Projet de Semestre I

ÉCOLE POLYTECHNIQUE FÉDÉRALE
DE LAUSANNE / SWITZERLAND (EPFL)

ALTIFIER

WEB ACCESSIBILITY ENHANCEMENT TOOL

Version 1.0

IN COLLABORATION WITH

W3C-WAI-ER

PROJECT SUPERVISOR

Afzal Ballim, MEDIA – LITH – DI – EPFL

Prof. Giovanny Coray, LITH – DI – EPFL

---

by Michael Vorburger <mike@vorburger.ch>

http://www.vorburger.ch/projects/alt

November 1998 – February 1999

ABSTRACT

The goal of this project («ALTifier – Web Accessibility Enhancement Tool») was to research and implement tools to generate textual alternatives such as the ALT attribute for IMG and other graphical HTML elements.

Often images and some other HTML tags lack a textual alternative. This makes them inaccessible to screen readers, non-visual/text-only browsers and Braille readers. Adding alternate descriptions for these tags is one important aspect of making pages more accessible.

On one hand, the project focuses on HTML authors with a tool to set ALT texts on a site-wide per-image basis, instead per each occurrence in HTML documents. The idea of this tool is motivate HTML authors to provide ALT text for all images by facilitating this job. A graphical (GUI) and command-line (CLI) version of such an application are presented.

On the other hand, for users surfing on existing sites with lack of ALT text, a filter tool tries to guess ALT text by heuristics. This tool can be used in a proxy server or CGI which filters/transforms HTML and reads pages from the original Web server, inserts missing ALT text by attempting to "guess" it, and sends them on to the Web client.

The heuristics used to guess alternate text range from looking at an image's height & width to identify simple cases such as bullets and rulers, to analyzing hypertext links for extraction of usefull document link titles.

The project report gives a detailed description of the implementation and explains design choices.

***keywords**: ALT, IMG, HTML, C++, Filtering, W3C, WAI, Web, Accessibility, Braille, Screen Reader, Proxy, CGI, HTML Tool, Blind or Visually Impaired People, User Interface Transformation*

CONTENTS

1 SPECIFICATION AND REQUIREMENTS

1.1 INTRODUCTION

Web Accessibility is the research topic that deals with how to make the Web accessible to people with disabilities such as the blind who use Braille devices or screen readers, to people with low bandwidth connections or old browsers, or to people using devices such as miniature user agents like mobile phones etc.

Often images and some other HTML tags lack a textual alternative. This makes them inaccessible to screen readers, non-visual/text-only browsers and Braille readers. Adding alternate descriptions for these tags is one important aspect of making such pages more accessible.

The main sources of information are the W3C's Web Accessibility Initiative (WAI[1]) or Webable[2]. Of major interest to this project are the "WAI Guidelines For Authoring Tools"[3] and the general "WAI Accessibility Guidelines"[4].

Note that accessibility should not be confused with Usability in Web Design, the research topic that deals with questions of clear structure and presentation of a Site, see for example Nielson's UseIt[5] articles.

This project develops a complete toolkit which can be used to add and edit accessibility enhancing textual alternatives, with three different main modules implemented to address the issues:

- Scanning HTML documents for textual alternatives, and rewriting HTML with new ALT
- Guessing missing ALT, based on various rules and heuristics as shown later
- ALT Registry, used as look-up "database" for the Guessing, incl. XML Export & Import

All the technically different forms of textual descriptions are technically stored in `ALT` or `TITLE` attributes or the content of HTML tags and will be called "ALT tags" or simply "ALT" in this paper. The first part of the toolkit is an HTML analyzer/scanner that allows to retrieve and set ALT independent of the different tag types, and the distinction of attribute or content. This scanner is HTML 4.0 compatible,

[1] http://www.w3.org/WAI

[2] http://www.webable.com

[3] http://www.w3.org/WAI/AU/WAI-AUTOOLS.html

[4] http://www.w3.org/TR/WD-WAI-PAGEAUTH

[5] http://www.useit.com

and supports the ALT or TITLE attributes of IMG, AREA, INPUT, APPLET, OBJECT and FRAME tags, as well as tag's content as opposed to an attribute, for example in the HTML 4.0 OBJECT nesting.

Another part of the toolkit, a so-called back-end module, can automatically guess ALT to a certain degree, by using information in the same and in linked pages, and by recognizing trivial ALT description such as "* " for bullets and "" (empty) for spacer images etc.

The back-end modules are then combined into three applications: An ALT enhancing HTML filter, an ALT GUI tool, and an ALT CLI tool. The GUI & CLI tool both allow to crawl an entire site automatically.

After presenting the fundamental idea of different "image classes" that appear on Web pages, a brief introduction to the application's general structure is presented. Following is a detailed description of the heuristics used to find alternate textual representation, and how various HTML tags are affected.

1.2 IMAGE CLASSES

Several "image classes" appear in HTML documents and can be distinguished based on the following criteria. These classes influence the automatic choice ("guessing" & "suggestion") of ALT text:

- **Illustrations** are images carrying information and graphically explain or interpret some information often contained in the surrounding text already. They are usually "big" and should have a meaningful `ALT` and ideally `LONGDESC`, both of which are difficult to "guess" automatically. (In theory, ALT for these images could sometimes be identified by looking at the textual context, meaning the preceding and following paragraphs, and applying some natural language recognition. This initial idea was dropped because repeating existing text seemed of limited practical use.)

- **Navigation** aid images are graphical buttons and similar images, which appear inside a link or image map. A short and useful ALT for an image of this class can be found by looking at the link target itself or using textual links with the same target. (In theory, chances are also high that OCR recognition would succeed for this kind of "button" image, often having "Next" or "Support" or similar text on it. The additional benefit of using OCR in the guessing did not seem substantial enough to lead to an implementation during this project, though.)

- **Presentation and Decoration:** This images are used to make a page "look nice", but they usually don't contain any valuable information. Some well known presentational images are graphical rulers and bullets, substituting `HR` and `UL`/`LI`. Images in this "well-known class" have standard and constant ALT, such as '`*)`' and '`-----`' or similar. Another simple example are transparent 1x1 GIF images often used by professional web designers for layout purposes. They represent another "well-known class" with `ALT=""`. (Such transparent GIFs could be recognized either by it's minimal file size, often 34-43 bytes only, or by their minimal image size, often 1 pixel only height or width.)

Note that a class such as "**icons & symbols**" does not fall into this categorization, as an icon could be anything from illustrational to navigational to presentational, depending on it's usage. Note also that a **thumbnail** image of the form `<a href="bigpict.jpg"><img src="minipict.gif"></a>` will usually belong into the category illustrations, not navigation, even though contained in a link.

1.3 TOOLKIT STRUCTURE, MODULES AND OVERALL ARCHITECTURE

Back-End Structure

ALTifier consists of a core engine (back-end modules) with the general functionality (scanning, storing, guessing, writing) which is used from several front-end applications:

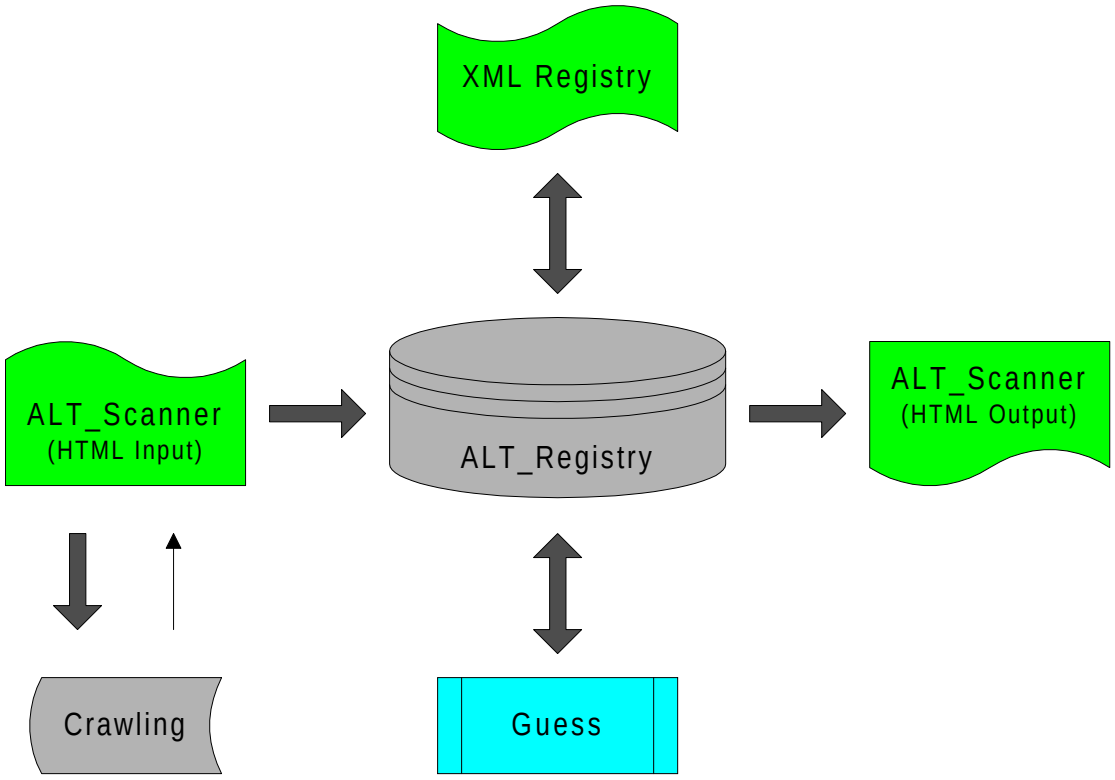


Image 1: ALTifier Toolkit General Structure (ALT_Scanner HTML Reading & Writing, ALT_Registry storage incl. XML Export/Import, Guess Engine, Crawling list)

- ALTifier lexical analyzer/scanner back end, to scan and write HTML.
  Built around a LEX definition, platform neutral C++ compiled by VC++ & gcc.
- ALT_Registry to store ALT information found by the scanner and needed by Guess.
- ALTifier heuristics engine back end, platform neutral C++ compiled by VC++ & gcc.
- Crawl engine for interactive Windows & UNIX front-ends, shared code with KISSfp[6]

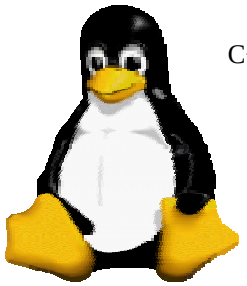
[6] http://www.vorburger.ch/kissfp

Front-Ends

Three front-ends were built around the above core engine:

- `ALT_Filter`, can for example be integrated HTTP proxy server[7] for auto-repair of ALT without human intervention. Not using crawling and XML back-end modules.
- UNIX command-line tool `ALT_Report` to retrieve ALT for an entire site, then edit them manually in an XML file. Written in simple C++ with gcc/Visual C++.
- Interactive site-wide Windows GUI front end.

