN
        Strings.WriteStrToFile(o,"HREF =");
        Strings.WriteStrToFile(o,a.s);
        Files.Write(o,' ');
    END;
    Strings.WriteStrToFile(o,"> ");
    Store.Store(o, f.text);
    Strings.WriteStrToFile(o,"</A>");
END StoreLinkGadget;

PROCEDURE Handle*  (F: Objects.Object; VAR M: Objects.ObjMsg);
BEGIN
    IF ((M IS Oberon.InputMsg) &
            (Gadgets.InActiveArea(F(Gadgets.Frame),M(Oberon.InputMsg))))
    THEN (* text inside a linkgadget has color blue *)
        M(Oberon.InputMsg).col:=LinkGadgetTextColor;
    END;

    IF M IS Objects.AttrMsg
```

```
    THEN LinkAttr(F(LinkGadget), M(Objects.AttrMsg))
   ELSIF ((M IS Objects.FileMsg) & (M(Objects.FileMsg).id=Objects.store))
    THEN StoreLinkGadget(M(Objects.FileMsg).R, F(LinkGadget));
   ELSIF M IS Oberon.InputMsg THEN
       IF 1 IN M (Oberon.InputMsg).keys
       THEN
           Loader.Load (F(LinkGadget).href);
       ELSE
           TextGadgets.FrameHandler (F,M);
       END;
   ELSE TextGadgets.FrameHandler (F,M);
   END;
END Handle;

PROCEDURE InitLinkGadget* (f : LinkGadget);
VAR
   cmesg : Objects.CopyMsg;
BEGIN
  (* There does not exist a Initialise function for TextNotes.
     To initialise a LinkGadget, which is a subclass of TextNote, first
     a new TextNote is created.
     All the variables contained in the TextNote are copied
     to the LinkGadget. *)

   TextGadgets.NewNote;

   cmesg.id:=Objects.deep; cmesg.obj:=f;
   TextGadgets.CopyFrame(cmesg, Objects.NewObj(TextGadgets.Frame),f );

   f.handle:= Handle;
   f.href:="";
   f.state0:={1,2,3,5,6,7,8,9,10,11,12,15};
END InitLinkGadget;

PROCEDURE NewLinkGadget*;
VAR f : LinkGadget;
BEGIN
   NEW(f);
   InitLinkGadget(f);
   Objects.NewObj := f;
END NewLinkGadget;

END LinkGadgets.
```

Notice that the absence of an Initialiser function for TextGadgets complicates the Initialising of the new gadget. If TextGadgets would have an initialise function, the function would become:

```
PROCEDURE InitLinkGadget* (f : LinkGadget);
BEGIN
    NEW(f);
```

```
    TextGadgets.Init(f);

    f.handle:=Handle;
    f.hef:="";
    f.state0:={1,2,3,5,6,7,8,9,10,11,12,15};
END InitLinkGadget;
```

Chapter 5

Design of the user interface

The user interface was designed using two important guidelines:

1. Use the interface style of existing Oberon programs.
2. Make the HTML documents look like similar to the way browsers show them.

In existing Oberon programs, users can:

- use the keyboard for character input
- use the mouse
- use the buttons in the menu border
- execute Oberon commands (which can be done using a tool)

The editing is done inside a TextGadget, so the major edit functions (keyboard input, setting the caret, copying, deleting, etc.) are exactly like the standard editor. Some buttons are put in the menu border for executing often used functions. These are **'close'** (standard button, used for closing the document), **'store'** for storing the current document, **'back'** for moving to the previous document and **'format'** for reformatting the text (see section 5.1). For other functions, Oberon commands are used.

The textgadget is placed in a panel, which also contains the title of the document, the URL of the document, and some buttons.

To make the HTML documents look similar to the way browsers show them, the tags are represented by gadgets. For each tag, a gadget has to be made. Usually, tags have a begin tag, and an end tag. To represent them, you have two choices:

1. Replace each tag with a gadget.
2. Replace all tags and the text between the tags by one gadget (which can of course, contain other gadgets).

For each HTML element with a begin and end tag (or more general: tags which are connected to another is some way), this decision has to be made. The second approach has been chosen for:

1. Hyperlinks. The text between the begin and end tag is placed in a subtype of textgadget. The text inside the textgadget is blue.

2. Forms. The text inside the form is placed in a subtype of textgadget. The background of a form is white, so the gadget can be easily identified as a form.

3. Within definition lists (<DL>..</DL>), the tags <DT> and <DD> and the text belonging to these tags, are placed in a panel. The panel has a TextGadget for the text belonging to the <DT> and the <DD>.

It is possible to make a gadget for a complete list, and place the items in separate textgadgets. This was not done because a lot of html texts would then have an awful lot of different windows (textgadgets), which would make the text look chaotic.
Therefore a different approach is used: the begin and end of a list are marked by gadgets, and each start of an item is marked by a gadget. This representation makes it possible to create 'incorrect' html texts, e.g. the end gadget of a list could be deleted. To make sure that the text remains correct html, some basic editing functions (such as deleting, copying) have to be modified. For instance, when a user wants to delete the end gadget of a list (and not the begin-gadget), the end gadget has to be placed at the beginning of the text that should be deleted (the rest can be deleted).

5.1 Formatting

By replacing the tags with gadgets, the text will look quite similar to the way browser show them.
But there still are some differences: for example, although headers are shown in a large font, there does not necessarily need to be a white space around them.
Similarly, it is possible to type text right before an item (the bullet) in a list. Placing the complete list in a new gadget does not completely solve the problem: you can still type text before this new gadget.
To make sure that an item always starts on a new line, there are two solutions:

1. Modify or create a Text-Viewer/Editor that will show items on a newline.

2. Make sure that in a text, all item gadgets are preceded by carriage returns.

The first approach is the most appealing, but modifying the TextGadget seems to be hard (there is no documentation about doing this, and it is not clear what the re-definable functions should do), and creating a new gadget from scratch is a lot of work.
Therefore, a variant of the second approach is actually used: a re-format function is created, that will place carriage returns in the 'right' place. This function is called directly after loading a document, and can later be called by the user.

Chapter 6

Design of the Editor

6.1 Global Structure

The internal representation of the HTML document is an Oberon 'Text'. A text is just a list of objects. All the tags in the original document are represented by objects.
To view and edit a text, the standard gadgets TextGadget and TextNote can be used. The only difference between TextGadgets and TextNote is that TextGadgets have a scroll bar and Textnotes don't. Unfortunately, TextGadgets only have a vertical scrollbar, and no horizontal scrollbar. This means that if a line does not fit inside the window, you cannot see (or edit) the invisible part (you can of course, split the line, and then see the rest).
The editor uses the TextGadget to view and edit the text. During the project, some disadvantages of Textgadgets where found (see design of the interface, evaluation).
The structure of editor can be summarised:

- the html document is internally stored as a Text.
- the tags in the document are replaced by objects (gadgets).
- a TextGadget shows the text and handles the editing.

While the user modifies the text, the editor has to make sure that the text remains a correct html text.
The tasks of the editor can be divided into:

- **Network.** The editor can be used as a browser, so when a link should be followed, a document should be loaded.
- **File Handling.** The editor must know which files are on the local disk, which files have already been loaded, etc.
- **Parsing.** When the document is in a (local) file, the document must be put in the datastructure that is used by the editor.

- **Viewing and Editing.** After the loading, the document is shown in a window and can be edited. This will be the largest part of the total program.
- **Storing.** After editing, the document must be stored, in the HTML format (this is the inverse of the parsing part).

6.2 Network Part

6.2.1 Introduction

One of the tasks of the editor is to get HTML documents through the internet. Documents are send through the internet with the protocol HTTP [16]. HTTP works quite similar to the email protocol.
It was not necessary to write the code for the HTTP protocol, since there exist public libraries that are able to handle HTTP. The 'WWW Library of Common Code' [17] is such a library. This library does not only contain code for accessing HTTP, but also FTP, Gopher, News, WAIS, telnet severs, etc. The library is written in C and runs on almost any UNIX machine.

6.2.2 Interface

For the editor, only one function is wanted: Load a document given its URL. The library does not contain this routine explicitly, so a new library based upon the WWW Library of Common Code was created. The interface of the new library consists of only one routine:

```
void load_html_page(char* adres);
```

This routine should load an html document. The address (URL) of the document is given as a parameter, and the document is placed in a file with the name `html_file` [1].
The new library contains

- The declaration of some variables needed by the library
- Initialisation of the library
- Within the function `load_html_page`: create a Request structure, and call HTLoadToStream routine.

The library is written in C, but this is no problem since Oberon System 3 can use shared object libraries (See [18], file ExternalRoutines.Mod for details).

[1] It is better to load the document to a temporary file, since it is possible that a user already has a file with the same name.

6.3 File Handling

The task of the file handling part of the editor is: retrieve a file given its URL or filename, and determine what to do with it (not all files are HTML documents).
Features that can be included are:

- history (used files are stored locally, to reduce network traffic and user response time)
- handling of .ps, .MPEG, .jpeg, etc. files. When such files are loaded, an external program must be used to display the information. A list of (extension, program) can be used to store the program names that must be used for files with certain extensions.

Only a limited version of the 'history' is implemented. The URLs of the visited documents are kept in a list. Users can go back to the previous document by clicking on a button. Then, the corresponding document is reloaded. This means that users will lose changes if they do not save the document before following a hyperlink.

6.4 Parsing

The parser is responsible for the translation of the html source text to the internal representation. This means, that the tags in the original text are replaced by objects (gadgets). Since not all the used documents will be correct in the sense of the HTML specification 2.0, the parser should be flexible: erroneous documents should be interpreted correctly (if possible).
Examples of common errors found in existing html documents:

- The document contains no header section.
- When a label is defined, the end tag (`</a>`) is missing.
- A tag has attributes which are not in the specification
- Using `&` instead of `&` when an & is meant.
- etc.

Some of the used 'error correction' tactics are:

- Unknown tags are skipped. This means that only supported tags can be edited. The editor can handle documents conformable to the HTML specification 2.0, new versions of the specification are not supported.
- A lot of tags have a begin and an end tag. (e.g. `<A>` and `</A>`). When one of these is missing, either the existing tag is deleted, or the missing tag is inserted.

6.5 Viewing and Editing

To show the Text, a TextGadget is used. Besides showing the Text, the TextGadget also processes the user input. Some of the functions are:

- Handling keyboard input
- Setting the caret
- Selecting text
- Deleting selected text
- Copying selected text to caret
- Resizing the window

6.6 Storing a document

The document must be stored in the html format (ASCII), so it cannot be stored as a normal oberon text.
The http language has a 'send' command, but what the receiver does with the message is not strictly specified (it can be directly saved, buffered, or just ignored).
So this does not seem very useful to send an edited document with http. Therefore this is not implemented, and documents can only be saved to a local file.
The implementation of the storing algorithm is quite simple: characters are just saved to a file. The gadgets will store themselves when they receive a 'FileMsg'.

Chapter 7

Implementation

The editor is implemented on the Sparc version of Oberon System 3 (version 1.5). A newer version of Oberon System 3 (version 2.0) is available for Microsoft windows, which has some features which could be used in the editor (see later). The Sparc version will be available soon.

The editor uses a lot of modules. For each group of related tags, a separate module is used in which the code for the new Gadgets is put. Separate modules are created for parsing, loading documents, storing documents and reformating documents. Some utility modules are created for code not belonging to any of the other modules.

Import structure of the modules
In the implementation, the code for parsing is put in the module Parser, the code for LinkGadgets in the module LinkGadgets.
When a html document is parsed, LinkGadgets are created. So the Parser module imports the LinkGadget module. When a LinkGadget is clicked, a new text is loaded, and then parsed. So, the Linkgadgets module imports the Parser module.
This means that the import structure contains a cycle. This gives compilation problems (it is possible in Oberon System 3 to create and compile such a program, but not without problems), and therefore is changed.
This can be solved in two ways:

1. Integrate some or all of the code. The code that contains the 'cycle' should be moved to a single module. The two modules can be merged to one module, or a third module can be created to which the code with the 'cycle' is moved.

2. In Oberon System 3, it is possible to create an object given the name (as a string) of the creator procedure. This makes it possible to create a LinkGaget, without importing the LinkGadget module. In this way, the import structure is free of cycles.

Example

In the following code, a LinkGadget is created (without importing the Module LinkGadgets), and its atttribute 'href' is set.

```
IMPORT Objects, Gadgets;

VAR
  o  : Objects.Object;
  am : Objects.AttrMsg;
BEGIN
   (* Create a LinkGadget *)
   o:=Gadgets.CreateObject("LinkGadgets.NewLinkGadget");

   (* set the href attribute to "test" *)
   am.id:=Objects.set;
   am.name:="href";
   am.s:="test";
   o.handle(o,am);
END
```

This is only possible because the symbol information is available at runtime. This means that at runtime, the address of a procedure can be found when the name of the procedure is given. This is the solution used in the implementation.

Main Panel and Document

The TextGadget in which the editing is done, is placed in a panel. This panel also contains TextFields for the title and the URL of the current HTML file. This panel is created similar to the way the Counter (4.2) was made.
To give the panel a menu bar, a 'Document' has to be created. A Document gadget acts as a wrapper for other types of gadgets, giving them a corresponding filename, menu bar text and printing capability. For the editor, only the menu bar is redifened. This is done in module HTEdit. When the main panel is resized, the TextGadget containing the HTML text, is also resized. This is done by giving the panel a new handler.

Links

An HTML Link (anchor) is represented by a gadget. The parser replaces the HTML text ... by a link gadget.
This gadget cannot be a standard gadget, but must be a newly designed gadget since it should have a slightly different behaviour.
The link gadget can be implemented by creating a subclass of a existing gadget. There are a number of standard gadgets that can be used for this: TextField, TextNote or NamePlate.

TextField and NamePlate can only contain one line. This will usually be enough but maybe not always. More importantly: these gadgets can contain only up to 64 characters. This is clearly not enough.
The TextNote seems to be the best. Here are some of the characteristics of TextNote.

- TextNote can contain multiple lines of text (a TextField cannot)
- TextNote can contain text of arbitrary length.
- The TextNote automatically resizes so that all text will be visible (when the state is set accordingly).

To create a new LinkGadget, the following steps have to be made:

1. Define a subclass of TextNote. This can best be done in a separate Module.
2. Add extra field to this new class to hold information as the 'href', etc.
3. Create a new handler for this class. The handler needs to take different action when the middle mouse button is pressed.

When the middle mouse button is pressed, the document to which the LinkGadget points, has to be loaded. Then, it has to be parsed, and finally it has to be shown in the TextGadget which contains the LinkGadget.
Before the TextGadget can show the new document, first the TextGadget has to be found. When the LinkGadget is clicked, it receives an InputMsg. This message is broadcast, and the objects that previously received this message (and passed it on), are kept in a list. In this path, the TextGadget (and the panel which contains it) can be found.

Images

A new gadget is created for images. It is a subclass of the existing PictureGadget. Unfortunately, the PictureGadget cannot load images in the formats .GIF and .JPEG, which are usually used within HTML documents.
The newer version of Oberon System 3 (version 2) does support these image types. Since this version will become available soon for SPARC machines, no effort has been made to extend the PictureGadget.
This means that in the current implementation, the images will not be shown on screen. Instead, a default picture is shown. When the new version becomes available, it will be easy to change the program to show the real images.

Fonts and Headers

The start of a new font is marked by an 'Invisible' gadget.
This gadget can either be visible or not, when it is visible it will show the tag in HTML representation (the attributes of the tag are not shown). The switching between visible and not visible can be done with the command `Invisible.Flip`.

There is a global boolean variable (defined in module Invisible), which marks whether the gadgets are 'visible' or 'invisible'. When the value changes, all invisible gadgets are notified. This is done with the existing model-view-controller mechanism.
The end of a font is marked likewise. The text between these to gadgets is put in a corresponding font.

Font changes can be made with the command `HeaderFonts.SetFont`. The task of this routine is not just to change the font of the marked text, but also to make sure that the text stays a correct html text (that is: make sure begin and end tags match).

Example

Suppose the text looks like this:
`this is an <it> example text</it>, and contains no real info.`

The text beginning at 'text' and ending at 'contains' is selected (here indicated by the underlining).
`this is an <it> example text</it>, and contains no real info.`

When the selected text is changed into bold font, the text should become:
`this is an <it> example </it><b>text, and contains</b> no real info.`

In the example, some existing tags have to be moved. In other cases, tags have to be deleted. The following rules can be used:

- If both the begin and end tag are inside the selected text, remove both.
- If only a begin tag is in the selected text, move in to the end of the text.
- If only an end tag is in the selected text, move it to the beginning of the text.

Special characters

The standard Oberon fonts contain a number of special characters used in HTML. The file 'SpecialChars.Test' contains a list of pairs (html representation, oberon character number). When the HTML document is parsed, the special characters are put in Oberon representation.
The special character tool is made from the list of characters. For each element of the list, a button is made and placed in the panel. When a button is pressed, the character will be inserted at the caret. This is done by sending an InputMsg, containing the character, to the window in which the caret is set.

Not all special characters contained in HTML are also available in the standard Oberon fonts. Currently, these characters are not supported by the editor. To support them the following changes can be made:

- Edit the fonts. The newer Oberon System 3 (version 2.0) contains a font editor. A lot of characters in the standard Oberon fonts are not used, so these characters can be changed.
- Create a new font containing only special characters.

The code concerning special characters is put in the module 'SpecialChars'.

Line Breaks
A line break is represented by a gadget showing an arrow (the one shown on a lot of keyboard on the enter key). The text after the line break should start on the next line. Unfortunatly, there is no way to assure this: a user can just set the cursor after the gadget, and insert text. See 10.2 for a possible solution to this problem.

Forms
Currently, only a few of the form related tags are supported. A form is represented by a subclass of TextGadget. Two types of the input tag are implemented: buttons and textfields.
Each form specifies a program which should be executed when the form is submitted. Browsers use the HTTP-daemon, but when the program is on the local disk, this is not necessary. When the program is executed directly, the development of the form and the program can be done on a machine without HTTP-daemon. To submit a form, the following steps have to be done:

1. Get the data from all the form elements.
2. Encode the data in the X-WWW-FORM-URLENCODED format.
3. When the HTTP-daemon is used, get the new document.
4. When a local program is used:
   (a) Create an environment in which the environment-variables 'CONTENT_TYPE', 'CONTENT_LENGTH' and 'REQUEST_METHOD' are set.
   (b) Create an input-file in which the form's content is put.
   (c) Redirect the standard input (to the input-file) and standard output.
   (d) Execute the program.
5. Show the document returned.

7.1 Utility modules

Module AssocList
An `AList` is a list of pairs (string, string). When the first string is given, the associated second string can be retrieved. Such a list can be loaded from a file.

Module Dirs

This module is used not by the editor, but by the FileSelector. The FileSelector is a reusable component, for the selection of a file by the user. When a file is chosen (clicked), a StringGadget is sent to all selected gadgets. When a LinkGadget receives such a message, it copies the string to its href attribute.
This module contains the code that is executed when a user clicks on a directory or on a file. When a directory is clicked, the Lists in the FileSelector panel are filled with the directories and files in the new directory. When a file is clicked by the user, an object containing the filename is sent to all selected objects.

ExternalRoutines

The routines that are used from shared-object-libraries, are linked in the module.

ExternalProgs

The editor only supports the html format. Documents in other formats, such as postscript, GIF, etc. can be viewed by using external viewers. These viewers can be specified in the file 'ExternalProgs.Text'. This file contains pairs (extension, program). When a file with a certain extension should be shown, the program associated with that extension is executed.

Module Strings

This module contains some functions for handling `ARRAY OF CHAR`s. These functions are: comparing, concatenation of strings, determining the length of a string, etc.

Module Tags

For a lot of HTML tags, new subclasses of existing gadgets had to be made. One of the extensions of the existing gadget is usually that the new gadget should have more attributes.
There are two ways you can add new attributes:

- Statically. The record is extended with the new attributes. The following code shows how this is done:

```
TYPE
  LinkGadget* = POINTER TO LinkGadgetDesc;
  LinkGadgetDesc* = RECORD (TextGadgets.FrameDesc)
       href* : ARRAY 200 OF CHAR;  (* destination of hyperlink *)
       name* : ARRAY 200 OF CHAR;  (* labelname of this anchor *)
  END;

PROCEDURE LinkAttr(L : LinkGadget; VAR M : Objects.AttrMsg);
(*  AttributeMessage for LinkGadgets *)
BEGIN
   IF M.id = Objects.get THEN
      IF M.name = "Gen" THEN M.class := Objects.String;
```

```
        COPY("LinkGadgets.NewLinkGadget", M.s); M.res := 0
      ELSIF M.name = "href" THEN M.class := Objects.String;
        COPY(L.href, M.s); M.res := 0
      ELSIF M.name = "name" THEN M.class := Objects.String;
        COPY(L.name, M.s); M.res :=0
      ELSE  TextGadgets.FrameHandler(L, M);
      END
    ELSIF M.id = Objects.set THEN
      IF M.name = "href" THEN
        IF M.class = Objects.String THEN COPY(M.s,L.href);
            M.res := 0 END;
      ELSIF M.name ="name" THEN
        IF M.class = Objects.String THEN COPY(M.s,L.name);
            M.res := 0 END;
      ELSE TextGadgets.FrameHandler(L, M);
      END
    ELSIF M.id = Objects.enum THEN
      (* only the attribute that the linkgadget defines are returned *)
      M.Enum("href");
      M.Enum("name");
    END
  END LinkAttr;

  PROCEDURE Handle(O : Objects.Object; M : Objects.ObjMsg);
  BEGIN
    IF M IS Objects.AttrMsg THEN
      LinkAttr(O(LinkGadget),M(AttrMsg));
    ELSIF
      .
      .
      .
    END;
  END Handle;
```

The amount of code that has to be produced is quite large.

- Dynamically. The new attributes are put in a linked list. The module Tags has code to simplify the coding. The addition of new attributes does not require extra code in the gadget class. Most code of editor uses this approach. See module Tags and module LinkGadgets for the code.

Module TextUtil

This module contains functions concerning Texts. An often used function is `InsertObjectInText`. As the name says, it inserts an object into a text.
A text is a list of pairs (library, reference number), so to be able to insert an object in a text, the object has to be put in a library.

The Oberon System 3 designers chose to store the reference number (of type INTEGER) in texts in one byte. This is done because font libraries never contain more than 255 objects, and since texts consist mostly of 'normal' characters, a considerable amount of storage space is saved.
Consequently, the objects have to be one of the first 255 objects in a library (when you try to insert an object with reference number 256, you will actually enter the object with reference number 0). HTML texts can easily contain more than 255 objects, so the objects can often not fit in a single library.
The module TextUtil has a variable of type library. This library is used by the procedure `InsertObjectInText`. When the library is 'full', a new one is created.

Chapter 8

To be done

Some shortcomings of the current implementation:

- The deletion of part of a document may make the document inconsistent.

  For instance, the start of an item list has to be marked. If this marker is deleted (and the items themselves not), the text is no longer correct HTML. The same for copying. The solution is similar to the modifying of textfonts (see Implementation, headers and fonts).

- Not all types of lists are implemented.

- Not all FORM elements are implemented, but the ones that are missing can implementented similar to those that are implemented.

- The proposed extension to the gadget system. Of course, this is much simpler when the source code of the Gadgets system is available.

- HTML specification 3.0 is not supported. Supporting the new HTML 3.0 tags (tables, math, etc.) would mean a lot of work, but would basically be done similar to the implemented ones.

Chapter 9

HTML 3.0

The WWW community is continuously wanting more features. This resulted in yet another HTML level, HTML 3.0. Presently, (August 1995) HTML 3.0 is still under development. The HTML 3.0 specification [19] provides a number of new features, and is broadly backwards compatible with HTML 2.0.
New features:

- Tables
- Math
- FIG for inline figures
- fine positioning control (e.g. horizontal tabs)
- existing tags can have numerous other attributes

9.1 New features

Tables
Tables can contain a wide range of content, such as headers, lists, paragraphs, forms, figures, preformatted text and even nested tables.
Tables start with an optional caption followed one or more rows. Each row is formed by one or more cells, which are differentiated into header and data cells. Cells can be merged across rows and columns.
Example:

