Tesina per il corso

LINGUAGGI E TRADUTORI

Politecnico di Torino

HTML++ DOCUMENT CLASS LIBRARAY IMPLEMENTING A TEXT-ONLY CONVERTER FOR AUTOMATICALLY ACCESSIBLE WEB SITES

Version 1.0

PROF. RIVOIRA

ASSI. M. TORCHIANO

Politecnico di Torino

Michael Vorburger (ERASMUS, EPFL/Switzerland)

July-August 1998

ABSTRACT

The goal of this project («TextOnly & HTML++ Class Libraray») was to implement a TextOnly converter for Web pages. This post-processor takes as input an HTML document with possibly rich formatting, using tables for layout and small IMG for bullets etc. It converts this to an output document again in proper HTML format (not ASCII text) which contains only text.

This text-only output format is an ideal input for Web accessibility devices such as screen readers or Braille devices, and for upcoming handheld devices with limited screen size & color depth. TextOnly is an HTML Filtering Tool according to the definition of the W3C's WAI ER IG[1].

Technically, this led to a C++ class library (framework) representing HTML documents and tags as various objects. A lexical analyzer and parser builds a document's structure in-memory as a linked C++ object tree. Once this object tree is built, recursive invocation of the `newTextOnly()` *method generates a text-only representation of the page.*

The hierarchical object representation allows a "smarter" text-only conversion than what current text-only scripts, mostly being written in Perl, can achieve. For example, graphical `IMG` *bullets are recognized and output as* `UL/LI`*, and* `IMG` *rulers are translated into HR. Other IMGs are divided into decorative and 'narrative' images, and treated following rules depending on their size, existing links etc. Superfluous white-space is eliminated.*

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

---

keywords: *TextOnly, HTML, C++, Filtering, W3C, WAI, Web, Accessibility, Braille, Screen Reader*

[1] World Wide Web Consortium's - Web Accessibility Initiative - Evaluation and Repair Interest Group; see http://www.w3.org/WAI/ER

CONTENTS

1 SPECIFICATION AND REQUIREMENTS

1.1 HTML++

The HTML++ library can be used in it's own right, ignoring the existence of the `newTextOnly()` methods. Any application that wants to parse HTML documents for "smart" post-processing could benefit from this library. For example, the HTML classes have the following methods:

- ♦ Navigational and content inquiries: `getNext()`, `getPrev()`, `getChild()`, `getParent()`, `HTMLTag::getAttribute(...)`
- ♦ "Smart" methods: `isBlank()`, `HTMLTagIMG::getNewALT()`, `HTMLTagIMG::isBullet()`

The object tree can be built through the parser from an HTML file. The lexical analyzer is built using the FLEX toolkit. The parser is built using the YACC toolkit. An alternative hand-wired parser was developed due to problems with documents not following the HTML specification, see §3.1 on page 7.

It is also possible to build the object tree dynamically during run-time from the application. A complete coding example is given in §2.2 "Using HTML++" on page 6.

1.2 TEXTONLY CONVERTER

The back-end engine that actually does the text-only conversion, performs the following tasks:

- ♦ An `IMG` contained in an `A-HREF` which also contains plain text is discarded.
- ♦ An `IMG` contained in an `A-HREF` which does not contain plain text is discarded and replaced by IMG's ALT text. If none is available, one is generated automatically.
- ♦ An `IMG` having certain dimensions is replaced by `HR` or `UL`/`LI`.
- ♦ A TABLE is discarded, replacing them by a linear enumeration of the cells contents.
- ♦ A Server Side Image MAP is discarded and replaced by a warning string.
- ♦ The BODY tag is replaced by a constant tag for white background, discarding background image and foreground colors for normal text and hypertext links.
- ♦ FORMs are preserved unmodified. Any Scripting (JavaScript) gets discarded, though.
- ♦ The following tags are preserved unmodified: H1-H6, ADDRESS, P, UL, OL, LI, DL, PRE, DIV, CENTER, BLOCKQUOTE, HR, XMP, LISTING, PLAINTEXT, A, BR; the font style elements TT, I, B, SUB, SUP, STRIKE; and the phrase elements EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE.
- ♦ Any tag not explicitly mentioned above is completely discarded. This includes familiar tags such as FONT/BASEFONT or BIG/SMALL, APPLET, and vendor specific extensions such as MARQUEE etc. (The U tag is discarded on purpose, I personnally feel underlining anything that is not a link is a strong usability violation.)

