

RXPath module v.91 (by robic0)

Source: <http://coding.derkeiler.com/Archive/Perl/comp.lang.perl.misc/2006-06/msg00590.html>

- *From:* robic0
 - *Date:* Thu, 08 Jun 2006 14:32:15 -0700
-

RXPath, Version .91 (by robic0) 6/8/06

- Non-normalized, parsed internal entities
- # Unicode character reference
- # General reference
- # Parameter reference
- Recursive expansion of general references in content and attvalue
- DTD entity not parsed yet
- Other bug fixes

```
#####  
# XML/Xhtml/Html - RXPath parse/edit/filter module (by robic0)  
# -----  
# Compliant w3c XML: 1.1  
# Resources:  
# Extensible Markup Language (XML) 1.1  
# W3C Recommendation 04 February 2004,  
# 15 April 2004  
# http://www.w3.org/TR/xml11/#NT-PITarget  
#####  
$|=1;  
package RXPath;  
use strict;  
use warnings;  
use Carp;  
use vars qw(@ISA);  
@ISA = qw();  
  
my $VERSION = .91;  
  
#=====  
# RXPath package globals  
#=====  
my (  
%Dflth,
```

```

%ErrMsg,
$Nstrt,$Nchar,$Name,
@UC_Nstart,@UC_Nchar,
$RxParseXP1,
$RxAttr,
$RxAttr_DL1,
$RxAttr_DL2,
$RxAttr_RM,
$RxPi,
$RxENTITY,
%dflt_general_ent_subst,
%dflt_parameter_ent_subst
);
my $parsinitflg = 0;

if (!$parsinitflg) {
  InitParser();
  $parsinitflg = 1;
}

#=====
# RXParse user methods
#=====
sub new
{
  my ($class, @args) = @_ ;
  my $self = {};
  $self->{'debug'} = 0;
  $self->{'ignore_errors'} = 0;
  Cleanup($self);
  setDfltHandlers($self);
  return bless ($self, $class);
}

sub original_content
{
  my $self = shift;
  if (defined $self->{'origcontent'} &&
    ref($self->{'origcontent'}) eq 'SCALAR') {
    return ${$self->{'origcontent'}};
  }
  return "";
}

sub setMode
{
  my ($self, @args) = @_ ;

  if (scalar(@args)) {
    while (my ($name, $val) = splice (@args, 0, 2)) {
      $name =~ s/^\s+//s; $name =~ s/\s+$//s;
    }
  }
}

```

```

if (lc($name) eq 'debug') {
$self->{'debug'} = 0;
$self->{'debug'} = 1 if (defined $val && $val);
}
elsif (lc($name) eq 'ignore_errors') {
$self->{'ignore_errors'} = 0;
$self->{'ignore_errors'} = 1 if (defined $val && $val);
}
# add more here
}
}
}

```

```

sub setDfltHandlers
{
my ($self, $name) = @_;
if (defined $name) {
$name =~ s/^\s+//s; $name =~ s/\s+$//s;
my $hname = "h".lc($name);
if (exists $Dflth{$hname}) {
$self->{$hname} = $Dflth{$hname};
}
} else {
foreach my $key (keys %Dflth) {
$self->{$key} = $Dflth{$key};
}
}
}

```

```

sub setHandlers
{
my ($self, @args) = @_;
my %oldh = ();
if (scalar(@args)) {
while (my ($name, $val) = splice (@args, 0, 2)) {
$name =~ s/^\s+//s; $name =~ s/\s+$//s;
my $hname = "h".lc($name);
if (exists $self->{$hname}) {
$oldh{$name} = $self->{$hname};
if (ref($val) eq 'CODE') {
$self->{$hname} = $val;
} else {
# fatal error if not a CODE ref
throwX($self, 'FATAL', '32', $name);
}
}
}
}
return %oldh;
}