```
<TABLE BORDER>
  <CAPTION>A test table with merged cells</CAPTION>
  <TR><TH ROWSPAN=2><TH COLSPAN=2>Average
      <TH ROWSPAN-2>other<BR>category<TH>Misc
  <TR><TH>height<TH>weight
  <TR><TH ALIGN=LEFT>males<TD>1.9<TD>0.003
```

```
  <TR><TH ALIGN=LEFT ROWSPAN=2>females<TD>1.7<TD>0.002
</TABLE>
```

Browsers should show something like:

A test table with merged cells

	Average		other category	Misc
	height	weight		
males	1.9	0.003		
females	1.7	0.002		

Math
The <MATH> element is used to include math expressions in the current line. The math expressions are made in a way similar to LaTeX. Example: the integral from a to b of f(x) over 1+x:

```
<MATH>&int;_a^b^{f(x)<over>1+x} dx</MATH>
```

Figures
The FIG element is used for figures. A figure can have, besides a picture:

- overlay picture(s), which will be placed over the picture.
- a caption
- a description text which is intended to convey the content of the figure for people with non-graphical user agents
- graphical hypertext links which are interpreted by the user agent (usually browser) rather than the server. If the user agent cannot show pictures, the links are shown as conventional hypertext links.

9.2 Adapting the editor to HTML 3.0

The idea behind the editor can remain the same. HTML tags are replaced by Gadgets. Three new complex Gadgets have to be made:

- Table Gadget.
- Math Gadget

- FIG Gadget

Remember: all gadgets are responsible for there own editing. This means that the three gadgets will become quite complex.

Chapter 10

Evaluation

The project started with two research questions:

- Is it possible to create a program that you can use both as a browser and as an editor for html document?
- Is Oberon System 3 with Gadgets suited for such a program?

The answer to the first question is simple: yes. The created editor is not comparable with browsers such as netscape or mosaic, but gives a pretty good idea of how the mentioned browsers show them. The reason for the differences is simple: to be able to edit the document, you will need some information about its structure. For instance, in a browser, you cannot see where a form starts, and where is ends. When you want to edit the document, you will need this information.
Some information such as the start and end of list, can be either hidden or shown. When this information is hidden, the document will be very readable. When you need this information, you can show it, and hide it when you are finished.
The second question can be divided into two sub-questions:

- How hard is it to create this type of program in Oberon System 3 with gadgets?
- Would it be easier/harder to make this program in another programing language/ environment?

The first question: The Oberon System 3 datastructure 'Text' can be used to internally store the HTML document, so no new datastructure has to be created. In Oberon System 3 it relatively easy to make new gadgets for the representation of tags. Less easy is it to make a completely satisfying viewer/editor for the Text. The existing TextGadget was used for this, but has some shortcomings. Section 10.2 proposes an extension to the Gadgets system that will improve the flexibility of the TextGadget.
The second question is harder to answer, since the Oberon System was the only programming environment tried. So, this has to be guessed. The gadget system is related to the way Smalltalk handles visible objects, so presumably the effort of implementing the editor in Smalltalk will be comparable. Developing the program in C/C++ with X or Motif will probably be more difficult, since you can relatively re-use little code.

10.1 Criticism on Oberon System 3

- Crashing. As a programmer it is not very difficult to make the Oberon environment crash. One of the reasons is that the environment has no multiprocessing. If you run a program that contains an endless loop, you have a crash of the Oberon environment. Normal users will have considerable less problems.

- Lack of source code / good documentation. The documentation is not organised at all: you get a number of text files, often explaining the same, and not complete at all.

  Since the documentation is poor, you would like to have the full source code, which is unfortunately not contained in the package.

- Differences in the packages for different machines. The packages for PCs and for SUN-OS machines contain different files.

- The user interface differs too greatly from the well known user interfaces (X, Microsoft Windows, Macintosh). New users will need an unnecessary long time to get used to this interface.

  Some weak points in the design of the Oberon user interface:

  - The menu bar has no pull down menus: you just get a number of buttons on a row and that's it. You will soon find out that there can only be a very small number of buttons in the menu.
  - The Oberon environment gives little feedback. To resize a window, you move the cursor to the a point near the border of the window. In other GUIs, you get a different cursor indicating that you can now resize the window. In the Oberon environment, you will just have to try and see if you're in the correct place, the cursor will remain the same.

- The interfaces to the different types of objects could be made more consistent.

  For instance, sometimes a handler is called 'Handle', and sometimes 'Handler', and not all modules export an object initialiser (for creating subclasses!).

10.2 Proposed extension to the gadget system

Currently, in Oberon System 3 a gadget is only responsible for its own appearance. When a gadget is placed in a container, the container is responsible for the position of the gadget. The gadget cannot give information to the container about where it wants to be put. So, I propose to change this. I would like to introduce a new message type (subclass of FrameMsg), say FormatPreferenceMsg. A container gadget can send this message to an object, and the object will return how it would like to be formated.

The following information can be included:

- Offset to the 'line'. Now, this can be changed by calling the function `Texts.ChangeLooks`. This function cannot be called by the object itself, because the object does not 'know' when it is placed in a Text (it receives no message).
- Setting leftmargin, rightmargin (relative to previous margins)
- Should the line be broken after the object
- Should the object be placed on a new line (no objects on the left of this object are allowed)
- Allow text to be floated on the right/left of the object. This is useful to allow text to float around (large) pictures.

Containers are allowed to ignore this information, and objects are not obliged to give this information. This means that this is a real extension: every object that is written for the current system will still work in the new system.
When this mechanism is implemented, the flexibility of the gadgets system will be increased.

Chapter 11

Conclusions and suggestions for further research

Oberon System 3 has a very flexible way of handling text. It can not only contain characters, but also arbitrary objects. This means that the usage of an editor can be extended simply by creating new types of objects. Any editor will be able to work with them. So, for instance, if someone implements a new class for editing mathematical functions, every editor can then handle mathematical functions.

This system showed to be very handy in creating an HTML editor. The basic functions of the editor do not need to be written, which saves a lot of time.

Ofcourse, there are always some little things you would like to improve. Especially the extension to the gadget system proposed in 10.2

Oberon System 3 is not yet a mature product. A lot of extensions have to be made, but the basis is good. To let Oberon System 3 become a success, some important improvements have to be made:

- Improve the documentation
- Make the UI look and behave more like the other 'important' GUIs like X, Microsoft Windows and Macintosh.

Appendix A

User Manual

A.1 Using Oberon

To start Oberon System 3, execute the command `oberon`. Oberon will startup with a full screen window. If your terminal hasn't got enough memory, you will need to start oberon with a small window (e.g. `oberon -geometry 600x400`). Figure A.1 shows the startup screen.

The upper right window is the system log. In the window below is an Edit window, in which the file System.Tool is loaded.

Mouse button functions

The mouse is heavily used, and Oberon users will need to learn the functions of the mouse buttons.

Left. Point at object. Usually, this sets the caret.

Middle. Execute a command.

Right. Select an object or group of objects.

Left interclick Middle. (press left button (and keep it down), press amd release middle button, and release left button) Copy selected objects to mouse cursor.
Middle interclick Right. Consume object.

If you click the middle mouse button, the text under the cursor is executed. The text must be of the syntax `Module.Procedure`. Examples: `System.Quit`, `System.Time`. When a parameter is needed, it can be typed right after the command. Example:
`Script.Open Welcome.Text`. (shown by oberon as ↑).

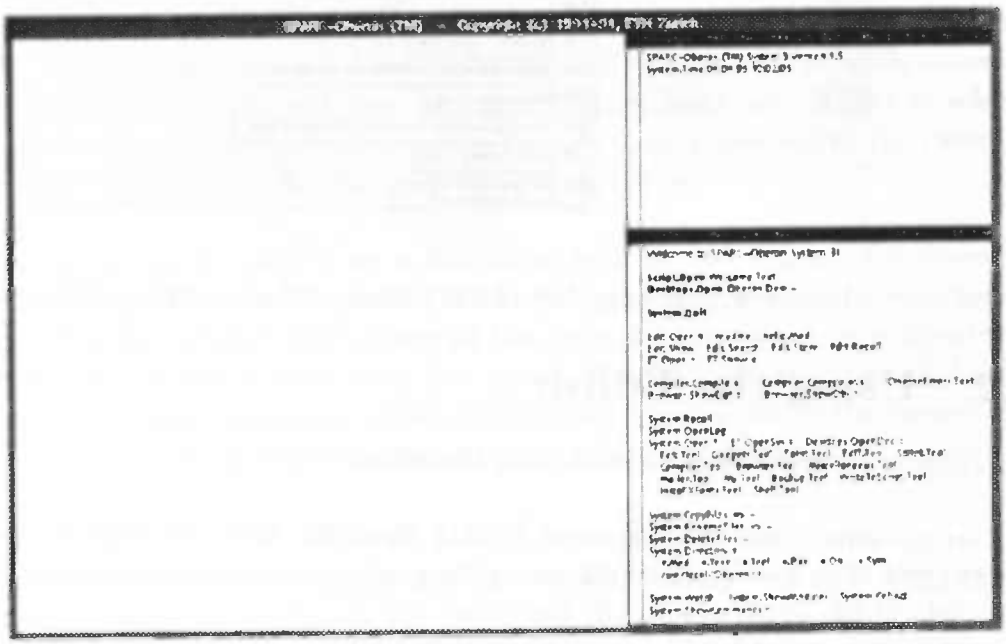

Figure A.1: The startup screen of oberon 3

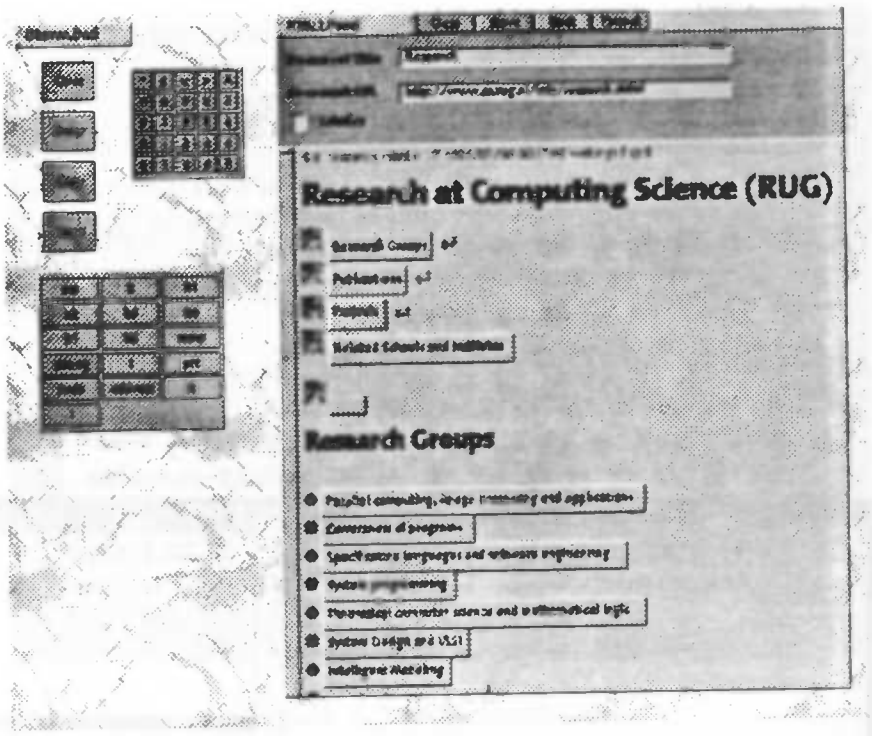

Figure A.2: A snapshot of the editor, and some editor tools.

A.2 Using the Editor

This chapter describes how to work with the editor.

To start the editor, use the command `HTEdit.OpenURL <URL>` or `HTEdit.OpenLocalFile <FILENAME>`. The first command opens a file given its http address, the second command opens a local file.
When the command is executed, a Panel (window) is opened in which the text is visible. The panel has the following buttons:

- **Close.** This closes the window.
- **Copy.** Creates a copy of the panel. In mosaic this button is call 'clone'.
- **Grow.** The window becomes larger, until it takes the full oberon window.
- **Store.** Save the document.
- **Back.** Move back to the previous document.
- **Format.** Reformats the text so that will look almost like the document is shown in mosaic.

Figure A.2 shows the editor, and some tools for editing. The same document, is shown by Mosaic in figure 2.2 The editing is done just like in the normal Oberon textgadgets, e.g. all the mouse clicks still apply.

The next paragraphs will discuss how the HTML tags are represented in the text, and how to edit them.

Comments

Comments are shown in purple. To set text in comment, first select the text, and then execute the command `HTEdit.Comment`. To uncomment text, use `HTEdit.Uncomment`.

Hyperlinks

Hyperlinks (in HTML language: anchors, the `<A>` tag), are shown inside a window. The text inside this window is blue. A link can be followed by clicking it (middle mouse button). A new link can be created with the command `HTEdit.NewLink`. If text is selected, this text will be inside the new link. Otherwise, the new link is inserted at the caret.
The attributes of a link can be changed with the inspector.

The FileSelector can be used to set a link to an existing file. Figure A.3 shows the FileSelector. A textfield labeled 'Current Directory' shows the path of the directory you are looking at. The list labeled 'File' shows all files contained by the current directory. To let a hyperlink mark to one of these files, first select the hyperlink, and then click on the filename. The list labeled 'Directory' shows all directories contained by the current directory. So see the files located in a directory, click on the directory name.

Lists

The start end and tags are normally not shown. Items are shown as a black dot. To see where the list begins and ends, use the command `Invisible.Flip`. Now, the tags will be shown. The tags can be hidden again by another execution of `Invisible.Flip`.

A new list can be made by first selecting the part of the text that will be inside the new list. Then, execute the command `HTEdit.CreateList`. The begin and end tag will be inserted, and a list item is insert at the beginning. More items can be created with the command `HTEdit.NewItem`. The new item will start at the caret position.

Pictures

For each picture in a HTML text, a default picture is shown. This is due to the fact that Oberon System 3 cannot show pictures stored in the used formats (GIF, JPEG). However, when the picture is clicked (middle mouse button), an external viewer is started that will show the 'real' picture.

A new picture can be created with the command `HTEdit.NewPicture`, or by clicking on the 'New Picture' button in the HTMLToolbox. The new picture will be inserted at the

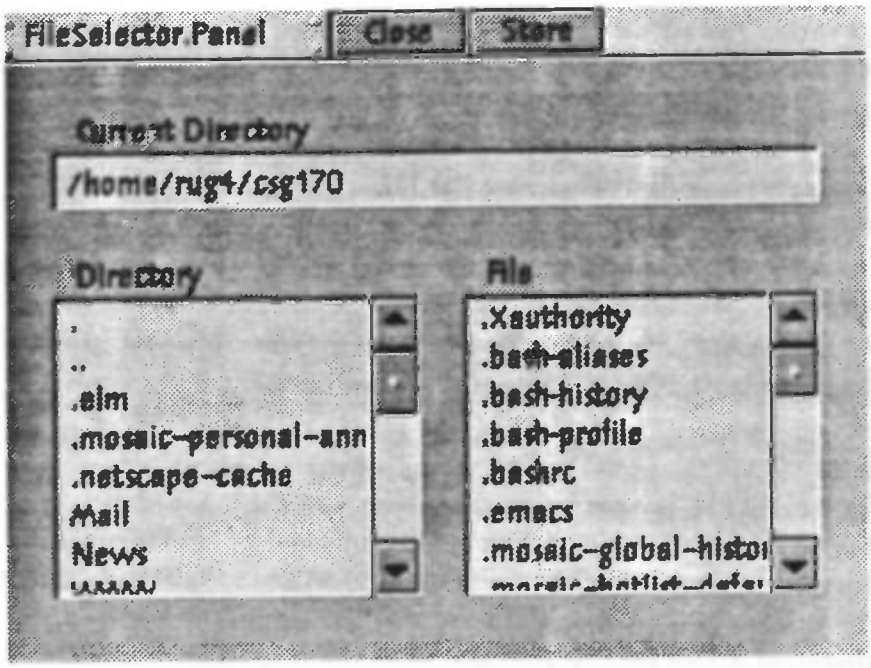

Figure A.3: The FileSelector

caret.

The attributes of a picture can be modified with the inspector.

Special Characters

Special characters, which are coded in html as `&string;`, are shown as the 'real' character. So, `è` will be shown as é.
A little tool is used to insert special characters. The tool can be opened by using the command `Gadgets.Insert SpecialChars.NewSpecialCharMap~`. A panel (window) is opened, which contains a button for each special character. To insert a special character, select a position in the text, and click one of the buttons. The character will be inserted at the selected position.

Headers and Fonts

A tool to insert headers/fonts in the text is opened with the command `Gadgets.Insert HeaderFonts.NewFontMap~`. A panel is opened which contains a button for each font/header. Changing a font is done as follows: First, select the part of the text that you want to modify. Then, click the button with the wanted font. The selected text will now be shown in the new font.

Forms

Forms are shown in a window with a white background. The attributes of the form can be modified with the inspector. New forms can be created with the command

`HTEdit.CreateForm`. The 'action' attribute contains the program that should be executed when the form is submitted. This program can be called using HTTP (this method is used by browsers), or called directly (which can be used for debugging new programs on machines without HTTP deamon). In the last case, the full path of the program should be given, preceded by 'file://localhost'.

Appendix B

Source Code

B.1 AssocList

```
MODULE AssocList;

(*
    An Assolist is a list of pairs (string, string).
    When one of both strings is given, the other can be found.
    These lists can be read from file.
*)

IMPORT
    Texts, Out, TextFrames, Strings, Unix, SYSTEM, Attributes;

TYPE
   AList* = POINTER TO AListDesc;
   AListDesc* = RECORD
        a*          : ARRAY 50 OF CHAR;
        b*           : ARRAY 64 OF CHAR;
        next*        : AList;
    END;

VAR viewers : AList;

PROCEDURE Load*(VAR al : AList; filename : ARRAY OF CHAR);
VAR r : Texts.Scanner;
       t : Texts.Text;
       last : AList;
BEGIN
  last:=NIL;
  t:=TextFrames.Text(filename);
  Texts.OpenReader(r,t,0);

  WHILE (~r.eot) DO
       Texts.Scan(r);
       IF ((r.class = Texts.Char) & (r.c = '(')) THEN
           (* create new AList record *)
```

```
            IF last=NIL THEN
               NEW(al);
               last:= al;
            ELSE
               NEW(last.next);
               last:=last.next;
            END;
            last.next := NIL;

            Texts.Scan(r);
             IF r.class=Texts.Name THEN
                   COPY(r.s, last.a);
                   (*Strings.CopyString(last.a , r.s);*)
             END;
             Texts.Scan(r);  (* comma *)
             Texts.Scan(r);  (* name of external viewer *)
             IF r.class = Texts.Name THEN
                   COPY (r.s, last.b);
             ELSIF r.class = Texts.Int THEN
                   Attributes.IntToStr(r.i, last.b);
             END;
             Texts.Scan(r); (* ')' *)
         END;
    END;
END Load;

PROCEDURE GetAssoc*(al : AList; a : ARRAY OF CHAR;  VAR b : ARRAY OF CHAR);
(* given a extension, find a program to view it *)
BEGIN
  WHILE ((al#NIL) & (~Strings.CompareIgnoreUpcase(a , al.a))) DO
      al:=al.next;
  END;
  IF al#NIL THEN
     COPY(al.b,b);
  ELSE
      b[0]:=0X;
  END;
END GetAssoc;

PROCEDURE GetAssoc2*(al : AList; VAR a : ARRAY OF CHAR;  b : ARRAY OF CHAR);
(* given a extension, find a program to view it *)
BEGIN
  WHILE ((al#NIL) & (~Strings.CompareIgnoreUpcase(b , al.b))) DO
      al:=al.next;
  END;
  IF al#NIL THEN
     COPY(al.a,a);
  ELSE
      a[0]:=0X;
  END;
END GetAssoc2;
```

```
PROCEDURE HasAssoc2* (al : AList; b : ARRAY OF CHAR) : BOOLEAN;
(* returns TRUE if there is an viewer associated with extension, otherwise FALSE *)
VAR a : ARRAY 200 OF CHAR;
BEGIN
   GetAssoc2(al, a, b);
   RETURN(a[0]#0X);
END HasAssoc2;

PROCEDURE HasAssoc* (al : AList; a : ARRAY OF CHAR) : BOOLEAN;
(* returns TRUE if there is an viewer associated with extension, otherwise FALSE *)
VAR b : ARRAY 200 OF CHAR;
BEGIN
   GetAssoc(al, a, b);
   RETURN(b[0]#0X);
END HasAssoc;

PROCEDURE PrintAList* (al : AList);
BEGIN
    WHILE (al#NIL) DO
        Out.String(al.a); Out.String(" : "); Out.String(al.b); Out.Ln;
        al:=al.next;
    END;
END PrintAList;

PROCEDURE Test*;
VAR al : AList;
BEGIN
    Load(al,"FontHeaders.Text");
    PrintAList(al);
END Test;

BEGIN
```

B.2 Dirs

```
MODULE Dirs;

IMPORT
    Oberon, Objects, Texts, Out, Display, Unix, Gadgets, Lists, SYSTEM, BasicGadgets,
    ExternalRoutines, TextUtil, Strings;

PROCEDURE SetFile*;
VAR t : Texts.Text;
        beg,i : LONGINT;
        r : Texts.Reader;
        fileName, s ,cd: ARRAY 200 OF CHAR;
        o : Objects.Object;
        ch : CHAR;
        par : Oberon.ParList;
        selm : Display.SelectMsg;
```

```
        conm : Display.ConsumeMsg;
        b : BasicGadgets.String;
BEGIN
    BasicGadgets.NewString;
    b:=Objects.NewObj(BasicGadgets.String);

    par:=Oberon.Par;
    t:=par.text; beg:=par.pos;
    Texts.OpenReader(r,t,beg);
    Texts.Read(r,ch);

    WHILE (ch=' ') DO Texts.Read(r,ch); END;
    i:=0;
    WHILE ((ch#' ') & ~r.eot) DO
        fileName[i]:=ch;    i:=i+1;
        Texts.Read(r,ch);
    END;
    fileName[i]:=0X;

    o:=Gadgets.FindObj(Gadgets.context, "CurDir");
    COPY(o(BasicGadgets.String).val, cd);

    s:="file://localhost";
    Strings.Concat(s, cd);
    Strings.Concat(s,"/");
    Strings.Concat(s, fileName);

    (* find all selected objects *)
    selm.F:=NIL;
    selm.id:=Display.get;
    selm.time:=-1;
    selm.sel:=NIL; selm.obj:=NIL;

    Display.Broadcast(selm);

    COPY(s, b.val);
    WHILE selm.obj#NIL DO
        (* drop a BasicGadget.String, containt the selected filename, on the gadget *)
        conm.F:=selm.obj(Display.Frame);
        conm.id:=Display.drop;
        conm.obj:=b;
        conm.u:=0;
        conm.v:=0;
        Display.Broadcast(conm);
        selm.obj:=selm.obj.dlink;
    END;
END SetFile;