1.2.1 Front End

The front-end of the system is easily interchangeable, and while this implementation has a simple command-line interface that post-processes an entire site off-line (crawling) for subsequent uploading to a server, in the future a Proxy front-end, CGI or ISAPI filter (M$) could be added.

1.3 PLATFORM & ENVIRONMENT

The original version was developed under MS Windows using Visual C++ 5.0, as this system was available and the author had prior usage experience. Most of the code is platform neutral, and a port to UNIX (gcc) should be possible in a reasonable amount of time. The HTML++ library uses RTTI, but most C++ compilers should provide this formerly advanced C++ feature by now.

1.4 BACKGROUND ON WEB ACCESSIBILITY

Web Accessibility is the researching 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 band-width connections or old browsers, or to people using devices such as miniature user agents like mobile phones etc.

The main sources of information are the W3C's Web Accessibility Initiative (WAI[2]) or Webable[3]. A text-only page generated by this tool should always pass an Accessibility Evaluator such as BOBBY[4] with no error and warning messages at all.

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.

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

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

[4] http://www.cast.org/bobby

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

2 USER MANUAL

2.1 USING TEXTONLY

The command-line front-end can be invoked very easily as follows:

```
textonly.exe index.html
```

This will create a text-only version of copy that page in the sub-directory `./text-only/` and search for all pages linked from the given document as well. The idea is to run this tool off-line on an entire local site, for subsequent uploading to a Web server. A log file named `log.txt` is written to the current directory.

Technically, this command-line front-end does not do much more then executing the following code:

```
HTMLSite site( argv[1] );
site.generateTextOnly();
```

This front-end *crawls* the entire site based on links found in pages. It correctly handles transformation of relative to absolute links and other issues coming up for this problem. The code for this *crawling* feature is largely shared with another project of the author, KISSfp[6], unrelated to TextOnly otherwise.

2.2 USING HTML++

```
#include "htmlclasses.h"
#include "htmlsite.h"
```

Here is an example of how to dynamically build the HTML object tree dynamically during run-time:

```
HTMLDocument* pDoc = new HTMLDocument( new HTMLText("Hello world!") );
```

This loads the file `index.html` and builds the corresponding object representation using the parser:

```
HTMLDocument* pDoc = new HTMLDocument( "index.html" );
```

This line displays the constructed HTML doc with correct indention etc:

```
if ( pText )
    cout << *pText << flush;
```

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

3 DISCUSSION

3.1 THE LEXICAL ANALYZER

The lexical analyzer has been implemented using FLEX. It was partially inspired by W3C's "A Lexical Analyzer for HTML and Basic SGML"[7]. That SGML lexical analyzer was considered far too complex though and my implementation is considerably simpler, and does mostly the same job for HTML.

The lexical analyzer "knows" when to ignore white space and when to return it. Details about this can be found in the comments of the source code on page 9.

3.2 THE PARSER

The YACC/BISON parser shown on page 11 is fully functional, but proved to have difficulties with HTML documents that do not 100% comply with the HTML specification[8]. For example, wrong neesting of a FONT tag gets the parser which is based on <FONT> ... </FONT> and similar pairs of tags completely mad. Unfortunately, this kind of non-compliant HTML code is very frequent on the Web, and is sometimes even generated by HTML authoring tools such as for example M$ FrontPage.

It was tried to counter this problem using YACC `error` symbols in most rules, which helped sometimes but would still leave far too many existing HTML pages 'unparsable'.

[7] http://www.w3.org/TR/WD-html-lex

[8] http://www.w3.org/pub/WWW/MarkUp

4 REFERENCES

[1] Compilers, Principles, Techniques and Tools. Aho, Sethi, Ullman. Addison-Wesley. ("Dragon Book")

[2] Lex und Yacc; Helmut Herold; Addison-Wesley

[3] Stroustrup, Bjarne: The C++ Programming Language; 2nd edition; Addison-Wesley 1991 (93)

[4] Meyers, Scott: Effective C++, 50 simple ways to improve your programs and design; A.W. 1992

[5] Coplien, James: Advanced C++ Programming Styles and Idioms; Addison-Wesley 1992

5 APPENDIX: SOME SOURCE CODE

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

This project consists of ca. 3000 lines of code, only 600 of which are shown here!