```

```

}

sub parse
{
my ($self, $data, @args) = @_ ;
if ($self->{'InParse'}) {
# fatal error if already in parse
throwX($self, 'FATAL', '30');
}
unless (defined $data) {
# fatal error if data source not defined
throwX($self, 'FATAL', '31');
}
$self->{'InParse'} = 1;

# use XP1 processor (for now)
$self->{'proctype'} = 'XP1';
if (ref($data) eq 'SCALAR') {
print "SCALAR ref\n" if ($self->{'debug'});
XP1 ($self, 1, $data);
}
elsif (ref(\$data) eq 'SCALAR') {
print "SCALAR string\n" if ($self->{'debug'});
XP1 ($self, 1, \$data);
} else {
if (ref($data) ne 'GLOB' && ref(\$data) ne 'GLOB') {
# data source not string or filehandle, nor reference to one
throwX($self, 'FATAL', '33');
}
print "GLOB ref or filehandle\n" if ($self->{'debug'});
XP1 ($self, 0, $data);
}
Cleanup($self);
}

#=====
# RXParse non-user methods
#=====
sub Cleanup
{
my $self = shift;
InitEntities($self);
$self->{'origcontent'} = undef;
$self->{'InParse'} = 0;
}

sub InitEntities
{
my $self = shift;
# initial compiled regexp
$self->{'Entities'} = "(?:amp)|(?:gt)|(?:lt)|(?:apos)|(?:quot)|(?:#(?:[0-9+)|(x[0-9a-fA-F+]))";

```

```
# ( 4 4|5 5)
$self->{'RxEntConv'} = qr/(.*?)(&|%)(\$self->{'Entities'});/s;
# 1 12 23 3
# initial entity hash
$self->{'general_ent_subst'} = {%dflt_general_ent_subst};
$self->{'parameter_ent_subst'} = {%dflt_parameter_ent_subst};
$self->{'ring_ent_subst'} = {};
}
```

```
sub XP1 # xp1 processor, parse only, non-edit
```

```
{
my ($self, $BUFFERED, $rpl_mk) = @_ ;
my ($markup_file);
my $parse_ln = "";
my $dyna_ln = "";
my $ref_parse_ln = \$parse_ln;
my $ref_dyna_ln = \$dyna_ln;
if ($BUFFERED) {
$ref_parse_ln = $rpl_mk;
$ref_dyna_ln = \$dyna_ln;
} else {
# assume its a ref to a global or global itself
$markup_file = $rpl_mk;
$ref_dyna_ln = $ref_parse_ln;
}
my $ln_cnt = 0;
my $complete_comment = 0;
my $complete_cdata = 0;
my @Tags = ();
my $havroot = 0;
my $last_cpos = 0;
my $done = 0;
my $content = "";
my $altcontent = undef;

$self->{'origcontent'} = \$content;

while (!$done)
{
$ln_cnt++;

# stream processing (if not buffered)
if (!$BUFFERED) {
if (!($_ = <$markup_file>)) {
# just parse what we have
$done = 1;
# boundry check for runaway
if (($complete_comment+$complete_cdata) > 0) {
$ln_cnt--;
}
} else {
```

```

$$ref_parse_ln .= $_;

## buffer if needing comment/cdata closure
next if ($complete_comment && !/-->/);
next if ($complete_cdata && !/\]\]/>/);

## reset comment/cdata flags
$complete_comment = 0;
$complete_cdata = 0;

## flag serialized comments/cdata buffering
if ((<!--)|(<![CDATA\()/)
{
if (defined $1) { # complete comment
if ($$ref_parse_ln !~ /<!--.*?-->/s) {
$complete_comment = 1;
next;
}
}
elsif (defined $2) { # complete cdata
if ($$ref_parse_ln !~ /<![CDATA\[.*?\]\]/>/s) {
$complete_cdata = 1;
next;
}
}
}
## buffer until '>' or eof
next if (!>/);
}
} else {
$ln_cnt = 1;
$done = 1;
}

## REGEX Parsing loop
while ($$ref_parse_ln =~ /$RxParseXP1/g)
{
## handle contents
if (defined $14) {
$content .= $14;
$last_cpos = pos($$ref_parse_ln);
next;
}
## valid content here ... can be taken off
print "-x20,\n" if ($self->{'debug'});
if (length ($content)) {
## check reserved characters in content
if ($content =~ /[\<>]/) {
$content =~ s/^\s+//s; $content =~ s\s+$/s;
## mark-up characters in content
throwX($self, 'OVR', '01', $content, $ref_parse_ln, $last_cpos, $ln_cnt);
}
}
}

