Parsing with PHP

Kore Nordmann <kore@php.net>

August 22, 2009

About me

- Kore Nordmann, <kore@php.net>, <kn@ez.no>
- Long time PHP developer
- Regular speaker, author, etc.
- Studies computer science in Dortmund
- Active open source developer:
  - eZ Components (Graph, WebDav, *Document*), Arbit, PHPUnit, Torii, PHPillow, KaForkL, Image 3D, WCV, ...

Outline

Parsing in PHP

- Parsers in PHP? Why the hell?

Parsing in PHP

- Parsers in PHP? Why the hell?
- Applications for parsers
  - Markup languages
  - Domain specific languages (DSL)
  - Language interpreters (template languages)

Parsing in PHP

- Parsers in PHP? Why the hell?
- Applications for parsers
  - Markup languages
  - Domain specific languages (DSL)
  - Language interpreters (template languages)
- So, who already wrote a parser in PHP?

The use of regular expressions

- Parsing = text processing = regular expressions; Right?

The use of regular expressions

- Parsing = text processing = regular expressions; Right?
- *No!* – regular expressions only work for regular languages. [1]

The use of regular expressions

- Parsing = text processing = regular expressions; Right?
- *No!* – regular expressions only work for regular languages. [1]
- Regular languages
  - … cannot express recursion
  - … grammars must be right-linear (right-regular)

The use of regular expressions

- Parsing = text processing = regular expressions; Right?
- *No!* – regular expressions only work for regular languages. [1]
- Regular languages
  - ... cannot express recursion
  - ... grammars must be right-linear (right-regular)
- What does that mean?

```
S ::= "(" A ")"
A ::= "foo"
    | S
```

Introducing PCRE

- *But:* PCRE are no real regular expression any more.

Introducing PCRE

- *But:* PCRE are no real regular expression any more.
- PCRE knows backreferences

```
S ::= "a" S "a" | "b"

( (a*) b \1 )
```

Introducing PCRE

- *But:* PCRE are no real regular expression any more.
- PCRE knows backreferences

```
S ::= "a" S "a" | "b"

( (a*) b \1 )
```

- . . . and recursion

```
S ::= "(" A ")"
A ::= "foo" | S

(
    \(
        ( (?>foo) | (?R) )
    \)
)
```

Using PCRE for Parsing?

- Seems to make people think: Use PCRE for parsing!

- You don't even ge

Using PCRE for Parsing?

- Seems to make people think: Use PCRE for parsing!
- A PCRE BBCode parser:

```
(
   (
      [^\[\]]*
        (?# Match an opening BBCode tag )
        \[([a-z]+)(?:=([^\]]+))?\]
           (?# The actual recursion )
           (?>[^\[\]]* | (?R) )
        (?# Match the closing tag )
        \[/\2\]
      [^\[\]]*
   )
)ix
```

- You don't even ge

Using PCRE for Parsing?

- Seems to make people think: Use PCRE for parsing!
- A PCRE BBCode parser:

```
(
    (
        [^\[\]]*
            (?# Match an opening BBCode tag )
            \[([a-z]+)(?:=([^\]]+))?\]
                (?# The actual recursion )
                (?>[^\[\]]* | (?R) )
            (?# Match the closing tag )
            \[/\2\]
        [^\[\]]*
    )
)ix
```

- This is useless, because:
  - It's unmaintainable
  - You don't even ge

Using PCRE for Parsing?

- Seems to make people think: Use PCRE for parsing!
- A PCRE BBCode parser:

```
(
    (
        [^\[\]]*
            (?# Match an opening BBCode tag )
            \[([a-z]+)(?:=([^\]]+))?\]
                (?# The actual recursion )
                (?>[^\[\]]* | (?R) )
            (?# Match the closing tag )
            \[/\2\]
        [^\[\]]*
    )
)ix
```

- This is useless, because:
  - It's unmaintainable
  - You don't get a syntax tree (AST)
  - You don't even get

Using PCRE for Parsing?