Platform & Environment



The back-end and CLI front-end was developed under MS Windows using Visual C++ 5.0 and LINUX gcc, because these systems were available and the author had prior usage experience. This is platform neutral C/C++.

The GUI front-end is based on Inprise's (former Borland) excellent RAD tool "C++ Builder" and is probably not easily portable to any other platform.

[7] What's a Proxy Server? See: http://webopedia.internet.com/TERM/p/proxy_server.html

2 USAGE

2.1 ALT_FILTER REPAIR

This is a sample of how the filter CLI front-end interface looks like:

```
mike@alinux:/home/mike/ALT/src > alt_filter

ALTifier 1.0 -- http://www.vorburger.ch/projects/alt/
(C) Copyright 1998-1999 Michael Vorburger (alpha ware)

USAGE: alt_filter FILE.HTML
       Reads FILE.HTML, improves ALT, and writes back to STDOUT.
```

This filter can be for example be "plugged" into a proxy server realized by the author in an earlier project[8] or any other proxy or CGI that can call an external HTML filter.

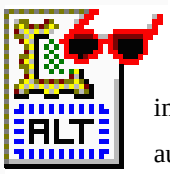
At the time of writing, a version of the filter was installed on the Accessibility Enhancement gateway (CGI) by Silas Brown from Cambridge University at http://ssb22.joh.cam.ac.uk/scripts/access.

Filtered HTML Sample

Original HTML Input	**Filtered Output**
`<img src="mikey.gif">`	`<img src="mikey.gif"` **`alt="Mikey Mouse"`**`>`
`<object data="mikey.gif">Mikey Mouse</object>`	`<object data="mikey.gif">Mikey Mouse</object>`
`<a href="/support.html">Support Area</a>`	`<a href="/support.html">Support Area</a>`
`<a href="/support.html"><img src=support_button.gif></a>`	`<a href="/support.html"><img src=support_button.gif` **`alt="Support Area"`**`></a>`
`...<area href=/support.html>...`	`...<area href=/support.html` **`alt="Support Area"`**`>...`
`<a href="http://www.vorburger.ch/kissfp'><img src=kissfp_button.gif></a>`	`<a href="http://www.vorburger.ch/kissfp'><img src=kissfp_button.gif` **`alt="http://www.vorburger.ch/kissfp"`**`></a>`

A complete example of filtered HTML showing all enhanced tags is given in the appendix.

2.2 WINDOWS GUI FOR WEB AUTHORS



Below is a snapshot of how the Windows GUI front-end interface looks like. Note that the idea clearly is to motivate HTML authors to set good ALT descriptions on all images manually; there is no 'Suggest All' or 'Quick Run' feature to set all ALT automatically with one click, and there is never going to be one for this reason.

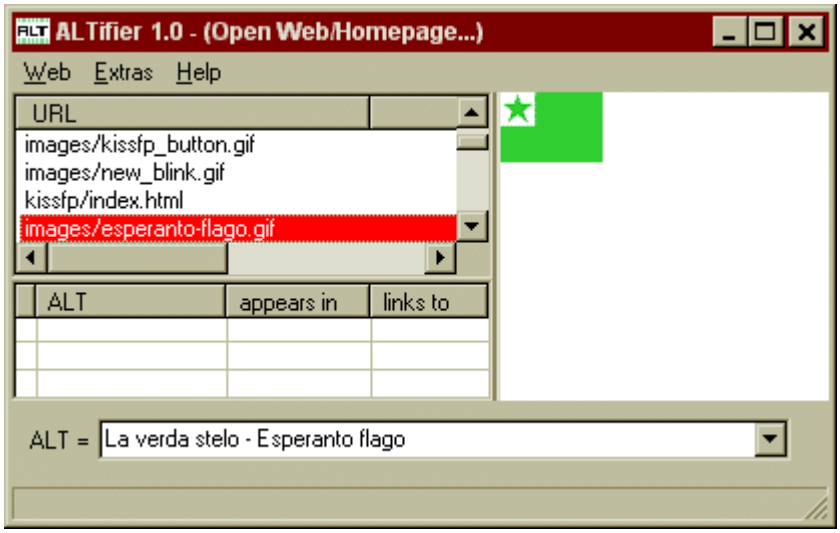


Image 2: The GUI Windows interface for ALTifier

The usage of this tool is straighforward: Menu "Web/Open..." asks for the homepage, which is used as starting point for crawling an entire site. The crawling is pretty quick and is typically a few seconds for medium sites with up to a few hundred pages.

The upper left pane shows all elements. When one is selected, the lower left pane displays all tags usings the element, and the right pane shows a preview of the element. The lower pane "ALT =" allows to edit ALT, clicking on the "Combobox" presents a list with automatic suggestions.

[8] http://www.vorburger.ch/projects/proxy (The author of this paper is not aware of the quality of the wwwoffle proxy server which "inspired" this simple proxy server. It seems to work well for normal usage. For a serious application, it might be worth investigating integration with the Squid or Harvest Proxy source code or W3C's HTTP library.)

2.3 UNIX CLI COMMAND FOR WEB AUTHORS

It could be of interest to organizational- and company websites who want to preserve a "corporate identity" in their ALT, and enforce certain ALT on all of their "sub-sites" possibly maintained by different departmental webmasters, to define these ALT in an XML file and have it automatically applied by a batch tool.

The ALT_Report front-end tool is a first step, which reads and exports the ALT Registry:

```
mike@alinux:/home/mike/ALT/src > alt_report
ALTifier 0.9  --BETA--  http://www.vorburger.ch/projects/alt/
(C) Copyright 1998-1999 Michael Vorburger (alpha ware)

USAGE: alt_report [HOMEPAGE-CRAWL] [-noguess]

Crawls homepage (default: index.html) and linked pages for Tags to Altify
Does "ALT guessing" on the registry, say -noguess to prevent this.

ALT Registry Output in different formats: alt_info.txt, AltText.txt,
alt_db.xml. alt_crawl.log has messages from the crawl engine not related
to ALT scanning.
```

XML

The XML format output of the `alt_report` tool is a beta based on a suggestion from a post in the W3C-WAI-ER-IG mailing list. Here is a sample XML output this front-end application generates:

```
<?xml version="1.0"?>
<alt-repository>

<img-alt-use>
<img-src>clock.class</img-src>
   <alt-text>If you use a Java-enabled browser, you would see an animated clock.
      <as-used-in>linked.html</as-used-in>
</alt-text></img-alt-use>

<img-alt-use>
<img-src>images/anybrowser3.gif</img-src>
   <alt-text>Best viewed with ANY browser
      <as-used-in>index.html</as-used-in>
</alt-text></img-alt-use>

<img-alt-use>
 <img-src>images/bluebult.gif</img-src>
   <alt-text>*
      <as-used-in>index.html</as-used-in>
</alt-text></img-alt-use>

<img-alt-use>
 <img-src>images/fun_line.gif</img-src>
   <alt-text>_____________________________________________________________
      <as-used-in>index.html</as-used-in>
</alt-text></img-alt-use>
```

```
<img-alt-use>
 <img-src>linked.html</img-src>
   <alt-text>more ALT test samples
      <as-used-in>index.html</as-used-in></alt-text>
   <alt-text>More Examples (OBJECT # APPLET)
      <as-used-in>linked.html</as-used-in></alt-text>
   <alt-text>more ALT test samples
      <as-used-in>frameset.html</as-used-in></alt-text>
   <alt-text>Test Samples 2
      <as-used-in>frameset.html</as-used-in></alt-text>
</img-alt-use>

</alt-repository>
```

See chapter "Future" for a short discussion of actual XML parsing.

ALTText.TXT

The `ALTText.txt` format is compatible with the ALT Registry of the A-Prompt project; see http://aprompt.snow.utoronto.ca/ This is a sample output of `alt_report` in `alt_info.txt` format:

```
Version 1
6  clock.class             If you use a Java-enabled browser, you would see...
1  images/anybrowser3.gif  Best viewed with ANY browser
1  images/bluebult.gif     *
1  images/fun_line.gif     ____________________________________________________
5  linked.html             More Examples (OBJECT & APPLET)
```

3 IMPLEMENTATION

3.1 ALT SCANNER & HTML TAGS TO ALTIFY

The HTML ALT "scanner" is technically spoken a Lexical Analyzer built using the LEX tool. It makes extensive use of LEX's advanced features such as exclusive stacked states and could not be implemented using regular expressions only. (The following is a brief overview only, please have a look at the source code of module `ALT_LEX.L` shown in the last chapter of this paper for details.)

When reading HTML, `ALT_LEX.L` reports each tag with a structure of the form (`type`, `src`, `alt`, `link`) calling the following function, where `link` can be NULL for some tags, while `src` cannot:

```
ALT_Tag* tag_found(ALT_TYPE type, cchar* src, char* alt, cchar* link);
```

Each tag that links to a page which needs to be crawled to analyze an entire site calls this function:

```
void crawl_found(cchar* url);
```

The following HTML tags are scanned for and supplied with an ALT or similar attribute suitable for text based browsing. The same module & lexical analyzer is also used to (re)write HTML:

- `<IMG SRC=`src `ALT=`alt`>` maps to (type=IMG, img-src=src, alt=alt, link-url=NULL).
- `<A HREF=url><IMG` or `OBJECT SRC="`button.gif`" ALT=`alt`></A>` is an image inside a link, often a button, and maps to (type=IMG-LINK, img-src=src, alt=alt, link-url=url). Note that IMG is the only tag inside A, apart from maybe whitespace, but with no text following or preceding the IMG tag, which we shall call a "pure IMG link" in this paper.
- `<A HREF=url>...`[9]`<IMG/OBJECT SRC="`button.gif`" ALT=`alt`>...</A>` is a non-pure link image, which returns type=IMG-LINK-NONPURE and src, alt & link as above. The heuristics "ALT guess" engine distinguishes this case from the above.
- `<A HREF=url>...</A>` is a normal textual link that is reported as (type=A_TEXT, img-src=url, alt=..., link-url=url) which allows the Guess engine to use this link's content if an IMG or other element points to the same page.

[9] In the content of an `<A>...</A>` tag, "..." means any text, that is anything except all <tags> and leading and trailing white space cut off. The same holds in the contents of `<OBJECT>` and `<APPLET>`.

- `<IMG SRC=src ALT=alt ISMAP>` is a server side image map and is reported as a special type: (type=IMG-ISMAP, img-src=src, alt=alt, link-url=NULL). This allows the heuristics engine to set a standard ALT.