```

```

}
if (!scalar(@Tags)) {
# $content =~ s/^\s+//s; $content =~ s/\s+$//s;
if ($content =~ /^[^\s]/s) {
## content at root level
throwX($self, 'OVR', '02', $content, $ref_parse_ln, $last_cpos, $ln_cnt);
}
}
# substitute special xml characters, then call content handler with $content
# -----
# $content has to be a constant if xml reserved chars
# are found, copy altered string to pass to handler
# otherwise pass original $content
# -----
if (defined ($saltcontent = convertEntities ($self, \ $content))) {
$self->{'hchar'}($self, $$saltcontent);
} else {
$self->{'hchar'}($self, $content);
}
#print "14 $content\n" if ($self->{'debug'});
#print "-x20,\n" if ($self->{'debug'});
$content = ";
}
#if ($show_pos && $debug) {
# my $rr = pos $$ref_parse_ln;
# print "$rr ";
#}

## <tag> or </tag> or <tag/>
if (defined $2)
{
my ($l1,$l3) = (length($1),length($3));
if (($l1+$l3)==0) { ## <tag>
if (!scalar(@Tags) && $havroot) {
## new root node <tag>
throwX($self, 'OVR', '03', $2, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
push @Tags,$2;
$havroot = 1;
# call start tag handler with $2
$self->{'hstart'}($self, $2);
}
elsif ($l1==1 && $l3==0) { ## </tag>
my $pval = pop @Tags;
if (!defined $pval) {
## missing start tag </tag>
throwX($self, 'OVR', '04', $2, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
elsif ($2 ne $pval) {
## expected closing tag </tag>
throwX($self, 'OVR', '05', $pval, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
}
}

```

```

}
# call end tag handler with $2
$self->{'hend'}($self, $2);
}
elsif ($I1==0 && $I3==1) { ## <tag/>
if (!scalar(@Tags) && $havroot) {
## new root node <tag/>
throwX($self, 'OVR', '06', $2, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
$havroot = 1; # first and only <root/>
# call start tag handler, then end tag handler, with $2
$self->{'hstart'}($self, $2);
$self->{'hend'}($self, $2);
} else {
## </node/> errors
## hard error, just report
throwX($self, 'HARD', '07', "$I2$I3", $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
next;
}
#print "2 TAG: $I2$I3\n" if ($self->{'debug'});
}
## <tag attrib/> or <tag attrib>
elsif (defined $5)
{
my $I7 = length($7);

## attributes
my $attref = getAttrARRAY($self, $6);
unless (ref($attref)) {
## missing or extra token
## hard error, just report
throwX($self, 'HARD', '08', $attref, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
next;
}
if ($I7==0) { ## <tag attrib>
if (!scalar(@Tags) && $havroot) {
## new root node
throwX($self, 'OVR', '03', $5, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
push @Tags,$5;
$havroot = 1;
# call start tag handler with $5 and attributes @{$attref}
$self->{'hstart'}($self, $5, @{$attref});
}
elsif ($I7==1) { ## <tag attrib/>
if (!scalar(@Tags) && $havroot) {
## new root node
throwX($self, 'OVR', '06', $7, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
$havroot = 1; # first and only <root attrib/>
# call start tag handler with $5 and attributes @{$attref}, then end tag handler with $5

```

```

$self->{'hstart'}($self, $5, @{$attref});
$self->{'hend'}($self, $5);
} else {
## syntax error
## hard error, just report
throwX($self, 'HARD', '07', "$5$6$7", $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
next;
}
#if ($self->{'debug'}) {
# print "5,6 TAG: $5 Attr: $6$7\n" ;
#}
}
## XMLDECL or PI (processing instruction)
elsif (defined $8)
{
my $pi = $8;
# xml declaration ?
if ($pi =~ /^xml(?:.*)$/) {
my $attref = getAttrARRAY($self, $1);
unless (ref($attref)) {
## missing or extra token in xmldecl
## hard error, just report
throwX($self, 'HARD', '14', $attref, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
next;
}
#if (!scalar(@{$attref})) {
# ## missing xmldecl parameters
# throwX($self, 'OVR', '15', $pi, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
#}
my ($version,$encoding,$standalone);
while (my ($name,$val) = splice (@{$attref}, 0, 2)) {
if ('version' eq lc($name) && !defined $version) {
if ($val !~ /^[0-9]\.[0-9]+$/) {
## invalid version character data in xmldecl
throwX($self, 'OVR', '16', "$name = '$val'", $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
$version = $val;
} elsif ('encoding' eq lc($name) && !defined $encoding) {
if ($val !~ /^[A-Za-z][\w\.-]*$/) {
## invalid encoding character data in xmldecl
throwX($self, 'OVR', '17', "$name = '$val'", $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
$encoding = $val;
} elsif ('standalone' eq lc($name) && !defined $standalone) {
if ($val !~ /^(?:yes)|(?:no)$/) {
## invalid standalone character data in xmldecl
throwX($self, 'OVR', '18', "$name = '$val'", $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
$standalone = ($val eq 'yes' ? 1 : 0);
} else {
## unknown xmldecl parameter