- Seems to make people think: Use PCRE for parsing!
- A PCRE BBCode parser:

```
(
  (
    [^\[\]]*
      (?# Match an opening BBCode tag )
      \[([a-z]+)(?:=([^\]]+))?\]
        (?# The actual recursion )
        (?>[^\[\]]* | (?R) )
      (?# Match the closing tag )
      \[/\2\]
    [^\[\]]*
  )
)ix
```

- This is useless, because:
  - It's unmaintainable
  - You don't get a syntax tree (AST)
  - You don't even get proper match arrays

Using of regular expressions

- Regular expressions are *not* for:

Using of regular expressions

- Regular expressions are *not* for:
    - Building the full parser for your markup language

Using of regular expressions

- Regular expressions are *not* for:
  - Building the full parser for your markup language
  - Finding contents in HTML documents

Using of regular expressions

- Regular expressions are *not* for:
  - Building the full parser for your markup language
  - Finding contents in HTML documents
- Regular expressions are perfect for:

Using of regular expressions

- Regular expressions are *not* for:
  - Building the full parser for your markup language
  - Finding contents in HTML documents
- Regular expressions are perfect for:
  - Tokenizing (example follows)

Using of regular expressions

- Regular expressions are *not* for:
  - Building the full parser for your markup language
  - Finding contents in HTML documents
- Regular expressions are perfect for:
  - Tokenizing (example follows)
  - Parse regular DSLs (seldom)

Outline

Find links

- http://www.the-art-of-web.com/php/parse-links/

```php
$url = "http://www.example.net/somepage.html";
$input = @file_get_contents($url) or die('Could not access file: $url');
$regexp = "<a\s[^>]*href=(\"??)([^\" >]*?)\\1[^>]*>(.*)<\/a>";
if ( preg_match_all( "/$regexp/siU", $input, $matches ) )
{
  // $matches[2] = array of link addresses
  // $matches[3] = array of link text - including HTML code
}
```

Find links

- The correct way: [2]

```php
$oldSetting = libxml_use_internal_errors( true );

$html = new DOMDocument();
$html->loadHtmlFile( $url );
$xpath = new DOMXPath( $html );
$links = $xpath->query( '//a' );
foreach ( $links as $link ) {
  echo $link->getAttribute( 'href' ), "\n";
}
```

Find links

- The correct way: [2]

```php
$oldSetting = libxml_use_internal_errors( true );

$html = new DOMDocument();
$html->loadHtmlFile( $url );
$xpath = new DOMXPath( $html );
$links = $xpath->query( '//a' );
foreach ( $links as $link ) {
  echo $link->getAttribute( 'href' ), "\n";
}
```

- Maintainable

Find links

- The correct way: [2]

```php
$oldSetting = libxml_use_internal_errors( true );

$html = new DOMDocument();
$html->loadHtmlFile( $url );
$xpath = new DOMXPath( $html );
$links = $xpath->query( '//a' );
foreach ( $links as $link ) {
  echo $link->getAttribute( 'href' ), "\n";
}
```

- Maintainable
- Correct

Find links

- The correct way: [2]

```php
$oldSetting = libxml_use_internal_errors( true );

$html = new DOMDocument();
$html->loadHtmlFile( $url );
$xpath = new DOMXPath( $html );
$links = $xpath->query( '//a' );
foreach ( $links as $link ) {
  echo $link->getAttribute( 'href' ), "\n";
}
```

- Maintainable
- Correct
- Handles recursion properly

BBCode-parser

- http://www.tutorials.de/forum/php-tutorials/279124-bbcode-mit-php-parsen.html

```php
function parseBBCode2HTML( $bb )
{
  $bb = preg_replace(
    '(\[b\](.*?)\[/b\])', '<b>$1</b>', $bb );
  $bb = preg_replace(
    '(\[i\](.*?)\[/i\])', '<i>$1</i>', $bb );
  $bb = preg_replace(
    '(\[color=([a-f\d]{6}?).*\](.*?)\[/color\])',
    '<font color="#$1">$2</font>', $bb );
  $bb = preg_replace(
    '(\[url=([^ ]+).*\](.*)\[/url\])',
    '<a href="$1">$2</a>', $bb );
  $bb = preg_replace(
    '(\n)', "<br/>\n", $bb );
  return $bb;
}
```