- `<AREA HREF=url ALT=alt>` client side image MAP maps to (type=AREA, img-src=url[10], alt=alt, link-url=url). Please note that ALT text for the full image map (`IMG` or `OBJECT` with `USEMAP`) is still required to tell the user that the image is an image map.

- `<INPUT TYPE="image" SRC=src ALT=alt [VALUE=]>` maps to (type=IMG, img-src=src, alt=alt, link-url=NULL). Note that type=IMG, as INPUT can be considered equivalent to IMG for the purpose of determining ALT text.

- `<APPLET [ALT=alt] (CODE=url | OBJECT=url)>...alt...</APPLET>` maps to (type=APPLET, alt=alt, img-src=url, link-url=NULL[11]). Note that the text is repeated in the content of the APPLET tag, when the scanner is writing HTML, if not already present.

- `<OBJECT [TITLE=alt] [ DATA=url | CLASSID=url ] >...alt...</OBJECT>` maps to (type=OBJECT, img-src=url, alt=alt, link-url=NULL). Url is set to either DATA or CLASSID, in this order of priority. ALT is repeated in the content for non HTML 4 aware browsers, with OBJECT nesting[12] handled correctly, that is writing ALT only as the content of the innermost OBJECT, and the TITLE attribute for the outermost OBJECT. Link is non-NULL if OBJECT appears in A as described above.

- `<FRAME SRC=url TITLE=title>` and `<IFRAME SRC=url TITLE=title>` maps to (type=FRAME, img-src=url, alt=title, link-url=NULL)

If no `ALT` attribute is found inside `IMG`, `AREA`, `INPUT` & `APPLET` tag, but a `TITLE` is present, the `TITLE` is returned as `ALT`. Similarly, if no ALT attribute is present in `INPUT/image`, but VALUE is, that is returned as alternative, but never written; because the LYNX documentation states: "Some document authors incorrectly use an ALT instead of VALUE attribute for this purpose. Lynx 'cooperates' by treating ALT as a synonym for VALUE when present in an INPUT tag with TYPE="image". (*This seems to be inconsistent with the latest HTML 4.0 specification.*) Please note that this concerns retrieving `ALT` only. When setting ALT the lexical analyzer will only write the correct `ALT` or `TITLE` attribute.

[10] The img-src of an AREA could theoretically be (temporarily) created by extracting/cutting the relevant part of the corresponding MAP/IMG/OBJECT. This could be of help for OCR ALT heuristics.

[11] If the APPLET has a special PARAM that denotes a linked URL, then that is used as link-url instead NULL. For now, this only recognizes a FrontPage proprietary notation.

Similarly to accepting `TITLE` instead of `ALT`, one could think of interpreting the `NAME`, `ID` or `CLASS` attributes. This was not implemented, because these attributes are of technical nature and are unlikely to provide a good textual alternative. Also, one could think of reading the beginning of a `LONGDESC` file for `ALT`. This was not implemented either, as it is unlikely an advanced HTML 4 author sets `LONGDESC` but not `ALT`.

If the lexical analyzer finds something like `<IMG ... ALT>`, notice the ALT attribute with no value, it returns ALT="". When writing, the attribute is 'completed' and output as full ALT="".[13]

Further points not implemented in this project, but possible as well:

- A server side image map could be queried for links by simulating clicks on a pixel raster.
- The `<NOFRAMES>`long-html-alt`</NOFRAMES>` tag inside `<FRAMESET>` could be supported, and/or it's absence be warned in the author mode front-end tools.
- `<FIELDSET>` tag that does not contain `LEGEND`. This new element FIELDSET allows authors to group boxes around form INPUT control. It is especially helpful to people with visual disabilities who may be accessing the form using a screen reader. However, for the FIELDSET to work properly, a LEGEND element, containing the header text for the grouping of controls must be added as the first element within the FIELDSET element.
- `<TEXTAREA>` can often present a problem for screen reader users because their labels, that tell users what to put in the box, may be placed on another line. To correct this problem, it is recommended that the field name (i.e. comments, etc.) be added as the *default field text* for edit boxes. For the TEXTAREA element the default text is simply the text appearing between the element's start and end tags. This serves to clarify the function of edit boxes whose text labels may have been cut off by the screen. A *NAME* should also be defined for both types of elements, especially if default text is not being used.
- "Cross-frame" images, meaning images linked to another frame. This is generated for example by MS PowerPoint, and is difficult to handle, because of the frame and the fact that there is no IMG tag, but just an image directly loaded into a frame. (This is deprecated use according to the example and comment in §2.11.5 of [WAI-GL-TECHNIQUES].)

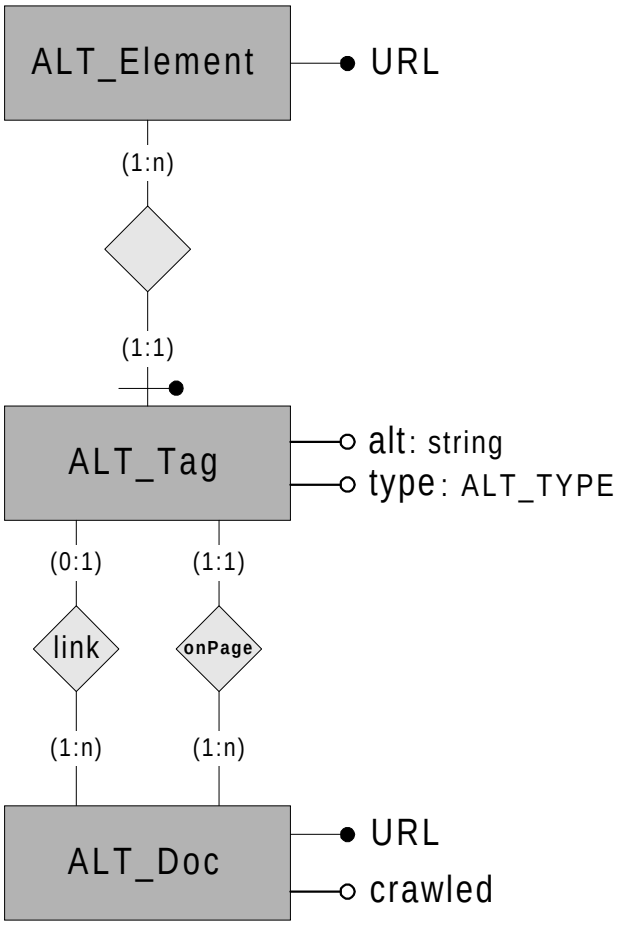
---

[12] In OBJECT nesting, the most multimedia intensive representation (ex. Java applet) is placed first. Then another OBJECT containing a different representation (ex. video or image) is placed between the start and end tags of the first OBJECT. Finally, a plain text description is placed between the start and end tags of the last representation, to be accessed by users who are blind or using text only browsers.

[13] MS FrontPage 'converts' ALT="" to ALT. For Lynx, these two constructions are not identical. For example, `<A ...><IMG ... ALT></A>` is displayed as `[LINK]`, whereas `<A ...><IMG ... ALT=""></A>` is correctly suppressed.

- Netscape specific <ILAYER> tag and multimedia <EMBED> tags.

3.2 ALT REGISTRY (STORAGE BACK-END MODULE)



ALT_Element is an "ALTifiable" HTML "element" that can have alternative textual attribute or content, such as a GIF file or referenced page. This element is used in the corresponding ALT_Tag(s):

ALT_Tag is one specific occurrence of an ALTifiable ALT_Element in one of following HTML Tags: IMG, AREA, INPUT, APPLET, OBJECT, FRAME:

```
struct ALT_Tag
{
    ALT_Element*  element
    char*         alt;
    ALT_TYPE      type;

    ALT_Doc*      onPage;
    ALT_Doc*      link;

    // ...
}
```

ALT_Element, ALT_Tag and ALT_Doc are integrated an in **ALT_DB**:

```
struct ALT_DB
{
    List<ALT_Element> Elements;
    List<ALT_Doc> Docs;

    ALT_Tag* Store(cchar* docurl,  ALT_TYPE type,            // called by
                   cchar* url,  cchar* alt,  cchar* link );  // LEX Scanner
    ALT_Element* Lookup(cchar* element_url);

    int Crawl(cchar* local_homepage);     // called by eg. GUI Front-End
    int ProcessDoc(FILE* in, FILE* out); // called by eg. Filter Front-End

    void Guess();                        // Entry point for ALT_Guess module
};
```

3.3 ALT GUESS HEURISTICS (HEURISTICS BACK-END)

The class member function `ALT_DB::Guess()` searches and enhances all tags in the Registry with no ALT or similar attribute already present, or with one present which is obviously automatic[14] and therefore not very informative. Alternatively a front-end such as the GUI tool can itself directly call the relevant function for one tag only:

```
alt_guess( ALT_Tag* tag, char* alt_Suggestions[], int max_Suggestions );
```

The function can construct and return a list with several ALT Suggestions. When acting as a HTML filter or proxy server, only one suggestion is requested and used. The following "heuristics" are used to "guess" ALT text, depending on the type reported by the lexical analyzer, each in order of priority. (This is a brief overview only, please have a look at the source code of module `ALT_GUESS.CPP` shown in the last chapter of this paper for details.)

Type = ALL

- For **pure** A-IMG/OBJECT-/A link, or AREA/FRAME, so for all elements which are a link, return the ALT text of this link as used in other occurrences, possibly with other elements and tags, if any.
- Check if ALT text for the element is already defined somewhere, lookup in ALT Registry.

Type = IMG-LINK-NONPURE

- For **non pure** IMG links, that is if there is some text between the `A`, `IMG` and `/A` tags, return an empty `ALT=""`. The reason for this is that such an image is likely to be a small inline decoration which, if it disappears in text browsing, is no loss of information as the existing link text suffices.

Type = IMG & OBJECT

- Set `ALT="------"` for graphical rulers. A graphical ruler decoration is identified if height > 1, width / height >= 10, width > 100, height < 50. The number of `'-'` characters is approximated dividing the pixel width by 10, but maximal 65.
- Set `ALT="* "` for graphical bullets. A graphical bullet decoration is detected if height > 5, width > 5, width / height <= 4, height < 30, width < 30.

- Set an empty ALT=""[15] if the IMG is "just" a decorative spacer. A spacer is recognized if the image has width or height = 1. (Checking for actual 1x1 GIF transparency or very small file size is not worth the trouble and waste of bandwidth.) The same ALT="" is applied to images with width or height = 0, as it can occur with special constructions such as external counter images etc, all of which are certainly of no interest to text-only agents.

- Return the **src filename**, cut off the path and extension of the file, replacing '_' and '-' by a blank space. This is based on the hope that webmasters or graphics designers give meaningful names to their images, like `/images/help.gif` (`ALT="help"`) etc.

Type = APPLET and OBJECT

- Return "`JAVA APPLET:` url" resp. "`OBJECT:` url" as default.