```

RXParse module v.91 (by robic0)

```

throwX($self, 'OVR', '19', "$name = '$val'", $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
}
if (!defined $version) {
# missing version in xmldecl
## hard error, just report
throwX($self, 'HARD', '20', $pi, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
next;
}
# call xmldecl handler
$self->{'hxmldecl'}($self, $version, $encoding, $standalone);
}
# PI – processing instruction
elsif ($pi =~ /$RxPi/) {
# call pi handler
$self->{'hproc'}($self, $1, $2);
} else {
# unknown PI data
throwX($self, 'HARD', '21', $pi, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
next;
}
#print "8 VERSION: $8\n" if ($self->{'debug'});
}
## META
elsif (defined $4) {
# If doctype is HTML then META is not closed
# parse meta data, call handler
$self->{'hmeta'}($self, $4);
#print "4 META: $4\n" if ($self->{'debug'});
}
## DOCTYPE
elsif (defined $9) {
# parse doctype, call handler
$self->{'hdoctype'}($self, $9);
#print "9 DOCTYPE: $9\n" if ($self->{'debug'});
}
## CDATA
elsif (defined $10) {
if (!scalar(@Tags)) {
## CDATA content at root
throwX($self, 'OVR', '09', $10, $ref_parse_ln, pos($$ref_parse_ln), $ln_cnt);
}
# call cdata handler
$self->{'hcdata'}($self, $10);
#print "10 CDATA: $10\n" if ($self->{'debug'});
}
## COMMENT
elsif (defined $11) {
# call comment handler
$self->{'hcomment'}($self, $11);
#print "11 COMMENT: $11\n" if ($self->{'debug'});
}

```

```

}
## ATTLIST
elsif (defined $12) {
# parse attlist, call handler
$self->{'hattlist'}($self, $12);
#print "12 ATTLIST: $12\n" if ($self->{'debug'});
}
## ENTITY
elsif (defined $13) {
# parse entity, call handler
my ($entdata, $entdata_added, $entname) = ($13, undef, "");
if ($entdata =~ /$RxENTITY/) {
if (defined $1) {
# general entity replacement
$entdata_added = addEntity($self, 0, $1, $3);
$entname = "&$1";
} else {
# parameter entity replacement
$entdata_added = addEntity($self, 1, $2, $3);
$entname = "&$2";
}
}
else {
# unknown ENTITY data
#
}
if (defined $entdata_added) {
$self->{'hentity'}($self, $entname, $$entdata_added);
} else {
$self->{'hentity'}($self, $entname, $entdata);
}
#print "13 ENTITY: $13\n" if ($self->{'debug'});
}
}
$$ref_dyna_ln = $content;
$content = "";
}
if (!$havroot) {
# not valid xml
throwX($self, 'OVR', '10');
}
if (scalar(@Tags)) {
my $str = "";
while (defined (my $etag = pop @Tags)) {
$str .= ", /$etag";
}
$str =~ s/^, +//;
# missing end tag
throwX($self, 'OVR', '11', $str);
}
if ($$ref_dyna_ln =~ /[^\s]/s) {

```

```

if ($$ref_dyna_ln =~ /[\<>]/) {
# mark-up characters in content
throwX($self, 'OVR', '12', $$ref_dyna_ln);
} else {
# content at root level (end)
throwX($self, 'OVR', '13', $$ref_dyna_ln);
}
}
}
$self->{'origcontent'} = undef;
return 1;
}

sub getAttrARRAY
{
my ($self, $attrstr) = @_;
my $aref = [];
my ($alt_attval, $attval, $rx);

while ($attrstr =~ s/$RxAttr//) {
push @{$aref}, $1;
if ($2 eq "") {
$rx = $RxAttr_DL1;
} else {
$rx = $RxAttr_DL2;
}
if ($attrstr =~ s/$$rx//) {
if (defined $1) {
push @{$aref}, $1;
next;
}
$attval = $2;
if (defined ($alt_attval = convertEntities ($self, $attval))) {
push @{$aref}, $alt_attval;
next;
}
push @{$aref}, $attval;
next;
}
return $attrstr;
}
if ($attrstr =~ /$RxAttr_RM/) {
$attrstr =~ s/^\s+//s; $attrstr =~ s/\s+$//s;
return $attrstr if (length($attrstr));
}
return $aref;
}

sub convertEntities
{
my ($self, $str_ref, $opts) = @_;
my $alt_str = "";