5.1 THE LEXICAL ANALYZER (FLEX)

```
/* FILE:    html_lex.l - the LEXICAL ANALYZER, written in (F)LEX
   PROJECT: TextOnly, see http://www.vorburger.ch/
   AUTHOR:  Michael Vorburger [mike@vorburger.ch]
*/

   #include <stdlib.h>
   #include <string.h>
   #include "htmlclasses.h"
   #include "html_yacc_tab.cpp.h"

   #define YY_NEVER_INTERACTIVE 1
   // also avoids isatty() invocation. (which is not defined on Win32)


%option case-insensitive

%x INTAG
%x INCOMMENT
%x INSPECTAG
%x XMPMODE
%x PLAINTEXTMODE

WS [ \t\n\r\b\f]
ID  [a-z01-9.\-:/\\]

   /* ID has to match http-eqiv and http:// or c:\ for unquoted literals
      and is allowed to start with number as well because used for numeric values.
*/

   #define LVAL(first) yylval.str = strdup(&yytext[first])
   #define LVAL2(first, cutlast) yytext[yyleng-cutlast] = '\0'; LVAL(first)
   #define NADA yylval.nada = NULL
%%
   static bool ITWS; // Discard Inter/Tag Whitespace between Tags? (see html_yacc.y)

"<HTML"    { BEGIN(INTAG); LVAL(1); ITWS=true; return(HTML); };
"</HTML"   { BEGIN(INTAG); NADA; ITWS=true; return(_HTML);   };
"<HEAD"    { BEGIN(INTAG); LVAL(1); return(HEAD);         };
"<META"    { BEGIN(INTAG); LVAL(1); return(META);         };
"</HEAD"   { BEGIN(INTAG); NADA; return(_HEAD);        };
"<TITLE"   { BEGIN(INTAG); LVAL(1); return(TITLE);        };
"</TITLE"  { BEGIN(INTAG); NADA; return(_TITLE);         };
"<BODY"    { BEGIN(INTAG); LVAL(1); ITWS=false; return(BODY);   };
"</BODY"   { BEGIN(INTAG); NADA; ITWS=true; return(_BODY);   };

"<IMG"     { BEGIN(INTAG); LVAL(1); return(IMG);   };
"<BR"      { BEGIN(INTAG); LVAL(1); return(BR); };
"<A"       { BEGIN(INTAG); LVAL(1); return(A);    };
"</A"      { BEGIN(INTAG); NADA; return(_A);    };
"<FONT"    { BEGIN(INTAG); LVAL(1); return(FONT); };
"</FONT"   { BEGIN(INTAG); NADA; return(_FONT); };
"<SCRIPT"  { BEGIN(INTAG); LVAL(1); ITWS=true; return(SCRIPT);  };
"</SCRIPT"{ BEGIN(INTAG); NADA; ITWS=false; return(_SCRIPT);   };

"<TABLE"   { BEGIN(INTAG); LVAL(1); ITWS=true; return(TABLE);   };
"</TABLE"  { BEGIN(INTAG); NADA; ITWS=false; return(_TABLE); };
```

```
"<TR"      { BEGIN(INTAG); LVAL(1); return(TR);        };
"</TR"     { BEGIN(INTAG); NADA; return(_TR);          };
"<TD"      { BEGIN(INTAG); LVAL(1); ITWS=false; return(TD);  };
"</TD"     { BEGIN(INTAG); NADA; ITWS=true; return(_TD);     };

"<PRE"       { BEGIN(INTAG); LVAL(1); return(PRE);     };