BBCode-parser

- Does not handle invalid markup:

```
Input: "Hello␣[b]world!":
=> Hello [b]world!

Input: "Hello␣world[/b]!":
=> Hello world[/b]!

Input: "[i]Hello␣[b]world[/i][/b]!":
=> <i>Hello <b>world</i></b>!
```

- Fails:
  - Does not report formatting errors
  - Creates invalid markup

BBCode-parser

- It is trivial to do it correct.
- Define tokens first:

```php
protected $tokens = array(
    'open'  => '(\\A\\[(?P<value>[a-z]+)\\])',
    'close' => '(\\A\\[/(?P<value>[a-z]+)\\])',
    'text'  => '(\\A(?P<value>[^\\[]+|\\[))',
);
```

BBCode-parser

- Tokenize input string:

```
public function tokenize( $string ) {
  $scanned = array();
  while (strlen($string)) {
    foreach ($this->tokens as $type => $expression) {
      if (preg_match($expression, $string, $match)) {
        $scanned[] = array(
          'type'    => $type,
          'content' => $match['value'],
        );
        $string  = substr($string, strlen($match[0]));
        continue 2;
      }
    }
    throw new Exception("Could not process: '$string'");
  }
  return $scanned;
}
```

BBCode-parser

- Build AST from token stream

```php
public function parse( array &$tokens, $tag = null ) {
  $ast = array();
  while ( $token = array_shift( $tokens ) ) {
    switch ( $token['type'] ) {
      case 'text':
        $ast[] = $token['content'];
        break;
      case 'open':
        $ast[] = array(
          'tag'     => $token['content'],
          'content' => $this->parse( $tokens, $token['content'] ),
        );
        break;
      case 'close':
        if ( $token['content'] !== $tag ) throw new Exception( "Unexpected closing tag: {$token['content']}." );
        return $ast;
    }
  }
  if ( $tag !== null ) throw new Exception( "Missing closing tag for $tag." );
  return $ast;
}
```

BBCode-parser

- Example result

```
Input: "[i]Hello␣[b]world[/b][/i]!":
AST:
array(2) {
  [0] => array(2) {
    ["tag"]     => string(1) "i"
    ["content"] => array(2) {
      [0] => string(6) "Hello␣"
      [1] => array(2) {
        ["tag"]     => string(1) "b"
        ["content"] => array(1) {
          [0] => string(5) "world"
        }
      }
    }
  }
  [1] => string(1) "!"
}
```

BBCode-parser

- Proper error messages:

```
Input: "Hello␣[b]world!":
=> Exception: Missing closing tag for b.

Input: "Hello␣world[/b]!":
=> Exception: Unexpected closing tag: b.

Input: "[i]Hello␣[b]world[/i][/b]!":
=> Exception: Unexpected closing tag: i.
```

- TODO (trivial):
  - Add context information to errors (line, position)
  - Parse attributes in tags

Simplified CSS parser

- Parse simplified CSS specifications:

```
page {
    page-size: "A4";
    page-orientation: "portrait";
    padding: "22mm 16mm";

    // Margin for pages specifies an additional
        outer border, which can be used
    // to cut if off later, f.e. in printing
    margin: "0mm";
}

para {
    margin: "3mm 0mm 1mm 0mm";
}
```

Simplified CSS parser

- The grammar:

```
File           ::=  Directive+
Directive      ::=  Address '{' Formatting* '}'
Formatting     ::=  Name ':' '"' Value '"' ';'
Name           ::=  [A-Za-z-]+
Value          ::=  [^"]+

Address        ::=  Element ( Rule )*
Rule           ::=  '>'? Element
Element        ::=  ElementName ( '.' ClassName | '#'
     ElementId )

ClassName      ::=  [A-Za-z_-]+
ElementName    ::=  XMLName* | '*'
ElementId      ::=  XMLName

* XMLName references to http://www.w3.org/TR/REC-
     xml/#NT-Name
```