PROCEDURE Directory*;
VAR t : Texts.Text;
        beg,i : LONGINT;
        r : Texts.Reader;
```

```
        ch : CHAR;
        dir : ARRAY 200 OF CHAR;
        dirListName, fileListName, com, oldDir, curDir  : ARRAY 200 OF CHAR;
        par : Oberon.ParList;
        dirList, fileList, o : Objects.Object;
        rm : Display.ModifyMsg;
        b : BOOLEAN;
        am : Objects.AttrMsg;
        oldw, oldh: INTEGER;
        cm : Display.ControlMsg;
        it : Lists.Item;
BEGIN
    par:=Oberon.Par;
    t:=par.text; beg:=par.pos;

    ExternalRoutines.GetWD(oldDir);
    o:=Gadgets.FindObj(Gadgets.context, "CurDir");
    COPY(o(BasicGadgets.String).val, curDir);
    i:=Unix.Chdir(SYSTEM.ADR(curDir));

    Texts.OpenReader(r,t,beg);
    Texts.Read(r,ch);
    WHILE ch=' ' DO Texts.Read(r,ch); END;
    i:=0;
    WHILE (ch#' ') DO
        dirListName[i]:=ch; i:=i+1;
        Texts.Read(r,ch);
    END;
    dirListName[i]:=0X;

    WHILE ch=' ' DO Texts.Read(r,ch); END;
    i:=0;
    WHILE (ch#0X) & (ch#0AX) & (ch#0DX) & (ch#9X) & (ch#' ') DO
        fileListName[i]:=ch;    i:=i+1;
        Texts.Read(r,ch);
    END;
    fileListName[i]:=0X;

    WHILE (ch=' ') DO Texts.Read(r,ch); END;
    i:=0;
    WHILE (ch#' ') DO
        dir[i]:=ch; i:=i+1;
        Texts.Read(r,ch);
    END;
    dir[i]:=0X;

    dirList:=Gadgets.FindObj(Gadgets.context, dirListName);
    fileList:=Gadgets.FindObj(Gadgets.context, fileListName);
    IF ((dirList#NIL) & (fileList#NIL)) THEN
        i:=Unix.Chdir(SYSTEM.ADR(dir));

        (* empty dirList *)
```

```
it:=dirList(Lists.List).items;
WHILE it.prev#NIL DO it:=it.prev; END;
WHILE it#NIL DO
    it.sel:=TRUE;
    it:=it.next;
END;
Lists.DeleteSelection(dirList(Lists.List));

(* empty fileList *)
it:=fileList(Lists.List).items;
WHILE it.prev#NIL DO it:=it.prev; END;
WHILE it#NIL DO
    it.sel:=TRUE;
    it:=it.next;
END;
Lists.DeleteSelection(fileList(Lists.List));


ExternalRoutines.Dir(dirList(Lists.List), fileList(Lists.List));

WITH dirList : Lists.List DO
    rm.F:=dirList; rm.id:=Display.extend;
    rm.mode:=Display.display; rm.F:=dirList(Lists.List);
    rm.X:=dirList.X; rm.Y:=dirList.Y;
    rm.W:=30; rm.H:=30;
    rm.dX:=0; rm.dY:=0; rm.dW:=dirList.W-rm.W; rm.dH:=dirList.H-rm.H;
    oldw:=dirList.W; oldh:=dirList.H;
    Display.Broadcast(rm);
    b:=TRUE;
    Lists.DeselectList(dirList, b);
END;

WITH dirList : Lists.List DO
    rm.F:=dirList; rm.id:=Display.extend;
    rm.mode:=Display.display; rm.F:=dirList(Lists.List);
    rm.X:=dirList.X; rm.Y:=dirList.Y; rm.W:=oldw; rm.H:=oldh;
    rm.dX:=0; rm.dY:=0; rm.dW:=dirList.W-rm.W; rm.dH:=dirList.H-rm.H;
    Display.Broadcast(rm);
END;

WITH fileList : Lists.List DO
    rm.F:=fileList; rm.id:=Display.extend;
    rm.mode:=Display.display; rm.F:=fileList(Lists.List);
    rm.X:=fileList.X; rm.Y:=fileList.Y;
    rm.W:=30; rm.H:=30;
    rm.dX:=0; rm.dY:=0; rm.dW:=fileList.W-rm.W; rm.dH:=fileList.H-rm.H;
    oldw:=fileList.W; oldh:=fileList.H;
    Display.Broadcast(rm);
END;

WITH fileList : Lists.List DO
    rm.F:=fileList; rm.id:=Display.extend;
```

```
            rm.mode:=Display.display; rm.F:=fileList(Lists.List);
            rm.X:=fileList.X; rm.Y:=fileList.Y; rm.W:=oldw; rm.H:=oldh;
            rm.dX:=0; rm.dY:=0; rm.dW:=fileList.W-rm.W; rm.dH:=fileList.H-rm.H;
            Display.Broadcast(rm);
        END;
    END;

    (* set current dir back, put first copy current dir to the TextFiel 'Current Dir' *)
    ExternalRoutines.GetWD(curDir);
    COPY(curDir, o(BasicGadgets.String).val);
    BasicGadgets.SetValue(o(BasicGadgets.String));

    i:=Unix.Chdir(SYSTEM.ADR(oldDir));
END Directory;
```

B.3 ExternalProgs

```
MODULE ExternalProgs;

(*
      A user editable file 'Viewers.Text', contains
     a list of pairs ( <extension> , <viewername> ),
      in which is defined what viewer should be user for each
     extension
*)

IMPORT Texts, Out, Unix, SYSTEM, AssocList, ExternalRoutines;

VAR viewers : AssocList.AList;

PROCEDURE Init;
BEGIN
    AssocList.Load(viewers, "ExternalViewers.Text");
END Init;

PROCEDURE GetProg*(ext : ARRAY OF CHAR;  VAR prog : ARRAY OF CHAR);
(* given a extension, find a program to view it *)
BEGIN
    AssocList.GetAssoc(viewers, ext, prog);
END GetProg;

PROCEDURE IsViewerAssociated* (ext : ARRAY OF CHAR) : BOOLEAN;
(* returns TRUE if there is an viewer associated with extension, otherwise FALSE *)
VAR prog : ARRAY 200 OF CHAR;
BEGIN
    RETURN (AssocList.HasAssoc(viewers, ext));
END IsViewerAssociated;

PROCEDURE GetFileName*(prog : ARRAY OF CHAR; VAR name : ARRAY OF CHAR);
```

```
VAR
    i, j : INTEGER;
BEGIN
    i:=0; j:=0;
    WHILE prog[i]#0X DO
        name[i-j]:=prog[i];
        IF prog[i]='/' THEN j:=i+1; END;
        i:=i+1;
    END;
    name[i-j]:=0X;
END GetFileName;

PROCEDURE ShowFile*(extension, filename : ARRAY OF CHAR);
VAR prog, progname : ARRAY 200 OF CHAR;
        argv : ARRAY 10 OF POINTER TO ARRAY 40 OF CHAR;
        res, envp : LONGINT;
BEGIN
     GetProg(extension, prog);

     IF prog[0]#0X THEN
        GetFileName(prog, progname);

        Out.String(prog);
        Out.Ln; Out.String(progname); Out.Ln;
        NEW(argv[0]); NEW(argv[1]); NEW(argv[2]); NEW(argv[3]);
        argv[4]:=NIL; argv[3]:=NIL;

        COPY(prog,argv[0]^);
        COPY(progname,argv[0]^);
        COPY(filename, argv[1]^);
        COPY("-display",argv[2]^);

        ExternalRoutines.get(envp);
        SYSTEM.PUT(SYSTEM.ADR(argv[3]), envp);

        res:=Unix.Fork();
        IF res#0 THEN
             Unix.Execve(SYSTEM.ADR(prog),SYSTEM.ADR(argv), SYSTEM.ADR(argv[4]));
        END;
        Out.String("ready"); Out.Ln;
     ELSE
        Out.String("Sorry, no viewer  associated with extension .");
        Out.String(extension); Out.Ln;
     END;
END ShowFile;

BEGIN
  Init;
```

B.4 ExternalRoutines

```
MODULE ExternalRoutines;

IMPORT Objects, Display, Gadgets, Attributes, Oberon, Kernel,SYSTEM,Out, Texts, Unix, Lists;

VAR
    libHTML : LONGINT;
    libc : LONGINT;
    redirectlib : LONGINT;

    HTEscape : PROCEDURE ( str : LONGINT; ch : CHAR) : LONGINT;
    loadHtmlPage : PROCEDURE ( Adr : LONGINT) : LONGINT;
    getenv : PROCEDURE (nameAdr : LONGINT) : LONGINT;
    opendir : PROCEDURE (dirName : LONGINT) : LONGINT;
    readdir : PROCEDURE (dir : LONGINT) : LONGINT;
    stat : PROCEDURE (pathAdr : LONGINT; buf : LONGINT) : LONGINT;
    getwd : PROCEDURE (pathname : LONGINT) : LONGINT;
    wait : PROCEDURE (i : LONGINT) : LONGINT;
    redirect : PROCEDURE;

     PROCEDURE StringLen(s : ARRAY OF CHAR) : LONGINT;
     VAR i : LONGINT;
     BEGIN
          i:=1;
          WHILE ORD(s[i])#0 DO i:=i+1; END;
          RETURN(i);
     END StringLen;

     PROCEDURE AppendToString (VAR s : ARRAY OF CHAR; t : ARRAY OF CHAR);
     VAR i,j,lent : LONGINT;
     BEGIN
        j:=StringLen(s);
        lent:=StringLen(t);
        i:=0;
         WHILE i<lent+1 DO
              s[j+i]:=t[i];
              i:=i+1;
         END;
     END AppendToString;

    PROCEDURE get* (VAR envp : LONGINT);
    VAR
        name : ARRAY 20 OF CHAR;
        dis : POINTER TO ARRAY 200 OF CHAR;
    BEGIN
        name:="DISPLAY";
        envp:=getenv(SYSTEM.ADR(name));
        SYSTEM.PUT(SYSTEM.ADR(dis), envp);
    END get;

    PROCEDURE GetWD*(VAR wd : ARRAY OF CHAR);
```

```
VAR
    name : ARRAY 20 OF CHAR;
    temp : LONGINT;
    res : POINTER TO ARRAY 200 OF CHAR;
BEGIN
    name:=".";
    temp:=getwd(SYSTEM.ADR(name));
    SYSTEM.PUT(SYSTEM.ADR(res), temp);
    Out.String("Pwd: "); Out.String(res^); Out.Ln;
    COPY(res^, wd);
END GetWD;

PROCEDURE Dir* (VAR f : Lists.List; h : Lists.List);
VAR p : POINTER TO Unix.Dirent;
        a,d, e,i, isdir, res, buf: LONGINT;
        b,g : BOOLEAN;
        name : ARRAY 20 OF CHAR;
        statbuf : Unix.Status;
        oldw, oldh: INTEGER;
BEGIN
    isdir:=16384;
    name:=".";

    d:=opendir(SYSTEM.ADR(name));
    WHILE e#0 DO
        e:=readdir(d);
        IF e#0 THEN
            SYSTEM.PUT(SYSTEM.ADR(p), e);
            res:=stat(SYSTEM.ADR(p.name), SYSTEM.ADR(statbuf));

            g:=TRUE;
            FOR i:=0 TO 15 DO
                b:=SYSTEM.BIT(SYSTEM.ADR(isdir),i);
                IF b THEN
                        a:=statbuf.mode;
                        b:=SYSTEM.BIT(SYSTEM.ADR(a),i);
                        g:=(g & b);
                END;
            END;
            IF g THEN
                Lists.InsertItem(f,p.name);
            ELSE
                Lists.InsertItem(h,p.name);
            END;
        END;
    END;
END Dir;

PROCEDURE Load* (s : ARRAY  OF CHAR);
VAR result : LONGINT;
BEGIN
    Out.String("Loading: "); Out.String(s); Out.Ln;
```

```
        result := loadHtmlPage(SYSTEM.ADR(s));
    END Load;

    PROCEDURE Encode* (VAR s : ARRAY OF CHAR);
    (* makes the string a encoded string. This encoding is used when
        forms are submitted *)
    VAR
        p : POINTER TO ARRAY OF CHAR;
        res : LONGINT;
        cp : ARRAY 200 OF CHAR;
        ch : CHAR;
    BEGIN
            (* WARNING: POSSIBLE WRITING IN ILLIGAL
                MEMORY (4 bytes before returned string) *)
        ch:=CHR(1);
        COPY(s, cp);
        res:=HTEscape(SYSTEM.ADR(cp), ch);
        res:=res-4;
        SYSTEM.PUT(SYSTEM.ADR(p), res);
        COPY(p^, s);
    END Encode;

    PROCEDURE Redirect*;
    BEGIN
        redirect;
    END Redirect;

    PROCEDURE Wait*(i : LONGINT) : LONGINT;
    VAR j : LONGINT;
    BEGIN
        RETURN(wait(i));
    END Wait;

BEGIN
    (* open the libraries used *)
    libHTML := Kernel.dlopen("/home/rug4/csg170/oberon/libhtml.so.1.1",1);
    IF libHTML # 0 THEN
        Kernel.dlsym(libHTML,"load_html_page",SYSTEM.VAL(LONGINT, loadHtmlPage));
        Kernel.dlsym(libHTML,"HTEscape",SYSTEM.VAL(LONGINT, HTEscape));
    ELSE
        Out.String("Error: libhtml.so.1.1 could not be mapped"); Out.Ln;
    END;

    libc:= Kernel.dlopen("/usr/lib/libc.so.1.9",1);
    IF libc # 0 THEN
        Kernel.dlsym(libc,"opendir",SYSTEM.VAL(LONGINT, opendir));
        Kernel.dlsym(libc,"readdir",SYSTEM.VAL(LONGINT, readdir));
        Kernel.dlsym(libc,"stat", SYSTEM.VAL(LONGINT,stat));
        Kernel.dlsym(libc,"getenv",SYSTEM.VAL(LONGINT, getenv));
        Kernel.dlsym(libc,"getwd",SYSTEM.VAL(LONGINT, getwd));
        Kernel.dlsym(libc,"wait", SYSTEM.VAL(LONGINT, wait));
    ELSE
```

```
        Out.String("Error: libc.so.1.9 could not be mapped");
    END;

    redirectlib:=Kernel.dlopen("/home/rug4/csg170/oberon/redirect.so.1.1",1);
    IF redirectlib#0 THEN
        Kernel.dlsym(redirectlib,"redirect", SYSTEM.VAL(LONGINT, redirect));
    ELSE
        Out.String("Error: redirectlib.so.1.1 could not be mapped");
    END;
```

B.5 Format

```
MODULE Format;

IMPORT
        Oberon, Objects, Texts,  Strings, Out, Fonts, Display, Figures,
        Invisible, ListGadgets, TextUtil;

CONST
    commentColor = 100;


    PROCEDURE FormatText*(t : Texts.Text; rightMargin : INTEGER);
    VAR
        newLine, breakIsSpace : BOOLEAN;
        w, i, noCR : INTEGER;
        r : Texts.Reader;
        ch : CHAR;
        ob : Objects.Object;
        lastFont : Objects.Library;
        lastSpaceX, offset : INTEGER;
        lastSpacePos, temp: LONGINT;
        nospace : BOOLEAN;

        PROCEDURE InsertCRBefore;
        BEGIN
            temp:=Texts.Pos(r);
            TextUtil.InsertCharInText(0AX, t, temp-1);
            Texts.OpenReader(r,t, temp+1);
        END InsertCRBefore;

        PROCEDURE InsertCR(offset : INTEGER);
        BEGIN
            temp:=Texts.Pos(r);
            TextUtil.InsertCharInText(0AX, t, temp+offset);
            Texts.OpenReader(r,t, temp+1);
        END InsertCR;

        PROCEDURE MoveOn;
        BEGIN
```

```
        IF ob IS Fonts.Char THEN
            IF ((ch=0AX) OR (ch=0DX)) THEN
                w:=4;  lastSpacePos:=Texts.Pos(r);
                lastSpaceX:=w; breakIsSpace:=FALSE;
                newLine:=TRUE;
            ELSE
                w:=w+ob(Fonts.Char).w+2; newLine:=FALSE;
                IF ch=" " THEN
                    lastSpacePos:=Texts.Pos(r);
                    lastSpaceX:=w; breakIsSpace:=TRUE; newLine:=FALSE;
                END;
            END
        ELSE (* ob is Display.Frame *)
            w:=w+ob(Display.Frame).W+1; newLine:=FALSE;
            lastSpacePos:=Texts.Pos(r);
            lastSpaceX:=w; breakIsSpace:=TRUE;
        END;
    END MoveOn;

    PROCEDURE ReadNext;
    BEGIN
        IF ~r.eot THEN
            Texts.Read(r,ch);
            IF ~r.eot THEN
                r.lib.GetObj(r.lib, ORD(ch), ob);
            END;
        END;
    END ReadNext;

BEGIN
    (* remove all carriage returns, except those in comments and in <pre> .. </pre? *)
    Texts.OpenReader(r,t,0);
    ReadNext;
    WHILE ~r.eot DO
        IF ((r.col#commentColor) & ((ch=0AX) OR (ch=0DX)) & (ob IS Fonts.Char)) THEN
            temp:=Texts.Pos(r);
            Texts.Delete(t, temp-1, temp);
            TextUtil.InsertCharInText(" ", t, temp-1);
            Texts.OpenReader(r,t, temp);
        END;
        IF ((ob IS Invisible.Frame) &
                 Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "<pre>")) THEN
            (* skip everything up to </pre> *)
            WHILE ~((ob IS Invisible.Frame) &
                 Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "</pre>")) DO
                ReadNext;
            END;
        END;
        Texts.Read(r,ch);
        IF ch#0X THEN r.lib.GetObj(r.lib, ORD(ch), ob); END;
    END;
```

```
Texts.OpenReader(r, t, 0);
w:=4;
lastSpacePos:=Texts.Pos(r);
lastSpaceX:=w;
breakIsSpace:=TRUE;
newLine:=FALSE;
nospace:=FALSE;

ReadNext;
WHILE (~r.eot (* & (ch#0X) *) ) DO
    IF (*ch#0X *) TRUE THEN
    WHILE ((r.col=commentColor) & (ob IS Fonts.Char)) DO
        MoveOn;
        ReadNext;
    END;  (**)
    IF ob IS Fonts.Char THEN
        IF ch=" " THEN
            IF (nospace & (Texts.Pos(r)#0)) THEN
                (* delete space *)
                temp:=Texts.Pos(r);
                Texts.Delete(t, temp-1, temp);
                Texts.OpenReader(r, t, temp-1);
                lastSpacePos:=Texts.Pos(r);
            ELSIF  (w>rightMargin) THEN
                (* insert carriage return just befor last space *)
                temp:=Texts.Pos(r);
                TextUtil.InsertCharInText(0AX, t, lastSpacePos);
                (* Delete the space *)
                IF breakIsSpace THEN
                    (* Delete the space *)
                    Texts.Delete(t, lastSpacePos-1, lastSpacePos);
                    Texts.OpenReader(r, t, temp);
                ELSE
                    Texts.OpenReader(r, t, temp+1);
                END;
                newLine:=FALSE;
                w:=4+w-lastSpaceX; lastSpacePos:=Texts.Pos(r);
                lastSpaceX:=w; breakIsSpace:=FALSE;
            ELSE
                lastSpacePos:=Texts.Pos(r);
                lastSpaceX:=w;
                breakIsSpace:=TRUE; nospace:=TRUE;
            END;
        ELSE
            newLine:=FALSE; nospace:=FALSE;
            w:=w+ob(Fonts.Char).w+2;
        END;
    ELSE (* ob is NOT a Fonts.Char *)
            nospace:=FALSE;
            w:=w+ob(Display.Frame).W+1;
            IF (ob IS Figures.PolyLine) THEN
                IF ~newLine THEN
```

```
        (* insert newline before hr *)
        InsertCRBefore;
    END;
    (* insert newline after hr *)
    temp:=Texts.Pos(r);
    TextUtil.InsertCharInText(0AX, t, temp);
    Texts.OpenReader(r,t, temp+1);
    w:=4;  lastSpaceX:=w; lastSpacePos:=Texts.Pos(r);
ELSIF ((ob IS Invisible.Frame) &
        (Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "<P>") OR
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "<h1>") OR
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "<h2>") OR
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "<h3>") OR
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "<h4>") OR
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "<h5>") OR
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "<h6>") OR
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "</h1>") OI
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "</h2>") OI
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "</h3>") OI
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "</h4>") OI
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "</h5>") OI
         Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "</h6>") )
            ) THEN
    IF ob(Invisible.Frame).Caption[1]='/' THEN offset:=1;
    ELSE offset:=0; END;
    IF ~newLine THEN
        (* insert newline before paragraph/ header *)
        InsertCR(-1+offset);
    END;
    (* insert another newline before paragraph / header *)
    InsertCR(-1+offset);
    w:=4;  lastSpaceX:=w; lastSpacePos:=Texts.Pos(r); newLine:=TRUE;
    nospace:=TRUE; breakIsSpace:=FALSE;
ELSIF ((ob IS Invisible.Frame) &
            Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "<pre>")
    (* skip everything up to </pre> *)
    WHILE ~((ob IS Invisible.Frame) &
            Strings.CompareIgnoreUpcase(ob(Invisible.Frame).Caption, "</pre>"
        MoveOn;
        ReadNext;
    END;
ELSIF (ob IS Invisible.Frame) THEN
    w:=w+ob(Display.Frame).W+1;
ELSIF (~newLine & (ob IS Figures.Circle)) THEN
    (* item, insert CR before it *)
    InsertCRBefore;
    w:=4+ob(Display.Frame).W+1;
    lastSpaceX:=w;
    lastSpacePos:=Texts.Pos(r);
ELSIF ((w>rightMargin) & (~(ob IS ListGadgets.brFrame))) THEN
    (* insert carriage return just befor last space *)
    temp:=Texts.Pos(r);
```

```
                    TextUtil.InsertCharInText(0AX, t, lastSpacePos);
                    (* Delete the space *)
                    IF breakIsSpace THEN
                        (* Delete the space *)
                        Texts.Delete(t, lastSpacePos-1, lastSpacePos);
                        Texts.OpenReader(r, t, temp);
                    ELSE
                        Texts.OpenReader(r, t, temp+1);
                    END;
                    newLine:=FALSE; breakIsSpace:=FALSE;
                    w:=4+w-lastSpaceX;
                END;
                IF (ob IS ListGadgets.brFrame) THEN
                    nospace:=TRUE;
                    temp:=Texts.Pos(r);
                    TextUtil.InsertCharInText(0AX, t, temp);
                    w:=4; lastSpaceX:=w; lastSpacePos:=temp;
                    Texts.OpenReader(r, t, temp+1);
                END;
                lastSpacePos:=Texts.Pos(r);
                lastSpaceX:=w;
                breakIsSpace:=FALSE; newLine:=FALSE;
            END;
        END;    ReadNext;
        END;
    END FormatText;
```

B.6 FormGadgets

```
MODULE FormGadgets;

IMPORT
    Objects, Oberon, Texts, TextGadgets, TextGadgets0, Out, Gadgets,
    Display, Attributes, Unix, SYSTEM, TextFrames,
    Panels, TextFields, BasicGadgets, Files, Store, Strings, Tags, ExternalRoutines,
    ExternalProgs, Loader;


TYPE
    FormGadget* = POINTER TO FormGadgetDesc;
    FormGadgetDesc* = RECORD (TextGadgets.FrameDesc)
       action*   : ARRAY 200 OF CHAR;
       method* : ARRAY 200 OF CHAR;
       enctype* : ARRAY 200 OF CHAR;
    END;