"</PRE"      { BEGIN(INTAG); NADA; return(_PRE);       };
"<LISTING"   { BEGIN(INTAG); LVAL(1); return(LISTING); };
"</LISTING"  { BEGIN(INTAG); return(_LISTING);         };

"<XMP>"            { BEGIN(XMPMODE); LVAL2(1,1); return(XMP);};
<XMPMODE>[.\n]     { yylval.c = yytext[0]; return(ONECHAR);  };
<XMPMODE>"</XMP>"  { BEGIN(INTAG); NADA; return(_XMP);       };

"<PLAINTEXT>"      { BEGIN(PLAINTEXTMODE); LVAL2(1,1); return(PLAINTEXT); };
<PLAINTEXTMODE>.*  { LVAL(0); return(TEXT); };


"<"\/?{ID}+          { BEGIN(INTAG); LVAL(1); return(TAGGENERAL);   };
<INTAG>{WS}*         ;
<INTAG>{ID}+=        { LVAL2(0,1); return(ATTRIBUTE);  };
<INTAG>{ID}*         { LVAL(0);    return(VALUE);   };
<INTAG>\"[^\"]*\"    |
<INTAG>\'[^\']*\'    { LVAL2(1,1); return(VALUE);   };
<INTAG>[.\n]         ;  /* ??? - Just ignore. */

"<"                                  { BEGIN(INSPECTAG);             };
<INSPECTAG>"!--"([^-]|-[^-])*"--"{WS}*  { LVAL2(3,2); return(COMMENT);    };
<INSPECTAG>[^\>]+                    { LVAL(0);    return(TAGSPECIAL); };

<INTAG,INSPECTAG>">"{WS}+/"<"          { BEGIN(INITIAL); if( !ITWS ) REJECT; }
<INTAG,INSPECTAG>">"                   { BEGIN(INITIAL); }


[^\<]*          { LVAL(0); return(TEXT); };


%%
   /*  yyprint prints the semantic lvalue of token to file.
      If the token has no defined lvalue, this function does nothing.
      YACC app can use: extern void yyprint (FILE* file, int token, YYSTYPE value);
                   #define YYPRINT(file, type, value)   yyprint(file, type, value)
   */
   void yyprint (FILE* file, int token, YYSTYPE value) {
      switch ( token ) {
         case TAGSPECIAL:
         case COMMENT:
         case TAGGENERAL:
         case ATTRIBUTE:
         case VALUE:
         case TEXT:
         case HEAD:
         case TITLE:
         case TABLE:
         case BODY:
         case A:
         case IMG:
         case META:
         case PRE:
         case LISTING:
         case XMP:
         case PLAINTEXT:
         case HTML:     { fprintf(file, "%s", value.str); break; };

         case ONECHAR:   { fprintf(file, "%c", value.c); break; };
      };
   };
   /* yywrap() called at EOF by LEX.
      Return 1 means we don't have more files to process. yylex() will return 0.
   */
   int yywrap() {
     return 1;
   };


   void set_yy_init(void) {
     yy_init = 1;
   };
```