Type = IMG-ISMAP

- Return "[`SERVER SIDE IMAGE MAP`]" as default for an IMG-ISMAP. An author should overwrite this suggestion and set an empty ALT="" if there are alternative textual links on that page.

It is ensured that empty ALT="" is never set inside a pure link, for none of the above points. Please note that this tool will never insert ALT text or comment such as "Place Alt text here" or anything similar as this is completely useless.

---

[14] An automatically generated ALT is detected if it contains the word "byte", "gif", "jpeg", as tools such as MS FrontPage would often insert these when simply setting ALT=SRC.

[15] http://www.w3.org/WAI/GL/wai-gl-techniques-19980918.html#spacer-images

3.4 ALT_FILTER IMPLEMENTATION (FRONT-END)

The implementation of the ALT_Filter is **very** simple and based on the other modules:

```
FILE* fin = fopen( argv[1], "r" );
if ( fin == NULL )
    return 1;

theDB.ProcessDoc( fin, NULL );      // first pass

theDB.Guess();

rewind(fin);
theDB.ProcessDoc( fin, stdout );    // second pass

fclose(fin);
```

Notice the two "passes" used: First we read and analyze and HTML document, whilst we fill the Registry. Then we enhance the Registry by Guessing. Then we read again and at the same time write, using additional ALT now present in the Registry.

See the note about a "non re-spawning model" in the chapter "Future" for possible improved guessing by a "site-wide" filter which retains the Registry between invocations of the filter.

3.5 ALT_GUI IMPLEMENTATION (FRONT-END)

The Windows GUI application as shown in a previous chapter was implemented using the C++ Builder toolkit by Inprise, formerly Borland. A discussion of it's details would lead out of the scope of this project report.

Note that, apart from with a lot of (great!) graphical RAD buzz called VCL around it, the compiler underlying C++ Builder is a standard C++ compiler, former Borland C++ in fact. This made it very easy to include the ALT back-end engine.

4 FUTURE EXTENSIONS & DIRECTIONS

4.1 EMBEDDING THE ALT_FILTER IN MORE APPLICATIONS

The ALT_Filter could be embedded into various other applications using an HTML filter. Further front-ends could consist of a CGI interface, an NSAPI (Netscape), ISAPI filter (M$ IIS) or any other similar technology.

At the time of writing, dicussions were under way to direclty integrate ALT_Filter in the source code of the Accessibility Enhancement gateway (CGI) by Silas Brown from Cambridge University; see http://ssb22.joh.cam.ac.uk/scripts/access, http://www.accu.org/access/public/accessinstall.htm, or http://ban.joh.cam.ac.uk/~ssb22/access.html

Direct integration into a browser's rendering engine (ALT) would be possible: This tool can currently not be used as a Netscape/Internet Explorer browser plug-in, but MS IE is said to have a COM hook to allow filtering.

Of particular interest would be integration with the LYNX text browser and/or implementation of an ALT Enhancing Apache Filter Module, provided there is interest from the development community behind these two open source projects.

Another strand for possible follow-up is speed optimization: As it is well known, a traditional CGI or proxy which calls an external binary looses a lot of time by process (re)spawning at each invocation. Switching to a technology like Fast-CGI[16] would be interesting in this light.

ALT_Filter would benefit very much from a "non re-spawning" model in another, non speed-related, way as well: The guessed ALT tags would gradually get better, because if permanently loaded, all pages and tags scanned so far could be added to the Registry which would gradually produce better results for subsequent pages & guessing because it maybe already saw the element and the required ALT. Note that this is the reason why ALT_Filter and ALT_Report do **not** produce the same results, ALT_Report is **better** because it can use information on a "site-wide" basis, not just the current page.

[16] http://www.fastcgi.com

4.2 EXTENDING ALT_CLI AND XML PARSING

The ALT_CLI front-end application is very simple so far and more of a "proof of concept" than a useful application, because it **only exports** the ALT_Registry in XML, but does not allow any importing so far.

This could easily be done by allowing an XML parser to read an ALT.XML file back into the ALT Registry. This could be achieved by using one of the now many available XML parsers or by programming a simple XML reader ourselves.

4.3 EXTENDING THE SCANNER

The ALT scanner is pretty powerful and complete given it's ability to analyze constructions like HTML 4.0 nested OBJECT etc. A few idea for extensions remain anyway:

- It was thought about allowing to set `TITLE` and maybe even `LONGDESC` additional to `ALT`. The idea has not yet been implemented because end users could get confused and the front-ends would get more complicated. (Note that `TITLE` is supported for `OBJECT` & `FRAME` which do not have `ALT`.) In a future version an additional field (GUI) to set the LONGDESC file of an image might make sense though.

- HTML 4.0 allows `<TABLE Summary=%Text>` which provides a summary of the table's purpose and structure, and could be scanned and allowed for editing as well.

- `<NOFRAMES>...</NOFRAMES>` inside `<FRAMESET>` and other tags mentioned in §3.1 could be supported as well.

- A new "start of line" flag could indicate if an IMG occurred at the "beginning of a line" (or cell etc) to facilitate ALT_Guess's decision if an IMG is a graphical bullet.

4.4 EXTENDING ALT_GUESS

- Note that we don't yet fetch other pages to retrieve a document's title for using it as ALT in a link. We rely only on links to pages that we already scanned or will scan in a short moment, when crawling. A future extension could provide a method `ALT_Doc::Get_Title()` to retrieve a link title from a document's TITLE, maybe having to read local file or do an HTTP fetch. It could use the Document Link Title idea outlined in an appendix to this paper.

- Another idea is to make the guessing of graphical bullets more reliable by checking for a "start of line" flag and maybe also counting the number of occurrences, e.g. only if more than once in the same document.

- If the image has a comment built in the image file, this comment could be used as ALT text. To the author's current knowledge GIF and PNG[17] can have comment, while JPEG cannot.

- Could request (query) ALT text from and submit new descriptions to some kind of "ALT text server", yet to be set up, using PICS and/or RDF. The basic idea of this is outlined in http://www.w3.org/WAI/altserv.htm.

- OCR, using an API interface to "Xerox Textbridge" or a similar program if installed. This is likely to succeed for IMG-LINK buttons which show "Home" etc.

A new option could allow to terminate all ALT text with a punctuation ('.'). This seems to be helpful to screen readers, causing them to briefly pause, according to comments expressed on a mailinglist.

4.5 DEVELOPING ALT_GUI INTO SHAREWARE

The author is currently thinking of developing the ALT_GUI application into a Shareware tool.

There could be an interest for a commercial shareware version if the registration fee is kept relatively low, that is around $10-$20; the author has some previous experience[18] in this market. Notice that the ALT_Filter repair tool should and will remain free though.

[17] See http://www.w3.org/TR/PNG-Chunks.html#C.tEXt

[18] http://www.vorburger.ch/kissfp

5 ACKNOWLEDGEMENTS

I first want to thank my parents and friends for everything they gave to me.

A lot of inspiration and feed-back for this project and tool has been drawn from and given by the W3C-WAI-**ER**-IG (World Wide Web Committee/Web Accessibility Initiave/Evaluation and Repair/Interest Group) mailing list and phone conferences, where the author of this paper actively participated. I remain particularly thankful to: Al Gilman <asgilman@access.digex.net>, Chris Ridpath <chris.ridpath@utoronto.ca>, Daniel Dardiller <Daniel.Dardailler@sophia.inria.fr> and Leonard R. Kasday <kasday@acm.org> of the Institute on Disabilities/UAP at Temple University in Philadelphia PA.

The author of this paper also passively participated in the W3C-WAI-IG (different from –ER–) and read messages related to the topic of this project, and I remain thankful for some interesting points brought up in that list as well.

I would like to thank my colleague from university Julien Merçay <jmercay@scdi.org> for the idea of possibly integrating ALT_Filter in an Apache Module.

Last but not least, I wish to thank the supervisor of this project, Afzal Ballim <ballim@di.epfl.ch> from the MEDIA Research Group of the Laboratoire d'Informatique Théorique (LITH) of the École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland.

6 REFERENCES

[1] "Textual Equivalents - techniques to be used by tools taking HTML as input and trying to come up with textual alternatives for all sort of visuals lacking their native description", Daniel Dardailler, based on comments received on the WAI ER IG list. http://www.w3.org/WAI/ER/text-equiv.htm

[2] "WAI Accessibility Guidelines: Page Authoring", G. Vanderheiden, W. Chisholm, and I. Jacobs, eds. Available on-line at: http://www.w3.org/TR/WD-WAI-PAGEAUTH

[3] "WAI Accessibility Guidelines: Page Authoring Techniques", G. Vanderheiden, W. Chisholm, and I. Jacobs, eds. Available on-line at: http://www.w3.org/WAI/wai-gl-techniques

[4] "WAI Accessibility Guidelines: Authoring Tools", J. Treviranus , J. Richards, N. Sicchia, I. Jacobs. Available on-line at: http://www.w3.org/WAI/AU/WAI-AUTOOLS.html

[5] "WAI User Agent Guidelines", J. Gunderson, I. Jacobs. Available on-line at: http://www.w3.org/WAI/UA/WD-WAI-USERAGENT/

[6] "HTML 4.0 Recommendation", D. Raggett, A. Le Hors, and I. Jacobs, eds. Available on-line at: http://www.w3.org/TR/REC-html40/

[7] For the graphical ruler/bullet/spacer decoration recognition rules: "HTML++ Class Library with HTML Parser – TextOnly Filter", Michael Vorburger, http://www.vorburger.ch/projects/textonly

[8] "The Three -tions of Accessibility-Aware HTML Authoring Tools", J. Richards. Available on-line at: http://www.utoronto.ca/atrc/rd/hm/3tions.htm

[9] "WAB: World Wide Web Access for Blind and Visually Impaired Computer Users", a proxy server that inserts list of links and headings, among others. http://www.inf.**ethz.ch**/department/IS/ea/blinds/

[10] Microsoft on Accessibility, at http://www.microsoft.com/enable/

[11] "Lex und Yacc", Helmut Herold, Addison-Wesley

7 APPENDIX

7.1 COPY OF AN INTRODUCTION TO THE "A-PROMPT" PROJECT

The following paragraphs are copied from the 'A-Prompt' Project Proposal, at the time of writing found at http://aprompt.snow.utoronto.ca/docs/a-prompt.doc – This text is reproduced here as it gives a good overview into why Web Accessibility Enhancement tools are needed.

"Many Internet users have disabilities and rely on an adaptive technology system in order to access web pages. Over the last few years there have been many advances in improving the web compatibility of client-side access technologies such as screen readers, magnifiers, and alternative mouse systems. However, server-side information, the web pages themselves, need to adhere to HTML standards and may need HTML accessibility provisions in order to optimize access to users with disabilities.