    TextField* = POINTER TO TextFieldDesc;
    TextFieldDesc* = RECORD (TextFields.TextFieldDesc)
        al : Tags.AttrList;
```

```
    END;

    Button* = POINTER TO ButtonDesc;
    ButtonDesc* =  RECORD(BasicGadgets.ButtonDesc);
        al : Tags.AttrList;
    END;

PROCEDURE SubmitFormLocal(prog, vars : ARRAY OF CHAR);
VAR
    len: POINTER TO ARRAY 300 OF CHAR;
    argv, env : ARRAY 5 OF POINTER TO ARRAY 100 OF CHAR;
    f : Files.File;
    o : Files.Rider;
    res, res2 : LONGINT;
    p : ARRAY 100 OF CHAR;
BEGIN
    NEW(len); NEW(argv[0]); NEW(argv[1]); NEW(argv[2]); argv[3]:=NIL;
    NEW(env[0]); NEW(env[1]); NEW(env[2]); env[3]:=NIL;
    COPY(prog, p);

    env[0]^:="REQUEST_METHOD=POST";
    env[1]^:="CONTENT_TYPE=application/x-www-form-urlencoded";
    env[2]^:="CONTENT_LENGTH=";
    Attributes.IntToStr(Strings.Length(vars), len^);
    Strings.Concat(env[2]^,len^);

    argv[1]^:="< inp";
    ExternalProgs.GetFileName(prog, argv[0]^);
    argv[2]^:="> tmp";
    argv[1]:=NIL;

    f:=Files.New("inp");
    Files.Set(o, f, 0);
    Strings.WriteStrToFile(o, vars);
    Files.Register(f);
    Files.Close(f);

    res:=Unix.Fork();
    IF res#0 THEN
        res2:=ExternalRoutines.Wait(0);
    ELSE
        ExternalRoutines.Redirect;
        Unix.Execve(SYSTEM.ADR(p), SYSTEM.ADR(argv), SYSTEM.ADR(env));
    END;
END SubmitFormLocal;

PROCEDURE Test*;
BEGIN
    SubmitFormLocal("/home/rug4/csg170/htmlprogs/a.out","a=4&b=10");
END Test;

PROCEDURE GetStringAttr(o: Objects.Object; n : ARRAY OF CHAR; VAR val : ARRAY OF CHAR);
```

```
VAR am : Objects.AttrMsg;
BEGIN
    am.id:=Objects.get;
    COPY(n, am.name);
    am.s[0]:=0X;
    am.res:=-1;

    o.handle(o, am);
    COPY(am.s, val);
END GetStringAttr;

PROCEDURE GetFormValues(F : FormGadget; VAR s : ARRAY OF CHAR);
VAR
    r : Texts.Finder;
    o : Objects.Object;
    first : BOOLEAN;
    name, val : ARRAY 100 OF CHAR;
BEGIN
    first:=TRUE;
    Texts.OpenFinder(r, F.text, 0);
    WHILE ~r.eot DO
        Texts.FindObj(r, o);
        IF (o#NIL) & ((o IS TextField) OR (o IS Button)) THEN
            (* append attributename=value to s *);
            GetStringAttr(o, "name", name);
            GetStringAttr(o, "value", val);

            ExternalRoutines.Encode(name);
            ExternalRoutines.Encode(val);
            IF (name[0]#0X) & (val[0]#0X) THEN
                IF ~first THEN
                    Strings.Concat(s, "&");
                END;
                Strings.Concat(s,name);
                Strings.Concat(s,"=");
                Strings.Concat(s,val);
                first:=FALSE;
            END;
        END;
    END; (* while *)
END GetFormValues;

PROCEDURE FindPanel(VAR p : Panels.Panel);
VAR obj : Objects.Object;
        name : ARRAY 200 OF CHAR;
BEGIN
    p:=NIL;
    obj:=Oberon.Par.obj;
    WHILE obj#NIL DO
        IF obj IS Panels.Panel THEN
            Gadgets.GetObjName(obj,name);
            IF name="thepanel" THEN p:=obj(Panels.Panel); END;
```

```
        END;
        obj:=obj.dlink;
    END;
END FindPanel;

PROCEDURE FindForm(VAR f : FormGadget);
VAR obj : Objects.Object;
        name : ARRAY 200 OF CHAR;
BEGIN
    f:=NIL;
    obj:=Oberon.Par.obj;
    WHILE obj#NIL DO
        IF obj IS FormGadget THEN
            Gadgets.GetObjName(obj,name);
            f:=obj(FormGadget);
        END;
        obj:=obj.dlink;
    END;
END FindForm;

PROCEDURE SubmitForm*;
VAR
    f : FormGadget;
    p : Panels.Panel;
    s, prog, protocol : ARRAY 3000 OF CHAR;
    t : Texts.Text;
BEGIN
    FindPanel(p);
    FindForm(f);

    GetStringAttr(f, "action", s);

    Loader.GetProtocol(s, protocol);
    IF Strings.CompareIgnoreUpcase(protocol, "HTTP") THEN
        Strings.Concat(s, "?");
        GetFormValues(f, s);
        Loader.Load(s, p);
    ELSIF Strings.CompareIgnoreUpcase(protocol, "FILE") THEN
        Loader.GetFile(s, prog); s[0]:=0X;
        GetFormValues(f, s);
        SubmitFormLocal(prog, s);

        t:=TextFrames.Text("tmp_out");
        Loader.ParseAndShow(t, "Response to Form", p);
    ELSE
        Out.String("Sorry, protocol "); Out.String(protocol);
        Out.String(" not supported..."); Out.Ln;
    END;
END SubmitForm;

PROCEDURE StoreInput(VAR o : Files.Rider; al : Tags.AttrList);
BEGIN
```

```
    Strings.WriteStrToFile(o,"<INPUT ");
    Tags.Store(o, al);
    Strings.WriteStrToFile(o,">");
END StoreInput;

PROCEDURE StoreForm(VAR o : Files.Rider; F : FormGadget);
BEGIN
    Strings.WriteStrToFile(o,"<FORM");
    IF F.action[0]#0X THEN
        Strings.WriteStrToFile(o," ACTION=");
        Strings.WriteStrToFile(o,F.action);
    END;
    IF F.method[0]#0X THEN
        Strings.WriteStrToFile(o," METHOD=");
        Strings.WriteStrToFile(o,F.method);
    END;
    IF F.enctype[0]#0X THEN
        Strings.WriteStrToFile(o," ACTION=");
        Strings.WriteStrToFile(o,F.enctype);
    END;
    Strings.WriteStrToFile(o,">");
    Store.Store(o, F.text);
    Strings.WriteStrToFile(o,"</FORM>");
END StoreForm;


PROCEDURE FormAttr(F : FormGadget;  VAR M : Objects.AttrMsg);
(*  AttributeMessage for LinkGadgets *)
BEGIN
     IF M.id = Objects.get THEN
        IF M.name = "Gen"
            THEN M.class := Objects.String; COPY("FormGadgets.NewFormGadget", M.s); M.res := 0
        ELSIF Strings.CompareIgnoreUpcase(M.name ,"action")
            THEN M.class := Objects.String ; COPY(F.action, M.s) ; M.res := 0
        ELSIF Strings.CompareIgnoreUpcase(M.name ,"method")
            THEN M.class := Objects.String; COPY(F.method, M.s); M.res :=0
        ELSIF Strings.CompareIgnoreUpcase(M.name ,"enctype")
            THEN M.class := Objects.String; COPY(F.enctype, M.s); M.res :=0
        ELSE  TextGadgets.FrameHandler(F, M);
        END
    ELSIF M.id = Objects.set THEN
        IF  Strings.CompareIgnoreUpcase(M.name ,"action") THEN
            IF M.class = Objects.String THEN COPY(M.s,F.action);  M.res := 0 END;
        ELSIF  Strings.CompareIgnoreUpcase(M.name ,"method") THEN
            IF M.class = Objects.String THEN COPY(M.s,F.method); M.res :=0 END;
        ELSIF Strings.CompareIgnoreUpcase(M.name ,"enctype") THEN
            IF M.class = Objects.String THEN COPY(M.s,F.enctype); M.res :=0 END;
        ELSE TextGadgets.FrameHandler(F, M);
        END
    ELSIF M.id = Objects.enum THEN
        (* only the attribute that the linkgadget defines are returned *)
        M.Enum("action");
```

```
        M.Enum("method");
        M.Enum("enctype");
    END
END FormAttr;

PROCEDURE FormHandle* (F: Objects.Object; VAR M: Objects.ObjMsg);
BEGIN
    IF M IS Objects.AttrMsg THEN FormAttr(F(FormGadget), M(Objects.AttrMsg))
    ELSIF ((M IS Objects.FileMsg) & (M(Objects.FileMsg).id=Objects.store)) THEN
        StoreForm(M(Objects.FileMsg).R, F(FormGadget));
    ELSE TextGadgets.FrameHandler(F,M);
    END;
END FormHandle;

PROCEDURE NewFormGadget*;
VAR f :FormGadget;
    cmesg : Objects.CopyMsg;
BEGIN
    NEW(f);
    TextGadgets.NewNote;

    cmesg.id:=Objects.deep; cmesg.obj:=f;
    TextGadgets.CopyFrame(cmesg,Objects.NewObj(TextGadgets.Frame),f);
    f.handle:=FormHandle;
    f.col:=-2;
    f(TextGadgets0.Frame).state0:={1,2,3,5,6,7,8,9,10,11,15};
    Objects.NewObj:=f;
END NewFormGadget;

PROCEDURE TextFieldHandler*(F: Objects.Object; VAR M: Objects.ObjMsg);
VAR a : Tags.Attr;
BEGIN
    IF M IS Objects.AttrMsg THEN
        IF Strings.CompareIgnoreUpcase (M(Objects.AttrMsg).name, "value") THEN
            M(Objects.AttrMsg).name:="Value";
            TextFields.TextFieldHandler(F,M);
        ELSIF ((M(Objects.AttrMsg).id = Objects.get) & (M(Objects.AttrMsg).name="Gen"))
        THEN M(Objects.AttrMsg).class := Objects.String;
                COPY("FormGadgets.NewTextField", M(Objects.AttrMsg).s);
                M(Objects.AttrMsg).res := 0;
        ELSE Tags.HandleAttr(F(TextField).al,M(Objects.AttrMsg)); END;
    ELSIF ((M IS Objects.FileMsg) & (M(Objects.FileMsg).id=Objects.store)) THEN
        StoreInput(M(Objects.FileMsg).R, F(TextField).al);
    ELSE TextFields.TextFieldHandler(F,M);
    END;
END TextFieldHandler;

PROCEDURE ButtonHandler* (F:Objects.Object; VAR M: Objects.ObjMsg);
VAR a : Tags.Attr;
        am : Objects.AttrMsg;
BEGIN
    IF (M IS Oberon.InputMsg) & (1 IN M(Oberon.InputMsg).keys) THEN
```

```
        Oberon.Par.obj:=M.dlink;
        SubmitForm;
    ELSIF M IS Objects.AttrMsg THEN
        IF ((M(Objects.AttrMsg).id = Objects.get) & (M(Objects.AttrMsg).name="Gen"))
            THEN M(Objects.AttrMsg).class := Objects.String;
                     COPY("FormGadgets.NewButton", M(Objects.AttrMsg).s);
                     M(Objects.AttrMsg).res := 0;
        ELSIF ((M(Objects.AttrMsg).id=Objects.set) &
                    Strings.CompareIgnoreUpcase(M(Objects.AttrMsg).name,"value") )
            THEN   am:=M(Objects.AttrMsg);
                        COPY("Caption",am.name);
                        BasicGadgets.ButtonHandler(F,am);
                        Tags.HandleAttr(F(Button).al,M(Objects.AttrMsg));
            ELSE  Tags.HandleAttr(F(Button).al,M(Objects.AttrMsg));
        END;
    ELSIF ((M IS Objects.FileMsg) & (M(Objects.FileMsg).id=Objects.store)) THEN
        StoreInput(M(Objects.FileMsg).R, F(Button).al);
    ELSE BasicGadgets.ButtonHandler(F,M);
    END;
END ButtonHandler;

PROCEDURE NewButton*;
VAR b : Button; am : Objects.AttrMsg;
BEGIN
    NEW(b);
    BasicGadgets.InitButton(b);

    am.name:="Cmd";
    am.id:=Objects.set;
    am.s:="FormGadgets.SubmitForm";
    am.res:=-1;
    b.handle(b, am);

    b.handle:=ButtonHandler;

    b.al:=NIL;
    Tags.AddString(b.al, "align","");
    Tags.AddString(b.al, "checked","");
    Tags.AddString(b.al, "maxlength","");
    Tags.AddString(b.al, "name","");
    Tags.AddString(b.al, "src","");
    Tags.AddString(b.al, "size","");
    Tags.AddString(b.al, "value","");

    b.H:=20;
    Objects.NewObj:=b;
END NewButton;


PROCEDURE NewTextField*;
VAR t : TextField;
BEGIN
```

```
    NEW(t);
    TextFields.InitTextField(t);

    t.handle:=TextFieldHandler;

    t.al:=NIL;
    Tags.AddString(t.al, "align","");
    Tags.AddString(t.al, "checked","");
    Tags.AddString(t.al, "maxlength","");
    Tags.AddString(t.al, "name","");
    Tags.AddString(t.al, "src","");
    Tags.AddString(t.al, "size","");
    Tags.AddString(t.al, "value","");

    Objects.NewObj:=t;
END NewTextField;



BEGIN
END FormGadgets.
```

B.7 HeaderFont

```
MODULE HeaderFont;

(* This module contains code for Headers and Fonts *)

IMPORT
    Objects,Out, Panels, BasicGadgets, Gadgets, Texts, Oberon,
    Strings, Invisible, AssocList, TextUtil;

TYPE
    FontElem = RECORD
        pos : LONGINT;
        fnt : ARRAY 20 OF CHAR;
        o : Objects.Object;
    END;

VAR hfList : AssocList.AList;

PROCEDURE IsHeaderOrFont* (s : ARRAY OF CHAR) : BOOLEAN;
BEGIN
    RETURN (AssocList.HasAssoc(hfList,s));
END IsHeaderOrFont;

PROCEDURE GetFont*(s : ARRAY OF CHAR; VAR l : Objects.Library);
VAR fntName : ARRAY 200 OF CHAR;
```

```
BEGIN
    AssocList.GetAssoc(hfList, s, fntName);
    l := Objects.ThisLibrary(fntName);
END GetFont;

PROCEDURE Init;
BEGIN
    Out.String("Loading font information"); Out.Ln;
    AssocList.Load(hfList,"HeaderFont.Text");
END Init;

PROCEDURE GetFontLib*(s: ARRAY OF CHAR; VAR fl : Objects.Library);
VAR l : Objects.Library;
BEGIN
    NEW(fl);
    GetFont(s,l);
    fl^:=l^;
END GetFontLib;

PROCEDURE NewFontMap*;
CONST
    buttonsizex=50;
    buttonsizey=20;
VAR p : Panels.Panel;
        b : BasicGadgets.Button;
        s : AssocList.AList;
        cx, cy: INTEGER;
        n : LONGINT;
        am : Objects.AttrMsg;
BEGIN
    NEW(p); Panels.InitPanel(p);
    s:=hfList;
    cx:=3; cy:=-25;
    WHILE s#NIL DO
        IF n>0 THEN
            BasicGadgets.NewButton;
            b:=Objects.NewObj(BasicGadgets.Button);
            COPY(s.a, b.caption);
            am.name:="Cmd";
            am.class:=Objects.String;
            am.id:=Objects.set;
            am.s:="HeaderFont.SetFont #Caption";
            b.handle(b,am);
            b.W:=buttonsizex; b.H:=buttonsizey;

            Panels.InsertChild(p, b, cx,cy);
            IF cx<70 THEN cx:=cx+buttonsizex+1;
            ELSE p.W:=cx+buttonsizex+2; cx:=3;  cy:=cy-buttonsizey-1;
            END;
        END;
        s:=s.next;
    END;
```

```
    IF cx=3 THEN cy:=cy+buttonsizey+1; END;
    p.H:=-cy+5;
    p.state:=p.state+{Gadgets.lockchildren};
    Objects.NewObj:=p;
END NewFontMap;


PROCEDURE SetFont*;
VAR
    fontName,s, s2  : ARRAY 20 OF CHAR;
    t : Texts.Text;
    beg, end, time, temp : LONGINT;
    f : Objects.Library;
    o : Objects.Object;
    fntStack : ARRAY 100 OF FontElem;
    fntStackP : INTEGER;
    r : Texts.Reader;
    ch : CHAR;

    PROCEDURE GetName(a : ARRAY OF CHAR; VAR b : ARRAY OF CHAR);
    VAR i,j : INTEGER;
    BEGIN
        IF a[1]='/' THEN i:=2 ELSE i:=1; END;
        j:=i;
        WHILE a[i]#'>' DO
            b[i-j]:=a[i];
            i:=i+1;
        END;
        b[i-j]:=0X;
    END GetName;

BEGIN
    TextUtil.GetParameter(fontName);
    Oberon.GetSelection(t, beg, end, time);

    IF t#NIL THEN
        GetFont(fontName, f);
        Texts.ChangeLooks(t, beg, end, {0}, f, 0, 0);

        s:="<"; Strings.Concat(s, fontName); Strings.Concat(s, ">");
        o:=Invisible.NewInvisible(s);
        TextUtil.InsertObjectInText(o, t, beg);

        s:="</"; Strings.Concat(s, fontName); Strings.Concat(s, ">");
        o:=Invisible.NewInvisible(s);
        TextUtil.InsertObjectInText(o, t, end+1);

        (* to do: check invisible frame represents a FONT tag *)

        fntStackP:=0;
        Texts.OpenReader(r, t, beg+1);
        WHILE Texts.Pos(r)#end+1 DO
```

```
            Texts.Read(r, ch);
            IF ~r.eot THEN
                r.lib.GetObj(r.lib, ORD(ch), o);
                IF (o IS Invisible.Frame) THEN
                    GetName(o(Invisible.Frame).Caption, s);
                    IF IsHeaderOrFont(s) THEN
                        IF (o(Invisible.Frame).Caption[1]='/') THEN

                            IF fntStackP>0 THEN
                                (* now, a pair of <SOME FONT> , </SOME FONT> is found *)
                                (* delete them *)
                                temp:=Texts.Pos(r);
                                (* first, delete end tag *)
                                Texts.Delete(t, temp-1, temp);
                                (* delete begin tag *)
                                Texts.Delete(t, fntStack[fntStackP-1].pos-1,
                                            fntStack[fntStackP-1].pos);
                                fntStackP:=fntStackP-1;
                                end:=end-2;
                                Texts.OpenReader(r, t, temp-2);
                            ELSE
                                (* a end font is found, move it to before
                                    the begining of the selected text *)
                                temp:=Texts.Pos(r);
                                Texts.Delete(t, temp-1, temp);
                                TextUtil.InsertObjectInText(o, t, beg);
                                Texts.OpenReader(r, t, temp);
                            END;
                        ELSE
                            (* push start font on stack *)
                            COPY(s, fntStack[fntStackP].fnt);
                            fntStack[fntStackP].pos:=Texts.Pos(r);
                            fntStack[fntStackP].o:=o;
                            fntStackP:=fntStackP+1;
                        END;
                    END;
                END;
            END;
        END; (* do *)
        (* move the tags on the stack to the end of the selected text *)
        WHILE fntStackP#0 DO
            TextUtil.InsertObjectInText(fntStack[fntStackP-1].o, t, end+2);
            Texts.Delete(t, fntStack[fntStackP-1].pos-1, fntStack[fntStackP-1].pos);
            fntStackP:=fntStackP-1; end:=end-1;
        END;
    END; (* if t#NIL *)
END SetFont;

BEGIN
    Init;
END HeaderFont.
```

B.8 HTEdit

```
MODULE HTEdit;

IMPORT
    Out, Files, Oberon, Objects, Display, ExternalRoutines, Gadgets, Panels, Texts,
    Documents, Desktops, TextFrames, TextGadgets, TextGadgets0, Strings, TextUtil,
    Store, MainPanel, Display3, Fonts, Format, Invisible, Parser, Loader;

VAR
    panel : Panels.Panel;
    toload : ARRAY 200 OF CHAR;

PROCEDURE ShowFont*;
VAR
    text : Texts.Text;
    beg, end, time : LONGINT;
    r : Texts.Reader;
    ch : CHAR;
    name : ARRAY 20 OF CHAR;
    i : INTEGER;
BEGIN
    Oberon.GetSelection(text, beg, end, time);
    Texts.OpenReader(r, text, beg);
    Texts.Read(r, ch);
    COPY(r.lib.name, name);
    i:=0;
    WHILE name[i]#'.' DO i:=i+1; END; name[i]:=0X;
    Out.String("Font: "); Out.String(name); Out.Ln;
END ShowFont;

PROCEDURE PanelHandler*(obj : Objects.Object; VAR M: Objects.ObjMsg);
(* if the panel resizes, also resize the textgadget call 'thetext' *)
VAR tg : Objects.Object;
        r : Display.ModifyMsg;
BEGIN
    IF ((M IS Display.ModifyMsg) & (M(Display.ModifyMsg).F=obj))THEN
        WITH M:Display.ModifyMsg DO
            IF ((M.id=Display.extend) OR (M.id=Display.reduce)) THEN
                    tg:=Gadgets.FindObj(obj, "thetext");
                    WITH tg: TextGadgets.Frame DO
                        r.id:=M.id;
                        r.F:=tg;
                        r.X:=tg.X; r.Y:=tg.Y-M.dH;
                        r.W:=M.W; r.H:=tg.H+M.dH;
                        r.dX:=0; r.dY:=-M.dH; r.dW:= r.W-tg.W; r.dH:=M.dH;
                        r.F:=tg;
                        Display.Broadcast(r);
                    END;
            END;
        END;
    END;
```

```
    Panels.PanelHandler(obj,M);
END PanelHandler;

PROCEDURE LoadPanel*(D: Documents.Document);
VAR obj : Objects.Object; main: Gadgets.Frame;
    F: Files.File; R: Files.Rider; name, name2: ARRAY 64 OF CHAR; ch: CHAR;
    tag,i : INTEGER; len: LONGINT; lib: Objects.Library;
    theframe: TextGadgets.Frame;
    r : Texts.Writer;
    thepanel : Panels.Panel;
    t : Texts.Text;
BEGIN
    main := NIL; D.W := 300; D.H := 200;
    F := Files.Old(D.name);
    IF F # NIL THEN
        Files.Set(R, F, 0); Files.ReadInt(R, tag);
        IF tag = Documents.Id THEN
            Files.ReadString(R, name); (* skip over tag *)
            Files.ReadInt(R, D.X); Files.ReadInt(R, D.Y);
            Files.ReadInt(R, D.W); Files.ReadInt(R, D.H);
            Files.Read(R, ch);
            IF ch = Objects.LibBlockId THEN
                NEW(lib); Objects.OpenLibrary(lib);
                Objects.LoadLibrary(lib, F, Files.Pos(R), len);
                lib.GetObj(lib, 0, obj); (* by default *)
                IF (obj # NIL) & (obj IS Gadgets.Frame) THEN main := obj(Gadgets.Frame) END;
                FOR i:=0 TO lib.maxref DO
                    lib.GetObj(lib,i,obj);
                    IF  ((obj#NIL) & (obj IS Panels.Panel)) THEN
                        obj.handle:=PanelHandler;
                        thepanel:=obj(Panels.Panel);
                        Gadgets.NameObj(thepanel,"thepanel");
                        panel := thepanel;
                    END;
                END;
            END
        END
    END;
    IF main = NIL THEN Panels.NewPanel; main := Objects.NewObj(Gadgets.Frame) END;
    Documents.Init(D, main);
    t:=TextFrames.Text(toload);
    Parser.Parse(t,thepanel);
    Format.FormatText(t, thepanel.W-14);
    theframe:=  MainPanel.GetTextGadget(thepanel);
    theframe.text:=t;
END LoadPanel;