Simplified CSS parser

- Tokens

```
T_WHITESPACE       => '(\\A\\s+)S',
T_COMMENT          => '(\\A/\\*.*\\*/)SUs',
T_COMMENT          => '(\\A//.*$)Sm',
T_START            => '(\\A\\{)S',
T_END              => '(\\A\\})S',
T_FORMATTING       => '(\\A(?P<name>[A-Za-z-]+)\\s*:\\
        s*"(?P<value>[^"]+)"\\s*;)S',
T_ADDRESS          => '(\\A' . $xmlName . ')S',
T_DESC_ADDRESS     => '(\\A>[\\t\\x20]+' . $xmlName .
        ')S',
T_ADDRESS_CLASS    => '(\\A\\.[A-Za-z_-]+)S',
T_ADDRESS_ID       => '(\\A#' . $xmlName . ')S',
```

Simplified CSS parser

- Common read() method

```
private function read( array $types, array &$tokens ) {
    $token = array_shift( $tokens );

    if ( !in_array( $token['type'], $types, true ) ) {
        $names = array();
        foreach ( $types as $type )
        {
            $names[] = $this->tokenNames[$type];
        }

        $this->triggerError( E_PARSE,
            "Expected one of: " . implode( ', ', $names ) . ", found " .
                $this->tokenNames[$token['type']] . '.',
            $this->file, $token['line'], $token['position']
        );
    }

    return $token;
}
```

Simplified CSS parser

- Simple domain specific LL(1) parser

```
$directives    = array();
$addressTokens = array( self::T_ADDRESS, self::T_DESC_ADDRESS, self::
     T_ADDRESS_ID, self::T_ADDRESS_CLASS );

while ( count( $tokens ) > 1 ) {
    $formats = array(); $address = array();

    do {
        $addressToken = $this->read( $addressTokens, $tokens );
        $address[] = $addressToken['match'][0];
    } while ( $tokens[0]['type'] !== self::T_START );

    $this->read( array( self::T_START ), $tokens );

    while ( $tokens[0]['type'] !== self::T_END ) {
        $format = $this->read( array( self::T_FORMATTING ), $tokens );
        $formats[$format['match']['name']] = $format['match']['value'];
    }

    $this->read( array( self::T_END ), $tokens );

    $directives[] = new ezcDocumentPdfCssDirective(
        $address,
        $formats,
        $this->file, $addressToken['line'], $addressToken['position']
    );
}
```

Regular DSLs

- Some domain specific languages are regular languages

Regular DSLs

- Some domain specific languages are regular languages
- Can be parsed using regular expressions

Regular DSLs

- Some domain specific languages are regular languages
- Can be parsed using regular expressions
- Like CSS border specifications

```
border: 1px solid #f00 2px dotted black;
```

Parsing CSS border specifications

- Size definition

```
(?:[+-]?\s*(?:\d*\.)?\d+)(?:mm|px|pt|in)?
```

Parsing CSS border specifications

- Size definition

```
(?:[+-]?\s*(?:\d*\.)?\d+)(?:mm|px|pt|in)?
```

- Border definition

```
(?:none|dotted|dashed|solid|double|groove|ridge|
    inset|outset|inherit)
```

Parsing CSS border specifications

- Size definition

```
(?:[+-]?\s*(?:\d*\.)?\d+)(?:mm|px|pt|in)?
```

- Border definition

```
(?:none|dotted|dashed|solid|double|groove|ridge|
    inset|outset|inherit)
```

- Color definitions:

```
(?:#?([0-9a-f])([0-9a-f])([0-9a-f])([0-9a-f])?)
(?:#?([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})([0-9a-
    f]{2})?)
(?:\s*rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s
    *([0-9]+)\s*\)\s*)
```

Parsing CSS border specifications

- Border style definition:

```
(?:
  (?:(?:[+-]?\s*(?:\d*\.)?\d+)(?:mm|px|pt|in)?\s*)?
  (?:(?:none|dotted|dashed|solid|double|groove|
      ridge|inset|outset|inherit)\s*)?
  (?:
    (?: transparent | none |
      (?:#?([0-9a-f])([0-9a-f])([0-9a-f])([0-9a-f])
          ?) |
      (?:#?([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})
          ([0-9a-f]{2})?) |
      (?:\s*rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\
          s*([0-9]+)\s*\)\s*) |
    )
  )?
)
```