The Internet has evolved into a system where the general consumer can easily create and post their own web pages. The World Wide Web Consortium (W3C) has taken steps in the right direction by adding accessibility provisions within the HTML, but this isn't necessarily enough. Many web designers use HTML editors such as Microsoft Front Page™, Adobe PageMill™ and SoftQuad HoTMetaL Pro™ and are unaware that HTML standards exist, let alone provision for accessibility.

One method of increasing the accessibility of the millions of web pages designed each year is to integrate HTML accessibility validation technology into commercial HTML authoring tools. This technology would be analogous to spell checkers and grammar checkers found in today's word processors. It would prompt users with pertinent information regarding HTML grammar and accessibility and offer suggestions for improvement. Many repetitive tasks would be automated, such as the addition of ALT-text or the replacement of server-side image maps with client-side image maps. The mere presence of such a system would increase general consumer awareness of access issues, regardless. Authoring tools with accessibility support will also help companies and academic institutions publish electronic information that comply with ADA and other disability legislation.

(...) Sidewalk curbcuts were originally installed to accommodate people in wheelchairs but they are now taken for granted by everyone, including mothers with baby carriages, shoppers with carts and rollerbladers. Electronic curbcuts also exist for technology. Accessible HTML lends to grammatically correct HTML which generate pages that are displayed properly regardless of browser or platform. Accessible HTML also ensures pages load quicker and transfer more easily between modalities, for user-agents other than desktop computers. Validation technology will enforce universal design principles that will benefit all web users."

7.2 DOCUMENT LINK TITLE (IDEA)

This is an idea of how to determine a document's "link title" for using it as ALT in for example an IMG link. See chapter "Future/Extending ALT_Guess" for details.

First, the `TITLE` tag is regarded. This document link title could be slightly different from TITLE, and is (can) be stored in the document in some META tag. It could be edited by the end user in author mode. This META tag could be set when a textual link to the document is found, allowing eg. image map links where this information is missing to "inherit" it from textual links.

If a TITLE contains any strings of the form ' – ' (space, dash, space) then the string until the occurrence of the such string is used as ALT. This is to eliminate long and repeated site descriptions in titles such as "Encountered Problems – SumInfo – Projects – Michael Vorburger's Homepage". This feature is not of first priority and can be turned off in the configuration. (If a "reverse site descriptive" title such as "Michael Vorburger's Homepage – Projects – SumInfo – Encountered Problems" is used, this could be recognized by checking if the start is always the same, namely the site's name.)

If no TITLE is found, the document's first H1 is used as ALT, if no H1 is present, the document's H2 is used, and so forth. Some maximum length (eg. 100-150 characters, or less?) might be imposed on the text. Cutting is always done at word borders. If no H_x is found, a short title is extracted from text that follows the BODY tag, taking eg. 100-150 (or less?) characters and cutting at word borders. If the link's A/HREF tag contains a # named destination, the TITLE check could be skipped and only the H_x check from the named destination, maybe a bit earlier, on, should be done. Another possibility would be to check `<META SUMMARY>` tags.

If the target document of a link is a frame set, the above procedure is executed on the first frame's content, or the frame named "main" if any, or the NOFRAME text if it seems to be reasonable. A reasonable NOFRAME text is one not containing more than 2-3 of the following keywords in the first 300 characters: "browser, no, not, please, upgrade, update, download, netscape, ns, ie, explorer". (This feature is not of first priority.)

7.3 COMPLETE ALT_FILTERED HTML EXAMPLE

```
<h2>A. IMG - Simple, in links, pure & non-pure</h2>

<p>1. A simple IMG without ALT, not linked nor used anywhere else:
<img src="images/news.gif" width=223 height=40 border=0 alt="news">
(Guess using filename from IMG-SRC URL, instead [IMAGE])

<p>2. Button in pure link, that is with empty content, absolute link:
<a href="http://www.vorburger.ch/kissfp/"><img src="images/kissfp.gif"
   border=0 alt="http://www.vorburger.ch/kissfp/"></a> (Guess using full URL
   of A-REF, not IMG-SRC, instead [LINK].)

or relative link:
<a href="../../kissfp/index.html"><img src="images/kissfp.gif" border=0
   alt="http://www.vorburger.ch/kissfp/"></a> (Guess using directory of A-
   REF, not IMG-SRC, instead [LINK]. Using directory because 'index' is
   pretty useless...)

<p>3. An IMG with ALT already set:
<img src="images/anybrowser3.gif" width=88 height=31 border=0 alt="Best
   viewed with ANY browser"> and using this IMG again, but forgot to set
   ALT:

<img src="images/anybrowser3.gif" width=88 height=31 border=0 alt="Best
   viewed with ANY browser"> (Guess using previous tag, retrieved from
   ALT_Registry, instead [IMAGE])

<p>4. Inline icon in non-pure link: <a href="http://www.yahoo.com">Go to
   http://www.yahoo.com<img src="images/leave-site.gif" border="0"
   alt=""></a> (Guess ALT="" because link content is not empty, instead
   [LINK])

See <a href=linked.html>more ALT test samples</a> on the next page.

<p>5. Button in pure link: <a href="linked.html">
<img src="images/next.gif" border="0" alt="more ALT test samples"></a>
(Guess using link's content text from other occurence of link, retrieved
   from ALT_Registry, instead [LINK])

<h2>B. Bullets & HR</h2>

<img src="images/curly.gif" width=16 height=17 alt="* "> A bullet (Guess "*
   " using bullet heuristic rules, instead [IMAGE])
<br><img src="images/bluebult.gif" width=13 height=13 alt="* "> Another
   bullet (heuristics is based on image size)

<p>The graphical horizontal ruler is an IMG and not a HR tag. (Guess "____"
   using ruler heuristic, based on size, variable width)

<p><center><img src="images/hr-construction.gif" width=590 height=8 border=0
   alt="____________________________________________________________"></center>


<h2>C. Specials</h2>

1. Transparent spacer GIF, used to enforce table cell width etc.
<img src="images/transparent.gif" width="10" height="1" alt="">
(Guess ALT="" because height=1, instead [IMAGE])

<p>2. Special Counter IMG with width=0 & height=0:
<img src="http://www.counter.com/cgi/counter.gif" width="0" height="0"
   alt=""> (Guess ALT="" because width=0 & height=0, instead [IMAGE])
```

```
<h2>D. INPUT</h2>

<form action="cgi-bin/form.pl" method="POST" enctype="application/x-www-
   form-urlencoded">Your name:
<input type="text" name="name" VALUE="Michael" alt="Michael">
<input type=image src=images/go.gif name=submit alt="go">
</form>


<h2>E. APPLET</h2>

<p>An applet with no ALT:
    <applet code="animation.class" codebase="_fpclass/" width="120"
   height="24" alt="JAVA APPLET: animation">JAVA APPLET: animation</applet>


<p>An applet with ALT attribute, but no pseudo-ALT in APPLET content:
    <applet code="fphover.class" codebase="_fpclass/" width="120"
   height="24"  alt="Java Applet, no Alt between APPLET & /APPLET">
    Java Applet, no Alt between APPLET & /APPLET</applet>

<A HREF="index.html">First sample page</a>

<p>An applet which is a link, eg. FrontPage's &quot;Hover buttons&quot; :
    <applet code="fphover.class" codebase="_fpclass/" width="120"
   height="24" alt="Java Applet, no Alt between APPLET & /APPLET">
      <param name="text" value="Hover-Button">
      <param name="color" value="#000080">
      <param name="hovercolor" value="#0000FF">
      <param name="textcolor" value="#FFFFFF">
      <param name="effect" value="glow">
      <param name="url" value="index.html" valuetype="ref">
    First sample page</applet>


<h2>G. OBJECT nested</h2>

<p>                                  <!-- First, try the Python applet -->
<OBJECT classid="http://www.observer.mars/the_earth.py"
 title="The Earth as seen from space.">
                                        <!-- Else, try the MPEG video -->
     <OBJECT data="the_earth.mpeg" type="application/mpeg">
                                          <!-- Else, try the GIF image -->
       <OBJECT data="images/the_earth.gif" type="image/gif">
                                            <!-- Else render the text -->
          The Earth as seen from space.
        </OBJECT>
     </OBJECT>
   </OBJECT>

<P>A simple OBJECT with no TITLE, no neested textual content:
<OBJECT classid=http://www.miamachina.it/analogclock.py
 title="OBJECT: http://www.miamachina.it/analogclock.py">
   OBJECT: http://www.miamachina.it/analogclock.py
</OBJECT>
```

8 SOME SOURCE CODE

Following is the source code of some core modules of ALTifier. Please note that important auxiliary modules and functions are not shown to keep this paper short.

This project consists of >3000 lines of code, only ca. 1000 (1/3) of which are shown here!