PROCEDURE StorePanel*(D: Documents.Document);
VAR
    obj       : Objects.Object;
    fileName: ARRAY 200 OF CHAR;
    tf          : TextGadgets.Frame;
```

```
BEGIN
    IF D.name # "" THEN
        obj := D.dsc;
        IF obj # NIL THEN
            fileName:="test_out";
            Store.StoreFile(fileName, obj(Panels.Panel));
        END
    END
END StorePanel;

PROCEDURE FindPanel(VAR p : Panels.Panel);
VAR
    obj ,v : Objects.Object;
    d : Documents.Document;
    am : Objects.AttrMsg;
    lm : Objects.LinkMsg;
   s : ARRAY 200 OF CHAR;
BEGIN
    d:=Desktops.CurDoc(Gadgets.context);
    p:=d.dsc(Panels.Panel);
END FindPanel;

PROCEDURE B*;  (* Back *)
VAR
    f : Display.Frame;
    url : ARRAY 200 OF CHAR;
    p : Panels.Panel;
BEGIN
    FindPanel(p);
    f:=MainPanel.GetTextGadget(p);
    Loader.PopText(url);        (* url now contains current URL *)
    Loader.PopText(url);        (* and now the previous URL *)
    Out.String("Previous: "); Out.String(url); Out.Ln;
    Loader.Load(url, p);
END B;

PROCEDURE F*; (* format *)
VAR
    f : Display.Frame;
    t : Texts.Text;
    p : Panels.Panel;
BEGIN
    FindPanel(p);
    Out.String("Formatting text"); Out.Ln;
    f:=MainPanel.GetTextGadget(p);
    t:=f(TextGadgets.Frame).text;
    f(TextGadgets.Frame).text:=NIL;
    Format.FormatText(t, f.W-50);
    f(TextGadgets.Frame).text:=t;
    MainPanel.Reset(f(TextGadgets.Frame));
END F;
```

```
PROCEDURE DocHandle*(D: Objects.Object; VAR M: Objects.ObjMsg);
BEGIN
    WITH D: Documents.Document DO
        IF M IS Objects.AttrMsg THEN
            WITH M: Objects.AttrMsg DO
                IF M.id = Objects.get THEN
                    IF M.name = "Gen" THEN M.class := Objects.String;
                                            M.s := "HTEdit.NewDoc"; M.res := 0
                    ELSIF M.name = "Adaptive" THEN M.class := Objects.Bool;
                                                    M.b := FALSE; M.res := 0
                    ELSIF M.name = "Icon" THEN M.class := Objects.String;
                                                M.s := "Icons.Book"; M.res := 0
                    ELSIF M.name = "Menu" THEN
                        M.class := Objects.String;
                        M.s := "Desktops.StoreDoc[Store] HTEdit.B[Back] ";
                        Strings.Concat(M.s, "HTEdit.F[Format]"); M.res := 0
                    ELSE Documents.Handler(D, M)
                    END
                ELSE Documents.Handler(D, M)
                END
            END
        ELSE Documents.Handler(D, M)
        END
    END
END DocHandle;

PROCEDURE NewDoc*;
VAR D: Documents.Document;
BEGIN NEW(D); D.Load := LoadPanel; D.Store := StorePanel; D.handle := DocHandle;
    D.W := 250; D.H := 200;
    Objects.NewObj := D
END NewDoc;

(* Opening under program control *)

PROCEDURE ShowGadget(name: ARRAY OF CHAR);
VAR D: Documents.Document;
BEGIN
    NewDoc; (* create a new document *)
    D := Objects.NewObj(Documents.Document);

    COPY(name, D.name);
    D.Load(D);

    Desktops.ShowDoc(D, FALSE);
END ShowGadget;

PROCEDURE GetParameter(VAR s1: ARRAY OF CHAR);
VAR
    par     : Oberon.ParList;
    t       : Texts.Text;
    s       : Texts.Scanner;
```

```
    r        : Texts.Reader;
    i, beg, end, time    : LONGINT;
    ch     : CHAR;
BEGIN
    par:=Oberon.Par;
    Texts.OpenScanner(s, par.text, par.pos); Texts.Scan(s);
    IF (s.class=Texts.Char) & (s.c="^") OR (s.line # 0) THEN
        Oberon.GetSelection(t, beg, end, time);
    ELSE t:= par.text; beg:=par.pos; END;

    (* now, the parameter of the command is in text t, starting at position pos *)
    Texts.OpenReader(r,t,beg);
    Texts.Read(r,ch); WHILE (ch=' ') DO Texts.Read(r,ch); END;
    i:=0;
    WHILE (ch#0X) & (ch#0AX) & (ch#0DX) & (ch#9X) & (ch#' ') DO
        s1[i]:=ch;  i:=i+1;
        Texts.Read(r,ch);
    END;
    s1[i]:=0X;
END GetParameter;


PROCEDURE OpenURL*;
VAR
    s1     : ARRAY 200 OF CHAR;
BEGIN
    GetParameter(s1);

    Loader.PushText(s1);
    ExternalRoutines.Load(s1);
    toload:="html_file";
    ShowGadget("HTML2.Panel");
    MainPanel.SetURL(panel, s1);
END OpenURL;

PROCEDURE OpenLocalFile*;
VAR
    s1, s2, wd : ARRAY 200 OF CHAR;
BEGIN
    GetParameter(s1);
    IF s1[0]#'/' THEN
        (* filename is relative to current path, concat pwd to filename *)
        ExternalRoutines.GetWD(wd);
        Strings.Concat(wd,"/"); Strings.Concat(wd,s1);
        COPY(wd, s1);
    END;

    s2:="file://localhost/";
    Strings.Concat(s2,s1);

    Out.String("Loading: "); Out.String(s2); Out.Ln;
    Loader.PushText(s2);
```

```
    COPY(s1, toload);
    ShowGadget("HTML2.Panel");
    MainPanel.SetURL(panel, s2);
END OpenLocalFile;

PROCEDURE CreateLink*;
VAR
    text : Texts.Text; beg,end,time : LONGINT;
    title: ARRAY 200 OF CHAR;
    link : Display.Frame;
    W : Texts.Writer;
BEGIN
    Oberon.GetSelection(text,beg,end,time);

    link:= Gadgets.CreateViewModel("LinkGadgets.NewLinkGadget", "");
    Strings.GetStringFromText(text, beg, end, title);
    Texts.OpenWriter(W);
    Texts.WriteString(W,title);

    Texts.Append (link(TextGadgets0.Frame).text,W.buf);
    Texts.Delete(text,beg,end);
    Texts.ChangeLooks(link(TextGadgets0.Frame).text, 0, link(TextGadgets0.Frame).text.len,
                      {1}, NIL, Display3.blue, 0);
    TextUtil.InsertObjectInText(link, text, beg);
END CreateLink;

PROCEDURE CreateULList*;
VAR
    text : Texts.Text; beg,end,time : LONGINT;
    title: ARRAY 200 OF CHAR;
    o : Objects.Object;
    W : Texts.Writer;
BEGIN
    Oberon.GetSelection(text,beg,end,time);

    o:=Invisible.NewInvisible("<UL>");
    TextUtil.InsertObjectInText(o, text, beg);

    o:=Gadgets.CreateObject("ListGadgets.NewListItem");
    TextUtil.InsertObjectInText(o, text, beg+1);

    o:=Invisible.NewInvisible("</UL>");
    TextUtil.InsertObjectInText(o, text, end+2);
END CreateULList;

PROCEDURE CreateListItem*;
VAR
    o : Objects.Object;
BEGIN
    o:=Gadgets.CreateObject("ListGadgets.NewListItem");
    Gadgets.Integrate(o);
END CreateListItem;
```

```
PROCEDURE CreateForm*;
VAR
    text : Texts.Text; pos : LONGINT;
    form, f : Display.Frame;
    M : Oberon.CaretMsg;
BEGIN
     M.id:=Oberon.get; M.car:=NIL; M.text:=NIL; M.F:=NIL;
    Display.Broadcast(M);
    IF M.car = NIL THEN Out.String("caret not set");  Out.Ln
    ELSE
        text:=M.text;
        pos:=M.pos;
        f:=M.car;

     form:= Gadgets.CreateViewModel("FormGadgets.NewFormGadget", "");
        form.W:=f.W-23;
     TextUtil.InsertObjectInText(form, text, pos);
    END;
END CreateForm;

END HTEdit.
```

B.9 Img

```
MODULE Img;

IMPORT Objects, Gadgets, Pictures, PictureGadgets, Out, Tags, Loader,
            LinkGadgets, Oberon, Panels, Files, Strings;

TYPE
    Image* = POINTER TO ImageDesc;
    ImageDesc*= RECORD (PictureGadgets.PictDesc);
        al* : Tags.AttrList;
    END;

PROCEDURE Store(VAR o : Files.Rider; F : Image);
BEGIN
    Strings.WriteStrToFile(o, "<IMG ");
    Tags.Store(o, F(Image).al);
    Strings.WriteStrToFile(o, ">");
END Store;

PROCEDURE ImgHandler*(F: Objects.Object; VAR M: Objects.ObjMsg);
VAR scr, ext : ARRAY 200 OF CHAR; a : Tags.Attr; p : Panels.Panel;
BEGIN
    IF ((M IS Objects.FileMsg) & (M(Objects.FileMsg).id=Objects.store)) THEN
        Store(M(Objects.FileMsg).R, F(Image));
    ELSIF M IS Objects.AttrMsg THEN
        IF ((M(Objects.AttrMsg).id = Objects.get) & (M(Objects.AttrMsg).name="Gen"))
```

```
        THEN
            M(Objects.AttrMsg).class := Objects.String;
            COPY("Img.NewImg", M(Objects.AttrMsg).s);
            M(Objects.AttrMsg).res := 0;
        ELSE Tags.HandleAttr(F(Image).al,M(Objects.AttrMsg)); END;
    ELSIF M IS Oberon.InputMsg THEN
        IF 1 IN M(Oberon.InputMsg).keys
        THEN
            Oberon.Par.obj:=M.dlink;
            LinkGadgets.FindPanel(p);
            Tags.GetAttr(F(Image).al,"src",a);
            Loader.Load(a.s, p);
        ELSE PictureGadgets.PictHandle(F,M); END;
    ELSE PictureGadgets.PictHandle(F,M);
    END;
END ImgHandler;

PROCEDURE NewImg*;
VAR
    pic : PictureGadgets.Picture;
    im : Image;
    cmesg : Objects.CopyMsg;
BEGIN
    PictureGadgets.NewPict;
    NEW(im); pic:=Objects.NewObj(PictureGadgets.Picture);
    cmesg.id:=Objects.deep; cmesg.obj:=im;
    Pictures.Open(pic.pict, "Default.Pict", TRUE);

    Gadgets.CopyFrame(cmesg, pic, im);
    im.pict:=pic.pict;
    im.mode:=pic.mode;
    im.handle:=ImgHandler;

    im.al:=NIL;
    Tags.AddString(im.al, "align","");
    Tags.AddString(im.al,"src","");
    Tags.AddString(im.al,"alt","");
    Tags.AddBoolean(im.al,"ismap", FALSE);

    Objects.NewObj:=im;
END NewImg;


BEGIN
```

B.10 Invisible

```
MODULE Invisible;

IMPORT
```

```
    Objects, Texts,Out, Gadgets, Display, Display3, Effects, BasicGadgets, Files,
    Strings, Tags;

TYPE
    Frame* = POINTER TO FrameDesc;
    FrameDesc* = RECORD (Gadgets.FrameDesc)
        al* : Tags.AttrList;
        Caption* : ARRAY 100 OF CHAR;
        xsize* : INTEGER;
        b : BOOLEAN;
    END;

VAR b : BasicGadgets.Boolean;

PROCEDURE Store(VAR o : Files.Rider; F : Frame);
VAR i : INTEGER;
BEGIN
    i:=0;
    WHILE F.Caption[i]#'>' DO
        Files.Write(o, F.Caption[i]);
        i:=i+1;
    END;
    IF F.al#NIL THEN
        Files.Write(o," ");
    END;
    Tags.Store(o, F.al);
    Files.Write(o,">");
END Store;

PROCEDURE RestoreFrame(F: Frame; M: Display3.Mask; x, y, w, h: INTEGER);
BEGIN
    Display3.ReplConst(M, -10, x, y, w, h, Display.replace);

    Display3.String(M, Display3.textC,x+2,y,"Syntax10.Scn.Fnt", F.Caption, Display.paint);
    IF Gadgets.selected IN F.state THEN
        Display3.FillPattern(M, Display3.white, Effects.selectpat, x, y, x, y,
                             w, h, Display.paint)
    END
END RestoreFrame;

PROCEDURE Update(F : Objects.Object);
VAR
    A: Display.ModifyMsg;
BEGIN
    WITH F: Frame DO
        A.id:=Display.extend; A.mode:=Display.display;
        A.F:=F; A.X:=F.X; A.Y:=F.Y; A.H:=F.H;
        A.dX:=0; A.dY:=0; A.dH:=0;
        IF F.obj(BasicGadgets.Boolean).val THEN
            A.W:=1;
        ELSE
            A.W:=F.xsize;
```

```
        END;
        A.dW:=A.W-F.W;
        IF F.obj(BasicGadgets.Boolean).val#F.b THEN
            Display.Broadcast(A); F.b:=F.obj(BasicGadgets.Boolean).val;
        END;
    END;
END Update;

PROCEDURE FrameHandler*(F: Objects.Object; VAR M: Objects.ObjMsg);
VAR x, y, w, h: INTEGER; F0: Frame; R: Display3.Mask;
BEGIN
    IF ((M IS Objects.FileMsg) & (M(Objects.FileMsg).id=Objects.store)) THEN
         Store(M(Objects.FileMsg).R, F(Frame));
    ELSE
    WITH F: Frame DO
        IF M IS Objects.AttrMsg THEN
            IF ((M(Objects.AttrMsg).id = Objects.get) & (M(Objects.AttrMsg).name="Gen"))
            THEN
                M(Objects.AttrMsg).class := Objects.String;
                COPY("Invisible.NewFrame", M(Objects.AttrMsg).s);
                M(Objects.AttrMsg).res := 0;
            ELSE Tags.HandleAttr(F(Frame).al,M(Objects.AttrMsg)); END;
        ELSIF M IS Display.FrameMsg THEN
            WITH M: Display.FrameMsg DO
                IF (M.F = NIL) OR (M.F = F) THEN    (* message addressed to this frame *)
                    (* calculate display coordinates *)
                    x := M.x + F.X; y := M.y + F.Y; w := F.W; h := F.H;
                    IF M IS Display.DisplayMsg THEN
                        WITH M: Display.DisplayMsg  DO
                            IF (M.id = Display.frame) OR (M.F = NIL) THEN
                                Gadgets.MakeMask(F, x, y, M.dlink, R);
                                RestoreFrame(F, R, x, y, w, h)
                            ELSIF M.id = Display.area THEN
                                Gadgets.MakeMask(F, x, y, M.dlink, R);
                                Display3.AdjustMask(R, x + M.u, y + h - 1 + M.v, M.w, M.h);
                                RestoreFrame(F, R, x, y, w, h)
                            END
                        END
                    ELSE Gadgets.framehandle(F, M)
                    END
                END
            END
        ELSE
            Gadgets.framehandle(F, M)
        END;
        IF (M IS Gadgets.UpdateMsg) THEN
            Update(F);
        END;
    END; END
END FrameHandler;

PROCEDURE InitFrame*(F: Frame); (* provided for later type extensions *)
```

```
BEGIN
    F.W := 0; F.H := 12; F.col := 1; F.handle := FrameHandler; F.Caption:="<UL>";
    F.xsize:=25;
    F.obj:=b;
    F.al:=NIL;
END InitFrame;

PROCEDURE NewFrame*;
VAR F: Frame;
BEGIN NEW(F); InitFrame(F); Objects.NewObj := F; F.b:=b.val;
END NewFrame;

PROCEDURE NewInvisible*(cap : ARRAY OF CHAR): Objects.Object;
VAR F: Frame;
BEGIN
    NEW(F); InitFrame(F);
    COPY(cap, F.Caption);
    F.xsize:=Strings.Length(cap)*7;
    RETURN (F);
END NewInvisible;

PROCEDURE Flip*;
BEGIN
    b.val:=~b.val;
    BasicGadgets.SetValue(b);
END Flip;

BEGIN
    BasicGadgets.NewBoolean;
    b:=Objects.NewObj(BasicGadgets.Boolean);
    b.val:=TRUE;
END Invisible.
```

B.11 LinkGadgets

```
MODULE LinkGadgets;

IMPORT
    Objects, Oberon, Texts, TextGadgets, TextGadgets0, Out, Gadgets,
    TextFrames, Loader, Panels, TextFields, BasicGadgets, Files, Store,
    Strings, Display, Tags, Display3;

CONST
    LinkGadgetTextColor*=Display3.blue;

TYPE
    LinkGadget* = POINTER TO LinkGadgetDesc;
    LinkGadgetDesc* = RECORD (TextGadgets.FrameDesc)
```

```
        al* : Tags.AttrList;
    END;


PROCEDURE LinkAttr(L : LinkGadget; VAR M : Objects.AttrMsg);
(*  AttributeMessage for LinkGadgets *)
BEGIN
    IF ((M.id = Objects.get) & (M.name = "Gen"))
    THEN
        M.class := Objects.String;
        COPY("LinkGadgets.NewLinkGadget", M.s);
        M.res := 0
    ELSE Tags.HandleAttr(L.al, M);
    END;
END LinkAttr;

PROCEDURE FindPanel*(VAR p : Panels.Panel);
VAR obj : Objects.Object;
        name : ARRAY 200 OF CHAR;
BEGIN
    p:=NIL;
    obj:=Oberon.Par.obj;
    WHILE obj#NIL DO
        IF obj IS Panels.Panel THEN
            Gadgets.GetObjName(obj,name);
            IF name="thepanel" THEN p:=obj(Panels.Panel); END;
        END;
        obj:=obj.dlink;
    END;
END FindPanel;

PROCEDURE StoreLinkGadget*(VAR o :Files.Rider; f : LinkGadget);
VAR a : Tags.Attr;
BEGIN
    Strings.WriteStrToFile(o, "<A ");
    Tags.GetAttr(f.al,"name",a);
    IF a.s[0]#0X THEN
        Strings.WriteStrToFile(o,"NAME=");
        Strings.WriteStrToFile(o,a.s);
        Files.Write(o,' ');
    END;
    Tags.GetAttr(f.al,"href",a);
    IF a.s#0X THEN
        Strings.WriteStrToFile(o,"HREF =");
        Strings.WriteStrToFile(o,a.s);
        Files.Write(o,' ');
    END;
    Strings.WriteStrToFile(o,"> ");
    Store.Store(o, f.text);
    Strings.WriteStrToFile(o,"</A>");
END StoreLinkGadget;
```

```
PROCEDURE Handle*  (F: Objects.Object; VAR M: Objects.ObjMsg);
VAR
    i, w, h : INTEGER;
    n : ARRAY 64 OF CHAR;
    t : Texts.Text;
    a : Tags.Attr;
    p : Panels.Panel;
BEGIN
    IF ((M IS Oberon.InputMsg) &
            (Gadgets.InActiveArea(F(Gadgets.Frame),M(Oberon.InputMsg))))
    THEN (* text inside a linkgadget has color blue *)
        M(Oberon.InputMsg).col:=LinkGadgetTextColor;
    END;

    IF ((M IS Display.ConsumeMsg) & (M(Display.ConsumeMsg).id=Display.drop)
                & (M(Display.FrameMsg).F=F)) THEN
        Tags.GetAttr(F(LinkGadget).al,"href", a);
        COPY(M(Display.ConsumeMsg).obj(BasicGadgets.String).val, a.s);
        Tags.SetAttr(F(LinkGadget).al, a);
    ELSIF M IS Objects.AttrMsg THEN LinkAttr(F(LinkGadget), M(Objects.AttrMsg))
    ELSIF ((M IS Objects.FileMsg) & (M(Objects.FileMsg).id=Objects.store))
        THEN StoreLinkGadget(M(Objects.FileMsg).R, F(LinkGadget));
    ELSIF M IS Oberon.InputMsg THEN
        IF 1 IN M (Oberon.InputMsg).keys
        THEN
            Oberon.Par.obj:=M.dlink;
            FindPanel(p);
            Tags.GetAttr(F(LinkGadget).al,"href",a);
            Loader.Load (a.s, p);
        ELSE
            TextGadgets.FrameHandler (F,M);
        END;
    ELSIF M IS Oberon.ConsumeMsg THEN
        Out.String("!"); TextGadgets.FrameHandler (F,M);
    ELSE TextGadgets.FrameHandler (F,M);
    END;
END Handle;

PROCEDURE InitLinkGadget* (f : LinkGadget);
VAR
    cmesg : Objects.CopyMsg;
BEGIN
   (* There does not exist a Initialise function for TextNotes.
       To initialise a LinkGadget, which is a subclass of TextNote, first
       a new TextNote is created (this function puts a new, initialised
       TextNote in Objects.NewObj). Then all the variables contained
       in the TextNote are copied to the LinkGadget.
       Finally, the LinkGadget handler is installed *)

    TextGadgets.NewNote;

    cmesg.id:=Objects.deep; cmesg.obj:=f;
```

```
    TextGadgets.CopyFrame(cmesg, Objects.NewObj(TextGadgets.Frame),f );

    f.handle:= Handle;
    f.H:=20;

    f.al:=NIL;
    Tags.AddString(f.al, "href","");
    Tags.AddString(f.al, "name","");
    f.state0:={1,2,3,5,6,7,8,9,10,11,12,15};
END InitLinkGadget;

PROCEDURE NewLinkGadget*;
VAR f : LinkGadget;
BEGIN
    NEW(f);
    InitLinkGadget(f);
    Objects.NewObj := f;
END NewLinkGadget;

END LinkGadgets.
```

B.12 ListGadgets

```
MODULE ListGadgets;

IMPORT
    Objects, Oberon, Texts, TextGadgets, Out, Gadgets, Display,
    Panels, TextFields, Figures, Display3, Effects, TextUtil, Tags;

TYPE
    brFrame* = POINTER TO brFrameDesc;
    brFrameDesc* = RECORD (Gadgets.FrameDesc)
    END;

    dtFrame* = POINTER TO dtFrameDesc;
    dtFrameDesc* = RECORD (Panels.PanelDesc)
        dt*, dd* : TextGadgets.Frame;
    END;


PROCEDURE RestoreBr(F: brFrame; M: Display3.Mask; x, y, w, h: INTEGER);
BEGIN
    Display3.ReplConst(M, -10, x, y, w, h, Display.replace);

    Display3.ReplConst(M, Display3.textC, x+5, y+3, 10,1, Display.replace);
    Display3.ReplConst(M, Display3.textC, x+15, y+3, 1, 7,  Display.replace);
    Display3.ReplConst(M, Display3.textC, x+6, y+2, 1, 3, Display.replace);
    Display3.ReplConst(M, Display3.textC, x+7, y+1, 1, 5, Display.replace);
    Display3.ReplConst(M, Display3.textC, x+7, y+0, 1, 7, Display.replace);
```

```
    IF Gadgets.selected IN F.state THEN
        Display3.FillPattern(M, Display3.white, Effects.selectpat, x, y, x, y,
                             w, h, Display.paint)
    END
END RestoreBr;