5.2 THE PARSER (YACC)

```
/* FILE:    html_yacc.y - the PARSER, written in YACC (Bison)
   PROJECT: TextOnly, see http://www.vorburger.ch/
   AUTHOR:  Michael Vorburger [mike@vorburger.ch]

   NOTE:    The actions in this grammar contain C++ code.
        Use bison -o html_yacc_tab.cpp to compile.
*/
%{
   #include <stdlib.h>
   #include <malloc.h>
   #include "htmlclasses.h"

   int yylex(void);
   void yyerror(char* s);

   #define YYMAXDEPTH 100000
   /* The default 10000 seems to be too less for HTML docs with
      strongly neested tables. -- OR HAS MY PARSER AN ERROR???
      (Does Left/Right recursiveness make any difference?)      */

   #ifdef _DEBUG
     #include <stdio.h>
     #include <string.h>
     #define YYERROR_VERBOSE  1
     #define YYDEBUG  1
   #endif

   /* Whitespace eliminiation in two ways:
      a) InterTag-whitespace (ITWS) is OK between <HTML>...<HEAD> or
<TABLE>...<TR>...<TD>
         but not in most 'mix' rules, because "<b>x</b> <i>y</i>"  !=
"<b>x</b><i>y</i>"
         (implemented in html_lex.l's rule: ">"{WS}+/"<" ...)
      b) Whitespace "compaction" compresses more than one whitespace to one blank,
         except if inside some 'preformatted' tag. (implemented in
HTMLText::HTMLText cstr)
   */

   bool  pretc = false;    // PRE-etc. = whitespace compaction in mix/TEXT?

   char  theBuffer[4096];
   char* pBuffer;

   extern HTMLDocument* theDocument; /* defined in htmldocument.cpp */
%}


%union {
   char          c;   /* can be returned by yylex  */
   char*         str; /* can be returned by yylex  */
   void*         nada;  /* can be returned by yylex  */
   HTMLElement*  elem;  /* only yyparse internal use */
   TagAttribute* att; /* only yyparse internal use */
};
```

```
%{ // This has to be AFTER %union because YYSTYPE needed.

   #ifdef _DEBUG
   extern void yyprint (FILE* file, int token, YYSTYPE value);
   #define YYPRINT(file, type, value)   yyprint(file, type, value)
   #define ERR(expected) fprintf(stderr, "SYNTAX ERROR: %s expected. ", expected); \
                   YYPRINT (stderr, yychar, yylval); fputc('\n', stderr)
   #else
   #define ERR(expected) fprintf(stderr, "SYNTAX ERROR: %s expected.\n ", expected)
   #endif
%}

%token <c>    ONECHAR
%token <str>  TAGSPECIAL COMMENT TAGGENERAL ATTRIBUTE VALUE TEXT
%token <str>  HTML  HEAD  TITLE  BODY  A TABLE TR TD SCRIPT
%token <nada> _HTML _HEAD _TITLE _BODY _A _TABLE _TR _TD _SCRIPT
%token <str>  FONT
%token <nada> _FONT
%token <str>  IMG META BR
%token <str>  PRE  LISTING  XMP PLAINTEXT
%token <nada> _PRE _LISTING _XMP

%type <elem> head title body mix doctrail comspecs metatitle trs tds /* xmp_block
*/
%type <att>  attribs

%%


document: comspecs HTML attribs comspecs head comspecs body _HTML attribs doctrail
         { theDocument -> first = CONNECT( $1, new HTMLTagB($2, $3, $9, CONNECT(
$4, $5 ), $10) );
          theDocument -> head = $5;
          theDocument -> body = $7;
          (theDocument -> head) -> setNext( CONNECT( $6, theDocument -> body) );
        };

comspecs: COMMENT comspecs   { $$ = new HTMLComment($1, $2); };
      | TAGSPECIAL comspecs{ $$ = new HTMLSpecial($1, $2); };
      | TEXT comspecs         { $$ = $2; /* DISCARD TEXT */   };
      | /* epsilon */      { $$ = NULL;           };
      | error comspecs     { $$ = $2; };

doctrail: TEXT doctrail      { $$ = new HTMLTextPRE($1, $2); };