Parsing CSS border specifications

```
(^\s*(?:(?P<m0>(?P<m00>(?:(?:(?:[+-]?\s*(?:\d*\.)?\d+)(?:mm|px|pt|in)?\s*)?(?:(?:none|dotted|
dashed|solid|double|groove|ridge|inset|outset|inherit)\s*)?(?:(?i:transparent|none|(?:#?([0-9a-f
])([0-9a-f])([0-9a-f])([0-9a-f])?)|(?:#?([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})?)
|(?:\s*rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\s*)|(?:\s*rgba\s*\(\s*([0-9]+)\s
*,\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\s*)))?)))|(?P<m1>(?P<m10>(?:(?:(?:[+-]?\s*(?:\d
*\.)?\d+)(?:mm|px|pt|in)?\s*)?(?:(?:none|dotted|dashed|solid|double|groove|ridge|inset|outset|
inherit)\s*)?(?:(?i:transparent|none|(?:#?([0-9a-f])([0-9a-f])([0-9a-f])([0-9a-f])?)|(?:#?([0-9a
-f]{2})([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})?)|(?:\s*rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s
*([0-9]+)\s*\)\s*)|(?:\s*rgba\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\
s*)))?))\s+(?P<m11>(?:(?:(?:[+-]?\s*(?:\d*\.)?\d+)(?:mm|px|pt|in)?\s*)?(?:(?:none|dotted|dashed|
solid|double|groove|ridge|inset|outset|inherit)\s*)?(?:(?i:transparent|none|(?:#?([0-9a-f])([0-9
a-f])([0-9a-f])([0-9a-f])?)|(?:#?([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})?)|(?:\s*
rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\s*)|(?:\s*rgba\s*\(\s*([0-9]+)\s*,\s
*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\s*)))?)))|(?P<m2>(?P<m20>(?:(?:(?:[+-]?\s*(?:\d*\.)
?\d+)(?:mm|px|pt|in)?\s*)?(?:(?:none|dotted|dashed|solid|double|groove|ridge|inset|outset|
inherit)\s*)?(?:(?i:transparent|none|(?:#?([0-9a-f])([0-9a-f])([0-9a-f])([0-9a-f])?)|(?:#?([0-9a
-f]{2})([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})?)|(?:\s*rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s
*([0-9]+)\s*\)\s*)|(?:\s*rgba\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\
s*)))?))\s+(?P<m21>(?:(?:(?:[+-]?\s*(?:\d*\.)?\d+)(?:mm|px|pt|in)?\s*)?(?:(?:none|dotted|dashed|
solid|double|groove|ridge|inset|outset|inherit)\s*)?(?:(?i:transparent|none|(?:#?([0-9a-f])([0-9
a-f])([0-9a-f])([0-9a-f])?)|(?:#?([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})?)|(?:\s*
rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\s*)|(?:\s*rgba\s*\(\s*([0-9]+)\s*,\s
*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\s*)))?))\s+(?P<m22>(?:(?:(?:[+-]?\s*(?:\d*\.)?\d+)
(?:mm|px|pt|in)?\s*)?(?:(?:none|dotted|dashed|solid|double|groove|ridge|inset|outset|inherit)\s
*)?(?:(?i:transparent|none|(?:#?([0-9a-f])([0-9a-f])([0-9a-f])([0-9a-f])?)|(?:#?([0-9a-f]{2})
([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})?)|(?:\s*rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s
*([0-9]+)\s*\)\s*)|(?:\s*rgba\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\
s*)))?)))|(?P<m3>(?P<m30>(?:(?:(?:[+-]?\s*(?:\d*\.)?\d+)(?:mm|px|pt|in)?\s*)?(?:(?:none|dotted|
dashed|solid|double|groove|ridge|inset|outset|inherit)\s*)?(?:(?i:transparent|none|(?:#?([0-9a-f
])([0-9a-f])([0-9a-f])([0-9a-f])?)|(?:#?([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})?)
|(?:\s*rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\s*)|(?:\s*rgba\s*\(\s*([0-9]+)\s
*,\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\s*)))?))\s+(?P<m31>(?:(?:(?:[+-]?\s*(?:\d*\.)?\
d+)(?:mm|px|pt|in)?\s*)?(?:(?:none|dotted|dashed|solid|double|groove|ridge|inset|outset|inherit)
\s*)?(?:(?i:transparent|none|(?:#?([0-9a-f])([0-9a-f])([0-9a-f])([0-9a-f])?)|(?:#?([0-9a-f]{2})
([0-9a-f]{2})([0-9a-f]{2})([0-9a-f]{2})?)|(?:\s*rgb\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s
*([0-9]+)\s*\)\s*)|(?:\s*rgba\s*\(\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*,\s*([0-9]+)\s*\)\
s*)))?))\s+(?P<m32>(?:(?:(?:[+-]?\s*(?:\d*\.)?\d+)(?:mm|px|pt|in)?\s*)?(?:(?:none|dotted|...
```