PROCEDURE BrHandler*(F: Objects.Object; VAR M: Objects.ObjMsg);
VAR x, y, w, h: INTEGER; R: Display3.Mask;
BEGIN
    WITH F: brFrame DO
        IF M IS Display.FrameMsg THEN
            WITH M: Display.FrameMsg DO
                IF (M.F = NIL) OR (M.F = F) THEN    (* message addressed to this frame *)
                    (* calculate display coordinates *)
                    x := M.x + F.X; y := M.y + F.Y; w := F.W; h := F.H;
                    IF M IS Display.DisplayMsg THEN
                        WITH M: Display.DisplayMsg  DO
                            IF (M.id = Display.frame) OR (M.F = NIL) THEN
                                Gadgets.MakeMask(F, x, y, M.dlink, R);
                                RestoreBr(F, R, x, y, w, h)
                            ELSIF M.id = Display.area THEN
                                Gadgets.MakeMask(F, x, y, M.dlink, R);
                                Display3.AdjustMask(R, x + M.u, y + h - 1 + M.v, M.w, M.h);
                                RestoreBr(F, R, x, y, w, h)
                            END
                        END
                    ELSE Gadgets.framehandle(F, M)
                    END
                END
            END
        ELSE
            Gadgets.framehandle(F, M)
        END
    END
END BrHandler;

PROCEDURE InitBr*(F: brFrame); (* provided for later type extensions *)
BEGIN F.W := 500; F.H := 10; F.col := 1; F.handle := BrHandler;
END InitBr;

PROCEDURE NewDT*;
VAR
    p : dtFrame;
    dt, dd : TextGadgets.Frame;
    cap1, cap2 : TextFields.Caption;
BEGIN
    NEW(p); Panels.InitPanel(p);

    TextFields.NewCaption; cap1:=Objects.NewObj(TextFields.Caption);
    TextFields.NewCaption; cap2:=Objects.NewObj(TextFields.Caption);
    TextGadgets.NewNote; dt:=Objects.NewObj(TextGadgets.Frame);
```

```
    TextGadgets.New; dd:=Objects.NewObj(TextGadgets.Frame);
    Texts.New; cap1.text:=Objects.NewObj(Texts.Text);
    Texts.New; cap2.text:=Objects.NewObj(Texts.Text);
    TextUtil.AppendString(cap1.text,"Term:");
    TextUtil.AppendString(cap2.text,"Def.:");
    TextFields.CalcCaptionSize(cap1, FALSE);
    TextFields.CalcCaptionSize(cap2, FALSE);

    Panels.InsertChild(p, cap1, 10,-20);
    Panels.InsertChild(p, cap2, 10,-60);

    dt.W:=380; dt.H:=30;
    Panels.InsertChild(p, dt, 60, -40);
    Gadgets.NameObj(dt,"dt");

    dd.W:=380; dd.H:=70;
    Panels.InsertChild(p, dd, 60, -120);
    Gadgets.NameObj(dd,"dd");

    p.W:=480; p.H:=130;
    p.dt:=dt; p.dd:=dd;

    Objects.NewObj:=p;
END NewDT;


PROCEDURE NewBr*;
VAR F : brFrame;
BEGIN
    NEW(F); InitBr(F); Objects.NewObj:=F;
END NewBr;

PROCEDURE NewListItem*;
VAR c : Figures.Circle;
        state : SET;
BEGIN
    state:={Figures.filled};
    c:=Figures.NewC(15,0,0,5,10,state);
    Objects.NewObj:=c;
END NewListItem;

PROCEDURE NewHR*;
VAR l    : Figures.PolyLine;
        x, y : ARRAY 3 OF INTEGER;
BEGIN
    x[0]:=0; x[1]:=400;
    y[0]:=0; y[1]:=0;
    l:=Figures.NewPolyLine(15,x,y,2,400,{});
    Objects.NewObj:=l;
END NewHR;
```

```
BEGIN
END ListGadgets.
```

B.13 Loader

```
MODULE Loader;

(* this module handles the loading of a document, given the 'href' *)
IMPORT
    Strings, Out, TextGadgets, ExternalRoutines, TextGadgets0, TextFrames, Texts, TextUtil,
    ExternalProgs, Parser, Objects, Gadgets, Panels, Display, Tags, MainPanel, Format;

VAR
    history : ARRAY 100 OF ARRAY 200 OF CHAR;
    historyN : INTEGER;

PROCEDURE PushText*(url : ARRAY OF CHAR);
BEGIN
    IF historyN<100 THEN
        COPY(url, history[historyN]);
        historyN:=historyN+1;
    END;
END PushText;

PROCEDURE ShowHistory;
VAR i : INTEGER;
BEGIN
    Out.String("------ history -----------"); Out.Ln;
    FOR i:=1 TO historyN DO
        Out.String(history[i]);
        Out.Ln;
    END;
    Out.String("-------------------------------");
    Out.Ln;
END ShowHistory;

PROCEDURE PopText*(VAR url : ARRAY OF CHAR);
BEGIN
    IF historyN>1 THEN
        COPY(history[historyN-1],url);
        historyN:=historyN-1;
    END;
END PopText;

PROCEDURE GetProtocol*(href : ARRAY OF CHAR; VAR protocol : ARRAY OF CHAR);
(*  GetProtocol tries to fiend the protocol, given a href. The protocol is everything    *)
(*  before ':', usual http, ftp or something like that.    *)
```

```
VAR i  : INTEGER;
BEGIN
   i:=0;
   WHILE ((href[i]#0X) & (href[i]#':')) DO
        protocol[i]:=href[i];
        i:=i+1;
    END;
    IF href[i]=':' THEN
        protocol[i]:=0X
    ELSE
        protocol[0]:=0X;
    END;
END GetProtocol;

PROCEDURE GetExtension(href : ARRAY OF CHAR; VAR extension: ARRAY OF CHAR);
(* GetExtension tries to fiend the extension of the file.                    *)
(* Pre: - the href string is zero-terminated  (the last char is a 0X).       *)
(*      - extension can contain relatively large strings (at most size(href) is needed  *)
VAR i,j  : INTEGER;
BEGIN
   i:=0; j:=0;
   WHILE (href[i]# 0X) DO
      IF ((href[i]='.') OR (href[i]='/')) THEN
          j:=i+1;
      ELSE
         extension[i-j]:=href[i];
      END;
       i:=i+1;
   END;
   extension[i-j] := 0X;
   IF i-j=0 THEN COPY("html",extension);
  END;
END GetExtension;

PROCEDURE GetFile*(href : ARRAY OF CHAR; VAR fileName : ARRAY OF CHAR);
VAR i, j : INTEGER;
BEGIN
    i:=0; j:=0;
    WHILE j#3 DO
        IF href[i]='/' THEN j:=j+1; END;
        i:=i+1;
    END;
    i:=i-1; j:=i;
    WHILE href[i]#0X DO
        fileName[i-j]:=href[i];
        i:=i+1;
    END;
    fileName[i-j]:=0X;
END GetFile;

PROCEDURE GetLabelName(href : ARRAY OF CHAR; VAR labelName : ARRAY OF CHAR);
(* A href may contain a label, e.g. href="tile#position". This procedure will
```

```
   return the characters after (so WITHOUT) the '#' in variable labelName *)
VAR i,j  : INTEGER;
BEGIN
    i:=0;
    WHILE ((href[i]#0X) & (href[i]#'#')) DO i:=i+1; END;
    IF href[i]=0X THEN
       labelName[0]:=0X
    ELSE
         i:=i+1; j:=0;
         WHILE href[i]#0X DO
             labelName[j]:=href[i];
             i:=i+1; j:=j+1;
         END;
         labelName[j]:=0X
    END;
END GetLabelName;

PROCEDURE GetBaseOfURL1(href : ARRAY OF CHAR; VAR base : ARRAY OF CHAR);
(* return from 'href' the protocol, the machine-adres, and the path,
    (in other words: remove filename + label)
    e.g. if href="http://www.some.where/user/csg/csg170/home.html#start",
           then "http://www.some.where/user/csg/csg170/" wil be returned
*)
VAR i,lastSlash : INTEGER;
BEGIN
    i:=0; lastSlash:=0;
    WHILE href[i]#0X DO
        base[i]:=href[i];
        IF href[i]='/' THEN lastSlash:=i; END;
        i:=i+1;
    END;
    base[lastSlash+1]:=0X;
END GetBaseOfURL1;

PROCEDURE GetBaseOfURL2(href : ARRAY OF CHAR; VAR base : ARRAY OF CHAR);
(* return from 'href' the protocol and the machine-adres,
    (in other words: remove path+ filename + label)
    e.g. if href="http://www.some.where/user/csg/csg170/home.html#start",
           then "http://www.some.where"  wil be returned
*)
VAR i, slashCount : INTEGER;
BEGIN
    (* The text up to the 3rd '/' is returned *)
    i:=0; slashCount:=0;
    WHILE ((href[i]#0X) & (slashCount#3)) DO
        base[i]:=href[i];
        IF href[i]='/' THEN slashCount:=slashCount+1; END;
        i:=i+1;
    END;
    base[i-1]:=0X;
END GetBaseOfURL2;
```

```
PROCEDURE MoveToAnchorWithName(name : ARRAY OF CHAR; theframe : TextGadgets.Frame);
(* This procedure is used when a link is followed to a label (anchor with NAME attribute),
    this procedure will scroll the textgadget so that the wanted label is in the upper left
    corner. *)
VAR
    f : Texts.Finder;
    found : BOOLEAN;
    obj : Objects.Object;
    oldPos : LONGINT;
    a : Tags.Attr;
    am : Objects.AttrMsg;
BEGIN
    Out.String("Moving to label: "); Out.String(name); Out.Ln;
    Texts.OpenFinder(f, theframe.text,0);
    found:=FALSE;
    WHILE (~found & ~f.eot) DO
        (* There is something funny with Finders. After using Texts.FindObj(f,obj),
            the position f.pos does NOT point to the position of obj, but to the
            NEXT object in the text. Therefore, it is neccesary to keep an old version
            of this value *)
        oldPos:=f.pos;
        Texts.FindObj(f, obj);
        IF obj#NIL THEN
            am.id:=Objects.get;
            am.name:="name";
            am.res:=-1;
            obj.handle(obj,am);
            found:= Strings.CompareIgnoreUpcase(name, am.s);
        END;
    END;
    IF found THEN
        TextGadgets0.ScrollTo(theframe,oldPos);
    ELSE
        Out.String("Label "); Out.String(name); Out.String(" not found!"); Out.Ln;
    END;
END MoveToAnchorWithName;

PROCEDURE ParseAndShow*(t : Texts.Text; href : ARRAY OF CHAR; thepanel : Panels.Panel);
VAR
    labelName : ARRAY 200 OF CHAR;
    theframe   : TextGadgets.Frame;
BEGIN
    (* to be able to go 'back', save the current URL *)
    PushText(href);

    (* show the URL inside the main panel *)
    MainPanel.SetURL(thepanel, href);

    (* parse the text *)
    Parser.Parse(t, thepanel);
```

```
    (* find the text frame in which the text is shown *)
    theframe:=MainPanel.GetTextGadget(thepanel);

    (* reformat the text *)
    Format.FormatText(t, theframe.W-50);

    (* let the text frame show the text *)
    theframe.text:=t;

    (* reset the text frame, the changes will now be shown *)
    MainPanel.Reset(theframe);

    (* move to a label *)
    GetLabelName(href,labelName);
    IF labelName[0]#0X THEN
        MoveToAnchorWithName(labelName, theframe);
    ELSE
        TextGadgets0.Locate(theframe, 0);
    END;
END ParseAndShow;

PROCEDURE LoadHtml*(href : ARRAY OF CHAR; thepanel : Panels.Panel);
VAR
    t : Texts.Text;
BEGIN
    ExternalRoutines.Load(href);
    t:=TextFrames.Text("html_file");

    ParseAndShow(t, href, thepanel);
END LoadHtml;

PROCEDURE Load*(href : ARRAY OF CHAR; thepanel : Panels.Panel);
VAR prot, ext , url, base, label, fileName : ARRAY 200 OF CHAR;
       t : Texts.Text;
BEGIN
     (* is this just a link to another point in the same document? *)
    IF href[0]="#" THEN
        GetLabelName(href,label);
        MoveToAnchorWithName(label, MainPanel.GetTextGadget(thepanel));
    ELSE
    (* determine the protocol used. *)
    GetProtocol(href,prot);
    IF prot[0]=0X THEN
        MainPanel.GetURL(thepanel, url);
        IF href[0]='/' THEN
            GetBaseOfURL2(url,base)
        ELSE
            GetBaseOfURL1(url,base);
        END;
        Strings.Concat(base,href);
        COPY(base,href);
        GetProtocol(href,prot);
```

```
    END;

    IF Strings.CompareIgnoreUpcase("http",prot) THEN
        (* http should be used to load file, determine extension of file *)
        GetExtension(href,ext);
        IF (Strings.CompareIgnoreUpcase("html", ext) OR
                    Strings.CompareIgnoreUpcase("shtml", ext)) THEN
              (* load a html file *)
              LoadHtml(href, thepanel);
        ELSE (* not a html file, use external viewer *)
             IF ExternalProgs.IsViewerAssociated(ext) THEN
                   ExternalRoutines.Load(href);
                   ExternalProgs.ShowFile(ext,"html_file");
             ELSE
                  (* not known what to do with file, assume that is html,
                      but generate warning *)
                 Out.String("Warning: no viewer associated with extension : .");
                 Out.String(ext); Out.Ln;
                 Out.String("File is assumed to be a HTML file..."); Out.Ln;
                 LoadHtml(href,thepanel);
             END;
        END;
    ELSIF Strings.CompareIgnoreUpcase("file",prot) THEN
        (* get full path + filename *)
        GetFile(href, fileName);
        Out.String("Loading: "); Out.String(fileName); Out.Ln;
         GetExtension(href,ext);
        IF (Strings.CompareIgnoreUpcase("html", ext) OR
                    Strings.CompareIgnoreUpcase("shtml", ext)) THEN
            t:=TextFrames.Text(fileName);
            ParseAndShow(t, href, thepanel);
        ELSE
                  ExternalProgs.ShowFile(ext, fileName);
        END;
    ELSE
        Out.String("Sorry, the protocol "); Out.String(prot);
        Out.String(" is not supported."); Out.Ln;
    END; END;
END Load;


BEGIN
    historyN:=1;
END Loader.
```

B.14 MainPanel

```
MODULE MainPanel;

IMPORT Objects, Panels, BasicGadgets, Gadgets, TextGadgets, TextGadgetsO, Display, TextFields;
```

```
PROCEDURE GetTextGadget*(panel : Panels.Panel): TextGadgets.Frame;
VAR obj : Objects.Object;
BEGIN
   obj:=Gadgets.FindObj(panel,"thetext");
   RETURN obj(TextGadgets.Frame);
END GetTextGadget;

PROCEDURE GetTitle*(panel : Panels.Panel; VAR a: ARRAY OF CHAR);
VAR obj : Objects.Object;
BEGIN
    obj := Gadgets.FindObj(panel,"title");
    COPY(obj(TextFields.TextField).val,a);
END GetTitle;

PROCEDURE SetTitle*(panel: Panels.Panel;  a: ARRAY OF CHAR);
VAR obj : Objects.Object;
BEGIN
    obj := Gadgets.FindObj(panel,"title");
    COPY(a,obj(TextFields.TextField).obj(BasicGadgets.String).val);
    BasicGadgets.SetValue(obj(TextFields.TextField).obj(BasicGadgets.String));
END SetTitle;

PROCEDURE GetURL*(panel: Panels.Panel;   VAR a: ARRAY OF CHAR);
VAR obj : Objects.Object;
BEGIN
    obj := Gadgets.FindObj(panel,"url");
    COPY(obj(TextFields.TextField).val,a);

    COPY(a,obj(TextFields.TextField).obj(BasicGadgets.String).val);
    BasicGadgets.SetValue(obj(TextFields.TextField).obj(BasicGadgets.String));
END GetURL;

PROCEDURE SetURL*(panel: Panels.Panel;  a: ARRAY OF CHAR);
VAR obj : Objects.Object;
BEGIN
    obj := Gadgets.FindObj(panel,"url");
(*  COPY(a, obj(TextFields.TextField).val); *)

    COPY(a,obj(TextFields.TextField).obj(BasicGadgets.String).val);
    BasicGadgets.SetValue(obj(TextFields.TextField).obj(BasicGadgets.String));
END SetURL;

PROCEDURE Reset*(theframe : TextGadgets.Frame);
VAR msg : Display.DisplayMsg;
BEGIN
    theframe.mask:=NIL;
    TextGadgets0.FormatFrame(theframe);
    TextGadgets0.Locate(theframe,0);
    msg.F:=NIL; msg.res:=-1;
    msg.id:=Display.frame;
    Display.Broadcast(msg);
```

```
END Reset;

END MainPanel.
```

B.15 Parser

```
MODULE Parser;

IMPORT
    Texts, Objects, Out, Oberon, TextGadgets, TextGadgets0, System,
    Display, Gadgets, Panels, BasicGadgets, Display3,
    Strings, TextUtil, HeaderFont, Tags, MainPanel,
    SpecialChars, Invisible, Format;

CONST
    LF = 0AX;    (* Linefeed *)
    CR = 0DX;    (* Carriage Return *)
    HT = 9X;       (* Horizontal Tab, value 9X is taken from Module EdiT, the normal
                          ASCII value for a hor. tab is 0BX..... *)

    MSL = 300; (* Maximum string length *)

(* to do: correct parsing of " and ' within the function ParseValue *)

PROCEDURE Parse*(t: Texts.Text; panel : Panels.Panel);
VAR
    pos              : LONGINT;   (* the text is parsed up to pos *)
    r                : Texts.Reader;
    ch             : CHAR;
    tag              : ARRAY MSL OF CHAR;
    begTag     : LONGINT;
    result       : BOOLEAN;
    o              : Objects.Object;
    theTag      : Tags.Tag;

    PROCEDURE IsSpace(ch : CHAR): BOOLEAN;
    BEGIN RETURN ((ch=' ') OR (ch=LF) OR (ch=CR) OR (ch=HT)); END IsSpace;

    PROCEDURE SkipSpace(t: Texts.Text; VAR pos : LONGINT);
    (* skip all spaces, linefeeds, comments, etc. *)
    VAR r: Texts.Reader; ch: CHAR;
    BEGIN
        Texts.OpenReader(r,t,pos); Texts.Read(r, ch);
        WHILE ((~r.eot) & ((ch=' ') OR (ch=LF) OR (ch=CR) OR (ch=HT)) ) DO
            pos:=pos+1; Texts.Read(r,ch);
        END;
    END SkipSpace;

    PROCEDURE DeleteTag(t: Texts.Text; VAR pos : LONGINT; tag : Tags.Tag);
    BEGIN
```

```
    Texts.Delete(t,tag.beg,tag.end);
    pos:=tag.beg;
END DeleteTag;

PROCEDURE ParseVariable(t : Texts.Text; VAR pos : LONGINT;
            VAR varName : ARRAY OF CHAR; VAR result : BOOLEAN);
(* try to find a variable name (for instance: 'href' or  'name' inside
    <A ......>  in text T, starting at pos.
    All spaces, linefeeds etc. must be skipped. If the next char is in
    ['a'..'z','A'..'Z'], then a variable name is  found.
    The variable name is then copied into varName. *)
VAR r : Texts.Reader;
        ch : CHAR;
       i : INTEGER;
BEGIN
    SkipSpace(t,pos);
    Texts.OpenReader(r,t,pos);
    Texts.Read(r,ch);
    i:=0;
    IF ( ~r.eot & (((ch>='a') & (ch<='z')) OR ((ch>='A') & (ch<='Z'))
                OR (ch='/')) ) THEN
        result:=TRUE;
        varName[i]:=ch;
        pos:=pos+1; i:=i+1;
        Texts.Read(r,ch);
        WHILE (~r.eot & ( ((ch>='a') & (ch<='z')) OR ((ch>='A') & (ch<='Z')) OR
                ((ch>='0') & (ch<='9')) )) DO
            varName[i]:=ch;
            pos:=pos+1; i:=i+1;
            Texts.Read(r,ch)
        END;
        (* put a zero-char at the end of the varName *)
        varName[i]:=0X;
    ELSE
        result:=FALSE;
    END;
END ParseVariable;

PROCEDURE ParseValue(t : Texts.Text; VAR pos : LONGINT; VAR value : ARRAY OF CHAR);
(*  with value is meant: the stuff following e.g. '<A href'. This
    is first a '=' sign, then the string. The string should be put in quotes,
    but this is not always done. The string is put in the variable 'value',
    always WITHOUT quotes. *)
VAR
    usesquotes : BOOLEAN;
    i : LONGINT;
    r : Texts.Reader;
    ch : CHAR;
BEGIN
    SkipSpace(t,pos);
    Texts.OpenReader(r,t,pos);
    Texts.Read(r,ch); pos:=pos +1;
```

```
    (* now, there should ch should be '='. If this is not so, skip the chars
        until the '=' is found *)
    WHILE ch#'=' DO Texts.Read(r,ch); pos:=pos+1; END;
    SkipSpace(t,pos);

    Texts.OpenReader(r,t,pos);
    Texts.Read(r,ch); pos:=pos+1;
    usesquotes := ch ='"';
    i:=0;

    IF usesquotes THEN
    Texts.Read(r,ch); pos:=pos+1;
        WHILE ((ch#'"') & (ch#'>')) DO
            value[i]:=ch; i:=i+1;
            Texts.Read(r,ch); pos:=pos+1;
        END;
        IF ch='>' THEN pos:=pos-1; END;
        value[i]:=0X;
    ELSE
        value[i]:=ch; i:=i+1;
        Texts.Read(r,ch); pos:=pos+1;
        WHILE (~IsSpace(ch) & (ch#'>')) DO
            value[i]:=ch; i:=i+1;
            Texts.Read(r,ch); pos:=pos+1;
        END;
        pos:=pos-1;
        value[i]:=0X;
    END;
END ParseValue;

PROCEDURE ParseTag(t: Texts.Text; VAR pos : LONGINT; begTag: LONGINT;
                tagName : ARRAY OF CHAR; VAR tag : Tags.Tag);
VAR
    r                : Texts.Reader;
    varName     : ARRAY 200 OF CHAR;
    val             : ARRAY 200 OF CHAR;
BEGIN
    Tags.InitTag(tag); tag.beg:=begTag;
    COPY(tagName, tag.tagName);
    SkipSpace(t,pos); Texts.OpenReader(r,t,pos); Texts.Read(r,ch);
    result:=TRUE;
    WHILE (~r.eot & (ch#'>') & result) DO
        ParseVariable(t,pos,varName,result);
        IF result THEN
            SkipSpace(t,pos); Texts.OpenReader(r,t,pos); Texts.Read(r,ch);
            IF ch="=" THEN
                ParseValue(t,pos, val);
                Tags.AddString(tag.vars, varName, val);
            ELSE
                Tags.AddBoolean(tag.vars, varName, TRUE);
            END;
            Texts.OpenReader(r,t,pos);
```

```
            Texts.Read(r,ch);
        END;
    END;
    SkipSpace(t,pos); Texts.OpenReader(r,t,pos); Texts.Read(r,ch);
    IF (~result & (ch#'>') & ~r.eot) THEN
        Out.String("Warning: Tag <"); Out.String(tagName);
        Out.String("> has wrong syntax!"); Out.Ln;
        WHILE (~r.eot & (ch#'>')) DO
            Texts.Read(r,ch);
            pos:=pos+1;
        END;
        IF r.eot THEN pos:=pos-1; END;
    END;
    pos:=pos+1;
    tag.end:=pos;
END ParseTag;