      | /* epsilon */      { $$ = NULL;           };
      | error doctrail     { $$ = $2; };

head : HEAD attribs metatitle _HEAD attribs
        { $$ = new HTMLTagB($1, $2, $5, $3, NULL); };

metatitle: META attribs metatitle { $$ = new HTMLTagMETA($2, $3); };
       | COMMENT metatitle    { $$ = new HTMLComment($1, $2); };
       | TAGSPECIAL metatitle   { $$ = new HTMLSpecial($1, $2); };
       | title metatitle      { $$ = CONNECT( $1, $2 );    };
       | /* epsilon */        { $$ = NULL;           };
       | error metatitle      { $$ = $2; };

title   : TITLE attribs TEXT _TITLE attribs
        { theDocument -> title = new HTMLText($3, NULL);
          $$ = new HTMLTagB($1, $2, $5, theDocument -> title, NULL);
        };

body : BODY attribs { } mix { } _BODY attribs
        { $$ = new HTMLTagB($1, $2, $7, $4, NULL); };
/* | BODY attribs mix
     { $$ = new HTMLTagU($1, $2, $3); };
*/
mix     : TEXT mix
        { $$ = ( !pretc ? new HTMLText($1, $2) : new HTMLTextPRE($1, $2) ); };
      | TAGGENERAL attribs mix  { $$ = new HTMLTagU($1, $2, $3);       };
      | IMG attribs mix      { $$ = new HTMLTagIMG($2, $3);           };

      | A attribs mix _A attribs mix
        { $$ = new HTMLTagAnchor($1, $2, $5, $3, $6); };

      | FONT attribs mix _FONT attribs mix
        { $$ = new HTMLTagFONT($1, $2, $5, $3, $6); };

      | PRE attribs { pretc=true; } mix _PRE { pretc=false; } attribs mix
        { $$ = new HTMLTagB($1, $2, $7, $4, $8); };

/*
      | LISTING attribs { pretc=true; } mix _LISTING { pretc=false; } attribs mix
        { $$ = new HTMLTagB($1, $2, $7, $4, $8); };

      | XMP { pBuffer = &theBuffer[0]; } xmp_block mix
        { $$ = new HTMLTagB($1, NULL, NULL, $3, $4); };

      | PLAINTEXT TEXT
        { $$ = new HTMLTagU($1, NULL, new HTMLTextPRE($2, NULL)); };
*/
      | SCRIPT attribs COMMENT _SCRIPT attribs mix
        { $$ = new HTMLScript($1, $2, $5, new HTMLComment($3, NULL), $6); };

      | COMMENT mix
        { $$ = new HTMLComment($1, $2); };

      | BR attribs mix
        { $$ = new HTMLTagBR($1, $2, $3); };

      | TAGSPECIAL mix
        { $$ = new HTMLSpecial($1, $2); };

      | /* epsilon */
        { $$ = NULL; };

      | error mix
        { $$ = $2; };

      | TABLE attribs trs _TABLE attribs mix
        { $$ = new HTMLTable($1, $2, $5, $3, $6); };

trs     : TR attribs tds _TR attribs trs  { $$ = new HTMLTagB($1, $2, $5, $3, $6);
};
      | /* epsilon */             { $$ = NULL; };
      | error                    { $$ = NULL; };

tds     : TD attribs mix _TD attribs tds  { $$ = new HTMLTagB($1, $2, $5, $3, $6);
};
      | /* epsilon */             { $$ = NULL; };
      | error                    { $$ = NULL; };

/*
xmp_block : ONECHAR xmp_block   { if ( pBuffer-theBuffer > sizeof(theBuffer) )
abort();
                           else *pBuffer++ = $1; };
        | _XMP           { *pBuffer = 0; $$ = new HTMLTextPRE(pBuffer, NULL);   };
*/

attribs : ATTRIBUTE VALUE attribs { $$ = new TagAttribute($1, $2, $3);   };
      | VALUE attribs        { $$ = new TagAttribute($1, "", $2);   };
      | /* empty */        { $$ = NULL; };
     /* CANNOT error attribs  ==> Shift/Reduce conflict with comspecs! */
%%


void yyerror(char* s) {
   // variable `yynerrs' contains the number of syntax errors encountered so far.
   printf ("yyerror: %s\n", s);
};
```