8.1 ALT_LEX.L

```
/* ===========================================================================
  * FILE:    alt_lex.l - Module SCANNER, written in (F)LEX
  * PROJECT: ALTifier, see http://www.vorburger.ch/projects/alt
  *
  * LAST MODIFIED: February 14, 1998
  * CREATED:       December 10, 1998
  *
  * AUTHOR:  Michael Vorburger [mike@vorburger.ch]
  *
  * COPYRIGHT (C) 1998 / 1999 BY MICHAEL VORBURGER (ALPHA WARE)
  *
  * Do not distribute with or (re)use this source code
  * without prior permission of the author.
  * ===========================================================================
 */


%{
    #include <string.h>
    #include "alt.h"
    #include "../../../shared-src/microsoft_borland.h"
    #include "../../../shared-src/webtools.h"
    #include "../../../shared-src/pathfoos.h"

    void ECHO_ALT();                            // FORWARD
    void ECHO_TITLE();                          // FORWARD
    void object_applet_close();                 // FORWARD
    void a_content_close();                     // FORWARD

    #ifndef CRAWL
        inline void crawl_found(cchar* url)  { return; };
    #endif

    /* IF GUI
    #define YY_FATAL_ERROR(msg) yy_gui_fatal_error( msg )
    char yy_gui_fatal_error( char msg[] ) { MsgBox ... }
    */
%}


 /* ---------------------------------------------------------------------------
  *  LEX commands and macro definitions
  */

%option case-insensitive
%option stack
%option never-interactive

    // #define YY_NEVER_INTERACTIVE 1
    #define PUSH yy_push_state
    #define POP  yy_pop_state

%x INIMG
%x INIMG_ISMAP
%x INAREA
%x ININPUT
```

```
%x INAPPLET
%x APPLET_CONTENT
%x INAPPLET_PARAM
%x INOBJECT
%x OBJECT_CONTENT
%x INFRAME

%x URL
%x COMMENT
%x PURE_CHECK
%x PURE_CHECK_IMG
%x ANY_OTHER_TAG

    /* WhiteSpaces+, Optional WS, one STRing, NotGreaterThans*, NotQuotes*
       STR is a string in "double quotes" or 'single quotes' or nada (untested)
       OQT is an Optional QuoTe, either double or single or none (optional)
       TAG is the start of a tag, CTAG is the end of a TAG. IMPORTANT: Always use
       eg. {TAG}A{WS} or {TAG}A{ETAG} because otherwise APPLET or AREA is matched
       as well!
    */

WS  [ \n\t\r\x0A\v\f]+
OWS [ \n\t\r\x0A\v\f]*
STR {OWS}(\"[^\"]+\")|(\'[^\']+\')|([^ \n\t\r\x0A\v\f\>]+)
NGT [^\>]*
NQT [^\"\'\><#]*
OQT [\"\']?

TAG     "<"{OWS}
ETAG    ({OWS}">")|({WS}{NGT}">")
                        /* ETAG must have WS/NGT and not OWS/NGT to prevent
                           matching /AREA or /APPLET in eg. {TAG}"/a"{ETAG} */


 /* ------------------------------------------------------------------------
  *  LOCAL VARIABLES (flags, counters, string buffers etc)
  */
    const int LEN = 256;        /* LEN (max) of most strings below           */

    static bool pure_A;         /* Is the <A> tag "pure" = no text?          */
    static int  in_OBJECT;      /* Inside an <OBJECT> tag, how deep neested? */
    static bool OBJECT_closed;  /* Inside an <OBJECT> tag, how deep neested? */
    static char src[LEN];       /* Value of the SRC attribute, IMG etc. tags */
    static char alt[LEN];       /* Value of the ALT attribute, all tags      */
    static char href[LEN];      /* Value of the HREF attribute, all tags     */
    static bool input_image;    /* Is this a type="image" INPUT tag?         */
    static char app_param_href[LEN];  /* "Hidden" HREF in an APPLET's PARAM? */
    static bool app_param_hasref;     /* Is this PARAM (maybe) a HREF?       */
    static char tag_content[LEN];     /* Content of APPLET/OBJECT tag        */
    static char* pc_tag_content;      /* dynamic pointer to the previous     */
    static int  img_width, img_height;  /* IMG's width= & height= attributes */
%%

 /* ========================================================================
  *  LEX RULES
  *
  *  This is executed every time yylex() is called:
  */
       href[0] = NAC;/* Important to do it here, because NOT in A, but in /A. */
       in_OBJECT = 0;
       base_external = false;
       BEGIN(INITIAL);    /* yyrestart() does *not* reset start condition */
```

```
 /* ------------------------------------------------------------------------
  *  ALT Rules
  */

{TAG}a{WS}{NGT}href={OQT} { ECHO; PUSH(PURE_CHECK); PUSH(URL); pure_A = true;
                            alt[0] = NAC; src[0] = NAC;
                            pc_tag_content=tag_content; *pc_tag_content = '\0'; }

<PURE_CHECK>{
        [^\><]*">"        { ECHO; /* Eat up any still open (A) tag, but
                                     don't open new one.                  */   }
        {TAG}img{WS}      { ECHO;
                            if ( src[0] != NAC ) {   // Maybe not the first IMG?
                               // then we have to "register" now because
                               // otherwise we would overwrite the previous
                               a_content_close();
                               tag_found( pure_A ? IMG_LINK : IMG_LINK_NONPURE,
                                          src, alt, href );
                               alt[0]=NAC; src[0]=NAC; // leave href unchanged!
                               pure_A = false;       // pure_A makes a difference
                                                     // for Guessing only, dummy.
                               // As we alreay have an IMG, it's ALT will be
                               // non-NULL, and the second would be okay if NULL.
                            }
                            PUSH(PURE_CHECK_IMG);
                          }
        {TAG}"/a"{ETAG}   { a_content_close();
                            if ( src[0] != NAC )  /* Did any IMG/OBJECT occur? */
                                tag_found( pure_A ? IMG_LINK : IMG_LINK_NONPURE,
                                           src, alt, href );
                              object_applet_close();
                              if ( !pure_A || tag_content[0] ) /* text A-link */
                                 tag_found( A_TEXT, href, tag_content, href );
                              ECHO; POP(); href[0] = NAC;                    }
        {TAG}             { ECHO; PUSH(ANY_OTHER_TAG);   /* pure_A unchanged */ }
        .|\n              { ECHO; if ( ( pc_tag_content - tag_content ) < LEN )
                                     *pc_tag_content++ = *yytext;            }
}

<ANY_OTHER_TAG>{NGT}">"  { ECHO; POP(); }


<PURE_CHECK_IMG>">"      { POP(); tag_lookup(IMG_LINK, src, alt, href);
                           ECHO_ALT();
                           /* Cannot tag_found() yet; delayed for pure-check */ }


<INIMG,INIMG_ISMAP,INAREA,ININPUT,INAPPLET,PURE_CHECK_IMG>{ /* Don't ECHO alt */
 {OWS}alt{OWS}={STR}{OWS}    { fputs(" ", yyout); strcpquote(alt, yytext, LEN); }
 {OWS}alt{OWS}               { fputs(" ", yyout); alt[0] = '\0';                }
 {OWS}title{OWS}={STR}{OWS}  { fputs(" ", yyout); if ( alt[0] == NAC )
                                                    strcpquote(alt, yytext, LEN);   }
 {OWS}title{OWS}             { fputs(" ", yyout);
                               if ( alt[0] == NAC ) alt[0]='\0';                }
 src{OWS}={STR}         { strcpquote(src, yytext, LEN); ECHO;                    }
 width{OWS}={STR}       { ECHO; char tmp[5];

                          strcpquote(tmp, yytext, 5); img_width = atoi(tmp);    }
 height{OWS}={STR}      { ECHO; char tmp[5];
                          strcpquote(tmp, yytext, 5); img_height = atoi(tmp);   }
}


<INFRAME,INOBJECT>{
   {OWS}title{OWS}={STR}{OWS}  { fputs(" ", yyout); /* Don't yet ECHO */
                                 strcpquote(alt, yytext, LEN);              }
   {OWS}title{OWS}             { fputs(" ", yyout); alt[0] = '\0';          }
   {OWS}alt{OWS}={STR}{OWS}    { fputs(" ", yyout); if ( alt[0] == NAC )
```

```
                                                strcpquote(alt, yytext, LEN);  }
   {OWS}alt{OWS}              { fputs(" ", yyout); if ( alt[0] == NAC )
                                                     alt[0]='\0';         }
}


{TAG}img{WS}   { ECHO; PUSH(INING); alt[0] = NAC;
                 src[0] = NAC; img_width = img_height = -1; /* NAV */         }
<INING>ismap   { ECHO; POP(); PUSH(INING_ISMAP);                             }
<INING>">"     { POP(); tag_found(IMG, src, alt, NULL, img_width, img_height);
                 ECHO_ALT();                                                 }
<INING_ISMAP>">" { POP(); tag_found(IMG_ISMAP, src, alt, NULL); ECHO_ALT();   }


{TAG}area{WS}        { ECHO; PUSH(INAREA); alt[0]=NAC; href[0]=NAC;             }
<INAREA>{
   href{OWS}={STR}   { ECHO; strcpquote(href, yytext, LEN); crawl_found(href);  }
   ">"               { POP(); tag_found(AREA, href, alt, href);
                          ECHO_ALT(); href[0] = NAC;                          }
}


{TAG}input{WS}                    { ECHO; PUSH(ININPUT); alt[0] = NAC;
                                    src[0] = NAC; input_image = false;        }
<ININPUT>{
        type{OWS}={OWS}"image"      { input_image = true; ECHO;               }
        {OWS}value{OWS}={STR}{OWS}  { if ( alt[0] == NAC )
                                         strcpquote(alt, yytext, LEN);
                                      ECHO;                                   }
        ">"                         { if ( input_image )
                                         tag_found(IMG, src, alt, NULL);
                                      POP(); ECHO_ALT();                      }
}


<INITIAL,PURE_CHECK>{
    {TAG}applet{WS} { ECHO; PUSH(INAPPLET); alt[0] = NAC; src[0] = NAC;
                      pc_tag_content = tag_content; *pc_tag_content = '\0';   }
}
<INAPPLET>{
        code{OWS}={STR}         { ECHO; strcpquote(src, yytext, LEN);         }
        object{OWS}={STR}       { ECHO; strcpquote(src, yytext, LEN);         }
        ">"                     { tag_lookup(APPLET, src, alt, NULL); ECHO_ALT();
                                  POP(); PUSH(APPLET_CONTENT);                }
}


<APPLET_CONTENT,OBJECT_CONTENT>{
        {TAG}object{ETAG}       { ++in_OBJECT; ECHO; }
        {TAG}"/object"{ETAG}    { if ( !OBJECT_closed ) {
                                    object_applet_close();
                                    tag_found(OBJECT, src, alt, href);
                                    src[0] = NAC;
                                    if ( alt[0] != NAC ) fputs(alt, yyout);
                                    OBJECT_closed = true;
                                  }
                                  if ( --in_OBJECT == 0 )
                                    POP();
                                  ECHO;                                       }
        {TAG}"/applet"{ETAG}   { object_applet_close();
                                  tag_found(APPLET, src, alt, href);
                                  src[0] = NAC;
                                  if ( alt[0] != NAC ) fputs(alt, yyout);
                                  ECHO; POP();                                }
        {TAG}param              { ECHO; PUSH(INAPPLET_PARAM);
                                  app_param_href[0] = '\0';
                                  app_param_hasref = false;                   }
        {WS}/{TAG}              { ECHO; /* Preserve layout, WS between Tags. */ }
```

```
        {TAG}                   { ECHO; PUSH(ANY_OTHER_TAG); /* No content   */ }
        .|\n                    { if ( ( pc_tag_content - tag_content ) < LEN )
                                      *pc_tag_content++ = *yytext;
                                  /* Don't yet ECHO  */                          }
}


<INAPPLET_PARAM>{
   {WS}name={OQT}url{OQT}   { ECHO; app_param_hasref = true;                  }
   {WS}value={STR}          { ECHO; strcpquote(app_param_href, yytext, LEN);  }
   {WS}valuetype={OQT}ref{OQT}  { ECHO; app_param_hasref = true;              }
   \>                           { ECHO; POP();
                                  if ( app_param_hasref ) {
                                    strcpy(href, app_param_href);
                                    crawl_found(href);             }         }
}


<INITIAL,PURE_CHECK>{TAG}object{WS} {
                      ECHO; PUSH(INOBJECT); alt[0] = NAC; src[0] = NAC;
                      pc_tag_content = tag_content; *pc_tag_content = '\0';
                      ++in_OBJECT; OBJECT_closed = false;                  }

<INOBJECT>{
     data{OWS}={STR} { ECHO; if ( in_OBJECT < 2 ) strcpquote(src, yytext, LEN); }
     classid{OWS}={STR}   { ECHO; if ( in_OBJECT < 2 && src[0]==NAC )
                            strcpquote(src, yytext, LEN);                    }
     ">"                  { tag_lookup(OBJECT, src, alt, href); ECHO_TITLE();
                            POP(); PUSH(OBJECT_CONTENT);                     }
}


{TAG}i?frame{WS}       { ECHO; PUSH(INFRAME); alt[0] = NAC; src[0] = NAC;       }
<INFRAME>{
        src{OWS}={STR} { ECHO; strcpquote(src, yytext, LEN); crawl_found(src);  }
        ">"            { POP(); tag_found(FRAME, src, alt, NULL); ECHO_TITLE(); }
}



 /* ------------------------------------------------------------------------
  *  General HTML Rules and document crawling,
  *  not directly related to ALT issues (except for <A>)
  */

<INITIAL,APPLET_CONTENT,OBJECT_CONTENT,PURE_CHECK>{
    {TAG}"!--"             { ECHO; PUSH(COMMENT); };
}
<COMMENT>.                 { ECHO;               };
<COMMENT>"-->"             { ECHO; POP();        };


 /* ------------------------------------------------------------------------
  *  HTML crawling rules: Go/PUSH(URL) whenever a link to crawl encountered:
     see also main rule for <A ...> above.

     NOTE: This is "useless" and not needed if !defined(CRAWLED)
     In this case, crawl_found() will be empty & logfile == /dev/nul.
     Because #ifdef etc. is not allowed in LEX rules, let's just scan
     anyway, without effect.
  */

{TAG}META{NGT}HTTP-EQUIV={OQT}Refresh{OQT}{NGT}CONTENT=[^=]+=  |
{TAG}"!--#include"{NGT}virtual={OQT}                           |
{TAG}ilayer{NGT}src={OQT}                              { ECHO; PUSH(URL); }


<URL>{
```

```
    {WS}         { fprintf(logfile, logmsg[24], current_url);                  }
    mailto:{NQT} { ECHO; POP();                                                }
    [a-z]+:{NQT} { ECHO; POP(); strcpquote(href, yytext, LEN);
                /* Don't crawl external http:// (file:)  but store for guess */ }
    #{NQT}  { ECHO; POP();    /* ignore # local name anchors    */             }
    {NQT}   { ECHO; POP(); crawl_found(yytext); strcpquote(href, yytext, LEN); }
    \"|\>   { ECHO; POP(); fprintf(logfile, logmsg[26], current_url);          }
}

    /* External BASE, eg. <base href="http://www.vorburger.ch/"> means that we
       cannot locally crawl this page. The flag base_external is checked in
       add_url() */

{TAG}base{NGT}href={OQT}"http://"     { ECHO; base_external = true; }


%%
 /* ============================================================================
  *  C FUNCTIONS
  */

    /* ECHO_ALT() & ECHO_TITLE() - Usually tag_found() or at least tag_lookup()
       (if tag_found not possible) will be called before ECHO_ to allow changing
       the ALT resp. TITLE by lookup.
    */
    void ECHO_ALT() {
        if ( alt[0] != NAC )
          { fputs(" alt=\"", yyout); fputs(alt, yyout); fputs( "\">", yyout); }
        else
            fputs( ">", yyout);
    };

    void ECHO_TITLE() {
        if ( alt[0] != NAC )
          { fputs(" title=\"", yyout); fputs(alt, yyout); fputs( "\">", yyout); }
        else
            fputs( ">", yyout);
    };



    void object_applet_close() {
        *pc_tag_content++ = '\0';
        strcp_nows(tag_content, tag_content, LEN);
        if ( tag_content[0] ) strcpy(alt, tag_content);
    };


    void a_content_close() {
        *pc_tag_content++ = '\0';
        strcp_nows(tag_content, tag_content, LEN);
        if ( tag_content[0] ) pure_A = false; else pure_A = true;
    };
```