PROCEDURE ParseComment(t: Texts.Text; VAR pos : LONGINT; begTag: LONGINT);
VAR
    endTag, endTag2 : LONGINT;
    endBeginTag : LONGINT;
    font       : Objects.Library;
BEGIN
    endBeginTag:=TextUtil.SearchStr(t,pos,'--');
    IF endBeginTag=pos+3 THEN pos:=pos+2; END;
    Texts.Delete(t, begTag, pos+1); pos:=begTag;
    endTag:=TextUtil.SearchStr(t,pos,'>');
    endTag2:=TextUtil.SearchStr(t,pos,'-->');

    font:=Objects.ThisLibrary("Syntax10.Scn.Fnt");
    Texts.ChangeLooks(t, begTag, endTag, {1}, font, 100,0);

    IF endTag=endTag2 THEN
        (* end tag is --> *)
        Texts.Delete(t,endTag-3, endTag);
        pos:=endTag-3;
    ELSE
        Texts.Delete(t,endTag-1, endTag);
        pos:=endTag-1;
    END;
END ParseComment;

PROCEDURE ParseFont(t: Texts.Text; VAR pos : LONGINT; tag : Tags.Tag);
(* Font/header tags are parsed in the following way:
    - Find the end tag.
    - Copy everything inside the <B> .. </B> in a new text
    - Change the font of the new text
    - recursively parse the new text
    - replace the part of the old text for the new text.
*)
VAR
    wr                  : Texts.Writer;
```

```
    tmpText              : Texts.Text;
    endTag,
    tagSize              : LONGINT;
    s                       : ARRAY 200 OF CHAR;
    newFont             : Objects.Library;
    o                        : Objects.Object;
BEGIN
    HeaderFont.GetFont(tag.tagName, newFont);
    IF newFont=NIL THEN Out.String("Error: could not get font"); Out.Ln; END;

    s:="</"; Strings.Concat(s, tag.tagName); Strings.Concat(s, ">");
    endTag := TextUtil.SearchStr(t,pos, s);
    tagSize:=Strings.Length(tag.tagName)+2;
    IF endTag#-1 THEN
        tmpText:=TextUtil.NewText();
(*      TextUtil.MoveText(t, tag.beg+tagSize, endTag-tagSize-1, tmpText); *)
        TextUtil.MoveText(t, pos, endTag-tagSize-1, tmpText);
        Texts.Delete(t, tag.beg, tag.beg+2*tagSize+1);

        (* tmpText now contains the text inside the <?> .. </?> *)
        Texts.ChangeLooks(tmpText,0,tmpText.len,{0},newFont,0,0);
        Parse(tmpText,panel);
        pos:=tag.beg+tmpText.len;
        Texts.OpenWriter(wr);
        Texts.Save(tmpText,0,tmpText.len,wr.buf);
        Texts.Insert(t,tag.beg,wr.buf);

        s:="<"; Strings.Concat(s, tag.tagName); Strings.Concat(s, ">");
        o:=Invisible.NewInvisible(s);
        TextUtil.InsertObjectInText(o, t, tag.beg);

        s:="</"; Strings.Concat(s, tag.tagName); Strings.Concat(s, ">");
        o:=Invisible.NewInvisible(s);
        TextUtil.InsertObjectInText(o, t, pos+1);

        pos:=pos+2;
    END;
END ParseFont;

PROCEDURE ParseTitle(t: Texts.Text; VAR pos : LONGINT; tag : Tags.Tag);
VAR
    endTag : LONGINT;
    title : ARRAY MSL OF CHAR;
    obj : Objects.Object;
BEGIN
    endTag := TextUtil.SearchStr(t,pos,"</TITLE>");
    IF endTag#-1 THEN
        Strings.GetStringFromText(t,pos,endTag-8,title);
        pos:=tag.beg;
        Texts.Delete(t, tag.beg, endTag);
        MainPanel.SetTitle(panel, title);
    ELSE
```

```
        Out.String("Warning: </TITLE> not found!"); Out.Ln;
        MainPanel.SetTitle(panel,"");
    END;
END ParseTitle;

PROCEDURE ParseImg(t: Texts.Text; VAR pos: LONGINT; tag : Tags.Tag);
VAR
    o : Objects.Object;
BEGIN
    o:=Gadgets.CreateObject("Img.NewImg");
    Tags.Merge(o, tag.vars);
    Texts.Delete(t,tag.beg,tag.end);
    pos:=tag.beg+1;
    TextUtil.InsertObjectInText(o,t,pos-1);
END ParseImg;

PROCEDURE ParseAnchor(t: Texts.Text; VAR pos : LONGINT; tag : Tags.Tag);
(* pos points to the character after '<A' ( usually is a space. ).
    The text is to be parsed up to the tag '</A>', which markes
    the end of the anchor.  tag.beg points to the '<' of the begin
    tag '</A>' *)
VAR
    endTag, startNextAnchor     : LONGINT;
    anchor                              : TextGadgets.Frame;
    width,height                        : INTEGER;
    o                                       : Objects.Object;
BEGIN
      o:=Gadgets.CreateObject("LinkGadgets.NewLinkGadget");
      anchor:=o(TextGadgets.Frame);
      Tags.Merge(o, tag.vars);

    (* find the position of  '</A> ' *)
    endTag := TextUtil.SearchStr(t,pos,"</A>");
    startNextAnchor := TextUtil.SearchStr(t,pos,"<A");

    (* there is not always a </A> for every <A>. so check if
        a) there is a next </a>      (endTag # -1 )
        b) the next <a> is after the next </a>
    *)

    IF ((endTag#-1) & ((startNextAnchor=-1) OR (endTag < startNextAnchor)) ) THEN
        Texts.Delete(t,endTag-4, endTag); (* delete end tag *)
        TextUtil.MoveText(t,pos,endTag-4, anchor(TextGadgets0.Frame).text);
        Texts.Delete(t,tag.beg, pos);
        Parse(anchor(TextGadgets0.Frame).text,panel);

        Format.FormatText(anchor(TextGadgets0.Frame).text, 500);
        Texts.ChangeLooks(anchor.text, 0, anchor.text.len,{1}, NIL, Display3.blue,0);
        pos:=tag.beg + 1;
    ELSE
       Texts.Delete(t, tag.beg,pos);
       pos:=tag.beg;
```

```
    END;

    TextUtil.CalcWidth(anchor.text, width, height);
    IF width<30 THEN width:=30; END;
    IF height<15 THEN height:=15; END;
    anchor.W:=width;
    anchor.H:=height;

    TextUtil.InsertObjectInText(anchor, t, tag.beg);
    Texts.ChangeLooks(t, tag.beg, tag.beg+1, {2}, NIL, 0, -9);
END ParseAnchor;

PROCEDURE ParseDT(t: Texts.Text; VAR pos: LONGINT; tag: Tags.Tag);
VAR
    nextDT, nextDD, nextEndDL, next, next2: LONGINT;
    o, o2 : Objects.Object;
    dt, dd : TextGadgets.Frame;
BEGIN
    Texts.Delete(t,tag.beg,pos);
    pos:=tag.beg;

    nextDT:=TextUtil.SearchStr(t,pos,"<DT>")-4;
    nextDD:=TextUtil.SearchStr(t,pos,"<DD>")-4;
    nextEndDL:=TextUtil.SearchStr(t,pos,"</DL>")-5;
    IF nextDD<0 THEN
        IF nextDT<0 THEN next:=nextEndDL;
        ELSE
            IF nextDT<nextEndDL THEN next:=nextDT
            ELSE next:=nextEndDL END;
        END
    ELSE
        IF nextDT<0 THEN next:=nextDD
        ELSE
            IF nextDT>nextDD THEN next:=nextDD
            ELSE next:=nextDT; END;
        END;
    END;

    IF nextDT<0 THEN next2:=nextEndDL
    ELSIF nextDT>nextEndDL THEN next2:=nextEndDL;
    ELSE next2:=nextDT;
    END;

    Out.String("dd: "); Out.Int(nextDD,3); Out.Ln;
    Out.String("dt: "); Out.Int(nextDT,3); Out.Ln;
    Out.String("/dl: "); Out.Int(nextEndDL,3); Out.Ln;
    Out.String("next: "); Out.Int(next,3); Out.Ln;

    o:=Gadgets.CreateObject("ListGadgets.NewDT");

    o2:=Gadgets.FindObj(o, "dd");
    dd:=o2(TextGadgets.Frame);
```

```
    o2:=Gadgets.FindObj(o, "dt");
    dt:=o2(TextGadgets.Frame);

    TextUtil.MoveText(t, pos, next, dt.text);
    IF (next=nextDD) THEN
        (* there is a <DD> associated with the DT *)
        next2:=next2-dt.text.len;
        TextUtil.MoveText(t, pos+4, next2, dd.text);
        Texts.Delete(t,pos, pos+4);
        Parse(dd.text, NIL);
    ELSE
        (* there is no <DD> belonging to this <DT> *)
        (* nothing to do! *)
    END;
    TextUtil.InsertObjectInText(o,t,pos);
    TextUtil.InsertCharInText(CHR(13),t,pos+1);
    pos:=pos+2;

    Parse(dt.text, NIL);
END ParseDT;

PROCEDURE ParseForm(t: Texts.Text; VAR pos : LONGINT; tag : Tags.Tag);
VAR
    width, height: INTEGER;
    o : Objects.Object;
    f : Display.Frame;
    endTag : LONGINT;
BEGIN
    o:= Gadgets.CreateObject("FormGadgets.NewFormGadget");
    Tags.Merge(o,tag.vars);
    Texts.Delete(t,tag.beg, tag.end);
    pos:=tag.beg;

    endTag:=TextUtil.SearchStr(t, pos, "</FORM>");
    Texts.Delete(t,endTag-7, endTag);
    TextUtil.MoveText(t,pos,endTag-7, o(TextGadgets0.Frame).text);
    Parse(o(TextGadgets0.Frame).text,panel);

    TextUtil.CalcWidth(o(TextGadgets0.Frame).text, width, height);
    IF width<30 THEN width:=30; END;
    IF height<15 THEN height:=15; END;
    o(Display.Frame).W:=width;
    o(Display.Frame).H:=height;

    TextUtil.InsertObjectInText(o, t, tag.beg);

    pos:=tag.beg + 1;
END ParseForm;

PROCEDURE ParseInput(t: Texts.Text; VAR pos : LONGINT; tag : Tags.Tag);
VAR
```

```
        o : Objects.Object;
        am : Objects.AttrMsg;
    BEGIN
        am.name:="type";
        am.id:=Objects.get;
        Tags.HandleAttr(tag.vars, am);
        IF Strings.CompareIgnoreUpcase(am.s,"submit") THEN
            o:=Gadgets.CreateObject("FormGadgets.NewButton");
        ELSE
            o:= Gadgets.CreateObject("FormGadgets.NewTextField");
        END;
        Tags.Merge(o,tag.vars);
        Texts.Delete(t,tag.beg, tag.end);
        TextUtil.InsertObjectInText(o, t, tag.beg);
        pos:= tag.beg+1;
    END ParseInput;

    PROCEDURE ParseInvisible(t : Texts.Text; VAR pos: LONGINT; tag : Tags.Tag);
    VAR s : ARRAY 100 OF CHAR;
    BEGIN
        DeleteTag(t, pos, tag);
        s:="<"; Strings.Concat(s, tag.tagName); Strings.Concat(s, ">");
        o:=Invisible.NewInvisible(s);
        Tags.Merge(o, tag.vars);
        TextUtil.InsertObjectInText(o, t, tag.beg);
        pos:=tag.beg+1;
    END ParseInvisible;

(* procedure Parse *)
BEGIN
    Texts.OpenReader(r,t,0);
    Texts.Read(r,ch);
    pos:=0;
    WHILE ~r.eot DO
        WHILE (~r.eot & (ch#'<') & (ch#'&')) DO
            pos:=pos+1;
            Texts.Read(r,ch);
        END;
        IF (~r.eot & (ch='&')) THEN
            begTag:=pos;
            WHILE ((ch#';') & (ch#' ') & (pos-begTag-1<MSL)) DO
                pos:=pos+1; Texts.Read(r,ch);
                tag[pos-begTag-1]:=ch;
            END;
            tag[pos-begTag-1]:=0X;
            SpecialChars.GetSpecialChar(tag, ch);
            TextUtil.InsertCharInText(ch,t,begTag);
            Texts.Delete(t, begTag+1, pos+2);
            pos:=begTag+1; Texts.OpenReader(r, t, pos); Texts.Read(r,ch );
        ELSIF (~r.eot & (ch='<' ))THEN
            begTag:=pos;
            pos:=pos+1;
```

```
ParseVariable(t, pos, tag, result);
IF result THEN
    ParseTag(t, pos, begTag, tag, theTag);
    IF Strings.CompareIgnoreUpcase(tag,"A") THEN
        ParseAnchor(t, pos, theTag);
    ELSIF Strings.CompareIgnoreUpcase(tag,"FORM") THEN
        ParseForm(t, pos, theTag);
    ELSIF Strings.CompareIgnoreUpcase(tag,"INPUT") THEN
        ParseInput(t, pos, theTag);
    ELSIF Strings.CompareIgnoreUpcase(tag,"IMG") THEN
        ParseImg(t, pos, theTag);
    ELSIF (Strings.CompareIgnoreUpcase(tag,"BODY")  OR
            Strings.CompareIgnoreUpcase(tag,"/BODY") OR
            Strings.CompareIgnoreUpcase(tag,"HEAD") OR
            Strings.CompareIgnoreUpcase(tag,"/HEAD") OR
            Strings.CompareIgnoreUpcase(tag,"/P") OR
            Strings.CompareIgnoreUpcase(tag,"HTML") OR
            Strings.CompareIgnoreUpcase(tag,"/HTML"))   THEN
        DeleteTag(t,pos, theTag);
    ELSIF Strings.CompareIgnoreUpcase(tag,"DT") THEN
        ParseDT(t,pos,theTag);
    ELSIF (Strings.CompareIgnoreUpcase(tag,"P")  OR
            Strings.CompareIgnoreUpcase(tag,"UL") OR
            Strings.CompareIgnoreUpcase(tag,"/UL") OR
            Strings.CompareIgnoreUpcase(tag,"DL") OR
            Strings.CompareIgnoreUpcase(tag,"/DL"))   THEN
        ParseInvisible(t,pos, theTag);
    ELSIF Strings.CompareIgnoreUpcase(tag,"BR") THEN
        DeleteTag(t, pos, theTag);
        o:= Gadgets.CreateViewModel("ListGadgets.NewBr","");
        TextUtil.InsertObjectInText(o, t, pos);
        pos:=pos+1;
    ELSIF Strings.CompareIgnoreUpcase(tag,"LI") THEN
        DeleteTag(t,pos, theTag);
        o:= Gadgets.CreateViewModel("ListGadgets.NewListItem","");
        TextUtil.InsertObjectInText(o, t, begTag);
        pos:=begTag+1;
    ELSIF Strings.CompareIgnoreUpcase(tag,"HR") THEN
        o:=Gadgets.CreateViewModel("ListGadgets.NewHR","");
        Tags.Merge(o, theTag.vars);
        DeleteTag(t, pos, theTag);
        TextUtil.InsertObjectInText(o, t, pos);
        pos:=pos+1;
    ELSIF HeaderFont.IsHeaderOrFont(tag) THEN
        ParseFont(t,pos,theTag);
    ELSIF Strings.CompareIgnoreUpcase(tag,"TITLE") THEN
        ParseTitle(t,pos,theTag);
    ELSE
        (* this tag is not supported (maybe not even HTML 2.0),
           just make an invisible gadget  *)
        ParseInvisible(t, pos, theTag);
        Out.String("Warning: <"); Out.String(tag);
```

```
                Out.String("> Unknown"); Out.Ln;
            END;

        ELSE (* probable a </.....> is found, where ... is a not supported tag *)
            Texts.OpenReader(r,t,pos);
            Texts.Read(r,ch);
            IF ch='!' THEN ParseComment(t,pos,begTag); END;
        END;
        Texts.OpenReader(r,t,pos);
        IF ~r.eot THEN
            Texts.Read(r,ch);
        END;
    END;
  END;
END Parse;


BEGIN
```

B.16 SpecialChars

```
MODULE SpecialChars;

IMPORT
    AssocList, Attributes, Out, BasicGadgets, Panels, Gadgets, Objects, Oberon,
    Display, Strings, Files;

VAR sc : AssocList.AList;

PROCEDURE GetSpecialChar*(name : ARRAY OF CHAR; VAR c : CHAR);
VAR
    number : ARRAY 10 OF CHAR;
    n : LONGINT;
BEGIN
    AssocList.GetAssoc(sc, name, number);
    Attributes.StrToInt(number, n);
    c:=CHR(n);
END GetSpecialChar;

PROCEDURE IsKnown*(name : ARRAY OF CHAR) : BOOLEAN;
BEGIN
    RETURN (AssocList.HasAssoc(sc,name));
END IsKnown;

PROCEDURE IsSpecialChar*(ch : CHAR) : BOOLEAN;
VAR
    s : ARRAY 10 OF CHAR;
BEGIN
    Attributes.IntToStr(ORD(ch), s);
    RETURN (AssocList.HasAssoc2(sc, s));
```

```
END IsSpecialChar;

PROCEDURE StoreSpecialChar*(VAR o : Files.Rider; ch : CHAR);
VAR
    s : ARRAY 10 OF CHAR;
    name : ARRAY 50 OF CHAR;
BEGIN
    Attributes.IntToStr(ORD(ch), s);
    AssocList.GetAssoc2(sc, name, s);
    IF name[0]#0X THEN
        Strings.WriteStrToFile(o, "&");
        Strings.WriteStrToFile(o, name);
        Strings.WriteStrToFile(o, ";");
    END;
END StoreSpecialChar;

PROCEDURE NewSpecialCharMap*;
CONST buttonsize=17;
VAR p : Panels.Panel;
        b : BasicGadgets.Button;
        s : AssocList.AList;
        cx, cy: INTEGER;
        n : LONGINT;
        am : Objects.AttrMsg;
BEGIN
    NEW(p); Panels.InitPanel(p);
    s:=sc;
    cx:=3; cy:=-20;
    WHILE s#NIL DO
        Attributes.StrToInt(s.b, n);
        IF n>0 THEN
            BasicGadgets.NewButton;
            b:=Objects.NewObj(BasicGadgets.Button);
            b.caption[0]:=CHR(n);
            b.caption[1]:=0X;
            am.name:="Cmd";
            am.class:=Objects.String;
            am.id:=Objects.set;
            am.s:="SpecialChars.InsertChar";
            b.handle(b,am);
            b.W:=buttonsize; b.H:=buttonsize;

            Panels.InsertChild(p, b, cx,cy);
            IF cx<70 THEN cx:=cx+buttonsize+1;
            ELSE p.W:=cx+buttonsize+2; cx:=3;  cy:=cy-buttonsize-1;
            END;
        END;
        s:=s.next;
    END;
    IF cx=3 THEN cy:=cy+buttonsize+1; END;
    p.H:=-cy+5;
    p.state:=p.state+{Gadgets.lockchildren};
```

```
    Objects.NewObj:=p;
END NewSpecialCharMap;

PROCEDURE InsertChar*;
VAR b : BasicGadgets.Button;
        ch : CHAR;
        cm : Oberon.CaretMsg;
        im : Oberon.InputMsg;
BEGIN
    IF Oberon.Par.obj IS BasicGadgets.Button THEN
        b:=Oberon.Par.obj(BasicGadgets.Button);
        ch:=b.caption[0];

        cm.id:=Oberon.get; cm.car:=NIL; cm.text:=NIL; cm.F:=NIL;
        Display.Broadcast(cm);
        IF cm.car=NIL THEN
            Out.String("Error: No caret set"); Out.Ln;
        ELSE
            im.X:=cm.car.X; im.Y:=cm.car.Y;
            im.id:=Oberon.consume;
            im.keys:={};
            im.F:=cm.F;
            im.ch:=ch;
            im.fnt:=NIL;
            Display.Broadcast(im);
        END;
    ELSE
        Out.String("Error!!");
    END;
END InsertChar;


BEGIN
    Out.String("Loading special character information"); Out.Ln;
    AssocList.Load(sc,"SpecialChars.Text");
END SpecialChars.
```

B.17 Strings

```
MODULE Strings;

(* This module contains some basic string operations.
     (A string is just an array of chars)                    *)

IMPORT Texts, Files, SYSTEM;

    PROCEDURE UpcaseChar*(ch : CHAR): CHAR;
    (* If a lower case char is given, return the upper case of the char,
        otherwise just return char *)
```

```
BEGIN
    IF ((ORD(ch)>= ORD('a')) & (ORD(ch)<=ORD('z'))) THEN
         RETURN CHR(ORD(ch)+ORD('A')-ORD('a'));
    ELSE RETURN ch;
    END;
END UpcaseChar;

PROCEDURE WriteStrToFile*(VAR R : Files.Rider; s : ARRAY OF CHAR);
VAR i : INTEGER;
        ch : CHAR;
BEGIN
    i:=0;
    WHILE s[i]#0X DO
        ch :=s[i];
        IF ch=0DX THEN ch:=0AX END;
        Files.Write(R,ch);
        i:=i+1;
    END;
END WriteStrToFile;

PROCEDURE CompareIgnoreUpcase* (s1, s2 : ARRAY OF CHAR): BOOLEAN;
(* Are strings s1 and s2 the same ?                          *)
(* The difference between small- and bigcaps is ignored, so  *)
(*   'this' and 'THIS'  are the same                         *)
(* Precondition: s1 and s2 are zero-terminated               *)
VAR i : INTEGER;
BEGIN
    i:=0;
   WHILE ((s1[i]#0X) & (s2[i]#0X) & (UpcaseChar(s1[i])=UpcaseChar(s2[i]))) DO
        i:=i+1;
   END;
   RETURN ((s1[i]=0X) & (s2[i]=0X));
END CompareIgnoreUpcase;

PROCEDURE Compare* (s1, s2 : ARRAY OF CHAR): BOOLEAN;
(* Are strings s1 and s2 the same ?                          *)
(* Precondition: s1 and s2 are zero-terminated               *)
VAR i : INTEGER;
BEGIN
   i:=0;
   WHILE ((s1[i]#0X) & (s2[i]#0X) & (s1[i]=s2[i])) DO
        i:=i+1;
  END;
  RETURN ((s1[i]=0X) & (s2[i]=0X));
END Compare;

PROCEDURE Length*(s : ARRAY OF CHAR) : INTEGER;
VAR c : INTEGER;
BEGIN
    c:=0;
    WHILE (s[c]#0X) DO c:=c+1; END;
    RETURN (c);
```

```
    END Length;

     PROCEDURE Concat*(VAR s1 : ARRAY OF CHAR; s2 : ARRAY OF CHAR);
      VAR i,j : INTEGER;
     BEGIN
         i:=0;  j:=0;
         WHILE s1[i]#0X DO i:=i+1 END;
         WHILE s2[j]#0X DO
              s1[i+j]:=s2[j];  j:=j+1;
         END;
         s1[i+j]:=0X;
     END Concat;

    PROCEDURE GetStringFromText*(text : Texts.Text; beg, end: LONGINT;
            VAR s  : ARRAY OF CHAR);
    VAR R : Texts.Reader; ch : CHAR; i : LONGINT;
    BEGIN
         Texts.OpenReader(R, text, beg);
         FOR i := beg TO end-1 DO
             Texts.Read(R,ch);
             s[i-beg] := ch;
         END;
         s[end-beg]:= 0X;
    END GetStringFromText;


BEGIN
END Strings.
```

B.18 Store

```
MODULE Store;

(* This module export one procedure:

     PROCEDURE Store(filename: ARRAY OF CHAR; t : Texts.Text);

    This procedure stores a html document in a file. Here, the internal representation is
    transformed into 'ordinary' html.