5.3 HTML_TEXTONLY.CPP

```
/* FILE:     html_textonly.cpp - HTML++ newTextOnly() methods implementations
   PROJECT:  TextOnly, see http://www.vorburger.ch/
   AUTHOR:   Michael Vorburger [mike@vorburger.ch]

   CREATED:  July 5, 1998
*/

#include <typeinfo.h>
#include <stdio.h>
#include "htmlclasses.h"

#include "../../../shared-src/pathfoos.h" // for nice_fsize()

#define TEXTONLY_CHILD  SAPPLY( this->child, newTextOnly() )
#define TEXTONLY_NEXTSAPPLY( this->next, newTextOnly() )

////////////////////////////////////////////////////////////////////
// HTMLDocument
//
HTMLDocument* HTMLDocument::newTextOnly() const {
   if ( !this->body ) return NULL;
   if ( !this->body->getChild() ) return NULL;

   HTMLDocument* textDoc = new HTMLDocument( this->title );
   if ( !textDoc || !textDoc->body ) return NULL;
   // textDoc->setMETA("GENERATOR", "Smart Text-Only; see
http://www.vorburger.ch/smato");

   textDoc->body->setChild( this->body->getChild()->newTextOnly() );
   return textDoc;
};

////////////////////////////////////////////////////////////////////
// HTMLElement
// This is the inherited default behaviour for all classes.
//
HTMLElement* HTMLElement::newTextOnly() const {
   return newClone( TEXTONLY_CHILD, TEXTONLY_NEXT );
};

////////////////////////////////////////////////////////////////////
// HTMLTagIMG
//
HTMLElement* HTMLTagIMG::newTextOnly() const {
   if ( this->isRuler() )
      return new HTMLTagU("HR", NULL, TEXTONLY_NEXT );

   HTMLText* alt = this->getNewALT();

   if ( const HTMLTagAnchor* A = getParent_Typed<HTMLTagAnchor>(this) ) {
      if ( getChild_Typed<HTMLText>(A) ) {
         delete alt; return this->newTextOnly_SkipThis(); }
      else
         return CONNECT(alt, TEXTONLY_NEXT);
   };
   if ( alt->isBlank() )
      { delete alt; return this->newTextOnly_SkipThis(); }
   else
      if ( this->isLinkNeeded() ) {
         delete alt; // because we use getNewALTExtended()
         return new HTMLTagAnchor("A",
              new TagAttribute("href", this->getAttribute("src"), NULL),
            NULL, this->getNewALTExtended(), TEXTONLY_NEXT);
      } else
         return CONNECT(alt, TEXTONLY_NEXT);
};

HTMLText* HTMLTagIMG::getNewALT() const {
   char* alt = Sstrdup( this->getAttribute("ALT") );

   if ( !alt ) {
      alt = Sstrdup( this->getAttribute("SRC") );
      if ( !alt ) return NULL;
      if ( char* pc = strrchr(alt, '/' ) ) {  // cut path
         pc = strdup(pc);
         free(alt);
         alt = pc;
      };
   };

   int l = strlen(alt);
   if ( l>7 && !stricmp( &alt[l-5], "Byte)" ) ) {    // if ALT ends in "Byte)"
      char* pc = strrchr( alt, '(' );
      if ( pc ) {                          // store filesize in this->bytes
         const_cast<HTMLTagIMG*>(this) -> bytes = atoi(pc+1);
         *pc = '\0';                       // and cut from ALT
      };
      if ( pc = strrchr( alt, '.' ) )
         *pc = '\0';                       // cut extension from auto-ALT
   };

   HTMLText* tag = new HTMLText(alt, NULL);
   free(alt);
   return tag;
};

HTMLText* HTMLTagIMG::getNewALTExtended() const {
   char extalt[40];
   HTMLText* alt = this->getNewALT();
   int h = this->getHeight();
   int w = this->getWidth();

   if ( w && h )
      sprintf( extalt, "%s (%ix%i Pixels", alt->getCP(), w, h );
   else
      extalt[0] = '\0';

   if ( int s = this->getBytes() ) {
      char  size[20];
      char* delim = ( extalt[0] ? ", " : " (" );

      strcat( extalt, delim );
      strcat( extalt, nice_fsize(size, s) );
   };
   if ( extalt[0] )
      strcat( extalt, ")" );

   delete alt;
   return new HTMLText(extalt, NULL);
};

int HTMLTagIMG::getBytes() const {
   return this->bytes;
};

bool HTMLTagIMG::isRuler() const {
   int h = this->getHeight();
   int w = this->getWidth();

   if ( h && w ) {
      if ( w/h >= 10 && w>100 && h<50 && h>3 )
         return true;
   };
   return false;
};

bool HTMLTagIMG::isBullet() const {
   int h = this->getHeight();
   int w = this->getWidth();

   if ( h && w ) {
      if ( w/h <= 4 && w<30 && w>5 && h<30 && h>5)
         return true;
   };
   return false;
};