8.2 ALT_GUESS.CPP

```
/* FILE:    alt_guess.cpp  -  The ALT "guess & heuristics engine"
   PROJECT: ALTifier, see http://www.vorburger.ch/projects/alt
   AUTHOR:  Michael Vorburger [mike@vorburger.ch]

   LAST MODIFIED: February, 1999
   CREATED:       January, 1999

   Code and rationals used herein are partly based on
   html_textonly.cpp, a module from a previous project
   named TextOnly, see http://www.vorburger.ch/projects/textonly
*/

#include "alt.h"
#include "../../../shared-src/pathfoos.h"

const int ALT_LEN = 256;


// FORWARD DECLARATIONS
//
bool isNULL( char* alt, ALT_TYPE type );
static void URL_to_ALT(cchar* url, char* ALT, int max = ALT_LEN);
static bool add_Suggestion( char* alt, char* Suggestions[],
                            int found_Suggestions );
static int from_fixString(char* fixALT, char* alt_Suggestions[],
                          int found_Suggestions);
static int from_scan_Tags(ALT_Tag* guessTag, ALT_Tag* t, bool needs_equal_link,
                          char* alt_Suggestions[], int found_Suggestions,
                          int max_Suggestions);


// ---------------------------------------------------------------------------
//  alt_guess(ALT_Tag* tag, char* alt_Suggestions[], int max_Suggestions)
//
//  We gradually fill the alt_Suggestions array using from_* in order of
// priority, until we have max_Suggestions, or cannot guess any further ideas
// for this tag.
//
// NOTE: PLEASE DON'T CHANGE THE ORDER OF THE IF()S UNLESS YOU KNOW EXACTLY WHY.
//
int alt_guess( ALT_Tag* tag, char* alt_Suggestions[], int max_Suggestions = 1 )
{
    int found_Suggestions = 0;

    // ---------
    // LOOP(...) do-while – Not yet implemented, but needed for GUI Suggestion

    // If this is a *LINK* tag, find ALT text of ANY tag that references
    // the page with same LINK. This has higher priority than the following
    // because eg. a "Next" button IMG could show the target and not the
    // same ALT as the next.gif had on the previous page.
    //
    if ( tag->link  &&  ( tag->type != IMG_LINK_NONPURE  ||  found_Suggestions>0
   ) )
        found_Suggestions = from_scan_Tags( tag, theDB.Lookup(tag->link->url)
        ->firstTag, false, alt_Suggestions, found_Suggestions, max_Suggestions);
    if ( found_Suggestions >= max_Suggestions )
        return found_Suggestions;
```

```
    // Find ALT text of same tag, as used on other pages (or later on this page)
    // with same LINK, that is href attribute etc is considered as well.
    //
    found_Suggestions = from_scan_Tags(tag, tag->element->firstTag, true,
                          alt_Suggestions, found_Suggestions, max_Suggestions);
    if ( found_Suggestions >= max_Suggestions )
        return found_Suggestions;


    // Find ALT text of same tag, as used on other pages (or later on this page)
    // and don't care about LINK.
    //
    found_Suggestions = from_scan_Tags(tag, tag->element->firstTag, false,
                          alt_Suggestions, found_Suggestions, max_Suggestions);
    if ( found_Suggestions >= max_Suggestions )
        return found_Suggestions;


    // For *NONPURE* IMG LINK suggest an empty ALT="" because there is explaining
    // text in the link and the image is likely to be a (small) inline decoration
    // which, if it disappears in text-only browsing, is no loss of real
    // information.
    //
    if ( tag->type == IMG_LINK_NONPURE ) {
        found_Suggestions = from_fixString("", alt_Suggestions,
                                                        found_Suggestions);
        if ( found_Suggestions >= max_Suggestions )
            return found_Suggestions;
    };


    // If this is a *LINK* tag, find ALT text by using the URL of the linked page
    //
    if ( tag->link ) {
        char ALT[ALT_LEN];
        URL_to_ALT( tag->link->url, ALT );
        found_Suggestions = from_fixString( ALT, alt_Suggestions,
                                                          found_Suggestions );
        if ( found_Suggestions >= max_Suggestions )
            return found_Suggestions;
    }


    // Horizontal ruler heuristics (IMG for HR)
    // NOTE: width/height == -1 means NOT present/read/set
    //
    if ( tag->type == IMG  &&  tag->img_width > 100  &&  tag->img_height > 1
      && tag->img_height < 50  &&  ( tag->img_width / tag->img_height >= 10 ) )
    {
        static char* ALT_HR =
"_____________________________________________________________________";
        int   ALT_HR_len = MIN(tag->img_width / 10, 65);
        ALT_HR[ALT_HR_len] = '\0';

        found_Suggestions = from_fixString( ALT_HR, alt_Suggestions,
                                                 found_Suggestions );
        ALT_HR[ALT_HR_len] = '_';   // ALT_HR has been copied, so restore.
        if ( found_Suggestions >= max_Suggestions )
            return found_Suggestions;
    }


    // Bullet heuristics (IMG for UL/LI)
    //
    if ( tag->type == IMG  &&  tag->img_width > 5  &&  tag->img_height > 5
      && tag->img_height < 30  &&  tag->img_width < 30
      &&  ( tag->img_width / tag->img_height <= 4 ) )
    {
```

```
        found_Suggestions = from_fixString( "* ", alt_Suggestions,
                                                found_Suggestions );
        if ( found_Suggestions >= max_Suggestions )
            return found_Suggestions;
    }


    // Decorative Spacer & invisible zero IMG
    //
    if ( tag->type == IMG  &&
         ( tag->img_width == 0  ||  tag->img_height == 0
        || tag->img_width == 1  ||  tag->img_height == 1 ) )
    {
        found_Suggestions = from_fixString( "", alt_Suggestions,
                                             found_Suggestions );
        if ( found_Suggestions >= max_Suggestions )
            return found_Suggestions;
    }


    // A server-side <IMG ... ISMAP> gets a fixed string
    // (if no other was found so far or more are requested)
    //
    if ( tag->type == IMG_ISMAP ) {
        found_Suggestions = from_fixString("[SERVER-SIDE IMAGE MAP]",
                                    alt_Suggestions, found_Suggestions);
        if ( found_Suggestions >= max_Suggestions )
            return found_Suggestions;
    };


    // APPLET
    //
    if ( tag->type == APPLET ) {
        char ALT[ALT_LEN];
        strcpy(ALT, "JAVA APPLET: ");
        char APPLET_src[ALT_LEN];
        URL_to_ALT( tag->element->url, APPLET_src );
        strncat( ALT, APPLET_src, ALT_LEN );
        found_Suggestions = from_fixString(ALT, alt_Suggestions,
                                                found_Suggestions);
        if ( found_Suggestions >= max_Suggestions )
            return found_Suggestions;
    };


    // OBJECT
    //
    if ( tag->type == OBJECT ) {
        char ALT[ALT_LEN];
        strcpy(ALT, "OBJECT: ");
        char OBJET_classid[ALT_LEN];
        URL_to_ALT( tag->element->url, OBJET_classid );
        strncat( ALT, OBJET_classid, ALT_LEN );
        found_Suggestions = from_fixString(ALT, alt_Suggestions,
                                                    found_Suggestions);
        if ( found_Suggestions >= max_Suggestions )
            return found_Suggestions;
    }


    // If still more ALT wanted, use the tag's own URL (that is eg. IMG src=)
    //
    char ALT[ALT_LEN];
    URL_to_ALT( tag->element->url, ALT );
    found_Suggestions = from_fixString( ALT, alt_Suggestions, found_Suggestions
    );
```

```
    return found_Suggestions;
};


// ---------------------------------------------------------------------------
//  add_Suggestion() - Helper function called from alt_guess. NOT exported!
//  Adds new suggestion but first checks if new alt is already in
//  Suggestions list. If not, inserts it and returns true, else false.
//
static bool add_Suggestion(char* alt, char* Suggestions[], int found_Suggestions)
{
    if ( isNULL(alt, IMG) )     // "IMG" here means we don't care, just a check!
        return false;

    for ( int i=0; i<found_Suggestions; i++)
        if ( strcmp( Suggestions[i], alt ) == 0 )
            return false;

    Suggestions[ found_Suggestions ] = alt;
    return true;  // alt_guess() will increment it's found_Suggestions counter.
}


// ---------------------------------------------------------------------------
//  alt_guess_theDB()  altifies the entire local Registery,
//                     using the above function.
//
//  One first has to crawl a site, using theDB.Crawl("index.html")
//  or read analyze at least one page, using theDB.AnalyzeDoc()
//
void ALT_DB::Guess()
{
    ALT_Element* e = theDB.Elements.list_head;
    while ( e ) {

        ALT_Tag* t = e->firstTag;
        while ( t ) {

            if ( isNULL(t->alt, t->type) )
            {
                char* alt = new char[ALT_LEN];
                if ( alt_guess(t, &alt) == 1 )
                {
                    if (  t->alt != NULL )
                        delete[] t->alt;
                    t->alt = alt;
                    t->guessed = true;
                }
                else
                    delete[] alt;
            }

            t = t->next;
        };

    e = (ALT_Element*)e->next;
    };
}


// ---------------------------------------------------------------------------
//   bool isNULL( char* alt, ALT_TYPE type )
//   Is ALT of type an "empty" ALT? Various reasons; eg. IMG/ALT="... byte"
//   is completely useless and stupid and reported as being empty/null.
//
bool isNULL( char* alt, ALT_TYPE type ) {
    if ( ( alt == NULL )  ||  ( *alt == NAC ) )
        return true;
```

```
    if ( type == IMG_LINK )  {                // inside a PURE IMG_LINK,
        char emptyALT[ ALT_LEN ];             // ALT="   " counts as NULL as well
        strcp_nows( emptyALT, alt, ALT_LEN ); // and is forbidden / replaced with
        if ( strlen( emptyALT ) == 0 ) {      // a guess.
            // LOG
            return true;
        }
    };

    if ( strstr( alt, "byte" )      // These presumably automatically
      || strstr( alt, "gif" )       // generated ALT (eg. by MS FrontPage)
      || strstr( alt, "jpg" )       // is nonsense and equals alt == NULL.
      || strstr( alt, "png" )
      || strstr( alt, "KB" )    )
        return true;

    return false;
};


// ---------------------------------------------------------------------------
//   void URL_to_ALT(cchar* url, char* ALT, int max = ALT_LEN) converts
//   an URL to ALT text, using either the entire URL (simple copy)
//   or just it's filename without extension or just the last directory.
//
static void URL_to_ALT(cchar* url, char* ALT, int max)
{
    // External links of type http:// (or ftp:// or anything else)
    // are returned as-is for ALT.
    //
    if ( strchr(url, ':') ) {
        strncpy(ALT, url, max);
        return;
    }

    // Otherwise (local files) use the filename without extension as ALT
    //
    strncpy( ALT, extract_FileName( url ), max );
    remove_extension( ALT );

    // "index.html" and "default.asp" etc. are not very usefull ALTs
    // so we provide the folder name before it
    //
    if ( stricmp(ALT, "index") == 0 || stricmp(ALT, "default") == 0 )
    {   char* directory_ALT = strcpy_new(url);
        if ( directory_ALT ) {
            remove_folder( directory_ALT );
            if ( directory_ALT[0] )
                strcpy( ALT, extract_FileName( directory_ALT ) );
            delete[] directory_ALT;
        }
    }

    str_replace_char(ALT, '_', ' ');
    str_replace_char(ALT, '-', ' ');

    // Capitalize first letter of every word... just to look nice.
}


// ---------------------------------------------------------------------------
//  int from_scan_Tags(...) scans the ALT Registry starting with Tag t for
//  other occurences of this tag. If needs_equal_link then only tags
//  embedded in the same link are considered. When found,
//  it add_Suggestion()s and in the end returns the number
//  of suggestions added.
//
```

```
static int from_scan_Tags(ALT_Tag* guessTag, ALT_Tag* t, bool needs_equal_link,
              char* alt_Suggestions[], int found_Suggestions,
              int max_Suggestions)
{
    while ( t )
    {
        if ( !isNULL(t->alt, t->type)
          && ( !needs_equal_link || t->link == guessTag->link )
          && add_Suggestion( t->alt, alt_Suggestions, found_Suggestions ) )
            if ( ++found_Suggestions >= max_Suggestions )
                return found_Suggestions;

        t = t->next;
    };

    return found_Suggestions;
}


// ---------------------------------------------------------------------------
//  from_fixString(char* fixALT, char* alt_Suggestions[], int found_Suggestions)
//  adds the constant fixed string fixALT to the list of suggestions.
//
//
static int from_fixString(char* fixALT, char* alt_Suggestions[], int
   found_Suggestions)
{   char* tmp_fixALT = strcpy_new( fixALT );

    if ( add_Suggestion( tmp_fixALT, alt_Suggestions, found_Suggestions ) )
        return ++found_Suggestions;
    else {
        if ( tmp_fixALT ) delete[] tmp_fixALT;
        return found_Suggestions;
    }
}
```