*)

IMPORT
    Texts, Out, Objects, Files, Figures, Fonts, Panels, TextGadgets,
    MainPanel, Strings, ListGadgets, SpecialChars;

CONST
    commentColor = 100;

PROCEDURE GetFontName(name : ARRAY OF CHAR; VAR  res : ARRAY OF CHAR);
```

```
VAR i : INTEGER;
BEGIN
    i:=0;
    WHILE name[i]#'.' DO
        res[i]:=name[i];
        i:=i+1;
    END;
    res[i]:=0X;
END GetFontName;

PROCEDURE Store*(VAR o : Files.Rider; t : Texts.Text);
VAR r        : Texts.Reader;
        ch   : CHAR;
        ob   : Objects.Object;
        fm   : Objects.FileMsg;
BEGIN
    Texts.OpenReader(r,t,0);
    Texts.Read(r, ch);
    WHILE ~r.eot DO
        r.lib.GetObj(r.lib, ORD(ch), ob);
        IF ob IS Figures.Circle THEN Strings.WriteStrToFile(o,"<LI> ");
        ELSIF ob IS Figures.PolyLine THEN Strings.WriteStrToFile(o,"<HR>");
        ELSIF ob IS ListGadgets.brFrame THEN Strings.WriteStrToFile(o,"<BR>");
        ELSIF ob IS Fonts.Char THEN
            (* normal char *)
            IF ch=0DX THEN ch:=0AX END;
            IF r.col=commentColor THEN
                Strings.WriteStrToFile(o,"<!--");
                Files.Write(o,ch);
                Texts.Read(r,ch);
                WHILE r.col=commentColor DO
                    IF ch=0DX THEN ch:=0AX END;
                    Files.Write(o,ch);
                    Texts.Read(r,ch);
                END;
                Strings.WriteStrToFile(o,"-->");
                Texts.OpenReader(r, t, Texts.Pos(r)-1);
            ELSE
                (* is it a special character ? *)
                IF SpecialChars.IsSpecialChar(ch) THEN
                    SpecialChars.StoreSpecialChar(o, ch);
                ELSE
                    Files.Write(o,ch);
                END;
            END;
        ELSE (* send FileMessage to object *)
            fm.id:=Objects.store;
            fm.R:=o;
            ob.handle(ob, fm);

            (* the Files.Rider has to be copied back! *)
            o:=fm.R;
```

```
        END;
        Texts.OpenReader(r, t, Texts.Pos(r));
        Texts.Read(r,ch);
    END;
END Store;

PROCEDURE StoreFile*(filename: ARRAY OF CHAR; p : Panels.Panel);
VAR
        f : Files.File;
       o : Files.Rider;
        t : Texts.Text;
        tf : TextGadgets.Frame;
        title : ARRAY 200 OF CHAR;
BEGIN
    tf:=MainPanel.GetTextGadget(p);
    t:=tf.text;
    Out.String("Saving file "); Out.String(filename); Out.Ln;

    f:=Files.New("test_out");
    Files.Set(o,f,0);
    Strings.WriteStrToFile(o,"<HTML>"); Files.Write(o, 0AX);
    Strings.WriteStrToFile(o,"<HEAD>"); Files.Write(o, 0AX);
    Strings.WriteStrToFile(o,"<TITLE>");
    MainPanel.GetTitle(p, title);
    Strings.WriteStrToFile(o, title);
    Strings.WriteStrToFile(o,"</TITLE>"); Files.Write(o,0AX);
    Strings.WriteStrToFile(o,"</HEAD>"); Files.Write(o, 0AX);
    Strings.WriteStrToFile(o,"<BODY>"); Files.Write(o, 0AX);

    Store(o,t);
    Strings.WriteStrToFile(o,"</BODY>"); Files.Write(o, 0AX);
    Strings.WriteStrToFile(o,"</HTML>"); Files.Write(o,0AX);

    Files.Register(f);
    Files.Close(f);
END StoreFile;

BEGIN
```

B.19 Tags

```
MODULE Tags;

IMPORT Strings, Out, Objects, Files;

CONST
    string = 0;
    bool = 1;

    getAttr = 25;
```

```
    setAttr = 26;

TYPE
    AttrList* = POINTER TO Attr;
    Attr* = RECORD
        name* : ARRAY 200 OF CHAR;
        tp*      : INTEGER;
        b*       : BOOLEAN;
        s*         : ARRAY 200 OF CHAR;
        next    : AttrList;
    END;

    Tag* = RECORD
        tagName*    : ARRAY 100 OF CHAR;
        vars*           : AttrList;
        beg*, end*   : LONGINT;
    END;

    AttrMsg* = RECORD (Objects.AttrMsg);
        al : AttrList;
    END;

VAR
    al1, al2 : AttrList;
    obj : Objects.Object;

    PROCEDURE AddAttr*(VAR al : AttrList; a : Attr);
    VAR al2: AttrList;
    BEGIN
        NEW(al2);
        al2^:=a;
        al2.next:=al;
        al:= al2;
    END AddAttr;

    PROCEDURE SetAttr*(VAR al : AttrList; a : Attr);
    VAR temp, next: AttrList;
    BEGIN
        temp:=al;
        WHILE ((temp#NIL) & (~Strings.CompareIgnoreUpcase(a.name , temp.name))) DO
         temp:=temp.next;
        END;
        IF temp#NIL THEN
            next:=temp.next;
            temp^:=a;
            temp.next:=next
        ELSE
            AddAttr(al, a);
        END;
    END SetAttr;

    PROCEDURE AddString*(VAR al : AttrList; name : ARRAY OF CHAR; s : ARRAY OF CHAR);
```

```
VAR a : Attr;
BEGIN
    a.tp:=string;
    COPY(s, a.s);
    COPY(name, a.name);
    SetAttr(al,a);
END AddString;

PROCEDURE AddBoolean*(VAR al : AttrList; name : ARRAY OF CHAR; b : BOOLEAN);
VAR a : Attr;
BEGIN
    a.tp:=bool;
    a.b:=b;
    COPY(name, a.name);
    SetAttr(al,a);
END AddBoolean;

PROCEDURE HasAttr*(al : AttrList; name : ARRAY OF CHAR) : BOOLEAN;
BEGIN
    WHILE ((al#NIL) & (~Strings.CompareIgnoreUpcase(name , al.name))) DO
        al:=al.next;
    END;
    RETURN (al#NIL);
END HasAttr;

PROCEDURE GetAttr*(al : AttrList; name : ARRAY OF CHAR; VAR a : Attr);
BEGIN
    WHILE ((al#NIL) & (~Strings.CompareIgnoreUpcase(name , al.name))) DO
        al:=al.next;
    END;
    IF al#NIL THEN a:=al^; END;
 END GetAttr;

PROCEDURE InitTag*(VAR t : Tag);
BEGIN t.vars:=NIL; END InitTag;

PROCEDURE HandleAttr*(VAR al : AttrList; VAR M : Objects.AttrMsg);
VAR a : Attr; temp : AttrList;
BEGIN
    IF M IS AttrMsg THEN
        IF M.id=getAttr THEN
            M(AttrMsg).al :=al; M.res:=0;
        ELSIF M.id=setAttr THEN
            al:=M(AttrMsg).al;
        END;
    ELSIF M.id=Objects.get THEN
        IF HasAttr(al,M.name) THEN
            GetAttr(al,M.name,a);
            IF a.tp=string THEN M.class:=Objects.String; COPY(a.s, M.s); M.res:=0;
            ELSIF a.tp=bool THEN M.class:=Objects.Bool; M.b:=a.b; M.res:=0;
            END;
        END;
```

```
    ELSIF M.id = Objects.set THEN
        COPY(M.name,a.name);
        IF M.class=Objects.String
            THEN a.tp:=string; COPY(M.s, a.s); SetAttr(al,a); M.res:=0;
        ELSIF M.class=Objects.Bool
            THEN a.tp:=bool; a.b:=M.b; SetAttr(al,a); M.res:=0;
        END;
    ELSIF M.id=Objects.enum THEN
        temp:=al;
        WHILE temp#NIL DO
            M.Enum(temp.name);
            temp:=temp.next;
        END;
    END;
END HandleAttr;


PROCEDURE add(name : ARRAY OF CHAR);
VAR a : Attr;
        am : Objects.AttrMsg;
BEGIN
    GetAttr(al2,name,a);
    am.id:=Objects.set;
    IF a.tp=string THEN am.class:=Objects.String; END;
    IF a.tp=bool THEN am.class:=Objects.Bool; END;
    COPY(name,am.name);
    COPY(a.s, am.s);
    am.b:=a.b;
    obj.handle(obj,am);
END add;

PROCEDURE Merge*(VAR o : Objects.Object; l2: AttrList);
VAR am: Objects.AttrMsg;
BEGIN
    obj:=o;
    al2:=l2;
    am.Enum:=add;
    am.id:=Objects.enum;
    HandleAttr(al2, am);
END Merge;

PROCEDURE ShowTag*(t : Tag);
VAR al : AttrList;
BEGIN
    Out.String("Tag: "); Out.String(t.tagName); Out.Ln;
    al:=t.vars;
    WHILE al#NIL DO
        Out.String("Variable: ");
        Out.String(al.name);
        IF al.tp=string THEN Out.String("     : "); Out.String(al.s); END;
        Out.Ln;
```

```
            al:=al.next;
        END;
    END ShowTag;

    PROCEDURE Store* (VAR o : Files.Rider; al : AttrList);
    BEGIN
        WHILE al#NIL DO
            IF ((al.tp=string) & (al.s[0]#0X)) THEN
                Strings.WriteStrToFile(o, al.name);
                Strings.WriteStrToFile(o, "=");
                Files.Write(o,'"');
                Strings.WriteStrToFile(o, al.s);
                Files.Write(o,'"');
                IF al.next#NIL THEN
                    Files.Write(o,' ');
                END;
            ELSIF ((al.tp=bool) & al.b) THEN
                Strings.WriteStrToFile(o, al.name);
                IF al.next#NIL THEN
                    Files.Write(o,' ');
                END;
            END;
            al:=al.next;
        END;
    END Store;
```

B.20 TextUtil

```
MODULE TextUtil;

IMPORT
        Oberon, Objects, Texts,  Strings, Out, Fonts, Display;


CONST
    maxlen = 128;
    commentColor = 100;

VAR W: Texts.Writer;

        search: RECORD
            time: LONGINT;
            len: INTEGER;
            buf: ARRAY maxlen OF CHAR;
            d: ARRAY maxlen OF SHORTINT
        END;

        l : Objects.Library;
```

```
PROCEDURE NewText*(): Texts.Text;
BEGIN
    Texts.New;
    RETURN (Objects.NewObj(Texts.Text));
END NewText;

PROCEDURE GetParameter*(VAR s1: ARRAY OF CHAR);
VAR
    par      : Oberon.ParList;
    t        : Texts.Text;
    s        : Texts.Scanner;
    r        : Texts.Reader;
    i, beg, end, time    : LONGINT;
    ch     : CHAR;
BEGIN
    par:=Oberon.Par;
    Texts.OpenScanner(s, par.text, par.pos); Texts.Scan(s);
    IF (s.class=Texts.Char) & (s.c="^") OR (s.line # 0) THEN
        Oberon.GetSelection(t, beg, end, time);
    ELSE t:= par.text; beg:=par.pos; END;

    (* now, the parameter of the command is in text t, starting at position pos *)
    Texts.OpenReader(r,t,beg);
    Texts.Read(r,ch); WHILE (ch=' ') DO Texts.Read(r,ch); END;
    i:=0;
    WHILE (ch#0X) & (ch#0AX) & (ch#0DX) & (ch#9X) & (ch#' ') DO
        s1[i]:=ch;  i:=i+1;
        Texts.Read(r,ch);
    END;
    s1[i]:=0X;
END GetParameter;


PROCEDURE InsertCharInText*(ch : CHAR;  text : Texts.Text; pos : LONGINT);
 VAR Wr: Texts.Writer;
BEGIN
    Texts.OpenWriter(Wr);
    Texts.Write(Wr,ch);
    Texts.Insert(text,pos, Wr.buf);
END InsertCharInText;

PROCEDURE AppendString*(VAR t : Texts.Text; s : ARRAY OF CHAR);
VAR Wr: Texts.Writer;
BEGIN
    Texts.OpenWriter(Wr);
    Texts.WriteString(Wr, s);
    Texts.Append(t, Wr.buf);
END AppendString;

 PROCEDURE InsertObjectInText*(obj : Objects.Object; text : Texts.Text; pos : LONGINT);
 VAR Wr: Texts.Writer;
         ref : INTEGER;
```

```
        c : CHAR;
BEGIN
        (* The object in inserted in the text at position pos.
             The following steps are needed:

                 1. If the library that is used (l) is full (that is, 255 objects
                    are in it, texts can only contain objects with
                    ref number <= 255), create a new one.
                 2. Insert the object in the library.
                 3. Open a writer, and set the used font to the private library
                 4. Write the reference-number of the object to the writer.
                 5.  Insert the Writer's buffer in the text at position pos
       *)
       IF  l.maxref=255 THEN
           Out.String("Library Full, creating new one.."); Out.Ln;
           NEW(l); Objects.OpenLibrary(l);
       END;

        (* open a writer and set the font to the private lib *)
        Texts.OpenWriter(Wr);
        Wr.lib := l;

        (* generate a reference number, and put the object in the library *)
        l.GenRef(l, ref);
   IF ref=0 THEN l.GenRef(l, ref); END;
        l.PutObj(l,ref, obj);

         c := CHR(ref);
        Texts.Write(Wr,c);
        Texts.Insert(text,pos ,Wr.buf);
 END InsertObjectInText;

PROCEDURE WriteStringToWriter*(VAR W : Texts.Writer; VAR s : ARRAY OF CHAR);
(* when a string contains carriage returns, the normal Texts.WriteString cannot
   be used (only the chars before the carriage return will be written). *)
VAR i : LONGINT;
BEGIN
    i:=0;
    WHILE (s[i]#0X) DO
         Texts.Write(W,s[i]);
         i:=i+1;
    END;
END WriteStringToWriter;

PROCEDURE MoveText*(VAR fromText : Texts.Text; beg, end : LONGINT; VAR toText : Texts.Text);
(* The piece of text from beg to end is moved to the other text. *)
VAR w : Texts.Writer;
BEGIN
    Texts.OpenWriter(W);
    Texts.Save(fromText, beg, end, W.buf);
    Texts.Append(toText, W.buf);
    Texts.Delete(fromText, beg, end);
```

```
END MoveText;

PROCEDURE CompileDk;
    VAR d, k, l: SHORTINT;
BEGIN k := 1; d := 1;
    WHILE k < search.len DO
        l := 0;
        WHILE (d + l < search.len) & (search.buf[l] = search.buf[d + l]) DO l := l + 1 END ;
        WHILE k <= d + l DO search.d[k] := d; k := k + 1 END ;
        d := d + 1
    END
END CompileDk;

PROCEDURE KMPsearch(text: Texts.Text; beg: LONGINT): LONGINT;
    VAR R: Texts.Reader; ch: CHAR; k: SHORTINT;
BEGIN
    IF search.len > 0 THEN
        Texts.OpenReader(R, text, beg); Texts.Read(R, ch);
        k := 0;
        WHILE ~R.eot DO
            IF ((ch = search.buf[k])  OR (Strings.UpcaseChar(ch) = search.buf[k])) THEN
                k := k + 1;
                IF k = search.len THEN RETURN Texts.Pos(R) - k + search.len
                ELSE Texts.Read(R, ch)
                END ;
            ELSIF k = 0 THEN Texts.Read(R, ch)
            ELSE k := k - search.d[k]
            END
        END
    END;
    RETURN -1
END KMPsearch;

PROCEDURE SearchStr*(t : Texts.Text; pos : LONGINT; s : ARRAY OF CHAR) : LONGINT;
 VAR i,j : INTEGER;
 BEGIN
      i:=0;
      WHILE (i#100) & (s[i]#0X) DO search.buf[i]:=Strings.UpcaseChar(s[i]); i:=i+1; END;
      search.time := Oberon.Time();
      search.len:=i;
      CompileDk;
      RETURN KMPsearch(t,pos);
 END SearchStr;

PROCEDURE CalcWidth*(t: Texts.Text; VAR width, height : INTEGER);
VAR r : Texts.Reader;
        w,h : INTEGER;
        ch : CHAR;
        ob : Objects.Object;
BEGIN
    w:=4; width:=0; h:= 0; height:=0;
    Texts.OpenReader(r,t,0);
```

```
        WHILE ~r.eot DO
            Texts.Read(r,ch);
            IF ((ch=0AX) OR (ch=0DX) OR (ch=0X)) THEN
                IF width<w THEN width:=w; END;
                height:=height+h+9; h:=0;
                w:=4;
            ELSE
                r.lib.GetObj(r.lib, ORD(ch), ob);
                IF ob IS Fonts.Char THEN
                (*  w:=w+ob(Fonts.Char).dx+1; *)
                    w:=w+ob(Fonts.Char).w+2;
                    IF  ob(Fonts.Char).h>h THEN h:=ob(Fonts.Char).h END;
                END;
                IF ob IS Display.Frame THEN
                    w:=w+ob(Display.Frame).W+1;
                    IF ob(Display.Frame).H>h THEN h:=ob(Display.Frame).H END;
                END;
            END;
        END;
        IF width<w THEN width:=w; END;
        height:=height+h+6; h:=0;
    END CalcWidth;

BEGIN
    NEW(l);
    Objects.OpenLibrary(l);
```

B.21 LibHTML.c

```
/* Loading HTML documents to file */
/* (simplification of WWW library */

#include <ctype.h>

#include "HTUtils.h" /* WWW general purpose macros */
#include "HTBrowse.h"
#include "HText.h"
#include "HTFormat.h"
#include "HTTCP.h" /* TCP/IP utilities */
#include "HTAnchor.h"   /* Anchor class */
#include "HTParse.h"    /* WWW address manipulation */
#include "HTAccess.h"   /* WWW document access network code */

#include "HTFWriter.h" /* For non-interactive output */
#include "HTMLGen.h" /* For reformatting HTML */
#include "HTFile.h" /* For Dir access flags */
#include "HTError.h"
#include "HTAlert.h"
#include "HTTP.h"
```

```
/* Public Variables
** ================
*/

PUBLIC char * HTAppName = "Libhtml";                    /* Application name */
PUBLIC char * HTAppVersion = VL;                /* Application version */

/* Private variables
** =================
*/
PRIVATE HTRequest * request;

/* dummy declarations to make sure that LINKER will not complain */
PUBLIC char *HTBinDir;
PUBLIC int HTDiag;
PUBLIC char *HTPostScript;
PUBLIC char *HTPutScript;
PUBLIC char *HTSearchScript;

/* Procedure that will remove the header from
** a just loaded file, and save it in the file 'html_file'
*/
void strip_header(void)
{  FILE *f1, *f2;
   char c;
   char line[1000];
   int i;

   fprintf(stdout,"Strip_header\n");

   f1 = fopen("myfile","r");
   f2 = fopen("html_file","w");

   for(i=0; i<6; i++)
     fgets(line,1000,f1);

   fscanf(f1,"%c",&c);

   while (! feof(f1))
     { fprintf(f2,"%c",c);
       fscanf(f1,"%c",&c);
     };

   fclose(f1);
   fclose(f2);
 };

/* MAIN PROGRAM
** ------------
*/
```

```
void load_html_page(char* adres)
{
    FILE *myout;
    HTStream *myStream;

    request =  HTRequest_new();

/* Force predefined presentations etc to be set up. Now it is taking
   care of non-interactive, where no HTSaveLocally() of call backs are used */

   HTInteractive = YES;
   HTFormatInit(request->conversions);

   fprintf(stdout,"\nLoading %s\n",adres);
   if ( (myout = fopen("myfile","w"))==NULL)
     {     fprintf(stdout,"\nError opening myfile!\n"); };

   myStream = HTFWriter_new (myout,NO);
   request->output_format = WWW_MIME;

   request->output_stream = myStream;
   HTLoadToStream (adres, NO, request);

   fclose(myout);

   if ( request->error_stack!=NULL)
      { fprintf(stdout,"Error has occured!!!!\n"); }
   else
      { strip_header();
      /* HTRequest_delete(request); */
    };

} /* load_html_page */



void  main()
{
}


/* LibHTML */
```

Glossary

Browser
A tool used to read electronic documents. Examples are Mosaic and Netscape.

Desktop
A window in which other windows can be placed.

HTML
HyperText Markup Language. A document format in which a lot of documents are stored on the internet.

HTTP
Hypertext Transfer Protocol. A generic stateless object-oriented protocol, used for passing information on the internet.

Inspector
A little tool that can be used to modify attributes of Gadgets. First select a gadget (with right mouse button), then click the 'inspect' button inside the inspector. The inspector will then show all attributes of the gadget.

SGML
Standard Generalized Markup Language as defined in ISO 8879:1986, Information Processing Text and Office Systems. HTML is defined by a document in SGML.

Tag
Descriptive markup. There are two kinds of tags; start-tags and end-tags.

URL
Universal Resource Locator. Indicates the address of a document.

Bibliography

[1] Douglas C. McArthur. World Wide Web & HTML. Dr. Dobb's Journal, December 1994, pp 18-26

[2] SGML Open Home Page. http://www.sgmlopen.org/

[3] T. Berners-Lee, D. Connolly. HyperText Markup Language Specification - 2.0. http://www.ucc.ie/html/

[4] World Wide Web Consortium. WWW Names and Addresses, URIs, URLs, URNs. URCs. http://www.w3.org/hypertext/WWW/Addressing/Addressing.html

[5] Andrew Davison. Clickable Images in HTML. Dr. Dobb's Journal, September 1995. pp 18-27

[6] Andrew Davison. Coding with HTML Forms. Dr. Dobb's Journal, June 1995, pp 70-109

[7] N.Wirth, J. Gutknecht. Project Oberon - The Design of an Operating System. and Compiler. Addison-Wesley.

[8] Institute for Computer Systems, of the ETH Zürich. The Oberon Home Page. http://ics.inf.ethz.ch/Oberon.html

[9] H. Marais. Oberon System 3. Dr.Dobb's Journal, October 1994, pp 42-50

[10] Johannes L. Marais. The Gadgets Guide. File 'GadgetsGuide.Text', included in Oberon 3 package.

[11] Johannes L. Marais. Gadgets Programmer Reference. File 'GadgetsReference.Text', included in Oberon 3 package.

[12] Johannes L. Marais. The Gadgets System. File 'GadgetsReadMe.Text' included in Oberon 3 package.

[13] Johannes L. Marais. Homepage of Hannes Marais. http://huxley.inf.ethz.ch/ marais/

[14] G. E. Krasner and S. T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. JOOP, vol 1, no 3, August/ September, 1988, pp 26-49,

[15] W. R. LaLonde, and J. R. Pugh. Inside Smalltalk: Volume 2. Prentice Hall.

[16] World Wide Web Consortium. Hypertext Transfer Protocol. http://www.w3.org/hypertext/WWW/Protocols/Overview.html

[17] TBL, JFG, Henrik Frystyk, Hakon W. Lie e.a. WWW Library of Common Code. http://www.w3.org/hypertext/WWW/Library/

[18] J. Templ. SPARC-Oberon User Guide. File 'SparcOberon.Text', included in Oberon 3 package.

[19] Dave Ragett. HyperText Markup Language Specification Version 3.0. ftp://ietf.cnri.reston.va.us/internet-drafts/draft-ietf-html-specv3-00.txt