bool HTMLTagIMG::isLinkNeeded() const {
   int h = this->getHeight();
   int w = this->getWidth();

   if ( h && w ) {
      if ( w<75 || w<75 )
         return false;
      else
         return true;
   } else
      return true;
};

/////////////////////////////////////////////////////////////////////
// HTMLTable
//

// bool isLI() Helper function determines,
// a) if only two columns in TD  and  b) and the second column child only
// one Element c) and that only element is an IMG  and  d) and that IMG is a bullet
//
static inline bool isLI(HTMLElement* tr) {
   if ( !tr ) return false;

   HTMLElement* firstTD = tr->getChild();
   if ( !firstTD ) return false;
   if ( !firstTD->getNext() ) return false;

   if ( firstTD->getNext()->getNext() == NULL ) {  // a)
      if ( !( firstTD->getChild() ) ) return false;

      if ( firstTD->getChild()->getNext() == NULL ) {  // b)
         HTMLTagIMG* img = dynamic_cast<HTMLTagIMG*>( firstTD->getChild() );
         if ( img )
            return img->isBullet(); }
         // if ( typeid(firstTD->getChild()) == typeid(HTMLTagIMG) ) {   // c)
         // if ( dynamic_cast<HTMLTagIMG*>(firstTD->getChild()) -> isBullet() ) // d)
         //    return true; };
   };
   return false;
};

HTMLElement* HTMLTable::newTextOnly() const {
   HTMLElement* tr = this->child;
   HTMLElement* li = NULL;
   HTMLElement* text = NULL;

   while ( isLI(tr) ) {
      // Why does tr->child->next->child->newTextOnly()
      // generate "cannot access protected member declared in class 'HTMLElement'

      HTMLElement* item = tr->getChild()->getNext()->getChild()->newTextOnly();
      li = CONNECT(li, new HTMLTagB( "LI", NULL, NULL, item, NULL) );
      tr = tr->getNext();
   };
   if ( li )
      text = new HTMLTagB( "UL", NULL, NULL, li, NULL );

   // if ( tr )
   // text = CONNECT(text, new HTMLTable("TABLE", this->attributes,
   //       this->endAttributes, tr->newTextOnly(), NULL) );

   while ( tr ) {
      HTMLElement* td = tr->getChild();
      while ( td ) {
         if ( td->getChild() != NULL ) {
            HTMLElement* cell = td->getChild()->newTextOnly();
            text = CONNECT(text, new HTMLTagU("P", NULL, cell) );
         };
         td = td->getNext();
      };
      tr = tr->getNext();
   };

   return CONNECT(text, TEXTONLY_NEXT);
};

//////////////////////////////////////////////////////////////////////
// HTMLTagAnchor
//
HTMLElement* HTMLTagAnchor::newTextOnly() const {
   HTMLElement* clickable = TEXTONLY_CHILD;
   if ( !clickable )
      clickable = new HTMLText("???-LINK-???", NULL);
   else {
      const HTMLText* isText = getChild_Typed<HTMLText>(clickable);
      if ( isText && isText->isBlank() ) {
         delete clickable;
         clickable = new HTMLText("???-LINK-???", NULL);
      };
   };

   // separate two immediately following links by " | "
   HTMLElement* next = TEXTONLY_NEXT;
   HTMLTagAnchor* isA = dynamic_cast<HTMLTagAnchor*>(next);
   if ( isA )
      next = new HTMLText(" | ", next);

   else {
      // separated two links which only have BLANK between by " | "
      HTMLText* isText = dynamic_cast<HTMLText*>(next);
      if ( isText ) {
         if ( isText->isBlank() && isText->getNext() ) {
            isA = dynamic_cast<HTMLTagAnchor*>( isText->getNext() );
            if ( isA ) {
               // delete NEXT/isText OHNE NEXT->next zu löschen!
               next = new HTMLText(" | ", isA);
            };
         };
      };
   };

   return new HTMLTagAnchor( "A", this->attributes, this->endAttributes,
                        clickable, next);  // NOT this->
};

//////////////////////////////////////////////////////////////////////
// HTMLTagFONT
//
HTMLElement* HTMLTagFONT::newTextOnly() const {
   return this->newTextOnly_SkipThis();
};

//////////////////////////////////////////////////////////////////////
// HTMLComment
```

```
HTMLElement* HTMLComment::newTextOnly() const {
   return this->newTextOnly_SkipThis();
};

//////////////////////////////////////////////////////////////////
// HTMLScript
//
HTMLElement* HTMLScript::newTextOnly() const {
   return this->newTextOnly_SkipThis();
};
```