Forum: WD 8.x

Topic: Creation Client Soap xml avex windev pour PagesJaunes

Subject: Creation Client Soap xml avex windev pour PagesJaunes

PostÃ© par: ouioui69

Contribution le : 3/12/2004 18:20:21

Voila mon Pb...je souhaite passer ces codes Perl en WD pour l'integrer à une appli WD. Le but est de parser le site PagesJaunes puis sur les reponses faire EcranVersTable ou XML...plutot que MECHANIZE.
Etant totalement imcompetent en Perl et débutant avec Windev je me rapproche de vous pour un coup de main.

QQ a t-il deja eu besoin de faire ce genre de manip via un client soap xml en utilisant le site http://labs.pagesjaunes.fr:9001 ??

//Code:

```
# Copyright 2004 France Telecom R&D LLC.  All Rights Reserved.
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the CeCILL free software licensing agreement
# available at http://www.inria.fr/presse/pre119.fr.html.

use URI;
use SOAP::Lite;
use Data::Dumper;

my $soap = SOAP::Lite->uri('http://labs.pagesjaunes.fr:9001/PagesJaunes')
  ->proxy("http://labs.pagesjaunes.fr:9001");

# Cheap argument parsing
while ($arg = pop @ARGV) { $parsexml = 1 if $arg eq '-parsexml' }
if ($parsexml) { $soap->outputxml(0) } else { $soap->outputxml(1) }

# Authenticate (You can also do this in the shell below if you'd prefer)
$soap->Authenticate("your_username", "your_password");

# Loop forever, evaluating each request and displaying the results.
while (1) {
  print "PagesJaunes SOAP> ";
  chomp($request = );
  $result = eval "$soap->$request";

  if    (!$result)   { warn "No result -- is the server running?n" }
  elsif (!$parsexml) { print $result }
  elsif ($result->fault) {
    print join ', ', $result->faultcode, $result->faultstring,
```

```
    $result->faultdetail;
  } else {
    print Data::Dumper::Dumper($result->{_content}), "n";
  }
}

exit 0;

//AUTRE SOLUTION TOUJOURS EN PERL
//package WWW::Search::Pagesjaunes;
use strict;
use Carp qw(carp croak);
use HTML::Form;
use WWW::Mechanize;
use HTML::TokeParser;
use HTTP::Request::Common;
use LWP::UserAgent;

$WWW::Search::Pagesjaunes::VERSION = '0.12';

sub ROOT_URL() { 'http://www.pagesjaunes.fr' }

sub new {
    my $class = shift;
    my $self  = {};
    my $ua    = shift() || WWW::Mechanize->new(
        env_proxy  => 1,
        keep_alive => 1,
        timeout    => 30,
        agent      => "WWW::Search::Pagesjaunes/$WWW::Search::Pagesjaunes::VERSION",
    );

    $self->{ua}    = $ua;
    $self->{limit} = 50;
    $self->{fast}  = 0;
    $self->{error} = 1;
    $self->{lang}  = 'FR';

    bless( $self, $class );
}

sub agent {
    my $self = shift;
    if ( $_[0] ) {
        my $old = $self->{ua};
        $self->{ua} = $_[0];
        return $old;
    }
    else {
```

```
        return $self->{ua};
    }
}

sub find {
    my $self = shift;
    my %opt  = @_;

    my $p = $opt{activite} ? 'j' : 'b';

    # Make the first request to pagesjaunes.fr
    $self->{URL} = ROOT_URL . "/p$p.cgi";


    if ( $self->{fast} ) {
        $self->{req} = POST(
            $self->{URL},
            [
                faire           => 'decode_input_image',
                DEFAULT_ACTION  => $p . 'f_inscriptions_req',
                lang            => $self->{lang},
                pays            => 'FR',
                srv             => uc("p$p"),
                TYPE_RECHERCHE  => 'ZZZ',
                input_image     => '',
                FRM_ACTIVITE    => $p eq 'j' ? $opt{activite} : undef,
                FRM_NOM         => $opt{nom},
                FRM_PRENOM      => $p eq 'b' ? $opt{prenom}   : undef,
                FRM_ADRESSE     => $opt{adresse},
                FRM_LOCALITE    => $opt{localite},
                FRM_DEPARTEMENT => $opt{departement},
                #'${p}F_INSCRIPTIONS_REQ.x' => 1,
                #'${p}F_INSCRIPTIONS_REQ.y' => 1,
            ]);
    }
    else {
        my $req = $self->{ua}->get($self->{URL});

        if ( !$req->content || !$req->is_success ) {
            croak('Error while retrieving the HTML page');
        }

        my @forms = HTML::Form->parse( $req->content, $self->{URL} );

        # BooK finds the form by grepping thru all of them, instead
        # of limiting ourselves to the first and second form.
        my ($form) = grep { $_->find_input('lang') } @forms;

        eval {
```

```
            # HTML::Form complains when you change hidden fields values.