Outline

The goal

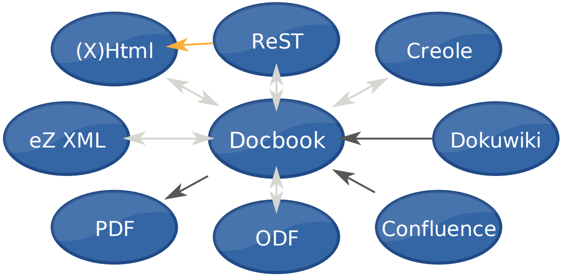
- Support coversions between document markup formats
- Applications in content management
  - Different input mechanisms
    - WYSISWYG editor (HTML)
    - Simple text editor (wiki markup)
    - Emails (ReST)
  - Different output formats
    - Web frontend (HTML)
    - Technical documentation management (Docbook)
    - Print (PDF)

The formats

- Currently supported formats
  - Docbook
  - (X)Html
  - eZ XML
  - ReST
  - Wiki
    - Dokuwiki, popular PHP based wiki (wiki.php.net) (read-only)
    - Creole, wiki markup standardization initiative
    - Confluence, Apache Atlassian wiki dialect (read-only)
- Currently in development
  - PDF (write only)
  - ODF

The architecture

- Docbook as central conversion format
  - Possible conversion shortcuts
  - Conversions always configurable and extensible



The languages

- Text based markup (RST, Wiki) is easy to write

The languages

- Text based markup (RST, Wiki) is easy to write
- . . . but can be horrible to parse properly.

The languages

- Text based markup (RST, Wiki) is easy to write
- . . . but can be horrible to parse properly.
- RST is a context-sensitive (type 1) language.
    - Easy to prove with the pumping lemma for context-free languages [3] on the title markup.

The languages

- Text based markup (RST, Wiki) is easy to write
- . . . but can be horrible to parse properly.
- RST is a context-sensitive (type 1) language.
  - Easy to prove with the pumping lemma for context-free languages [3] on the title markup.
  - There are no general parser approaches for context-sensitive languages.

The languages

- Text based markup (RST, Wiki) is easy to write
- . . . but can be horrible to parse properly.
- RST is a context-sensitive (type 1) language.
  - Easy to prove with the pumping lemma for context-free languages [3] on the title markup.
  - There are no general parser approaches for context-sensitive languages.
  - The document component uses a manually crafted pseudo shift-reduce-parser for those languages.

Conclusion

- Design your language with care.

Conclusion

- Design your language with care.
- Design your parser with language properties in mind.

Conclusion

- Design your language with care.
- Design your parser with language properties in mind.
- Do *not* try to *parse* with regular expressions.

The end

- Open questions?
- Further remarks?
- Contact
  - Mail: <kore@php.net>
  - Web: `http://kore-nordmann.de/` (Slides will be available here soonish)
  - Twitter: `http://twitter.com/koredn`

Links

- Some further links
  - http://kore-nordmann.de/blog/0081_parse_html_extract_data_from_html.html
  - http://kore-nordmann.de/blog/do_NOT_parse_using_regexp.html
  - http://ezcomponents.org/docs/tutorials/Document

Bibliography I

[1] K. Nordmann.
Do not try parsing with regular expressions.
http://kore-nordmann.de/blog/do_NOT_parse_using_regexp.html, July 2007.

[2] K. Nordmann.
Extracting data from html.
http://kore-nordmann.de/blog/0081_parse_html_extract_data_from_html.html, Februrary 2009.

[3] Wikipedia.
Pumping lemma for context-free languages — wikipedia, the free encyclopedia, 2009.
[Online; accessed 4-August-2009].