8.3 ALT_REGISTRY.H

```
/* ---------------------------------------------------------------------------
   FILE:    alt_registry.h  -  The ALT "database" (in-memory)
   PROJECT: ALTifier, see http://www.vorburger.ch/projects/alt
   AUTHOR:  Michael Vorburger [mike@vorburger.ch]

   LAST MODIFIED: January, 1999
   CREATED:       January, 1999
*/

// ---------------------------------------------------------------------------
//  THIS IS INCLUDED ONLY BY ALT.H
//  FUNCTIONS IN ALT_REGISTRY.CPP DECLARED THERE.
// ---------------------------------------------------------------------------


#ifndef ALT_REGISTRY_H
#define ALT_REGISTRY_H 1

#include "../../../shared-src/microsoft_borland.h"
#include "../../../shared-src/pathfoos.h"


struct ALT_Doc;
struct ALT_Element;
```

```
// ------------------------------------------------------------------------
//  A general base class for simple linked lists.
//
struct List_Element
{
    List_Element* next;
    char*         url;        // url, used as a "key" for sorting and look-up

    List_Element(cchar* u, List_Element* n)
        : next(n) { url = new_strcpy( u ); };

    virtual ~List_Element()
        { if ( url ) delete[] url; };

private:
    List_Element(void);
};


template<class Element> struct List
{
    Element* list_head;
    Element* Lookup(cchar* url);
    Element* Lookup(cchar* url, bool& isNew);
    Element* getNext(Element* e) { return (Element*)(e->next); };

    void Reset();
    List() : list_head(NULL) { };
    virtual ~List() { Reset(); };
};


// ------------------------------------------------------------------------
// One specific occurence of an ALTifiable HTML TAG
//
struct ALT_Tag
{
    ALT_Element*  element;    // ptr to it's "key" URL etc.
    ALT_Doc*      onPage;     // what HTML doc does this specific tag appear in?

    ALT_TYPE      type;       // as which type is the element used in this tag?
    char*         alt;        // what's the ALT in this tag?
    bool          guessed;    // was the alt text just guessed?
    ALT_Doc*      link;       // does this tag link do a doc? (Used in Guessing)

    int           img_width;  // IMG's width= & height= attributes, undefined= -1
    int           img_height; // YES, not "nice" and subclassing would be better.
                              // because -1 means NOT present/read/set

    ALT_Tag*      next;      // next occurence of this element, same or other doc


    ALT_Tag(ALT_Element* e, ALT_Doc* p, ALT_TYPE t, const char* a, ALT_Doc* l)
        : element(e), onPage(p), type(t), link(l), guessed(false)
        { alt = new_strcpy(a); next = NULL; img_width = img_height = -1; };

    ~ALT_Tag() { if ( alt ) delete[] alt; }

private:
    ALT_Tag();
};
```

```
// ------------------------------------------------------------------------
// One ALTifiable "element" such as a GIF or referenced page, which is used
// in the corresponding ALT_Tags.
//
struct ALT_Element : List_Element
{
    ALT_Tag*      firstTag;   // first specific Tag which uses this element
    ALT_Tag*      lastTag;    // last Tag which uses this element (speed-up ins)

    ALT_Element(cchar* url, List_Element* n)
        : List_Element(url, n) { firstTag = lastTag = NULL; };

private:
    ALT_Element();
};


// ------------------------------------------------------------------------
// One specific HTML document
//
struct ALT_Doc : List_Element
{
    ALT_Doc(cchar* url, List_Element* n, cchar* ref)
        : List_Element(url, n) { crawled = false; refby=new_strcpy(ref); };
    ALT_Doc(cchar* url, List_Element* n)
        : List_Element(url, n) { crawled = false; refby=NULL; };

    bool   crawled;     // has this doc already been crawled?
    char*  refby;       // who (first) referenced this doc? (when crawling)

     virtual ~ALT_Doc()
        { if ( refby ) delete[] refby; };

private:
    ALT_Doc();
};


// ------------------------------------------------------------------------
// ALT_DB - The whole story together...
//
struct ALT_DB
{
    List<ALT_Element> Elements;
    List<ALT_Doc> Docs;

    ALT_Tag* Store(cchar* docurl,  ALT_TYPE type,
                   cchar* url,  cchar* alt,  cchar* link );

    ALT_Element* Lookup(cchar* element_url);

    int Crawl(cchar* local_homepage);
    int ProcessDoc(FILE* in, FILE* out);

    void Guess();
};


#endif /* ALT_REGISTRY_H */
```