            local $^W;
            $form->value( 'lang', $self->{lang} );

            $form->value( 'FRM_ACTIVITE', $opt{activite} ) if $opt{activite};
            $form->value( 'FRM_NOM',      $opt{nom} );
            $form->value( 'FRM_PRENOM',   $opt{prenom} )   if !$opt{activite};
            $form->value( 'FRM_ADRESSE',  $opt{adresse} );
            $form->value( 'FRM_LOCALITE', $opt{localite} );
            $form->value( 'FRM_DEPARTEMENT', $opt{departement} );
        };
        croak "Cannot fill the pagesjaunes request form. try with the 'fast' optionn" if $@;

        $self->{limit} = $opt{limit} || $self->{limit};

        $self->{req} = $form->click;
    }

    return $self;
}

sub results {
    my $self = shift;

    my $result_page = $self->{ua}->request( $self->{req} )->content;

    my $parser      = HTML::TokeParser->new( $result_page );

    # All the
 tags are transformed to '§¤§', to separate
    # multiple phone numbers
    $parser->{textify} = {
        'br' => sub() { '§¤§' }
    };

    my @results;

    if ( $self->{limit} == 0 ) {
        $self->{has_more} = 0;
        return @results;
    }

    # XXX This is a really crude parsing of the data, but it seems to
    # get the job done.
    #
    #
    #
    #
```

```
#       Name
#       Â
#
#

#       Address
#       (télécopie)? Phone
#
#
#
#
$self->{has_more} = 0;

while ( my $token = $parser->get_tag("table") ) {
   next
     unless $token->[1]
     && $token->[1]{class}
     && $token->[1]{class} eq 'fdcadreinscr';
   {    # We're inside an entry table

       $parser->get_tag("td");    # The first  is the name
       my $name = _trim( $parser->get_trimmed_text('/td') );

       $parser->get_tag("td");    # The second  is the address
       my $address = _trim( $parser->get_trimmed_text('/td') );
       $address =~ s/W*|.*$//g;

       $parser->get_tag("td");    # The third  is the phone number
       my $phone = _trim( $parser->get_trimmed_text('/td') );
       my @phones = map { _trim($_); s/.(s*d)/$1/; $_ }  split(/§¤§/, $phone);

       # The third  tag is either the mail or the descr, depending
       # on the class
       my @emails = ('');
       my $tag = $parser->get_tag("td");
       if ( $tag->[1]{class} && $tag->[1]{class} eq 'txtinscr'){
          my $email  = _trim( $parser->get_trimmed_text('/td') );
          @emails = map { _trim($_); s/Mails*:s*//; $_ }  split(/§¤§/, $email);
       }

       push(
           @results,
           WWW::Search::Pagesjaunes::Entry->new(
              $name, $address, [ @phones ], [ @emails ]
           )
       );

       return @results if --$self->{limit} == 0;
   }
```

```
    }

    foreach my $form ( HTML::Form->parse( $result_page, $self->{URL} ) ) {
        if (   $form->find_input('faire')
            && $form->value('faire') eq 'inscriptions_suivant' )
        {
            $self->{has_more} = 1;
            $self->{req}      = $form->click();
        }
    }

    # If there was no result, we look for an error message in the HTML page
    if ( !@results && $self->{error} ) {
        $parser = HTML::TokeParser->new( $result_page );
        while ( my $token = $parser->get_tag("font") ) {
            next
              unless $token->[1]
              && $token->[1]{color}
              && $token->[1]{color} eq '#ff0000';
            $parser->{textify} = {
                'br' => sub() { " " }
            };
            carp _trim( $parser->get_trimmed_text('/font') ) . "n";
        }
    }

    wantarray ? @results : $results[0];
}

sub _trim {
    $_[0] =~ s/xa0/ /g;       # Transform the Â  into whitespace
    $_[0] =~ s/^s*|s*$//g;
    $_[0] =~ s/s+/ /g;
    $_[0];
}

sub limit {
    my $self = shift;
    $self->{limit} = $_[0] || $self->{limit};
}

sub has_more { $_[0]->{has_more} }

package WWW::Search::Pagesjaunes::Entry;

# The entry object is a blessed array with the following indices:
# 0 - Name
# 1 - Address
# 2 - Arrayref of phone numbers
```

```
# 3 - E-mail (pj)
# 4 - Notes  (pj)

sub new     {
    my $class = shift;
    bless [ @_ ], $class
}
sub name    { $_[0]->[0] }
sub address { $_[0]->[1] }
sub phone   { $_[0]->[2] }
sub email   { $_[0]->[3] }
sub entry   {
    # Name      Address     First email      Phones
    $_[0]->[0], $_[0]->[1], $_[0]->[3]->[0], @{ @{ $_[0] }[2] },
}

1;
```