

Perl/SOAP::Lite: Rewrite response XML for ASP.NET compatibility

So you finally hacked up your nifty SOAP::Lite web service only to find that it works fine with SOAP::Lite or PHP clients, but ASP.NET terribly fails?

Yes, I should mention, that you must of course write up a WSDL first, especially for .NET, I'll cover that topic in a follow-up.

This post however refers to a hack that I have done to SOAP::Lite to allow for dynamic response rewriting for different SOAP client implementations.

So, without loosing to many words, you'll find a code sample down below.

This sample script is intended to be run through the CGI of your favorite webserver, let's say it's URL would be something like **<http://webservice.acme.nowhere/webservice.cgi>**.

So, here's the code, which I have hopefully documented well enough, otherwise feel free to ask ;-)

```
#!/usr/bin/perl -w

# package MySoapHandler
#
# here we have a self-contained package within the script which we need
# to override the SOAP::Transport::HTTP built-in functions.
#
# this is required to add some compatibility support for SOAP clients
# that don't get along well with SOAP::Lite's native representation
# of the XML response object.
# An example to this is ASP.NET
#
package MySoapHandler;

use SOAP::Transport::HTTP;
use Data::Dumper;

use vars qw(@ISA);
@ISA = qw(SOAP::Transport::HTTP::CGI);

# sub make_response
#
# override SOAP::Transport::HTTP::make_response,
# in here we basically apply some rewritings to
# the XML output before sending it to the client
# we check on the HTTP_USER_AGENT to do this dynamically
#
sub make_response {
    my $self = shift;          # get the class name
    my $self_funcname = (caller(0))[3]; # get the function name
    printf STDERR sprintf( "%s: Call to %s/%s(%s): dispatch started ...n", POSIX::strftime("%m-%d-%Y %H:%M:%S", localtime),
    $self, $self_funcname, join(',', @_ ) );
```

```
my($code, $response) = @_;
```

```
# check the HTTP USER AGENT first
# since we want to stay compatible to SOAP::Lite and PHP clients,
# we apply our special output handling stuff to all other clients
#
# you may, of course, also switch this around and apply this
# strictly to the ASP.NET user agent
#
if( $ENV{HTTP_USER_AGENT} !~ /SOAP::Lite/ && $ENV{HTTP_USER_AGENT} !~ /PHP/ ) {
    printf STDERR sprintf ("USER_AGENT is '%s', applying rewriting to XML stream.n", $ENV{HTTP_USER_AGENT});

    # in here, you may now apply all sorts of regex magic
    # to perform your rewriting on the '$response' variable content,
    # let's say, you want to reply your Method Response XML entity,
    # you'd do so like this:
    #
    $response =~ s||g;
    $response =~ s||g;

    # be verbose on what we've done
    #
    printf STDERR sprintf( "XML stream after rewriting:n%sn", $response );

} else {
    printf STDERR sprintf ("USER_AGENT is '%s', no rewriting of the XML stream is needed.n", $ENV{HTTP_USER_AGENT});
}

printf STDERR sprintf( "%s: Call to %s/%s: dispatch completed.n", POSIX::strftime("%m-%d-%Y %H:%M:%S", localtime),
$self, $self_funcname );

    # return the response by invoking the parent package's native function
    #
    my $result = $self->SUPER::make_response($code, $response);
}

1;
# end: package MySoapHandler #

# normally, we would dispatch calls like this:
#
# SOAP::Transport::HTTP::CGI->dispatch_to('/path/to/my/lib/dir', 'my::service')->handle;
#
# since we override the base class with our own,
```

```
# we use the object provided by our self-contained package above,  
# so, basically, it's the same ;-)  
#  
MySoapHandler  
-> dispatch_to('/path/to/my/lib/dir', 'my::service')  
-> handle;  
  
exit;  
__END__
```