```



```

} else {
$alt_str .= $self->{'general_ent_subst'}->{$entname};
}
$self->{'ring_ent_subst'}->{$entname} = undef;
$res = 1;
}
} else {
$alt_str .= "$1$2$3";
}
} else {
# Parameter reference
if (($opts & 4) && exists $self->{'parameter_ent_subst'}->{$3}) {
$alt_str .= "$1$self->{'parameter_ent_subst'}->{$3}";
$res = 1;
} else {
$alt_str .= "$1$2$3";
}
}
}
}
}
if ($res) {
$alt_str .= substr $$str_ref, pos($$str_ref);
return \$alt_str;
}
return undef;
}

sub getEntityUchar
{
my ($self, $code) = @_ ;
if (($code >= 0x01 && $code <= 0xD7FF) ||
($code >= 0xE000 && $code <= 0xFFFFD) ||
($code >= 0x10000 && $code <= 0x10FFFF)) {
return chr($code);
}
return undef;
}

sub addEntity
{
my ($self, $peflag, $entname, $entval) = @_ ;

# Non-normalized, internal entities only
# (no external defs yet, ie:SYSTEM/PUBLIC/NDATA)
return undef unless
($entval =~ s/^\s*('[^']*?)\s*$/1/s || $entval =~ s/^\s*"(["]*)"\s*$/1/s);

# Replacement text: convert parameter and character references only
my ($alt_entval);
if (defined ($alt_entval = convertEntities ($self, \$entval, 5))) {
$entval = $$alt_entval;
}

```

```

}
my $enttype = 'general_ent_subst';
$enttype = 'parameter_ent_subst' if ($peflag);

if (exists $self->{'$enttype'}->{$entname}) {
# warn, pre-existing ent name
return undef;
}
$self->{'$enttype'}->{$entname} = $entval;
$self->{'Entities'} .= "|(?:$entname)";
# recompile regexp
$self->{'RxEntConv'} = qr/(.*?)(&|%)($self->{'Entities'})/s;
return \ $entval;
}

# default handlers
# -----
sub dflt_start {
my ($self, $el, @attr) = @_;
if ($self->{'debug'}) {
print "start _: $el\n";
while (my ($name,$val) = splice (@attr, 0,2)) {
print " "x12,"$name = $val\n";
}
}
}

sub dflt_char {
my ($self, $str) = @_;
if ($self->{'debug'}) {
print "char _: $str\n";
print "- "x20,"\n";
}
}

sub dflt_end { my ($self, $el) = @_;print "end _: /$el\n" if ($self->{'debug'});}
sub dflt_cdata { my ($self, $str) = @_;print "cdata _: $str\n" if ($self->{'debug'});}
sub dflt_comment { my ($self, $str) = @_;print "comnt _: $str\n" if ($self->{'debug'});}
sub dflt_meta { my ($self, $str) = @_;print "meta _: $str\n" if ($self->{'debug'});}
sub dflt_attlist { my ($self, $parm) = @_;print "attlist_h _: $parm\n" if ($self->{'debug'});}
sub dflt_doctype { my ($self, $parm) = @_;print "doctype_h _: $parm\n" if ($self->{'debug'});}
sub dflt_element { my ($self, $parm) = @_;print "element_h _: $parm\n" if ($self->{'debug'});}

sub dflt_entity {
my ($self, $entname, $entval) = @_;
if ($self->{'debug'}) {
print "entity_h _: $entname = $entval\n";
}
}

sub dflt_xmldecl {
my ($self, $version, $encoding, $standalone) = @_;

```

```

if ($self->{'debug'}) {
print "xmldecl_h _: version = $version\n" if (defined $encoding);
print " "x14,"encoding = $encoding\n" if (defined $encoding);
print " "x14,"standalone = $standalone\n" if (defined $standalone);
}
}
sub dflt_proc {
my ($self, $target, $data) = @_;

if ($self->{'debug'}) {
print "proc_h _: target = $target\n";
print " "x14,"data = $data\n";
}
}
sub dflt_error {my ($self, $errlvl, $errno, $estr, $estr_basic) = @_;print "$estr\n" if ($self->{'debug'});}

```

```

# =====
# RXParse global init
# =====
sub InitParser
{
%Dflth = (
'hstart' => \&dflt_start,
'hend' => \&dflt_end,
'hchar' => \&dflt_char,
'hcdata' => \&dflt_cdata,
'hcomment' => \&dflt_comment,
'hmeta' => \&dflt_meta,
'hattlist' => \&dflt_attlist,
'hentity' => \&dflt_entity,
'hdoctype' => \&dflt_doctype,
'helement' => \&dflt_element,
'hxmldecl' => \&dflt_xmldecl,
'hproc' => \&dflt_proc,
'herror' => \&dflt_error,
);
@UC_Nstart = (
"\x{C0}-\x{D6}",
"\x{D8}-\x{F6}",
"\x{F8}-\x{2FF}",
"\x{370}-\x{37D}",
"\x{37F}-\x{1FFF}",
"\x{200C}-\x{200D}",
"\x{2070}-\x{218F}",
"\x{2C00}-\x{2FEF}",
"\x{3001}-\x{D7FF}",
"\x{F900}-\x{FDCF}",
"\x{FDF0}-\x{FFFD}",

```



```
'11' => "\"missing end tag '%s'\", \$datastr",
'12' => "\"mark-up or reserved characters in content (end), malformed element? '%s'\", \$datastr",
'13' => "\"content at root level (end): '%s'\", \$datastr",
'14' => "\"invalid, missing or extra tokens in xmldecl assignment (line %s, col %s): %s\", \$line, \$col,
\$datastr",
'15' => "\"missing xmldecl parameters (line %s, col %s): %s\", \$line, \$col, \$datastr",
'16' => "\"invalid 'version' character data in xmldecl (line %s, col %s): %s\", \$line, \$col, \$datastr",
'17' => "\"invalid 'encoding' character data in xmldecl (line %s, col %s): %s\", \$line, \$col, \$datastr",
'18' => "\"invalid 'standalone' character data in xmldecl (line %s, col %s): %s\", \$line, \$col, \$datastr",
'19' => "\"unknown xmldecl parameter (line %s, col %s): %s\", \$line, \$col, \$datastr",
'20' => "\"missing xmldecl 'version' parameter (line %s, col %s): %s\", \$line, \$col, \$datastr",
'21' => "\"unknown or missing processing instruction parameters (line %s, col %s): '%s'\", \$line, \$col,
\$datastr",
'30' => "\"already in parse\"",
'31' => "\"data source not defined\"",
'32' => "\"handler '%s' is not a CODE reference\", \$datastr",
'33' => "\"data source not string or filehandle, nor reference to one\"",
);
}
```

```
sub throwX
```

```
{
my ($self, $errlvl, $errno, $datastr, $lrefseg, $cseg_err, $l_tot) = @_ ;
my ($line, $col, $estr, $estr_basic) = (0,0,"");
if (defined $lrefseg) {
($line,$col) = getRealColumn($lrefseg, $l_tot, $cseg_err);
}
die "No such error message ($errno)\n" if (!exists $ErrMsg{$errno});

my $ctmpl = "\"$estr_basic = sprintf ($ErrMsg{$errno});";
eval $ctmpl;
$estr = "rp_error_$errno, $estr_basic";
```

```
# call error handler
```

```
$self->{'herror'}($self, $errlvl, $errno, $estr, $estr_basic);
```

```
if ($errlvl eq 'FATAL') {
Cleanup($self); croak $estr."\n";
}
elsif (!$self->{'ignore_errors'} && ($errlvl eq 'HARD' || $errlvl eq 'OVR')) {
Cleanup($self); croak $estr."\n";
}
}
```

```
sub getRealColumn
```

```
{
my ($lrefseg, $l_tot, $cseg_err) = @_ ;
my $cseg_offset = 0;
my $save_pos = pos($lrefseg);
pos($lrefseg) = 0;
my ($lseg_tot, $lseg_offset) = (0,1);
```

```
while ($$lrefseg =~ /\n/g) {  
  $lseg_tot++;  
  if (pos($$lrefseg) < $cseg_err) {  
    $cseg_offset = pos($$lrefseg);  
    $lseg_offset++;  
    next;  
  }  
  if ($l_tot <= 1) {  
    $lseg_tot = $l_tot;  
    last;  
  }  
  }  
  pos($$lrefseg) = $save_pos;  
  return ($l_tot-$lseg_tot+$lseg_offset, $cseg_err-$cseg_offset);  
}
```

1;

__END__

.